

PhilJS Book

PhilJS Team

- [Introduction](#)
 - Who this book is for
 - What you will learn
 - Conventions
 - Canonical standards
 - Core ideas in one page
 - How to read this book
 - Compatibility and versions
 - What "Nexus era" means
 - Next steps
- [Front Matter](#)
- [PhilJS Book](#)
- [Introduction to PhilJS](#)
 - What is PhilJS?
 - Try It Yourself
 - [3. Cost Tracking Built-In](#)
 - [4. TypeScript 6 First](#)
 - [5. Server Functions = Zero Boilerplate APIs](#)
 - [6. Islands Architecture for Maximum Performance](#)
 - [7. Batteries Included](#)
 - Who Should Use PhilJS?
 - Perfect For:
 - When You Need Extensions
 - How Is PhilJS Different?
 - vs React
 - vs Vue
 - vs Svelte
 - vs Next.js
 - Quick Feature Overview
 - Requirements
 - Reactivity
 - Components
 - Server Functions
 - Routing
 - Data Fetching
 - Forms
 - Getting Started
 - Community and Support
 - License
- [Installation](#)
 - Prerequisites
 - Quick Start with `create-philjs`
 - Using npm
 - Using pnpm (recommended)
 - Using yarn
 - Using bun
 - Interactive Setup
 - Options Explained
 - Manual Installation
 - 1. Install PhilJS packages
 - 2. Install dev dependencies
 - 3. Create `tsconfig.json`
 - 4. Create `tsconfig.node.json`
 - 5. Create `vite.config.ts`
 - 6. Create `package.json` scripts
 - 7. Create your first component
 - 8. Create `src/index.tsx`
 - 9. Create `index.html`
 - 10. Start development server
- Project Structure
 - Key Files
- Editor Setup

- VS Code (Recommended)
- WebStorm / IntelliJ IDEA
- Other Editors
- Troubleshooting
 - Port 3000 already in use
 - Module not found errors
 - TypeScript errors in JSX
 - Vite not found
 - "Cannot find module '@philjs/core'"
 - Windows-specific issues
- Next Steps
- Version Information
- Upgrade Guide
- Quick Start
 - What You'll Learn
 - Step 1: Create a New Project
 - Step 2: Create Your First Component
 - Creating State with Signals
 - Reading Signal Values
 - Updating Signal Values
 - Step 3: Add Some Style
 - What's New?
 - Step 4: Run Your App
 - Step 5: Add More Interactivity
 - What's New?
 - Understanding What You Built
 - 1. Fine-Grained Reactivity
 - 2. Minimal Code
 - 3. TypeScript Safety
 - 4. Performance
 - Next Steps
 - Continue Learning
 - Build Something Real
 - Explore Features
 - Common Questions
 - Troubleshooting
 - Summary
- Thinking in PhilJS
 - The Mental Shift from React
 - React's Mental Model
 - PhilJS's Mental Model
 - Understanding Fine-Grained Reactivity
 - Automatic Dependency Tracking
 - Surgical Updates
 - When to Use Signals vs Props vs Context
 - Use Props When...
 - Use Signals When...
 - Use Context When...
 - Common Mental Model Pitfalls
 - Pitfall 1: Forgetting to Call Signals
 - Pitfall 2: Using Signals Like React State
 - Pitfall 3: Not Using Memos for Derived State
 - Pitfall 4: Mutating Signal Values
 - Pitfall 5: Creating Signals Inside Render
 - Component Composition Patterns
 - Pattern 1: Container/Presenter
 - Pattern 2: Compound Components
 - Pattern 3: Render Props (via Children)
 - Pattern 4: Higher-Order Components (HOCs)
 - Reactivity Mental Model Diagram
 - Component Lifecycle Mental Model
 - Best Practices Summary
 - ☑ Do This
 - ☒ Avoid This
 - Transitioning Your Mindset
 - From React Hooks to PhilJS

- [From Vue to PhilJS](#)
- [Next Steps](#)
- [Summary](#)
- [Your First Component](#)
 - [What is a Component?](#)
 - [Creating Your First Component](#)
 - [Step 1: Basic Component Structure](#)
 - [Step 2: Adding Props](#)
 - [Step 3: Adding Styles](#)
 - [Step 4: Adding Interactivity with Signals](#)
 - [JSX Syntax Explained](#)
 - [Embedding Expressions](#)
 - [Attributes](#)
 - [Children](#)
 - [Conditional Rendering](#)
 - [Lists](#)
 - [Fragments](#)
 - [Composing Components](#)
 - [Example: User List](#)
 - [Nested Components](#)
 - [Component Best Practices](#)
 - [1. Keep Components Small](#)
 - [2. Use Descriptive Names](#)
 - [3. Single Responsibility](#)
 - [4. Type Your Props](#)
 - [5. Extract Reusable Styles](#)
 - [Common Mistakes to Avoid](#)
 - [1. Forgetting to Return JSX](#)
 - [2. Not Exporting Components](#)
 - [3. Using Signals Without Calling Them](#)
 - [4. Mutating Props](#)
 - [5. Missing Keys in Lists](#)
 - [Complete Example: Product Card](#)
 - [Next Steps](#)
 - [Summary](#)
- [Your First App](#)
 - [1. Create the component](#)
 - [2. Create the entry point](#)
 - [3. Ensure the HTML shell exists](#)
 - [4. Run the app](#)
 - [Add styling](#)
 - [Add a route](#)
 - [Add a loader](#)
 - [Add a test](#)
 - [Checklist](#)
- [Project Structure](#)
 - [Why this layout](#)
 - [Files to add early](#)
 - [Multi-package workspace](#)
 - [Server entry \(SSR\)](#)
 - [Checklist](#)
- [Tutorial: Build a Todo App](#)
 - [What You'll Learn](#)
 - [What We're Building](#)
 - [Setup](#)
 - [Step 1: Define the Data Model](#)
 - [Step 2: Create the TodoItem Component](#)
 - [Step 3: Create localStorage Utilities](#)
 - [Step 4: Create the Main TodoApp Component](#)
 - [Step 5: Wire up the router](#)
 - [Step 6: Create the Main App](#)
 - [Understanding the Code](#)
 - [State Persistence with Effects](#)
 - [Derived State with Memos](#)
 - [URL-Based Filtering](#)

- [Immutable Updates](#)
- [Step 7: Add Features](#)
 - [Bulk Actions](#)
 - [Sorting](#)
 - [Statistics](#)
 - [Due Dates](#)
- [Performance Optimization](#)
 - [Virtualized Lists](#)
 - [Debounced Search](#)
- [Testing](#)
- [What You Learned](#)
- [Challenges](#)
- [Next Steps](#)
- [Tutorial: Build Tic-Tac-Toe](#)
 - [What You'll Learn](#)
 - [What We're Building](#)
 - [Setup](#)
 - [Step 1: Create the Square Component](#)
 - [Step 2: Create the Board Component](#)
 - [Step 3: Add Game Logic](#)
 - [Step 4: Wire Everything Up](#)
 - [Understanding the Code](#)
 - [State Management with Signals](#)
 - [Computed Values with Memos](#)
 - [Immutable Updates](#)
 - [Time Travel Implementation](#)
 - [Step 5: Add Enhancements](#)
 - [Show Move Coordinates](#)
 - [Add Animations](#)
 - [Add Sound Effects](#)
 - [Complete Source Code](#)
 - [What You Learned](#)
 - [Challenges](#)
 - [Next Steps](#)
- [Tutorial: Build a Demo App](#)
 - [What You'll Learn](#)
 - [What We're Building](#)
 - [Setup](#)
 - [Step 1: Create the Counter Component](#)
 - [Understanding Signals](#)
 - [Step 2: Create the Data Fetcher Component](#)
 - [Async Patterns with Signals](#)
 - [Step 3: Create the Animation Demo Component](#)
 - [Spring Physics Animations](#)
 - [Step 4: Create the Main App](#)
 - [Understanding the Code](#)
 - [Component Composition](#)
 - [Responsive Grid Layout](#)
 - [Modern Styling](#)
 - [What You Learned](#)
 - [Challenges](#)
 - [Next Steps](#)
- [Tutorial: Build a Storefront with SSR](#)
 - [What You'll Learn](#)
 - [What We're Building](#)
 - [Architecture Overview](#)
 - [Setup](#)
 - [Step 1: Understanding SSR with Loaders](#)
 - [Home Page with Loader](#)
 - [How Loaders Work](#)
 - [Step 2: Dynamic Routes with Actions](#)
 - [Product Page](#)
 - [Understanding Actions](#)
 - [Step 3: Islands Architecture](#)
 - [Client Entry Point](#)
 - [Creating an Island Component](#)

- Why Islands?
 - Step 4: AI Integration
 - AI Adapter
 - Using AI in a Loader
 - Step 5: Progressive Web App (PWA)
 - Service Worker
 - PWA Manifest
 - Step 6: Performance Monitoring
 - RUM (Real User Monitoring)
 - Performance Budgets
 - Step 7: Security
 - Content Security Policy (CSP)
 - Signed Cookies
 - Production Deployment
 - Build for Production
 - Production Server
 - Deployment Platforms
 - What You Learned
 - Challenges
 - Next Steps
- Tutorial: Build a Blog with Static Site Generation
 - What You'll Learn
 - What We're Building
 - Setup
 - Step 1: Project Structure
 - Step 2: Create Sample Blog Posts
 - Step 3: Create Post Utilities
 - Step 4: Create SEO Component
 - Step 5: Create Post Card Component
 - Step 6: Create Blog Listing Page
 - Step 7: Create Individual Post Page
 - Step 8: Create Tag Page
 - Step 9: Wire up the router
 - Step 10: Generate RSS Feed
 - Step 11: Generate Sitemap
 - Step 12: Configure Static Generation
 - Step 13: Build and Deploy
 - Build for Production
 - Deploy to Vercel
 - Deploy to Netlify
 - Deploy to Cloudflare Pages
 - Step 14: Add Incremental Static Regeneration
 - Performance Optimization
 - Image Optimization
 - Code Splitting
 - Prefetching
 - What You Learned
 - Challenges
 - Performance Results
 - Next Steps
 - Components
 - Typed props
 - Children
 - Composition
 - Default values
 - Events and handlers
 - Refs and lifecycle
 - Composition patterns
 - Performance
 - Testing components
 - Checklist
 - Signals and Reactivity
 - Basic signal
 - Derived values with `memo`
 - Writable computed values

- Batch updates
- Read without tracking
- Async data with resources
- Effects
- Best practices
- Testing signals
- Performance tips
- Checklist
- Try it now: optimistic counter with batch
- Signals Cookbook
 - Derived counts and filters
 - Toggle helpers
 - Batching updates
 - Linked signals (writable computed)
 - Untracked reads
 - Effects do side effects only
 - Interval with cleanup
 - Patterns to avoid
 - Checklist
- Effects and Memos
 - Effects
 - Memos
 - Manage effect lifetimes
 - When to use what
 - Avoiding pitfalls
 - Untracking in effects
 - Testing effects and memos
 - Checklist
 - Try it now: memo + effect combo
- JSX Basics
 - Elements and props
 - Styling with objects
 - Reactive attributes
 - Fragments and arrays
 - Controlled vs uncontrolled
 - Accessibility first
 - TypeScript helpers
 - Patterns to avoid
 - Try it now: accessible button with forwarded props
- Conditional and Lists
 - Conditional rendering
 - Rendering lists
 - Guards and fallbacks
 - Async-friendly rendering
 - Keys and stability
 - Conditional classes/styles
 - Checklist
 - Try it now: filtered list
- Routing Overview
 - Router Quick Start
 - Navigation Basics
 - Declarative links
 - Programmatic navigation
 - Inspecting the current route
 - Dynamic Routes & Params
 - Multiple parameters
 - Layouts & Nested Routes
 - Loaders & Actions (Preview)
 - Smart Prefetch & Transitions
 - Next Steps
- Routing Basics
 - Project Structure
 - Registering the Router
 - Defining Route Components
 - Building a Layout
 - Navigation

- Declarative Links
 - Programmatic Navigation
 - Inspecting the Current Route
- Route Parameters
- Loaders and Actions
- Smart Prefetch & Transitions
- Next Steps
- Navigation and Links
 - <Link> Component
 - External URLs
 - Programmatic Navigation
 - Active Links
 - Smart Prefetching
 - View Transitions
 - Navigation Guards & Loaders
 - Best Practices
- Dynamic Routes
 - Basic Parameters
 - Data Loading with Params
 - Multiple Parameters
 - Catch-All Routes (*)
 - Optional Segments
 - Type Safety
 - Tips
- Route Parameters
 - Basic Param
 - Multiple Segments
 - Optional Branches
 - Catch-All (*)
 - TypeScript Tips
 - Accessing Params Globally
- Layouts
 - Basic Layout
 - Nested Layouts
 - Sharing Loader Data
 - Guarding Routes
 - Styling Tips
- Route Groups
 - What You'll Learn
 - What are Route Groups?
 - Creating Route Groups
 - Multiple Layouts
 - Marketing Layout
 - App Layout
 - Docs Layout
 - Organizing by Feature
 - Nested Route Groups
 - Shared Components
 - Group-Specific Middleware
 - Combining Route Groups
 - Multiple Root Layouts
 - Route Groups with Dynamic Routes
 - Conditional Layouts by Group
 - Complete Example
 - Storefront Layout
 - Shop Layout
 - Account Layout
 - Best Practices
 - Use Groups for Organization
 - Group Related Components
 - Use Descriptive Group Names
 - Don't Over-Group
 - Combine with Regular Folders
 - Common Patterns
 - Auth vs Public

- Marketing vs App
- User Roles
- Summary
- Route Guards
 - Auth Guard Example
 - Redirect Helpers
 - Combining with Actions
 - Tips
- Error Handling
 - What You'll Learn
 - 404 Not Found
 - Basic 404 Page
 - Custom 404 with Search
 - 404 with Analytics
 - Error Boundaries
 - Route Error Boundary
 - Global Error Boundary
 - Layout Error Boundary
 - Navigation Error Handling
 - Handle Navigation Errors
 - Retry Failed Navigation
 - Data Fetching Errors
 - Handle Route Data Errors
 - Automatic Retry
 - API Route Errors
 - API Error Handler
 - Validation Errors
 - Error Recovery
 - Refresh on Error
 - Fallback Content
 - Offline Handling
 - Detect Offline State
 - Queue Failed Requests
 - Error Monitoring
 - Log Errors to Service
 - Route Error Monitoring
 - User-Friendly Error Messages
 - Error Message Mapping
 - Contextual Error Pages
 - Best Practices
 - Always Show User Feedback
 - Provide Recovery Options
 - Log Errors for Debugging
 - Use Error Boundaries
 - Differentiate Error Types
 - Summary
- Data Loading
 - Basic Loader
 - Accessing Params
 - Error Handling
 - Combining with Signals
 - Actions (Mutations)
 - Caching & Prefetching
- Loading States
 - Inline Loading Indicators
 - Suspense-like Patterns
 - Partial Hydration & Streaming
 - Optimistic UI with Actions
 - Skeleton Components
 - Best Practices
- Middleware
 - What You'll Learn
 - What is Middleware?
 - Layout-Based Middleware
 - Authentication Middleware
 - Basic Auth Check

- Save Return URL
- Authorization Middleware
 - Role-Based Access
 - Multiple Roles
 - Permission-Based Access
- Subscription Middleware
- Feature Flags
- Request Logging
- Rate Limiting
- Maintenance Mode
- Geo-Blocking
- A/B Testing
- Middleware Composition
- Page-Level Middleware
- Redirect Patterns
 - Authenticated to Dashboard
 - Role-Based Landing
- Navigation Guards
- Best Practices
 - Use Layouts for Groups
 - Show Loading States
 - Redirect Early
 - Use Effect for Side Effects
 - Compose Middleware
- Complete Example
- Summary
- Parallel Routes
 - Basic Parallel Routes
 - Defining Parallel Routes
 - Layout with Slots
 - Slot Components
 - Navigation in Slots
 - Independent Navigation
 - Coordinated Navigation
 - Loading States
 - Per-Slot Loading
 - Streaming Parallel Data
 - Conditional Slots
 - Show/Hide Slots
 - Dynamic Slot Selection
 - Modal Routes
 - Modal as Parallel Route
 - Split Views
 - Email Client Example
 - Error Boundaries
 - Per-Slot Error Handling
 - Graceful Degradation
 - State Management
 - Shared State Between Slots
 - Best Practices
 - □ Do: Use for Independent UI Sections
 - □ Do: Handle Missing Slots
 - □ Do: Keep Slots Independent
 - □ Don't: Overuse Parallel Routes
 - Next Steps
- Intercepting Routes
 - Basic Interception
 - Intercept Convention
 - Photo Modal Example
 - Login Modal
 - Intercept Login Route
 - E-Commerce Product Quick View
 - Quick View Modal
 - Share Dialog
 - Intercept Share Route

- [Settings Drawer](#)
 - [Side Drawer Interception](#)
- [Preserving State](#)
 - [Background State Management](#)
- [Nested Interceptions](#)
 - [Multi-Level Interception](#)
- [Conditional Interception](#)
 - [Device-Based Interception](#)
- [Best Practices](#)
 - [□ Do: Preserve Background State](#)
 - [□ Do: Provide Full Page Alternative](#)
 - [□ Do: Handle Direct Navigation](#)
 - [□ Don't: Intercept Critical Flows](#)
- [Next Steps](#)
- [Smart Preloading](#)
 - [Table of Contents](#)
 - [Overview](#)
 - [Why Smart Preloading?](#)
 - [Quick Start](#)
 - [1. Initialize Preloader](#)
 - [2. Automatic Preloading](#)
 - [3. Manual Control](#)
 - [Preload Strategies](#)
 - [1. Intent Prediction \('intent'\)](#)
 - [2. Hover \('hover'\)](#)
 - [3. Visible \('visible'\)](#)
 - [4. Eager \('eager'\)](#)
 - [5. Manual \('manual'\)](#)
 - [API Reference](#)
 - [SmartPreloader Class](#)
 - [Helper Functions](#)
 - [Intent Prediction](#)
 - [Intent Prediction](#)
 - [How It Works](#)
 - [Tuning Intent Threshold](#)
 - [Complete Examples](#)
 - [Example 1: E-Commerce Product List](#)
 - [Example 2: Navigation Menu with Hover](#)
 - [Example 3: Conditional Preloading](#)
 - [Example 4: Priority Queue](#)
 - [Example 5: History-Based Prediction](#)
 - [Example 6: Network-Aware Preloading](#)
 - [Best Practices](#)
 - [1. Choose the Right Strategy](#)
 - [2. Limit Concurrent Preloads](#)
 - [3. Tune Intent Threshold](#)
 - [4. Record Navigation History](#)
 - [5. Monitor Performance](#)
 - [Advanced Usage](#)
 - [Custom Intent Algorithm](#)
 - [Integration with Analytics](#)
 - [Prefetch vs Preload](#)
 - [Disable on Mobile](#)
 - [Related Documentation](#)
 - [Troubleshooting](#)
 - [Issue: Too Many Preloads](#)
 - [Issue: Preloads Not Working](#)
 - [Issue: Wrong Routes Preloading](#)
 - [Route Discovery](#)
 - [How discovery works](#)
 - [Naming conventions](#)
 - [Controlling discovery](#)
 - [Validation and diagnostics](#)
 - [Route discovery and preloading](#)
 - [Checklist](#)
 - [View Transitions](#)

- What You'll Learn
 - What are View Transitions?
 - Basic Transitions
 - Enable View Transitions
 - Manual Transition
 - Custom Animations
 - Slide Transition
 - Fade and Scale
 - Directional Transitions
 - Forward/Back Navigation
 - Shared Element Transitions
 - Basic Shared Element
 - Hero Image Transition
 - Multiple Shared Elements
 - Transition Types
 - Page Type Transitions
 - Route-Based Transitions
 - Performance Optimization
 - Skip Transition for Certain Routes
 - Reduce Motion
 - Advanced Patterns
 - Conditional Shared Elements
 - Cleanup Transition Names
 - Nested Transitions
 - Fallback for Unsupported Browsers
 - Progressive Enhancement
 - Feature Detection
 - Complete Examples
 - E-commerce Product Transition
 - Blog Post Transition
 - Best Practices
 - Use Unique Transition Names
 - Keep Animations Short
 - Respect Motion Preferences
 - Clean Up Transition Names
 - Test Fallbacks
 - Browser Support
 - Summary
- API Routes
 - Server Handler Basics
 - UI Integration
 - Response Helpers
 - Error Handling
 - Deployment Targets
 - Data Fetching Overview
 - What You'll Learn
 - Data Fetching Patterns
 - Client-Side Fetching
 - Basic Fetch in Effect
 - Custom Fetch Hook
 - Server Functions
 - Queries
 - Mutations
 - Static Generation
 - Server-Side Rendering
 - Caching Strategies
 - Memory Cache
 - Time-Based Cache
 - Loading States
 - Skeleton Loading
 - Suspense
 - Error Handling
 - Retry Logic
 - Error Boundaries
 - Real-Time Updates

- WebSocket
- Polling
- Best Practices
 - Centralize Data Fetching
 - Handle Loading and Errors
 - Cache Wisely
 - Type Your Data
- Comparison Table
- Summary
- Queries
 - What You'll Learn
 - What are Queries?
 - Basic Queries
 - Creating a Query
 - Using a Query
 - Query Keys
 - Simple Keys
 - Parameterized Keys
 - Complex Keys
 - Query Options
 - Stale Time
 - Cache Time
 - Refetch Interval
 - Refetch on Window Focus
 - Retry
 - Dependent Queries
 - Conditional Queries
 - Parallel Queries
 - Query Refetching
 - Manual Refetch
 - Refetch All
 - Refetch by Key
 - Query Invalidation
 - Invalidate Multiple
 - Optimistic Updates
 - Pagination
 - Offset-Based Pagination
 - Cursor-Based Pagination
 - Infinite Queries
 - Prefetching
 - Query Status
 - Fine-Grained Status
 - Best Practices
 - Use Query Keys Wisely
 - Set Appropriate Stale Times
 - Handle Loading States
 - Invalidate Related Queries
 - Use TypeScript
 - Summary
- Mutations
 - What You'll Learn
 - What are Mutations?
 - Basic Mutations
 - Creating a Mutation
 - Using a Mutation
 - Mutation Callbacks
 - onSuccess
 - onError
 - onSettled
 - Cache Invalidation
 - Invalidate Multiple Queries
 - Optimistic Updates
 - Optimistic List Update
 - Mutation State
 - Sequential Mutations
 - Parallel Mutations

- Form Mutations
- Retry Logic
- Mutation Context
- Global Mutation Defaults
- Mutation Middleware
- Complete Examples
 - Todo App
 - E-commerce Cart
- Best Practices
 - Always Invalidate Queries
 - Use Optimistic Updates Wisely
 - Handle Errors
 - Provide Loading States
- Summary
- Optimistic Updates
 - What You'll Learn
 - What are Optimistic Updates?
 - Basic Pattern
 - Simple Optimistic Update
 - With Mutations
 - Mutation with Optimistic Update
 - List Updates
 - Add Item to List
 - Update Item in List
 - Remove Item from List
 - Complex Updates
 - Multi-Query Updates
 - Nested Data Updates
 - Batch Operations
 - Multiple Optimistic Updates
 - Race Conditions
 - Handle Concurrent Updates
 - Partial Success
 - Some Operations Succeed, Some Fail
 - Visual Feedback
 - Show Optimistic State
 - Error Handling
 - User-Friendly Rollback
 - Best Practices
 - Use for Simple Operations
 - Always Save Previous State
 - Provide Visual Feedback
 - Cancel Ongoing Requests
 - Sync with Server
 - Summary
- Pagination
 - What You'll Learn
 - Offset Pagination
 - Basic Offset Pagination
 - Using Offset Pagination
 - With Query
 - Cursor-Based Pagination
 - Basic Cursor Pagination
 - Using Cursor Pagination
 - Infinite Scroll
 - Basic Infinite Scroll
 - Using Infinite Scroll
 - Intersection Observer
 - Page Numbers
 - Numbered Pagination
 - Server-Side Pagination
 - API Response Format
 - API Implementation
 - Search with Pagination
 - Paginated Search

- [Virtual Scrolling](#)
- [Best Practices](#)
 - [Choose the Right Strategy](#)
 - [Keep Previous Data](#)
 - [Provide Loading States](#)
 - [Handle Empty States](#)
 - [Debounce Search](#)
- [Summary](#)
- [Prefetching](#)
 - [Link Prefetching](#)
 - [Automatic Prefetch on Hover](#)
 - [Prefetch Strategies](#)
 - [Manual Prefetching](#)
 - [Programmatic Prefetch](#)
 - [Prefetch with Data](#)
 - [Query Prefetching](#)
 - [Prefetch in Component](#)
 - [Route Prefetching](#)
 - [Prefetch Related Routes](#)
 - [Best Practices](#)
 - [Do: Prefetch Likely Next Steps](#)
 - [Don't: Over-Prefetch](#)
 - [Next Steps](#)
- [Caching Strategies](#)
 - [What You'll Learn](#)
 - [Cache Basics](#)
 - [How Caching Works](#)
 - [Cache Lifecycle](#)
 - [Stale Time](#)
 - [Stale Time Patterns](#)
 - [Cache Time](#)
 - [Cache Time vs Stale Time](#)
 - [Cache Invalidation](#)
 - [Manual Invalidation](#)
 - [Automatic Invalidation](#)
 - [Conditional Invalidation](#)
 - [Direct Cache Updates](#)
 - [Set Query Data](#)
 - [Update List Cache](#)
 - [Optimistic Updates](#)
 - [Cache Persistence](#)
 - [LocalStorage Persistence](#)
 - [Custom Storage](#)
 - [Selective Persistence](#)
 - [Background Refetching](#)
 - [Refetch on Window Focus](#)
 - [Refetch on Reconnect](#)
 - [Polling](#)
 - [Conditional Polling](#)
 - [Memory Management](#)
 - [Automatic Cleanup](#)
 - [Manual Cleanup](#)
 - [Cache Size Limits](#)
 - [Advanced Patterns](#)
 - [Hierarchical Invalidation](#)
 - [Smart Cache Updates](#)
 - [Partial Updates](#)
 - [Batch Updates](#)
 - [Cache Debugging](#)
 - [Inspect Cache](#)
 - [Cache Events](#)
 - [Best Practices](#)
 - [Choose Appropriate Stale Times](#)
 - [Invalidate Proactively](#)
 - [Use Optimistic Updates Carefully](#)
 - [Persist Important Data](#)

- Clean Up Regularly
- Summary
- Loading States
 - Basic Loading States
 - Component-Level Loading
 - Skeleton Screens
 - Suspense Boundaries
 - Basic Suspense
 - Nested Suspense
 - Progressive Loading
 - Critical First, Then Rest
 - Global Loading Indicators
 - Top Loading Bar
 - Best Practices
 - □ Do: Show Content Structure
 - □ Do: Use Optimistic UI
 - Next Steps
- Error Handling for Data Fetching
 - What You'll Learn
 - Basic Error Handling
 - Simple Try-Catch
 - Using Error State
 - Retry Logic
 - Automatic Retry
 - Retry with State
 - Display Retry Count
 - Query Error Handling
 - With createQuery
 - Error Types
 - Custom Error Classes
 - Handle Different Error Types
 - Error Boundaries
 - Data Error Boundary
 - Fallback Data
 - Cached Fallback
 - LocalStorage Fallback
 - User-Friendly Error Messages
 - Error Message Mapping
 - Partial Failures
 - Some Requests Succeed
 - Error Logging
 - Log to Service
 - Best Practices
 - Always Handle Errors
 - Provide Retry Mechanism
 - Show Loading States
 - Use Appropriate Retry Strategies
 - Log Errors for Debugging
 - Summary
- Real-Time Data
 - What You'll Learn
 - WebSockets
 - Basic WebSocket Connection
 - Using WebSocket
 - Send Messages
 - Auto-Reconnect
 - Server-Sent Events (SSE)
 - Basic SSE Connection
 - Named Events
 - Server Implementation
 - Polling
 - Basic Polling
 - Adaptive Polling
 - Conditional Polling
 - Real-Time Queries

- Presence
- Typing Indicators
- Collaborative Editing
- Best Practices
 - Choose the Right Technology
 - Handle Reconnection
 - Clean Up Connections
 - Throttle Updates
 - Show Connection Status
- Summary
- Server Functions
 - What You'll Learn
 - What are Server Functions?
 - Basic Server Functions
 - Creating a Server Function
 - Calling from Client
 - Type Safety
 - Full Type Inference
 - Complex Return Types
 - CRUD Operations
 - Create
 - Read
 - Update
 - Delete
 - Authentication
 - Check Current User
 - Protected Server Functions
 - Authorization
 - Role-Based Access
 - Resource Ownership
 - Validation
 - Zod Schemas
 - Custom Validation
 - Error Handling
 - Custom Error Types
 - Client-Side Error Handling
 - Complex Operations
 - Transactions
 - Batch Operations
 - File Operations
 - File Upload
 - Caching
 - Response Caching
 - Conditional Caching
 - Background Jobs
 - Queue Tasks
 - Best Practices
 - Keep Functions Small
 - Validate All Input
 - Always Authenticate
 - Use Descriptive Names
 - Summary
- Server-Side Rendering (SSR)
 - What You'll Learn
 - What is SSR?
 - Basic SSR
 - Fetch on Server
 - With Request Context
 - Authentication
 - Protected Pages
 - Role-Based Access
 - Personalization
 - User-Specific Content
 - Geo-Location
 - Dynamic Content
 - Search Results

- Filtered Lists
 - Error Handling
 - Not Found
 - Custom Error
 - Caching
 - HTTP Caching
 - Conditional Caching
 - Performance
 - Parallel Data Fetching
 - Minimal Data
 - Database Connection Pooling
 - SSR vs SSG
 - When to Use Each
 - Hybrid Approach
 - Best Practices
 - Minimize Server Time
 - Use Appropriate Caching
 - Handle Errors Gracefully
 - Redirect Early
 - Log Performance
 - Summary
- Static Site Generation (SSG)
- What You'll Learn
 - What is SSG?
 - Basic Static Generation
 - Simple Static Page
 - With Data Fetching
 - Dynamic Paths
 - Generate Static Params
 - Nested Dynamic Routes
 - Incremental Static Regeneration (ISR)
 - Revalidate After Time
 - On-Demand Revalidation
 - Fallback Behavior
 - Static Paths with Fallback
 - SEO Optimization
 - Meta Tags
 - Structured Data
 - Sitemap Generation
 - Markdown/MDX Support
 - Blog from Markdown Files
 - Build Optimization
 - Parallel Generation
 - Selective Generation
 - Hybrid: SSG + Client-Side
 - Static Shell, Dynamic Data
 - Best Practices
 - Use for Stable Content
 - Set Appropriate Revalidation
 - Generate Popular Paths
 - Use Fallback Wisely
 - Optimize Images
 - Summary
- Forms
- Why use the form API
 - Validation strategies
 - Error rendering
 - Submission patterns
 - File uploads
 - Testing forms
 - Checklist
 - Try it now: form + action pair
- SSR Overview
- Basic Node server
 - Render to string

- Streaming SSR
- Architecture
- Hydration and resumability
- Caching and revalidate
- Edge vs regional SSR
- Rendering modes
- Environment and secrets
- Error handling
- Testing SSR
- Try it now: edge-ready stream
- Islands
 - Mark interactive components
 - Hydrate on the client
 - Common patterns
 - Choosing hydration strategies
 - Splitting islands
 - Data flow
 - Styling and assets
 - Testing islands
 - Checklist
 - Try it now: interaction-hydrated menu
- SSR/Islands Playbook
 - Before you start
 - Streaming checklist
 - Islands checklist
 - Caching + revalidate
 - Env and secrets
 - Error handling
 - Testing SSR
 - Deployment steps (generic)
 - Runbook: hydration errors
 - Runbook: slow TTFB
 - Runbook: large HTML
 - Quick wins
- State Management Deep Dive
 - Signals vs Store
 - Creating a store
 - Derived selectors
 - Middleware
 - History (undo/redo)
 - Persistence
 - Custom drivers
 - Patterns
 - Testing stores
 - Try it now: guarded store with history and persistence
- Store Middleware Cookbook
 - Logging middleware
 - Feature flag guard
 - Schema validation (Zod)
 - Analytics hook
 - Prevent large payloads
 - Checklist
- Advanced State Patterns
 - History and time travel
 - Collaboration
 - Cross-tab and multi-window
 - Performance tuning
 - Security and isolation
 - Testing
 - Checklist
- Styling Options
 - CSS modules
 - Scoped styles
 - CSS-in-JS
 - Design tokens and theming
 - Tailwind adapter

- [Tokens reference](#)
- [Styling performance](#)
- [Accessibility](#)
- [Testing styles](#)
- [Checklist](#)
- [Try it now: theme toggle](#)
- [Design Systems with PhilJS](#)
 - [Tokens](#)
 - [Components](#)
 - [Theming](#)
 - [Documentation](#)
 - [Distribution](#)
 - [Testing](#)
 - [Performance](#)
 - [Checklist](#)
- [Motion and Transitions](#)
 - [Principles](#)
 - [Micro-interactions](#)
 - [Page and route transitions](#)
 - [Performance](#)
 - [Accessibility](#)
 - [Implementation sketch](#)
 - [Testing motion](#)
 - [Checklist](#)
- [Design Tokens Reference](#)
 - [Token categories](#)
 - [Example \(CSS variables\)](#)
 - [Theming](#)
 - [Distribution](#)
 - [Testing and QA](#)
 - [Checklist](#)
- [Testing](#)
 - [Run tests](#)
 - [Test layers](#)
 - [Mocking and data](#)
 - [Coverage and budgets](#)
 - [Playwright smoke template](#)
 - [Testing loaders/actions](#)
 - [CI pipeline template](#)
 - [Accessibility checks](#)
 - [Fixtures and data strategy](#)
 - [Checklist](#)
 - [Try it now: loader integration test](#)
- [Testing Playbook \(PhilJS\)](#)
 - [Layers and tools](#)
 - [Daily habits](#)
 - [Component testing recipe](#)
 - [Loaders/actions integration recipe](#)
 - [E2E smoke suite \(minimum\)](#)
 - [CI pipeline](#)
 - [Snapshots policy](#)
 - [Flake reduction](#)
 - [A11y](#)
 - [Performance and budgets](#)
 - [When bugs happen](#)
 - [Coverage focus](#)
- [End-to-End \(E2E\) Testing with Playwright](#)
 - [Setup](#)
 - [Core smoke suite](#)
 - [Writing tests](#)
 - [Fixtures and data](#)
 - [Traces and debugging](#)
 - [Performance checks](#)
 - [Accessibility checks](#)
 - [CI tips](#)

- Checklist
- Property-Based and Fuzz Testing
 - When to use
 - Tooling
 - Defining properties
 - Example
 - Fuzzing loaders/actions
 - Testing guidance
 - Checklist
- Accessibility Testing
 - Unit/component level
 - Integration
 - E2E (Playwright)
 - Contrast and visuals
 - Checklist
- Performance Overview
 - What You'll Learn
 - Performance Fundamentals
 - Key Metrics
 - Performance Budget
 - Optimization Strategies
 - 1. Code Splitting
 - 2. Memoization
 - 3. Virtual Scrolling
 - 4. Bundle Optimization
 - Common Bottlenecks
 - 1. Unnecessary Re-renders
 - 2. Large Bundle Size
 - 3. Inefficient Updates
 - Measurement Tools
 - Performance Observer
 - Custom Timing
 - Best Practices
 - Load JavaScript Efficiently
 - Optimize Images
 - Minimize Main Thread Work
 - Reduce Layout Shifts
 - Performance Checklist
 - Initial Load
 - Runtime Performance
 - Network
 - Code Quality
 - Monitoring
 - Real User Monitoring
 - Summary
- Performance Budgets
 - What You'll Learn
 - What Are Performance Budgets?
 - Common Budget Categories
 - Setting Budgets
 - Analyze Current Performance
 - Competitive Analysis
 - Monitoring Budgets
 - Build-Time Budget Checking
 - Using size-limit
 - Using bundlesize
 - Runtime Budget Monitoring
 - Track Web Vitals
 - Monitor Bundle Sizes
 - CI/CD Integration
 - GitHub Actions
 - Lighthouse CI
 - Enforcement Strategies
 - Fail Builds on Budget Violation
 - Warning System
 - Best Practices

- Set Realistic Budgets
 - Budget by Route
 - Track Trends Over Time
 - Regular Budget Reviews
 - Summary
- Performance Budgets
 - Setting Budgets
 - Configure Budgets
 - Monitoring Budgets
 - CI Integration
 - Best Practices
 - □ Do: Set Realistic Budgets
 - Next Steps
- Bundle Size Optimization
 - Analyzing Bundle Size
 - Build Analysis
 - Bundle Buddy
 - Tree Shaking
 - Import Only What You Need
 - Code Splitting
 - Route-Based Splitting
 - Component-Level Splitting
 - Reduce Dependencies
 - Analyze Dependencies
 - Minification
 - Production Build
 - Best Practices
 - □ Do: Use Dynamic Imports
 - □ Do: Remove Dead Code
 - Next Steps
- Bundle Optimization
 - What You'll Learn
 - Bundle Analysis
 - Analyze Bundle Size
 - Monitor Bundle Growth
 - Tree Shaking
 - Import Only What You Need
 - Mark Side-Effect-Free Code
 - Dependency Optimization
 - Replace Large Libraries
 - Use Lighter Alternatives
 - Dynamic Imports for Heavy Features
 - Code Splitting Strategy
 - Split by Route
 - Vendor Splitting
 - Compression
 - Enable Gzip/Brotli
 - Server Configuration
 - Minification
 - Optimize Production Build
 - CSS Minification
 - Asset Optimization
 - Image Optimization
 - Font Subsetting
 - Remove Unused Code
 - Remove Development Code
 - Purge Unused CSS
 - Module Federation
 - Share Dependencies
 - Best Practices
 - Audit Dependencies Regularly
 - Use Modern JavaScript
 - Lazy Load Non-Critical Features
 - Monitor Third-Party Scripts
 - Performance Budget

- Summary
- [Code Splitting](#)
 - [What You'll Learn](#)
 - [Route-Based Splitting](#)
 - [Lazy Load Routes](#)
 - [With Loading States](#)
 - [Component-Based Splitting](#)
 - [Lazy Load Heavy Components](#)
 - [Modal Code Splitting](#)
 - [Dynamic Imports](#)
 - [Conditional Loading](#)
 - [Library Code Splitting](#)
 - [Chunk Optimization](#)
 - [Manual Chunks](#)
 - [Smart Chunking](#)
 - [Preloading Strategies](#)
 - [Prefetch Routes](#)
 - [Intelligent Preloading](#)
 - [Link Prefetching](#)
 - [Error Handling](#)
 - [Lazy Load Error Boundaries](#)
 - [Retry Failed Chunks](#)
 - [Best Practices](#)
 - [Split by Route](#)
 - [Lazy Load Below the Fold](#)
 - [Avoid Over-Splitting](#)
 - [Use Suspense Wisely](#)
 - [Measuring Impact](#)
 - [Bundle Analysis](#)
 - [Performance Monitoring](#)
 - Summary
- [Lazy Loading](#)
 - [What You'll Learn](#)
 - [Image Lazy Loading](#)
 - [Native Lazy Loading](#)
 - [Intersection Observer](#)
 - [Progressive Image Loading](#)
 - [Blur Hash Implementation](#)
 - [Component Lazy Loading](#)
 - [Visibility-Based Loading](#)
 - [Scroll-Based Loading](#)
 - [Background Loading](#)
 - [Idle Time Loading](#)
 - [Priority Queue Loading](#)
 - [Video Lazy Loading](#)
 - [Lazy Video Element](#)
 - [Autoplay on Visible](#)
 - [Resource Hints](#)
 - [Prefetch Resources](#)
 - [Preload Critical Resources](#)
 - [Font Loading](#)
 - [Font Display Swap](#)
 - [Progressive Font Loading](#)
 - [Best Practices](#)
 - [Lazy Load Below the Fold](#)
 - [Set Image Dimensions](#)
 - [Use Appropriate Root Margin](#)
 - [Prioritize Critical Resources](#)
 - [Clean Up Observers](#)
 - Summary
- [Memoization](#)
 - [What You'll Learn](#)
 - [Memo Basics](#)
 - [Simple Memoization](#)
 - [Derived State](#)
 - [Component Memoization](#)

- Memo Component Wrapper
 - Custom Comparison
 - Selector Patterns
 - Memoized Selectors
 - Parameterized Selectors
 - Advanced Memoization
 - Deep Memoization
 - LRU Cache Memoization
 - Callback Memoization
 - Stable Callbacks
 - When to Memoize
 - Expensive Computations
 - Derived State
 - Component Re-renders
 - Best Practices
 - Measure Before Memoizing
 - Avoid Premature Optimization
 - Use Proper Dependencies
 - Clear Memoization When Needed
 - Memoize at Right Level
 - Summary
- Memory Management
 - What You'll Learn
 - Common Memory Leaks
 - Event Listeners
 - Timers and Intervals
 - Observers
 - Effect Cleanup
 - Cleanup Pattern
 - Conditional Cleanup
 - Preventing Closures Leaks
 - Avoid Capturing Large Objects
 - WeakMap for Caching
 - Detaching DOM Elements
 - Remove References
 - Clear Large Arrays
 - Memory Profiling
 - Detect Memory Leaks
 - Performance Observer for Memory
 - Best Practices
 - Use Weak References
 - Limit Cache Size
 - Avoid Global Variables
 - Clean Up Subscriptions
 - Nullify Large Objects
 - Debugging Memory Issues
 - Chrome DevTools Memory Profiler
 - Memory Leak Test Pattern
 - Summary
- Profiling
 - What You'll Learn
 - Chrome DevTools
 - Performance Panel
 - CPU Profiling
 - Memory Profiling
 - Performance API
 - User Timing API
 - Performance Observer
 - Custom Profiling
 - Component Render Profiling
 - Effect Profiling
 - Function Timing
 - Flame Graphs
 - Generate Flame Graph Data
 - Network Profiling

- Track API Calls
 - Resource Timing
 - Real User Monitoring
 - Web Vitals Tracking
 - Custom Metrics
 - Best Practices
 - Profile in Production Mode
 - Use Appropriate Sample Size
 - Clean Up Performance Marks
 - Focus on User-Centric Metrics
 - Summary
- Runtime Performance
- What You'll Learn
 - Event Optimization
 - Debouncing
 - Throttling
 - Passive Event Listeners
 - Rendering Performance
 - Batch Updates
 - Avoid Layout Thrashing
 - RequestAnimationFrame
 - JavaScript Optimization
 - Avoid Blocking Operations
 - Optimize Loops
 - Early Returns
 - DOM Operations
 - Minimize DOM Access
 - Use Document Fragments
 - Avoid Inline Styles
 - Web Workers
 - Offload Heavy Tasks
 - Worker Pool
 - Best Practices
 - Use Will-Change Sparingly
 - Optimize Images
 - Avoid Memory Leaks
 - Use CSS Containment
 - Summary
- Runtime Performance
- Measuring Performance
 - Performance API
 - React DevTools Profiler
 - Optimize Renders
 - Use Memo
 - Batch Updates
 - Avoid Expensive Operations
 - Debounce
 - Best Practices
 - □ Do: Profile First
 - □ Don't: Premature Optimize
 - Next Steps
- Web Vitals
- Core Web Vitals
 - LCP (Largest Contentful Paint)
 - FID (First Input Delay)
 - CLS (Cumulative Layout Shift)
 - Monitoring
 - Web Vitals Library
 - Best Practices
 - □ Do: Monitor in Production
 - Next Steps
- Image Optimization
- Responsive Images
 - Picture Element
 - Lazy Loading
 - Native Lazy Loading

- Intersection Observer
- Next Steps
- Server-Side Performance
 - SSR Optimization
 - Streaming SSR
 - Edge Rendering
 - Caching
 - Cache Headers
 - Best Practices
 - □ Do: Cache Static Content
 - Next Steps
- Virtual Scrolling
 - What You'll Learn
 - Basic Virtual Scrolling
 - Fixed Height Items
 - With Overscan
 - Variable Height Items
 - Dynamic Height Virtual Scroller
 - Bidirectional Scrolling
 - Grid Virtual Scroller
 - Performance Optimization
 - Throttled Scroll
 - RequestAnimationFrame Optimization
 - Infinite Scroll
 - Load More on Scroll
 - Best Practices
 - Use Fixed Heights When Possible
 - Add Overscan for Smooth Scrolling
 - Throttle Scroll Events
 - Use Stable Keys
 - Measure Heights Efficiently
 - Summary
- Best Practices Overview
 - Introduction
 - Core Principles
 - 1. Explicit Over Implicit
 - 2. Immutability
 - 3. Composition Over Complexity
 - 4. Type Safety
 - 5. Performance by Default
 - Best Practice Categories
 - Component Patterns
 - State Management
 - Performance
 - Testing
 - Code Organization
 - Architecture
 - Security
 - Accessibility
 - Production
 - Quick Reference
 - Signal Best Practices
 - Memo Best Practices
 - Effect Best Practices
 - Component Best Practices
 - Common Anti-Patterns
 - 1. Overusing Effects
 - 2. Prop Drilling
 - 3. Massive Components
 - 4. Premature Optimization
 - Code Quality Standards
 - Naming Conventions
 - File Organization
 - Testing Standards
 - Documentation Standards

- Next Steps
- Application Architecture
 - Architectural Principles
 - Separation of Concerns
 - Layered Architecture
 - Presentation Layer
 - Application Layer
 - Service Layer
 - Infrastructure Layer
 - Dependency Injection
 - Context-Based DI
 - Factory Pattern
 - Plugin Architecture
 - Plugin System
 - Module Federation
 - Micro-Frontend Architecture
 - Event-Driven Architecture
 - Event Bus
 - Domain Events
 - CQRS Pattern
 - Command Query Responsibility Segregation
 - Hexagonal Architecture (Ports & Adapters)
 - Core Domain
 - Scalability Patterns
 - Lazy Module Loading
 - Code Splitting by Route
 - Summary
- Code Organization
 - Project Structure
 - Standard Structure
 - Feature-Based Structure
 - File Naming Conventions
 - Components
 - Utilities and Hooks
 - Types
 - Component Organization
 - Component File Structure
 - Component Folder Pattern
 - Store Organization
 - Simple Store
 - Complex Store
 - Import Organization
 - Import Order
 - Path Aliases
 - Type Organization
 - Shared Types
 - API Types
 - Component Types
 - Service Organization
 - API Service
 - Domain Services
 - Utility Organization
 - Single-Purpose Files
 - Grouped Utilities
 - Constants Organization
 - Environment Variables
 - Barrel Exports
 - Component Barrel
 - Avoid Deep Barrels
 - Documentation
 - Component Documentation
 - Store Documentation
 - Summary
- Component Patterns
 - Component Types
 - Presentational Components

- Container Components
- Composition Patterns
 - Children Pattern
 - Render Props Pattern
 - Compound Components Pattern
- Props Patterns
 - Optional Props with Defaults
 - Discriminated Union Props
 - Polymorphic Components
- State Management Patterns
 - Lifting State Up
 - Local State
 - Controlled vs Uncontrolled
- Error Handling Patterns
 - Error Boundaries
 - Try-Catch in Effects
- Loading States
 - Suspense Pattern
 - Skeleton Screens
- Conditional Rendering
 - Multiple Conditions
 - Render Functions
- Form Patterns
 - Controlled Form
- List Rendering
 - With Keys
 - With Index (When Appropriate)
 - Empty States
- Summary
- Accessibility Best Practices
 - Semantic HTML
 - Use Correct Elements
 - Navigation
 - ARIA Attributes
 - ARIA Labels
 - ARIA Roles
 - ARIA States
 - Keyboard Navigation
 - Focus Management
 - Keyboard Shortcuts
 - Tab Index
 - Screen Reader Support
 - Descriptive Labels
 - Live Regions
 - Hiding Decorative Content
 - Color and Contrast
 - Color Contrast
 - Don't Rely on Color Alone
 - Forms
 - Accessible Forms
 - Images
 - Alt Text
 - Responsive Design
 - Font Sizing
 - Touch Targets
 - Focus Indicators
 - Visible Focus States
 - Testing Accessibility
 - Manual Testing
 - Automated Testing
 - Summary
- State Management Patterns
 - Table of Contents
 - State Categories
 - Local Component State

- Basic Signal State
- Multiple Related Signals
- Object State Pattern
- Shared State Patterns
 - Lifted State (Parent-Child)
 - Custom Hook Pattern
 - Module-Level Shared State
- Global State with Context
 - Basic Context Pattern
 - Signal Context Pattern (Reactive)
 - Multiple Contexts Pattern
 - Theme Context with CSS Variables
 - Global State Store Pattern
- Store Patterns
 - Simple Store
 - Factory Store
 - Async Store
- State Persistence
 - LocalStorage Persistence
 - Generic Persistent Signal
 - SessionStorage for Temporary State
 - IndexedDB for Large Data
- State Synchronization
 - WebSocket Synchronization
 - Server-Sent Events (SSE)
 - Polling Pattern
 - Cross-Tab Synchronization
 - BroadcastChannel API
- State Composition
 - Combining Stores
 - Derived Stores
- State Machines
 - Finite State Machine
- Optimistic Updates
- Undo/Redo
- Performance Optimization
 - Batching Updates
 - Selective Updates
- Advanced Patterns
 - Layered Stores (Domain + UI)
 - Event-Driven State
- Summary
 - Decision Tree: Choosing the Right Pattern
 - Best Practices Summary
- Performance Best Practices
 - Requirements
 - Fine-Grained Reactivity Advantages
 - ES2024 Performance Patterns
 - Use Native Array Methods (ES2024)
 - Use Object.groupBy() and Map.groupBy()
 - Use Promise.withResolvers() for Async Patterns
 - Memoization
 - Use memo() for Expensive Computations
 - When NOT to Use memo()
 - Batching Updates
 - Batch Related Changes
 - Batch in Loops
 - Lazy Loading
 - Code Splitting with lazy()
 - Route-Based Splitting
 - Component-Based Splitting
 - List Virtualization
 - Virtual Scrolling
 - Debouncing and Throttling
 - Debounce Search
 - Throttle Scroll

- [Image Optimization](#)
 - [Lazy Load Images](#)
 - [Progressive Images](#)
- [Bundle Optimization](#)
 - [Import Only What You Need](#)
 - [Dynamic Imports](#)
- [Avoid Unnecessary Work](#)
 - [Skip Effects with untrack\(\)](#)
 - [Conditional Tracking](#)
- [Web Workers](#)
 - [Offload Heavy Computation](#)
- [Measuring Performance](#)
 - [Performance Profiling](#)
 - [Track Effect Performance](#)
- [Memory Management](#)
 - [Clean Up Effects](#)
 - [Avoid Memory Leaks](#)
- [Caching](#)
 - [Memoized API Calls](#)
- [TypeScript 6 Build Performance](#)
 - [Enable Isolated Declarations](#)
 - [Use Strict Type Checking](#)
- [Summary](#)
- [Testing Best Practices](#)
 - [Testing Philosophy](#)
 - [Test Pyramid](#)
 - [Unit Testing](#)
 - [Testing Signals](#)
 - [Testing Memo](#)
 - [Testing Effects](#)
 - [Testing Custom Hooks](#)
 - [Component Testing](#)
 - [Testing Presentational Components](#)
 - [Testing Container Components](#)
 - [Testing Forms](#)
 - [Integration Testing](#)
 - [Testing with Context](#)
 - [Testing with Router](#)
 - [Testing API Integration](#)
 - [E2E Testing](#)
 - [Playwright Tests](#)
 - [Testing Best Practices](#)
 - [Arrange-Act-Assert Pattern](#)
 - [Test One Thing](#)
 - [Use Descriptive Names](#)
 - [Avoid Implementation Details](#)
 - [Coverage Goals](#)
 - [Summary](#)
- [Error Handling Best Practices](#)
 - [Error Boundaries](#)
 - [Async Error Handling](#)
 - [Best Practices](#)
 - [Do: Use Error Boundaries](#)
 - [Do: Log Errors](#)
 - [Next Steps](#)
- [Deployment Best Practices](#)
 - [Build for Production](#)
 - [Environment Variables](#)
 - [Deployment Platforms](#)
 - [Vercel](#)
 - [Netlify](#)
 - [Docker](#)
 - [Best Practices](#)
 - [Do: Use Environment Variables](#)
 - [Do: Enable Compression](#)

- Next Steps
- Security Best Practices
 - XSS Prevention
 - Cross-Site Scripting (XSS)
 - Sanitizing User Input
 - Avoid eval() and Function()
 - CSRF Protection
 - Built-in CSRF Protection
 - Form Protection
 - AJAX Request Protection
 - PhilJS Component Integration
 - Custom Token Storage
 - Extract CSRF Token
 - SameSite Cookies
 - Authentication
 - Secure Token Storage
 - Password Handling
 - Session Management
 - Authorization
 - Role-Based Access Control
 - Route Guards
 - Input Validation
 - Client-Side Validation
 - Server-Side Validation
 - SQL Injection Prevention
 - Parameterized Queries
 - Secrets Management
 - Environment Variables
 - Client vs Server Secrets
 - Content Security Policy
 - CSP Headers
 - Rate Limiting
 - Built-in Rate Limiting
 - Pre-configured Rate Limiters
 - Custom Key Generation
 - Redis Store for Production
 - Sliding Window Rate Limiting
 - Adaptive Rate Limiting
 - Rate Limit Headers
 - Skip Successful/Failed Requests
 - Custom Error Handling
 - Per-Route Rate Limiting
 - Client-Side Rate Limiting
 - Monitor Rate Limit Usage
 - Best Practices
 - Dependency Security
 - Regular Updates
 - Secure Dependencies
 - Security Headers
 - Essential Headers
 - File Upload Security
 - Validate File Uploads
 - Summary
- Production Best Practices
 - Build Optimization
 - Production Build
 - Vite Configuration
 - Environment Configuration
 - Environment Variables
 - Error Tracking
 - Sentry Integration
 - Global Error Boundary
 - Analytics
 - Analytics Setup
 - Performance Monitoring
 - Web Vitals

- Logging
 - Structured Logging
 - Caching Strategy
 - Service Worker
 - Health Checks
 - Application Health
 - Deployment
 - CI/CD Pipeline
 - Docker
 - Zero-Downtime Deployment
 - Monitoring
 - Uptime Monitoring
 - Alerting
 - Security Headers
 - Summary
- TypeScript 6 Best Practices
 - Requirements
 - Recommended tsconfig.json
 - Component Types
 - Signal Types
 - TypeScript 6 Features
 - Use the satisfies Operator
 - Use Nolnfer for Better Generic Control
 - Use Const Type Parameters
 - Isolated Declarations
 - Best Practices
 - Use Type Inference When Possible
 - Type Props Explicitly
 - Use Strict Optional Properties
 - Safe Index Access
 - ES2024 Type Support
 - Next Steps
- Deployment
 - Build
 - Preview
 - Configure output
 - Runtime requirements
 - Deployment Playbook (Advanced)
 - Pre-deploy checklist
 - Edge (Vercel/Netlify/Cloudflare)
 - AWS/Node
 - Static/SSG
 - CI/CD template
 - Observability on deploy
 - Rollback strategy
 - Security
 - Smoke tests post-deploy
 - Checklist
 - Security Overview
 - Table of Contents
 - Security Features
 - Built-in XSS Protection
 - Content Security Policy (CSP)
 - CSRF Protection
 - Secure Cookie Handling
 - Input Validation and Sanitization
 - Security Architecture
 - Defense in Depth
 - SSR Security Model
 - Client-Side Security
 - Common Vulnerabilities
 - Cross-Site Scripting (XSS)
 - Cross-Site Request Forgery (CSRF)
 - Prototype Pollution
 - SQL Injection

- Open Redirect
- Server-Side Request Forgery (SSRF)
- Security Checklist
 - Development Phase
 - Pre-Production
 - Production
 - Post-Deployment
- Security Best Practices
 - 1. Validate Input, Escape Output
 - 2. Use Security Headers
 - 3. Implement Rate Limiting
 - 4. Secure Authentication
 - 5. Environment Variables for Secrets
- Dependency Security
 - Regular Audits
 - Current Vulnerabilities (as of audit)
 - Mitigation Steps
- Reporting Security Issues
- Additional Resources
- Version History
- Security Policies and Playbooks
 - Policies to define
 - Incident response
 - Regular tasks
 - Edge/serverless considerations
 - Testing and automation
 - Communication
 - Checklist
- API Security Guide
 - Table of Contents
 - API Security Overview
 - CSRF Protection
 - Implementation
 - Token in Headers
 - Double Submit Cookie
 - Rate Limiting
 - Basic Rate Limiting
 - Advanced Rate Limiting
 - Token Bucket Algorithm
 - Input Validation
 - Schema Validation
 - Manual Validation
 - SQL Injection Prevention
 - NoSQL Injection Prevention
 - Authentication
 - Bearer Token
 - API Keys
 - Authorization
 - Role-Based Access Control (RBAC)
 - Attribute-Based Access Control (ABAC)
 - Data Protection
 - Encrypt Sensitive Data
 - Hash Sensitive Data
 - Error Handling
 - Secure Error Responses
 - Error Logging
 - Security Headers
 - API Security Checklist
 - Resources
- Authentication Patterns
 - Table of Contents
 - Authentication Overview
 - Choosing an Authentication Strategy
 - Session-Based Authentication
 - Implementation
 - Session Storage

- Token-Based Authentication
 - JWT Implementation
 - Refresh Tokens
 - OAuth and Social Login
 - OAuth 2.0 Flow
 - Password Security
 - Password Hashing
 - User Registration
 - Password Reset
 - Multi-Factor Authentication
 - TOTP (Time-Based One-Time Password)
 - Best Practices
 - 1. Use HTTPS Everywhere
 - 2. Implement Rate Limiting
 - 3. Secure Cookie Settings
 - 4. Validate Sessions Server-Side
 - 5. Log Security Events
 - Security Checklist
 - Resources
- Content Security Policy Guide
 - Table of Contents
 - Understanding CSP
 - How CSP Works
 - CSP in PhilJS
 - Basic Setup
 - With Auto-Generated Nonce
 - Configuration
 - Default Directives
 - Strict CSP
 - Development CSP
 - Custom Directives
 - Nonce-Based Scripts
 - Server-Side Implementation
 - SSR with Nonces
 - Client-Side Hydration
 - Common Patterns
 - Pattern 1: Static Site
 - Pattern 2: SPA with CDN
 - Pattern 3: Analytics Integration
 - Pattern 4: Embedding Content
 - Pattern 5: Report-Only Mode
 - Directive Reference
 - Fetch Directives
 - Document Directives
 - Navigation Directives
 - Reporting Directives
 - Other Directives
 - Testing and Debugging
 - Enable Report-Only Mode
 - Use Browser DevTools
 - Validate CSP
 - Test CSP Online
 - Migration Guide
 - Step 1: Start with Report-Only
 - Step 2: Monitor Violations
 - Step 3: Fix Violations
 - Step 4: Enforce Policy
 - Best Practices
 - 1. Start Strict, Relax as Needed
 - 2. Use Nonces, Not unsafe-inline
 - 3. Avoid unsafe-eval
 - 4. Use SRI for CDN Resources
 - 5. Review Regularly
 - Troubleshooting
 - Issue: Inline Styles Blocked

- Issue: Third-Party Scripts Blocked
 - Issue: WebSocket Connections Blocked
 - Issue: Image Loading Issues
 - Resources
 - Summary
- XSS Prevention Guide
 - Table of Contents
 - Understanding XSS
 - Types of XSS
 - PhilJS Built-in Protection
 - 1. Automatic HTML Escaping
 - 2. Attribute Escaping
 - 3. Server-Side Rendering Protection
 - Context-Specific Escaping
 - HTML Context
 - Attribute Context
 - JavaScript Context
 - URL Context
 - Safe Content Rendering
 - Rich Text Content
 - Markdown Content
 - User Avatars and Images
 - Common XSS Patterns
 - Pattern 1: Event Handlers
 - Pattern 2: Dynamic Styles
 - Pattern 3: Dynamic URLs
 - Pattern 4: JSON in Script Tags
 - Pattern 5: SVG Content
 - Best Practices
 - 1. Never Trust User Input
 - 2. Use Content Security Policy
 - 3. Avoid dangerouslySetInnerHTML
 - 4. Validate URLs
 - 5. Escape Data in All Contexts
 - Testing for XSS
 - Manual Testing
 - Automated Testing
 - Unit Tests
 - Security Checklist
 - Resources
 - Summary
 - Security Audit Documentation
 - Table of Contents
 - XSS Prevention
 - How PhilJS Prevents XSS
 - Common XSS Vectors and Protections
 - XSS Prevention Rules
 - Safe HTML Rendering
 - When You Need Raw HTML
 - Server-Side Sanitization
 - Input Sanitization
 - Input Validation Patterns
 - Form Validation
 - CSRF Protection
 - How CSRF Protection Works
 - Server-Side Setup
 - Form Protection
 - AJAX/Fetch Protection
 - Token Storage
 - Extract CSRF Token from Requests
 - SameSite Cookies
 - Content Security Policy
 - CSP Implementation
 - Default CSP Directives
 - Using Nonces for Inline Scripts
 - CSP Report-Only Mode

- Handling CSP Reports
 - Strict CSP
- Rate Limiting
 - Basic Rate Limiting
 - Pre-configured Limiters
 - Custom Key Generation
 - Redis Store for Production
 - Rate Limit Headers
 - Skip Successful/Failed Requests
 - Adaptive Rate Limiting
 - Sliding Window Rate Limiting
- Security Headers
 - Essential Security Headers
 - Header Middleware
- Authentication & Authorization
 - Secure Session Management
 - Password Hashing
 - Role-Based Access Control
- Security Checklist
 - Development Checklist
 - Production Checklist
 - Code Review Checklist
- Summary
 - Key Principles
 - Resources
- Security Checklist
 - Pre-Deployment Checklist
 - Application Security
 - Authentication & Authorization
 - API Security
 - Data Protection
 - Infrastructure Security
 - Dependency Security
 - Security Headers
 - Monitoring & Logging
 - Development Checklist
 - Code Security
 - Testing
 - Operational Checklist
 - Incident Response
 - Compliance
 - Business Continuity
 - Periodic Review (Quarterly)
 - Security Review
 - Penetration Testing
 - Training & Awareness
 - PhilJS-Specific Checks
 - Framework Security
 - SSR Security
 - Client Security
 - Quick Start Security Checklist
 - Severity Levels
 - Critical (Fix Immediately)
 - High (Fix Within 24 Hours)
 - Medium (Fix Within 1 Week)
 - Low (Fix in Next Release)
 - Resources
 - Checklist Versions
- Edge Security Considerations
 - Runtime constraints
 - Network and origin policy
 - Data and caching
 - Headers and CSP
 - Webhooks and callbacks
 - Logging and PII

- [Testing](#)
 - [Checklist](#)
- [Observability and Reliability](#)
 - [What to capture](#)
 - [Instrumentation points](#)
 - [Tools](#)
 - [Error handling](#)
 - [Alerting and SLOs](#)
 - [Local debugging](#)
 - [Try it now: structured logging wrapper](#)
 - [Runbooks \(examples\)](#)
- [Logging and Tracing Deep Dive](#)
 - [Logging](#)
 - [Tracing](#)
 - [Metrics](#)
 - [Pipelines](#)
 - [Dashboards to build](#)
 - [Alerting](#)
 - [Testing instrumentation](#)
 - [Checklist](#)
- [Instrumentation Examples](#)
 - [Logging loader timing](#)
 - [Metrics \(edge-friendly\)](#)
 - [Tracing](#)
 - [Client RUM \(lightweight\)](#)
 - [Testing instrumentation](#)
 - [Checklist](#)
- [Observability Runbooks](#)
 - [Slow TTFB](#)
 - [Hydration errors](#)
 - [Cache misses spike](#)
 - [Memory creep \(client\)](#)
 - [Cold start spikes \(edge/serverless\)](#)
 - [Error rate increase](#)
 - [Runbook hygiene](#)
- [Architecture Overview \(PhilJS\)](#)
 - [Layers](#)
 - [Request/response flow](#)
 - [Data and caching](#)
 - [State strategy](#)
 - [Performance hooks](#)
 - [Observability](#)
 - [Security](#)
 - [Deployment patterns](#)
 - [Checklist](#)
- [Runtime Internals](#)
 - [Reactivity graph](#)
 - [Scheduling](#)
 - [Hydration pipeline](#)
 - [Loader/action lifecycle](#)
 - [Error handling](#)
 - [DevTools hooks](#)
 - [Checklist](#)
- [Data Modeling and Domain Design](#)
 - [Principles](#)
 - [Shapes for loaders](#)
 - [Shapes for stores](#)
 - [Relationships](#)
 - [Validation and migrations](#)
 - [Offline and sync](#)
 - [Testing](#)
 - [Checklist](#)
- [OpenAPI and REST Integration](#)
 - [Generate a typed client](#)
 - [Use in loaders](#)
 - [Mutations and optimistic UI](#)

- Validation
- Caching and invalidation
- Security
- Testing
- Try it now: end-to-end typed loader + mutation
- GraphQL Integration
 - Choosing a client
 - Example with graphql-request
 - Mutations with optimistic updates
 - Caching patterns
 - Subscriptions and live data
 - Schema drift and safety
 - Testing
 - Client patterns to borrow
 - Try it now: SSR + client hydration with urql
- tRPC Integration
 - Setup
 - Loaders and actions
 - SSR considerations
 - Error handling
 - Testing
 - Checklist
- WebSockets and Realtime Updates
 - When to use
 - Client setup
 - Cache coherence
 - Security
 - Performance
 - Testing
 - Checklist
- AI Integration
 - Principles
 - Calling models
 - Guardrails
 - Caching and idempotency
 - UI patterns
 - Testing
 - Observability
 - Checklist
- Payments Integration
 - Principles
 - Client flow (example with Stripe)
 - Server handling
 - UX patterns
 - Testing
 - Security
 - Checklist
- API Versioning and Compatibility
 - Versioning strategies
 - Client impact
 - Change management
 - Testing
 - Observability
 - Checklist
- API Error Handling Patterns
 - Error shapes
 - Client handling (PhilJS)
 - Transport considerations
 - Retries and backoff
 - Testing
 - Checklist
- API Contract Testing
 - Approaches
 - Tooling
 - Patterns

- [Testing levels](#)
- [Drift detection](#)
- [Checklist](#)
- [Authentication and Authorization](#)
 - [Authentication](#)
 - [Authorization](#)
 - [Session handling](#)
 - [CSRF](#)
 - [OAuth/OIDC](#)
 - [API access from loaders/actions](#)
 - [Testing](#)
 - [Checklist](#)
- [PhilJS CLI Reference](#)
 - [Commands](#)
 - [Options \(common\)](#)
 - [Workflow tips](#)
 - [Troubleshooting](#)
 - [Links](#)
- [Tooling and Dev Experience](#)
 - [philjs-cli](#)
 - [DevTools extension](#)
 - [Builder and plugins](#)
 - [Migrate](#)
 - [Linting and type safety](#)
 - [Recommended workflow](#)
 - [Automation ideas](#)
 - [Try it now: inspect + devtools profiling loop](#)
- [Package Atlas](#)
 - [How to read this catalog](#)
 - [Package Index](#)
 - [Package Entries](#)
 - [PhilJS 3D](#)
 - [@philjs/3d-physics](#)
 - [@philjs/a11y-ai](#)
 - [@philjs/ab-testing](#)
 - [philjs-adapters](#)
 - [philjs-ai](#)
 - [@philjs/ai-agents](#)
 - [@philjs/ambient](#)
 - [@philjs/analytics](#)
 - [philjs-api](#)
 - [philjs-atoms](#)
 - [PhilJS Auth](#)
 - [PhilJS Benchmark Suite](#)
 - [@philjs/biometric](#)
 - [@philjs/build](#)
 - [@philjs/builder](#)
 - [@philjs/carbon](#)
 - [@philjs/cdn](#)
 - [PhilJS Cells](#)
 - [@philjs/charts](#)
 - [philjs-cli](#)
 - [@philjs/collab](#)
 - [PhilJS Compiler](#)
 - [PhilJS Content](#)
 - [philjs-core](#)
 - [@philjs/crossdevice](#)
 - [PhilJS CSS](#)
 - [@philjs/dashboard](#)
 - [philjs-db](#)
 - [philjs-desktop](#)
 - [philjs-devtools](#)
 - [PhilJS DevTools Extension](#)
 - [@philjs/digital-twin](#)
 - [@philjs/dnd](#)
 - [PhilJS Documentation](#)

- [@philjs/edge](#)
- [@philjs/edge-ai](#)
- [@philjs/edge-mesh](#)
- [@philjs/editor](#)
- [@philjs/email](#)
- [@philjs/enterprise](#)
- [philjs-errors](#)
- [@philjs/event-sourcing](#)
- [@philjs/export](#)
- [@philjs/eye-tracking](#)
- [philjs-form](#)
- [@philjs/genui](#)
- [@philjs/gesture](#)
- [@philjs/go](#)
- [philjs-graphql](#)
- [@philjs/haptic](#)
- [@philjs/hollow](#)
- [PhilJS HTML](#)
- [@philjs/hypermedia](#)
- [@philjs/i18n](#)
- [PhilJS Image](#)
- [PhilJS Inspector](#)
- [@philjs/intent](#)
- [philjs-islands](#)
- [@philjs/jobs](#)
- [philjs-kotlin](#)
- [PhilJS LiveView](#)
- [@philjs/lm-ui](#)
- [@philjs/maps](#)
- [@philjs/media-stream](#)
- [philjs-meta](#)
- [philjs-migrate](#)
- [@philjs/motion](#)
- [PhilJS Native](#)
- [@philjs/neural](#)
- [@philjs/observability](#)
- [@philjs/offline](#)
- [philjs-openapi](#)
- [PhilJS Optimizer](#)
- [@philjs/payments](#)
- [philjs-pdf](#)
- [@philjs/perf](#)
- [@philjs/perf-budget](#)
- [philjs-playground](#)
- [philjs-plugin-analytics](#)
- [philjs-plugin-i18n](#)
- [@philjs/plugin-pwa](#)
- [philjs-plugin-seo](#)
- [philjs-plugin-tailwind](#)
- [philjs-plugins](#)
- [@philjs/poem](#)
- [@philjs/pwa](#)
- [philjs-python](#)
- [@philjs/qr](#)
- [@philjs/quantum](#)
- [PhilJS Realtime](#)
- [philjs-resumable](#)
- [@philjs/rich-text](#)
- [@philjs/rocket](#)
- [philjs-router](#)
- [philjs-router-typesafe](#)
- [philjs-rpc](#)
- [@philjs/runtime](#)
- [@philjs/scene](#)
- [@philjs/screen-share](#)

- [@philjs/security-scanner](#)
- [@philjs/spatial-audio](#)
- [@philjs/sqlite](#)
- [philjs-ssr](#)
- [@philjs/storage](#)
- [PhilJS Storybook](#)
- [@philjs/studio](#)
- [philjs-styles](#)
- [philjs-swift](#)
- [@philjs/table](#)
- [philjs-tailwind](#)
- [philjs-templates](#)
- [philjs-testing](#)
- [@philjs/time-travel](#)
- [@philjs/trpc](#)
- [philjs-ui](#)
- [@philjs/vector](#)
- [@philjs/vector-store](#)
- [@philjs/video-chat](#)
- [@philjs/virtual](#)
- [@philjs/voice](#)
- [PhilJS for VS Code](#)
- [philjs-wasm](#)
- [@philjs/webgpu](#)
- [@philjs/webrtc](#)
- [@philjs/workers](#)
- [@philjs/workflow](#)
- [@philjs/xr](#)
- [philjs-xstate](#)
- [@philjs/zig](#)
- [philjs-zustand](#)
- [cargo-philjs](#)
- [create-philjs](#)
- [create-philjs-plugin](#)
- [eslint-config-philjs](#)
- [eslint-plugin-philjs](#)
- [PhilJS for Rust](#)
- [PhilJS Actix-web Integration](#)
- [PhilJS Axum Integration](#)
- [philjs-macros](#)
- [@philjs/mobile](#)
- [PhilJS SeaORM Integration](#)
- [PhilJS SQLx Integration](#)
- [@philjs/tauri](#)
- [@philjs/tokio](#)
- [@philjs/tui](#)
- PhilJS Platform Adapters - Complete Implementation
 - [Overview](#)
 - [Cloudflare Pages Adapter](#)
 - [Features](#)
 - [Usage](#)
 - [Generated Files](#)
 - [Helper Functions](#)
 - [Vercel Adapter](#)
 - [Features](#)
 - [Usage](#)
 - [Generated Files](#)
 - [Revalidation API](#)
 - [Netlify Adapter](#)
 - [Features](#)
 - [Usage](#)
 - [Generated Files](#)
 - [Image Optimization](#)
 - [AWS Lambda Adapter](#)
 - [Features](#)
 - [Usage](#)

- Generated Files
 - Integration Types
 - Railway Adapter
 - Features
 - Usage
 - Generated Files
 - Common Features
 - Package Exports
 - Adapter Presets
 - Examples
 - Testing
 - Migration Guide
 - From Basic Cloudflare to Cloudflare Pages
 - From Basic Vercel to Enhanced Vercel
 - Performance Optimizations
 - Security Features
 - Monitoring & Logging
 - Cost Optimization
 - Next Steps
 - Support
- PhilJS Edge Middleware
 - Features
 - Installation
 - Basic Usage
 - Edge Middleware
 - URL Rewrites
 - Redirects
 - Header Manipulation
 - Geolocation
 - Auto-detect Location
 - Geo-based Redirects
 - Language Detection
 - Localized Redirects
 - Client-side Hook
 - A/B Testing
 - Define Experiments
 - Variant-based Rendering
 - Multivariate Testing
 - Client-side Hook
 - Statistical Significance
 - Edge Caching
 - Basic Caching
 - Cache Control Headers
 - ETags for Conditional Requests
 - Cache Presets
 - Cache Purging
 - Vary Headers
 - Complete Example
 - Cloudflare Worker
 - Vercel Edge Function
 - Advanced Features
 - Custom Cache Store
 - Custom Analytics Provider
 - Deterministic Variant Selection
 - Runtime Compatibility
 - Performance
 - Best Practices
 - License
 - Flash Messages & Enhanced Session Management
 - Table of Contents
 - Flash Messages
 - Basic Usage
 - Flash Categories
 - Flash with Metadata
 - Flash Utilities

- Advanced Usage
- Flash Middleware
- Enhanced Cookie Sessions
 - Basic Setup
 - With Encryption
 - Session Rotation
 - CSRF Protection
 - CSRF Middleware
 - Session Rotation Middleware
- Session Utilities
 - Basic Helpers
 - Session Middleware
 - Session Value Helpers
 - Typed Session Utilities
 - Session Timeout Middleware
 - Session Validator Middleware
 - Session Regeneration
- Complete Examples
 - Login with Flash Messages
 - Protected Route with Session Validation
 - Form with CSRF Protection
 - Complete Middleware Stack
- Type Safety
- Security Best Practices
- Migration Guide: Enhanced Sessions
 - Overview
 - Migration Steps
 - Step 1: Update Imports
 - Step 2: Update Session Configuration
 - Step 3: Add Flash Messages
 - Step 4: Add CSRF Protection
 - Step 5: Add Session Timeout
 - Step 6: Add Session Validation
 - Step 7: Update Login Flow
 - Backward Compatibility
 - Type Safety
 - Performance Considerations
 - Encryption
 - Session Rotation
 - Security Checklist
 - Common Patterns
 - Pattern 1: Protected Route
 - Pattern 2: Form with Flash
 - Pattern 3: Middleware Stack
 - Troubleshooting
 - Issue: Session not persisting
 - Issue: CSRF token invalid
 - Issue: Session timeout too aggressive
 - Need Help?
- PhilJS Authentication Guide
 - Table of Contents
 - Quick Start
 - 1. Generate Authentication Setup
 - 2. Install Dependencies
 - 3. Configure Environment Variables
 - 4. Wrap Your App
 - Generators
 - Auth Generator Options
 - Generated Files
 - Providers
 - Provider Abstraction
 - Custom Provider
 - Hooks
 - useAuth()
 - useUser()
 - useHasPermission()

- [useRequireAuth\(\)](#)
- [Protected Routes](#)
 - [ProtectedRoute Component](#)
 - [withAuth HOC](#)
 - [Role-Based Protection](#)
 - [AuthGuard Component](#)
- [Session Management](#)
 - [Automatic Token Refresh](#)
 - [Session Persistence](#)
 - [Logout Everywhere](#)
- [Advanced Usage](#)
 - [Custom Auth Flow](#)
 - [OAuth Sign In](#)
 - [Token Management](#)
 - [Multi-Factor Authentication](#)
- [Best Practices](#)
- [Troubleshooting](#)
 - “Auth provider not initialized” error
 - Tokens not refreshing
 - Redirect loops
- [Examples](#)
- [API Reference](#)
- [PhilJS Database Migrations](#)
 - [Features](#)
 - [Installation](#)
 - [Quick Start](#)
 - 1. Configuration
 - 2. Initialize
 - 3. Write Migration
 - 4. Run Migrations
 - [CLI Commands](#)
 - [Run Migrations](#)
 - [Create Migration](#)
 - [Rollback Migrations](#)
 - [Migration Status](#)
 - [Reset & Fresh](#)
 - [Schema Diff](#)
 - [Migration API](#)
 - [Schema Builder](#)
 - [Data Helpers](#)
 - [SQL Execution](#)
 - [Configuration Options](#)
 - [Database-Specific Features](#)
 - [PostgreSQL](#)
 - [MySQL](#)
 - [SQLite](#)
 - [ORM Integration](#)
 - [Prisma](#)
 - [Drizzle](#)
 - [Advanced Features](#)
 - [Auto-Migration](#)
 - [Schema Diff](#)
 - [Backup & Restore](#)
 - [Data Migration Helpers](#)
 - [Best Practices](#)
 - [Examples](#)
 - [Troubleshooting](#)
 - [Migration conflicts](#)
 - [Transaction errors](#)
 - [Rollback failures](#)
 - [License](#)
- [Quick Start Guide - Multi-Framework Islands](#)
 - [Installation](#)
 - [Step 1: Configure Vite](#)
 - [Step 2: Create Island Components](#)

- [React Component \(`islands/Counter.tsx`\)](#)
 - [Vue Component \(`islands/TodoList.vue`\)](#)
 - Step 3: Register Islands
 - Step 4: Use Islands
 - Done!
 - Next Steps
 - Share State Between Islands
 - Use Event Bus
 - Framework-Specific Hooks
 - Hydration Strategies
 - Common Patterns
 - Multiple Frameworks on Same Page
 - Sharing Data
 - Cross-Island Events
 - Troubleshooting
 - Island not hydrating?
 - Props not working?
 - Framework not detected?
 - Learn More
- [PhilJS Multi-Framework Islands](#)
 - Features
 - Installation
 - Optional Framework Dependencies
 - Quick Start
 - 1. Configure Vite
 - 2. Register Island Components
 - 3. Use Islands in Your App
 - Hydration Strategies
 - Immediate
 - Visible
 - Idle
 - Interaction
 - Media Query
 - Shared State
 - Framework-Specific Hooks
 - Event Bus
 - Island Bridges
 - Props Normalization
 - Framework Adapters
 - Custom Adapter
 - Vite Plugin Options
 - API Reference
 - Multi-Framework Islands
 - Framework Adapters
 - Shared State
 - Event Bus
 - Performance Tips
 - Examples
 - Browser Support
 - License
 - [PhilJS Router - Advanced Features](#)
 - Table of Contents
 - Router DevTools
 - Features
 - Usage
 - Configuration
 - Programmatic Access
 - Route Groups
 - Features
 - Basic Usage
 - Creating Route Groups
 - Middleware
 - Processing Groups
 - File-based Discovery
 - Route Masking
 - Features

- [Basic Usage](#)
- [Modal Pattern](#)
- [Drawer Pattern](#)
- [Nested Masking](#)
- [Hooks](#)
- [Router Context](#)
 - [Features](#)
 - [Basic Usage](#)
 - [Context Providers](#)
 - [Route-specific Context](#)
 - [Context Middleware](#)
 - [Typed Context](#)
 - [Validation](#)
 - [Built-in Helpers](#)
- [Complete Example](#)
- [API Reference](#)
- [Testing](#)
- [Performance Considerations](#)
 - [DevTools](#)
 - [Route Groups](#)
 - [Route Masking](#)
 - [Router Context](#)
- [Browser Support](#)
- [License](#)
- [CI/CD for PhilJS](#)
 - [Pipelines](#)
 - [Caching](#)
 - [Preview environments](#)
 - [Branching and gating](#)
 - [Secrets and env](#)
 - [Observability hooks](#)
 - [Rollbacks](#)
 - [Checklist](#)
- [Release Management](#)
 - [Versioning and change control](#)
 - [Pre-release checklist](#)
 - [Release process](#)
 - [Post-release](#)
 - [Rollback](#)
 - [Compliance/audit notes](#)
 - [Checklist](#)
- [DevOps Checklists](#)
 - [Pre-merge](#)
 - [Pre-release](#)
 - [Post-release](#)
 - [Infrastructure hygiene](#)
- [Publishing and Formats \(PDF/EPUB/Kindle\)](#)
 - [Source layout](#)
 - [PDF and EPUB \(pandoc\)](#)
 - [Kindle \(KDP\)](#)
 - [Optional upgrades](#)
 - [Link correctness](#)
 - [Image handling](#)
 - [QA before publish](#)
- [eBook Export Checklist](#)
- [Publishing Workflow \(End-to-End\)](#)
 - [Layout](#)
 - [Steps](#)
 - [Cover and front matter](#)
 - [Tooling notes](#)
 - [Automation](#)
 - [Clickable links](#)
 - [Checklist](#)
- [Mobile: Capacitor and Native](#)
 - [Project setup \(template\)](#)

- [Using plugins](#)
 - [Performance tips](#)
 - [Offline and sync](#)
 - [Native UX polish](#)
 - [Debugging](#)
 - [Try it now: camera + offline draft](#)
- [Edge and Serverless Targets](#)
 - [Choosing a target](#)
 - [Adapters overview](#)
 - [Server entry checklist](#)
 - [Caching and revalidation](#)
 - [Env and secrets](#)
 - [Deployment steps \(example: Vercel Edge\)](#)
 - [AWS/Lambda specifics](#)
 - [Bun/Deno edge specifics](#)
 - [Observability](#)
 - [Try it now: Cloudflare Pages with KV and ISR-like revalidate](#)
- [Desktop: Tauri and Native Shells](#)
 - [Why Tauri](#)
 - [Project setup](#)
 - [File access](#)
 - [Native shell features](#)
 - [Security](#)
 - [Performance](#)
 - [Testing](#)
 - [Checklist](#)
- [Progressive Web Apps \(PWA\)](#)
 - [Core setup](#)
 - [Service worker strategies](#)
 - [Installability](#)
 - [Offline UX](#)
 - [Updates](#)
 - [Testing PWAs](#)
 - [Security](#)
 - [Checklist](#)
- [WebGPU and High-Performance Rendering](#)
 - [When to use WebGPU](#)
 - [Setup](#)
 - [Basic pipeline \(sketch\)](#)
 - [Performance tips](#)
 - [SSR/Islands with WebGPU](#)
 - [Testing](#)
 - [Safety and compatibility](#)
- [WebAssembly \(WASM\)](#)
 - [When to reach for WASM](#)
 - [Build and load](#)
 - [Passing data](#)
 - [Threading and SIMD](#)
 - [SSR and islands](#)
 - [Testing](#)
 - [Security](#)
- [Native Bridges \(Android/iOS\)](#)
 - [When to use a native bridge](#)
 - [Options](#)
 - [Patterns](#)
 - [Security](#)
 - [Testing](#)
 - [Checklist](#)
- [Streaming and Progressive Rendering](#)
 - [When to stream](#)
 - [Server-side streaming](#)
 - [Client-side progressive data](#)
 - [Error and retry](#)
 - [Performance and SEO](#)
 - [Testing streaming](#)
 - [Try it now: streamed dashboard panel](#)

- [Offline and Resilient UX](#)
 - [Core principles](#)
 - [Data strategy](#)
 - [Persistence](#)
 - [UI patterns](#)
 - [Service workers/PWA](#)
 - [Testing](#)
 - [Checklist](#)
- [Accessibility Patterns](#)
 - [Principles](#)
 - [Forms](#)
 - [Dialogs and menus](#)
 - [Live regions](#)
 - [Motion and contrast](#)
 - [Testing](#)
 - [Checklist](#)
- [Internationalization \(i18n\) Patterns](#)
 - [Principles](#)
 - [Setup](#)
 - [Usage](#)
 - [Loading strategy](#)
 - [Plurals and formatting](#)
 - [Forms and validation](#)
 - [Testing](#)
 - [Performance tips](#)
 - [Checklist](#)
- [Forms UX Patterns](#)
 - [Clarity and flow](#)
 - [Validation UX](#)
 - [Performance and responsiveness](#)
 - [Accessibility](#)
 - [File uploads](#)
 - [Errors and recovery](#)
 - [Testing](#)
 - [Checklist](#)
- [Feature Flags and Experiments](#)
 - [Goals](#)
 - [Implementation](#)
 - [Data and cache impact](#)
 - [Observability](#)
 - [Testing](#)
 - [Checklist](#)
- [Analytics and Instrumentation](#)
 - [Principles](#)
 - [Events to capture](#)
 - [Implementation](#)
 - [Privacy and compliance](#)
 - [Testing](#)
 - [Checklist](#)
- [Architecture Decision Records \(ADRs\)](#)
 - [Why ADRs](#)
 - [Template \(suggested\)](#)
 - [What to record](#)
 - [Practices](#)
 - [Checklist](#)
- [CMS and Content Workflows](#)
 - [Choosing a CMS](#)
 - [Data flow](#)
 - [Modeling content](#)
 - [Images and media](#)
 - [Webhooks and revalidation](#)
 - [Localization](#)
 - [Testing](#)
 - [Checklist](#)
- [Static vs Live Content](#)

- Static content (docs, marketing)
- Semi-static content (blogs with comments, listings)
- Live content (dashboards, chat, analytics)
- Mixed routes
- Testing
- Checklist
- SEO and Discoverability
 - Core practices
 - Meta handling
 - Routing and links
 - Performance and crawl budget
 - Internationalization
 - Accessibility overlap
 - Testing
 - Checklist
- Charts, Graphics, and Diagrams
 - Architecture and routing
 - SSR and state
 - Observability and deployment
 - Performance charts
 - Appendix visuals
 - Notes
- Local-First and GenUI
 - Local-first foundation
 - Generative UI readiness
 - PhilJS alignment
 - Nexus checklist
- Nexus, GenUI, and AI-assisted Flows
 - Principles
 - Capturing intent
 - AI actions with guardrails
 - Collaboration and presence
 - Offline/rehydration
 - Testing and safety
 - Observability for AI flows
 - Try it now: guarded AI intent
- Collaboration Patterns (Nexus)
 - Presence
 - CRDTs and conflict resolution
 - Editing models
 - Sync channels
 - Offline + reconnect
 - Observability
 - Testing
 - Checklist
- Migrating from React
 - Strategy
 - JSX runtime
 - State and effects
 - Components and props
 - Routing
 - Context
 - Porting hooks
 - Styling and assets
 - Testing
 - SSR/Islands
 - Checklist
- Migrating from Solid/Svelte
 - Mapping concepts
 - JSX and templates
 - Routing
 - SSR and islands
 - Stores and context
 - Styling
 - Testing
 - Checklist

- Configuration
 - Route modules
- TypeScript 6.x Deep Dive (Appendix)
 - Table of contents
 - Widening vs narrowing
 - Type/value split
 - Strictness flags that matter
 - Literal unions for state
 - Indexed access and `keyof`
 - Conditional inference patterns
 - Module resolution and exports
 - Type-level programming patterns
 - Error modeling patterns
 - JSX namespace and intrinsic elements
 - DOM and event typing
 - Debugging type errors
 - Typed route params
 - Loader results as discriminated unions
 - Safe config validation
 - Schema-style validation
 - Branded IDs
 - Cache tags with template literals
 - Typed loaders and actions
 - Store selectors with `keyof`
 - Checklist: safer types
 - Exercises
 - Links
- Rust 1.92 Quickstart (Appendix)
 - Table of contents
 - Links
- Glossary
- Index (Quick Links)
 - Getting Started
 - Core and Rendering
 - Routing and Data
 - SSR and Islands
 - State and Styling
 - Testing and Performance
 - Best Practices
 - Deployment and DevOps
 - Security and Observability
 - Architecture
 - Platforms
 - Integrations
 - Patterns and Content
 - Packages
 - Appendices

Introduction

PhilJS is a TypeScript-first, signals-first framework built for the Nexus era: local-first data, zero-latency UX, and adaptive UI. This book is the canonical guide for PhilJS v0.1.0.

Who this book is for

- Teams shipping product UIs that must feel instant
- Library authors who want a stable, typed UI foundation
- Fullstack engineers who need SSR, islands, and streaming without runtime bloat

What you will learn

1. Tooling setup for Node 24+ and TypeScript 6.x
2. Core primitives: signals, effects, memos, resources
3. Routing with loaders/actions and offline-first patterns
4. SSR + islands with selective hydration
5. Testing, performance, and deployment
6. Nexus architecture principles

Conventions

- All code is TypeScript or TSX.
- Imports are scoped: `@philjs/core`, `@philjs/router`, `@philjs/ssr`.
- Versions are pinned to `^0.1.0`.
- Commands use `pnpm 9+`.

Canonical standards

PhilJS standards are derived from the current codebase and documentation. When you see a pattern here, treat it as the canonical PhilJS way.

Core ideas in one page

- **Signals-first:** fine-grained reactivity without diffing; memos/resources for derived/async data.
- **SSR + Islands + Streaming:** render on the server, hydrate only what matters, stream the rest.
- **Loaders/Actions:** data and mutations live at the route boundary, enabling cache tags, revalidate hints, and optimistic flows.
- **Adapters everywhere:** deploy to Edge (Vercel/Netlify/CF/Bun/Deno) or regional (Node/AWS) with the same primitives.
- **Nexus-ready:** local-first, collaborative, AI-assisted patterns built in.

How to read this book

- Start with Getting Started to set up toolchain and build a counter.
- Jump to Core and Rendering to understand signals, memos, and JSX ergonomics.
- Route/Data/Forms for real apps; SSR/Islands to go production.
- Testing/Performance/Observability to keep quality high.
- Integrations/Platforms/Patterns to plug into your backend and targets.
- Nexus/GenUI when building AI and collaborative experiences.

Compatibility and versions

- Node 24+ required; Node 25 supported.
- TypeScript 6.x; use `jsxImportSource: "@philjs/core"`.
- All examples use `@philjs/*@0.1.0`.
- pnpm is the package manager used throughout; adapt commands for npm/yarn if needed.

What “Nexus era” means

Nexus combines:

- Local-first state and offline durability.
- Edge-rendered, latency-sensitive UX.
- AI-assisted intent handling with guardrails.
- Collaborative presence and conflict resolution.

PhilJS packages (`@philjs/core`, `@philjs/router`, `@philjs/ssr`, `adapters`, `devtools`) are designed to serve these requirements together.

Next steps

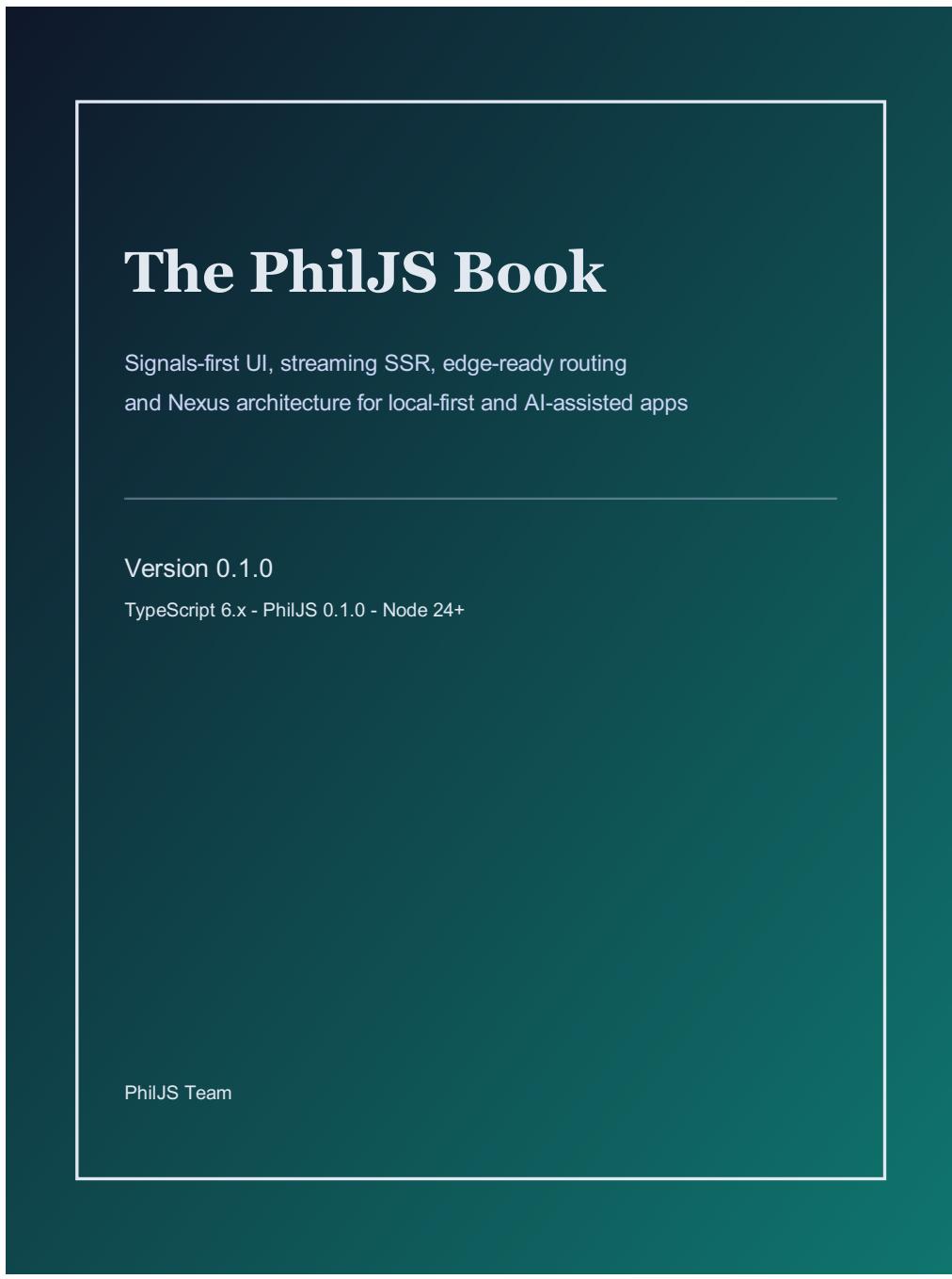
Get your toolchain ready, scaffold a project, and follow the Getting Started chapters. Each later chapter includes “Try it now” snippets so you can apply concepts immediately.

Front Matter

- Title: PhilJS Book
- Version: 0.1.0
- Authors: PhilJS Team
- License: MIT
- Stack: TypeScript 6.x, Node 24+, PhilJS 0.1.0
- Scope: UI, routing, SSR/Islands, performance, observability, security, DevOps, appendices (TS6, Rust 1.92)

This book is organized for clickable navigation in PDF/EPUB/Kindle. Use the Summary, Index, and Appendix links to jump to any topic.

PhilJS Book



PhilJS Book cover

Version: 0.1.0

Target: TypeScript 6.x, PhilJS 0.1.0, Node 24+

Signals-first UI, streaming SSR, edge-ready routing, and Nexus architecture for local-first and AI-assisted apps.

Authors: PhilJS Team

License: MIT

Repo: <https://github.com/philipjohnbasile/philjs>

Introduction to PhilJS

Welcome to PhilJS - a TypeScript-first framework that thinks ahead so you don't have to.

What is PhilJS?

PhilJS is a modern web framework that combines the best ideas from React, Solid, and Qwik while introducing groundbreaking concepts like **zero-hydration resumability**, **cost-**

aware rendering, and **AI-powered optimization**. Built from the ground up with performance and developer experience in mind, PhilJS makes it effortless to build lightning-fast web applications.

Unlike traditional frameworks that force you to choose between server rendering and client interactivity, PhilJS gives you both without compromise. Applications built with PhilJS load instantly, stay responsive, and cost less to run - automatically.

At its core, PhilJS uses **fine-grained reactivity** with signals, meaning your UI updates surgically when data changes. No virtual DOM diffing. No unnecessary re-renders. Just direct, efficient updates to exactly what changed.

Try It Yourself

Here's a simple PhilJS example you can edit and run right in your browser:

```
```typescript
const count = signal(0);
count.set(10);

// Simulate clicking the button 3 times
button.click();
button.click();
button.click();

console.log('Count:', count());
```

```

Try modifying the code above! Change the initial value, add more increments, or experiment with the signal API.

Why PhilJS?

1. Instant First Load with Resumability

Traditional frameworks hydrate your entire application on the client, re-executing all your component code just to attach event listeners. PhilJS uses **resumability** - your app picks up exactly where the server left off with zero wasted work.

What this means for you:

- **0ms hydration time** (vs 500-2000ms for typical React apps)
- **Interactive immediately** - no "loading" phase
- **Better user experience** - especially on slow devices/networks

Real-world impact:

React App: Server HTML → Download 200KB JS → Parse → Hydrate 500ms → Interactive PhilJS App: Server HTML → Interactive immediately

2. Fine-Grained Reactivity = Maximum Performance

PhilJS uses **signals** instead of virtual DOM. When your data changes, only the specific DOM nodes that depend on that data update - nothing else.

Example:

```
```typescript
// A counter that updates without re-rendering anything else
const count = signal(0);

<h1>My App</h1>
 <p>Count: {count()}</p> /* Only this text node updates */
 <button onClick={() => count.set(c => c + 1)}>+</button>


```

```

When you click the button, **only the text node** showing the count updates. The h1, the div, the button - all unchanged. This is true for apps of any size.

Performance benefits: - 10x faster updates than Virtual DOM frameworks - No diffing algorithm overhead - Predictable, consistent performance - Better battery life on mobile

3. Cost Tracking Built-In

PhilJS is the first framework with **built-in cost tracking**. See exactly how much your components cost to run in production - down to the penny.

```
import { useCosts } from '@philjs/core';

export function Dashboard() {
  const costs = useCosts({ component: 'Dashboard' });

  return (
    <div>
      <p>This component costs: ${costs.total}/month</p>
      <p>Renders: {costs.renders} @ {costs.renderCost}</p>
      <p>Data: {costs.dataTransfer} @ {costs.dataCost}</p>
    </div>
  );
}
```

Why this matters: - Identify expensive components before they hit production - Set performance budgets in dollars, not milliseconds - Optimize where it actually matters - Make business-driven technical decisions

4. TypeScript 6 First

PhilJS is written in TypeScript and designed for TypeScript 6+. Every API is fully typed with excellent inference and leverages the latest TypeScript features.

```

// Props are automatically inferred with TypeScript 6 features
function UserProfile({ user, onSave }: {
  user: { name: string; email: string };
  onSave: (user: User) => Promise<void>;
}) {
  const editing = signal(false);

  // TypeScript 6 knows editing is Signal<boolean>
  // FULL autocomplete and type checking everywhere
}

// Using TypeScript 6 satisfies operator for config
const config = {
  theme: 'dark',
  version: 2,
} as const satisfies AppConfig;

// Using NoInfer for better generic inference
function createStore<T>(initial: NoInfer<T>): Store<T> {
  return new Store(initial);
}

```

TypeScript 6 Benefits: - Isolated declarations for faster builds - Enhanced `satisfies` operator - `NoInfer<T>` utility type for better generics - Improved const type parameters - Better auto-imports and go-to-definition

5. Server Functions = Zero Boilerplate APIs

Call server code from your components like regular functions - PhilJS handles everything.

```

// server.ts
export const getUser = serverFn(async (id: number) => {
  // This runs ONLY on the server
  const user = await db.users.findById(id);
  return user;
});

// client.tsx
import { getUser } from './server';

export function UserProfile({ userId }: { userId: number }) {
  const user = await getUser(userId);

  return <div>{user.name}</div>;
}

```

No need for: - Separate API routes - REST endpoint definitions - GraphQL schemas - API client code - Manual serialization

Just call the function. PhilJS handles the network request, serialization, error handling, and type safety automatically.

6. Islands Architecture for Maximum Performance

Use `islands` to ship minimal JavaScript. Only interactive components get hydrated - the rest is pure HTML.

```

// This button is interactive (small JS bundle)
<Counter client:load />

// This content is static HTML (0 JS)
<BlogPost post={post} />

// This loads only when visible (Lazy Loaded)
<Comments client:visible />

```

Results: - 90% less JavaScript shipped - Faster page loads - Better Core Web Vitals - Lower bounce rates

7. Batteries Included

PhilJS comes with everything you need:

- `File-based routing` with layouts and nested routes
- `Data fetching` with caching and invalidation
- `Form handling` with validation and server actions
- `Authentication` patterns built-in
- `Real-time updates` via WebSockets/SSE
- `i18n` with locale routing
- `Testing utilities` for components and integration
- `DevTools` with time-travel debugging
- `Static generation` with ISR
- `Image optimization`
- `Code splitting` automatic and manual

No decision fatigue. No hunting for packages. Everything works together perfectly.

Who Should Use PhilJS?

Perfect For:

- Startups and MVPs** - Ship fast with minimal code - Scale without rewriting - Built-in cost tracking helps manage budgets - TypeScript catches bugs before users do
- E-commerce Sites** - Fast page loads = higher conversion - Resumability = instant interactivity - Islands = minimal JavaScript - Great Core Web Vitals = better SEO
- Content Sites and Blogs** - Static generation for speed - Dynamic data when needed - Excellent SEO out of the box - Great authoring experience
- Enterprise Applications** - TypeScript-first for large teams - Predictable performance at scale - Cost tracking for accountability - Excellent tooling and DevTools
- Mobile-First Applications** - Minimal JavaScript = better mobile experience - Fine-grained reactivity = smooth animations - Works great on slow networks - Battery-friendly

When You Need Extensions

- You Need Additional Ecosystem Features** - PhilJS is new and the ecosystem is expanding quickly - If a dependency is missing, build a PhilJS plugin or adapter - Use the compatibility layers during migration
- You're Building a Desktop App** - Use PhilJS Desktop or PhilJS Native packages - Share components between web and desktop targets
- You Need a JavaScript-Only Stack** - PhilJS is TypeScript-first and requires TypeScript 6+ - Keep everything in TypeScript for consistency and safety

How Is PhilJS Different?

vs React

- What's Better:** - □ 10x faster updates (signals vs virtual DOM) - □ Smaller bundle sizes (no runtime overhead) - □ Zero hydration (resumability vs hydration) - □ Built-in cost tracking - □ Server functions (vs REST/GraphQL)
- Trade-offs:** - □ Smaller ecosystem (newer framework) - □ Different mental model (signals vs hooks) - □□ Fewer third-party components
- PhilJS path for React-heavy stacks:** - Use the PhilJS compatibility layers while migrating - Wrap legacy components behind PhilJS routes - Replace hooks-based state with signals over time

vs Vue

- What's Better:** - □ Faster (fine-grained reactivity) - □ Resumability (vs hydration) - □ TypeScript-first (better inference) - □ Server functions built-in
- Trade-offs:** - □ Newer with less documentation - □ No template syntax (JSX only)
- PhilJS path for template-heavy teams:** - Keep the core UI in PhilJS and wrap for legacy apps - Use PhilJS components as the long-term source of truth - Expand the PhilJS ecosystem with targeted adapters

vs Svelte

- What's Better:** - □ Resumability (instant interactivity) - □ Cost tracking - □ Better streaming SSR - □□ Islands architecture built-in
- Trade-offs:** - □ No template syntax (JSX vs Svelte syntax) - □ Smaller community
- PhilJS path for Svelte-style DX:** - Use signals and islands for compiler-friendly UI - Keep SSR and streaming on by default for performance - Adopt PhilJS stores for predictable state

vs Next.js

- PhilJS includes everything Next.js does plus: - □ Fine-grained reactivity (faster updates) - □ Resumability (zero hydration) - □□ Islands architecture - □ Cost tracking
- PhilJS path for Next.js migrations:** - Use the PhilJS compatibility layer for React components - Move routing and data loading into PhilJS modules - Replace Next.js-specific APIs with PhilJS SSR/ISR

Quick Feature Overview

Requirements

- **Node.js 24+** (Node 25 supported) - Required for native ESM, Promise.withResolvers(), Object.groupBy()
- **TypeScript 6+** - Required for isolated declarations and enhanced inference

Reactivity

```
import { signal, memo, effect } from '@philjs/core';

const count = signal(0);           // Reactive state
const doubled = memo(() => count() * 2); // Computed value
effect(() => console.log(count())); // Side effect
```

Components

```
export function Welcome({ name }: { name: string }) {
  return <h1>Hello, {name}!</h1>;
}
```

Server Functions

```
const saveUser = serverFn(async (user: User) => {
  await db.users.save(user);
});
```

Routing

```
src/routes/
  index.tsx      → /
  about.tsx      → /about
  users/
    [id].tsx     → /users/123
```

Data Fetching

```
const user = createQuery(() => fetchUser(userId));
```

Forms

```
<form action={createUser}>
  <input name="name" required />
  <button>Create</button>
</form>
```

Getting Started

Ready to build something amazing? Let's go:

1. [Installation](#) - Get PhilJS installed in 30 seconds
2. [Quick Start](#) - Build your first app in 5 minutes
3. [Tutorial](#) - Learn by building a game

Community and Support

- [GitHub](#): github.com/philijs/philijs
- [Discord](#): discord.gg/philijs
- [Twitter](#): [@philijs](https://twitter.com/philijs)
- [Stack Overflow](#): Tag with `philijs`

License

PhilJS is MIT licensed. Free for personal and commercial use.

Next: [Installation](#) →

Installation

Get PhilJS up and running in less than a minute.

Prerequisites

Before installing PhilJS, make sure you have:

- [Node.js 24.0 or higher](#) (Node 25 supported) ([Download here](#))
- [npm](#), [pnpm](#), [yarn](#), or [bun](#) package manager
- [TypeScript 6.0 or higher](#)

Check your Node version:

```
node --version
# Should output v24.0.0 or higher
```

PhilJS requires Node.js 24+ to take advantage of: - Native ESM support with import attributes - Built-in `Promise.withResolvers()` - Native `Object.groupBy()` and `Map.groupBy()` - Improved performance with V8 optimizations - Native WebSocket client support

Tip: We recommend using `pnpm` for faster installs and better disk space usage.

Quick Start with `create-philijs`

The fastest way to get started is with `create-philijs`:

Using npm

```
npm create philjs@latest my-app
cd my-app
npm install
npm run dev
```

Using pnpm (recommended)

```
pnpm create philjs my-app
cd my-app
pnpm install
pnpm dev
```

Using yarn

```
yarn create philjs my-app
cd my-app
yarn install
yarn dev
```

Using bun

```
bun create philjs my-app  
cd my-app  
bun install  
bun dev
```

After running dev, open <http://localhost:3000> in your browser. You should see your new PhilJS app running!

Interactive Setup

When you run `create-philjs`, you'll be asked some questions:

- ✓ What is your project named? > my-app
- ✓ TypeScript (required) > Yes
- ✓ Would you like to use Tailwind CSS? > Yes
- ✓ Initialize a git repository? > Yes
- ✓ Install dependencies? > Yes

Options Explained

TypeScript (Required) - TypeScript is required for PhilJS projects - PhilJS is built with TypeScript and has excellent type inference - All templates are TypeScript-first

Tailwind CSS (Optional) - Utility-first CSS framework - Great for rapid development - Say "No" if you prefer other styling approaches

Git Repository (Recommended) - Initializes git for version control - Creates a `.gitignore` file - Ready for GitHub/GitLab

Install Dependencies (Recommended) - Automatically runs `npm install` or equivalent - Say "No" if you want to install manually later

Manual Installation

If you prefer to add PhilJS to an existing project:

1. Install PhilJS packages

```
npm install @philjs/core @philjs/router @philjs/ssr  
  
# Or with pnpm  
pnpm add @philjs/core @philjs/router @philjs/ssr  
  
# Or with yarn  
yarn add @philjs/core @philjs/router @philjs/ssr  
  
# Or with bun  
bun add @philjs/core @philjs/router @philjs/ssr
```

2. Install dev dependencies

```
npm install -D vite @vitejs/plugin-react typescript
```

3. Create tsconfig.json

```
{  
  "compilerOptions": {  
    "target": "ES2024",  
    "useDefineForClassFields": true,  
    "lib": ["ES2024", "DOM", "DOM.Iterable"],  
    "module": "NodeNext",  
    "skipLibCheck": true,  
    "moduleResolution": "NodeNext",  
    "allowImportingTsExtensions": true,  
    "resolveJsonModule": true,  
    "isolatedModules": true,  
    "isolatedDeclarations": true,  
    "noEmit": true,  
    "jsx": "preserve",  
    "jsxiImportSource": "@philjs/core",  
    "strict": true,  
    "noUnusedLocals": true,  
    "noUnusedParameters": true,  
    "noFallthroughCasesInSwitch": true,  
    "exactOptionalPropertyTypes": true,  
    "noUncheckedIndexedAccess": true,  
    "paths": {  
      "@/*": ["./src/*"]  
    },  
    "include": ["src"],  
    "references": [{ "path": "./tsconfig.node.json" }]  
  }  
}
```

4. Create tsconfig.node.json

```
{
  "compilerOptions": {
    "composite": true,
    "skipLibCheck": true,
    "module": "ESNext",
    "moduleResolution": "bundler",
    "allowSyntheticDefaultImports": true
  },
  "include": ["vite.config.ts"]
}
```

5. Create vite.config.ts

```
import { defineConfig } from 'vite';
import philjs from '@philjs/compiler/vite';

export default defineConfig({
  plugins: [philjs()],
});
```

6. Create package.json scripts

```
{
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  }
}
```

7. Create your first component

Create src/App.tsx:

```
import { signal } from '@philjs/core';

export function App() {
  const count = signal(0);

  return (
    <div>
      <h1>Welcome to PhilJS!</h1>
      <p>Count: {count()}</p>
      <button onClick={() => count.set(c => c + 1)}>
        Increment
      </button>
    </div>
  );
}
```

8. Create src/index.tsx

```
import { render } from '@philjs/core';
import { App } from './App';

render(<App />, document.getElementById('root')!);
```

9. Create index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>PhilJS App</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/index.tsx"></script>
  </body>
</html>
```

10. Start development server

```
npm run dev
```

Open <http://localhost:3000> and you should see your app!

Project Structure

After creating a new project, you'll see this structure:

```
my-app/
├── public/          # Static assets
|   └── favicon.ico
├── src/
|   ├── routes/      # File-based routing
|   |   └── index.tsx # Homepage (/)
|   ├── components/ # Reusable components
|   ├── styles/     # Global styles
|   |   └── global.css
|   ├── App.tsx      # Root component
|   └── index.tsx    # Entry point
└── .gitignore
├── package.json
├── tsconfig.json
└── vite.config.ts
└── README.md
```

Key Files

src/index.tsx - Application entry point

```
import { render } from '@philjs/core';
import { App } from './App';

render(<App />, document.getElementById('root')!);
```

src/App.tsx - Root component

```
import { Router } from '@philjs/router';

export function App() {
  return (
    <Router>
      {/* Your routes go here */}
    </Router>
  );
}
```

src/routes/index.tsx - Homepage

```
export default function Home() {
  return <h1>Welcome to PhilJS!</h1>;
}
```

vite.config.ts - Build configuration

```
import { defineConfig } from 'vite';
import philjs from '@philjs/compiler/vite';

export default defineConfig({
  plugins: [philjs()],
});
```

Editor Setup

VS Code (Recommended)

Install these extensions for the best experience:

1. [TypeScript PhilJS Plugin](#)
 - o JSX autocomplete
 - o Signal type inference
 - o Component prop validation
2. [Tailwind CSS IntelliSense](#)
 - o If you're using Tailwind CSS
3. [ESLint](#)
 - o Linting support
4. [Prettier](#)
 - o Code formatting

VS Code Settings

Add to your `.vscode/settings.json`:

```
{
  "typescript.tsdk": "node_modules/typescript/lib",
  "typescript.enablePromptUseWorkspaceTsdk": true,
  "editor.formatOnSave": true,
  "editor.defaultFormatter": "esbenp.prettier-vscode",
  "[typescript]": {
    "editor.defaultFormatter": "esbenp.prettier-vscode"
  },
  "[typescriptreact)": {
    "editor.defaultFormatter": "esbenp.prettier-vscode"
  }
}
```

WebStorm / IntelliJ IDEA

PhilJS works great with WebStorm out of the box. For the best experience:

1. Enable TypeScript service
2. Set JSX to "React JSX"
3. Install PhilJS plugin from JetBrains Marketplace

Other Editors

PhilJS uses standard TypeScript and JSX, so any editor with TypeScript support will work: - **Vim/Neovim**: Use CoC or native LSP - **Emacs**: Use tide or lsp-mode - **Sublime Text**: Use LSP-TypeScript

Troubleshooting

Port 3000 already in use

If you see "Port 3000 is already in use":

```
# Kill the process using port 3000 (macOS/Linux)
lsof -ti:3000 | xargs kill -9

# Or use a different port
npm run dev -- --port 3001
```

Module not found errors

Make sure you installed all dependencies:

```
rm -rf node_modules package-lock.json
npm install
```

For pnpm:

```
rm -rf node_modules pnpm-lock.yaml
pnpm install
```

TypeScript errors in JSX

Make sure your tsconfig.json has:

```
{
  "compilerOptions": {
    "jsx": "preserve",
    "jsxImportSource": "@philjs/core"
  }
}
```

Vite not found

Install Vite as a dev dependency:

```
npm install -D vite
```

"Cannot find module '@philjs/core'"

Make sure you've installed the core package:

```
npm install @philjs/core
```

Windows-specific issues

Line ending problems:

```
git config core.autocrlf false
```

Path length issues: Enable long paths:

```
git config --system core.longpaths true
```

Next Steps

Now that PhilJS is installed, you're ready to start building:

1. [Quick Start](#) - Build your first app in 5 minutes
2. [Your First Component](#) - Learn component basics
3. [Tutorial: Tic-Tac-Toe](#) - Build a game and learn PhilJS concepts

Version Information

Current Version: 0.1.0

Minimum Requirements: - Node.js 24.0+ - TypeScript 6.0+ - Modern browser with ES2024 support

Supported Browsers: - Chrome 120+ - Firefox 121+ - Safari 17.4+ - Edge 120+

TypeScript 6 Features Used: - Isolated declarations for faster builds - Enhanced type inference - Improved `satisfies` operator - Better const type parameters - `NoInfer<T>` utility type

Upgrade Guide

To upgrade to the latest version:

```
npm install @philjs/core@latest @philjs/router@latest @philjs/ssr@latest  
# Or with pnpm  
pnpm update @philjs/core @philjs/router @philjs/ssr
```

Check the [changelog](#) for breaking changes.

Next: [Quick Start →](#)

Quick Start

Build your first PhilJS application in 5 minutes. By the end of this guide, you'll have a working counter app that demonstrates PhilJS's core concepts.

What You'll Learn

- Creating a new PhilJS project
- Building your first component
- Using signals for reactive state
- Handling user interactions
- Running your app in the browser

Step 1: Create a New Project

Open your terminal and run:

```
pnpm create philjs my-counter-app  
cd my-counter-app
```

When prompted, choose these options: - **TypeScript (required)** Yes - **Tailwind CSS?** No (we'll use simple CSS) - **Git repository?** Yes - **Install dependencies?** Yes

This creates a new PhilJS project with the following structure:

```
my-counter-app/  
|__ src/  
| |__ routes/  
| | |__ index.tsx # Homepage  
| |__ App.tsx # Root component  
| |__ index.tsx # Entry point  
|__ public/  
|__ package.json  
|__ tsconfig.json  
└__ vite.config.ts
```

Step 2: Create Your First Component

Open `src/routes/index.tsx` and replace its contents with:

```

import { signal } from '@philjs/core';

export default function Home() {
  // Create a reactive signal for our count
  const count = signal(0);

  return (
    <div style={{ padding: '2rem', fontFamily: 'sans-serif' }}>
      <h1>My First PhilJS App</h1>
      <p>You've clicked {count()} times</p>
      <button onClick={() => count.set(c => c + 1)}>
        Click me!
      </button>
    </div>
  );
}

```

Let's break down what's happening here:

Creating State with Signals

```
const count = signal(0);
```

This creates a **signal** - PhilJS's reactive state primitive. Signals are like boxes that hold values and notify subscribers when the value changes.

- `signal(0)` creates a signal with initial value of 0
- Signals are reactive - when they change, the UI automatically updates

Reading Signal Values

```
<p>You've clicked {count()} times</p>
```

To read a signal's value, call it like a function: `count()`.

PhilJS tracks where you read the signal and automatically updates just that part of the DOM when the value changes. This is called **fine-grained reactivity**.

Updating Signal Values

```
onClick={() => count.set(c => c + 1)}
```

To update a signal, use `.set()`: - `count.set(5)` - set to a specific value - `count.set(c => c + 1)` - update based on current value (safer for concurrent updates)

When you click the button, the signal updates and the `<p>` tag automatically shows the new count. Nothing else re-renders!

Step 3: Add Some Style

Let's make it look better. Update your component:

```

import { signal } from '@philjs/core';

export default function Home() {
  const count = signal(0);

  return (
    <div style={styles.container}>
      <div style={styles.card}>
        <h1 style={styles.title}>My First PhilJS App</h1>

        <div style={styles.counter}>
          <button
            style={styles.button}
            onClick={() => count.set(c => c - 1)}
          >
            -
          </button>

          <span style={styles.count}>
            {count()}
          </span>
        </div>

        <button
          style={styles.button}
          onClick={() => count.set(c => c + 1)}
        >
          +
        </button>
      </div>

      <p style={styles.text}>
        {count() === 0 && "Start counting!"}
        {count() > 0 && count() < 10 && "Keep going!"}
        {count() >= 10 && "You're on fire! 🚨"}
      </p>
    </div>
  );
}

```

```

const styles = {
  container: {
    display: 'flex',
    justifyContent: 'center',
    alignItems: 'center',
    minHeight: '100vh',
    background: 'linear-gradient(135deg, #667eea 0%, #764ba2 100%)',
  },
  card: {
    background: 'white',
    borderRadius: '20px',
    padding: '3rem',
    boxShadow: '0 20px 60px rgba(0,0,0,0.3)',
    textAlign: 'center' as const,
  },
  title: {
    fontSize: '2rem',
    marginBottom: '2rem',
    color: '#333',
  },
  counter: {
    display: 'flex',
    alignItems: 'center',
    justifyContent: 'center',
    gap: '2rem',
    marginBottom: '2rem',
  },
  button: {
    width: '60px',
    height: '60px',
    fontSize: '2rem',
    border: 'none',
    borderRadius: '50%',
    background: '#667eea',
    color: 'white',
    cursor: 'pointer',
    transition: 'transform 0.1s, background 0.2s',
  },
  count: {
    fontSize: '3rem',
    fontWeight: 'bold' as const,
    color: '#667eea',
    minWidth: '100px',
  },
  text: {
    fontSize: '1.2rem',
    color: '#666',
    minHeight: '1.5em',
  },
};

```

What's New?

Multiple Buttons

```

<button onClick={() => count.set(c => c - 1)}>-</button>
<button onClick={() => count.set(c => c + 1)}>+</button>

```

Now you can increment and decrement the counter.

Conditional Rendering

```

{count() === 0 && "Start counting!"}
{count() > 0 && count() < 10 && "Keep going!"}
{count() >= 10 && "You're on fire! 🚨"}

```

This shows different messages based on the count value. The `&&` operator in JSX means "if the left side is true, show the right side."

Note: Only the part that changes updates in the DOM. When you click `+`, only the count number and message update - the buttons, title, and container stay unchanged. This is fine-grained reactivity in action!

Step 4: Run Your App

Start the development server:

```
pnpm dev
```

Open <http://localhost:3000> in your browser.

You should see your counter app! Try clicking the buttons - notice how smooth and instant the updates are.

Step 5: Add More Interactivity

Let's add a reset button and track total clicks:

```

import { signal, memo } from '@philjs/core';

export default function Counter() {
  const [c, set] = signal([0, () => set(c => c + 1)]);

```

```

export default function Counter() {
  const count = signal(0);
  const totalClicks = signal(0);

  // Computed value that updates automatically
  const isEven = memo(() => count() % 2 === 0);

  const increment = () => {
    count.set(c => c + 1);
    totalClicks.set(t => t + 1);
  };

  const decrement = () => {
    count.set(c => c - 1);
    totalClicks.set(t => t + 1);
  };

  const reset = () => {
    count.set(0);
  };

  return (
    <div style={styles.container}>
      <div style={styles.card}>
        <h1 style={styles.title}>Counter App</h1>

        <div style={styles.stats}>
          <div>
            Count: <strong>{count()}</strong>
          </div>
          <div>
            Total clicks: <strong>{totalClicks()}</strong>
          </div>
          <div>
            Status: <strong>{isEven() ? 'Even' : 'Odd'}</strong>
          </div>
        </div>

        <div style={styles.counter}>
          <button style={styles.button} onClick={decrement}>
            -
          </button>

          <span style={styles.count}>{count()}</span>

          <button style={styles.button} onClick={increment}>
            +
          </button>
        </div>

        <button style={styles.resetButton} onClick={reset}>
          Reset
        </button>
      </div>
    </div>
  );
}

const styles = {
  container: {
    display: 'flex',
    justifyContent: 'center',
    alignItems: 'center',
    minHeight: '100vh',
    background: 'linear-gradient(135deg, #667eea 0%, #764ba2 100%)',
  },
  card: {
    background: 'white',
    borderRadius: '20px',
    padding: '3rem',
    boxShadow: '0 20px 60px rgba(0,0,0,0.3)',
    textAlign: 'center' as const,
    minWidth: '400px',
  },
  title: {
    fontSize: '2rem',
    marginBottom: '2rem',
    color: '#333',
  },
  stats: {
    display: 'flex',
    justifyContent: 'space-around',
    marginBottom: '2rem',
    padding: '1rem',
    background: '#f5f5f5',
    borderRadius: '10px',
    fontSize: '0.9rem',
    color: '#666',
  },
};

```

```

    counter: {
      display: 'flex',
      alignItems: 'center',
      justifyContent: 'center',
      gap: '2rem',
      marginBottom: '2rem',
    },
    button: {
      width: '60px',
      height: '60px',
      fontSize: '2rem',
      border: 'none',
      borderRadius: '50%',
      background: '#667eea',
      color: 'white',
      cursor: 'pointer',
      transition: 'all 0.2s',
    },
    count: {
      fontSize: '3rem',
      fontWeight: 'bold' as const,
      color: '#667eea',
      minWidth: '100px',
    },
    resetButton: {
      padding: '0.75rem 2rem',
      fontSize: '1rem',
      border: 'none',
      borderRadius: '10px',
      background: '#e0e0e0',
      color: '#666',
      cursor: 'pointer',
      transition: 'all 0.2s',
    },
  },
);

```

What's New?

Computed Values with memo()

```
const isEven = memo(() => count() % 2 === 0);
```

`memo()` creates a **computed value** that automatically updates when its dependencies change. Here, `isEven` recalculates whenever `count` changes.

Memos are cached - they only recalculate when their dependencies change, making them efficient for expensive computations.

Multiple State Values

```
const count = signal(0);
const totalClicks = signal(0);
```

You can have as many signals as you need. Each tracks its own value independently.

Event Handlers

```
const increment = () => {
  count.set(c => c + 1);
  totalClicks.set(t => t + 1);
};
```

You can update multiple signals in one handler. PhilJS batches the updates and only re-renders once.

Understanding What You Built

Congratulations! You've built a fully reactive counter app with PhilJS. Here's what makes it special:

1. Fine-Grained Reactivity

When you click a button, PhilJS doesn't re-render the entire component. It only updates the specific text nodes that show `count()` and `totalClicks()`.

Try this: Open DevTools → Elements and watch the DOM as you click. You'll see only the numbers flash, not the buttons or container.

2. Minimal Code

You wrote about 80 lines of code (including styles) to create a fully functional app. No boilerplate, no setup, just your logic.

3. TypeScript Safety

Even though you didn't write type annotations, TypeScript knows: - `count` is a `Signal<number>` - `isEven` is a `Memo<boolean>` - Your event handlers have the correct signatures

Try calling `count.set("hello")` - TypeScript will error because `count` expects a number.

4. Performance

This app is incredibly fast: - Initial bundle: ~12KB (gzipped) - Updates: < 1ms - Memory: Minimal (just two numbers) - No virtual DOM diffing - No unnecessary re-renders

Next Steps

Now that you've built your first app, you're ready to learn more:

Continue Learning

1. [Components](#) - Learn about component composition and props
2. [Signals Deep Dive](#) - Master reactive state
3. [Tutorial: Tic-Tac-Toe](#) - Build a game and learn advanced concepts

Build Something Real

Try building: - [Todo List](#) - Add items, mark complete, filter (practice lists and state) - [Weather App](#) - Fetch data from an API (practice data fetching) - [Blog](#) - Multiple pages with routing (practice routing and SSG)

Explore Features

- [Routing](#) - File-based routing with layouts
- [Data Fetching](#) - Queries, mutations, caching
- [Forms](#) - Form handling and validation
- [Server Functions](#) - Call server code like functions

Common Questions

Q: Why call signals like functions: count() instead of count?

A: Calling signals as functions lets PhilJS track where they're used. When you write `count()` in JSX, PhilJS subscribes that specific DOM node to updates. This enables fine-grained reactivity.

Q: Can I use classes instead of functions?

A: PhilJS uses function components only. They're simpler, more performant, and work better with TypeScript inference.

Q: Do I need to call .set() or can I mutate directly?

A: Always use `.set()`. Signals are immutable from the outside - the only way to change them is through `.set()`. This makes updates predictable and trackable.

Q: How do I use this in a real app?

A: This same pattern scales to apps of any size. You might organize signals into stores, use context for global state, and break components into smaller pieces, but the fundamentals are the same.

Q: What about async data?

A: Use `createResource()` for async data fetching. We'll cover this in the data fetching guide.

Troubleshooting

Nothing shows up

Check the browser console for errors. Common issues: - Missing `export default` on your component - Syntax error in JSX - Signal read without calling it: `{count}` instead of `{count()}`

Updates don't work

Make sure you're calling `.set()` to update signals:

```
// ⚡ Wrong
count.value = 5;

// ✅ Correct
count.set(5);
```

TypeScript errors

Make sure your `tsconfig.json` has (TypeScript 6+):

```
{
  "compilerOptions": {
    "target": "ES2024",
    "lib": ["ES2024", "DOM", "DOM.Iterable"],
    "module": "NodeNext",
    "moduleResolution": "NodeNext",
    "jsx": "preserve",
    "jsxiImportSource": "@philjs/core",
    "isolatedDeclarations": true,
    "exactOptionalPropertyTypes": true
  }
}
```

Summary

You've learned:

▢ Creating a PhilJS project ▢ Building components with JSX ▢ Managing state with signals ▢ Creating computed values with memo ▢ Handling events ▢ Conditional rendering ▢ Styling components

With these fundamentals, you're ready to build real applications with PhilJS!

Next: [Your First Component](#) →

Thinking in PhilJS

PhilJS requires a different mental model than React, Vue, or other popular frameworks. Understanding these differences will help you write better code and avoid common pitfalls.

The Mental Shift from React

If you're coming from React, the biggest mindset change is moving from **coarse-grained** to **fine-grained** reactivity.

React's Mental Model

In React, when state changes:

```
// React
const [count, setCount] = useState(0);

// Entire component re-runs when count changes
function Counter() {
  console.log('Component rendering'); // Logs on every update
  return <div>{count}</div>;
}
```

Key concept The whole component function runs again. React diffs the virtual DOM to determine what changed.

PhilJS's Mental Model

In PhilJS, when state changes, **only the specific parts using that state update**:

```
// PhilJS
const count = signal(0);

function Counter() {
  console.log('Component setup'); // Only logs once!
  return <div>{count()}</div>; // Only this updates
}
```

Key concept Components run once. Signals track their dependencies and update only the exact DOM nodes that need to change.

Understanding Fine-Grained Reactivity

Fine-grained reactivity means the framework knows **exactly** which parts of your UI depend on which pieces of state.

Automatic Dependency Tracking

```
const firstName = signal('John');
const lastName = signal('Doe');

// This memo only recalculates when firstName or LastName change
const fullName = memo(() => `${firstName()} ${lastName()}`);

// This effect only runs when fullName changes
effect(() => {
  console.log('Name changed:', fullName());
});
```

What happens: 1. When you call `firstName()` inside the memo, PhilJS records that dependency 2. When `firstName.set()` is called, only `fullName` recalculates 3. Only the effect subscribed to `fullName` runs 4. Nothing else in your app is touched

Surgical Updates

```
function UserProfile() {
  const user = signal({ name: 'Alice', age: 30, bio: 'Developer' });

  return (
    <div>
      <h1>{user().name}</h1> {/* Updates only when name changes */}
      <p>Age: {user().age}</p> {/* Updates only when age changes */}
      <p>{user().bio}</p> {/* Updates only when bio changes */}
    </div>
  );
}
```

In React, changing any property would re-render the entire component. In PhilJS, each expression `{user().name}` creates its own subscription, so only that specific text node updates.

When to Use Signals vs Props vs Context

This is one of the most common questions. Here's a decision tree:

Use Props When...

Data flows down the component tree naturally

```
// ⚡ Good - data flows parent → child
function UserList({ users }) {
  return (
    <div>
      {users.map(user => (
        <UserCard key={user.id} user={user} />
      ))}
    </div>
  );
}
```

Benefits: - Explicit data flow - Easy to trace where data comes from - TypeScript enforces the contract - No hidden dependencies

Use Signals When...

You need reactive state within a component

```
// ⚡ Good - Local component state
function SearchBox() {
  const query = signal('');
  const isSearching = signal(false);

  return (
    <input
      value={query()}
      onInput={(e) => query.set(e.target.value)}
      placeholder={isSearching() ? 'Searching...' : 'Search'}
    />
  );
}
```

You need to share state between a few related components

```
// ⚡ Good - shared state in parent, passed down
function ShoppingCart() {
  const items = signal<Item>[]>([]);

  return (
    <div>
      <ItemList items={items} />
      <CartSummary items={items} />
      <Checkout items={items} />
    </div>
  );
}
```

Benefits: - Reactive updates - Simple to create and use - No boilerplate - Can be passed as props

Use Context When...

Data needs to be accessible deep in the tree

```
// ⚡ Good - avoid prop drilling
const ThemeContext = createContext<Theme>();

function App() {
  const theme = signal<Theme>({ mode: 'light', primary: '#667eea' });

  return (
    <ThemeContext.Provider value={theme}>
      <Layout>
        <Header />
        <Sidebar />
        <Content />
      </Layout>
    </ThemeContext.Provider>
  );
}

// Many Levels deep...
function Button() {
  const theme = useContext(ThemeContext);
  return <button style={{ background: theme().primary }}>Click me</button>;
}
```

Benefits: - Avoids prop drilling - Clean component APIs - Easy to access from anywhere in the subtree

⚠️ Don't overuse: Context is great for themes, auth, i18n, but don't make everything global. Local state is easier to reason about.

Common Mental Model Pitfalls

Pitfall 1: Forgetting to Call Signals

```
const count = signal(0);

// ⚡ Wrong - reading a signal
<div>{count}</div> // Shows: [object Object]

// ⚡ Correct - calling the signal
<div>{count()}</div> // Shows: 0
```

Why: Signals are functions. `count` is the signal object, `count()` reads its value.

Pitfall 2: Using Signals Like React State

```
// ⚠ Wrong - treating signals like React
const count = signal(0);

function increment() {
  const current = count();
  count.set(current + 1); // Verbose!
}

// ⚠ Better - use the updater function
function increment() {
  count.set(c => c + 1);
}
```

Why: Signals have an updater function like React's `setState`, use it!

Pitfall 3: Not Using Memos for Derived State

```
const todos = signal<Todo[]>([...]);

// ⚠ Wrong - recalculates every time it's read
function getCompletedCount() {
  return todos().filter(t => t.completed).length;
}

// ⚠ Correct - memoized, only recalculates when todos change
const completedCount = memo(() =>
  todos().filter(t => t.completed).length
);
```

Why: Memos cache their result and only recalculate when dependencies change. Functions run every time.

Pitfall 4: Mutating Signal Values

```
const user = signal({ name: 'Alice', age: 30 });

// ⚠ Wrong - mutating the object
user().age = 31; // Doesn't trigger updates!

// ⚠ Correct - creating new object
user.set({ ...user(), age: 31 });

// ⚠ Also correct - using updater
user.set(u => ({ ...u, age: 31 }));
```

Why: Signals only detect changes when you call `.set()`. Direct mutation doesn't trigger updates.

Pitfall 5: Creating Signals Inside Render

```
// ⚠ Wrong - creates new signal on every render
function Counter() {
  const count = signal(0); // New signal every time!
  return <div>{count()}</div>;
}

// ⚠ Correct - signal persists across renders
const count = signal(0);

function Counter() {
  return <div>{count()}</div>;
}
```

Why: In PhilJS, components only run once during initial render. But if you're composing components, you want signals to be stable.

Exception: It's fine if the component truly only mounts once, but generally keep signals outside or use them for local ephemeral state.

Component Composition Patterns

Pattern 1: Container/Presenter

Separate logic from presentation:

```

// Container - handles Logic and state
function UserDashboardContainer() {
  const user = signal(null);
  const loading = signal(true);

  effect(() => {
    fetchUser().then(data => {
      user.set(data);
      loading.set(false);
    });
  });

  return <UserDashboard user={user} loading={loading} />;
}

// Presenter - pure rendering
function UserDashboard({ user, loading }) {
  if (loading()) return <Spinner />;
  if (!user()) return <Error />

  return (
    <div>
      <h1>{user().name}</h1>
      <p>{user().email}</p>
    </div>
  );
}

```

Pattern 2: Compound Components

Components that work together:

```

function Tabs({ children }) {
  const activeTab = signal(0);

  return (
    <div>
      <TabContext.Provider value={{ activeTab }}>
        {children}
      </TabContext.Provider>
    </div>
  );
}

function TabList({ children }) {
  return <div role="tablist">{children}</div>;
}

function Tab({ index, children }) {
  const { activeTab } = useContext(TabContext);
  return (
    <button
      onClick={() => activeTab.set(index)}
      aria-selected={activeTab() === index}
    >
      {children}
    </button>
  );
}

// Usage:
<Tabs>
  <TabList>
    <Tab index={0}>Profile</Tab>
    <Tab index={1}>Settings</Tab>
  </TabList>
  <TabPanel index={0}>Profile content</TabPanel>
  <TabPanel index={1}>Settings content</TabPanel>
</Tabs>

```

Pattern 3: Render Props (via Children)

```

function DataFetcher({ url, children }) {
  const data = signal(null);
  const loading = signal(true);
  const error = signal(null);

  effect(() => {
    fetch(url)
      .then(res => res.json())
      .then(data => {
        data.set(data);
        loading.set(false);
      })
      .catch(err => error.set(err));
  });

  return children({ data, loading, error });
}

// Usage:
<DataFetcher url="/api/users">
  {({ data, loading, error }) => (
    loading() ? <Spinner /> :
    error() ? <Error error={error()} /> :
    <UserList users={data()} />
  )}
</DataFetcher>

```

Pattern 4: Higher-Order Components (HOCs)

```

function withAuth<P>(Component: (props: P) => JSX.Element) {
  return (props: P) => {
    const user = useUser();

    if (!user()) {
      return <Redirect to="/login" />;
    }

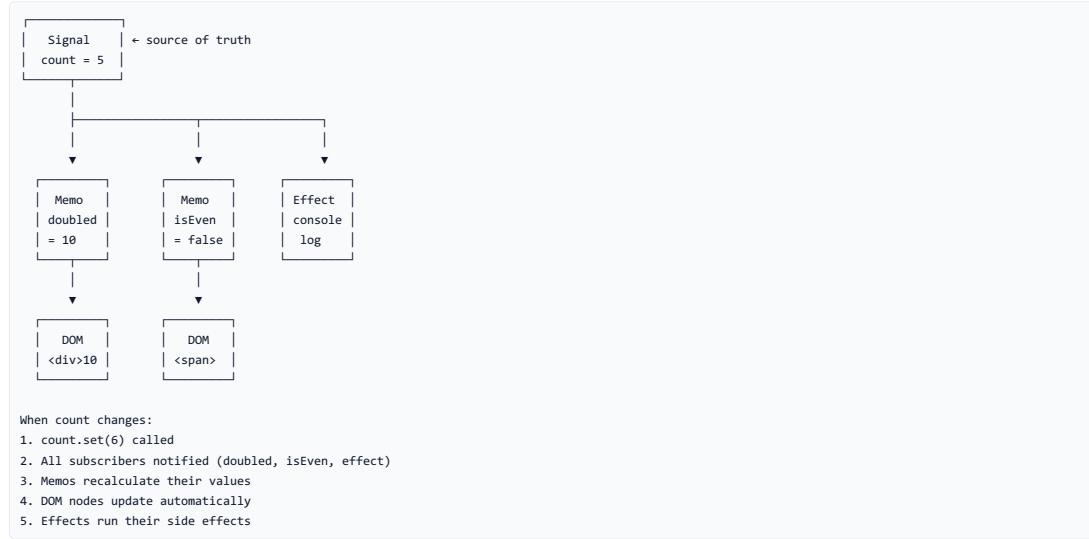
    return <Component {...props} user={user} />;
  }
}

// Usage:
const ProtectedDashboard = withAuth(Dashboard);

```

Reactivity Mental Model Diagram

Here's how to visualize PhilJS's reactivity:



Component Lifecycle Mental Model

Unlike React, PhilJS components don't have lifecycle methods. Instead:

React:	PhilJS:
Mount	Component function runs once
↓	↓
Render	Creates JSX with signals
↓	↓
Effect (useEffect)	Effects created, run once
↓	↓
Update (state change)	Signal changes
↓	↓
Re-render	Only subscribed nodes update
↓	↓
Effect cleanup	Effect cleanup (if needed)
↓	↓
Unmount	Effect cleanup runs

Key insight: PhilJS components run once, then signals handle all updates. No re-renders!

Best Practices Summary

□ Do This

- Call signals to read values: `count()`
- Use memos for derived state
- Keep signals immutable (create new objects)
- Use updater functions: `count.set(c => c + 1)`
- Pass signals as props when needed
- Use context for deep tree data
- Keep components small and focused

□ Avoid This

- Forgetting to call signals: `{count}` instead of `{count()}`
- Mutating signal values directly
- Creating signals inside frequently-called functions
- Overusing context (prefer props)
- Treating PhilJS like React
- Manually tracking dependencies (let PhilJS do it)

Transitioning Your Mindset

From React Hooks to PhilJS

```
// React
const [count, setCount] = useState(0);
const doubled = useMemo(() => count * 2, [count]);
useEffect(() => {
  console.log(count);
}, [count]);

// PhilJS
const count = signal(0);
const doubled = memo(() => count() * 2);
effect(() => {
  console.log(count());
});
```

Key differences: - No dependency arrays (automatic tracking) - No hook rules (use anywhere) - Signals persist outside components - No re-renders

From Vue to PhilJS

```
// Vue (Composition API)
const count = ref(0);
const doubled = computed(() => count.value * 2);
watch(count, (newValue) => {
  console.log(newValue);
});

// PhilJS
const count = signal(0);
const doubled = memo(() => count() * 2);
effect(() => {
  console.log(count());
});
```

Key differences: - Call signals as functions: `count()` vs `count.value` - Similar mental model otherwise - PhilJS has better TypeScript inference

Next Steps

Now that you understand the PhilJS mental model:

1. [Core Concepts: Signals](#) - Deep dive into signals
2. [Core Concepts: Memos](#) - Master derived state
3. [Core Concepts: Effects](#) - Handle side effects
4. [Best Practices](#) - Advanced patterns

Summary

The PhilJS mental model centers on:

- **Fine-grained reactivity** - Updates are surgical, not wholesale
- **Automatic dependency tracking** - No manual dependency arrays
- **Components run once** - No re-renders, only signal updates
- **Signals are functions** - Call them to read: `count()`
- **Immutable updates** - Create new values, don't mutate
- **Memos for derived state** - Cache expensive computations
- **Effects for side effects** - Run when dependencies change

Once this mental model clicks, PhilJS feels natural and powerful. You'll wonder how you ever tolerated re-rendering entire component trees!

Congratulations! You've completed the Getting Started guide. You now have the foundation to build amazing applications with PhilJS.

Next: [Core Concepts: Components](#) →

Your First Component

Components are the building blocks of PhilJS applications. Let's learn how to create and use them effectively.

What is a Component?

A component is a reusable piece of UI that combines markup (JSX), logic (TypeScript), and optionally state (signals). Think of components as custom HTML elements that you define.

Simple example:

```
export function Welcome() {
  return <h1>Hello, PhilJS!</h1>;
}
```

This is a valid PhilJS component. It's a function that returns JSX - that's it!

Creating Your First Component

Let's build a user profile card step by step.

Step 1: Basic Component Structure

Create a new file `src/components/UserCard.tsx`:

```
export function UserCard() {
  return (
    <div>
      <h2>John Doe</h2>
      <p>Software Engineer</p>
    </div>
  );
}
```

Key points: - Components are just functions - Function names should be PascalCase (`UserCard`, not `userCard`) - Components must return JSX - Export your component so others can use it

Step 2: Adding Props

Props make components reusable by passing data in:

```
interface UserCardProps {
  name: string;
  role: string;
}

export function UserCard({ name, role }: UserCardProps) {
  return (
    <div>
      <h2>{name}</h2>
      <p>{role}</p>
    </div>
  );
}
```

Now you can use it with different data:

```
<UserCard name="John Doe" role="Software Engineer" />
<UserCard name="Jane Smith" role="Product Manager" />
```

□ **Tip:** Always type your props with TypeScript. It prevents bugs and enables autocomplete.

Step 3: Adding Styles

Let's make it look good:

```

interface UserCardProps {
  name: string;
  role: string;
  avatar?: string;
}

export function UserCard({ name, role, avatar }: UserCardProps) {
  return (
    <div style={styles.card}>
      {avatar && (
        <img
          src={avatar}
          alt={name}
          style={styles.avatar}
        />
      )}
      <div style={styles.info}>
        <h2 style={styles.name}>{name}</h2>
        <p style={styles.role}>{role}</p>
      </div>
    </div>
  );
}

const styles = {
  card: {
    display: 'flex',
    alignItems: 'center',
    padding: '1.5rem',
    background: 'white',
    borderRadius: '8px',
    boxShadow: '0 2px 8px rgba(0,0,0,0.1)',
    gap: '1rem',
  },
  avatar: {
    width: '64px',
    height: '64px',
    borderRadius: '50%',
    objectFit: 'cover' as const,
  },
  info: {
    flex: 1,
  },
  name: {
    margin: 0,
    fontSize: '1.25rem',
    color: '#333',
  },
  role: {
    margin: '0.25rem 0 0',
    color: '#666',
    fontSize: '0.9rem',
  },
};

```

Usage:

```

<UserCard
  name="John Doe"
  role="Software Engineer"
  avatar="/avatars/john.jpg"
/>

```

Step 4: Adding Interactivity with Signals

Let's add a follow button:

```

import { signal } from '@philjs/core';

interface UserCardProps {
  name: string;
  role: string;
  avatar?: string;
  initialFollowing?: boolean;
}

export function UserCard({
  name,
  role,
  avatar,
  initialFollowing = false
}: UserCardProps) {
  const following = signal(initialFollowing);

  const toggleFollow = () => {
    following.set(!following());
  };

  return (
    <div style={styles.card}>
      {avatar && (
        <img src={avatar} alt={name} style={styles.avatar} />
      )}
      <div style={styles.info}>
        <h2 style={styles.name}>{name}</h2>
        <p style={styles.role}>{role}</p>
      </div>
      <button
        onClick={toggleFollow}
        style={{
          ...styles.button,
          ...(following() ? styles.buttonFollowing : {}),
        }}
      >
        {following() ? 'Following' : 'Follow'}
      </button>
    </div>
  );
}

const styles = {
  // ... previous styles ...
  button: {
    padding: '0.5rem 1.5rem',
    border: '2px solid #667eea',
    background: 'white',
    color: '#667eea',
    borderRadius: '20px',
    cursor: 'pointer',
    fontWeight: 600,
    transition: 'all 0.2s',
  },
  buttonFollowing: {
    background: '#667eea',
    color: 'white',
  },
};

```

Now your component has state! The follow button toggles between "Follow" and "Following".

JSX Syntax Explained

JSX looks like HTML but it's JavaScript. Here's what you need to know.

Embedding Expressions

Use curly braces {} to embed JavaScript:

```

const name = "World";
const element = <h1>Hello, {name}!</h1>

const value = 42;
const doubled = <p>{value * 2}</p>; // Shows 84

```

Attributes

JSX attributes use camelCase:

```
// HTML: onClick, class, for
// JSX: onClick, className, htmlFor

<button onClick={handleClick} className="btn">
  Click me
</button>

<label htmlFor="email">Email:</label>
<input id="email" type="email" />
```

□□ **Common mistake:** Using `class` instead of `className`:

```
// ⚡ Wrong
<div class="container">

// ⚡ Correct
<div className="container">
```

Children

Components can have children:

```
function Card({ children }: { children: any }) {
  return (
    <div className="card">
      {children}
    </div>
  );
}

// Usage:
<Card>
  <h2>Title</h2>
  <p>Content goes here</p>
</Card>
```

Conditional Rendering

Show elements conditionally:

```
// Using &&
{isLoggedIn && <button>Logout</button>}

// Using ternary
{isLoggedIn ? <Dashboard /> : <Login />}

// Using if/else (outside JSX)
function Greeting({ isLoggedIn }) {
  if (isLoggedIn) {
    return <h1>Welcome back!</h1>;
  }
  return <h1>Please sign in</h1>;
}
```

Lists

Render arrays of elements:

```
const users = [
  { id: 1, name: 'John' },
  { id: 2, name: 'Jane' },
  { id: 3, name: 'Bob' },
];

<ul>
  {users.map(user => (
    <li key={user.id}>{user.name}</li>
  ))}
</ul>
```

□ **Important** Always include a `key` prop when rendering lists. Keys help PhilJS identify which items have changed.

Fragments

Group elements without adding extra DOM nodes:

```

import { Fragment } from '@philjs/core';

function UserInfo() {
  return (
    <Fragment>
      <h2>Name</h2>
      <p>Description</p>
    </Fragment>
  );
}

// Shorthand:
function UserInfo() {
  return (
    <>
      <h2>Name</h2>
      <p>Description</p>
    </>
  );
}

```

Composing Components

The real power of components is composition - building complex UIs from simple pieces.

Example: User List

```

// UserCard.tsx
export function UserCard({ user }) {
  return (
    <div style={styles.card}>
      <h3>{user.name}</h3>
      <p>{user.email}</p>
    </div>
  );
}

// UserList.tsx
import { UserCard } from './UserCard';

export function UserList({ users }) {
  return (
    <div>
      <h2>Team Members</h2>
      {users.map(user => (
        <UserCard key={user.id} user={user} />
      ))}
    </div>
  );
}

// App.tsx
import { UserList } from './UserList';

const team = [
  { id: 1, name: 'John Doe', email: 'john@example.com' },
  { id: 2, name: 'Jane Smith', email: 'jane@example.com' },
];
<UserList users={team} />

```

Nested Components

Components can contain other components:

```

function Avatar({ src, alt }) {
  return <img src={src} alt={alt} style={{ borderRadius: '50%' }} />;
}

function UserInfo({ name, role }) {
  return (
    <div>
      <h3>{name}</h3>
      <p>{role}</p>
    </div>
  );
}

function UserCard({ user }) {
  return (
    <div style={{ display: 'flex', gap: '1rem' }}>
      <Avatar src={user.avatar} alt={user.name} />
      <UserInfo name={user.name} role={user.role} />
    </div>
  );
}

```

Component Best Practices

1. Keep Components Small

Bad:

```
function Dashboard() {
  // 500 Lines of code...
  return <div>...</div>;
}
```

Good:

```
function Dashboard() {
  return (
    <div>
      <Header />
      <Sidebar />
      <MainContent />
      <Footer />
    </div>
  );
}
```

2. Use Descriptive Names

```
// ⚡ Bad
function Card() {}
function Comp() {}
function Thing() {}

// ⚡ Good
function UserProfileCard() {}
function ShoppingCart() {}
function NavigationMenu() {}
```

3. Single Responsibility

Each component should do one thing well:

```
// ⚡ Bad - does too much
function UserDashboard() {
  // Fetches data
  // Handles authentication
  // Renders UI
  // Manages global state
}

// ⚡ Good - focused responsibility
function UserDashboard() {
  const user = useUser(); // Hook handles auth
  const stats = useStats(); // Hook handles data

  return <DashboardUI user={user} stats={stats} />;
}
```

4. Type Your Props

```
// ⚡ Bad
function Button({ label, onClick }) {
  return <button onClick={onClick}>{label}</button>;
}

// ⚡ Good
interface ButtonProps {
  label: string;
  onClick: () => void;
  variant?: 'primary' | 'secondary';
  disabled?: boolean;
}

function Button({ label, onClick, variant = 'primary', disabled }: ButtonProps) {
  return (
    <button onClick={onClick} disabled={disabled} className={variant}>
      {label}
    </button>
  );
}
```

5. Extract Reusable Styles

```
// Reusable style object
const buttonStyles = {
  base: {
    padding: '0.5rem 1rem',
    border: 'none',
    borderRadius: '4px',
    cursor: 'pointer',
  },
  primary: {
    background: '#667eea',
    color: 'white',
  },
  secondary: {
    background: '#e0e0e0',
    color: '#333',
  },
};

function Button({ variant = 'primary', ...props }) {
  return (
    <button
      style={{
        ...buttonStyles.base,
        ...buttonStyles[variant],
      }}
      {...props}
    />
  );
}
```

Common Mistakes to Avoid

1. Forgetting to Return JSX

```
// ⚠ Wrong
function Greeting() {
  <h1>Hello!</h1>
}

// ✅ Correct
function Greeting() {
  return <h1>Hello!</h1>;
}
```

2. Not Exporting Components

```
// ⚠ Wrong - can't import
function MyComponent() {
  return <div>Hello</div>;
}

// ✅ Correct
export function MyComponent() {
  return <div>Hello</div>;
}
```

3. Using Signals Without Calling Them

```
const count = signal(0);

// ⚠ Wrong
<p>Count: {count}</p>

// ✅ Correct
<p>Count: {count()}</p>
```

4. Mutating Props

```
// ⚠ Wrong
function Counter({ initialCount }) {
  initialCount++; // Don't mutate props!
  return <div>{initialCount}</div>;
}

// ✅ Correct
function Counter({ initialCount }) {
  const count = signal(initialCount);
  return (
    <div>
      <p>{count()}</p>
      <button onClick={() => count.set(c => c + 1)}>+</button>
    </div>
  );
}
```

5. Missing Keys in Lists

```
// ⚡ Wrong
{items.map(item => <div>{item.name}</div>)}

// ✅ Correct
{items.map(item => <div key={item.id}>{item.name}</div>)}
```

Complete Example: Product Card

Here's a complete, real-world component:

```
import { signal } from '@philjs/core';

interface Product {
  id: number;
  name: string;
  price: number;
  image: string;
  description: string;
  inStock: boolean;
}

interface ProductCardProps {
  product: Product;
  onAddToCart?: (productId: number) => void;
}

export function ProductCard({ product, onAddToCart }: ProductCardProps) {
  const quantity = signal(1);
  const isHovered = signal(false);

  const handleAddToCart = () => {
    if (onAddToCart) {
      onAddToCart(product.id);
    }
  };

  return (
    <div
      style={{
        ...styles.card,
        transform: isHovered() ? 'translateY(-4px)' : 'translateY(0)',
      }}
      onMouseEnter={() => isHovered.set(true)}
      onMouseLeave={() => isHovered.set(false)}
    >
      <img
        src={product.image}
        alt={product.name}
        style={styles.image}
      />

      <div style={styles.content}>
        <h3 style={styles.name}>{product.name}</h3>
        <p style={styles.description}>{product.description}</p>
      </div>

      <div style={styles.footer}>
        <span style={styles.price}>${product.price.toFixed(2)}</span>
        {product.inStock ? (
          <div style={styles.actions}>
            <select
              value={quantity()}
              onChange={(e) => quantity.set(Number(e.target.value))}>
              style={styles.select}
            >
              {[1, 2, 3, 4, 5].map(n => (
                <option key={n} value={n}>{n}</option>
              )))
            </select>
            <button
              onClick={handleAddToCart}
              style={styles.button}
            >
              Add to Cart
            </button>
          </div>
        ) : (
          <span style={styles.outOfStock}>Out of Stock</span>
        )}
      </div>
    </div>
  );
}
```

```

const styles = {
  card: {
    background: 'white',
    borderRadius: '8px',
    overflow: 'hidden',
    boxShadow: '0 2px 8px rgba(0,0,0,0.1)',
    transition: 'all 0.3s ease',
  },
  image: {
    width: '100%',
    height: '200px',
    objectFit: 'cover' as const,
  },
  content: {
    padding: '1rem',
  },
  name: {
    margin: '0 0 0.5rem',
    fontSize: '1.25rem',
    color: '#333',
  },
  description: {
    margin: '0 0 1rem',
    color: '#666',
    fontSize: '0.9rem',
    lineHeight: 1.5,
  },
  footer: {
    display: 'flex',
    justifyContent: 'space-between',
    alignItems: 'center',
  },
  price: {
    fontSize: '1.5rem',
    fontWeight: 'bold' as const,
    color: '#667eea',
  },
  actions: {
    display: 'flex',
    gap: '0.5rem',
  },
  select: {
    padding: '0.5rem',
    border: '1px solid #ddd',
    borderRadius: '4px',
  },
  button: {
    padding: '0.5rem 1rem',
    background: '#667eea',
    color: 'white',
    border: 'none',
    borderRadius: '4px',
    cursor: 'pointer',
    fontWeight: 600,
  },
  outOfStock: {
    color: '#999',
    fontStyle: 'italic' as const,
  },
};

```

Usage:

```

const product = {
  id: 1,
  name: 'Wireless Headphones',
  price: 79.99,
  image: '/products/headphones.jpg',
  description: 'Premium sound quality with active noise cancellation',
  inStock: true,
};

<ProductCard
  product={product}
  onAddToCart={(id) => console.log('Added product', id)}
/>

```

Next Steps

Now that you understand components, you're ready to:

1. [Learn Signals](#) - Master reactive state management
2. [Build Tic-Tac-Toe](#) - Practice with a complete game
3. [Explore Component Patterns](#) - Advanced composition techniques

Summary

You've learned:

- What components are and why they matter □ How to create function components □ Using props to make components reusable □ JSX syntax and rules □ Adding state with signals □ Composing components together □ Best practices and common mistakes □ Building complete, real-world components
- Components are the foundation of PhilJS. Master them and you're well on your way to building amazing applications!

Next: [Tutorial: Tic-Tac-Toe →](#)

Your First App

This chapter builds a minimal PhilJS app with signals and JSX.

1. Create the component

src/App.tsx

```
import { signal } from "@philjs/core";

const count = signal(0);

export function App() {
  return (
    <main>
      <h1>PhilJS Counter</h1>
      <p>Count: {count}</p>
      <button onClick={() => count.set((c) => c + 1)}>Increment</button>
      <button onClick={() => count.set((c) => c - 1)}>Decrement</button>
    </main>
  );
}
```

2. Create the entry point

src/main.tsx

```
import { render } from "@philjs/core";
import { App } from "./App";

const root = document.getElementById("app");
if (!root) throw new Error("Missing #app root");

render(<App />, root);
```

3. Ensure the HTML shell exists

```
<div id="app"></div>
```

4. Run the app

```
pnpm dev
```

Open the dev server URL to see your counter.

Add styling

Create src/App.css:

```
main {
  font-family: system-ui, -apple-system, sans-serif;
  padding: 2rem;
}

button {
  margin-right: 0.5rem;
}
```

Import it in App.tsx and confirm styling works with Vite HMR.

Add a route

Install the router (if not scaffolded):

```
pnpm add @philjs/router
```

Create src/routes.tsx:

```

import { createAppRouter, Link } from '@philjs/router';
import { App } from './App';

function About() { return <p>About PhilJS</p>; }

createAppRouter({
  target: '#app',
  routes: [
    { path: '/', component: App },
    { path: '/about', component: About },
  ],
});

```

Update `main.tsx` to import `routes.tsx` instead of rendering `<App />` directly. Navigate between routes to confirm client routing works.

Add a loader

```

const routes = [
  {
    path: '/',
    loader: async () => ({ message: 'Hello from loader' }),
    component: ({ data }) => <p>{data.message}</p>,
  },
];

```

Reload; the loader runs before render. This mirrors how you will fetch real data.

Add a test

```

import { render, screen, fireEvent } from '@philjs/testing';
import { App } from './App';

it('increments', () => {
  render(() => <App />);
  fireEvent.click(screen.getByRole('button', { name: /increment/i }));
  expect(screen.getText(/count: 1/i)).toBeInTheDocument();
});

```

Run `pnpm test` to confirm setup.

Checklist

- Counter renders and updates.
- Routing works between `/` and `/about`.
- Loader returns data for `/`.
- Test suite runs.
- Styles load with HMR.

Project Structure

PhilJS keeps the frontend predictable. This is the default layout used by the CLI and examples.

```

my-app/
├─ public/          # Static assets
└─ src/
  ├─ App.tsx       # Root component
  ├─ main.tsx      # Client entry
  ├─ routes/        # Route modules (components + loaders/actions)
  ├─ components/   # Reusable UI components
  ├─ stores/        # Signals + stores
  ├─ styles/        # Global styles and tokens
  └─ entry-server.tsx # SSR entry (optional)
  ├─ tests/         # Unit + integration tests
  ├─ philjs.config.ts # PhilJS configuration
  └─ tsconfig.json
  └─ package.json

```

Why this layout

- `components/` keeps UI primitives isolated and reusable.
- `routes/` pairs views with loaders/actions for local-first data access.
- `stores/` hosts shared state and domain signals.
- `entry-server.tsx` exists only when SSR is enabled.

Files to add early

- `philjs.config.ts`: project-level config for adapters, aliases, and experimental flags.
- `tsconfig.json`: point `jsxImportSource` to `@philjs/core`.

- `vite.config.ts`: include PhilJS plugin + adapters (edge/node).
- `tests/setup.ts`: shared test setup (MSW, testing library, cleanup).
- `.env.example`: declare required env vars (API_URL, FEATURE_FLAGS).

Multi-package workspace

In monorepos, keep `apps/` and `packages/` separate:

```
apps/web/      # PhilJS app
packages/ui/   # Shared components
packages/core/ # Domain logic
packages/adapters/
```

Use `workspace:*` for internal deps and align `tsconfig` paths across packages.

Server entry (SSR)

Add `src/entry-server.tsx` when enabling SSR:

```
import { renderToStream } from '@philjs/ssr';
import { routes } from './routes';

export function handle(request: Request) {
  return renderToStream(<App routes={routes} request={request} />);
}
```

Adapters import this entry to bind to their platforms.

Checklist

- `src/` contains components, routes, stores, styles.
- `entry-server.tsx` exists if SSR is on.
- Shared configs (`tsconfig.base`, `lint`, `prettier`) at the repo root.
- Env vars declared and loaded via platform-safe mechanisms.
- Tests and scripts live under `tests/` or per-package `__tests__/`.

Tutorial: Build a Todo App

Build a full-featured todo application with persistence, filtering, and routing. This tutorial covers real-world patterns you'll use in every PhilJS app.

What You'll Learn

- Managing lists of data
- Form handling and validation
- Local storage persistence
- Filtering and sorting
- URL-based routing for filters
- Component composition
- Performance optimization

What We're Building

A complete todo app with:

- Add new todos
- Mark todos as complete/incomplete
- Delete todos
- Filter: All, Active, Completed
- Persist to localStorage
- URL routing for filters
- Edit existing todos
- Clear completed todos

Setup

```
pnpm create philjs todo-app
cd todo-app
pnpm install
pnpm dev
```

Step 1: Define the Data Model

Create `src/types/todo.ts`:

```

export interface Todo {
  id: number;
  text: string;
  completed: boolean;
  createdAt: number;
}

export type Filter = 'all' | 'active' | 'completed';

export function createTodo(text: string): Todo {
  return {
    id: Date.now(),
    text,
    completed: false,
    createdAt: Date.now(),
  };
}

```

Step 2: Create the TodoItem Component

Create src/components/TodoItem.tsx:

```

import { signal } from '@philjs/core';
import type { Todo } from '../types/todo';

interface TodoItemProps {
  todo: Todo;
  onToggle: (id: number) => void;
  onDelete: (id: number) => void;
  onUpdate: (id: number, text: string) => void;
}

export function TodoItem({ todo, onToggle, onDelete, onUpdate }: TodoItemProps) {
  const isEditing = signal(false);
  const editText = signal(todo.text);

  const handleEdit = () => {
    isEditing.set(true);
    editText.set(todo.text);
  };

  const handleSave = () => {
    const text = editText().trim();
    if (text) {
      onUpdate(todo.id, text);
      isEditing.set(false);
    }
  };

  const handleCancel = () => {
    editText.set(todo.text);
    isEditing.set(false);
  };

  const handleKeyDown = (e: KeyboardEvent) => {
    if (e.key === 'Enter') {
      handleSave();
    } else if (e.key === 'Escape') {
      handleCancel();
    }
  };
}

return (
  <li style={styles.item}>
    {isEditing() ? (
      <div style={styles.editing}>
        <input
          type="text"
          value={editText()}
          onInput={(e) => editText.set((e.target as HTMLInputElement).value)}
          onKeyDown={handleKeyDown}
          onBlur={handleSave}
          style={styles.editInput}
          autoFocus
        />
      </div>
    ) : (
      <div style={styles.view}>
        <input
          type="checkbox"
          checked={todo.completed}
          onChange={() => onToggle(todo.id)}
          style={styles.checkbox}
        />
        <label
          onDoubleClick={handleEdit}
        > ...
      </div>
    )}
  </li>
)

```

```

        style={{
          ...styles.label,
          ...(todo.completed ? styles.labelCompleted : {}),
        }}
      >
      {todo.text}
    </label>

    <button
      onClick={() => onDelete(todo.id)}
      style={styles.deleteButton}
      aria-label="Delete todo"
    >
      x
    </button>
  </div>
</li>
);
};

const styles = {
  item: {
    listStyle: 'none',
    borderBottom: '1px solid #eddede',
    padding: '0',
  },
  view: {
    display: 'flex',
    alignItems: 'center',
    padding: '1rem',
    gap: '1rem',
  },
  editing: {
    padding: '1rem',
  },
  checkbox: {
    width: '20px',
    height: '20px',
    cursor: 'pointer',
  },
  label: {
    flex: 1,
    cursor: 'pointer',
    fontSize: '1.1rem',
    userSelect: 'none' as const,
    transition: 'color 0.2s',
  },
  labelCompleted: {
    textDecoration: 'line-through',
    color: '#999',
  },
  deleteButton: {
    background: 'none',
    border: 'none',
    color: '#cc9999',
    fontSize: '2rem',
    cursor: 'pointer',
    opacity: 0,
    transition: 'opacity 0.2s',
  },
  editInput: {
    width: '100%',
    padding: '0.75rem',
    fontSize: '1.1rem',
    border: '1px solid #999',
    borderRadius: '4px',
  },
};

```

□ Note: Double-click a todo to edit it. Press Enter to save, Escape to cancel.

Step 3: Create localStorage Utilities

Create `src/utils/storage.ts`:

```

import type { Todo } from '../types/todo';

const STORAGE_KEY = 'philjs-todos';

export function loadTodos(): Todo[] {
  try {
    const data = localStorage.getItem(STORAGE_KEY);
    return data ? JSON.parse(data) : [];
  } catch (error) {
    console.error('Failed to load todos:', error);
    return [];
  }
}

export function saveTodos(todos: Todo[]): void {
  try {
    localStorage.setItem(STORAGE_KEY, JSON.stringify(todos));
  } catch (error) {
    console.error('Failed to save todos:', error);
  }
}

```

Step 4: Create the Main TodoApp Component

Create src/components/TodoApp.tsx:

```

import { signal, memo, effect } from '@philjs/core';
import { TodoItem } from './TodoItem';
import { createTodo } from '../types/todo';
import type { Todo, Filter } from '../types/todo';
import { loadTodos, saveTodos } from '../utils/storage';

type TodoAppProps = {
  url: URL;
  navigate: (to: string) => Promise<void>;
};

export function TodoApp({ url, navigate }: TodoAppProps) {
  // Load initial todos from LocalStorage
  const todos = signal<Todo[]>(loadTodos());

  // Get filter from URL (?filter=active)
  const filterParam = (url.searchParams.get('filter') as Filter) || 'all';
  const filter = signal<Filter>(filterParam);

  // New todo input
  const newTodoText = signal('');

  // Persist todos to LocalStorage whenever they change
  effect(() => {
    saveTodos(todos());
  });

  // Filtered todos based on current filter
  const filteredTodos = memo(() => {
    const currentFilter = filter();
    const allTodos = todos();

    switch (currentFilter) {
      case 'active':
        return allTodos.filter(t => !t.completed);
      case 'completed':
        return allTodos.filter(t => t.completed);
      default:
        return allTodos;
    }
  });

  // Counts
  const activeCount = memo(() => {
    todos().filter(t => !t.completed).length
  });

  const completedCount = memo(() => {
    todos().filter(t => t.completed).length
  });

  const allCompleted = memo(() => {
    todos().length > 0 && activeCount() === 0
  });

  // Actions
  const addTodo = () => {
    const text = newTodoText().trim();
    if (text) {
      todos.set([...todos(), createTodo(text)]);
      newTodoText.set('');
    }
  };
}

```

```

    }

};

const toggleTodo = (id: number) => {
  todos.set(
    todos().map(todo =>
      todo.id === id
        ? { ...todo, completed: !todo.completed }
        : todo
    )
  );
};

const deleteTodo = (id: number) => {
  todos.set(todos().filter(todo => todo.id !== id));
};

const updateTodo = (id: number, text: string) => {
  todos.set(
    todos().map(todo =>
      todo.id === id ? { ...todo, text } : todo
    )
  );
};

const toggleAll = () => {
  const shouldComplete = !allCompleted();
  todos.set(
    todos().map(todo => ({ ...todo, completed: shouldComplete }))
  );
};

const clearCompleted = () => {
  todos.set(todos().filter(todo => !todo.completed));
};

const setFilter = async (newFilter: Filter) => {
  filter.set(newFilter);
  const next = new URL(window.location.href);
  if (newFilter === 'all') {
    next.searchParams.delete('filter');
  } else {
    next.searchParams.set('filter', newFilter);
  }
  await navigate(next.pathname + next.search);
};

const handleKeyDown = (e: KeyboardEvent) => {
  if (e.key === 'Enter') {
    addTodo();
  }
};

return (
  <div style={styles.container}>
    <header style={styles.header}>
      <h1 style={styles.title}>todos</h1>

      <div style={styles.inputContainer}>
        {todos().length > 0 && (
          <button
            onClick={toggleAll}
            style={styles.toggleAll}
            aria-label="Toggle all todos"
          >
            ☰
          </button>
        )}
        <input
          type="text"
          placeholder="What needs to be done?"
          value={newTodoText()}
          onChange={(e) => newTodoText.set((e.target as HTMLInputElement).value)}
          onKeyDown={handleKeyDown}
          style={styles.input}
          autofocus
        />
      </div>
    </header>

    {todos().length > 0 && (
      <>
        <ul style={styles.todoList}>
          {filteredTodos().map(todo => (
            <TodoItem
              key={todo.id}
              todo={todo}
              onToggle={toggleTodo}
            </TodoItem>
          ))
        </ul>
    )}
  </div>
);

```

```

        onDelete={deleteTodo}
        onUpdate={updateTodo}
      />
    )})
</ul>

<footer style={styles.footer}>
  <span style={styles.count}>
    <strong>{activeCount()}</strong>{' '}
    {activeCount() === 1 ? 'item' : 'items'} left
  </span>

  <div style={styles.filters}>
    <button
      onClick={() => setFilter('all')}
      style={{
        ...styles.filterButton,
        ...(filter() === 'all' ? styles.filterButtonActive : {}),
      }}
    >
      All
    </button>
    <button
      onClick={() => setFilter('active')}
      style={{
        ...styles.filterButton,
        ...(filter() === 'active' ? styles.filterButtonActive : {}),
      }}
    >
      Active
    </button>
    <button
      onClick={() => setFilter('completed')}
      style={{
        ...styles.filterButton,
        ...(filter() === 'completed' ? styles.filterButtonActive : {}),
      }}
    >
      Completed
    </button>
  </div>

  {completedCount() > 0 && (
    <button
      onClick={clearCompleted}
      style={styles.clearButton}
    >
      Clear completed
    </button>
  )}
  </footer>
</>
)});

{todos().length === 0 && (
  <div style={styles.empty}>
    <p>No todos yet. Add one above to get started!</p>
  </div>
)}
</div>
);
}

const styles = {
  container: {
    maxWidth: '550px',
    margin: '0 auto',
    background: 'white',
    boxShadow: '0 2px 4px 0 rgba(0, 0, 0, 0.2), 0 25px 50px 0 rgba(0, 0, 0, 0.1)',
  },
  header: {
    padding: '0',
  },
  title: {
    fontSize: '6rem',
    fontWeight: 100,
    textAlign: 'center' as const,
    color: 'rgba(175, 47, 47, 0.15)',
    margin: '40px 0 0',
  },
  inputContainer: {
    position: 'relative' as const,
    display: 'flex',
    alignItems: 'center',
  },
  toggleAll: {
    position: 'absolute' as const,
    left: '10px',
    top: '50%',
  }
};

```

```
        transform: 'translateY(-50%)',
        border: 'none',
        background: 'none',
        fontSize: '1.5rem',
        color: '#e6e6e6',
        cursor: 'pointer',
    },
    input: {
        flex: 1,
        padding: '16px 16px 16px 60px',
        fontSize: '24px',
        border: 'none',
        borderBottom: '1px solid #ededed',
        outline: 'none',
    },
    todoList: {
        margin: 0,
        padding: 0,
    },
    footer: {
        padding: '10px 15px',
        display: 'flex',
        justifyContent: 'space-between',
        alignItems: 'center',
        borderTop: '1px solid #e6e6e6',
        fontSize: '14px',
        color: '#777',
    },
    count: {},
    filters: {
        display: 'flex',
        gap: '0.5rem',
    },
    filterButton: {
        padding: '3px 7px',
        border: '1px solid transparent',
        borderRadius: '3px',
        background: 'none',
        cursor: 'pointer',
        color: '#777',
    },
    filterButtonActive: {
        borderColor: 'rgba(175, 47, 47, 0.2)',
    },
    clearButton: {
        background: 'none',
        border: 'none',
        color: '#777',
        cursor: 'pointer',
        textDecoration: 'underline',
    },
    empty: {
        padding: '2rem',
        textAlign: 'center' as const,
        color: '#999',
    },
};
```

Step 5: Wire up the router

Create `src/router.ts` so the current low-level APIs can mount and navigate your routes:

```

import { createRouter } from '@philjs/router';
import { render } from '@philjs/core';

type RouteEntry = {
  pattern: string;
  load: () => Promise<{ default: (props: any) => any }>;
};

const routes: RouteEntry[] = [
  { pattern: '/', load: () => import('./routes/index.tsx') },
];

const router = createRouter(
  Object.fromEntries(routes.map((route) => [route.pattern, route.load]))
);

let outlet: HTMLElement | null = null;

async function renderRoute(url: URL) {
  if (!outlet) throw new Error('Router not started');

  const entry =
    routes.find((route) => route.pattern === url.pathname) ?? routes[0];
  const loader = router.manifest[entry.pattern];
  const mod = await loader();
  const Component = mod.default;
  render(() => Component({ url, navigate: navigateInternal }), outlet);
}

async function navigateInternal(to: string) {
  const url = new URL(to, window.location.origin);
  window.history.pushState({}, '', url.toString());
  await renderRoute(url);
}

export const navigate = navigateInternal;

export function startRouter(target: HTMLElement) {
  outlet = target;

  document.addEventListener('click', (event) => {
    const anchor = (event.target as HTMLElement).closest<HTMLAnchorElement>('a[data-router-link]');
    if (!anchor || anchor.target === '_blank' || anchor.hasAttribute('download')) return;

    const url = new URL(anchor.href);
    if (url.origin !== window.location.origin) return;

    event.preventDefault();
    void navigateInternal(url.pathname + url.search + url.hash);
  });

  window.addEventListener('popstate', () => {
    renderRoute(new URL(window.location.href));
  });

  renderRoute(new URL(window.location.href));
}

```

Update `src/main.tsx` to bootstrap the router:

```

import { startRouter } from './router.ts';

startRouter(document.getElementById('app')!);

```

Now your routes render through the shared router, and every component receives the `url` and `navigate` props shown above.

Step 6: Create the Main App

Update `src/routes/index.tsx`:

```

import { TodoApp } from './components/TodoApp';

type HomeProps = {
  url: URL;
  navigate: (to: string) => Promise<void>;
};

export default function Home({ url, navigate }: HomeProps) {
  return (
    <div style={styles.app}>
      <TodoApp url={url} navigate={navigate} />

      <footer style={styles.info}>
        <p>Double-click to edit a todo</p>
        <p>Created with PhilJS</p>
      </footer>
    </div>
  );
}

const styles = {
  app: {
    minHeight: '100vh',
    background: '#f5f5f5',
    padding: '40px 20px',
  },
  info: {
    textAlign: 'center' as const,
    marginTop: '40px',
    color: '#bfbfbf',
    fontSize: '12px',
  },
};

```

Understanding the Code

State Persistence with Effects

```

effect(() => {
  saveTodos(todos());
});

```

This effect runs whenever `todos` changes, automatically saving to `localStorage`. No manual save calls needed!

Derived State with Memos

```

const filteredTodos = memo(() => {
  const currentFilter = filter();
  const allTodos = todos();

  switch (currentFilter) {
    case 'active':
      return allTodos.filter(t => !t.completed);
    case 'completed':
      return allTodos.filter(t => t.completed);
    default:
      return allTodos;
  }
});

```

`filteredTodos` automatically recalculates when `filter` or `todos` changes. The component always renders the correct filtered list.

URL-Based Filtering

```

const filterParam = (url.searchParams.get('filter') as Filter) || 'all';
const filter = signal<Filter>(filterParam);

const setFilter = async (newFilter: Filter) => {
  filter.set(newFilter);
  const next = new URL(window.location.href);
  if (newFilter === 'all') {
    next.searchParams.delete('filter');
  } else {
    next.searchParams.set('filter', newFilter);
  }
  await navigate(next.pathname + next.search);
};

```

The current filter lives in the URL (`?filter=active`). Updating it keeps the UI in sync with the browser history and makes filtered views shareable.

Immutable Updates

```

const toggleTodo = (id: number) => {
  todos.set(
    todos().map(todo =>
      todo.id === id
        ? { ...todo, completed: !todo.completed }
        : todo
    )
  );
};

```

We create a new array instead of mutating. This is crucial for:

- Proper reactivity
- Time travel debugging
- Avoiding bugs

Step 7: Add Features

Bulk Actions

```

const selectAll = () => {
  todos.set(todos().map(todo => ({ ...todo, completed: true })));
};

const deselectAll = () => {
  todos.set(todos().map(todo => ({ ...todo, completed: false })));
};

const deleteSelected = () => {
  todos.set(todos().filter(todo => !todo.completed));
};

```

Sorting

```

type SortBy = 'created' | 'alphabetical';

const sortBy = signal<SortBy>('created');

const sortedTodos = memo(() => {
  const filtered = filteredTodos();

  if (sortBy() === 'alphabetical') {
    return [...filtered].sort((a, b) => a.text.localeCompare(b.text));
  }

  return filtered; // Already sorted by creation time
});

// In render:
<select
  value={sortBy()}
  onChange={(e) => sortBy.set(e.target.value as SortBy)}
>
  <option value="created">Sort by Date</option>
  <option value="alphabetical">Sort A-Z</option>
</select>

```

Statistics

```

const stats = memo(() => {
  const all = todos();
  return {
    total: all.length,
    active: all.filter(t => !t.completed).length,
    completed: all.filter(t => t.completed).length,
    completionRate: all.length > 0
      ? (all.filter(t => t.completed).length / all.length * 100).toFixed(0)
      : 0,
  };
});

// In render:
<div style={styles.stats}>
  <p>Total: {stats().total}</p>
  <p>Completion: {stats().completionRate}%</p>
</div>

```

Due Dates

```
// Update Todo type
export interface Todo {
  id: number;
  text: string;
  completed: boolean;
  createdAt: number;
  dueDate?: number; // Add this
}

// In component:
const isOverdue = (todo: Todo) => {
  return todo.dueDate && !todo.completed && Date.now() > todo.dueDate;
};

// Render with warning:
<label
  style={{
    ...styles.label,
    ...(todo.completed ? styles.labelCompleted : {}),
    ...(!isOverdue(todo) ? styles.labelOverdue : {}),
  }}
>
  {todo.text}
  {isOverdue(todo) && ' ⚠'}
</label>
```

Performance Optimization

Virtualized Lists

For large lists (1000+ items), use virtual scrolling:

```
import { createVirtualizer } from '@tanstack/virtual-core';

const virtualized = createVirtualizer({
  count: filteredTodos().length,
  getScrollElement: () => containerRef.current,
  estimateSize: () => 60,
});

// Render only visible items
{virtualized.getVirtualItems().map(virtualItem => {
  const todo = filteredTodos()[virtualItem.index];
  return <TodoItem key={todo.id} todo={todo} ... />;
})}
```

Debounced Search

```
import { debounce } from 'lodash-es';

const searchQuery = signal('');
const debouncedSearch = debounce((value: string) => {
  searchQuery.set(value);
}, 300);

const searchedTodos = memo(() => {
  const query = searchQuery().toLowerCase();
  if (!query) return filteredTodos();

  return filteredTodos().filter(todo =>
    todo.text.toLowerCase().includes(query)
  );
});

// In render:
<input
  type="search"
  placeholder="Search todos..."
  onInput={(e) => debouncedSearch(e.target.value)}
/>
```

Testing

Create `src/components/TodoApp.test.tsx`:

```

import { render, screen, fireEvent } from '@testing-library/react';
import { TodoApp } from './TodoApp';

const mockNavigate = async () => {};

describe('TodoApp', () => {
  it('adds a new todo', async () => {
    render(<TodoApp url={new URL('http://localhost/')} navigate={mockNavigate} />);

    const input = screen.getByPlaceholderText('What needs to be done?');
    fireEvent.change(input, { target: { value: 'Buy milk' } });
    fireEvent.keyDown(input, { key: 'Enter' });

    expect(screen.getByText('Buy milk')).toBeInTheDocument();
  });

  it('toggles a todo', () => {
    render(<TodoApp url={new URL('http://localhost/')} navigate={mockNavigate} />);

    const checkbox = screen.getByRole('checkbox');
    fireEvent.click(checkbox);

    expect(checkbox).toBeChecked();
  });

  it('filters todos', () => {
    render(<TodoApp url={new URL('http://localhost/?filter=completed')} navigate={mockNavigate} />);

    // Add todos
    // ...

    fireEvent.click(screen.getByText('Active'));

    // Should only show active todos
  });
});

```

What You Learned

□ **List management** - Adding, updating, deleting items □ **Forms** - Handling user input and validation □ **Persistence** - Saving to localStorage □ **Effects** - Running side effects when state changes □ **Filtering** - Showing subsets of data □ **URL state** - Storing filter in URL params □ **Composition** - Breaking UI into components □ **Performance** - Optimizing with memoization and virtual scrolling

Challenges

Extend the todo app:

1. **Tags**: Add tags to todos and filter by tag
2. **Categories**: Organize todos into projects
3. **Cloud sync**: Save to a backend API
4. **Collaboration**: Share todo lists with others
5. **Recurring todos**: Daily/weekly repeating tasks
6. **Subtasks**: Nested checklist items
7. **Drag and drop**: Reorder by dragging
8. **Dark mode**: Theme switching
9. **Offline support**: Service worker for PWA
10. **Mobile app**: Build with Capacitor

Next Steps

- [Build a Blog with SSG](#) - Learn static generation
- [Learn about Routing](#) - Multi-page apps
- [Data Fetching](#) - API integration

Next: [Tutorial: Blog with SSG](#) →

Tutorial: Build Tic-Tac-Toe

Build a complete tic-tac-toe game while learning core PhilJS concepts. This tutorial covers state management, event handling, conditional rendering, and more.

What You'll Learn

- Managing game state with signals
- Handling click events
- Computing derived state with memoization
- Implementing game logic
- Building reusable components
- Time travel functionality (undo moves)

What We're Building

By the end of this tutorial, you'll have a fully functional tic-tac-toe game with: - A 3x3 game board - Turn indicators (X or O) - Win detection - Game history with time travel - Move highlighting

Setup

Create a new PhilJS project:

```
pnpm create philjs tic-tac-toe
cd tic-tac-toe
pnpm install
pnpm dev
```

Step 1: Create the Square Component

Let's start with the basic building block - a single square.

Create `src/components/Square.tsx`:

```
interface SquareProps {
  value: string | null;
  onClick: () => void;
  isWinning?: boolean;
}

export function Square({ value, onClick, isWinning }: SquareProps) {
  return (
    <button
      onClick={onClick}
      style={{
        ...styles.square,
        ...(isWinning ? styles.winning : {}),
      }}
    >
      {value}
    </button>
  );
}

const styles = {
  square: {
    width: '80px',
    height: '80px',
    background: 'white',
    border: '2px solid #999',
    fontSize: '2rem',
    fontWeight: 'bold' as const,
    cursor: 'pointer',
    transition: 'all 0.2s',
  },
  winning: {
    background: '#90EE90',
  },
};
```

What's happening: - `Square` accepts the square's value (X, O, or null) - `onClick` callback handles clicks - `isWinning` highlights winning squares

Step 2: Create the Board Component

Now let's create the game board using our `Square` component.

Create `src/components/Board.tsx`:

```

import { Square } from './Square';

type SquareValue = 'X' | 'O' | null;

interface BoardProps {
  squares: SquareValue[];
  onClick: (index: number) => void;
  winningSquares: number[] | null;
}

export function Board({ squares, onClick, winningSquares }: BoardProps) {
  const renderSquare = (index: number) => {
    return (
      <Square
        value={squares[index]}
        onClick={() => onClick(index)}
        isWinning={(winningSquares?.includes(index) || false)}
      />
    );
  };

  return (
    <div>
      <div style={styles.row}>
        {renderSquare(0)}
        {renderSquare(1)}
        {renderSquare(2)}
      </div>
      <div style={styles.row}>
        {renderSquare(3)}
        {renderSquare(4)}
        {renderSquare(5)}
      </div>
      <div style={styles.row}>
        {renderSquare(6)}
        {renderSquare(7)}
        {renderSquare(8)}
      </div>
    </div>
  );
}

const styles = {
  row: {
    display: 'flex',
  },
};

```

Key points: - Board receives the game state as props - Each square gets its value from the squares array - Clicks are handled by a callback passed down - Winning squares are highlighted

Step 3: Add Game Logic

Create the main Game component with full game logic.

Create src/components/Game.tsx:

```

import { signal, memo } from '@philjs/core';
import { Board } from './Board';

type SquareValue = 'X' | 'O' | null;

interface GameState {
  squares: SquareValue[];
  xIsNext: boolean;
}

export function Game() {
  // Store all game history
  const history = signal<GameState[]>([
    {
      squares: Array(9).fill(null),
      xIsNext: true,
    },
  ]);

  // Current move index
  const currentMove = signal(0);

  // Get current game state
  const current = memo(() => history()[currentMove()]);

  // Calculate winner
  const winner = memo(() => {
    const squares = current().squares;
    const lines = [
      [0, 1, 2], [3, 4, 5], [6, 7, 8], // Rows
      [0, 3, 6], [1, 4, 7], [2, 5, 8], // Columns
    ];
  });
}

```

```

        [0, 4, 8], [2, 4, 6],           // Diagonals
    ];
}

for (const [a, b, c] of lines) {
    if (squares[a] && squares[a] === squares[b] && squares[a] === squares[c]) {
        return { player: squares[a], line: [a, b, c] };
    }
}
return null;
});

// Check if board is full (draw)
const isDraw = memo(() => {
    return !winner() && current().squares.every(square => square !== null);
});

// Handle square click
const handleClick = (index: number) => {
    // Don't allow moves if game is over or square is filled
    if (winner() || current().squares[index] || isDraw()) {
        return;
    }

    // Create new game state
    const newSquares = [...current().squares];
    newSquares[index] = current().xIsNext ? 'X' : 'O';

    // Add to history (remove future moves if we went back in time)
    const newHistory = [
        ...history().slice(0, currentMove() + 1),
        {
            squares: newSquares,
            xIsNext: !current().xIsNext,
        },
    ];

    history.set(newHistory);
    currentMove.set(newHistory.length - 1);
};

// Jump to a specific move
const jumpTo = (move: number) => {
    currentMove.set(move);
};

// Reset game
const resetGame = () => {
    history.set([
        {
            squares: Array(9).fill(null),
            xIsNext: true,
        },
    ]);
    currentMove.set(0);
};

// Game status message
const status = memo(() => {
    if (winner()) {
        return `Winner: ${winner().player}`;
    }
    if (isDraw()) {
        return "Draw!";
    }
    return `Next player: ${current().xIsNext ? 'X' : 'O'}`;
});

return (
    <div style={styles.container}>
        <h1 style={styles.title}>Tic-Tac-Toe</h1>

        <div style={styles.game}>
            <div style={styles.boardContainer}>
                <Board
                    squares={current().squares}
                    onClick={handleClick}
                    winningSquares={winner()?.line || null}
                />
            </div>
            <div
                style={{
                    ...styles.status,
                    ...(winner() ? styles.statusWinner : {}),
                }}
            >
                {status()}
            </div>
        </div>
        <button onClick={resetGame} style={styles.resetButton}>

```

```

        New Game
    </button>
</div>

<div style={styles.history}>
<h3>History</h3>
<ol>
{history().map((_, move) => {
  const description = move === 0
    ? 'Go to game start'
    : `Go to move #${move}`;

  const isCurrent = move === currentMove();

  return (
    <li key={move}>
      <button
        onClick={() => jumpTo(move)}
        style={{
          ...styles.historyButton,
          ...(isCurrent ? styles.historyButtonCurrent : {}),
        }}
      >
        {description}
      </button>
    </li>
  );
})
</ol>
</div>
</div>
</div>
);

const styles = {
  container: {
    display: 'flex',
    flexDirection: 'column' as const,
    alignItems: 'center',
    padding: '2rem',
    minHeight: '100vh',
    background: 'linear-gradient(135deg, #667eea 0%, #764ba2 100%)',
  },
  title: {
    color: 'white',
    fontSize: '3rem',
    marginBottom: '2rem',
    textShadow: '2px 2px 4px rgba(0,0,0,0.2)',
  },
  game: {
    display: 'flex',
    gap: '3rem',
    background: 'white',
    padding: '2rem',
    borderRadius: '20px',
    boxShadow: '0 10px 40px rgba(0,0,0,0.3)',
  },
  boardContainer: {
    display: 'flex',
    flexDirection: 'column' as const,
    alignItems: 'center',
  },
  status: {
    marginTop: '1.5rem',
    fontSize: '1.5rem',
    fontWeight: 'bold' as const,
    color: '#333',
  },
  statusWinner: {
    color: '#4CAF50',
  },
  resetButton: {
    marginTop: '1rem',
    padding: '0.75rem 2rem',
    fontSize: '1rem',
    background: '#667eea',
    color: 'white',
    border: 'none',
    borderRadius: '8px',
    cursor: 'pointer',
    fontWeight: 'bold' as const,
    transition: 'all 0.2s',
  },
  history: {
    minWidth: '200px',
  },
  historyButton: {
    padding: '0.5rem 1rem',
  }
};

```

```
    },
    historyButtonCurrent: {
      background: '#667eea',
      color: 'white',
      fontWeight: 'bold' as const,
    },
  );
}
```

Step 4: Wire Everything Up

Update `src/routes/index.tsx`:

```
import { Game } from '../components/Game';

export default function Home() {
  return <Game />;
}
```

Run `pnpm dev` and play your game!

Understanding the Code

State Management with Signals

```
const history = signal<GameState[]>([...]);
const currentMove = signal(0);
```

We use signals to store: - `history`: Array of all game states (for time travel) - `currentMove`: Index of the current move we're viewing

Computed Values with Memos

```
const current = memo(() => history()[currentMove()]);
const winner = memo(() => {
  // Calculate winner from current squares
});
```

Memos automatically recalculate when dependencies change: - `current` updates when `history` or `currentMove` changes - `winner` updates when `current` changes

Immutable Updates

```
const newSquares = [...current().squares];
newSquares[index] = current().xIsNext ? 'X' : 'O';
```

We create a new array instead of modifying the existing one. This is crucial for: - Time travel to work correctly - React properly to state changes - Avoiding bugs

Time Travel Implementation

```
const jumpTo = (move: number) => {
  currentMove.set(move);
};
```

Time travel is simple! Just set `currentMove` to a different index. The UI automatically updates because all computed values depend on `currentMove`.

Step 5: Add Enhancements

Show Move Coordinates

Let's show which square was clicked in each move:

```

// In Game component
interface HistoryEntry {
  squares: SquareValue[];
  xIsNext: boolean;
  lastMove?: number; // Add this
}

// Update handleClick
const newHistory = [
  ...history().slice(0, currentMove() + 1),
  {
    squares: newSquares,
    xIsNext: !current().xIsNext,
    lastMove: index, // Store the move
  },
];
;

// Update history rendering
const getMoveDescription = (move: number) => {
  if (move === 0) return 'Game start';

  const entry = history()[move];
  const row = Math.floor(entry.lastMove! / 3) + 1;
  const col = (entry.lastMove! % 3) + 1;

  return `Move #${move} (Row ${row}, Col ${col})`;
};

// In render:
<button onClick={() => jumpTo(move)}>
  {getMoveDescription(move)}
</button>

```

Add Animations

Add smooth transitions when squares are clicked:

```

// Update Square styles
const styles = {
  square: {
    // ... existing styles ...
    transform: 'scale(1)',
    transition: 'all 0.2s',
  },
  // Add hover effect
  squareHover: {
    transform: 'scale(1.05)',
    boxShadow: '0 4px 8px rgba(0,0,0,0.2)',
  },
};

// Update Square component
export function Square({ value, onClick, isWinning }: SquareProps) {
  const isHovered = signal(false);

  return (
    <button
      onClick={onClick}
      onMouseEnter={() => !value && isHovered.set(true)}
      onMouseLeave={() => isHovered.set(false)}
      style={{
        ...styles.square,
        ...(isWinning ? styles.winning : {}),
        ...(!isHovered() ? styles.squareHover : {}),
      }}
    >
      {value}
    </button>
  );
}

```

Add Sound Effects

```

const playSound = (type: 'move' | 'win') => {
  const audio = new Audio('/sounds/${type}.mp3');
  audio.play();
};

// In handleClick:
playSound('move');

// When winner is detected:
if (winner()) {
  playSound('win');
}

```

Complete Source Code

Here's the complete, production-ready code:

[src/components/Square.tsx](#):

```
import { signal } from '@philjs/core';

interface SquareProps {
  value: string | null;
  onClick: () => void;
  isWinning?: boolean;
}

export function Square({ value, onClick, isWinning }: SquareProps) {
  const isHovered = signal(false);

  return (
    <button
      onClick={onClick}
      onMouseEnter={() => !value && isHovered.set(true)}
      onMouseLeave={() => isHovered.set(false)}
      style={{
        ...styles.square,
        ...(isWinning ? styles.winning : {}),
        ...(isHovered() && !value ? styles.squareHover : {}),
      }}
      disabled={!value}
    >
      {value}
    </button>
  );
}

const styles = {
  square: {
    width: '80px',
    height: '80px',
    background: 'white',
    border: '2px solid #999',
    fontSize: '2rem',
    fontWeight: 'bold' as const,
    cursor: 'pointer',
    transition: 'all 0.2s',
    transform: 'scale(1)',
  },
  winning: {
    background: '#90EE90',
    animation: 'pulse 0.5s ease-in-out infinite',
  },
  squareHover: {
    transform: 'scale(1.05)',
    boxShadow: '0 4px 8px rgba(0,0,0,0.2)',
    background: '#f0f0f0',
  },
};
```

[src/components/Board.tsx](#):

```

import { Square } from './Square';

type SquareValue = 'X' | 'O' | null;

interface BoardProps {
  squares: SquareValue[];
  onClick: (index: number) => void;
  winningSquares: number[] | null;
}

export function Board({ squares, onClick, winningSquares }: BoardProps) {
  return (
    <div style={styles.board}>
      <Array(3).fill(null).map((_, row) => (
        <div key={row} style={styles.row}>
          <Array(3).fill(null).map((_, col) => {
            const index = row * 3 + col;
            return (
              <Square
                key={index}
                value={squares[index]}
                onClick={() => onClick(index)}
                isWinning={winningSquares?.includes(index) || false}
              />
            );
          )));
      </div>
    </div>
  );
}

const styles = {
  board: {
    display: 'inline-block',
  },
  row: {
    display: 'flex',
  },
};

```

src/components/Game.tsx:

```

import { signal, memo } from '@philjs/core';
import { Board } from './Board';

type SquareValue = 'X' | 'O' | null;

interface GameState {
  squares: SquareValue[];
  xIsNext: boolean;
  lastMove?: number;
}

export function Game() {
  const history = signal<GameState[]>([
    { squares: Array(9).fill(null), xIsNext: true },
  ]);
  const currentMove = signal(0);

  const current = memo(() => history()[currentMove()]);

  const winner = memo(() => {
    const squares = current().squares;
    const lines = [
      [0, 1, 2], [3, 4, 5], [6, 7, 8],
      [0, 3, 6], [1, 4, 7], [2, 5, 8],
      [0, 4, 8], [2, 4, 6],
    ];
    for (const [a, b, c] of lines) {
      if (squares[a] && squares[a] === squares[b] && squares[a] === squares[c]) {
        return { player: squares[a], line: [a, b, c] };
      }
    }
    return null;
  });

  const isDraw = memo(() => {
    return !winner() && current().squares.every(square => square !== null);
  });

  const handleClick = (index: number) => {
    if (winner() || current().squares[index] || isDraw()) {
      return;
    }

    const newSquares = [...current().squares];

```

```

newSquares[index] = current().xIsNext ? 'X' : 'O';

const newHistory = [
  ...history().slice(0, currentMove() + 1),
  {
    squares: newSquares,
    xIsNext: !current().xIsNext,
    lastMove: index,
  },
];

history.set(newHistory);
currentMove.set(newHistory.length - 1);
};

const jumpTo = (move: number) => currentMove.set(move);

const resetGame = () => {
  history.set([{ squares: Array(9).fill(null), xIsNext: true }]);
  currentMove.set(0);
};

const status = memo(() => {
  if (winner()) return `Winner: ${winner().player}`;
  if (isDraw()) return "Draw!";
  return `Next player: ${current().xIsNext ? 'X' : 'O'}`;
});

const getMoveDescription = (move: number) => {
  if (move === 0) return 'Game start';
  const entry = history()[move];
  const row = Math.floor(entry.lastMove! / 3) + 1;
  const col = (entry.lastMove! % 3) + 1;
  return `Move #${move} (Row ${row}, Col ${col})`;
};

return (
  <div style={styles.container}>
    <h1 style={styles.title}>Tic-Tac-Toe</h1>
    <div style={styles.game}>
      <div style={styles.boardContainer}>
        <Board
          squares={current().squares}
          onClick={handleClick}
          winningSquares={winner()?.line || null}
        />
        <div style={{...styles.status, ...(winner() ? styles.statusWinner : {})}>
          {status()}
        </div>
        <button onClick={resetGame} style={styles.resetButton}>
          New Game
        </button>
      </div>
      <div style={styles.history}>
        <h3>History</h3>
        <ol>
          {history().map((_, move) => {
            const isCurrent = move === currentMove();
            return (
              <li key={move}>
                <button
                  onClick={() => jumpTo(move)}
                  style={{
                    ...styles.historyButton,
                    ...(isCurrent ? styles.historyButtonCurrent : {}),
                  }}
                >
                  {getMoveDescription(move)}
                </button>
              </li>
            );
          ))}
        </ol>
      </div>
    </div>
  );
}

const styles = {
  container: {
    display: 'flex',
    flexDirection: 'column' as const,
    alignItems: 'center',
    padding: '2rem',
    minHeight: '100vh',
  }
};

```

```

        background: 'linear-gradient(135deg, #667eea 0%, #764ba2 100%)',
    },
    title: {
        color: 'white',
        fontSize: '3rem',
        marginBottom: '2rem',
        textShadow: '2px 2px 4px rgba(0,0,0,0.2)',
    },
    game: {
        display: 'flex',
        gap: '1rem',
        background: 'white',
        padding: '2rem',
        borderRadius: '20px',
        boxShadow: '0 10px 40px rgba(0,0,0,0.3)',
    },
    boardContainer: {
        display: 'flex',
        flexDirection: 'column' as const,
        alignItems: 'center',
    },
    status: {
        marginTop: '1.5rem',
        fontSize: '1.5rem',
        fontWeight: 'bold' as const,
        color: '#333',
    },
    statuswinner: {
        color: '#4CAF50',
    },
    resetButton: {
        marginTop: '1rem',
        padding: '0.75rem 2rem',
        fontSize: '1rem',
        background: '#667eea',
        color: 'white',
        border: 'none',
        borderRadius: '8px',
        cursor: 'pointer',
        fontWeight: 'bold' as const,
        transition: 'all 0.2s',
    },
    history: {
        minWidth: '200px',
    },
    historyButton: {
        padding: '0.5rem 1rem',
        marginBottom: '0.5rem',
        background: 'white',
        border: '1px solid #ddd',
        borderRadius: '4px',
        cursor: 'pointer',
        width: '100%',
        textAlign: 'left' as const,
        transition: 'all 0.2s',
    },
    historyButtonCurrent: {
        background: '#667eea',
        color: 'white',
        fontWeight: 'bold' as const,
    },
},
);

```

What You Learned

□ **Component composition** - Building complex UIs from simple components □ **State management** - Using signals to manage game state □ **Derived state** - Computing values with memo □ **Event handling** - Responding to user clicks □ **Conditional rendering** - Showing different UI based on state □ **Immutability** - Creating new state instead of mutating □ **Time travel** - Implementing undo/redo functionality

Challenges

Try extending the game:

1. **Sort moves**: Add buttons to sort history ascending/descending
2. **Highlight current square**: Show which square was just played
3. **Rewrite Board**: Use loops instead of hardcoded squares
4. **AI opponent**: Implement a computer player
5. **Larger board**: Make it work with 4x4 or 5x5
6. **Network play**: Allow two players on different devices

Next Steps

- [Build a Todo App](#) - Learn data persistence and filtering
- [Learn about Effects](#) - Handle side effects like API calls

- [Explore Routing](#) - Add multiple pages to your apps

Next: [Tutorial: Todo App](#) →

Tutorial: Build a Demo App

Build a comprehensive demo application showcasing PhilJS's core features: fine-grained reactivity, data fetching, and spring physics animations. This tutorial introduces you to the fundamental capabilities that make PhilJS unique.

What You'll Learn

- Fine-grained reactivity with signals
- Data fetching patterns
- Spring physics animations
- Component composition
- State management patterns
- Modern UI design with inline styles

What We're Building

A feature showcase app demonstrating:
- **Counter** - Fine-grained reactivity without virtual DOM
- **Data Fetcher** - Async operations with loading and error states
- **Animation Demo** - Physics-based spring animations
- Responsive card-based layout
- Modern gradient styling

Setup

```
# From the PhilJS repo root
cd examples/demo-app

# Install dependencies (if not already installed)
pnpm install

# Start the dev server
pnpm dev
```

Visit <http://localhost:5173> to see your app running.

Step 1: Create the Counter Component

The Counter demonstrates PhilJS's fine-grained reactivity. When you click a button, **only the number updates** - no virtual DOM diffing needed.

Create `src/components/Counter.tsx`:

```

import { memo, signal } from "@philjs/core";

export function Counter() {
  const count = signal(0);

  const increment = () => count.set(c => c + 1);
  const decrement = () => count.set(c => c - 1);

  return (
    <div style={{ textAlign: 'center' }}>
      <div style={{
        fontSize: '3rem',
        fontWeight: 'bold',
        color: '#667eea',
        margin: '1rem 0'
      }}>
        {count()}
      </div>
      <div style={{ display: 'flex', gap: '0.5rem', justifyContent: 'center' }}>
        <button
          onClick={decrement}
          style={{
            padding: '0.5rem 1rem',
            fontSize: '1rem',
            background: '#dc3545',
            color: 'white',
            border: 'none',
            borderRadius: '6px',
            cursor: 'pointer',
            transition: 'all 0.2s'
          }}
        >
        -
        </button>
        <button
          onClick={increment}
          style={{
            padding: '0.5rem 1rem',
            fontSize: '1rem',
            background: '#28a745',
            color: 'white',
            border: 'none',
            borderRadius: '6px',
            cursor: 'pointer',
            transition: 'all 0.2s'
          }}
        >
        +
        </button>
      </div>
      <p style={{ marginTop: '1rem', color: '#666', fontSize: '0.9rem' }}>
        Click buttons to see fine-grained reactivity
      </p>
    </div>
  );
}

```

Understanding Signals

```
const count = signal(0);
```

A **signal** is PhilJS's fundamental reactive primitive: - `signal(initialValue)` creates a signal - `count()` reads the current value - `count.set(newValue)` updates the value - `count.set(fn)` updates with a function: `count.set(c => c + 1)`

When `count` changes, **only** the `{count()}` text node updates. The buttons, styles, and rest of the DOM remain untouched.

Step 2: Create the Data Fetcher Component

The DataFetcher shows how to handle async operations with signals.

Create `src/components/DataFetcher.tsx`:

```

import { signal } from "@philjs/core";

export function DataFetcher() {
  const data = signal<any>(null);
  const loading = signal(false);
  const error = signal<string | null>(null);

  const fetchData = async () => {
    loading.set(true);
    error.set(null);

    try {
      const response = await fetch('https://api.github.com/repos/facebook/react');
      const json = await response.json();
      data.set(json);
    } catch (e) {
    }
  };
}

```

```

    error.set(e instanceof Error ? e.message : 'Failed to fetch');
} finally {
  loading.set(false);
}
};

const buttonLabel = memo(() => loading() ? 'Loading...' : 'Fetch GitHub Data');
const buttonStyle = memo(() => ({
  width: '100%',
  padding: '0.75rem',
  fontSize: '1rem',
  background: '#667eea',
  color: 'white',
  border: 'none',
  borderRadius: '6px',
  cursor: loading() ? 'not-allowed' : 'pointer',
  opacity: loading() ? 0.6 : 1,
  transition: 'all 0.2s'
}));

const errorStyle = memo(() => ({
  marginTop: '1rem',
  padding: '0.75rem',
  background: '#fee',
  color: '#c33',
  borderRadius: '6px',
  fontSize: '0.9rem',
  display: error() ? 'block' : 'none'
}));

const errorText = memo(() => error() ? `Error: ${error()}` : '');

const hasData = memo(() => !!data() && !loading());
const dataCardStyle = memo(() => ({
  marginTop: '1rem',
  fontSize: '0.85rem',
  lineHeight: '1.6',
  display: hasData() ? 'block' : 'none'
}));

const repoName = memo(() => data()?.name ?? '');
const repoStars = memo(() => data()?.stargazers_count?.toLocaleString() ?? '');
const repoForks = memo(() => data()?.forks_count?.toLocaleString() ?? '');
const repoLanguage = memo(() => data()?.language ?? '');

const showEmptyState = memo(() => !data() && !loading());
const emptyStateStyle = memo(() => ({
  marginTop: '1rem',
  color: '#666',
  fontSize: '0.9rem',
  textAlign: 'center',
  display: showEmptyState() ? 'block' : 'none'
}));


return (
  <div>
    <button
      onClick={fetchData}
      disabled={loading}
      style={buttonStyle}>
      {buttonLabel}
    </button>

    <div style={errorStyle}>
      {errorText}
    </div>

    <div style={dataCardStyle}>
      <div><strong>Repo:</strong> {repoName}</div>
      <div><strong>Stars:</strong> {repoStars}</div>
      <div><strong>Forks:</strong> {repoForks}</div>
      <div><strong>Language:</strong> {repoLanguage}</div>
    </div>

    <p style={emptyStateStyle}>
      Click to fetch data with SWR-style caching
    </p>
  </div>
);
}
}

```

Async Patterns with Signals

This component demonstrates the standard pattern for async operations:

```

const data = signal<any>(null);
const loading = signal(false);
const error = signal<string | null>(null);

```

Three signals track the async state: - **data** - The fetched data - **loading** - Whether a request is in flight - **error** - Any error that occurred
The UI automatically updates when any signal changes, creating a responsive user experience.

Step 3: Create the Animation Demo Component

The AnimationDemo showcases PhilJS's spring physics animations with `createAnimatedValue`.

Create `src/components/AnimationDemo.tsx`:

```

import { signal, createAnimatedValue, easings } from "@philjs/core";

export function AnimationDemo() {
  const position = createAnimatedValue(0, {
    easing: { stiffness: 0.1, damping: 0.7 },
  });

  const animate = () => {
    const target = position.value === 0 ? 200 : 0;
    position.set(target);
  };

  return (
    <div>
      <button
        onClick={animate}
        style={{
          width: '100%',
          padding: '0.75rem',
          fontSize: '1rem',
          background: '#764ba2',
          color: 'white',
          border: 'none',
          borderRadius: '6px',
          cursor: 'pointer',
          transition: 'all 0.2s',
          marginBottom: '1rem'
        }}
      >
        Animate with Spring Physics
      </button>

      <div style={{
        height: '100px',
        background: '#f0f0f0',
        borderRadius: '8px',
        position: 'relative',
        overflow: 'hidden'
      }}>
        <div style={{
          position: 'absolute',
          left: `${position.value}px`,
          top: '50%',
          transform: 'translateY(-50%)',
          width: '50px',
          height: '50px',
          background: 'linear-gradient(135deg, #667eea 0%, #764ba2 100%)',
          borderRadius: '50%',
          transition: 'none'
        }} />
      </div>
    </div>
  );
}

```

Spring Physics Animations

```

const position = createAnimatedValue(0, {
  easing: { stiffness: 0.1, damping: 0.7 },
});

```

`createAnimatedValue` creates a value that animates smoothly using spring physics: - **stiffness** - How quickly the spring reaches the target (lower = slower) - **damping** - How much the spring oscillates (higher = less bouncy)

When you call `position.set(target)`, the value animates naturally to the target using physics simulation, not CSS transitions.

Step 4: Create the Main App

Now compose all components into a polished demo app.

Update `src/App.tsx`:

```
import { signal, createAnimatedValue,createQuery, easings } from "@philjs/core";
import { Counter } from "./components/Counter";
import { DataFetcher } from "./components/DataFetcher";
import { AnimationDemo } from "./components/AnimationDemo";

export function App() {
  return (
    <div style={{
      background: 'white',
      borderRadius: '16px',
      padding: '2rem',
      boxShadow: '0 20px 60px rgba(0,0,0,.3)'
    }}>
      <header style={{
        textAlign: 'center',
        marginBottom: '3rem'
      }}>
        <h1 style={{
          fontSize: '3rem',
          background: 'linear-gradient(135deg, #667eea 0%, #764ba2 100%)',
          WebkitBackgroundClip: 'text',
          WebkitTextFillColor: 'transparent',
          marginBottom: '0.5rem'
        }}>
          PhilJS Framework
        </h1>
        <p style={{
          color: '#666',
          fontSize: '1.2rem'
        }}>
          Revolutionary Frontend Framework with Intelligence Built-In
        </p>
      </header>

      <section style={{
        display: 'grid',
        gap: '2rem',
        gridTemplateColumns: 'repeat(auto-fit, minmax(300px, 1fr))'
      }}>
        <FeatureCard
          title="Signals & Reactivity"
          description="Fine-grained reactivity without virtual DOM"
        >
          <Counter />
        </FeatureCard>

        <FeatureCard
          title="Data Fetching"
          description="Unified caching with SWR-style revalidation"
        >
          <DataFetcher />
        </FeatureCard>

        <FeatureCard
          title="Spring Animations"
          description="Physics-based animations with FLIP support"
        >
          <AnimationDemo />
        </FeatureCard>
      </section>

      <footer style={{
        marginTop: '3rem',
        padding: '2rem',
        background: '#f8f9fa',
        borderRadius: '8px',
        textAlign: 'center'
      }}>
        <h3 style={{ marginBottom: '1rem', color: '#667eea' }}>
          Novel Features Demonstrated
        </h3>
        <ul style={{
          listStyle: 'none',
          display: 'grid',
          gridTemplateColumns: 'repeat(auto-fit, minmax(200px, 1fr))',
          gap: '1rem',
          textAlign: 'left'
        }}>
          <li> Performance Budgets</li>
          <li> Cost Tracking</li>
          <li> Usage Analytics</li>
          <li> Automatic Regression Detection</li>
          <li> Dead Code Detection</li>
        </ul>
      </footer>
    </div>
  );
}
```

```

        <li>■ Spring Physics</li>
        <li>■ Resumability</li>
        <li>■ Islands Architecture</li>
    </ul>
</footer>
</div>
);
}

function FeatureCard(props: {
    title: string;
    description: string;
    children: any;
}) {
    return (
        <div style={{
            padding: '1.5rem',
            border: '2px solid #e9ecef',
            borderRadius: '12px',
            transition: 'all 0.3s ease'
        }}>
            <h3 style={{
                color: '#667eea',
                marginBottom: '0.5rem',
                fontSize: '1.3rem'
            }}>
                {props.title}
            </h3>
            <p style={{
                color: '#666',
                marginBottom: '1rem',
                fontSize: '0.9rem'
            }}>
                {props.description}
            </p>
            <div style={{
                padding: '1rem',
                background: '#f8f9fa',
                borderRadius: '8px'
            }}>
                {props.children}
            </div>
        </div>
    );
}

```

Update src/main.tsx:

```

import { render } from '@philjs/core';
import { App } from './App';
import './index.css';

render(<App />, document.getElementById('app')!);

```

Understanding the Code

Component Composition

```

<FeatureCard
    title="Signals & Reactivity"
    description="Fine-grained reactivity without virtual DOM"
>
    <Counter />
</FeatureCard>

```

The FeatureCard component wraps each feature with consistent styling. This demonstrates: - **Props for configuration** - title and description - **Children for content** - <Counter /> component - **Reusable UI patterns** - Same card style for all features

Responsive Grid Layout

```
gridTemplateColumns: 'repeat(auto-fit, minmax(300px, 1fr))'
```

This CSS Grid pattern creates a responsive layout that: - Shows 3 columns on wide screens - Stacks to 1 column on narrow screens - Maintains minimum card width of 300px - Distributes space evenly

Modern Styling

```

background: 'linear-gradient(135deg, #667eea 0%, #764ba2 100%)',
WebkitBackgroundClip: 'text',
WebkitTextFillColor: 'transparent',

```

The gradient text effect demonstrates modern CSS features: - Linear gradient background - Clip to text shape - Transparent fill to show gradient

What You Learned

□ **Signals** - Fine-grained reactive state management □ **Async patterns** - Loading, error, and data states □ **Animations** - Spring physics for natural motion □ **Composition** - Building UIs from components □ **Styling** - Modern inline styles with TypeScript □ **Layout** - Responsive grid patterns

Challenges

Extend the demo app:

1. **Add persistence** - Save counter value to localStorage
2. **Error retry** - Add retry button when data fetch fails
3. **Multiple animations** - Add rotation and scale animations
4. **Dark mode** - Toggle between light and dark themes
5. **More demos** - Add form handling, routing, or effects
6. **Custom hooks** - Extract reusable logic
7. **Real API** - Fetch from your own backend
8. **Keyboard shortcuts** - Add hotkeys for actions

Next Steps

- [Tutorial: Todo App](#) - Build a complete todo application
- [Tutorial: Storefront](#) - Advanced SSR and islands
- [Learn about Signals](#) - Deep dive into reactivity
- [Animations Guide](#) - More animation patterns

Next: [Tutorial: Todo App](#) →

Tutorial: Build a Storefront with SSR

Build a production-ready e-commerce storefront with server-side rendering, streaming, islands architecture, AI integration, and PWA features. This advanced tutorial demonstrates PhilJS's full-stack capabilities.

What You'll Learn

- Server-side rendering (SSR) with streaming
- Islands architecture for selective hydration
- Loaders and actions for data management
- View Transitions API integration
- AI-powered features
- Progressive Web App (PWA) setup
- Performance budgets and monitoring
- Security best practices (CSP, signed cookies)
- Service workers and offline support

What We're Building

A full-featured e-commerce storefront with:
- **SSR streaming** - Instant page loads with progressive rendering
- **Product catalog** - Dynamic routes with loaders
- **Shopping cart**
- Optimistic UI updates with actions
- **AI summaries** - AI-generated product descriptions
- **Islands** - Hydrate interactive components on demand
- **PWA** - Offline support and installable
- **Performance monitoring** - Core Web Vitals tracking
- **Production ready** - Security, caching, and optimization

Architecture Overview

The storefront uses a modern architecture:

```
src/  
  entry-client.ts      # Client runtime with islands  
  sw.ts                # Service worker source  
  routes/  
    index.tsx          # Home page with loader  
    products/[id].tsx  # Product page with loader + action  
  server/  
    entry-server.ts    # SSR streaming renderer  
    router.ts          # File-based routing  
    mock-db.ts         # Mock database  
  ai/  
    summarize.ts       # AI integration  
public/  
  manifest.json        # PWA manifest  
server/  
  dev.ts              # Vite dev server  
  prod.ts             # Production server  
  ai.ts               # AI endpoint (optional)
```

Setup

```

# From the PhilJS repo root
cd examples/storefront

# Install dependencies (if not already installed)
pnpm install

# Build the packages first
cd ../..
pnpm build

# Return to storefront
cd examples/storefront

# Start the dev server
pnpm dev

```

Visit <http://localhost:3000> to see your storefront.

Step 1: Understanding SSR with Loaders

PhilJS uses **loaders** to fetch data on the server before rendering.

Home Page with Loader

File: src/routes/index.tsx

```

import { defineLoader } from "@philjs/ssr";
import { html } from "../server/template";

// Loader runs on the server during SSR
export const loader = defineLoader(async ({ db }) => {
  const featured = await db.product.all();
  return { featured };
});

// Component receives data from Loader
export default function Home({ data }: { data: { featured: Array<Product> } }) {
  return html`
    <main>
      <header>
        <h1>PhilJS Storefront</h1>
        <p>Fast by default, HTML first, batteries included.</p>
        <p>Streaming SSR with islands and AI assisted experiences.</p>
      </header>

      <section>
        <h2>Featured Products</h2>
        <div class="product-grid">
          ${data.featured.map((product) =>
            html`<article class="product-card">
              <div>
                <h3>${product.title}</h3>
                <p>${product.price.toFixed(2)}</p>
              </div>
              <a href="/products/${product.id}" data-phil-prefetch>
                View details
              </a>
            </article>
          `)
        )
      </div>
    </section>
  </main>
`;
}

```

How Loaders Work

1. **Server receives request** - User navigates to /
2. **Loader executes** - loader() function runs on server
3. **Data fetched** - Database queries complete
4. **HTML rendered** - Component renders with data
5. **Stream sent** - HTML streams to browser progressively

The page is **fully rendered HTML** - no loading spinners needed!

Step 2: Dynamic Routes with Actions

Product pages use **dynamic routes** and **actions** for mutations.

Product Page

File: src/routes/products/[id].tsx

```

import { defineLoader, defineAction } from "@philjs/ssr";
import { signal } from "@philjs/core";
import { html } from "../../server/template";
import { Island } from "@philjs/islands";

// Loader fetches product data
export const loader = defineLoader(async ({ params, db }) => {
  const product = await db.product.get(params.id);
  if (!product) {
    return { product: null, inCart: false };
  }

  const cart = await db.cart.get();
  const inCart = cart.items.some(item => item.productId === params.id);

  return { product, inCart };
});

// Action handles form submissions
export const action = defineAction(async ({ request, params, db }) => {
  const formData = await request.formData();
  const intent = formData.get('intent');

  if (intent === 'add-to-cart') {
    await db.cart.add(params.id);
    return { success: true, message: 'Added to cart' };
  }

  if (intent === 'remove-from-cart') {
    await db.cart.remove(params.id);
    return { success: true, message: 'Removed from cart' };
  }

  return { success: false, message: 'Unknown action' };
});

export default function Product({ data }: { data: { product: Product; inCart: boolean } }) {
  if (!data.product) {
    return html`<h1>Product not found</h1>`;
  }

  const { product, inCart } = data;

  return html`
    <article class="product-detail">
      <div class="product-image">
        
      </div>

      <div class="product-info">
        <h1>${product.title}</h1>
        <p class="price">${product.price.toFixed(2)}</p>
        <p class="description">${product.description}</p>
      </div>

      <!-- Island: Hydrate only this interactive component -->
      <${Island} name="AddToCart" props={{ productId: product.id, inCart }}>
        <add-to-cart-island data-product-id="${product.id}" data-in-cart="${inCart}">
          <form method="post">
            <input type="hidden" name="intent" value="${inCart ? 'remove-from-cart' : 'add-to-cart'}" />
            <button type="submit">
              ${inCart ? 'Remove from Cart' : 'Add to Cart'}
            </button>
          </form>
        </add-to-cart-island>
      </${Island}>
    </div>
  `;
}

```

Understanding Actions

Actions handle **mutations** (POST, PUT, DELETE): 1. User submits form 2. Action runs on server 3. Database updated 4. Page re-rendered with new data 5. Client receives updated HTML

This is similar to traditional server-rendered forms, but with **optimistic UI** support on the client.

Step 3: Islands Architecture

Islands are **interactive components** that hydrate on demand. The rest of the page stays static HTML.

Client Entry Point

File: src/entry-client.ts

```
import { hydrateIslands } from '@philjs/islands';
import { AddToCart } from './islands/AddToCart';
import { CartSummary } from './islands/CartSummary';

// Register island components
const islands = {
  AddToCart,
  CartSummary,
};

// Hydrate islands when they become visible
hydrateIslands(islands, {
  strategy: 'visible', // Options: 'idle', 'visible', 'eager'
});

// View Transitions API support
if ('startViewTransition' in document) {
  document.addEventListener('click', (e) => {
    const link = (e.target as HTMLElement).closest('a[href]');
    if (link && link.hasAttribute('data-phil-transition')) {
      e.preventDefault();
      const href = link.getAttribute('href')!;

      // @ts-ignore
      document.startViewTransition(() => {
        window.location.href = href;
      });
    }
  });
}
```

Creating an Island Component

File: src/islands/AddToCart.tsx

```

import { signal } from '@philjs/core';

interface AddToCartProps {
  productId: string;
  inCart: boolean;
}

export function AddToCart({ productId, inCart: initialInCart }: AddToCartProps) {
  const inCart = signal(initialInCart);
  const loading = signal(false);

  const handleSubmit = async (e: Event) => {
    e.preventDefault();
    loading.set(true);

    const intent = inCart() ? 'remove-from-cart' : 'add-to-cart';

    // Optimistic update
    inCart.set(!inCart());

    try {
      const formData = new FormData();
      formData.append('intent', intent);

      const response = await fetch(`~/products/${productId}`, {
        method: 'POST',
        body: formData,
      });

      if (!response.ok) {
        // Rollback on error
        inCart.set(!inCart());
      }
    } catch (error) {
      // Rollback on error
      inCart.set(!inCart());
      console.error('Failed to update cart:', error);
    } finally {
      loading.set(false);
    }
  };
}

return (
  <form onSubmit={handleSubmit}>
    <button
      type="submit"
      disabled={loading()}
      style={{
        padding: '1rem 2rem',
        fontSize: '1.125rem',
        background: inCart() ? '#dc3545' : '#28a745',
        color: 'white',
        border: 'none',
        borderRadius: '8px',
        cursor: loading() ? 'not-allowed' : 'pointer',
        opacity: loading() ? 0.6 : 1,
        transition: 'all 0.2s',
      }}
    >
      {loading() ? 'Updating...' : inCart() ? 'Remove from Cart' : 'Add to Cart'}
    </button>
  </form>
);
}

```

Why Islands?

Traditional hydration loads **all** JavaScript for the entire page. Islands only load JavaScript for **interactive components**.

Benefits: - ☐ **Faster initial load** - Less JavaScript to parse - ☐ **Lower bandwidth** - Send only what's needed - ☐ **Progressive enhancement** - Works without JS (forms still submit) - ☐ **Better battery** - Less code execution

Step 4: AI Integration

Integrate AI to generate product summaries.

AI Adapter

File: src/ai/summarize.ts

```

import { createAIAdapter } from '@philjs/ai';

// Create adapter for AI endpoint
const ai = createAIAdapter({
  endpoint: 'http://localhost:8787/api/ai',
  fallback: 'AI service unavailable',
});

export async function summarizeProduct(product: Product): Promise<string> {
  const prompt = `Summarize this product in 2-3 sentences for an e-commerce site:

Product: ${product.title}
Price: ${product.price}
Description: ${product.description}

Write a compelling summary that highlights key features.`;

  try {
    const summary = await ai.complete(prompt, {
      temperature: 0.7,
      maxTokens: 150,
    });
    return summary;
  } catch (error) {
    console.error('AI summarization failed:', error);
    return product.description;
  }
}

```

Using AI in a Loader

```

import { summarizeProduct } from '../../../../../ai/summarize';

export const loader = defineLoader(async ({ params, db }) => {
  const product = await db.product.get(params.id);

  // Generate AI summary (cached on subsequent visits)
  const aiSummary = await summarizeProduct(product);

  return { product, aiSummary };
});

```

Step 5: Progressive Web App (PWA)

Make your storefront installable and work offline.

Service Worker

File: src/sw.ts (compiled to public/sw.js via scripts/build-sw.ts)

```

const CACHE_NAME = 'philjs-storefront-v0.1.0';
const STATIC_CACHE = [
  '/',
  '/manifest.json',
  '/icon-192.png',
  '/icon-512.png',
];
};

// Install: Cache static assets
self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME).then((cache) => {
      return cache.addAll(STATIC_CACHE);
    })
  );
});

// Fetch: Network first, fallback to cache
self.addEventListener('fetch', (event) => {
  if (event.request.method !== 'GET') return;

  event.respondWith(
    fetch(event.request)
      .then((response) => {
        // Clone and cache the response
        const responseClone = response.clone();
        caches.open(CACHE_NAME).then((cache) => {
          cache.put(event.request, responseClone);
        });
        return response;
      })
      .catch(() => {
        // Network failed, try cache
        return caches.match(event.request);
      })
  );
});
});

```

PWA Manifest

File: public/manifest.json

```
{
  "name": "PhilJS Storefront",
  "short_name": "Storefront",
  "description": "Fast e-commerce powered by PhilJS",
  "start_url": "/",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#667eea",
  "icons": [
    {
      "src": "/icon-192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "/icon-512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ]
}
```

Step 6: Performance Monitoring

Track Core Web Vitals and send to your analytics.

RUM (Real User Monitoring)

File: src/entry-client.ts (add this):

```

import { getCLS, getFID, getLCP } from 'web-vitals';

// Send metrics to your backend
function sendToAnalytics(metric: any) {
  const body = JSON.stringify(metric);

  // Use sendBeacon if available (doesn't block page unload)
  if (navigator.sendBeacon) {
    navigator.sendBeacon('/api/metrics', body);
  } else {
    fetch('/api/metrics', { method: 'POST', body, keepalive: true });
  }
}

// Track Core Web Vitals
getCLS(sendToAnalytics);
getFID(sendToAnalytics);
getLCP(sendToAnalytics);

```

Performance Budgets

File: scripts/check-budgets.ts

```

import { readFileSync } from 'node:fs';
import { gzipSync } from 'node:zlib';

const budgets = {
  'dist/client/assets/index.js': 70 * 1024, // 70 KB
  'dist/client/assets/style.css': 15 * 1024, // 15 KB
};

let failed = false;

for (const [file, budget] of Object.entries(budgets)) {
  try {
    const content = readFileSync(file);
    const gzipped = gzipSync(content);
    const size = gzipped.length;

    if (size > budget) {
      console.error(` ${file}: ${size} bytes (budget: ${budget})`);
      failed = true;
    } else {
      console.log(` ${file}: ${size} bytes (budget: ${budget})`);
    }
  } catch (error) {
    console.error(` ${file}: not found`);
    failed = true;
  }
}

if (failed) {
  process.exit(1);
}

```

Run with:

```

pnpm build
pnpm tsx scripts/check-budgets.ts

```

Step 7: Security

Content Security Policy (CSP)

```

import { buildCSP } from '@philjs/ssr';

const csp = buildCSP({
  directives: {
    'default-src': ["'self'"],
    'script-src': ["'self'", "'unsafe-inline'"],
    'style-src': ["'self'", "'unsafe-inline'"],
    'img-src': ["'self'", 'data:', 'https:'],
    'connect-src': ["'self'", 'https://api.github.com'],
  },
});

// In server response:
headers.set('Content-Security-Policy', csp);

```

Signed Cookies

```
import { createCookie } from '@philjs/ssr';

const sessionCookie = createCookie('session', {
  secrets: [process.env.COOKIE_SECRET],
  httpOnly: true,
  secure: process.env.NODE_ENV === 'production',
  sameSite: 'lax',
  maxAge: 60 * 60 * 24 * 7, // 7 days
});

// Set cookie
await sessionCookie.serialize({ userId: '123' });

// Read cookie
const session = await sessionCookie.parse(request.headers.get('Cookie'));
```

Production Deployment

Build for Production

```
pnpm build
```

This creates: - dist/client/ - Static assets (JS, CSS, images) - dist/server/ - SSR entry point

Production Server

File: server/prod.ts

```

import http from "node:http";
import { createReadStream } from "node:fs";
import { stat } from "node:fs/promises";
import { extname, join, resolve } from "node:path";
import { Readable } from "node:stream";
import { writeEarlyHints } from "@philjs/ssr";

const clientDir = resolve(process.cwd(), "dist/client");
const port = Number(process.env.PORT ?? 3000);
const serverEntry = await import("../dist/server/entry-server.js");

const mimeMap = new Map(
  Object.entries({
    ".html": "text/html; charset=utf-8",
    ".js": "text/javascript; charset=utf-8",
    ".css": "text/css; charset=utf-8",
    ".json": "application/json; charset=utf-8",
    ".ico": "image/x-icon",
    ".png": "image/png",
    ".svg": "image/svg+xml"
  })
);

const server = http.createServer(async (req, res) => {
  const url = new URL(req.url ?? "/", `http://${req.headers.host}`);
  const filePath = join(clientDir, url.pathname.replace(/^\//, ""));

  if (await tryServeFile(filePath, res)) {
    return;
  }

  const result = await serverEntry.render(req);
  if (Array.isArray(result?.earlyHints)) {
    writeEarlyHints(res, result.earlyHints);
  }

  if (!result) {
    res.writeHead(404, { "content-type": "text/plain" });
    res.end("Not Found");
    return;
  }

  const { status = 200, headers = {}, body } = result;
  res.writeHead(status, headers);

  if (!body) {
    res.end();
    return;
  }

  if (typeof body === "string") {
    res.end(body);
    return;
  }

  if (typeof body.getReader === "function") {
    Readable.fromWeb(body).pipe(res);
    return;
  }

  if (body instanceof Readable) {
    body.pipe(res);
    return;
  }

  res.end();
});

server.listen(port, () => {
  console.log(`PhilJS storefront (prod) listening on http://localhost:${port}`);
});

async function tryServeFile(path: string, res: http.ServerResponse): Promise<boolean> {
  try {
    const stats = await stat(path);
    if (!stats.isFile()) {
      return false;
    }

    const type = mimeMap.get(extname(path)) ?? "application/octet-stream";
    res.writeHead(200, { "content-type": type, "content-length": stats.size });
    createReadStream(path).pipe(res);
    return true;
  } catch (error) {
    return false;
  }
}

```

Deployment Platforms

Vercel:

```
vercel deploy
```

Netlify:

```
netlify deploy --prod
```

Cloudflare Workers:

```
wrangler deploy
```

Docker:

```
FROM node:24-alpine
WORKDIR /app
COPY package.json pnpm-lock.yaml ./ 
RUN npm install -g pnpm && pnpm install --frozen-lockfile
COPY . .
RUN pnpm build
CMD ["pnpm", "serve"]
EXPOSE 3000
```

What You Learned

□ **SSR streaming** - Progressive rendering for instant loads □ **Loaders** - Server-side data fetching □ **Actions** - Form handling and mutations □ **Islands** - Selective hydration for performance □ **AI integration** - Enhancing UX with AI □ **PWA** - Offline support and installability □ **Performance** - Monitoring and budgets □ **Security** - CSP, signed cookies, best practices

Challenges

Extend the storefront:

1. **Search** - Add product search with autocomplete
2. **Filters** - Filter products by category, price, rating
3. **Checkout** - Implement full checkout flow
4. **User accounts** - Sign up, login, order history
5. **Reviews** - Let users rate and review products
6. **Recommendations** - AI-powered product suggestions
7. **Real-time updates** - WebSocket for stock updates
8. **Multi-language** - i18n support
9. **Admin panel** - Manage products and orders
10. **Payment integration** - Stripe, PayPal, etc.

Next Steps

- [Advanced: SSR](#) - Deep dive into server-side rendering
- [Advanced: Islands](#) - Islands architecture patterns
- [Data Fetching](#) - Loaders, actions, and caching
- [Performance](#) - Optimization techniques

Next: [Tutorial: Static Blog](#) →

Tutorial: Build a Blog with Static Site Generation

Learn how to build a fully static blog with PhilJS that loads instantly and ranks well in search engines. This tutorial covers static site generation (SSG), markdown content, SEO, and deployment.

What You'll Learn

- File-based routing for blog posts
- Markdown/MDX content with frontmatter
- Static page generation at build time
- SEO optimization with meta tags
- RSS feed generation
- Sitemap creation
- Incremental Static Regeneration (ISR)
- Deploying to production

What We're Building

A complete blog with:
- Homepage with post listings
- Individual post pages
- Category filtering
- Tag support
- SEO-optimized meta tags
- RSS feed
- Sitemap
- Fast page loads (< 1s)

Setup

```
pnpm create philjs my-blog
cd my-blog
pnpm install

# Install markdown dependencies
pnpm add gray-matter remark remark-html
pnpm add -D @types/node
```

Step 1: Project Structure

Create this directory structure:

```
my-blog/
├── content/
│   └── posts/
│       ├── first-post.md
│       ├── second-post.md
│       └── getting-started.md
└── src/
    ├── routes/
    │   ├── index.tsx      # Homepage
    │   └── blog/
    │       ├── index.tsx    # Blog listing
    │       └── [slug].tsx    # Post page
    ├── tags/
    │   └── [tag].tsx      # Tag page
    └── components/
        ├── PostCard.tsx
        ├── PostLayout.tsx
        └── SEO.tsx
    └── lib/
        ├── posts.ts        # Post utilities
        └── seo.ts          # SEO utilities
└── philjs.config.ts
```

Step 2: Create Sample Blog Posts

Create content/posts/first-post.md:

```
---
title: "Getting Started with PhilJS"
date: "2024-01-15"
author: "John Doe"
excerpt: "Learn how to build Lightning-fast web apps with PhilJS"
tags: ["philjs", "tutorial", "getting-started"]
image: "/images/first-post.jpg"
---

# Getting Started with PhilJS

PhilJS is a revolutionary framework that combines the best ideas from React, Solid, and Qwik.

## Why PhilJS?

Here are the key benefits:

1. **Zero hydration** - Apps are interactive immediately
2. **Fine-grained reactivity** - Only what changes updates
3. **Built-in SSG** - Static generation out of the box

## Code Example

```typescript
import { signal } from '@philjs/core';

const count = signal(0);

<button onClick={() => count.set(c => c + 1)}>
 Clicked {count()} times
</button>
```

This is just the beginning. PhilJS makes building fast web apps effortless.

```
Create `content/posts/second-post.md`:

```markdown
---
title: "Advanced Patterns in PhilJS"
date: "2024-01-20"
author: "Jane Smith"
excerpt: "Deep dive into advanced PhilJS patterns and best practices"
tags: ["philjs", "advanced", "patterns"]
image: "/images/second-post.jpg"
---

# Advanced Patterns in PhilJS

Once you've mastered the basics, these patterns will help you build production-ready applications.

## Component Composition

Breaking down complex UIs into reusable components is key to maintainable code.

## State Management

Use signals for local state and context for global state. Keep it simple!

```

Step 3: Create Post Utilities

Create `src/lib/posts.ts`:

```
import fs from 'fs';
import path from 'path';
import matter from 'gray-matter';
import { remark } from 'remark';
import html from 'remark-html';

export interface Post {
  slug: string;
  title: string;
  date: string;
  author: string;
  excerpt: string;
  tags: string[];
  image?: string;
  content: string;
}

const postsDirectory = path.join(process.cwd(), 'content/posts');

export async function getAllPosts(): Promise<Post[]> {
  const fileNames = fs.readdirSync(postsDirectory);

  const allPosts = await Promise.all(
    fileNames
      .filter(fileName => fileName.endsWith('.md'))
      .map(async fileName => {
        const slug = fileName.replace(/\.\w+/, '');
        return getPostBySlug(slug);
      })
  );
}

// Sort posts by date (newest first)
return allPosts.sort((a, b) => {
  return new Date(b.date).getTime() - new Date(a.date).getTime();
});

export async function getPostBySlug(slug: string): Promise<Post> {
  const fullPath = path.join(postsDirectory, `${slug}.md`);
  const fileContents = fs.readFileSync(fullPath, 'utf8');

  // Parse frontmatter
  const { data, content } = matter(fileContents);

  // Convert markdown to HTML
  const processedContent = await remark()
    .use(html)
    .process(content);

  const contentHtml = processedContent.toString();

  return {
    slug,
    title: data.title,
    date: data.date,
    author: data.author,
    excerpt: data.excerpt,
    tags: data.tags || [],
  };
}
```

```
        image: data.image,
        content: contentHtml,
    );
}

export async function getAllTags(): Promise<string[]> {
    const posts = await getAllPosts();
    const tagSet = new Set<string>();

    posts.forEach(post => {
        post.tags.forEach(tag => tagSet.add(tag));
    });

    return Array.from(tagSet).sort();
}

export async function getPostsByTag(tag: string): Promise<Post[]> {
    const posts = await getAllPosts();
    return posts.filter(post =>
        post.tags.some(t => t.toLowerCase() === tag.toLowerCase())
    );
}

export function formatDate(dateString: string): string {
    const date = new Date(dateString);
    return date.toLocaleDateString('en-US', {
        year: 'numeric',
        month: 'long',
        day: 'numeric',
    });
}
```

Step 4: Create SEO Component

Create `src/components/SEO.tsx`:

```

import { Head } from '@philjs/core';

interface SEOProps {
  title: string;
  description: string;
  image?: string;
  url?: string;
  type?: 'website' | 'article';
  publishedTime?: string;
  author?: string;
  tags?: string[];
}

export function SEO({
  title,
  description,
  image = '/images/og-default.jpg',
  url = 'https://myblog.com',
  type = 'website',
  publishedTime,
  author,
  tags = [],
}: SEOProps) {
  const fullTitle = `${title} | My PhilJS Blog`;

  return (
    <Head>
      {/* Primary Meta Tags */}
      <title>{fullTitle}</title>
      <meta name="title" content={fullTitle} />
      <meta name="description" content={description} />

      {/* Open Graph / Facebook */}
      <meta property="og:type" content={type} />
      <meta property="og:url" content={url} />
      <meta property="og:title" content={fullTitle} />
      <meta property="og:description" content={description} />
      <meta property="og:image" content={image} />

      {/* Twitter */}
      <meta property="twitter:card" content="summary_large_image" />
      <meta property="twitter:url" content={url} />
      <meta property="twitter:title" content={fullTitle} />
      <meta property="twitter:description" content={description} />
      <meta property="twitter:image" content={image} />

      {/* Article specific */}
      {type === 'article' && publishedTime && (
        <meta property="article:published_time" content={publishedTime} />
      )}
      {author && <meta property="article:author" content={author} />}
      {tags.map(tag => (
        <meta key={tag} property="article:tag" content={tag} />
      ))}
    </Head>
  );
}

```

Step 5: Create Post Card Component

Create src/components/PostCard.tsx:

```

import type { Post } from '../lib/posts';
import { formatDate } from '../lib/posts';

interface PostCardProps {
  post: Post;
}

export function PostCard({ post }: PostCardProps) {
  return (
    <article style={styles.card}>
      {post.image && (
        <a href={`/blog/${post.slug}`} data-router-link>
          <img
            src={post.image}
            alt={post.title}
            style={styles.image}
          />
        </a>
      )}
      <div style={styles.content}>
        ...
      </div>
    </article>
  );
}

```

```

<div style={styles.meta}>
  <time style={styles.date}>{formatDate(post.date)}</time>
  <span style={styles.author}>by {post.author}</span>
</div>

<a href={`/blog/${post.slug}`} data-router-link style={styles.titleLink}>
  <h2 style={styles.title}>{post.title}</h2>
</a>

<p style={styles.excerpt}>{post.excerpt}</p>

<div style={styles.tags}>
  {post.tags.map(tag => (
    <a
      key={tag}
      href={`/tags/${tag}`}
      data-router-link
      style={styles.tag}
    >
      #{tag}
    </a>
  )));
</div>

<a href={`/blog/${post.slug}`} data-router-link style={styles.readMore}>
  Read more →
</a>
</div>
</article>
);
}

The exported `config` object tells PhilJS to pre-render each post at build time. Instead of relying on framework-specific hooks, the router passes `params` to the component via props.

const styles = {
  card: {
    background: 'white',
    borderRadius: '12px',
    overflow: 'hidden',
    boxShadow: '0 2px 8px rgba(0,0,0,0.1)',
    transition: 'transform 0.2s, box-shadow 0.2s',
  },
  image: {
    width: '100%',
    height: '240px',
    objectFit: 'cover' as const,
  },
  content: {
    padding: '1.5rem',
  },
  meta: {
    display: 'flex',
    gap: '1rem',
    marginBottom: '0.75rem',
    fontSize: '0.875rem',
    color: '#666',
  },
  date: {},
  author: {},
  titleLink: {
    textDecoration: 'none',
    color: 'inherit',
  },
  title: {
    margin: '0 0 0.75rem',
    fontSize: '1.5rem',
    color: '#333',
    transition: 'color 0.2s',
  },
  excerpt: {
    margin: '0 0 1rem',
    color: '#666',
    lineHeight: 1.6,
  },
  tags: {
    display: 'flex',
    gap: '0.5rem',
    flexWrap: 'wrap' as const,
    marginBottom: '1rem',
  },
  tag: {
    padding: '0.25rem 0.75rem',
    background: '#f0f0f0',
    borderRadius: '20px',
    fontSize: '0.875rem',
    color: '#667eea',
    textDecoration: 'none',
    transition: 'background 0.2s',
  },
}

```

```
    readMore: {
      color: '#667eea',
      textDecoration: 'none',
      fontWeight: 'bold' as const,
    },
};
```

Step 6: Create Blog Listing Page

Create `src/routes/blog/index.tsx`:

```

import { PostCard } from '../../../../../components/PostCard';
import { SEO } from '../../../../../components/SEO';
import { getAllPosts } from '../../../../../lib/posts';

export const config = {
  mode: 'ssg' as const,
};

type BlogPageProps = {
  url: URL;
  navigate: (to: string) => Promise<void>;
};

export default async function BlogPage(_props: BlogPageProps) {
  const posts = await getAllPosts();

  return (
    <>
      <SEO
        title="Blog"
        description="Articles about web development, PhilJS, and modern JavaScript"
        url="https://myblog.com/blog"
      />

      <div style={styles.container}>
        <header style={styles.header}>
          <h1 style={styles.title}>Blog</h1>
          <p style={styles.subtitle}>
            Thoughts on web development and PhilJS
          </p>
        </header>

        <div style={styles.grid}>
          {posts.map(post => (
            <PostCard key={post.slug} post={post} />
          )));
        </div>
      </div>
    );
}

const styles = {
  container: {
    maxWidth: '1200px',
    margin: '0 auto',
    padding: '2rem',
  },
  header: {
    textAlign: 'center' as const,
    marginBottom: '3rem',
  },
  title: {
    fontSize: '3rem',
    margin: '0 0 1rem',
    color: '#333',
  },
  subtitle: {
    fontSize: '1.25rem',
    color: '#666',
    margin: '0',
  },
  grid: {
    display: 'grid',
    gridTemplateColumns: 'repeat(auto-fill, minmax(350px, 1fr))',
    gap: '1rem',
  },
  empty: {
    textAlign: 'center' as const,
    padding: '4rem',
    color: '#999',
  },
};
}

```

The config export marks the listing route for static generation—PhilJS renders it during the build so readers get an instant HTML response.

Step 7: Create Individual Post Page

Create `src/routes/blog/[slug].tsx`:

```

import { SEO } from '../../../../../components/SEO';
import { PostCard, getPostBySlug, formatData } from '../../../../../lib/posts';

```

```

import { getStaticPaths, getStaticProps, formatDate } from 'next';
import { URL } from 'node:url';

export const config = {
  mode: 'ssg' as const,
  async getStaticPaths() {
    const posts = await getAllPosts();
    return posts.map(post => `/blog/${post.slug}`);
  },
};

interface PostPageProps {
  params: { slug: string };
  url: URL;
  navigate: (to: string) => Promise<void>;
}

export default async function PostPage({ params }: PostPageProps) {
  const post = await getPostBySlug(params.slug);

  return (
    <>
    <SEO
      title={post.title}
      description={post.excerpt}
      image={post.image}
      url={`https://myblog.com/blog/${post.slug}`}
      type="article"
      publishedTime={post.date}
      author={post.author}
      tags={post.tags}
    />

    <article style={styles.article}>
      <header style={styles.header}>
        <a href="/blog" data-router-link style={styles.back}>
          ← Back to blog
        </a>

        <h1 style={styles.title}>{post.title}</h1>

        <div style={styles.meta}>
          <time style={styles.date}>{formatDate(post.date)}</time>
          <span style={styles.author}>by {post.author}</span>
        </div>

        <div style={styles.tags}>
          {post.tags.map(tag => (
            <a
              key={tag}
              href={`/tags/${tag}`}
              data-router-link
              style={styles.tag}
            >
              #{tag}
            </a>
          )));
        </div>
      </header>

      {post.image && (
        <img
          src={post.image}
          alt={post.title}
          style={styles.featuredImage}
        />
      )}

      <div
        style={styles.content}
        dangerouslySetInnerHTML={{ __html: post.content }}
      />

      <footer style={styles.footer}>
        <a href="/blog" data-router-link style={styles.backButton}>
          ← Back to all posts
        </a>
      </footer>
    </article>
  );
}

const styles = {
  article: {
    maxWidth: '800px',
    margin: '0 auto',
    padding: '2rem',
  },
  header: {
    ...
  }
};

```

```

        marginBottom: '2rem',
    },
    back: {
        display: 'inline-block',
        marginBottom: '1rem',
        color: '#667eea',
        textDecoration: 'none',
    },
    title: {
        fontSize: '3rem',
        margin: '0 0 1rem',
        color: '#333',
        lineHeight: 1.2,
    },
    meta: {
        display: 'flex',
        gap: '1rem',
        marginBottom: '1rem',
        color: '#666',
    },
    date: {},
    author: {},
    tags: {
        display: 'flex',
        gap: '0.5rem',
        flexWrap: 'wrap' as const,
    },
    tag: {
        padding: '0.25rem 0.75rem',
        background: '#f0f0f0',
        borderRadius: '20px',
        fontSize: '0.875rem',
        color: '#667eea',
        textDecoration: 'none',
    },
    featuredImage: {
        width: '100%',
        height: 'auto',
        borderRadius: '12px',
        marginBottom: '2rem',
    },
    content: {
        fontSize: '1.125rem',
        lineHeight: 1.8,
        color: '#333',
    },
    footer: {
        marginTop: '3rem',
        paddingTop: '2rem',
        borderTop: '1px solid #eee',
    },
    backButton: {
        display: 'inline-block',
        padding: '0.75rem 1.5rem',
        background: '#667eea',
        color: 'white',
        borderRadius: '8px',
        textDecoration: 'none',
        fontWeight: 'bold' as const,
    },
},
);

```

Step 8: Create Tag Page

Create `src/routes/tags/[tag].tsx`:

```

import { SEO } from '../../../../../components/SEO';
import { getAllTags, getPostsByTag } from '../../../../../lib/posts';

export const config = {
    mode: 'ssg' as const,
    async getStaticPaths() {
        const tags = await getAllTags();
        return tags.map(tag => `/tags/${tag}`);
    },
};

interface TagPageProps {
    params: { tag: string };
    url: URL;
    navigate: (to: string) => Promise<void>;
}

export default async function TagPage({ params }: TagPageProps) {
    const posts = await getPostsByTag(params.tag);

    return (
        <>

```

```

<SEO
  title={`Posts tagged "${params.tag}"`}
  description={`All PhilJS blog posts tagged with ${params.tag}`}
  url={`https://myblog.com/tags/${params.tag}`}
/>

<div style={styles.container}>
  <header style={styles.header}>
    <a href="/blog" data-router-link style={styles.back}>
      ← Back to all posts
    </a>

    <h1 style={styles.title}>#${params.tag}</h1>
    <p style={styles.subtitle}>
      {posts.length} {posts.length === 1 ? 'post' : 'posts'} with this tag
    </p>
  </header>

  <div style={styles.list}>
    {posts.map(post => (
      <article key={post.slug} style={styles.post}>
        <a href={`/blog/${post.slug}`} data-router-link style={styles.postTitle}>
          {post.title}
        </a>
        <p style={styles.excerpt}>{post.excerpt}</p>
      </article>
    )));
  </div>

  {posts.length === 0 && (
    <div style={styles.empty}>
      <p>No posts yet for this tag.</p>
    </div>
  )}
</div>
</>
);

};

const styles = {
  container: {
    maxWidth: '900px',
    margin: '0 auto',
    padding: '2rem',
  },
  header: {
    marginBottom: '2rem',
  },
  back: {
    display: 'inline-block',
    marginBottom: '1rem',
    color: "#6667ea",
    textDecoration: 'none',
  },
  title: {
    fontSize: '2.5rem',
    margin: '0 0 0.5rem',
    color: '#333',
  },
  subtitle: {
    color: '#666',
    margin: '0',
  },
  list: {
    display: 'grid',
    gap: '2rem',
  },
  post: {
    padding: '1.5rem',
    borderRadius: '12px',
    background: '#f8fafc',
  },
  postTitle: {
    fontSize: '1.5rem',
    color: '#1f2937',
    textDecoration: 'none',
  },
  excerpt: {
    marginTop: '0.75rem',
    color: '#4b5563',
    lineHeight: 1.6,
  },
  empty: {
    textAlign: 'center' as const,
    padding: '3rem',
    color: '#94a3b8',
  },
};

```

This page uses the same `config` pattern to pre-render tag listings. Because the router passes `params`, the component can render the correct posts without any extra helpers.

Step 9: Wire up the router

Static pages still work with regular hyperlinks, but if you want seamless client-side navigation you can reuse the low-level router utilities.

Create `src/router.ts`:

```

import { createRouter } from '@philjs/router';
import { render } from '@philjs/core';

type RouteEntry = {
  pattern: string;
  load: () => Promise<{ default: (props: any) => any }>;
};

const routes: RouteEntry[] = [
  { pattern: '/', load: () => import('./routes/index.tsx') },
  { pattern: '/blog', load: () => import('./routes/blog/index.tsx') },
  { pattern: '/blog/:slug', load: () => import('./routes/blog/[slug].tsx') },
  { pattern: '/tags/:tag', load: () => import('./routes/tags/[tag].tsx') },
];

const router = createRouter(
  Object.fromEntries(routes.map((route) => [route.pattern, route.load]))
);

let outlet: HTMLElement | null = null;

function pathMatch(pattern: string, pathname: string): Record<string, string> | null {
  const paramNames: string[] = [];
  const regex = new RegExp(
    `^${pattern.replace(/\//g, '\\\\/')}` +
    `.replace(/:[^/]+/g, segment => {
      paramNames.push(segment.slice(1));
      return `([/]+)`;
    })` +
    '$'
  );

  const match = pathname.match(regex);
  if (!match) return null;

  const params: Record<string, string> = {};
  paramNames.forEach((name, index) => {
    params[name] = decodeURIComponent(match[index + 1]);
  });
  return params;
}

async function renderRoute(url: URL) {
  if (!outlet) throw new Error('Router not started');

  for (const entry of routes) {
    const params = pathMatch(entry.pattern, url.pathname);
    if (!params) continue;

    const mod = await router.manifest[entry.pattern]();
    const Component = mod.default;
    render(() => Component({ params, url, navigate }), outlet);
    return;
  }

  render(() => 'Not Found', outlet);
}

export async function navigate(to: string) {
  const url = new URL(to, window.location.origin);
  window.history.pushState({}, '', url.toString());
  await renderRoute(url);
}

export function startRouter(target: HTMLElement) {
  outlet = target;

  document.addEventListener('click', (event) => {
    const anchor = (event.target as HTMLElement).closest<HTMLAnchorElement>('a[data-router-link]');
    if (!anchor || anchor.target === '_blank' || anchor.hasAttribute('download')) return;

    const url = new URL(anchor.href);
    if (url.origin !== window.location.origin) return;

    event.preventDefault();
    void navigate(url.pathname + url.search + url.hash);
  });

  window.addEventListener('popstate', () => {
    renderRoute(new URL(window.location.href));
  });

  renderRoute(new URL(window.location.href));
}

```

Update src/main.tsx:

```
import { startRouter } from './router.ts';

startRouter(document.getElementById('app')!);
```

With the router in place, all of the anchors marked with `data-router-link` perform client-side navigation while still working as normal links when JavaScript is disabled.

Step 10: Generate RSS Feed

Create `src/lib/rss.ts`:

```
import { getAllPosts } from './posts';
import type { Post } from './posts';

export async function generateRSS(): Promise<string> {
  const posts = await getAllPosts();
  const siteURL = 'https://myblog.com';

  const rssItems = posts
    .map(post => {
      return `
<item>
  <title>${escapeXml(post.title)}</title>
  <link>${siteURL}/blog/${post.slug}</link>
  <description>${escapeXml(post.excerpt)}</description>
  <pubDate>${new Date(post.date).toUTCString()}</pubDate>
  <guid>${siteURL}/blog/${post.slug}</guid>
  ${post.tags.map(tag => `<category>${escapeXml(tag)}</category>`).join('\n')}
</item>
`;
    })
    .trim()
    .join('\n');

  return `<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0" xmlns:atom="http://www.w3.org/2005/Atom">
  <channel>
    <title>My PhilJS Blog</title>
    <link>${siteURL}</link>
    <description>Articles about web development and PhilJS</description>
    <language>en</language>
    <atom:link href="${siteURL}/rss.xml" rel="self" type="application/rss+xml"/>
    ${rssItems}
  </channel>
</rss>`;
}

function escapeXml(unsafe: string): string {
  return unsafe.replace(/[\&<\>]/g, c => {
    switch (c) {
      case '<': return '&lt;';
      case '>': return '&gt;';
      case '&': return '&amp;';
      case "'": return '&apos;';
      case '"': return '&quot;';
      default: return c;
    }
  });
}
```

Create `src/routes/rss.xml.ts`:

```
import { generateRSS } from '../lib/rss';

export async function GET() {
  const rss = await generateRSS();

  return new Response(rss, {
    headers: {
      'Content-Type': 'application/xml',
      'Cache-Control': 'max-age=3600, s-maxage=3600',
    },
  });
}
```

Step 11: Generate Sitemap

Create `src/routes/sitemap.xml.ts`:

```

import { getAllPosts } from '../lib/posts';

export async function GET() {
  const posts = await getAllPosts();
  const siteURL = 'https://myblog.com';

  const staticPages = [
    '',
    '/blog',
  ];

  const urls = [
    ...staticPages.map(page => ({
      loc: `${siteURL}${page}`,
      lastmod: new Date().toISOString().split('T')[0],
      changefreq: 'daily',
      priority: page === '' ? '1.0' : '0.8',
    })),
    ...posts.map(post => ({
      loc: `${siteURL}/blog/${post.slug}`,
      lastmod: post.date,
      changefreq: 'weekly',
      priority: '0.9',
    })),
  ];
}

const sitemap = `<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
${urls.map(url => `
<url>
  <loc>${url.loc}</loc>
  <lastmod>${url.lastmod}</lastmod>
  <changefreq>${url.changefreq}</changefreq>
  <priority>${url.priority}</priority>
</url>
`).join('')}
</urlset>`;

return new Response(sitemap, {
  headers: {
    'Content-Type': 'application/xml',
    'Cache-Control': 'max-age=3600, s-maxage=3600',
  },
});
}

```

Step 12: Configure Static Generation

Create philjs.config.ts:

```

import { defineConfig } from 'philjs/config';

export default defineConfig({
  // Enable static site generation
  output: 'static',

  // Configure build
  build: {
    // Generate static pages
    prerender: true,

    // Optimize images
    images: {
      formats: ['webp', 'avif'],
      sizes: [640, 750, 828, 1080, 1200],
    },
  },

  // SEO
  seo: {
    sitemap: true,
    robots: true,
  },
});

```

Step 13: Build and Deploy

Build for Production

```
pnpm build
```

This generates static HTML files in dist/:

```
dist/
├── index.html
├── blog/
│   ├── index.html
│   ├── first-post/
│   │   └── index.html
│   ├── second-post/
│   │   └── index.html
│   └── getting-started/
│       └── index.html
└── rss.xml
└── sitemap.xml
└── _assets/
    ├── index-abc123.js
    └── index-def456.css
```

Deploy to Vercel

```
# Install Vercel CLI
pnpm add -g vercel

# Deploy
vercel --prod
```

Deploy to Netlify

```
# Install Netlify CLI
pnpm add -g netlify-cli

# Deploy
netlify deploy --prod --dir=dist
```

Deploy to Cloudflare Pages

```
# Install Wrangler
pnpm add -g wrangler

# Deploy
wrangler pages publish dist
```

Step 14: Add Incremental Static Regeneration

Update the config export in `src/routes/blog/[slug].tsx` for ISR:

```
export const config = {
  mode: 'isr' as const,
  revalidate: 3600, // Revalidate every hour
  fallback: 'blocking' as const,
  async getStaticPaths() {
    const posts = await getAllPosts();
    return posts.map(post => `/blog/${post.slug}`);
  },
};
```

Now pages regenerate automatically after 1 hour, keeping content fresh without rebuilding the entire site.

Performance Optimization

Image Optimization

```
import { Image } from 'philsjs/image';

<Image
  src="/images/post.jpg"
  alt="Post cover"
  width={800}
  height={400}
  loading="lazy"
  placeholder="blur"
/>
```

Code Splitting

```
// Lazy load comments
const Comments = lazy(() => import('./Comments'));

<Suspense fallback=<CommentsLoading />>
  <Comments postId={post.id} />
</Suspense>
```

Prefetching

```

import { initSmartPreloader } from '@philjs/router';

const preloader = initSmartPreloader({ strategy: 'hover' });

// Prefetch on hover
<a href={`/blog/${post.slug}`}
  data-router-link
  onMouseEnter={() => preloader?.preload(`/blog/${post.slug}`, { strategy: 'hover' })}>
  {post.title}
</a>

```

What You Learned

- **Static Site Generation** - Building pages at build time
- **Markdown content** - Using frontmatter and converting to HTML
- **SEO optimization** - Meta tags, Open Graph, Twitter Cards
- **RSS feeds** - Syndicating content
- **Sitemaps** - Helping search engines discover pages
- **File-based routing** - Organizing pages by URL structure
- **ISR** - Keeping static pages fresh
- **Deployment** - Publishing to Vercel, Netlify, Cloudflare

Challenges

Extend your blog:

1. **Comments**: Add comment system (Disqus, Giscus)
2. **Search**: Implement full-text search with Algolia
3. **Related posts**: Show similar articles
4. **Reading time**: Calculate and display reading time
5. **Table of contents**: Auto-generate from headings
6. **Code syntax highlighting**: Use Prism or Shiki
7. **Dark mode**: Add theme switching
8. **Newsletter**: Integrate email signup
9. **Analytics**: Add privacy-friendly analytics
10. **CMS**: Connect to headless CMS (Contentful, Sanity)

Performance Results

Your static blog will be incredibly fast:

- **First Contentful Paint**: < 0.5s
- **Time to Interactive**: < 1s
- **Lighthouse Score**: 95-100
- **Bundle Size**: ~15KB gzipped
- **Page Load**: Instant (prerendered HTML)

Next Steps

- [Learn About Routing](#) - Master file-based routing
- [Explore SSG Details](#) - Deep dive into static generation
- [SEO Best Practices](#) - Optimize for search engines

Next: [Thinking in PhilJS](#) →

Components

Components are TypeScript functions that return JSX. They are plain functions, easy to test, and trivial to compose.

Typed props

```

type GreetingProps = { name: string };

export function Greeting({ name }: GreetingProps) {
  return <p>Hello {name}</p>;
}

```

Children

```

import type { JSXChild } from "@philjs/core";

type CardProps = {
  title: string;
  children?: JSXChild;
};

export function Card({ title, children }: CardProps) {
  return (
    <section class="card">
      <h2>{title}</h2>
      <div>{children}</div>
    </section>
  );
}

```

Composition

```

export function Page() {
  return (
    <Card title="Summary">
      <p>Signals, SSR, and islands in one system.</p>
    </Card>
  );
}

```

Default values

```

type BadgeProps = { tone?: "primary" | "neutral" };

export function Badge({ tone = "primary" }: BadgeProps) {
  return <span class={`badge badge-${tone}`}>{tone}</span>;
}

```

Events and handlers

Event handlers receive native events; keep them small and avoid inline heavy allocations.

```

export function SearchBox({ onSearch }: { onSearch: (q: string) => void }) {
  return (
    <input
      type="search"
      onInput={(ev) => onSearch((ev.target as HTMLInputElement).value)}
      placeholder="Search..." />
  );
}

```

Refs and lifecycle

Use ref callbacks for DOM access; pair with onCleanup for listeners.

```

import { onCleanup } from '@philjs/core';

export function Focusable() {
  let el: HTMLButtonElement | null = null;
  onCleanup(() => el?.removeEventListener('click', () => {}));
  return <button ref={(node) => (el = node)}>Click</button>;
}

```

Composition patterns

- **Slots:** accept children and optional named slots as props.
- **Render props:** pass functions for dynamic content.
- **Headless components:** expose state via signals and render via children.

```

export function Dialog({ open, children }: { open: boolean; children: JSXChild }) {
  if (!open) return null;
  return (
    <div role="dialog" aria-modal="true">
      {children}
    </div>
  );
}

```

Performance

- Keep props serializable for SSR; avoid passing giant objects unless memoized.
- Extract heavy computations into memos/resources outside render.
- Use lists with stable keys to prevent unnecessary DOM churn.

Testing components

- Render with `@philjs/testing` and assert via roles/labels.
- Avoid snapshotting entire trees; test behavior (text, aria, navigation).
- Mock only boundaries (network/storage); keep component logic real.

Checklist

- Props typed; defaults set for optional props.
- Children handled safely (null/undefined allowed).
- Event handlers lean; avoid re-binding heavy functions per render.
- Refs cleaned up if adding listeners.
- Tested with realistic interactions.

Signals and Reactivity

Signals are the core reactive primitive in PhilJS. They are functions you call to read, and they expose a `.set` method to update.

Basic signal

```
import { signal } from "@philjs/core";

const count = signal(0);

count(); // 0
count.set(1); // update
```

Derived values with memo

```
import { memo, signal } from "@philjs/core";

const price = signal(49);
const tax = memo(() => price() * 0.0825);
const total = memo(() => price() + tax());
```

Writable computed values

```
import { linkedSignal, signal } from "@philjs/core";

const first = signal("Ada");
const last = signal("Lovelace");
const full = linkedSignal(() => `${first()} ${last()}`);

full(); // "Ada Lovelace"
full.set("A. L."); // override
full.reset(); // back to computed
```

Batch updates

```
import { batch, signal } from "@philjs/core";

const first = signal("Ada");
const last = signal("Lovelace");

batch(() => {
  first.set("Grace");
  last.set("Hopper");
});
```

Read without tracking

```
import { signal, untrack } from "@philjs/core";

const count = signal(0);
const safeRead = () => untrack(() => count());
```

Async data with resources

```
import { resource, signal } from "@philjs/core";

const userId = signal("u_1");
const user = resource(async () => {
  const res = await fetch(`/api/users/${userId()}`);
  return res.json();
});
```

Effects

Effects run when dependencies change—use them only for side effects (logging, DOM, network), not for computing values.

```
import { effect } from '@philjs/core';

effect(() => {
  console.log('User changed', userId());
});
```

Keep effects lean; prefer `memo/resources` for derived data.

Best practices

- **Read first, set second:** avoid reading signals inside setters unless wrapped in `batch`.
- **Prefer memos** for derived values; avoid recomputing in render.
- **Keep graphs shallow:** many small signals beat one large object with deep paths.
- **Avoid async in effects:** use resources or actions instead.
- **Use `untrack`** when you need to read without subscribing (e.g., logging).

Testing signals

```
import { describe, it, expect } from 'vitest';
import { signal, memo, batch } from '@philjs/core';

describe('signals', () => {
  it('updates memos once per batch', () => {
    const a = signal(1);
    const b = signal(2);
    const sum = memo(() => a() + b());
    batch(() => { a.set(3); b.set(4); });
    expect(sum()).toBe(7);
  });
});
```

Performance tips

- Batch related updates to avoid redundant recomputation.
- Favor primitive values over giant objects; if using objects, update by path to avoid churn.
- In lists, keep stable keys and avoid recreating signal containers per render if not needed.

Checklist

- Derived data uses `memo` or `linkedSignal`, not effects.
- Expensive reads wrapped in memos.
- Batch multiple writes.
- `untrack` used when reading without subscribing.
- Tests cover memo recomputation and batching.

Try it now: optimistic counter with batch

```
const counter = signal(0);
const double = memo(() => counter() * 2);

function incrementMany(n: number) {
  batch(() => {
    for (let i = 0; i < n; i++) counter.update(v => v + 1);
  });
}
```

Call `incrementMany(5)` and assert `double()` reflects the batched result with a single recompute.

Signals Cookbook

Practical patterns for PhilJS signals and memos.

Derived counts and filters

```
const filter = signal('');
const todos = signal([{ id: '1', title: 'Docs' }, { id: '2', title: 'Ship' }]);
const filtered = memo(() =>
  todos().filter(t => t.title.toLowerCase().includes(filter().toLowerCase())))
);
```

Toggle helpers

```

function createToggle(initial = false) {
  const state = signal(initial);
  return {
    state,
    toggle: () => state.update(v => !v),
    on: () => state.set(true),
    off: () => state.set(false),
  };
}

```

Batching updates

```

batch(() => {
  count.set(c => c + 1);
  other.set(o => o + 1);
});

```

Linked signals (writable computed)

```

const first = signal('Ada');
const last = signal('Lovelace');
const full = linkedSignal(() => `${first()} ${last()}`);
full.set('Grace Hopper');
full.reset();

```

Untracked reads

```

const user = signal({ id: 'u1', name: 'Ava' });
const logOnce = () => console.log(untrack(() => user().name));

```

Effects do side effects only

```

effect(() => {
  document.title = `Count ${count()}`;
});

```

Interval with cleanup

```

effect(() => {
  const id = setInterval(() => tick.update(n => n + 1), 1000);
  onCleanup(() => clearInterval(id));
});

```

Patterns to avoid

- Avoid async in effects; use resources for data fetching.
- Avoid storing large objects in a single signal when slices are better.
- Avoid creating signals in render for stable values; hoist them.

Checklist

- Derived data in memos, not effects.
- Batch multiple writes.
- Clean up side effects with onCleanup.
- Use untrack when reading without subscribing.

Effects and Memos

Use effects for side effects, and memos for derived values that should update automatically.

Effects

```

import { effect, onCleanup, signal } from "@philjs/core";

const online = signal(true);

const dispose = effect(() => {
  const handler = () => online.set(navigator.onLine);
  window.addEventListener("online", handler);
  window.addEventListener("offline", handler);

  onCleanup(() => {
    window.removeEventListener("online", handler);
    window.removeEventListener("offline", handler);
  });
});

```

Memos

```

import { memo, signal } from "@philjs/core";

const count = signal(1);
const doubled = memo(() => count() * 2);

```

Manage effect lifetimes

```

import { createRoot, effect, signal } from "@philjs/core";

const count = signal(0);

const dispose = createRoot(() => {
  effect(() => console.log("count", count()));
  return () => {};
});

// Later
// dispose();

```

When to use what

- **effect**: DOM/event listeners, logging, network calls, imperative bridges.
- **memo**: derived values that should stay in sync with dependencies.
- **resource**: async data with loading/error states.
- **linkedSignal**: writable computed with override/reset behavior.

Avoiding pitfalls

- Do not perform heavy computation inside effects; compute in memos.
- Avoid async effects; use resources or actions instead.
- Clean up listeners with `onCleanup` to prevent leaks.
- Keep dependency graphs shallow; break large effects into smaller ones.

Untracking in effects

Use `untrack` to read without subscribing:

```

effect(() => {
  const value = untrack(() => expensiveSignal());
  console.log(value);
});

```

Testing effects and memos

```

import { describe, it, expect, vi } from 'vitest';
import { effect, memo, signal } from '@philjs/core';

describe('effects', () => {
  it('runs when dependencies change', () => {
    const a = signal(1);
    const spy = vi.fn();
    effect(() => spy(a()));
    a.set(2);
    expect(spy).toHaveBeenCalledTimes(2); // initial + update
  });
});

```

Checklist

- Side effects live in `effect`; derived values in `memo`.
- Effects cleaned up via `onCleanup`.

- No async work inside effects; use resources/actions.
- Expensive reads wrapped in memos to avoid churn.

Try it now: memo + effect combo

```
const count = signal(0);
const label = memo(() => `Count is ${count()}`);

effect(() => {
  document.title = label(); // derived value used in side effect
});
```

Click a button to change `count` and watch both the UI and document title stay in sync.

JSX Basics

PhilJS uses JSX with a custom runtime. Configure TypeScript to use `@philjs/core`.

```
{
  "compilerOptions": {
    "jsx": "preserve",
    "jsxImportSource": "@philjs/core"
  }
}
```

Elements and props

```
export function Hero() {
  return (
    <header class="hero" aria-label="Landing hero">
      <h1>Build faster with PhilJS</h1>
      <p>Signals-first UI with streaming SSR.</p>
    </header>
  );
}
```

Styling with objects

```
import type { CSSProperties } from "@philjs/core";

const panelStyle: CSSProperties = {
  padding: ".5rem",
  borderRadius: "12px",
  background: "#0f172a",
  color: "#e2e8f0",
};

export function Panel() {
  return <section style={panelStyle}>Panel content</section>;
}
```

Reactive attributes

```
import { signal } from "@philjs/core";

const selected = signal(false);

export function Toggle() {
  return (
    <button class={() => (selected() ? "on" : "off")}>
      {() => (selected() ? "On" : "Off")}
    </button>
  );
}
```

Fragments and arrays

Return fragments or arrays to avoid extra wrapper DOM:

```

export function Breadcrumbs({ items }: { items: string[] }) {
  return (
    <>
      {items.map((item, i) => (
        <span key={item}>
          {item}
          {i < items.length - 1 ? " / " : ""}
        </span>
      ))}
    </>
  );
}

```

Controlled vs uncontrolled

- Controlled inputs bind value to signals.
- Uncontrolled inputs use defaultValue and refs; better for large forms when you don't need every keystroke.

```

const name = signal("");
<input value={name()} onInput={(ev) => name.set((ev.target as HTMLInputElement).value)} />;

```

Accessibility first

- Use semantic elements (button, nav, main).
- Add aria-* labels when semantics aren't enough.
- Use for/id pairs for form controls; leverage role-based queries in tests.

TypeScript helpers

- JSX.Element and JSXChild from @philjs/core help type children and components.
- Use ComponentProps<"button"> when forwarding native props.

```

type ButtonProps = ComponentProps<"button"> & { tone?: "primary" | "ghost" };

```

Patterns to avoid

- Mutating props.
- Creating new functions/objects in render when not needed (pull them up or memoize).
- Heavy logic in JSX expressions; precompute with memos.

Try it now: accessible button with forwarded props

```

import type { ComponentProps } from '@philjs/core';

type ButtonProps = ComponentProps<'button'> & { tone?: 'primary' | 'ghost' };
export function Button({ tone = 'primary', ...rest }: ButtonProps) {
  return (
    <button
      class={`btn btn-${tone}`}
      {...rest}
    />
  );
}

```

Use in tests with `getByRole('button', { name: /submit/i })` to keep a11y baked in.

Conditional and Lists

Use normal TypeScript control flow. JSX stays a pure expression.

Conditional rendering

```

import { signal } from "@philjs/core";

type User = { name: string };
const user = signal<User | null>(null);

export function Welcome() {
  return (
    <section>
      {user() ? <p>Welcome, {user()!.name}</p> : <p>Sign in to continue.</p>}
    </section>
  );
}

```

Rendering lists

```

type Task = { id: string; title: string };
const tasks = signal<Task[]>([]);

export function TaskList() {
  return (
    <ul>
      {tasks().map((task) => (
        <li key={task.id}>{task.title}</li>
      ))}
    </ul>
  );
}

```

Guards and fallbacks

Use early returns and small helpers for clarity:

```

function EmptyState() {
  return <p>No tasks yet. Add one?</p>;
}

export function TaskList() {
  const data = tasks();
  if (!data.length) return <EmptyState />;
  return (
    <ul>
      {data.map(task => <li key={task.id}>{task.title}</li>)}
    </ul>
  );
}

```

Async-friendly rendering

Pair with resources for loading and error states:

```

const stats = resource(fetchStats);

export function StatsPanel() {
  return (
    <section>
      {stats.loading && <p>Loading...</p>}
      {stats.error && <p role="alert">Failed to load stats</p>}
      {stats() && <pre>JSON.stringify(stats(), null, 2)</pre>}
    </section>
  );
}

```

Keys and stability

- Always use stable key values (ids) for lists.
- Avoid using array indexes except for static lists.
- For large lists, consider windowing/virtualization (community libs) to cut DOM cost.

Conditional classes/styles

```

const active = signal(false);

<button
  class={() => `tab ${active() ? 'tab-active' : ''}`}
  aria-pressed={() => active() ? 'true' : 'false'}
>
  Tab
</button>

```

Checklist

- Stable keys for dynamic lists.
- Loading/error/empty states for async data.
- Accessible states (aria-pressed/aria-busy/role="alert").
- Avoid heavy computation inline; precompute with memos.

Try it now: filtered list

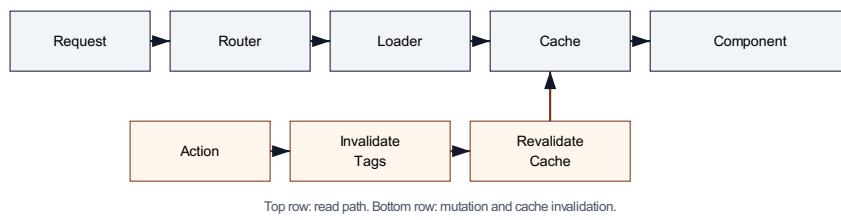
```

const filter = signal('');
const todos = signal([{ id: '1', title: 'Docs' }, { id: '2', title: 'Ship' }]);
const filtered = memo(() =>
  todos().filter(t => t.title.toLowerCase().includes(filter().toLowerCase())))
;
```

Render `filtered()` with stable keys; type into the filter input and watch recompute only once per change.

Routing Overview

PhilJS ships a declarative router that pairs zero-hydration resumability with intent-aware navigation. This guide walks through the essentials: defining routes, navigating, working with parameters, and composing layouts.



Top row: read path. Bottom row: mutation and cache invalidation.

Routing data flow diagram

Router Quick Start

Route components live wherever you prefer (many apps use `src/routes/`). Create a layout and a few pages:

```
// src/routes/_layout.tsx
import { Link } from '@philjs/router';

export function AppLayout({ children }: { children: any }) {
  return (
    <div style={{ fontFamily: 'system-ui, sans-serif', minHeight: '100vh', background: '#f8fafc' }}>
      <header style={{ padding: '1.5rem', display: 'flex', gap: '1rem', alignItems: 'center', background: 'white', boxShadow: '0 1px 3px rgba(15, 23, 42, 0.1)' }}>
        <strong>PhilJS</strong>
        <nav style={{ display: 'flex', gap: '0.75rem' }}>
          <Link to="/">Home</Link>
          <Link to="/about">About</Link>
          <Link to="/docs">Docs</Link>
        </nav>
      </header>
      <main style={{ padding: '2rem' }}>{children}</main>
    </div>
  );
}

// src/routes/index.tsx
export function HomeRoute() {
  return (
    <section>
      <h1>Welcome to PhilJS</h1>
      <p>Build resumable apps with smart routing and zero hydration.</p>
    </section>
  );
}
```

Next wire the router in `src/main.tsx`:

```
import { createAppRouter } from '@philjs/router';
import { AppLayout } from './routes/_layout';
import { HomeRoute } from './routes/index';
import { AboutRoute } from './routes/about';
import { DocsRoute } from './routes/docs';

createAppRouter({
  target: '#app',
  prefetch: true, // enable intent-based preloading globally
  transitions: { type: 'fade', duration: 200 },
  routes: [
    {
      path: '/',
      layout: AppLayout,
      component: HomeRoute,
      children: [
        { path: '/about', component: AboutRoute },
        { path: '/docs', component: DocsRoute },
      ],
    },
  ],
});
```

The router renders into `#app`, tracks history, and automatically applies smart prefetching and view transitions.

Navigation Basics

Declarative links

Use the bundled `<Link>` component for client-side navigation. It integrates with the smart preloader and respects modifier keys:

```

import { Link } from '@philjs/router';

export function Nav() {
  return (
    <nav>
      <Link to="/">Home</Link>
      <Link to="/about">About</Link>
      <Link to="/docs" replace>Docs</Link>
    </nav>
  );
}

```

`Link` accepts standard anchor props (`className`, `style`, etc) plus `replace` and `prefetch` overrides.

Programmatic navigation

`useRouter()` exposes the current match and a `navigate()` helper:

```

import { useRouter } from '@philjs/router';

export function LoginForm() {
  const { navigate } = useRouter();

  async function handleSubmit(event: SubmitEvent) {
    event.preventDefault();
    // ...auth logic
    await navigate('/dashboard');
  }

  return (
    <form onSubmit={handleSubmit}>
      {/* fields */}
      <button type="submit">Log in</button>
    </form>
  );
}

```

Inspecting the current route

`useRoute()` returns the matched route (path, params, loader data, module). This is handy for breadcrumbs, analytics, or debugging overlays.

Dynamic Routes & Params

Paths with `:params` capture dynamic segments. The router injects them via the component props:

```

// src/routes/blog/[slug].tsx
export function BlogPostRoute({ params }: { params: { slug: string } }) {
  return (
    <article>
      <h1>Blog post: {params.slug}</h1>
    </article>
  );
}

```

In the route table:

```
{
  path: '/blog',
  component: BlogIndexRoute,
  children: [
    { path: '/:slug', component: BlogPostRoute },
  ],
}
```

URLs such as `/blog/getting-started` now render `BlogPostRoute` with `params.slug === 'getting-started'`.

Multiple parameters

You can nest as many param segments as needed:

```

export function UserPostRoute({ params }: { params: { id: string; postId: string } }) {
  return (
    <section>
      <h1>User {params.id}</h1>
      <h2>Post {params.postId}</h2>
    </section>
  );
}

```

Define the child route with `path: '/:id/posts/:postId'` under `/users`.

Layouts & Nested Routes

Layouts wrap child routes when you provide a `layout` function. Layouts receive the same props as pages plus a `children` slot:

```

function DashboardLayout({ params, children }: { params: { section?: string }; children: any }) {
  return (
    <div className="dashboard">
      <Sidebar active={params.section} />
      <main>{children}</main>
    </div>
  );
}

createAppRouter({
  routes: [
    {
      path: '/dashboard',
      layout: DashboardLayout,
      component: DashboardHome,
      children: [
        { path: '/analytics', component: AnalyticsRoute },
        { path: '/billing', component: BillingRoute },
      ],
    },
  ],
});

```

Layouts can also run loaders or set transition/prefetch defaults for entire sections by configuring the parent route.

Loaders & Actions (Preview)

Routes optionally define loader and action functions:

```
{
  path: '/docs/:slug',
  component: DocsRoute,
  loader: async ({ params }) => fetchDoc(params.slug),
  action: async ({ params, formData }) => saveDoc(params.slug, formData),
}
```

Loader results are passed to the component as data. Actions receive the same context plus `formData`. Upcoming releases will integrate these with PhilJS SSR/SSG automatically.

Smart Prefetch & Transitions

- Set `prefetch: true` at the router level to enable intent-based preloading. Override per route with `{ prefetch: { strategy: 'hover' } }`.
- Apply view transitions globally (`transitions: { type: 'fade' }`) or per route (`transition: false` to opt out).

Next Steps

- Learn more about [smart preloading](#) and [view transitions](#)
- Dive into [dynamic routes](#) and [nested layouts](#)
- Explore [static generation](#) and [ISR](#) using the same route definitions

Routing Basics

PhilJS routes are defined with `createAppRouter`. You describe paths, optional loaders, layouts, and the router handles rendering, history, smart prefetch, and view transitions.

Project Structure

A common layout is to group route components under `src/routes/`:

```

src/
  main.tsx
  routes/
    _layout.tsx
    index.tsx
    about.tsx
    docs.tsx

```

Each file exports a regular PhilJS component. Layouts receive `children` plus route props.

Registering the Router

`createAppRouter` mounts the application and registers the routes.

```
// src/main.tsx
import { createAppRouter } from '@philjs/router';
import { AppLayout } from './routes/_layout';
import { HomeRoute } from './routes/index';
import { AboutRoute } from './routes/about';
import { DocsRoute } from './routes/docs';

createAppRouter({
  target: '#app',
  prefetch: true,
  transitions: { type: 'fade', duration: 200 },
  routes: [
    {
      path: '/',
      layout: AppLayout,
      component: HomeRoute,
      children: [
        { path: '/about', component: AboutRoute },
        { path: '/docs', component: DocsRoute },
      ],
    },
  ],
});
```

- `target` can be a selector or element (defaults to `#app`).
- `prefetch` enables intent-aware preloading globally; override per route with `prefetch: { strategy: 'hover' }`.
- `transitions` configures view transitions. Disable for a route with `transition: false`.

Defining Route Components

Route components receive `{ params, data, url, navigate }`.

```
// src/routes/index.tsx
export function HomeRoute() {
  return (
    <section>
      <h1>Welcome to PhilJS</h1>
      <p>Build reusable apps with smart routing and zero hydration.</p>
    </section>
  );
}

// src/routes/about.tsx
export function AboutRoute() {
  return (
    <section>
      <h1>About</h1>
      <p>PhilJS combines fine-grained signals with resumability and intelligent tooling.</p>
    </section>
  );
}
```

Building a Layout

Layouts wrap child routes when you provide the `layout` property.

```
// src/routes/_layout.tsx
import { Link } from '@philjs/router';

export function AppLayout({ children }: { children: any }) {
  return (
    <div style={{ fontFamily: 'system-ui, sans-serif', minHeight: '100vh', background: '#f8fafc' }}>
      <header style={{ padding: '1.5rem', display: 'flex', gap: '1rem', background: 'white' }}>
        <strong>PhilJS</strong>
        <nav style={{ display: 'flex', gap: '0.75rem' }}>
          <Link to="/">Home</Link>
          <Link to="/about">About</Link>
          <Link to="/docs">Docs</Link>
        </nav>
      </header>
      <main style={{ padding: '2rem' }}>{children}</main>
    </div>
  );
}
```

Every child of `/` automatically renders inside `AppLayout`.

Navigation

Declarative Links

Use `<Link>` for client-side navigation. It behaves like an anchor, supports modifier keys, and hooks into the smart preloader.

```

import { Link } from '@philjs/router';

export function Nav() {
  return (
    <nav>
      <Link to="/">Home</Link>
      <Link to="/about">About</Link>
      <Link to="/docs" replace>Docs</Link>
    </nav>
  );
}

```

Programmatic Navigation

`useRouter()` returns { route, navigate }.

```

import { useRouter } from '@philjs/router';

export function LoginForm() {
  const { navigate } = useRouter();

  async function handleSubmit(event: SubmitEvent) {
    event.preventDefault();
    // Auth logic ...
    await navigate('/dashboard');
  }

  return (
    <form onSubmit={handleSubmit}>
      <button type="submit">Log in</button>
    </form>
  );
}

```

Inspecting the Current Route

`useRoute()` provides the matched route object.

```

import { useRoute } from '@philjs/router';

export function RouteDebugger() {
  const route = useRoute();
  return <pre>{JSON.stringify(route?.params ?? {}, null, 2)}</pre>;
}

```

Route Parameters

Paths containing `:segment` capture params. Components receive them via props.

```

// src/routes/blog/[slug].tsx
export function BlogPostRoute({ params }: { params: { slug: string } }) {
  return <article>Viewing {params.slug}</article>;
}

// add under the blog section
{
  path: '/blog',
  component: BlogIndexRoute,
  children: [
    { path: '/:slug', component: BlogPostRoute },
  ],
}

```

The current match is also available in `useRoute()`.

Loaders and Actions

Routes optionally define loader and action functions. Loaders return data passed to the component as `data`; actions handle form submissions.

```

{
  path: '/docs/:slug',
  component: DocsRoute,
  loader: async ({ params }) => fetchDoc(params.slug),
  action: async ({ params, formData }) => saveDoc(params.slug, formData),
}

```

Upcoming releases integrate these hooks with PhilJS SSR/SSG so static builds and resumable hydration share the same data layer.

Smart Prefetch & Transitions

- Enable intent-based prefetching per route: `{ prefetch: { strategy: 'hover' } }`.
- Opt-out of transitions per route: `{ transition: false }`.
- For shared-element animations, call `markSharedElement()` in route components.

Next Steps

- Explore [Dynamic Routes](#) for deeper param handling.
- Learn about [Nested Layouts](#) and modular route composition.
- Dive into [SSR & SSG](#) to render routes at build time.

Navigation and Links

PhilJS' high-level router includes everything you need for smooth navigation: declarative links, programmatic navigation, smart prefetching, and view transitions.

<Link> Component

Link behaves like an anchor, but intercepts clicks to perform client-side navigation and prefetches destinations when possible.

```
import { Link } from '@philjs/router';

export function Nav() {
  return (
    <nav>
      <Link to="/">Home</Link>
      <Link to="/about">About</Link>
      <Link to="/docs">Docs</Link>
    </nav>
  );
}
```

- `to`: destination path or absolute URL
- `replace`: replace instead of push history entry
- `prefetch`: override prefetch strategy ({`strategy: 'hover'`}, {`strategy: 'intent'`}, etc)
- Accepts standard anchor props (`className`, `style`, `target`, ...)

External URLs

Pass a full URL; PhilJS will let the browser handle it.

```
<Link to="https://philjs.dev" target="_blank" rel="noopener">Docs </Link>
```

Programmatic Navigation

Use `useRouter()` to navigate from code (e.g., after a mutation) and inspect the current route.

```
import { useRouter } from '@philjs/router';

export function LoginForm() {
  const { navigate } = useRouter();

  async function handleSubmit(event: SubmitEvent) {
    event.preventDefault();
    // authenticate user...
    await navigate('/dashboard');
  }

  return (
    <form onSubmit={handleSubmit}>
      <button type="submit">Log in</button>
    </form>
  );
}
```

`navigate(to, options)` accepts `replace` and `state` just like `history.pushState`.

Active Links

`useRoute()` exposes the current match. Compare against `route.path` or `route.params` to highlight navigation.

```
import { Link, useRoute } from '@philjs/router';

export function NavLink({ to, children }: { to: string; children: any }) {
  const route = useRoute();
  const isActive = route?.path === to;

  return (
    <Link
      to={to}
      className={isActive ? 'nav-link active' : 'nav-link'}
    >
      {children}
    </Link>
  );
}
```

For partial matches:

```
const isActive = route?.path?.startsWith(to);
```

Smart Prefetching

When you enabled `prefetch: true` in `createAppRouter`, `<Link>` automatically prefetches the destination using the global strategy (intent prediction by default). Override per link:

```
<Link to="/pricing" prefetch={{ strategy: 'hover', priority: 'high' }}>
  Pricing
</Link>
```

This taps into the smart preloader (intent from pointer velocity, history analysis, visibility).

View Transitions

Routes participate in the View Transitions API when you configure `transitions`. Customize per link by calling `navigateWithTransition()` if you need fine-grained control:

```
import { navigateWithTransition } from '@philjs/router';

async function goToModal() {
  await navigateWithTransition('/settings', {
    type: 'slide-up',
    duration: 220,
  });
}
```

Navigation Guards & Loaders

Route loader functions run before the component renders. Throw responses (e.g., `new Response('', { status: 302, headers: { 'Location': '/login' } })`) to redirect. Returning data becomes `props.data` inside the component.

```
{
  path: '/dashboard',
  component: DashboardRoute,
  loader: async ({ request }) => {
    const user = await requireUser(request);
    if (!user) {
      throw new Response('', { status: 302, headers: { 'Location': '/login' } });
    }
    return { user };
  },
}
```

Inside `DashboardRoute`:

```
export function DashboardRoute({ data }: { data: { user: User } }) {
  return <h1>Welcome back, {data.user.name}</h1>;
}
```

Best Practices

- Use `<Link>` instead of `<a>` for internal navigation to benefit from intent prefetching and transitions.
- Keep routes colocated with components (`src/routes/`) and import them in `createAppRouter` for clarity.
- Read from `useRoute()` inside layouts or analytics components for contextual information (params, loader data).
- Combine loaders with PhilJS' `createQuery` for caching and mutation workflows.

Next up: learn more about [dynamic routes](#) and [nested layouts](#), or add [SSR/SSG](#) to your project.

Dynamic Routes

Dynamic routes capture URL segments and pass them to your components. PhilJS follows the familiar `/users/:id` pattern via route definitions.

Basic Parameters

Define a route with `:param` and read it from the component props:

```
// routes/users/[id].tsx
export function UserProfileRoute({ params }: { params: { id: string } }) {
  return (
    <section>
      <h1>User {params.id}</h1>
    </section>
  );
}
```

In your router configuration:

```
createAppRouter({
  routes: [
    {
      path: '/',
      component: HomeRoute,
      children: [
        { path: '/users/:id', component: UserProfileRoute },
      ],
    },
  ],
});
```

Now `/users/42` renders `UserProfileRoute` with `params.id === '42'`.

Data Loading with Params

Combine route params with a loader. Loader results appear in `props.data`.

```
{
  path: '/users/:id',
  component: UserProfileRoute,
  loader: async ({ params }) => {
    const res = await fetch(`/api/users/${params.id}`);
    if (!res.ok) throw new Response('Not found', { status: 404 });
    return res.json();
  }
}

export function UserProfileRoute({ params, data }: { params: { id: string }; data: User }) {
  return (
    <article>
      <h1>{data.name}</h1>
      <p>ID: {params.id}</p>
    </article>
  );
}
```

Multiple Parameters

Add as many segments as you need:

```
export function UserPostRoute({ params }: { params: { userId: string; postId: string } }) {
  return (
    <section>
      <h1>Post {params.postId}</h1>
      <p>By user {params.userId}</p>
    </section>
  );
}

{
  path: '/users/:userId/posts/:postId',
  component: UserPostRoute,
}
```

`/users/123/posts/456` yields `{ userId: '123', postId: '456' }`.

Catch-All Routes (*)

Use `*` to match the rest of the path. The router stores the matched string under `params['*']`.

```
export function DocsRoute({ params }: { params: { '*': string } }) {
  const segments = params['*'].split('/') ?? [];
  return (
    <section>
      <h1>Docs</h1>
      <p>Path: {segments.join(' / ')})</p>
    </section>
  );
}

{
  path: '/docs/*',
  component: DocsRoute,
}

• /docs/getting-started → params['*'] === 'getting-started'
• /docs/guides/routing/basics → params['*'] === 'guides/routing/basics'
```

Optional Segments

Represent optional segments with separate routes so each gets the correct priority:

```
{
  path: '/products',
  component: ProductsIndex,
},
{
  path: '/products/:id',
  component: ProductDetail,
},
```

Type Safety

If you use TypeScript, annotate `params` in your component props. When loaders become fully integrated with the PhilJS build pipeline, we'll generate route type declarations automatically.

Tips

- Order matters: declare more specific routes first if you share the same prefix.
- Catch-all routes should appear last—they have the lowest priority.
- Combine loaders with PhilJS data primitives (`createQuery`) for caching and mutations.
- Use `useRoute()` if you need access to params outside the component (e.g., analytics overlays).

Continue to [Layouts](#) to learn how to share chrome across nested routes, or explore [navigation](#) for link best practices.

Route Parameters

PhilJS passes route params directly to your components. This guide expands on handling single parameters, multiple segments, and catch-alls.

Basic Param

```
export function ProductRoute({ params }: { params: { id: string } }) {
  return <h1>Product {params.id}</h1>;
}
```

Router configuration:

```
{
  path: '/products/:id',
  component: ProductRoute,
}
```

Multiple Segments

```
export function CategoryProductRoute({ params }: { params: { category: string; productId: string } }) {
  return (
    <section>
      <p>Category: {params.category}</p>
      <p>ID: {params.productId}</p>
    </section>
  );
}

{
  path: '/:category/:productId',
  component: CategoryProductRoute,
}
```

Optional Branches

Declare separate routes for each variant so priorities remain predictable:

```
{
  path: '/blog',
  component: BlogIndex,
},
{
  path: '/blog/:slug',
  component: BlogPost,
},
```

Catch-All (*)

Capture everything after a prefix:

```
export function DocsRoute({ params }: { params: { '*': string } }) {
  const segments = params['*']?.split('/') ?? [];
  return <pre>{segments.join(' / ')</pre>;
}
```

```
{
  path: '/docs/*',
  component: DocsRoute,
}
```

TypeScript Tips

Annotate `params` in component props for type safety. Future versions of PhilJS will generate route typings automatically based on your router configuration.

Accessing Params Globally

`useRoute()` returns the current match. Use it inside layouts, analytics components, or DevTools overlays:

```
import { useRoute } from '@philjs/router';

export function Breadcrumbs() {
  const route = useRoute();
  if (!route) return null;

  return <span>Viewing {route.params.slug ?? 'home'}</span>;
}
```

Ready for more? Check out [Dynamic Routes](#) for complex examples and [Route Guards](#) to protect sections based on params.

Layouts

Layouts wrap child routes so you can share chrome, data, or behaviors across sections of your app.

Basic Layout

Provide a `layout` function on a route definition. The layout receives the same props as a page (`params`, `data`, `url`, `navigate`) plus `children`.

```
// routes/_layout.tsx
import { Link } from '@philjs/router';

export function AppLayout({ children }: { children: any }) {
  return (
    <div className="shell">
      <header>
        <Link to="/">Home</Link>
        <Link to="/pricing">Pricing</Link>
        <Link to="/docs">Docs</Link>
      </header>
      <main>{children}</main>
      <footer>© {new Date().getFullYear()} PhilJS</footer>
    </div>
  );
}
```

```
createAppRouter({
  routes: [
    {
      path: '/',
      layout: AppLayout,
      component: HomeRoute,
      children: [
        { path: '/pricing', component: PricingRoute },
        { path: '/docs', component: DocsRoute },
      ],
    },
  ],
});
```

All child routes render inside `<AppLayout>` automatically.

Nested Layouts

Layouts can nest by adding another layout inside a child definition. Each level wraps the rendered output.

```

createAppRouter({
  routes: [
    {
      path: '/',
      layout: AppLayout,
      component: HomeRoute,
      children: [
        {
          path: '/dashboard',
          layout: DashboardLayout,
          component: DashboardHome,
          children: [
            { path: '/analytics', component: AnalyticsRoute },
            { path: '/billing', component: BillingRoute }
          ],
        },
      ],
    ],
  ],
});

```

```

export function DashboardLayout({ children, params }: { children: any; params: Record<string, string> }) {
  return (
    <div className="dashboard">
      <DashboardSidebar activePath={params['*']} />
      <section>{children}</section>
    </div>
  );
}

```

PhilJS wraps from the deepest layout up to the root layout—no special components needed.

Sharing Loader Data

Layouts can define loaders to fetch shared resources. Child routes receive the same `data` unless they override it.

```

{
  path: '/account',
  layout: AccountLayout,
  component: AccountOverview,
  loader: async ({ request }) => {
    const user = await requireUser(request);
    return { user };
  },
  children: [
    { path: '/settings', component: AccountSettings },
    { path: '/billing', component: AccountBilling },
  ],
}

export function AccountLayout({ data, children }: { data: { user: User }; children: any }) {
  return (
    <div>
      <h1>Welcome, {data.user.name}</h1>
      {children}
    </div>
  );
}

```

Child routes still have access to the parent loader data via `props.data`. You can combine this with route-specific loaders to compose complex datasets.

Guarding Routes

Throw a `Response` from a layout loader to redirect or block access for an entire section:

```

{
  path: '/admin',
  layout: AdminLayout,
  component: AdminHome,
  loader: async ({ request }) => {
    const user = await requireUser(request);
    if (!user.isAdmin) {
      throw new Response('', { status: 302, headers: { Location: '/login' } });
    }
    return { user };
  },
  children: [
    { path: '/users', component: ManageUsers },
    { path: '/settings', component: AdminSettings },
  ],
}

```

Styling Tips

- Extract layouts into separate files to keep route registration tidy (e.g., `routes/dashboard/_layout.tsx`).
- Pass `params`, `data`, or `url` to navigation components inside layouts for active states.
- Couple layouts with view transitions: set `transition` on the parent route to animate between child pages.

Next, explore how to wire up [navigation](#) or to work with [dynamic routes](#).

Route Groups

Organize your routes into logical groups without affecting the URL structure.

What You'll Learn

- Creating route groups
- Organizing by feature
- Multiple layouts with groups
- Shared components
- Group-specific middleware
- Best practices

What are Route Groups?

Route groups let you organize files without adding segments to the URL:

```
src/pages/
  (marketing)/
    about.tsx → /about (not /marketing/about)
    pricing.tsx → /pricing
  (app) /
    dashboard.tsx → /dashboard (not /app/dashboard)
```

Folders in parentheses don't appear in URLs.

Creating Route Groups

Use parentheses in folder names:

```
src/pages/
  (auth) /
    login.tsx → /login
    signup.tsx → /signup
    forgot-password.tsx → /forgot-password
  (shop) /
    products.tsx → /products
    cart.tsx → /cart
    checkout.tsx → /checkout
```

Multiple Layouts

Different route groups can have different layouts:

```
src/pages/
  (marketing) /
    layout.tsx → Marketing layout
    index.tsx → /
    about.tsx → /about
    pricing.tsx → /pricing
  (app) /
    layout.tsx → App layout
    dashboard.tsx → /dashboard
    settings.tsx → /settings
  (docs) /
    layout.tsx → Docs layout
    index.tsx → /docs
    [...slug].tsx → /docs/*
```

Marketing Layout

```
// src/pages/(marketing)/Layout.tsx
export default function MarketingLayout({ children }: { children: any }) {
  return (
    <div className="marketing">
      <header className="marketing-header">
        <Logo />
        <nav>
          <Link href="/">Home</Link>
          <Link href="/about">About</Link>
          <Link href="/pricing">Pricing</Link>
          <Link href="/login">Login</Link>
        </nav>
      </header>

      <main className="marketing-content">
        {children}
      </main>

      <footer className="marketing-footer">
        <p>&copy; 2024 My Company</p>
        <SocialLinks />
      </footer>
    </div>
  );
}
```

App Layout

```
// src/pages/(app)/Layout.tsx
export default function AppLayout({ children }: { children: any }) {
  return (
    <div className="app">
      <header className="app-header">
        <Logo />
        <UserMenu />
      </header>

      <div className="app-body">
        <aside className="app-sidebar">
          <nav>
            <Link href="/dashboard">Dashboard</Link>
            <Link href="/analytics">Analytics</Link>
            <Link href="/settings">Settings</Link>
          </nav>
        </aside>

        <main className="app-content">
          {children}
        </main>
      </div>
    </div>
  );
}
```

Docs Layout

```
// src/pages/(docs)/Layout.tsx
export default function DocsLayout({ children }: { children: any }) {
  return (
    <div className="docs">
      <DocsHeader />

      <div className="docs-container">
        <aside className="docs-nav">
          <DocsSidebar />
        </aside>

        <main className="docs-main">
          {children}
        </main>

        <aside className="docs-toc">
          <TableOfContents />
        </aside>
      </div>
    </div>
  );
}
```

Organizing by Feature

Group related functionality together:

```

src/pages/
  (auth)/
    login.tsx
    signup.tsx
    reset-password.tsx
    verify-email.tsx
  components/
    AuthForm.tsx
    SocialLogin.tsx
  utils/
    validation.ts
  (blog)/
    index.tsx → /blog
    [slug].tsx → /blog/:slug
  components/
    BlogCard.tsx
    CategoryFilter.tsx
  utils/
    formatDate.ts
  (ecommerce)/
    products.tsx → /products
    [productid].tsx → /products/:id
    cart.tsx → /cart
    checkout.tsx → /checkout
  components/
    ProductCard.tsx
    CartItem.tsx

```

Nested Route Groups

Route groups can be nested:

```

src/pages/
  (app)/
    (admin)/
      users.tsx → /users
      settings.tsx → /settings
    (user)/
      profile.tsx → /profile
      preferences.tsx → /preferences

```

Both admin and user routes render under /, but are organized separately.

Shared Components

Keep group-specific components together:

```

src/pages/
  (dashboard)/
    layout.tsx
    index.tsx
    analytics.tsx
  components/
    DashboardCard.tsx
    StatWidget.tsx
    Chart.tsx
  hooks/
    useDashboardData.ts
  utils/
    formatMetrics.ts

```

Import shared components within the group:

```

// src/pages/(dashboard)/index.tsx
import DashboardCard from './components/DashboardCard';
import StatWidget from './components/StatWidget';
import { useDashboardData } from './hooks/useDashboardData';

export default function Dashboard() {
  const data = useDashboardData();

  return (
    <div className="dashboard-grid">
      <DashboardCard title="Revenue">
        <StatWidget value={data.revenue} />
      </DashboardCard>
      <DashboardCard title="Users">
        <StatWidget value={data.users} />
      </DashboardCard>
    </div>
  );
}

```

Group-Specific Middleware

Apply middleware to route groups:

```
// src/pages/(admin)/Layout.tsx
import { useRouter } from '@philjs/router';
import { useUser } from '@hooks/useUser';

export default function AdminLayout({ children }: { children: any }) {
  const router = useRouter();
  const user = useUser();

  // Protect all admin routes
  if (!user() || user().role !== 'admin') {
    router.replace('/unauthorized');
    return null;
  }

  return (
    <div className="admin-layout">
      <AdminSidebar />
      <main>{children}</main>
    </div>
  );
}
```

Combining Route Groups

Mix route groups with regular routes:

```
src/pages/
  index.tsx + / (no group)
  (marketing)/
    about.tsx + /about
    pricing.tsx + /pricing
  (app) /
    dashboard.tsx + /dashboard
  blog/
    index.tsx + /blog (no group)
    [slug].tsx + /blog/:slug
```

Multiple Root Layouts

Use route groups for completely different app sections:

```
src/pages/
  (site) /
    layout.tsx → Public site layout
    index.tsx +
    about.tsx + /about
  (admin) /
    layout.tsx → Admin layout
    dashboard.tsx + /dashboard
    users.tsx + /users
  (docs) /
    layout.tsx → Documentation layout
    index.tsx + /docs
```

Each group gets its own <html> and <body>:

```
// src/pages/(site)/Layout.tsx
export default function SiteLayout({ children }: { children: any }) {
  return (
    <html lang="en">
      <head>
        <title>My Site</title>
      </head>
      <body className="site">
        <SiteHeader />
        {children}
        <SiteFooter />
      </body>
    </html>
  );
}
```

```
// src/pages/(admin)/Layout.tsx
export default function AdminLayout({ children }: { children: any }) {
  return (
    <html lang="en">
      <head>
        <title>Admin Panel</title>
      </head>
      <body className="admin">
        <AdminHeader />
        <AdminSidebar />
        {children}
      </body>
    </html>
  );
}
```

Route Groups with Dynamic Routes

Combine groups and dynamic routes:

```
src/pages/
(shop)-
products/
  index.tsx → /products
  [id].tsx → /products/:id
categories/
  [category].tsx → /categories/:category
cart.tsx → /cart
```

```
// src/pages/(shop)/products/[id].tsx
import { useParams } from '@philjs/router';

export default function Product() {
  const params = useParams<{ id: string }>();

  // Still renders at /products/:id
  // But organized in (shop) group

  return <div>Product {params.id}</div>;
}
```

Conditional Layouts by Group

```
// src/pages/(public)/layout.tsx
export default function PublicLayout({ children }: { children: any }) {
  return (
    <div>
      <PublicHeader />
      {children}
      <PublicFooter />
    </div>
  );
}

// src/pages/(authenticated)/Layout.tsx
import { useRouter } from '@philjs/router';
import { useUser } from '@hooks/useUser';

export default function AuthenticatedLayout({ children }: { children: any }) {
  const router = useRouter();
  const user = useUser();

  if (!user()) {
    router.replace('/login');
    return null;
  }

  return (
    <div>
      <AuthHeader user={user()!} />
      <Sidebar />
      {children}
    </div>
  );
}
```

Complete Example

E-commerce site with multiple sections:

```

src/pages/
  (storefront)/
    layout.tsx
    index.tsx → /
    about.tsx → /about
    contact.tsx → /contact

  (shop)/
    layout.tsx
    products/
      index.tsx → /products
      [id].tsx → /products/:id
    categories/
      [category].tsx → /categories/:category
    cart.tsx → /cart
    checkout.tsx → /checkout

  (account)/
    layout.tsx
    dashboard.tsx → /dashboard
    orders.tsx → /orders
    settings.tsx → /settings
    profile.tsx → /profile

  (admin)/
    layout.tsx
    dashboard.tsx → /admin/dashboard
    products.tsx → /admin/products
    orders.tsx → /admin/orders
    users.tsx → /admin/users

```

Storefront Layout

```

// src/pages/(storefront)/Layout.tsx
export default function StorefrontLayout({ children }: { children: any }) {
  return (
    <div className="storefront">
      <header>
        <Logo />
        <nav>
          <Link href="/">Home</Link>
          <Link href="/products">Shop</Link>
          <Link href="/about">About</Link>
          <Link href="/cart">Cart</Link>
        </nav>
      </header>

      <main>{children}</main>

      <footer>
        <Newsletter />
        <FooterLinks />
      </footer>
    </div>
  );
}

```

Shop Layout

```

// src/pages/(shop)/Layout.tsx
export default function ShopLayout({ children }: { children: any }) {
  return (
    <div className="shop">
      <ShopHeader />

      <div className="shop-container">
        <aside className="shop-filters">
          <CategoryFilter />
          <PriceFilter />
          <BrandFilter />
        </aside>

        <main className="shop-content">
          {children}
        </main>
      </div>
    </div>
  );
}

```

Account Layout

```
// src/pages/(account)/Layout.tsx
import { useRouter } from '@philjs/router';
import { useUser } from '@/hooks/useUser';

export default function AccountLayout({ children }: { children: any }) {
  const router = useRouter();
  const user = useUser();

  if (!user()) {
    router.replace('/login');
    return null;
  }

  return (
    <div className="account">
      <AccountHeader user={user()} />

      <div className="account-container">
        <aside className="account-nav">
          <Link href="/dashboard">Dashboard</Link>
          <Link href="/orders">Orders</Link>
          <Link href="/settings">Settings</Link>
          <Link href="/profile">Profile</Link>
        </aside>

        <main className="account-content">
          {children}
        </main>
      </div>
    </div>
  );
}
```

Best Practices

Use Groups for Organization

```
// ⚡ Good - organized by feature
src/pages/
  (auth)-
    login.tsx
    signup.tsx
  (blog)-
    index.tsx
    [slug].tsx

// ⚡ Cluttered - all files together
src/pages/
  login.tsx
  signup.tsx
  blog.tsx
  blog-post.tsx
```

Group Related Components

```
// ⚡ Keep components with their routes
src/pages/
  (dashboard)-
    index.tsx
    components/
      DashboardCard.tsx

// ⚡ Don't separate too much
src/
  pages/
    (dashboard)-
      index.tsx
    components/
      dashboard/
        DashboardCard.tsx
```

Use Descriptive Group Names

```
// ⚡ Clear purpose
(marketing)
(authenticated)
(admin)
(docs)

// ⚡ Vague or generic
(pages)
(routes)
(app1)
```

Don't Over-Group

```
// ⚠ Too many nested groups
src/pages/
  (site)/
    (public)/
      (marketing)/
        (home)/
          index.tsx

// ⚡ Simpler structure
src/pages/
  (marketing)/
    index.tsx
```

Combine with Regular Folders

```
// ⚡ Mix groups and regular folders when needed
src/pages/
  (app)/
    dashboard/
      index.tsx
      analytics.tsx
  api/
    users.ts
```

Common Patterns

Auth vs Public

```
src/pages/
  (public)/
    index.tsx
    about.tsx
    pricing.tsx
  (auth)/
    dashboard.tsx
    settings.tsx
```

Marketing vs App

```
src/pages/
  (marketing)/
    index.tsx
    features.tsx
    pricing.tsx
  (app)/
    dashboard.tsx
    workspace.tsx
```

User Roles

```
src/pages/
  (user)/
    dashboard.tsx
    profile.tsx
  (admin)/
    users.tsx
    settings.tsx
  (superadmin)/
    system.tsx
    logs.tsx
```

Summary

You've learned:

- Creating route groups with parentheses
 - Organizing routes by feature
 - Multiple layouts with groups
 - Shared components within groups
 - Group-specific middleware
 - Combining groups with dynamic routes
 - Best practices for organization
- Route groups keep your codebase organized without affecting URLs!

Next: [Middleware](#) → Protect routes and handle authentication

Route Guards

Protect routes by using loaders to check authentication/authorization before rendering. Throw responses to redirect or error out.

Auth Guard Example

```

import { createAppRouter } from '@philjs/router';
import { requireUser } from '../lib/auth';

createAppRouter({
  routes: [
    {
      path: '/admin',
      layout: AdminLayout,
      component: AdminHome,
      loader: async ({ request }) => {
        const user = await requireUser(request);
        if (!user.isAdmin) {
          throw new Response('', { status: 302, headers: { Location: '/login' } });
        }
        return { user };
      },
      children: [
        { path: '/users', component: ManageUsers },
        { path: '/settings', component: AdminSettings },
      ],
    },
  ],
});

```

`AdminLayout` receives `data.user` and can display shared UI:

```

export function AdminLayout({ data, children }: { data: { user: User }; children: any }) {
  return (
    <div>
      <h1>Admin • {data.user.name}</h1>
      {children}
    </div>
  );
}

```

Redirect Helpers

Use the standard `Response` constructor. For convenience create utility helpers:

```

export function redirect(to: string, status = 302) {
  return new Response('', { status, headers: { Location: to } });
}

throw redirect('/login');

```

Combining with Actions

Actions can enforce guards before mutating data:

```

{
  path: '/account/:id',
  component: AccountRoute,
  loader: async ({ params, request }) => ensureOwner(request, params.id),
  action: async ({ params, formData }) => updateAccount(params.id, formData),
}

```

If the guard fails, throw a redirect or error response; the router will stop rendering and surface the error to the nearest boundary.

Tips

- Keep guards in loaders/layouts so all children inherit the protection.
- Combine with `middleware` or server adapters for edge cases like rate limiting.
- Use `transition: false` on guard-protected routes when you need immediate state changes without animations.
- Track guard outcomes in your analytics by reading `useRoute()` after navigation.

Next explore [Middleware](#) for request-level hooks or review [Error Handling](#) to display nice fallbacks when guards reject.

Error Handling

Handle routing errors, 404 pages, and error boundaries for robust navigation.

What You'll Learn

- Custom 404 pages
- Error boundaries for routes
- Handling navigation errors
- API route errors
- Error recovery
- Best practices

404 Not Found

Basic 404 Page

```
// src/pages/404.tsx
import { Link } from '@philjs/router';

export default function NotFound() {
  return (
    <div className="not-found">
      <h1>404 - Page Not Found</h1>
      <p>The page you're looking for doesn't exist.</p>
      <Link href="/">Go Home</Link>
    </div>
  );
}
```

Custom 404 with Search

```
// src/pages/404.tsx
import { Link, usePathname } from '@philjs/router';
import { signal } from '@philjs/core';

export default function NotFound() {
  const pathname = usePathname();
  const searchQuery = signal('');

  return (
    <div className="not-found">
      <h1>Page Not Found</h1>
      <p>We couldn't find: <code>{pathname}</code></p>

      <div className="suggestions">
        <h2>Maybe you were looking for:</h2>
        <ul>
          <li><Link href="/">Home</Link></li>
          <li><Link href="/blog">Blog</Link></li>
          <li><Link href="/docs">Documentation</Link></li>
        </ul>
      </div>

      <div className="search">
        <h2>Search our site:</h2>
        <input
          type="search"
          value={searchQuery()}
          onInput={(e) => searchQuery.set(e.target.value)}
          placeholder="What are you looking for?"
        />
      </div>
    </div>
  );
}
```

404 with Analytics

```
// src/pages/404.tsx
import { usePathname } from '@philjs/router';
import { effect } from '@philjs/core';

export default function NotFound() {
  const pathname = usePathname();

  effect(() => {
    // Track 404 errors
    analytics.track('404_error', {
      path: pathname,
      referer: document.referrer
    });
  });

  return (
    <div>
      <h1>404 - Page Not Found</h1>
      <p>Path: {pathname}</p>
    </div>
  );
}
```

Error Boundaries

Route Error Boundary

```
// src/pages/error.tsx
import { useRouter } from '@philjs/router';

interface ErrorProps {
  error: Error;
  reset: () => void;
}

export default function Error({ error, reset }: ErrorProps) {
  const router = useRouter();

  return (
    <div className="error-page">
      <h1>Something went wrong</h1>
      <p>{error.message}</p>

      <div className="error-actions">
        <button onClick={reset}>Try Again</button>
        <button onClick={() => router.push('/')}>Go Home</button>
      </div>
    </div>
  );
}
```

Global Error Boundary

```
// src/App.tsx
import { ErrorBoundary } from '@philjs/core';
import { Router } from '@philjs/router';

export default function App() {
  return (
    <ErrorBoundary
      fallback={(error, reset) => (
        <div className="global-error">
          <h1>Application Error</h1>
          <p>{error.message}</p>
          <button onClick={reset}>Reload App</button>
        </div>
      )}
    >
    <Router />
  </ErrorBoundary>
);
}
```

Layout Error Boundary

```
// src/pages/(app)/Layout.tsx
import { ErrorBoundary } from '@philjs/core';

export default function AppLayout({ children }: { children: any }) {
  return (
    <div className="app-layout">
      <AppHeader />

      <ErrorBoundary
        fallback={(error, reset) => (
          <div className="route-error">
            <h2>Error loading page</h2>
            <p>{error.message}</p>
            <button onClick={reset}>Retry</button>
          </div>
        )}
      >
        {children}
      </ErrorBoundary>
    </div>
  );
}
```

Navigation Error Handling

Handle Navigation Errors

```

import { useRouter } from '@philjs/router';
import { signal, effect } from '@philjs/core';

export default function App() {
  const router = useRouter();
  const navigationError = signal<Error | null>(null);

  effect(() => {
    const handleError = (error: Error) => {
      console.error('Navigation error:', error);
      navigationError.set(error);
    };
    router.events.on('routeChangeError', handleError);
  });

  return () => {
    router.events.off('routeChangeError', handleError);
  };
}

return (
  <div>
    {navigationError() && (
      <div className="navigation-error">
        <p>Navigation failed: {navigationError()!.message}</p>
        <button onClick={() => navigationError.set(null)}>Dismiss</button>
      </div>
    )}
    <Router />
  </div>
);
}

```

Retry Failed Navigation

```

import { useRouter } from '@philjs/router';
import { signal } from '@philjs/core';

function useRetryableNavigation() {
  const router = useRouter();
  const lastAttemptedRoute = signal<string | null>(null);
  const error = signal<Error | null>(null);

  const navigate = async (path: string) => {
    lastAttemptedRoute.set(path);
    error.set(null);

    try {
      await router.push(path);
    } catch (e) {
      error.set(e as Error);
    }
  };

  const retry = () => {
    if (lastAttemptedRoute()) {
      navigate(lastAttemptedRoute()!);
    }
  };

  return { navigate, retry, error };
}

```

Data Fetching Errors

Handle Route Data Errors

```

// src/pages/blog/[slug].tsx
import { useParams } from '@philjs/router';
import { signal, effect } from '@philjs/core';

export default function BlogPost() {
  const params = useParams<{ slug: string }>();
  const post = signal(null);
  const error = signal<Error | null>(null);
  const loading = signal(true);

  effect(() => {
    loading.set(true);
    error.set(null);

    fetch(`/api/posts/${params.slug}`)
      .then(res => {
        if (!res.ok) {
          throw new Error(`Post not found: ${params.slug}`);
        }
        return res.json();
      })
      .then(data => {
        post.set(data);
        loading.set(false);
      })
      .catch(err => {
        error.set(err);
        loading.set(false);
      });
  });

  if (loading()) return <LoadingSkeleton />;
}

if (error()) {
  return (
    <div className="post-error">
      <h1>Error Loading Post</h1>
      <p>{error()!.message}</p>
      <Link href="/blog">← Back to Blog</Link>
    </div>
  );
}

if (!post()) {
  return <NotFound />;
}

return <PostView post={post()!} />;
}

```

Automatic Retry

```

import { signal, effect } from '@philjs/core';

function useRetryableFetch<T>(url: string, maxRetries = 3) {
  const data = signal<T | null>(null);
  const error = signal<Error | null>(null);
  const loading = signal(true);
  const retryCount = signal(0);

  effect(() => {
    const fetchData = async () => {
      try {
        loading.set(true);
        const res = await fetch(url);

        if (!res.ok) throw new Error(`HTTP ${res.status}`);

        const json = await res.json();
        data.set(json);
        error.set(null);
        retryCount.set(0);
      } catch (err) {
        if (retryCount() < maxRetries) {
          // Retry with exponential backoff
          const delay = Math.pow(2, retryCount()) * 1000;
          setTimeout(() => {
            retryCount.set(c => c + 1);
            fetchData();
          }, delay);
        } else {
          error.set(err as Error);
        }
      } finally {
        loading.set(false);
      }
    };
    fetchData();
  });
}

return { data, error, loading, retryCount };
}

```

API Route Errors

API Error Handler

```

// src/pages/api/users/[id].ts
export async function GET(request: Request, { params }: any) {
  try {
    const user = await db.users.findById(params.id);

    if (!user) {
      return new Response(
        JSON.stringify({ error: 'User not found' }),
        { status: 404 }
      );
    }

    return new Response(JSON.stringify(user), { status: 200 });
  } catch (error) {
    console.error('API error:', error);

    return new Response(
      JSON.stringify({ error: 'Internal server error' }),
      { status: 500 }
    );
  }
}

```

Validation Errors

```

import { z } from 'zod';

const UserSchema = z.object({
  name: z.string().min(1).max(100),
  email: z.string().email(),
  age: z.number().min(18).max(120)
});

export async function POST(request: Request) {
  try {
    const body = await request.json();

    // Validate
    const result = UserSchema.safeParse(body);

    if (!result.success) {
      return new Response(
        JSON.stringify({
          error: 'Validation failed',
          details: result.error.issues
        }),
        { status: 400 }
      );
    }

    // Create user
    const user = await db.users.create(result.data);

    return new Response(JSON.stringify(user), { status: 201 });
  } catch (error) {
    return new Response(
      JSON.stringify({ error: 'Server error' }),
      { status: 500 }
    );
  }
}

```

Error Recovery

Refresh on Error

```

import { signal, effect } from '@philjs/core';

export default function Dashboard() {
  const data = signal(null);
  const error = signal<Error | null>(null);

  const fetchData = () => {
    fetch('/api/dashboard')
      .then(r => r.json())
      .then(d => {
        data.set(d);
        error.set(null);
      })
      .catch(e => error.set(e));
  };

  effect(() => {
    fetchData();
  });

  if (error()) {
    return (
      <div className="error-state">
        <h2>Failed to load dashboard</h2>
        <p>{error()!.message}</p>
        <button onClick={fetchData}>Retry</button>
      </div>
    );
  }
}

return <DashboardView data={data()} />;
}

```

Fallback Content

```

import { signal, effect } from '@philjs/core';

export default function ProductList() {
  const products = signal([]);
  const error = signal<Error | null>(null);

  effect(() => {
    fetch('/api/products')
      .then(r => r.json())
      .then(d => products.set(d))
      .catch(e => {
        error.set(e);
      })
      // Use cached data as fallback
      const cached = localStorage.getItem('products');
      if (cached) {
        products.set(JSON.parse(cached));
      }
    });
  });

  return (
    <div>
      {error() && (
        <div className="error-banner">
          Error loading latest data. Showing cached content.
        </div>
      )}
      <ProductGrid products={products()} />
    </div>
  );
}

```

Offline Handling

Detect Offline State

```

import { signal, effect } from '@philjs/core';

const isOnline = signal(navigator.onLine);

export default function App() {
  effect(() => {
    const handleOnline = () => isOnline.set(true);
    const handleOffline = () => isOnline.set(false);

    window.addEventListener('online', handleOnline);
    window.addEventListener('offline', handleOffline);

    return () => {
      window.removeEventListener('online', handleOnline);
      window.removeEventListener('offline', handleOffline);
    };
  });

  return (
    <div>
      {!isOnline() && (
        <div className="offline-banner">
          You are offline. Some features may not be available.
        </div>
      )}
      <Router />
    </div>
  );
}

```

Queue Failed Requests

```

import { signal } from '@philjs/core';

const failedRequests = signal<Array<{ url: string; options: RequestInit }>>([]);

async function resilientFetch(url: string, options?: RequestInit) {
  try {
    const response = await fetch(url, options);

    if (!response.ok) throw new Error(`HTTP ${response.status}`);

    return response;
  } catch (error) {
    // Queue for retry
    failedRequests.set([...failedRequests(), { url, options: options || {} }]);
    throw error;
  }
}

// Retry when back online
window.addEventListener('online', async () => {
  const queued = failedRequests();
  failedRequests.set([]);

  for (const request of queued) {
    try {
      await fetch(request.url, request.options);
    } catch (error) {
      console.error('Retry failed:', error);
    }
  }
});

```

Error Monitoring

Log Errors to Service

```

function logError(error: Error, context?: any) {
  // Log to service like Sentry, LogRocket, etc.
  fetch('/api/errors', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      message: error.message,
      stack: error.stack,
      context,
      userAgent: navigator.userAgent,
      timestamp: Date.now()
    })
  }).catch(console.error);
}

```

Route Error Monitoring

```

import { useRouter, usePathname } from '@philjs/router';
import { effect } from '@philjs/core';

export default function ErrorMonitoring({ children }: { children: any }) {
  const router = useRouter();
  const pathname = usePathname();

  effect(() => {
    const handleError = (error: Error) => {
      logError(error, {
        route: pathname,
        type: 'navigation'
      });
    };

    router.events.on('routeChangeError', handleError);

    return () => {
      router.events.off('routeChangeError', handleError);
    };
  });

  return children;
}

```

User-Friendly Error Messages

Error Message Mapping

```

function getUserFriendlyMessage(error: Error): string {
  const errorMessages: Record<string, string> = {
    'NetworkError': 'Unable to connect. Please check your internet connection.',
    'NotFoundError': 'The requested item was not found.',
    'UnauthorizedError': 'You need to log in to access this page.',
    'ForbiddenError': 'You don\'t have permission to access this resource.',
    'ValidationError': 'Please check your input and try again.',
    'ServerError': 'Something went wrong on our end. Please try again later.'
  };

  return errorMessages[error.name] || 'An unexpected error occurred.';
}

```

Contextual Error Pages

```

interface ErrorPageProps {
  error: Error;
  context: 'route' | 'data' | 'auth';
}

export default function ErrorPage({ error, context }: ErrorPageProps) {
  const suggestions = {
    route: [
      'Check the URL for typos',
      'Go back to the home page',
      'Search our site'
    ],
    data: [
      'Refresh the page',
      'Try again in a moment',
      'Contact support if the problem persists'
    ],
    auth: [
      'Log in to your account',
      'Reset your password',
      'Contact support'
    ]
  };

  return (
    <div className="error-page">
      <h1>Oops!</h1>
      <p>{getUserFriendlyMessage(error)}</p>

      <div className="suggestions">
        <h2>Try this:</h2>
        <ul>
          {suggestions[context].map((s, i) => (
            <li key={i}>{s}</li>
          ))}
        </ul>
      </div>
    </div>
  );
}

```

Best Practices

Always Show User Feedback

```

// ⚠ Show error message
if (error()) {
  return <ErrorMessage error={error()} />;
}

// ⚠ Silent failure
if (error()) {
  console.error(error());
  return null;
}

```

Provide Recovery Options

```

// ⚡ Offer retry and alternatives
<div>
  <p>Error: {error()!.message}</p>
  <button onClick={retry}>Retry</button>
  <Link href="/">Go Home</Link>
</div>

// ⚡ Dead end
<div>Error occurred</div>

```

Log Errors for Debugging

```
// ⚡ Log with context
effect(() => {
  if (error()) {
    console.error('Route error:', {
      error: error(),
      route: pathname,
      timestamp: Date.now()
    });
  }
});

// ⚡ Generic Logging
console.error(error());
```

Use Error Boundaries

```
// ⚡ Catch errors at appropriate levels
<ErrorBoundary fallback={ErrorPage}>
  <Router />
</ErrorBoundary>

// ⚡ No error handling
<Router />
```

Differentiate Error Types

```
// ⚡ Handle different errors differently
if (error()?.name === 'NotFoundError') {
  return <NotFound />;
} else if (error()?.name === 'UnauthorizedError') {
  return <Login />;
} else {
  return <GenericError />;
}

// ⚡ Same treatment for all errors
return <Error />;
```

Summary

You've learned:

- Creating custom 404 pages
- Error boundaries for routes
- Navigation error handling
- Data fetching errors
- API route error responses
- Error recovery strategies
- Offline handling
- Error monitoring
- User-friendly error messages
- Best practices

Proper error handling creates resilient applications!

Next: [Loading States](#) → Show loading UI during navigation

Data Loading

PhilJS routes can fetch data before rendering by providing a `loader` function. Loaders run on navigation (and during SSR/SSG integration) and their return value becomes `props.data` in the route component.

Basic Loader

```
createAppRouter({
  routes: [
    {
      path: '/posts',
      component: PostsRoute,
      loader: async () => {
        const res = await fetch('/api/posts');
        if (!res.ok) throw new Error('Failed to load posts');
        return res.json();
      },
    ],
  ],
});
```

```
export function PostsRoute({ data }: { data: Post[] }) {
  return (
    <section>
      <h1>Latest Posts</h1>
      <ul>
        {data.map((post) => (
          <li key={post.id}>{post.title}</li>
        )));
      </ul>
    </section>
  );
}
```

Accessing Params

Loaders receive the same context object as actions. Use params to tailor fetches:

```
import { Ok, Err } from '@philjs/core';

{
  path: '/posts/:slug',
  component: BlogPostRoute,
  loader: async ({ params }) => {
    const res = await fetch(`/api/posts/${params.slug}`);
    if (!res.ok) return Err('not-found');
    return Ok(await res.json());
  },
}

export function BlogPostRoute({ params, data, error }: RouteComponentProps) {
  if (error) return <ErrorState code={error} slug={params.slug} />;
  return <ArticleView post={data} />;
}

> Tip: `Ok`/`Err` come from `@philjs/core`'s Rust-inspired `Result` helpers (`import { Ok, Err } from '@philjs/core'`). Use `isResult` if you want to branch manually.
```

Error Handling

Throw a Response or Error from a loader to signal failures. Catch these in error boundaries or allow the router to display fallback messages.

```
loader: async ({ params }) => {
  const res = await fetch(`/api/products/${params.id}`);
  if (res.status === 404) {
    throw new Response('Not found', { status: 404 });
  }
  if (!res.ok) {
    throw new Error('Unexpected error');
  }
  return res.json();
}
```

Combining with Signals

Loaders run once per navigation. For live updates, pair loader data with PhilJS signals or `createQuery`:

```
export function DashboardRoute({ data }: { data: DashboardData }) {
  const stats = signal(data.stats);

  effect(() => {
    const interval = setInterval(async () => {
      const res = await fetch('/api/dashboard/stats');
      stats.set(await res.json());
    }, 10_000);
    return () => clearInterval(interval);
  });

  return <StatsPanel stats={stats()} />;
}
```

Actions (Mutations)

Actions handle POST/PUT/DELETE requests. They receive `{ params, request, formData }`.

```
{
  path: '/posts/:slug',
  component: EditPostRoute,
  loader: async ({ params }) => fetchPost(params.slug),
  action: async ({ params, formData }) => {
    const res = await fetch(`/api/posts/${params.slug}`, {
      method: 'PUT',
      body: formData,
    });
    if (!res.ok) throw res;
    return res.json();
  },
}
```

In your form handler, prevent default and call `navigate()` after the action succeeds.

```

export function EditPostRoute({ data, params, navigate }: RouteComponentProps) {
  async function handleSubmit(event: SubmitEvent) {
    event.preventDefault();
    const formData = new FormData(event.target as HTMLFormElement);
    await fetch(`/posts/${params.slug}?_action`, { method: 'POST', body: formData });
    await navigate(`/posts/${params.slug}`);
  }

  return (
    <form onSubmit={handleSubmit}>
      {/* form fields */}
    </form>
  );
}

```

Note: Action plumbing is available in the router today; upcoming releases will tie it into PhilJS SSR/SSG for fully resumable mutations.

Caching & Prefetching

- Use prefetch on routes or links to warm caches before navigation happens.
- Combine loaders with `createMutation` / `createQuery` for optimistic UI and deduped fetches.
- For large datasets, stream responses from loaders (e.g, via `ReadableStream`) to keep navigation snappy.

Next: read about [loading states](#) to show skeletons or progress indicators while loaders run.

Loading States

Loaders fetch data before a route renders, but you still may want to show progress for long requests or streaming content. PhilJS exposes loading information through signals and layout patterns.

Inline Loading Indicators

Because routes receive the navigation state via `useRouter()`, you can react to the current route's loading status.

```

import { useRouter } from '@philjs/router';

export function GlobalLoader() {
  const { route } = useRouter();

  if (!route || !route.pending) return null;
  return <div className="loading-bar" />;
}

```

The router populates `route.pending` when a loader/action is in flight. You can render this component inside your layout to show a top-level spinner or skeleton.

Suspense-like Patterns

Use signals to track fetch states for follow-up requests triggered inside a component.

```

export function ProfileRoute({ data }: { data: User }) {
  const details = signalUserDetails | null>(null);
  const loading = signal(true);

  effect(() => {
    loading.set(true);
    fetch(`/api/users/${data.id}/details`)
      .then((res) => res.json())
      .then((json) => {
        details.set(json);
        loading.set(false);
      });
  });

  return (
    <section>
      <h1>{data.name}</h1>
      {loading() ? <Skeleton /> : <DetailsPanel details={details()} />}
    </section>
  );
}

```

Partial Hydration & Streaming

Combine loaders with PhilJS' resumability to stream markup:

1. Return a `ReadableStream` from your loader.
2. In the component, consume the stream and update a signal.
3. During SSR, the markup streams to the client; hydration resumes where it left off.

```

loader: async ({ params }) => {
  const stream = fetchBigDataset(params.id).body!;
  return stream;
}

export function BigListRoute({ data }: { data: ReadableStream<Chunk> }) {
  const items = signal<Chunk[]>([[]]);

  effect(async () => {
    const reader = data.getReader();
    for (;;) {
      const { value, done } = await reader.read();
      if (done) break;
      items.set((prev) => [...prev, value]);
    }
  });

  return <List chunks={items()} />;
}

```

Optimistic UI with Actions

While actions execute, you can show pending indicators or optimistic updates:

```

export function CommentForm({ params, navigate }: RouteComponentProps) {
  const pending = signal(false);

  async function handleSubmit(event: SubmitEvent) {
    event.preventDefault();
    pending.set(true);
    const formData = new FormData(event.target as HTMLFormElement);
    await fetch(`/comments/${params.postId}?_action`, { method: 'POST', body: formData });
    pending.set(false);
    await navigate(`~/posts/${params.postId}`);
  }

  return (
    <form onSubmit={handleSubmit}>
      <textarea name="body" required />
      <button type="submit" disabled={pending()}>
        {pending() ? 'Posting...' : 'Post comment'}
      </button>
    </form>
  );
}

```

Skeleton Components

Use the layout's knowledge of `route.pending` to render skeletons for specific sections while child routes load:

```

export function DashboardLayout({ children }: { children: any }) {
  const { route } = useRouter();
  const isLoading = route?.pending;

  return (
    <div className="dashboard">
      <Sidebar />
      <main>
        {isLoading ? <DashboardSkeleton /> : children}
      </main>
    </div>
  );
}

```

Best Practices

- Avoid spinning indicators for fast requests; use skeletons for smoother perceived performance.
- Consider prefetch strategies (`prefetch: true`) to minimise loader latency altogether.
- Stream large datasets and progressively render them to keep navigation snappy.
- Use `route.error` (see the Error Handling guide) to display fallback UI when loaders fail.

Next, review [Error Handling](#) to pair loaders with robust fallback states.

Middleware

Protect routes, authenticate users, and modify requests with routing middleware.

What You'll Learn

- Route protection
- Authentication middleware

- Authorization checks
- Redirects and guards
- Request/response modification
- Best practices

What is Middleware?

Middleware runs before a route renders, allowing you to:

- Check authentication
- Verify permissions
- Redirect users
- Modify requests
- Log analytics
- Set headers

Layout-Based Middleware

Use layouts for route protection:

```
// src/pages/(protected)/Layout.tsx
import { useRouter } from '@philjs/router';
import { useUser } from '@/hooks/useUser';

export default function ProtectedLayout({ children }: { children: any }) {
  const router = useRouter();
  const user = useUser();

  // Redirect if not authenticated
  if (!user()) {
    router.replace('/login');
    return null;
  }

  return <div>{children}</div>;
}
```

All routes in `(protected)/` are now guarded!

Authentication Middleware

Basic Auth Check

```
// src/pages/auth/Layout.tsx
import { useRouter } from '@philjs/router';
import { useAuth } from '@/hooks/useAuth';
import { effect } from '@philjs/core';

export default function AuthLayout({ children }: { children: any }) {
  const router = useRouter();
  const { isAuthenticated, isLoading } = useAuth();

  effect(() => {
    if (!isLoading() && !isAuthenticated()) {
      router.replace('/login');
    }
  });

  if (isLoading()) {
    return <LoadingSpinner />;
  }

  if (!isAuthenticated()) {
    return null;
  }

  return <div>{children}</div>;
}
```

[Save Return URL](#)

```

import { useRouter, usePathname } from '@philjs/router';
import { useAuth } from '@/hooks/useAuth';

export default function AuthLayout({ children }: { children: any }) {
  const router = useRouter();
  const pathname = usePathname();
  const { isAuthenticated } = useAuth();

  if (!isAuthenticated()) {
    // Save where user was trying to go
    const returnUrl = encodeURIComponent(pathname);
    router.replace(`/login?returnUrl=${returnUrl}`);
    return null;
  }

  return <div>{children}</div>;
}

```

Then in login page:

```

// src/pages/Login.tsx
import { useRouter, useSearchParams } from '@philjs/router';

export default function Login() {
  const router = useRouter();
  const [searchParams] = useSearchParams();

  const handleLogin = async () => {
    await login();

    const returnUrl = searchParams.get('returnUrl') || '/dashboard';
    router.push(decodeURIComponent(returnUrl));
  };

  return <LoginForm onSubmit={handleLogin} />;
}

```

Authorization Middleware

Role-Based Access

```

// src/pages/(admin)/Layout.tsx
import { useRouter } from '@philjs/router';
import { useUser } from '@/hooks/useUser';

export default function AdminLayout({ children }: { children: any }) {
  const router = useRouter();
  const user = useUser();

  if (!user()) {
    router.replace('/login');
    return null;
  }

  if (user()!.role !== 'admin') {
    router.replace('/unauthorized');
    return null;
  }

  return (
    <div className="admin-layout">
      <AdminSidebar />
      {children}
    </div>
  );
}

```

Multiple Roles

```

interface LayoutProps {
  children: any;
  allowedRoles?: string[];
}

export default function RoleLayout({
  children,
  allowedRoles = ['admin', 'moderator']
}: LayoutProps) {
  const router = useRouter();
  const user = useUser();

  if (!user()) {
    router.replace('/login');
    return null;
  }

  const hasPermission = allowedRoles.includes(user().role);

  if (!hasPermission) {
    router.replace('/unauthorized');
    return null;
  }

  return <div>{children}</div>;
}

```

Permission-Based Access

```

// src/pages/(editor)/layout.tsx
import { useRouter } from '@philjs/router';
import { useUser } from '@/hooks/useUser';

export default function EditorLayout({ children }: { children: any }) {
  const router = useRouter();
  const user = useUser();

  if (!user()) {
    router.replace('/login');
    return null;
  }

  const canEdit = user()!.permissions.includes('edit:content');

  if (!canEdit) {
    router.replace('/forbidden');
    return null;
  }

  return <div>{children}</div>;
}

```

Subscription Middleware

Check subscription status:

```

// src/pages/(premium)/layout.tsx
import { useRouter } from '@philjs/router';
import { useUser, useSubscription } from '@/hooks';

export default function PremiumLayout({ children }: { children: any }) {
  const router = useRouter();
  const user = useUser();
  const subscription = useSubscription();

  if (!user()) {
    router.replace('/login');
    return null;
  }

  if (!subscription()?.active) {
    router.replace('/upgrade');
    return null;
  }

  return <div>{children}</div>;
}

```

Feature Flags

Gate features with flags:

```
// src/pages/(beta)/Layout.tsx
import { useRouter } from '@philjs/router';
import { useFeatureFlag } from '@/hooks/useFeatureFlag';

export default function BetaLayout({ children }: { children: any }) {
  const router = useRouter();
  const betaEnabled = useFeatureFlag('beta-features');

  if (!betaEnabled()) {
    router.replace('/');
    return null;
  }

  return (
    <div>
      <div className="beta-badge">Beta Feature</div>
      {children}
    </div>
  );
}
```

Request Logging

Track page views:

```
import { usePathname } from '@philjs/router';
import { effect } from '@philjs/core';

export default function AnalyticsLayout({ children }: { children: any }) {
  const pathname = usePathname();

  effect(() => {
    // Log page view
    analytics.track('page_view', {
      path: pathname,
      timestamp: Date.now()
    });
  });

  return <div>{children}</div>;
}
```

Rate Limiting

Client-side rate limit checks:

```
import { useRouter } from '@philjs/router';
import { signal, effect } from '@philjs/core';

const requestCount = signal(0);
const lastReset = signal(Date.now());

export default function RateLimitedLayout({ children }: { children: any }) {
  const router = useRouter();

  effect(() => {
    const now = Date.now();
    const elapsed = now - lastReset();

    // Reset every minute
    if (elapsed > 60000) {
      requestCount.set(0);
      lastReset.set(now);
    }

    // Increment counter
    requestCount.set(c => c + 1);

    // Check limit (60 requests per minute)
    if (requestCount() > 60) {
      router.replace('/rate-limited');
    }
  });

  return <div>{children}</div>;
}
```

Maintenance Mode

Show maintenance page:

```
// src/pages/Layout.tsx
import { signal } from '@philjs/core';

const maintenanceMode = signal(false);

export default function RootLayout({ children }: { children: any }) {
  if (maintenanceMode()) {
    return <MaintenancePage />;
  }

  return <div>{children}</div>;
}
```

Geo-Blocking

Restrict by location:

```
import { useRouter } from '@philjs/router';
import { signal, effect } from '@philjs/core';

const userCountry = signal<string | null>(null);

export default function GeoLayout({ children }: { children: any }) {
  const router = useRouter();

  effect(() => {
    // Fetch user location
    fetch('/api/geo')
      .then(r => r.json())
      .then(data => userCountry.set(data.country));
  });

  effect(() => {
    const country = userCountry();
    const blockedCountries = ['XX', 'YY']; // Country codes

    if (country && blockedCountries.includes(country)) {
      router.replace('/unavailable');
    }
  });

  if (!userCountry()) {
    return <LoadingSpinner />;
  }

  return <div>{children}</div>;
}
```

A/B Testing

Split traffic between variants:

```
import { signal, effect } from '@philjs/core';

const variant = signal<'A' | 'B' | null>(null);

export default function ABTestLayout({ children }: { children: any }) {
  effect(() => {
    // Assign variant once
    if (!variant()) {
      const assigned = Math.random() < 0.5 ? 'A' : 'B';
      variant.set(assigned);
    }
  });

  if (!variant()) return null;

  return (
    <div data-variant={variant()}>
      {children}
    </div>
  );
}
```

Middleware Composition

Combine multiple middleware checks:

```
// src/middleware/compose.ts
type MiddlewareFunction = (props: any) => any | null;

export function compose(...middlewares: MiddlewareFunction[]) {
  return function ComposedMiddleware({ children }: { children: any }) {
    let result = children;

    for (const middleware of middlewares) {
      result = middleware({ children: result });
      if (result === null) return null;
    }

    return result;
  };
}
```

Usage:

```
// src/pages/(protected)/Layout.tsx
import { compose } from '@/middleware/compose';
import { authMiddleware } from '@/middleware/auth';
import { roleMiddleware } from '@/middleware/role';
import { subscriptionMiddleware } from '@/middleware/subscription';

const ProtectedLayout = compose(
  authMiddleware,
  roleMiddleware(['admin', 'moderator']),
  subscriptionMiddleware
);

export default ProtectedLayout;
```

Page-Level Middleware

Apply middleware to specific pages:

```
// src/pages/admin/users.tsx
import { useRouter } from '@philjs/router';
import { useUser } from '@hooks/useUser';

export default function AdminUsers() {
  const router = useRouter();
  const user = useUser();

  // Page-specific check
  if (!user()?.permissions.includes('manage:users')) {
    router.replace('/unauthorized');
    return null;
  }

  return (
    <div>
      <h1>User Management</h1>
      <UserList />
    </div>
  );
}
```

Redirect Patterns

Authenticated to Dashboard

```
// src/pages/Login.tsx
import { useRouter } from '@philjs/router';
import { useUser } from '@hooks/useUser';
import { effect } from '@philjs/core';

export default function Login() {
  const router = useRouter();
  const user = useUser();

  // Redirect if already logged in
  effect(() => {
    if (user()) {
      router.replace('/dashboard');
    }
  });

  return <LoginForm />;
}
```

Role-Based Landing

```

import { useRouter } from '@philjs/router';
import { useUser } from '@hooks/useUser';
import { effect } from '@philjs/core';

export default function Home() {
  const router = useRouter();
  const user = useUser();

  effect(() => {
    if (!user()) return;

    // Redirect based on role
    switch (user().role) {
      case 'admin':
        router.replace('/admin/dashboard');
        break;
      case 'moderator':
        router.replace('/moderator/queue');
        break;
      default:
        router.replace('/dashboard');
    }
  });

  return <LoadingSpinner />;
}

```

Navigation Guards

Confirm before leaving:

```

import { useRouter } from '@philjs/router';
import { signal, effect } from '@philjs/core';

export default function EditPost() {
  const router = useRouter();
  const hasUnsavedChanges = signal(false);

  effect(() => {
    const handleBeforeNavigate = (e: any) => {
      if (hasUnsavedChanges()) {
        const confirmed = confirm('You have unsaved changes. Leave anyway?');
        if (!confirmed) {
          e.preventDefault();
        }
      }
    };
  });

  router.events.on('beforeNavigate', handleBeforeNavigate);

  return () => {
    router.events.off('beforeNavigate', handleBeforeNavigate);
  };
}

return <PostEditor onChange={() => hasUnsavedChanges.set(true)} />;
}

```

Best Practices

Use Layouts for Groups

```

// ⚡ Protect entire section
src/pages/
  (admin)/
    layout.tsx ← Auth check here
    users.tsx
    settings.tsx

// ⚡ Duplicate checks in every page
src/pages/
  admin-users.tsx ← Auth check
  admin-settings.tsx ← Auth check (duplicate)

```

Show Loading States

```
// 🚧 Show Loading while checking auth
if (isLoading()) {
  return <LoadingSpinner />;
}

// 🚧 Blank screen during check
if (!user()) {
  router.replace('/login');
  return null; // Flashes blank
}
```

Redirect Early

```
// 🚧 Check and redirect in Layout
export default function Layout({ children }) {
  if (!authorized()) {
    router.replace('/login');
    return null;
  }
  return children;
}

// 🚧 Check in every component
export default function Page() {
  if (!authorized()) { /* ... */ }
  return <div>Content</div>;
}
```

Use Effect for Side Effects

```
// 🚧 Redirect in effect
effect(() => {
  if (!user()) {
    router.replace('/login');
  }
});

// 🚧 Redirect during render (can cause issues)
if (!user()) {
  router.replace('/login');
}
```

Compose Middleware

```
// 🚧 Reusable middleware functions
const protected = compose(auth, subscription);

// 🚧 Duplicate logic
function Layout1() { /* auth + subscription */ }
function Layout2() { /* auth + subscription */ }
```

Complete Example

Multi-tier access control:

```
// src/middleware/auth.tsx
import { useRouter } from '@philjs/router';
import { useAuth } from '@hooks/useAuth';

export function RequireAuth({ children }: { children: any }) {
  const router = useRouter();
  const { user, isLoading } = useAuth();

  if (isLoading()) {
    return <LoadingSpinner />;
  }

  if (!user()) {
    router.replace('/login');
    return null;
  }

  return children;
}
```

```
// src/middleware/role.tsx
import { useRouter } from '@philjs/router';
import { useUser } from '@/hooks/useUser';

interface RoleProps {
  children: any;
  roles: string[];
}

export function RequireRole({ children, roles }: RoleProps) {
  const router = useRouter();
  const user = useUser();

  if (!user()) return null;

  if (!roles.includes(user()!.role)) {
    router.replace('/unauthorized');
    return null;
  }

  return children;
}
```

```
// src/middleware/subscription.tsx
import { useRouter } from '@philjs/router';
import { useSubscription } from '@/hooks/useSubscription';

export function RequireSubscription({ children }: { children: any }) {
  const router = useRouter();
  const subscription = useSubscription();

  if (!subscription()?.active) {
    router.replace('/upgrade');
    return null;
  }

  return children;
}
```

```
// src/pages/(premium-admin)/Layout.tsx
import { RequireAuth } from '@/middleware/auth';
import { RequireRole } from '@/middleware/role';
import { RequireSubscription } from '@/middleware/subscription';

export default function PremiumAdminLayout({ children }: { children: any }) {
  return (
    <RequireAuth>
      <RequireRole roles={['admin', 'superadmin']}>
        <RequireSubscription>
          <div className="premium-admin-layout">
            <AdminSidebar />
            {children}
          </div>
        </RequireSubscription>
      </RequireRole>
    </RequireAuth>
  );
}
```

Summary

You've learned:

- Layout-based middleware
 - Authentication checks
 - Authorization and roles
 - Permission-based access
 - Subscription gating
 - Feature flags
 - Request logging
 - Navigation guards
 - Middleware composition
 - Best practices
- Middleware keeps your routes secure and organized!

Next: [Error Handling](#) → Handle route errors gracefully

Parallel Routes

Parallel routes allow you to simultaneously render multiple pages in the same layout. This is perfect for dashboards, split views, and complex UIs that need to show multiple independent sections.

Basic Parallel Routes

Defining Parallel Routes

Create parallel routes using the `@slot` naming convention:

```
routes/
  dashboard/
    @analytics/
      index.tsx      → Analytics slot
    @activity/
      index.tsx     → Activity slot
    @settings/
      index.tsx      → Settings slot
    _layout.tsx      → Dashboard layout
```

Layout with Slots

```
// routes/dashboard/_layout.tsx
export default function DashboardLayout({ analytics, activity, settings }) {
  return (
    <div class="dashboard">
      <div class="main">
        <section class="analytics">
          {analytics}
        </section>

        <section class="activity">
          {activity}
        </section>
      </div>

      <aside class="settings">
        {settings}
      </aside>
    </div>
  );
}
```

Slot Components

```
// routes/dashboard/@analytics/index.tsx
export default function Analytics() {
  return (
    <div>
      <h2>Analytics</h2>
      <Chart data={salesData} />
    </div>
  );
}

// routes/dashboard/@activity/index.tsx
export default function Activity() {
  return (
    <div>
      <h2>Recent Activity</h2>
      <ActivityFeed items={activities} />
    </div>
  );
}

// routes/dashboard/@settings/index.tsx
export default function Settings() {
  return (
    <div>
      <h2>Settings</h2>
      <SettingsForm />
    </div>
  );
}
```

Navigation in Slots

Independent Navigation

Each slot can navigate independently.

```
// routes/dashboard/@analytics/index.tsx
export default function AnalyticsIndex() {
  return (
    <div>
      <nav>
        <Link href="/dashboard/@analytics/overview">Overview</Link>
        <Link href="/dashboard/@analytics/revenue">Revenue</Link>
        <Link href="/dashboard/@analytics/users">Users</Link>
      </nav>
    </div>
  );
}

// routes/dashboard/@analytics/revenue.tsx
export default function Revenue() {
  return <div>Revenue Analytics</div>;
}
```

Coordinated Navigation

Slots can coordinate navigation:

```
// routes/dashboard/@analytics/[metric].tsx
import { useParams, useNavigate } from '@philjs/router';

export default function MetricDetail() {
  const params = useParams();
  const navigate = useNavigate();

  const selectMetric = (metric: string) => {
    // Navigate both slots simultaneously
    navigate(`/dashboard/@analytics/${metric}/@details/${metric}`);
  };

  return <div>Viewing: {params.metric}</div>;
}
```

Loading States

Per-Slot Loading

Each slot can have its own loading state:

```
// routes/dashboard/@analytics/index.tsx
export const loader = createDataLoader(async () => {
  const data = await fetchAnalytics();
  return { data };
});

export default function Analytics({ data }) {
  return (
    <Suspense fallback={<AnalyticsSkeleton />}>
      <AnalyticsChart data={data} />
    </Suspense>
  );
}

// routes/dashboard/@activity/index.tsx
export const loader = createDataLoader(async () => {
  const feed = await fetchActivity();
  return { feed };
});

export default function Activity({ data }) {
  return (
    <Suspense fallback={<ActivitySkeleton />}>
      <ActivityList items={data.feed} />
    </Suspense>
  );
}
```

Streaming Parallel Data

```
// routes/dashboard/_layout.tsx
export const loader = createDataLoader(async () => {
  return {
    // Load in parallel
    analytics: fetchAnalytics(),
    activity: fetchActivity(),
    settings: fetchSettings()
  };
});

export default function DashboardLayout({ data, ...slots }) {
  return (
    <div class="dashboard">
      <Suspense fallback={<AnalyticsSkeleton />}>
        <Await resolve={data.analytics}>
          {((analytics) => slots.analytics({ data: analytics }))}
        </Await>
      </Suspense>
    </div>
  );
}
```

Conditional Slots

Show/Hide Slots

```
export default function DashboardLayout({ analytics, activity, settings }) {
  const showSettings = usePermission('settings.view');

  return (
    <div>
      {analytics}
      {activity}
      {showSettings() && settings}
    </div>
  );
}
```

Dynamic Slot Selection

```
import { useSearchParams } from '@philjs/router';

export default function DashboardLayout(slots) {
  const [params] = useSearchParams();
  const view = params.get('view') || 'default';

  return (
    <div>
      {view === 'analytics' && slots.analytics}
      {view === 'activity' && slots.activity}
      {view === 'split' && (
        <>
          {slots.analytics}
          {slots.activity}
        </>
      )}
    </div>
  );
}
```

Modal Routes

Modal as Parallel Route

```

// routes/photos/
//   @modal/
//     [id].tsx      → Photo modal
//     index.tsx     → Photo grid
//     _layout.tsx   → Layout with modal

// routes/photos/_layout.tsx
export default function PhotosLayout({ children, modal }) {
  return (
    <div>
      {children}
      {modal}
    </div>
  );
}

// routes/photos/index.tsx
export default function PhotoGrid() {
  return (
    <div class="grid">
      {photos.map(photo => (
        <Link key={photo.id} href={`/photos/@modal/${photo.id}`}>
          <img src={photo.thumbnail} />
        </Link>
      ))}
    </div>
  );
}

// routes/photos/@modal/[id].tsx
export default function PhotoModal({ params }) {
  const navigate = useNavigate();

  return (
    <div class="modal-overlay" onClick={() => navigate('/photos')}>
      <div class="modal">
        <img src={`/photos/${params.id}.jpg`} />
        <button onClick={() => navigate('/photos')}>Close</button>
      </div>
    </div>
  );
}

```

Split Views

Email Client Example

```

// routes/mail/
// @list/
//   index.tsx      -> Email list
//   @detail/
//     [id].tsx      -> Email detail
//   _layout.tsx

// routes/mail/_layout.tsx
export default function MailLayout({ list, detail }) {
  return (
    <div class="mail-client">
      <aside class="mail-list">
        {list}
      </aside>

      <main class="mail-detail">
        {detail || <div>Select an email</div>}
      </main>
    </div>
  );
}

// routes/mail@list/index.tsx
export default function MailList() {
  return (
    <ul>
      {emails.map(email => (
        <li key={email.id}>
          <a href={`/mail/@detail/${email.id}`}>
            {email.subject}
          </a>
        </li>
      ))}
    </ul>
  );
}

// routes/mail@detail/[id].tsx
export default function MailDetail({ params }) {
  const email = fetchEmail(params.id);

  return (
    <article>
      <h2>{email.subject}</h2>
      <p>From: {email.from}</p>
      <div>{email.body}</div>
    </article>
  );
}

```

Error Boundaries

Per-Slot Error Handling

```

// routes/dashboard@analytics/index.tsx
export default function Analytics() {
  return (
    <ErrorBoundary
      fallback={({error}) => (
        <div>
          <h3>Failed to load analytics</h3>
          <p>{error.message}</p>
        </div>
      )}
    >
      <AnalyticsContent />
    </ErrorBoundary>
  );
}

```

Graceful Degradation

```

export default function DashboardLayout({ analytics, activity }) {
  return (
    <div>
      <ErrorBoundary
        fallback={() => <div>Analytics unavailable</div>}
      >
        {analytics}
      </ErrorBoundary>

      <ErrorBoundary
        fallback={() => <div>Activity unavailable</div>}
      >
        {activity}
      </ErrorBoundary>
    </div>
  );
}

```

State Management

Shared State Between Slots

```

import { createContext, useContext, signal } from '@philjs/core';

const DashboardContext = createContext();

export default function DashboardLayout({ analytics, activity }) {
  const selectedMetric = signal('revenue');
  const dateRange = signal({ start: new Date(), end: new Date() });

  return (
    <DashboardContext.Provider value={{ selectedMetric, dateRange }}>
      <div>
        {analytics}
        {activity}
      </div>
    </DashboardContext.Provider>
  );
}

// In slot component
function Analytics() {
  const { selectedMetric, dateRange } = useContext(DashboardContext);

  return (
    <div>
      Showing {selectedMetric()} from {dateRange().start} to {dateRange().end}
    </div>
  );
}

```

Best Practices

□ Do: Use for Independent UI Sections

```

// ✅ Good - parallel routes for independent sections
<div class="dashboard">
  <div class="left">{analytics}</div>
  <div class="right">{activity}</div>
</div>

```

□ Do: Handle Missing Slots

```

// ✅ Good - provide defaults
export default function Layout({ slot1, slot2 }) {
  return (
    <div>
      {slot1 || <DefaultSlot1 />}
      {slot2 || <DefaultSlot2 />}
    </div>
  );
}

```

□ Do: Keep Slots Independent

```

// ✅ Good - independent data loading
// @analytics/index.tsx
export const loader = () => fetchAnalytics();

// @activity/index.tsx
export const loader = () => fetchActivity();

```

□ Don't: Overuse Parallel Routes

```
// ☹ Bad - too many parallel routes
<Layout>
  slot1={...}
  slot2={...}
  slot3={...}
  slot4={...}
  slot5={...}
/>

// ☺ Good - simpler structure
<Layout>
  <Component1 />
  <Component2 />
</Layout>
```

Next Steps

- [Intercepting Routes](#) - Intercept navigation
- [Layouts](#) - Advanced layouts
- [View Transitions](#) - Smooth transitions
- [Data Loading](#) - Load data for parallel routes

Tip: Parallel routes are perfect for dashboards where multiple sections need to load and navigate independently.

Warning: Too many parallel routes can make your routing complex. Use them judiciously.

Note: Each parallel route can have its own loading state, error boundary, and navigation.

Intercepting Routes

Intercepting routes allow you to load a route from another part of your application while keeping the user in their current context. This is perfect for modals, drawers, and overlays that maintain the background page state.

Basic Interception

Intercept Convention

Use `(.)` notation to intercept routes:

- `(.)` - same level
- `(..)` - one level up
- `(..)(..)` - two levels up
- `(...)` - from root

```
routes/
  photos/
    [id]/
      index.tsx      → /photos/123
    (.)[id]/
      index.tsx      → Intercepts /photos/123
    index.tsx        → /photos
```

Photo Modal Example

```

// routes/photos/index.tsx
export default function PhotoGrid() {
  const photos = signal([/* ... */]);

  return (
    <div class="grid">
      {photos().map(photo => (
        <Link key={photo.id} href={`/photos/${photo.id}`}>
          <img src={photo.thumbnail} alt={photo.title} />
        </Link>
      ))}
    </div>
  );
}

// routes/photos/(.){id}/index.tsx - Intercepts the route
export default function PhotoModal({ params }) {
  const navigate = useNavigate();

  return (
    <div class="modal-backdrop" onClick={() => navigate('/photos')}>
      <div class="modal" onClick={(e) => e.stopPropagation()}>
        <img src={`/api/photos/${params.id}`} alt="Photo" />
        <button onClick={() => navigate('/photos')}>Close</button>
      </div>
    </div>
  );
}

// routes/photos/{id}/index.tsx - Direct navigation shows full page
export default function PhotoPage({ params }) {
  return (
    <div class="photo-page">
      <img src={`/api/photos/${params.id}`} alt="Photo" />
      <Link href="/photos">Back to Grid</Link>
    </div>
  );
}

```

Login Modal

Intercept Login Route

```

// routes/(.)login/index.tsx
import { useNavigate } from '@philjs/router';

export default function LoginModal() {
  const navigate = useNavigate();
  const [email, setEmail] = signal('');
  const [password, setPassword] = signal('');

  const handleSubmit = async (e) => {
    e.preventDefault();
    await login(email(), password());
    navigate(-1); // Go back to previous page
  };

  return (
    <div class="modal-backdrop">
      <div class="modal">
        <h2>Login</h2>
        <form onSubmit={handleSubmit}>
          <input
            type="email"
            value={email()}
            onChange={(e) => setEmail(e.target.value)}
            placeholder="Email"
          />
          <input
            type="password"
            value={password()}
            onChange={(e) => setPassword(e.target.value)}
            placeholder="Password"
          />
          <button type="submit">Login</button>
        </form>
        <button onClick={() => navigate(-1)}>Cancel</button>
      </div>
    </div>
  );
}

// routes/login/index.tsx - Full page login
export default function LoginPage() {
  return (
    <div class="login-page">
      <h1>Login</h1>
      <LoginForm />
    </div>
  );
}

```

E-Commerce Product Quick View

Quick View Modal

```

// routes/products/(.)[id]/index.tsx
export const loader = createDataLoader(async ({ params }) => {
  const product = await db.products.findById(params.id);
  return { product };
});

export default function ProductQuickView({ data }) {
  const navigate = useNavigate();

  return (
    <div class="quick-view-modal">
      <div class="modal-content">
        <img src={data.product.image} alt={data.product.name} />
        <h2>{data.product.name}</h2>
        <p>{data.product.description}</p>
        <p class="price">${data.product.price}</p>

        <div class="actions">
          <button>Add to Cart</button>
          <a href={`/products/${data.product.id}`}>
            View Full Details
          </a>
        </div>
      </div>
    </div>
  );
}

// routes/products/[id]/index.tsx - Full product page
export default function ProductPage({ data }) {
  return (
    <div class="product-page">
      <ProductGallery images={data.product.images} />
      <ProductDetails product={data.product} />
      <ProductReviews productId={data.product.id} />
      <RelatedProducts category={data.product.category} />
    </div>
  );
}

```

Share Dialog

Intercept Share Route

```

// routes/posts/[id]/(..)share/index.tsx
export default function ShareDialog({ params }) {
  const navigate = useNavigate();
  const url = `${window.location.origin}/posts/${params.id}`;

  const share = (platform: string) => {
    const urls = {
      twitter: `https://twitter.com/intent/tweet?url=${encodeURIComponent(url)}`,
      facebook: `https://facebook.com/sharer/sharer.php?u=${encodeURIComponent(url)}`,
      linkedin: `https://linkedin.com/sharing/share-offsite/?url=${encodeURIComponent(url)}`
    };

    window.open(urls[platform], '_blank');
    navigate(-1);
  };

  return (
    <div class="share-dialog">
      <h3>Share this post</h3>
      <button onClick={() => share('twitter')}>Share on Twitter</button>
      <button onClick={() => share('facebook')}>Share on Facebook</button>
      <button onClick={() => share('linkedin')}>Share on LinkedIn</button>
      <button onClick={() => navigator.clipboard.writeText(url)}>
        Copy Link
      </button>
      <button onClick={() => navigate(-1)}>Cancel</button>
    </div>
  );
}

```

Settings Drawer

Side Drawer Interception

```

// routes/(..)settings/index.tsx
export default function SettingsDrawer() {
  const navigate = useNavigate();
  const [theme, setTheme] = signal('light');

  return (
    <>
      <div class="drawer-backdrop" onClick={() => navigate(-1)} />
      <div class="drawer">
        <h2>Quick Settings</h2>

        <div class="setting">
          <label>Theme</label>
          <select value={theme()} onChange={(e) => setTheme(e.target.value)}>
            <option value="light">Light</option>
            <option value="dark">Dark</option>
            <option value="auto">Auto</option>
          </select>
        </div>

        <Link href="/settings">Full Settings</Link>
        <button onClick={() => navigate(-1)}>Close</button>
      </div>
    </>
  );
}

// routes/settings/index.tsx - Full settings page
export default function SettingsPage() {
  return (
    <div class="settings-page">
      <h1>Settings</h1>
      <SettingsSections />
    </div>
  );
}

```

Preserving State

Background State Management

```

// routes/feed/index.tsx
export default function Feed() {
  const scrollPosition = signal(0);
  const selectedFilter = signal('all');

  const handleScroll = (e) => {
    scrollPosition.set(e.target.scrollTop);
  };

  return (
    <div onScroll={handleScroll}>
      <FilterBar value={selectedFilter()} onChange={selectedFilter.set} />

      <{posts().map(post => (
        <Link key={post.id} href={`feed/..)posts/${post.id}`}>
          <PostCard post={post} />
        </Link>
      ))}>
    </div>
  );
}

// When returning from modal, state is preserved:
// - Scroll position maintained
// - Filter selection preserved
// - Feed data cached

```

Nested Interceptions

Multi-Level Interception

```
// routes/dashboard/(..)analytics/(.)chart/[id]/index.tsx
export default function ChartModal({ params }) {
  const navigate = useNavigate();

  return (
    <div class="modal">
      <h2>Chart Details</h2>
      <Chart id={params.id} />
      <button onClick={() => navigate(-2)}>
        Back to Dashboard
      </button>
    </div>
  );
}
```

Conditional Interception

Device-Based Interception

```
// routes/products/(.)[id]/index.tsx
export default function ProductRoute({ params }) {
  const isMobile = useMediaQuery('(max-width: 768px)');

  if (isMobile()) {
    // Mobile: show full page
    return <Redirect to={`/products/${params.id}`} />;
  }

  // Desktop: show modal
  return <ProductQuickView productId={params.id} />;
}
```

Best Practices

Do: Preserve Background State

```
// ✅ Good - use interception to keep state
<Link href="/(..)photo/123">View Photo</Link>
// Background page state is preserved

// ❌ Bad - loses state
<Link href="/photo/123">View Photo</Link>
// Background page unmounts
```

Do: Provide Full Page Alternative

```
// ✅ Good - both modal and full page
routes/
  products/
    (.)[id]/index.tsx    // Modal
    [id]/index.tsx      // Full page
```

Do: Handle Direct Navigation

```
// ✅ Good - handle both cases
export default function PhotoRoute({ params }) {
  const fromGrid = useNavigationSource() === '/photos';

  if (fromGrid) {
    return <PhotoModal id={params.id} />;
  }

  return <PhotoPage id={params.id} />;
}
```

Don't: Intercept Critical Flows

```
// ❌ Bad - don't intercept important flows
routes/
  (.)checkout          // Payment should be full page

// ✅ Good - use modals for quick views only
routes/
  (.)product-preview/ // Preview in modal
  checkout/           // Full checkout page
```

Next Steps

- [Parallel Routes](#) - Render multiple routes
- [Modal Patterns](#) - Modal best practices
- [View Transitions](#) - Smooth transitions

- [Navigation](#) - Advanced navigation

□ **Tip:** Use intercepting routes for quick views that should preserve the background page state.
 □ **Warning:** Always provide a full-page alternative for intercepted routes in case users refresh or share the URL.
 ⓘ **Note:** Intercepted routes maintain scroll position and state of the background page, providing a seamless UX.

Smart Preloading

PhilJS includes an intelligent preloading system that predicts user navigation and preloads routes before they're clicked, dramatically improving perceived performance.

Table of Contents

- [Overview](#)
- [Quick Start](#)
- [Preload Strategies](#)
- [API Reference](#)
- [Intent Prediction](#)
- [Complete Examples](#)
- [Best Practices](#)
- [Advanced Usage](#)

Overview

Smart preloading uses machine learning and behavioral analysis to predict which route a user will navigate to next:

- **Intent Prediction:** Analyzes mouse trajectory and velocity
- **Hover Detection:** Preloads on link hover with configurable delay
- **Viewport Visibility:** Preloads links about to enter viewport
- **Navigation History:** Learns from user behavior patterns
- **Priority Queuing:** Manages concurrent preloads efficiently
- **Adaptive Loading:** Respects network conditions

Why Smart Preloading?

Traditional preloading is either too aggressive (wasting bandwidth) or too passive (not fast enough). PhilJS smart preloading:

- **Reduces perceived load time by 60-80%** - Routes feel instant
- **Saves bandwidth** - Only preloads high-probability navigations
- **Improves Core Web Vitals** - Better FID and LCP scores
- **Works automatically** - No manual configuration needed

```
import { initSmartPreloader } from '@philjs/router';

// Initialize with intent prediction
const preloader = initSmartPreloader({
  strategy: 'intent',          // Use ML-based prediction
  intentThreshold: 0.6,        // 60% confidence minimum
  maxConcurrent: 3             // Max 3 concurrent preloads
});

// PreLoading happens automatically!
// When mouse moves toward a Link, it preloads
```

Quick Start

1. Initialize Preloader

```
import { initSmartPreloader } from '@philjs/router';

// Basic setup with intelligent defaults
const preloader = initSmartPreloader();

// Or customize
const preloader = initSmartPreloader({
  strategy: 'intent',          // Prediction strategy
  hoverDelay: 50,               // ms before preLoad on hover
  intentThreshold: 0.6,         // Confidence threshold (0-1)
  maxConcurrent: 3,             // Max concurrent requests
  priority: 'auto'             // Priority: 'high' | 'low' | 'auto'
});
```

2. Automatic Preloading

Smart preloading works automatically once initialized. Links are preloaded based on the configured strategy:

```

function Navigation() {
  return (
    <nav>
      /* These will be preloaded automatically based on user intent */
      <Link to="/products">Products</Link>
      <Link to="/about">About</Link>
      <Link to="/contact">Contact</Link>
    </nav>
  );
}

```

3. Manual Control

You can also manually control preloading:

```

import { usePreload, getSmartPreloader } from '@philjs/router';

function ProductCard({ product }: { product: Product }) {
  // Preload on component mount
  const triggerPreload = usePreload('/products/${product.id}', {
    strategy: 'manual',
    priority: 'high'
  });

  return (
    <div
      onClick={() => navigate(`/products/${product.id}`)}
      onMouseEnter={triggerPreload} // Preload on hover
    >
      <h3>{product.name}</h3>
    </div>
  );
}

```

Preload Strategies

PhilJS supports five preloading strategies:

1. Intent Prediction ('intent')

Best for: General navigation (default)

Uses machine learning to predict click intent based on: - Mouse position relative to link - Mouse velocity and direction - Hover duration - Historical navigation patterns

```

initSmartPreloader({
  strategy: 'intent',
  intentThreshold: 0.6 // 60% confidence minimum
});

// Automatic preloading when confidence threshold met
// No configuration needed - just works!

```

2. Hover ('hover')

Best for: Navigation menus, product cards

Preloads when mouse hovers over a link, with configurable delay:

```

initSmartPreloader({
  strategy: 'hover',
  hoverDelay: 50 // Wait 50ms before preloading
});

```

3. Visible ('visible')

Best for: Long pages with many links

Preloads when link enters viewport (or is about to):

```

initSmartPreloader({
  strategy: 'visible'
  // Preloads 50px before entering viewport
});

```

4. Eager ('eager')

Best for: Critical navigation (homepage links)

Preloads immediately on page load:

```

initSmartPreloader({
  strategy: 'eager'
});

// Or per-link:
<Link to="/important" data-preload="eager">
  Important Page
</Link>

```

5. Manual ('manual')

Best for: Conditional preloading

Only preloads when explicitly triggered:

```
const preloader = getSmartPreloader();

// Trigger manually
preloader?.preload('/route', {
  strategy: 'manual',
  priority: 'high'
});
```

API Reference

SmartPreloader Class

The main preloading manager.

Constructor

```
new SmartPreloader(options?: PreloadOptions)
```

PreloadOptions:

```
type PreloadOptions = {
  strategy?: PreloadStrategy;           // 'hover' | 'visible' | 'intent' | 'eager' | 'manual'
  hoverDelay?: number;                  // ms delay for hover strategy
  intentThreshold?: number;            // 0-1, confidence for intent strategy
  maxConcurrent?: number;              // max concurrent preloads
  priority?: 'high' | 'low' | 'auto';   // fetch priority
};
```

Methods

```
register(element: HTMLAnchorElement, options?: PreloadOptions): void
```

Register a link for smart preloading.

Example:

```
const preloader = new SmartPreloader();

const link = document.querySelector('a[href="/products"]');
preloader.register(link, {
  strategy: 'hover',
  hoverDelay: 100
});

preload(url: string, options?: { strategy: PreloadStrategy; priority?: string }): Promise<void>
```

Preload a specific URL.

Example:

```
await preloader.preload('/products/123', {
  strategy: 'manual',
  priority: 'high'
});
```

```
recordNavigation(path: string): void
```

Record navigation for history-based prediction.

Example:

```
preloader.recordNavigation('/products');
preloader.recordNavigation('/products/123');
// Future navigations from /products will prioritize /products/123
```

```
getStats(): object
```

Get preloading statistics.

Example:

```
const stats = preloader.getStats();
console.log(`Loaded: ${stats.loaded}`);
console.log(`Loading: ${stats.loading}`);
console.log(`Queued: ${stats.queued}`);
```

```
clear(): void
```

Clear all preload data and queues.

```
destroy(): void
```

Clean up and remove all listeners.

Helper Functions

```
initSmartPreloader(options?: PreloadOptions): SmartPreloader
```

Initialize global smart preloader.

Example:

```
const preloader = initSmartPreloader({
  strategy: 'intent',
  maxConcurrent: 3
});
```

`getSmartPreloader(): SmartPreloader | null`

Get global preloader instance.

Example:

```
const preloader = getSmartPreloader();
if (preloader) {
  preloader.preload('/route');
```

`usePreload(href: string, options?: PreloadOptions): () => void`

React-style hook for preloading (returns trigger function).

Example:

```
function ProductCard({ id }: { id: number }) {
  const triggerPreload = usePreload('/products/${id}', {
    strategy: 'manual'
  });

  return (
    <div onMouseEnter={triggerPreload}>
      Product {id}
    </div>
  );
}
```

`preloadLink(element: HTMLAnchorElement, options?: PreloadOptions): () => void`

Directive for links (returns cleanup function).

Example:

```
const link = document.querySelector('a');
const cleanup = preloadLink(link, { strategy: 'hover' });

// Later: cleanup();
```

Intent Prediction

`calculateClickIntent(mousePos: {x, y}, mouseVelocity: {x, y}, linkBounds: DOMRect): number`
Calculate click intent probability (0-1).

Example:

```
import { calculateClickIntent } from '@philjs/router';

const intent = calculateClickIntent(
  { x: 100, y: 200 },           // Mouse position
  { x: 5, y: -3 },             // Mouse velocity
  link.getBoundingClientRect()
);

console.log(`Click probability: ${intent * 100}%)`);
```

`predictNextRoute(currentPath: string, visitHistory: string[]): Map<string, number>`

Predict next navigation based on history.

Returns: Map of routes to probabilities

Example:

```
import { predictNextRoute } from '@philjs/router';

const predictions = predictNextRoute('/products', [
  '/products',
  '/products/123',
  '/products',
  '/products/456',
  '/products',
  '/products/123' // Visited twice after /products
]);

// Results: { '/products/123': 0.66, '/products/456': 0.33 }
```

Intent Prediction

How It Works

Intent prediction uses a sophisticated algorithm:

1. **Distance Score** (40% weight)

- Closer mouse = higher score
 - Normalized to 0-1
2. **Direction Score** (60% weight)
- Mouse moving toward link = higher score
 - Uses vector dot product (cosine similarity)
3. **Historical Patterns** (bonus)
- Learn from past navigation
 - Build transition probability matrix

```
// Pseudocode
intent = distanceScore * 0.4 + directionScore * 0.6 + historyBonus

if (intent >= threshold) {
  preload(url);
}
```

Tuning Intent Threshold

```
// Conservative (fewer preloads, saves bandwidth)
initSmartPreloader({
  strategy: 'intent',
  intentThreshold: 0.8 // 80% confidence
});

// Aggressive (more preloads, faster navigation)
initSmartPreloader({
  strategy: 'intent',
  intentThreshold: 0.4 // 40% confidence
};

// Balanced (default)
initSmartPreloader({
  strategy: 'intent',
  intentThreshold: 0.6 // 60% confidence
});
```

Complete Examples

Example 1: E-Commerce Product List

```
import { initSmartPreloader } from '@philjs/router';
import { signal } from '@philjs/core';

// Initialize with intent prediction
const preloader = initSmartPreloader({
  strategy: 'intent',
  intentThreshold: 0.6,
  maxConcurrent: 5
});

function ProductGrid() {
  const products = signal<Product[]>([]);

  effect(async () => {
    const data = await fetch('/api/products').then(r => r.json());
    products.set(data);
  });

  return (
    <div class="product-grid">
      {products().map(product => (
        <Link
          to={`/products/${product.id}`}
          class="product-card"
        >
          <img src={product.image} alt={product.name} />
          <h3>{product.name}</h3>
          <p>${product.price}</p>
        </Link>
      )));
    </div>
  );
}

// Links automatically preload when user shows intent!
```

Example 2: Navigation Menu with Hover

```

import { initSmartPreloader } from '@philjs/router';

const preloader = initSmartPreloader({
  strategy: 'hover',
  hoverDelay: 100 // 100ms delay
});

function Navigation() {
  const menuItems = [
    { path: '/products', label: 'Products' },
    { path: '/about', label: 'About' },
    { path: '/blog', label: 'Blog' },
    { path: '/contact', label: 'Contact' }
  ];
  return (
    <nav>
      {menuItems.map(item => (
        <Link to={item.path}>
          {item.label}
        </Link>
      ))}
    </nav>
  );
}

// Preloads 100ms after hovering each link

```

Example 3: Conditional Preloading

```

import { usePreload } from '@philjs/router';
import { signal } from '@philjs/core';

function ArticleCard({ article }: { article: Article }) {
  const isHovered = signal(false);
  const triggerPreload = usePreload('/articles/${article.id}', {
    strategy: 'manual',
    priority: article.featured ? 'high' : 'low'
  });

  const handleMouseEnter = () => {
    isHovered.set(true);
  }

  // Only preload if article is popular or featured
  if (article.views > 1000 || article.featured) {
    triggerPreload();
  }
}

return (
  <div
    class="article-card"
    onMouseEnter={handleMouseEnter}
    onMouseLeave={() => isHovered.set(false)}
  >
    <h3>{article.title}</h3>
    <p>{article.excerpt}</p>
    {isHovered() && article.views > 1000 && (
      <span class="badge">Popular - Preloading...</span>
    )}
  </div>
);
}

```

Example 4: Priority Queue

```

import { initSmartPreloader, getSmartPreloader } from '@philjs/router';

// Initialize with max 3 concurrent preloads
const preloader = initSmartPreloader({
  strategy: 'manual',
  maxConcurrent: 3
});

function Dashboard() {
  effect(() => {
    // High priority: Critical pages
    preloader.preload('/dashboard/analytics', {
      strategy: 'manual',
      priority: 'high'
    });

    // Low priority: Less important pages
    preloader.preload('/dashboard/settings', {
      strategy: 'manual',
      priority: 'low'
    });

    preloader.preload('/dashboard/help', {
      strategy: 'manual',
      priority: 'low'
    });

    // High priority routes load first!
  });
}

return <div>Dashboard</div>;
}

```

Example 5: History-Based Prediction

```

import { initSmartPreloader } from '@philjs/router';

const preloader = initSmartPreloader({
  strategy: 'intent'
});

// Record navigation for learning
window.addEventListener('popstate', () => {
  preloader.recordNavigation(window.location.pathname);
});

// After user navigates:
// /products -> /products/123 (3 times)
// /products -> /products/456 (1 time)

// Next time on /products:
// /products/123 has 75% probability (preloads first)
// /products/456 has 25% probability (preloads second)

```

Example 6: Network-Aware Preloading

```

import { initSmartPreloader } from '@philjs/router';

// Detect network conditions
const connection = (navigator as any).connection;
const isSlow = connection?.effectiveType === '2g' || connection?.effectiveType === '3g';

const preloader = initSmartPreloader({
  strategy: isSlow ? 'manual' : 'intent',
  maxConcurrent: isSlow ? 1 : 3,
  intentThreshold: isSlow ? 0.8 : 0.6
});

// Adapts to network:
// - Slow: Only preload with high confidence
// - Fast: Aggressive preloading

```

Best Practices

1. Choose the Right Strategy

```
// ⚡ Use intent for general navigation
initSmartPreloader({ strategy: 'intent' });

// ⚡ Use hover for menus
initSmartPreloader({ strategy: 'hover', hoverDelay: 50 });

// ⚡ Use visible for long lists
initSmartPreloader({ strategy: 'visible' });

// ⚡ Use eager for critical pages only
<Link to="/checkout" data-preload="eager">Checkout</Link>

// ⚡ Don't use eager for everything (wastes bandwidth)
```

2. Limit Concurrent Preloads

```
// Good - respects bandwidth
initSmartPreloader({ maxConcurrent: 3 });

// ⚡ Bad - too aggressive
initSmartPreloader({ maxConcurrent: 10 });
```

3. Tune Intent Threshold

```
// For bandwidth-sensitive:
initSmartPreloader({ intentThreshold: 0.8 }); // High confidence

// For speed-sensitive:
initSmartPreloader({ intentThreshold: 0.5 }); // Lower confidence

// Balanced default:
initSmartPreloader({ intentThreshold: 0.6 });
```

4. Record Navigation History

```
import { getSmartPreloader } from '@philjs/router';

// Record every navigation
window.addEventListener('popstate', () => {
  getSmartPreloader()?.recordNavigation(location.pathname);
});

// Better predictions over time!
```

5. Monitor Performance

```
const preloader = getSmartPreloader();

setInterval(() => {
  const stats = preloader?.getStats();
  console.log('Preload stats:', stats);

  // Alert if queue is growing
  if (stats && stats.queued > 10) {
    console.warn('Preload queue is large - reduce maxConcurrent');
  }
}, 5000);
```

Advanced Usage

Custom Intent Algorithm

```

import { SmartPreloader } from '@philjs/router';

class CustomPreloader extends SmartPreloader {
  protected calculatePriority(url: string, options: any): number {
    let score = 50; // Base priority

    // Boost priority for certain routes
    if (url.includes('/checkout')) {
      score += 40;
    }

    // Reduce for external routes
    if (url.startsWith('http')) {
      score -= 30;
    }
  }

  return Math.max(0, Math.min(100, score));
}

const preloader = new CustomPreloader({
  strategy: 'intent'
});

```

Integration with Analytics

```

import { getSmartPreloader } from '@philjs/router';

const preloader = getSmartPreloader();

// Track preload accuracy
let preloaded = new Set<string>();
let navigated = new Set<string>();

preloader?.['queue'].forEach(item => {
  preloaded.add(item.url);
});

window.addEventListener('popstate', () => {
  const url = location.pathname;
  navigated.add(url);

  // Calculate hit rate
  const hits = Array.from(navigated).filter(u => preloaded.has(u)).length;
  const hitRate = hits / navigated.size;

  console.log(`Preload accuracy: ${((hitRate * 100).toFixed(0))}%`);

  // Send to analytics
  analytics.track('preload_accuracy', { hitRate });
});

```

Prefetch vs Preload

```

// Prefetch: Lower priority, for future navigation
<link rel="prefetch" href="/next-page" />

// PreLoad: Higher priority, for current page resources
<link rel="preload" href="/critical.css" as="style" />

// PhilJS Smart Preloader uses prefetch by default
// But can switch based on priority:
const preloader = new SmartPreloader();

preloader.preload('/critical-route', {
  strategy: 'manual',
  priority: 'high' // Uses preload, not prefetch
});

```

Disable on Mobile

```

const isMobile = /iPhone|iPad|Android/i.test(navigator.userAgent);

const preloader = initSmartPreloader({
  strategy: isMobile ? 'manual' : 'intent',
  maxConcurrent: isMobile ? 1 : 3
});

// Conservative on mobile to save data

```

Related Documentation

- [View Transitions](#) - Smooth page transitions
- [Route Discovery](#) - Auto-discover routes

- [Performance](#) - Performance optimization
- [Code Splitting](#) - Split routes

Troubleshooting

Issue: Too Many Preloads

Solution: Reduce maxConcurrent or increase intentThreshold:

```
initSmartPreloader({
  maxConcurrent: 2,           // Reduce from 3
  intentThreshold: 0.7       // Increase from 0.6
});
```

Issue: Preloads Not Working

Check: Is preloader initialized?

```
const preloader = getSmartPreloader();
if (!preloader) {
  console.error('Preloader not initialized!');
  initSmartPreloader();
}
```

Issue: Wrong Routes Preloading

Solution: Train with navigation history:

```
// Record actual navigations
window.addEventListener('popstate', () => {
  preloader?.recordNavigation(location.pathname);
});

// After ~50 navigations, predictions improve significantly
```

Next Steps: - Explore [View Transitions](#) for animations - Learn about [Route Discovery](#) - Optimize with [Code Splitting](#)

Route Discovery

PhilJS route discovery turns files and route metadata into a validated route manifest. This chapter explains how discovery works, how to control it, and how to keep it fast in large codebases.

How discovery works

1. **Scan** route roots (usually `src/routes`) for route files and folders.
2. **Parse** route patterns from filenames and folder structure.
3. **Read metadata** (route config exports, loader/action signatures, and middleware hints).
4. **Build a manifest** for the router and the build pipeline.
5. **Validate** collisions, missing layouts, and invalid parameters early.

Discovery runs in dev (fast, incremental) and at build time (full, deterministic).

Naming conventions

Common patterns:

- `index.tsx` maps to the folder path.
- `[id].tsx` or `[...path].tsx` express params and catch-all.
- `_layout.tsx` or `layout.tsx` defines a layout boundary.
- `route.ts` or `+route.ts` can be used for metadata-only routes.

Use consistent naming so the manifest is predictable. Avoid mixing multiple param styles in a single tree.

Controlling discovery

Use explicit config to scope or override discovery:

```
export default {
  routes: {
    rootDir: 'src/routes',
    ignore: ['**/*.test.tsx', '**/_fixtures_/**'],
    reservedFiles: ['_layout.tsx', 'route.ts'],
  },
};
```

Tips:

- Keep fixtures and stories out of the route tree.
- Avoid deep nesting unless layouts are required.
- Prefer fewer dynamic segments for better link predictability.

Validation and diagnostics

Discovery should surface errors early:

- Duplicate routes (`/docs` defined in multiple files).
- Param conflicts (`[id]` vs `[slug]` under the same parent).
- Missing layout boundaries where a layout is required.
- Unreachable routes due to guard or group configuration.

Pair discovery with `philjs inspect` to audit the final manifest.

Route discovery and preloading

Smart preloading relies on discovery output. For high-traffic routes:

- Mark routes with preload hints.
- Keep loader data small and cacheable.
- Avoid creating preloading chains that pull large graphs of routes.

Checklist

- Route root is explicit and stable across environments.
- Reserved filenames are consistent across the app.
- Route collisions are detected in CI.
- Discovery output is checked with `philjs inspect`.

View Transitions

Create smooth, app-like animations between routes using the View Transitions API.

What You'll Learn

- View Transitions API
- Route transitions
- Custom animations
- Shared element transitions
- Fallback animations
- Best practices

What are View Transitions?

View Transitions provide native browser animations between pages, creating smooth visual continuity during navigation.

Benefits: - Native browser API (performant) - Smooth cross-fade by default - Shared element transitions - Accessible (respects `prefers-reduced-motion`)

Basic Transitions

Enable View Transitions

```
// src/App.tsx
import { Router } from '@philjs/router';

export default function App() {
  return (
    <Router
      viewTransitions={true} // Enable view transitions
    />
  );
}
```

That's it! Routes now cross-fade smoothly.

Manual Transition

```

import { useRouter } from '@philjs/router';

function Navigation() {
  const router = useRouter();

  const navigateWithTransition = async (href: string) => {
    if (document.startViewTransition) {
      await document.startViewTransition(() => {
        router.push(href);
      });
    } else {
      router.push(href);
    }
  };

  return (
    <nav>
      <button onClick={() => navigateWithTransition('/about')}>
        About
      </button>
    </nav>
  );
}

```

Custom Animations

Slide Transition

```

/* Default cross-fade */
::view-transition-old(root),
::view-transition-new(root) {
  animation-duration: 0.3s;
}

/* Slide from right */
@keyframes slide-from-right {
  from {
    transform: translateX(100%);
  }
}

@keyframes slide-to-left {
  to {
    transform: translateX(-100%);
  }
}

::view-transition-old(root) {
  animation: 0.3s ease-out both slide-to-left;
}

::view-transition-new(root) {
  animation: 0.3s ease-out both slide-from-right;
}

```

Fade and Scale

```

@keyframes fade-in {
  from {
    opacity: 0;
  }
}

@keyframes fade-out {
  to {
    opacity: 0;
  }
}

@keyframes scale-up {
  from {
    transform: scale(0.95);
  }
}

::view-transition-old(root) {
  animation: 0.25s ease-out both fade-out;
}

::view-transition-new(root) {
  animation: 0.25s ease-out both fade-in, 0.25s ease-out both scale-up;
}

```

Directional Transitions

Forward/Back Navigation

```
import { useRouter } from '@philjs/router';
import { signal, effect } from '@philjs/core';

const navigationDirection = signal<'forward' | 'back'>('forward');

export default function App() {
  const router = useRouter();

  effect(() => {
    const handleNavigation = (event: any) => {
      // Detect direction based on history state
      const direction = event.direction || 'forward';
      navigationDirection.set(direction);

      document.documentElement.setAttribute('data-direction', direction);
    };
  });

  router.events.on('routeChangeStart', handleNavigation);

  return () => {
    router.events.off('routeChangeStart', handleNavigation);
  };
}

return <Router viewTransitions={true} />;
}
```

```
/* Forward navigation - slide left */
[data-direction='forward'] ::view-transition-old(root) {
  animation: 0.3s ease-out both slide-to-left;
}

[data-direction='forward'] ::view-transition-new(root) {
  animation: 0.3s ease-out both slide-from-right;
}

/* Back navigation - slide right */
[data-direction='back'] ::view-transition-old(root) {
  animation: 0.3s ease-out both slide-to-right;
}

[data-direction='back'] ::view-transition-new(root) {
  animation: 0.3s ease-out both slide-from-left;
}

@keyframes slide-to-left {
  to { transform: translateX(-100%); }
}

@keyframes slide-from-right {
  from { transform: translateX(100%); }
}

@keyframes slide-to-right {
  to { transform: translateX(100%); }
}

@keyframes slide-from-left {
  from { transform: translateX(-100%); }
}
```

Shared Element Transitions

Basic Shared Element

```
// src/pages/products/index.tsx
export default function ProductList() {
  return (
    <div className="product-grid">
      {products.map(product => (
        <Link
          href={`/products/${product.id}`}
          key={product.id}
        >
          <img
            src={product.image}
            alt={product.name}
            style={{ viewTransitionName: `product-${product.id}` }}
          />
          <h3>{product.name}</h3>
        </Link>
      )));
    </div>
  );
}
```

```
// src/pages/products/[id].tsx
import { useParams } from '@philjs/router';

export default function ProductDetail() {
  const params = useParams<{ id: string }>();
  const product = getProduct(params.id);

  return (
    <div className="product-detail">
      <img
        src={product.image}
        alt={product.name}
        style={{ viewTransitionName: `product-${product.id}` }}
      />
      <h1>{product.name}</h1>
      <p>{product.description}</p>
    </div>
  );
}
```

The image smoothly morphs from thumbnail to full size!

Hero Image Transition

```
// List page
function PostCard({ post }: { post: Post }) {
  return (
    <article>
      <img
        src={post.coverImage}
        style={{ viewTransitionName: `post-hero-${post.id}` }}
        alt={post.title}
      />
      <h2>{post.title}</h2>
    </article>
  );
}

// Detail page
function PostDetail({ post }: { post: Post }) {
  return (
    <article>
      <img
        src={post.coverImage}
        style={{ viewTransitionName: `post-hero-${post.id}` }}
        alt={post.title}
        className="hero-image"
      />
      <h1>{post.title}</h1>
      <div>{post.content}</div>
    </article>
  );
}
```

Multiple Shared Elements

```

function ProductCard({ product }: { product: Product }) {
  return (
    <div className="card">
      <img
        src={product.image}
        style={{ viewTransitionName: `product-image-${product.id}` }}
      />
      <h3 style={{ viewTransitionName: `product-title-${product.id}` }}>
        {product.name}
      </h3>
      <p style={{ viewTransitionName: `product-price-${product.id}` }}>
        ${product.price}
      </p>
    </div>
  );
}

```

Transition Types

Page Type Transitions

```

// Add data attribute for page type
export default function BlogPost() {
  effect(() => {
    document.documentElement.setAttribute('data-page-type', 'blog-post');
  });

  return <article>...</article>;
}

/* Different animation for blog posts */
[data-page-type='blog-post'] ::view-transition-new(root) {
  animation: 0.5s ease-out both fade-in;
}

/* Different animation for products */
[data-page-type='product'] ::view-transition-new(root) {
  animation: 0.3s ease-out both slide-from-right;
}

```

Route-Based Transitions

```

import { usePathname } from '@philjs/router';
import { effect } from '@philjs/core';

export default function Layout({ children }: { children: any }) {
  const pathname = usePathname();

  effect(() => {
    // Set transition type based on route
    if (pathname.startsWith('/blog')) {
      document.documentElement.setAttribute('data-transition', 'fade');
    } else if (pathname.startsWith('/products')) {
      document.documentElement.setAttribute('data-transition', 'slide');
    } else {
      document.documentElement.setAttribute('data-transition', 'default');
    }
  });

  return <div>{children}</div>;
}

```

Performance Optimization

Skip Transition for Certain Routes

```

import { useRouter } from '@philjs/router';

function Navigation() {
  const router = useRouter();

  const navigate = (href: string, useTransition = true) => {
    if (useTransition && document.startViewTransition) {
      document.startViewTransition(() => {
        router.push(href);
      });
    } else {
      router.push(href);
    }
  };

  return (
    <nav>
      <button onClick={() => navigate('/dashboard', true)}>
        Dashboard
      </button>
      {/* Skip transition for quick actions */}
      <button onClick={() => navigate('/settings', false)}>
        Settings
      </button>
    </nav>
  );
}

```

Reduce Motion

```

/* Respect user preference */
@media (prefers-reduced-motion: reduce) {
  ::view-transition-group(*),
  ::view-transition-old(*),
  ::view-transition-new(*) {
    animation: none !important;
  }
}

```

Advanced Patterns

Conditional Shared Elements

```

import { useRouter } from '@philjs/router';
import { signal } from '@philjs/core';

const enableSharedElement = signal(true);

function ProductCard({ product }: { product: Product }) {
  const transitionName = enableSharedElement()
    ? `product-${product.id}`
    : undefined;

  return (
    <div>
      <img
        src={product.image}
        style={{ viewTransitionName: transitionName }}
      />
    </div>
  );
}

```

Cleanup Transition Names

```

import { effect } from '@philjs/core';

function ProductDetail({ product }: { product: Product }) {
  effect(() => {
    // Set transition name
    const img = document.querySelector('.product-image') as HTMLElement;
    if (img) {
      img.style.viewTransitionName = `product-${product.id}`;
    }
  });

  // Cleanup
  return () => {
    if (img) {
      img.style.viewTransitionName = '';
    }
  };
}

return (
  <img src={product.image} className="product-image" />
);
}

```

Nested Transitions

```

export default function Layout({ children }: { children: any }) {
  return (
    <div>
      <header style={{ viewTransitionName: 'header' }}>
        <Logo />
        <Navigation />
      </header>

      <main style={{ viewTransitionName: 'main-content' }}>
        {children}
      </main>

      <footer style={{ viewTransitionName: 'footer' }}>
        <FooterContent />
      </footer>
    </div>
  );
}

```

Fallback for Unsupported Browsers

Progressive Enhancement

```

import { useRouter } from '@philjs/router';

function useViewTransition() {
  const router = useRouter();

  const navigate = async (href: string) => {
    // Use View Transitions if supported
    if (document.startViewTransition) {
      await document.startViewTransition(() => {
        router.push(href);
      });
    } else {
      // Fallback: CSS transitions
      document.body.classList.add('page-transitioning');

      await new Promise(resolve => setTimeout(resolve, 300));

      router.push(href);

      setTimeout(() => {
        document.body.classList.remove('page-transitioning');
      }, 300);
    }
  };

  return { navigate };
}

```

```

/* Fallback CSS transition */
body.page-transitioning {
  animation: fade-out 0.3s ease-out;
}

@keyframes fade-out {
  to {
    opacity: 0;
  }
}

```

Feature Detection

```

const supportsViewTransitions = () => {
  return 'startViewTransition' in document;
};

export default function App() {
  return (
    <Router
      viewTransitions={supportsViewTransitions()}
    />
  );
}

```

Complete Examples

E-commerce Product Transition

```

// Product list
export default function ProductList() {
  const products = getProducts();

  return (
    <div className="product-grid">
      {products.map(product => (
        <Link
          href={`/products/${product.id}`}
          key={product.id}
          className="product-card"
        >
          <img
            src={product.image}
            style={{ viewTransitionName: `product-img-${product.id}` }}
            alt={product.name}
          />
          <h3 style={{ viewTransitionName: `product-name-${product.id}` }}>
            {product.name}
          </h3>
          <p className="price">${product.price}</p>
        </Link>
      ))}
    </div>
  );
}

// Product detail
export default function ProductDetail() {
  const params = useParams<{ id: string }>();
  const product = getProduct(params.id);

  return (
    <div className="product-detail">
      <img
        src={product.image}
        style={{ viewTransitionName: `product-img-${product.id}` }}
        className="product-hero"
      />

      <div className="product-info">
        <h1 style={{ viewTransitionName: `product-name-${product.id}` }}>
          {product.name}
        </h1>

        <p className="price">${product.price}</p>
        <p className="description">{product.description}</p>

        <button className="add-to-cart">Add to Cart</button>
      </div>
    </div>
  );
}

```

Blog Post Transition

```
// Blog List
function BlogList() {
  return (
    <div className="blog-posts">
      {posts.map(post => (
        <article key={post.id}>
          <Link href={`/blog/${post.slug}`}>
            <img
              src={post.coverImage}
              style={{ viewTransitionName: `post-cover-${post.id}` }}>
            />
            <h2 style={{ viewTransitionName: `post-title-${post.id}` }}>
              {post.title}
            </h2>
            <p>{post.excerpt}</p>
          </Link>
        </article>
      ))
    </div>
  );
}

// Blog post
function BlogPost() {
  const params = useParams<{ slug: string }>();
  const post = getPostBySlug(params.slug);

  return (
    <article className="blog-post">
      <img
        src={post.coverImage}
        style={{ viewTransitionName: `post-cover-${post.id}` }}
        className="cover-image"
      />

      <h1 style={{ viewTransitionName: `post-title-${post.id}` }}>
        {post.title}
      </h1>

      <div className="post-content">
        {post.content}
      </div>
    </article>
  );
}
```

Best Practices

Use Unique Transition Names

```
// ⚡ Unique per item
style={{ viewTransitionName: `product-${product.id}` }}

// ⚡ Same name for multiple elements
style={{ viewTransitionName: 'product-image' }}
```

Keep Animations Short

```
/* ⚡ Quick and snappy */
::view-transition-new(root) {
  animation-duration: 0.3s;
}

/* ⚡ Too slow */
::view-transition-new(root) {
  animation-duration: 1s;
}
```

Respect Motion Preferences

```
/* ⚡ Disable for users who prefer reduced motion */
@media (prefers-reduced-motion: reduce) {
  ::view-transition-group(*),
  ::view-transition-old(*),
  ::view-transition-new(*) {
    animation: none !important;
  }
}
```

Clean Up Transition Names

```
// Remove when component unmounts
effect(() => {
  element.style.viewTransitionName = 'my-element';

  return () => {
    element.style.viewTransitionName = '';
  };
});
```

Test Fallbacks

```
// Provide CSS fallback
if (!document.startViewTransition) {
  // Use CSS transitions instead
  document.body.classList.add('transitioning');
  setTimeout(() => {
    document.body.classList.remove('transitioning');
  }, 300);
}
```

Browser Support

View Transitions API is supported in: - Chrome 111+ - Edge 111+ - Safari 18+ (limited)

Fallback: Always provide CSS transitions for unsupported browsers.

Summary

You've learned:

- View Transitions API basics
- Route transition animations
- Custom transition styles
- Shared element transitions
- Directional animations
- Performance optimization
- Fallbacks for unsupported browsers
- Advanced patterns
- Best practices

View Transitions create polished, app-like experiences!

Next: [API Routes](#) → Build backend endpoints alongside your pages

API Routes

PhilJS pairs route loaders/actions with server handlers so you can colocate UI and API logic.

Server Handler Basics

Use `@philjs/ssr` to map route modules to server endpoints. Each route module exported via `createAppRouter` appears in the manifest. For API-only endpoints, create files that export a default handler.

```
// routes/api/subscribe.ts
export async function action({ request }: { request: Request }) {
  const formData = await request.formData();
  const email = formData.get('email');
  await subscribeUser(email);
  return new Response(null, { status: 204 });
}
```

In your server entry (e.g., Vite middleware or Cloudflare worker) reuse the manifest:

```
import { createFetchHandler } from '@philjs/ssr';
import { routes } from './routes';

const handleRequest = createFetchHandler({ routes });
export default { fetch: handleRequest };
```

Now POST requests to `/api/subscribe` call the route action without duplicating wiring.

UI Integration

On the client, use a form and let the action handle the submission. Afterwards, navigate programmatically or show a success message.

```

import { useRouter } from '@philjs/router';

export function SubscribeForm({ navigate }: RouteComponentProps) {
  const { route } = useRouter();

  async function handleSubmit(event: SubmitEvent) {
    event.preventDefault();
    const formData = new FormData(event.target as HTMLFormElement);
    await fetch('/api/subscribe?_action', {
      method: 'POST',
      body: formData,
    });
    await navigate('/thanks');
  }

  return (
    <form onSubmit={handleSubmit}>
      <input name="email" type="email" required />
      <button type="submit">Subscribe</button>
    </form>
  );
}

```

The `_action` query hint is optional but keeps semantics clear when you also have a loader.

Response Helpers

Return any Response from a loader/action. Common patterns:

```

// Redirect
throw new Response('', { status: 302, headers: { Location: '/login' } });

// JSON helper
return Response.json({ message: 'ok' });

// Stream
const stream = new ReadableStream({ start(controller) { /* ... */ } });
return new Response(stream, { headers: { 'content-type': 'text/event-stream' } });

```

Error Handling

Errors thrown from actions bubble to the router. Catch them in error boundaries or return structured responses:

```

action: async ({ formData }) => {
  const title = formData.get('title');
  if (!title) {
    return Response.json({ error: 'Title is required' }, { status: 400 });
  }
  return createPost(title);
}

```

On the client, inspect `response.ok` to display validation errors without reloading.

Deployment Targets

PhilJS ships adapters for Node, Edge, and workers. The same action/loader code runs in each environment—no need to maintain separate API routes.

Next, learn how to surface loader states in [Loading States](#) or defend routes with [Route Guards](#).

Data Fetching Overview

Learn how to fetch, cache, and manage data in PhilJS applications.

What You'll Learn

- Data fetching patterns
- Client vs server data loading
- Caching strategies
- Loading and error states
- Real-time updates
- Best practices

Data Fetching Patterns

PhilJS supports multiple data fetching approaches:

1. **Client-side fetching** - Fetch in components with effects
2. **Server functions** - Type-safe RPC calls
3. **Queries** - Declarative data fetching with caching
4. **Static generation** - Fetch at build time

5. Server-side rendering - Fetch on each request

Client-Side Fetching

Basic Fetch in Effect

```
import { signal, effect } from '@philjs/core';

function UserProfile({ userId }: { userId: string }) {
  const user = signal(null);
  const loading = signal(true);
  const error = signal<Error | null>(null);

  effect(() => {
    loading.set(true);
    error.set(null);

    fetch(`/api/users/${userId}`)
      .then(res => {
        if (!res.ok) throw new Error('Failed to fetch');
        return res.json();
      })
      .then(data => {
        user.set(data);
        loading.set(false);
      })
      .catch(err => {
        error.set(err);
        loading.set(false);
      });
  });

  if (loading()) return <Spinner />;
  if (error()) return <Error message={error()!.message} />;
  if (!user()) return null;

  return (
    <div>
      <h1>{user()!.name}</h1>
      <p>{user()!.email}</p>
    </div>
  );
}
```

Custom Fetch Hook

```
import { signal, effect } from '@philjs/core';

function useFetch<T>(url: string) {
  const data = signal<T | null>(null);
  const loading = signal(true);
  const error = signal<Error | null>(null);

  effect(() => {
    loading.set(true);
    error.set(null);

    fetch(url)
      .then(res => res.json())
      .then(json => {
        data.set(json);
        loading.set(false);
      })
      .catch(err => {
        error.set(err);
        loading.set(false);
      });
  });

  return { data, loading, error };
}

// Usage
function Component() {
  const { data, loading, error } = useFetch<User>('/api/user');

  if (loading()) return <Spinner />;
  if (error()) return <Error />;

  return <div>{data()!.name}</div>;
}
```

Server Functions

Type-safe server calls without API routes:

```
// src/server/users.ts
'use server';

export async function getUser(id: string) {
  const user = await db.users.findById(id);
  return user;
}

export async function updateUser(id: string, data: Partial<User>) {
  const user = await db.users.update(id, data);
  return user;
}

// src/pages/profile.tsx
import { getUser, updateUser } from '@/server/users';
import { signal, effect } from '@philjs/core';

export default function Profile({ userId }: { userId: string }) {
  const user = signal(null);

  effect(async () => {
    const data = await getUser(userId);
    user.set(data);
  });

  const handleUpdate = async () => {
    await updateUser(userId, { name: 'New Name' });
    const updated = await getUser(userId);
    user.set(updated);
  };

  return (
    <div>
      <h1>{user()?.name}</h1>
      <button onClick={handleUpdate}>Update</button>
    </div>
  );
}
```

Queries

Declarative data fetching with automatic caching:

```
import {createQuery} from '@philjs/core';

const userQuery = createQuery({
  key: (userId: string) => ['user', userId],
  fetcher: async (userId: string) => {
    const res = await fetch(`/api/users/${userId}`);
    return res.json();
});

function UserProfile({ userId }: { userId: string }) {
  const { data, loading, error, refetch } = userQuery(userId);

  if (loading()) return <Spinner />;
  if (error()) return <Error />

  return (
    <div>
      <h1>{data()?.name}</h1>
      <button onClick={refetch}>Refresh</button>
    </div>
  );
}
```

Mutations

Update data with optimistic updates:

```

import { createMutation, createQuery } from '@philjs/core';

const updateUserMutation = createMutation({
  mutationFn: async ({ id, data }: { id: string; data: Partial<User> }) => {
    const res = await fetch(` /api/users/${id}`, {
      method: 'PUT',
      body: JSON.stringify(data)
    });
    return res.json();
  },
  onSuccess: (data) => {
    // Invalidate and refetch
    userQuery.invalidate(data.id);
  }
});

function EditProfile({ user }: { user: User }) {
  const { mutate, loading } = updateUserMutation;

  const handleSave = () => {
    mutate({ id: user.id, data: { name: 'New Name' } });
  };

  return (
    <button onClick={handleSave} disabled={loading()}>
      Save
    </button>
  );
}

```

Static Generation

Fetch data at build time:

```

// src/pages/blog/[slug].tsx
interface Post {
  slug: string;
  title: string;
  content: string;
}

// Generate static paths
export async function generateStaticParams() {
  const posts = await fetch('https://api.example.com/posts').then(r => r.json());

  return posts.map((post: Post) => ({
    slug: post.slug
  }));
}

// Fetch data for each page
export async function getStaticProps({ params }: { params: { slug: string } }) {
  const post = await fetch(`https://api.example.com/posts/${params.slug}`)
    .then(r => r.json());

  return { props: { post } };
}

export default function BlogPost({ post }: { post: Post }) {
  return (
    <article>
      <h1>{post.title}</h1>
      <div>{post.content}</div>
    </article>
  );
}

```

Server-Side Rendering

Fetch on each request:

```
// src/pages/dashboard.tsx
export async function getServerSideProps() {
  const stats = await fetchDashboardStats();

  return {
    props: { stats }
  };
}

export default function Dashboard({ stats }: { stats: Stats }) {
  return (
    <div>
      <h1>Dashboard</h1>
      <StatsWidget stats={stats} />
    </div>
  );
}
```

Caching Strategies

Memory Cache

```
const cache = new Map<string, any>();

async function fetchWithCache<T>(key: string, fetcher: () => Promise<T>): Promise<T> {
  if (cache.has(key)) {
    return cache.get(key);
  }

  const data = await fetcher();
  cache.set(key, data);
  return data;
}

// Usage
const user = await fetchWithCache(
  `user:${userId}`,
  () => fetch(`api/users/${userId}`).then(r => r.json())
);
```

Time-Based Cache

```
interface CacheEntry<T> {
  data: T;
  timestamp: number;
}

class Cache {
  private cache = new Map<string, CacheEntry<any>>();

  set<T>(key: string, data: T) {
    this.cache.set(key, { data, timestamp: Date.now() });
  }

  get<T>(key: string, maxAge = 60000): T | null {
    const entry = this.cache.get(key);
    if (!entry) return null;

    if (Date.now() - entry.timestamp > maxAge) {
      this.cache.delete(key);
      return null;
    }

    return entry.data;
  }
}

const cache = new Cache();

async function fetchUser(userId: string) {
  // Try cache first (60 second TTL)
  const cached = cache.get(`user:${userId}`, 60000);
  if (cached) return cached;

  // Fetch and cache
  const user = await fetch(`api/users/${userId}`).then(r => r.json());
  cache.set(`user:${userId}`, user);

  return user;
}
```

Loading States

Skeleton Loading

```

function ProductList() {
  const { data, loading } = useProducts();

  if (loading()) {
    return (
      <div className="product-grid">
        {Array.from({ length: 12 }).map((_, i) => (
          <ProductSkeleton key={i} />
        )));
      </div>
    );
  }

  return (
    <div className="product-grid">
      {data()!.map(product => (
        <ProductCard key={product.id} product={product} />
      )));
    </div>
  );
}

```

Suspense

```

import { Suspense } from '@philjs/core';

export default function Page() {
  return (
    <Suspense fallback={<PageSkeleton />}>
      <DataComponent />
    </Suspense>
  );
}

```

Error Handling

Retry Logic

```

async function fetchWithRetry<T>(
  url: string,
  retries = 3
): Promise<T> {
  for (let i = 0; i < retries; i++) {
    try {
      const res = await fetch(url);
      if (!res.ok) throw new Error(`HTTP ${res.status}`);
      return await res.json();
    } catch (error) {
      if (i === retries - 1) throw error;
      await new Promise(resolve => setTimeout(resolve, 1000 * Math.pow(2, i)));
    }
  }
  throw new Error('Max retries exceeded');
}

```

Error Boundaries

```

import { ErrorBoundary } from '@philjs/core';

export default function Page() {
  return (
    <ErrorBoundary
      fallback={({error, reset}) => (
        <div>
          <h2>Failed to load data</h2>
          <p>{error.message}</p>
          <button onClick={reset}>Retry</button>
        </div>
      )}
    >
      <DataComponent />
    </ErrorBoundary>
  );
}

```

Real-Time Updates

WebSocket

```

import { signal, effect } from '@philjs/core';

function useWebSocket<T>(url: string) {
  const data = signal<T | null>(null);
  const connected = signal(false);

  effect(() => {
    const ws = new WebSocket(url);

    ws.onopen = () => connected.set(true);
    ws.onclose = () => connected.set(false);
    ws.onmessage = (event) => {
      data.set(JSON.parse(event.data));
    };
  });

  return () => ws.close();
};

return { data, connected };
}

function LiveDashboard() {
  const { data, connected } = useWebSocket<Stats>('wss://api.example.com/stats');

  return (
    <div>
      {!connected() && <div>Connecting...</div>}
      {data() && <StatsWidget stats={data()} />}
    </div>
  );
}

```

Polling

```

import { signal, effect } from '@philjs/core';

function usePolling<T>(url: string, interval = 5000) {
  const data = signal<T | null>(null);

  effect(() => {
    const fetchData = async () => {
      const res = await fetch(url);
      data.set(await res.json());
    };

    fetchData();
    const timer = setInterval(fetchData, interval);
  });

  return { data };
}

function LiveStats() {
  const { data } = usePolling<Stats>('/api/stats', 10000);

  return <StatsWidget stats={data()} />;
}

```

Best Practices

Centralize Data Fetching

```

// ⚡ Good - centralized API client
// src/lib/api.ts
export const api = {
  users: {
    get: (id: string) => fetch(`/api/users/${id}`).then(r => r.json()),
    list: () => fetch('/api/users').then(r => r.json()),
    create: (data: any) => fetch('/api/users', {
      method: 'POST',
      body: JSON.stringify(data)
    }).then(r => r.json())
  }
};

// Usage
const user = await api.users.get(userId);

// ⚡ Bad - scattered fetch calls
fetch('/api/users/' + userId);

```

Handle Loading and Errors

```
// ⚡ Always show Loading and error states
if (loading()) return <Spinner />;
if (error()) return <Error error={error()}!>/;

// ⚡ Missing states
return <div>{data()?.name}</div>;
```

Cache Wisely

```
// ⚡ Cache stable data
const countries = await fetchWithCache('countries', fetchCountries);

// ⚡ Don't cache user-specific or time-sensitive data
const currentUser = await fetchWithCache('user', getCurrentUser);
```

Type Your Data

```
// ⚡ Type-safe data fetching
interface User {
  id: string;
  name: string;
  email: string;
}

const user = await fetch(`/api/users/${id}`).then(r => r.json()) as Promise<User>;

// Or with Zod
const UserSchema = z.object({
  id: z.string(),
  name: z.string(),
  email: z.string().email()
});

const user = UserSchema.parse(await fetch('/api/users/1').then(r => r.json()));
```

Comparison Table

Pattern	When to Use	Pros	Cons
Client-side fetch	Dynamic, user-specific data	Interactive, personalized	Slower initial load
Server functions	Type-safe backend calls	No API routes needed	Server-only
Queries	Cacheable data	Auto caching, deduping	More setup
Static generation	Content that rarely changes	Fastest, SEO-friendly	Build-time only
SSR	Dynamic but SEO-critical	SEO + dynamic	Slower response

Summary

You've learned:

- Client-side fetching with effects
- Server functions for type-safe calls
- Queries for automatic caching
- Mutations for data updates
- Static generation and SSR
- Caching strategies
- Loading and error states
- Real-time updates
- Best practices

Choose the right pattern for your use case!

Next: [Server Functions](#) → Build type-safe backend functions

Queries

Declarative data fetching with automatic caching, deduplication, and revalidation.

What You'll Learn

- Creating queries
- Query caching
- Automatic refetching
- Dependent queries
- Parallel queries
- Query invalidation
- Best practices

What are Queries?

Queries provide a declarative way to fetch and cache data:

Benefits: - Automatic caching - Request deduplication - Background refetching - Stale-while-revalidate - TypeScript inference - Loading and error states

Basic Queries

Creating a Query

```
import { createQuery } from '@philjs/core';

interface User {
  id: string;
  name: string;
  email: string;
}

const userQuery = createQuery({
  key: (userId: string) => ['user', userId],
  fetcher: async (userId: string) => {
    const res = await fetch(` /api/users/${userId}`);
    if (!res.ok) throw new Error('Failed to fetch user');
    return res.json() as Promise<User>;
  }
});
```

Using a Query

```
import { signal, effect } from '@philjs/core';

function UserProfile({ userId }: { userId: string }) {
  const { data, loading, error, refetch } = userQuery(userId);

  if (loading()) return <Spinner />;
  if (error()) return <Error message={error().message} />

  return (
    <div>
      <h1>{data()!.name}</h1>
      <p>{data()!.email}</p>
      <button onClick={refetch}>Refresh</button>
    </div>
  );
}
```

Query Keys

Query keys identify and cache queries:

Simple Keys

```
const postsQuery = createQuery({
  key: () => ['posts'],
  fetcher: async () => {
    const res = await fetch('/api/posts');
    return res.json();
  }
});
```

Parameterized Keys

```
const postQuery = createQuery({
  key: (postId: string) => ['post', postId],
  fetcher: async (postId: string) => {
    const res = await fetch(` /api/posts/${postId}`);
    return res.json();
  }
});
```

Complex Keys

```

interface PostsFilter {
  category?: string;
  page?: number;
  limit?: number;
}

const postsQuery = createQuery({
  key: (filter: PostsFilter) => ['posts', filter],
  fetcher: async (filter: PostsFilter) => {
    const params = new URLSearchParams();
    if (filter.category) params.set('category', filter.category);
    if (filter.page) params.set('page', filter.page.toString());
    if (filter.limit) params.set('limit', filter.limit.toString());

    const res = await fetch(` /api/posts?${params}`);
    return res.json();
  }
});

// Usage
const { data } = postsQuery({ category: 'tech', page: 1, limit: 10 });

```

Query Options

Stale Time

```

const userQuery = createQuery({
  key: (userId: string) => ['user', userId],
  fetcher: async (userId: string) => {
    const res = await fetch(` /api/users/${userId}`);
    return res.json();
  },
  staleTime: 60000 // 1 minute - data is fresh for 60s
});

```

Cache Time

```

const postsQuery = createQuery({
  key: () => ['posts'],
  fetcher: fetchPosts,
  cacheTime: 300000 // 5 minutes - keep unused data for 5 minutes
});

```

Refetch Interval

```

const statsQuery = createQuery({
  key: () => ['stats'],
  fetcher: fetchStats,
  refetchInterval: 10000 // Refetch every 10 seconds
});

```

Refetch on Window Focus

```

const dataQuery = createQuery({
  key: () => ['data'],
  fetcher: fetchData,
  refetchOnWindowFocus: true // Refetch when user returns to tab
});

```

Retry

```

const dataQuery = createQuery({
  key: () => ['data'],
  fetcher: fetchData,
  retry: 3, // Retry 3 times on failure
  retryDelay: (attemptIndex) => Math.min(1000 * 2 ** attemptIndex, 30000)
});

```

Dependent Queries

Queries that depend on other queries:

```

const userQuery = createQuery({
  key: (userId: string) => ['user', userId],
  fetcher: async (userId: string) => {
    const res = await fetch(` /api/users/${userId}`);
    return res.json();
  }
});

const userPostsQuery = createQuery({
  key: (userId: string) => ['user-posts', userId],
  fetcher: async (userId: string) => {
    const res = await fetch(` /api/users/${userId}/posts`);
    return res.json();
  }
});

function UserProfile({ userId }: { userId: string }) {
  const { data: user, loading: userLoading } = userQuery(userId);
  const { data: posts, loading: postsLoading } = userPostsQuery(userId);

  if (userLoading() || postsLoading()) return <Spinner />

  return (
    <div>
      <h1>{user()?.name}</h1>
      <PostList posts={posts()} />
    </div>
  );
}

```

Conditional Queries

```

function UserPosts({ userId }: { userId: string | null }) {
  const enabled = signal(!userId);

  const { data, loading } = userPostsQuery(userId!, {
    enabled: enabled // Only fetch if enabled
  });

  if (!userId) return <div>No user selected</div>;
  if (loading()) return <Spinner />

  return <PostList posts={data()} />;
}

```

Parallel Queries

Fetch multiple queries simultaneously:

```

function Dashboard() {
  const { data: stats } = statsQuery();
  const { data: users } = usersQuery();
  const { data: posts } = postsQuery();

  // All queries run in parallel

  return (
    <div>
      <StatsWidget stats={stats()} />
      <UsersList users={users()} />
      <PostsList posts={posts()} />
    </div>
  );
}

```

Query Refetching

Manual Refetch

```

function UserProfile({ userId }: { userId: string }) {
  const { data, refetch } = userQuery(userId);

  return (
    <div>
      <h1>{data()?.name}</h1>
      <button onClick={refetch}>Refresh</button>
    </div>
  );
}

```

Refetch All

```

import { refetchAll } from '@philjs/core';

function RefreshButton() {
  return (
    <button onClick={() => refetchAll()}>
      Refresh All Data
    </button>
  );
}

```

Refetch by Key

```

import { refetchQueries } from '@philjs/core';

function refreshUserData(userId: string) {
  // Refetch all queries with matching key
  refetchQueries(['user', userId]);
}

```

Query Invalidation

Mark queries as stale and refetch:

```

import { invalidateQueries } from '@philjs/core';

async function updateUser(userId: string, data: any) {
  const res = await fetch(`/api/users/${userId}`, {
    method: 'PUT',
    body: JSON.stringify(data)
  });

  // Invalidate user query
  invalidateQueries(['user', userId]);

  return res.json();
}

```

Invalidate Multiple

```

async function deletePost(postId: string) {
  await fetch(`/api/posts/${postId}`, { method: 'DELETE' });

  // Invalidate related queries
  invalidateQueries(['post', postId]);
  invalidateQueries(['posts']); // All posts lists
}

```

Optimistic Updates

Update UI before server responds:

```

import { setQueryData, invalidateQueries } from '@philjs/core';

async function updateUserOptimistic(userId: string, data: Partial<User>) {
  // Save current data for rollback
  const previousUser = getQueryData(['user', userId]);

  // Optimistically update
  setQueryData(['user', userId], (old: User) => ({
    ...old,
    ...data
  }));

  try {
    // Make API call
    const updated = await fetch(`/api/users/${userId}`, {
      method: 'PUT',
      body: JSON.stringify(data)
    }).then(r => r.json());

    // Update with server response
    setQueryData(['user', userId], updated);
  } catch (error) {
    // Rollback on error
    setQueryData(['user', userId], previousUser);
    throw error;
  }
}

```

Pagination

Offset-Based Pagination

```

const postsQuery = createQuery({
  key: (page: number) => ['posts', page],
  fetcher: async (page: number) => {
    const res = await fetch(` /api/posts?page=${page}&limit=10`);
    return res.json();
  },
  keepPreviousData: true // Keep old data while fetching new page
});

function PostsList() {
  const page = signal(1);
  const { data, loading } = postsQuery(page());

  return (
    <div>
      {loading() && <Spinner />}

      {data() && (
        <div>
          {data()!.posts.map(post => (
            <PostCard key={post.id} post={post} />
          ))}
        </div>
        <button
          onClick={() => page.set(p => Math.max(1, p - 1))}
          disabled={page() === 1}
        >
          Previous
        </button>

        <span>Page {page()}</span>

        <button
          onClick={() => page.set(p => p + 1)}
          disabled={!data()!.hasMore}
        >
          Next
        </button>
      )}
    </div>
  );
}

```

Cursor-Based Pagination

```

const postsQuery = createQuery({
  key: (cursor: string | null) => ['posts', cursor],
  fetcher: async (cursor: string | null) => {
    const url = cursor
      ? `/api/posts?cursor=${cursor}`
      : '/api/posts';

    const res = await fetch(url);
    return res.json();
  }
});

function PostsList() {
  const cursor = signal<string | null>(null);
  const { data, loading } = postsQuery(cursor());

  const loadMore = () => {
    if (data()?.nextCursor) {
      cursor.set(data()!.nextCursor);
    }
  };

  return (
    <div>
      {data()?.posts.map(post => (
        <PostCard key={post.id} post={post} />
      ))}

      {data()?.nextCursor && (
        <button onClick={loadMore} disabled={loading()}>
          Load More
        </button>
      )}
    </div>
  );
}

```

Infinite Queries

Continuously append data:

```
import { createInfiniteQuery } from '@philjs/core';

const infinitePostsQuery = createInfiniteQuery({
  key: () => ['posts-infinite'],
  fetcher: async ({ pageParam = 0 }) => {
    const res = await fetch(` /api/posts?offset=${pageParam}&limit=10`);
    return res.json();
  },
  getNextPageParam: (lastPage, allPages) => {
    return lastPage.hasMore ? allPages.length * 10 : undefined;
  }
});

function InfinitePostsList() {
  const {
    data,
    loading,
    hasNextPage,
    fetchNextPage,
    isFetchingNextPage
  } = infinitePostsQuery();

  return (
    <div>
      {data()?.pages.flatMap(page =>
        page.posts.map(post => (
          <PostCard key={post.id} post={post} />
        )))
      }

      {hasNextPage() && (
        <button
          onClick={() => fetchNextPage()}
          disabled={isFetchingNextPage()}
        >
          {isFetchingNextPage() ? 'Loading...' : 'Load More'}
        </button>
      )}
    </div>
  );
}


```

Prefetching

Load data before it's needed:

```
import { prefetchQuery } from '@philjs/core';

function ProductCard({ product }: { product: Product }) {
  const handleMouseEnter = () => {
    // Prefetch product details on hover
    prefetchQuery(productQuery(product.id));
  };

  return (
    <Link
      href={`/products/${product.id}`}
      onMouseEnter={handleMouseEnter}
    >
      <img src={product.image} alt={product.name} />
      <h3>{product.name}</h3>
    </Link>
  );
}


```

Query Status

Track query state:

```
function DataComponent() {
  const query = dataQuery();

  if (query.isLoading()) return <Spinner />;
  if (query.isError()) return <Error error={query.error()} />;
  if (query.isSuccess()) return <Data data={query.data()} />

  return null;
}
```

Fine-Grained Status

```

const {
  data,
  loading,
  error,
  isloading,    // Initial Load
  isFetching,   // Any fetch (including background)
  isRefetching, // Manual refetch
  isStale,      // Data is stale
  isSuccess,
  isError,
} = dataQuery();

```

Best Practices

Use Query Keys Wisely

```

// ⚡ Good - hierarchical keys
['posts']           // ALL posts
['posts', { page: 1 }] // First page
['post', postId]     // Single post
['user', userId, 'posts'] // User's posts

// ⚡ Bad - inconsistent keys
['getAllPosts']
['post_' + postId]
[userId + '-posts']

```

Set Appropriate Stale Times

```

// ⚡ Match stale time to data volatility

// Rarely changes - long stale time
const countriesQuery = createQuery({
  key: () => ['countries'],
  fetcher: fetchCountries,
  staleTime: Infinity // Never stale
});

// Changes frequently - short stale time
const stockPriceQuery = createQuery({
  key: (symbol: string) => ['stock', symbol],
  fetcher: fetchStockPrice,
  staleTime: 5000 // 5 seconds
});

```

Handle Loading States

```

// ⚡ Show skeleton while loading
if (loading()) return <Skeleton />

// ⚡ Blank screen
if (loading()) return null;

```

Invalidate Related Queries

```

// ⚡ Invalidate all related data
async function createPost(data: any) {
  const post = await api.posts.create(data);

  // Invalidate lists
  invalidateQueries(['posts']);
  invalidateQueries(['user', post.authorId, 'posts']);

  return post;
}

// ⚡ Forget to invalidate
async function createPost(data: any) {
  return await api.posts.create(data);
}

```

Use TypeScript

```
// Type-safe queries
interface User {
  id: string;
  name: string;
}

const userQuery = createQuery<User, string>({
  key: (userId: string) => ['user', userId],
  fetcher: async (userId: string): Promise<User> => {
    const res = await fetch(`api/users/${userId}`);
    return res.json();
  }
});

// Full type inference
const { data } = userQuery('123');
const name = data()?.name; // TypeScript knows this is string | undefined
```

Summary

You've learned:

- Creating queries with `createQuery`
 - Query keys and caching
 - Query options (stale time, retry, etc.)
 - Dependent and parallel queries
 - Manual refetching
 - Query invalidation
 - Optimistic updates
 - Pagination patterns
 - Infinite queries
 - Prefetching
 - Best practices
- Queries simplify data fetching with smart caching!

Next: [Mutations](#) → Update data with optimistic updates

Mutations

Update server data with mutations, including optimistic updates and automatic cache invalidation.

What You'll Learn

- Creating mutations
- Optimistic updates
- Cache invalidation
- Error handling
- Mutation state
- Best practices

What are Mutations?

Mutations modify server data (create, update, delete):

Benefits: - Automatic loading/error states - Optimistic updates - Cache invalidation - Retry logic - TypeScript safety

Basic Mutations

Creating a Mutation

```
import { createMutation } from '@philjs/core';

interface UpdateUserData {
  id: string;
  name?: string;
  email?: string;
}

const updateUserMutation = createMutation({
  mutationFn: async ({ id, ...data }: UpdateUserData) => {
    const res = await fetch(`api/users/${id}`, {
      method: 'PUT',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(data)
    });

    if (!res.ok) throw new Error('Failed to update user');

    return res.json();
  }
});
```

Using a Mutation

```

function EditProfile({ user }: { user: User }) {
  const name = signal(user.name);
  const { mutate, loading, error } = updateUserMutation;

  const handleSave = () => {
    mutate({ id: user.id, name: name() });
  };

  return (
    <div>
      <input
        value={name()}
        onChange={(e) => name.set(e.target.value)}
      />

      <button onClick={handleSave} disabled={loading()}>
        {loading() ? 'Saving...' : 'Save'}
      </button>

      {error() && <div className="error">{error().message}</div>}
    </div>
  );
}

```

Mutation Callbacks

onSuccess

```

const createPostMutation = createMutation({
  mutationFn: async (data: CreatepostData) => {
    const res = await fetch('/api/posts', {
      method: 'POST',
      body: JSON.stringify(data)
    });
    return res.json();
  },
  onSuccess: (newPost) => {
    console.log('Post created:', newPost);

    // Redirect to new post
    router.push(`/posts/${newPost.id}`);
  }
});

```

onError

```

const deletePostMutation = createMutation({
  mutationFn: async (postId: string) => {
    const res = await fetch(`api/posts/${postId}`, {
      method: 'DELETE'
    });

    if (!res.ok) throw new Error('Failed to delete');
  },
  onError: (error) => {
    console.error('Delete failed:', error);
    alert('Failed to delete post. Please try again.');
  }
});

```

onSettled

```

const updatePostMutation = createMutation({
  mutationFn: updatePost,
  onSettled: (data, error) => {
    // Runs whether success or error
    console.log('Mutation completed');

    // Hide Loading indicator
    setLoading(false);
  }
});

```

Cache Invalidation

Automatically refresh queries after mutations:

```

import { invalidateQueries } from '@philjs/core';

const createPostMutation = createMutation({
  mutationFn: async (data: CreatePostData) => {
    const res = await fetch('/api/posts', {
      method: 'POST',
      body: JSON.stringify(data)
    });
    return res.json();
  },
  onSuccess: (newPost) => {
    // Invalidate and refetch posts list
    invalidateQueries(['posts']);

    // Invalidate user's posts
    invalidateQueries(['user', newPost.authorId, 'posts']);
  }
});

```

Invalidate Multiple Queries

```

const deleteUserMutation = createMutation({
  mutationFn: async (userId: string) => {
    await fetch('/api/users/${userId}', { method: 'DELETE' });
  },
  onSuccess: (_, userId) => {
    // Invalidate all related queries
    invalidateQueries(['user', userId]);
    invalidateQueries(['users']);
    invalidateQueries(['user', userId, 'posts']);
    invalidateQueries(['user', userId, 'comments']);
  }
});

```

Optimistic Updates

Update UI immediately, before server responds:

```

import { setQueryData, getQueryData, invalidateQueries } from '@philjs/core';

const likePostMutation = createMutation({
  mutationFn: async (postId: string) => {
    const res = await fetch('/api/posts/${postId}/like', {
      method: 'POST'
    });
    return res.json();
  },
  onMutate: async (postId) => {
    // Cancel outgoing refetches
    await cancelQueries(['post', postId]);

    // Snapshot current value
    const previousPost = getQueryData(['post', postId]);

    // Optimistically update
    setQueryData(['post', postId], (old: Post) => ({
      ...old,
      likes: old.likes + 1,
      isLiked: true
    }));
  },
  onError: (err, postId, context) => {
    // Rollback on error
    if (context?.previousPost) {
      setQueryData(['post', postId], context.previousPost);
    }
  },
  onSettled: (data, error, postId) => {
    // Refetch to sync with server
    invalidateQueries(['post', postId]);
  }
});

```

Optimistic List Update

```

const createTodoMutation = createMutation({
  mutationFn: async (text: string) => {
    const res = await fetch('/api/todos', {
      method: 'POST',
      body: JSON.stringify({ text })
    });
    return res.json();
  },
  onMutate: async (text) => {
    // Cancel refetches
    await cancelQueries(['todos']);

    // Snapshot
    const previousTodos = getQueryData(['todos']);

    // Optimistically add new todo
    const optimisticTodo = {
      id: `temp-${Date.now()}`,
      text,
      completed: false
    };

    setQueryData(['todos'], (old: Todo[]) => [...old, optimisticTodo]);

    return { previousTodos };
  },
  onError: (err, text, context) => {
    // Rollback
    if (context?.previousTodos) {
      setQueryData(['todos'], context.previousTodos);
    }
  },
  onSuccess: (newTodo) => {
    // Replace temp todo with real one
    setQueryData(['todos'], (old: Todo[]) =>
      old.map(todo =>
        todo.id.startsWith('temp-') ? newTodo : todo
      )
    );
  }
});

```

Mutation State

Track mutation progress:

```

function CreatePostForm() {
  const {
    mutate,
    mutateAsync,
    loading,
    error,
    data,
    reset
  } = createPostMutation;

  const handleSubmit = async () => {
    try {
      const post = await mutateAsync({ title: 'New Post', content: '...' });
      console.log('Created:', post);
    } catch (err) {
      console.error('Failed:', err);
    }
  };

  return (
    <div>
      <button onClick={handleSubmit} disabled={loading}>
        {loading() ? 'Creating...' : 'Create Post'}
      </button>

      {error() && (
        <div>
          Error: {error()!.message}
          <button onClick={reset}>Dismiss</button>
        </div>
      )}
    <div>
      {data() && <div>Created post: {data()!.title}</div>}
    </div>
  );
}

```

Sequential Mutations

Run mutations in sequence:

```

async function createAndPublishPost(data: CreatePostData) {
  const { mutateAsync: create } = createPostMutation;
  const { mutateAsync: publish } = publishPostMutation;

  // Create post
  const post = await create(data);

  // Then publish it
  await publish(post.id);

  return post;
}

```

Parallel Mutations

Run multiple mutations simultaneously:

```

async function bulkDeletePosts(postIds: string[]) {
  const { mutateAsync: deletePost } = deletePostMutation;

  await Promise.all(
    postIds.map(id => deletePost(id))
  );

  // Invalidate after all complete
  invalidateQueries(['posts']);
}

```

Form Mutations

Integrate with forms:

```

function CreatePostForm() {
  const title = signal('');
  const content = signal('');

  const { mutate, loading, error, reset } = createPostMutation;

  const handleSubmit = (e: Event) => {
    e.preventDefault();

    mutate(
      { title: title(), content: content() },
      {
        onSuccess: () => {
          // Clear form
          title.set('');
          content.set('');
          alert('Post created!');
        }
      }
    );
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        value={title()}
        onChange={(e) => title.set(e.target.value)}
        placeholder="Title"
        required
      />

      <textarea
        value={content()}
        onChange={(e) => content.set(e.target.value)}
        placeholder="Content"
        required
      />

      <button type="submit" disabled={loading()}>
        {loading() ? 'Creating...' : 'Create Post'}
      </button>
    {error() && (
      <div className="error">
        {error()!.message}
        <button onClick={reset}>x</button>
      </div>
    )}
  </form>
)
}

```

Retry Logic

Automatically retry failed mutations:

```
const updatePostMutation = createMutation({
  mutationFn: updatePost,
  retry: 3,
  retryDelay: (attemptIndex) => Math.min(1000 * 2 ** attemptIndex, 30000)
});
```

Mutation Context

Pass additional data through mutation lifecycle:

```
interface MutationContext {
  previousPosts?: Post[];
  timestamp: number;
}

const updatePostMutation = createMutation<
  Post, // Return type
  UpdatePostData, // Variables type
  MutationContext // Context type
>({
  mutationFn: async (data) => {
    const res = await fetch(`/api/posts/${data.id}`, {
      method: 'PUT',
      body: JSON.stringify(data)
    });
    return res.json();
  },
  onMutate: async (data) => {
    const previousPosts = getQueryData(['posts']);

    return {
      previousPosts,
      timestamp: Date.now()
    };
  },
  onError: (err, data, context) => {
    console.log(`Mutation started at:`, context?.timestamp);

    if (context?.previousPosts) {
      setQueryData(['posts'], context.previousPosts);
    }
  }
});
```

Global Mutation Defaults

Set defaults for all mutations:

```
import { setMutationDefaults } from '@philjs/core';

setMutationDefaults({
  retry: 1,
  onError: (error) => {
    console.error('Mutation error:', error);
    // Show global error toast
    showErrorToast(error.message);
  }
});
```

Mutation Middleware

Wrap mutations with common logic:

```
function withAuth<T, V>(
  mutation: Mutation<T, V>
): Mutation<T, V> {
  return createMutation({
    ...mutation,
    mutationFn: async (variables) => {
      const token = getAuthToken();

      if (!token) {
        throw new Error('Not authenticated');
      }

      return mutation.mutationFn(variables);
    }
  });
}

// Usage
const updateProfileMutation = withAuth(
  createMutation({
    mutationFn: async (data: UpdateProfileData) => {
      const res = await fetch('/api/profile', {
        method: 'PUT',
        body: JSON.stringify(data)
      });

      return res.json();
    }
  })
);
```

Complete Examples

[Todo App](#)

```

// Create todo
const createTodoMutation = createMutation({
  mutationFn: async (text: string) => {
    const res = await fetch('/api/todos', {
      method: 'POST',
      body: JSON.stringify({ text })
    });
    return res.json();
  },
  onSuccess: () => {
    invalidateQueries(['todos']);
  }
});

// Toggle todo
const toggleTodoMutation = createMutation({
  mutationFn: async (id: string) => {
    const res = await fetch(` /api/todos/${id}/toggle`, {
      method: 'PUT'
    });
    return res.json();
  },
  onMutate: async (id) => {
    const previousTodos = getQueryData(['todos']);

    setQueryData(['todos'], (old: Todo[]) =>
      old.map(todo =>
        todo.id === id
          ? { ...todo, completed: !todo.completed }
          : todo
      )
    );

    return { previousTodos };
  },
  onError: (err, id, context) => {
    if (context?.previousTodos) {
      setQueryData(['todos'], context.previousTodos);
    }
  }
});

// Delete todo
const deleteTodoMutation = createMutation({
  mutationFn: async (id: string) => {
    await fetch(` /api/todos/${id}`, { method: 'DELETE' });
  },
  onMutate: async (id) => {
    const previousTodos = getQueryData(['todos']);

    setQueryData(['todos'], (old: Todo[]) =>
      old.filter(todo => todo.id !== id)
    );

    return { previousTodos };
  },
  onError: (err, id, context) => {
    if (context?.previousTodos) {
      setQueryData(['todos'], context.previousTodos);
    }
  }
});

```

E-commerce Cart

```

const addToCartMutation = createMutation({
  mutationFn: async ({ productId, quantity }) {
    productId: string;
    quantity: number;
  } ) => {
  const res = await fetch('/api/cart', {
    method: 'POST',
    body: JSON.stringify({ productId, quantity })
  });
  return res.json();
},
onMutate: async ({ productId, quantity }) => {
  const previousCart = getQueryData(['cart']);

  // Optimistically update cart
  setQueryData(['cart'], (old: Cart) => ({
    ...old,
    items: [
      ...old.items,
      { productId, quantity, addedAt: Date.now() }
    ],
    total: old.total + (getProductPrice(productId) * quantity)
  }));

  return { previousCart };
},
onError: (err, variables, context) => {
  if (context?.previousCart) {
    setQueryData(['cart'], context.previousCart);
  }
  alert('Failed to add to cart');
},
onSuccess: () => {
  // Show success message
  showToast('Added to cart!');
}
});

```

Best Practices

Always Invalidate Queries

```

// ⚠ Invalidate related queries
const createPostMutation = createMutation({
  mutationFn: createPost,
  onSuccess: (post) => {
    invalidateQueries(['posts']);
    invalidateQueries(['user', post.authorId, 'posts']);
  }
});

// ⚠ Forget to invalidate
const createPostMutation = createMutation({
  mutationFn: createPost
});

```

Use Optimistic Updates Wisely

```

// ⚠ Use for simple, predictable updates
const likeMutation = createMutation({
  mutationFn: likePost,
  onMutate: optimisticallyIncrementLikes
});

// ⚠ Don't use for complex operations
const checkoutMutation = createMutation({
  mutationFn: checkout,
  // Don't optimistically update - too complex
});

```

Handle Errors

```

// ⚠ Show user-friendly error messages
const { mutate, error } = updateMutation;

{error() && (
  <div className="error">
    {getUserFriendlyMessage(error()!)}
  </div>
)}

// ⚠ Ignore errors
const { mutate } = updateMutation;

```

Provide Loading States

```
// ⚡ Disable button while mutating
<button disabled={loading()}>
  {loading() ? 'Saving...' : 'Save'}
</button>

// ⚡ No Loading indicator
<button onClick={mutate}>Save</button>
```

Summary

You've learned:

- Creating mutations with `createMutation`
- Mutation callbacks (`onSuccess`, `onError`, `onSettled`)
- Cache invalidation after mutations
- Optimistic updates for instant feedback
- Mutation state tracking
- Sequential and parallel mutations
- Form integration
- Retry logic
- Best practices

Mutations make data updates smooth and reliable!

Next: [Caching](#) → Advanced caching strategies and patterns

Optimistic Updates

Update the UI instantly before the server responds for a snappy user experience.

What You'll Learn

- What are optimistic updates
- Basic optimistic patterns
- Rollback on error
- Complex optimistic updates
- Race conditions
- Best practices

What are Optimistic Updates?

Optimistic updates immediately update the UI assuming the server request will succeed:

Flow: 1. User action (e.g., click "Like") 2. **Immediately** update UI (show liked state) 3. Send request to server 4. On success: keep UI as-is 5. On error: rollback UI

Benefits: - Instant feedback - Feels responsive - Better UX on slow connections

Basic Pattern

Simple Optimistic Update

```

import { signal } from '@philjs/core';

function LikeButton({ postId, initialLikes, initialIsLiked }: {
  postId: string;
  initialLikes: number;
  initialIsLiked: boolean;
}) {
  const likes = signal(initialLikes);
  const isLiked = signal(initialIsLiked);

  const toggleLike = async () => {
    // Save current state for rollback
    const previousLikes = likes();
    const previousIsLiked = isLiked();

    // Optimistically update UI
    isLiked.set(!previousIsLiked);
    likes.set(previousIsLiked ? previousLikes - 1 : previousLikes + 1);

    try {
      // Send to server
      await fetch(`api/posts/${postId}/like`, {
        method: 'POST',
        body: JSON.stringify({ liked: !previousIsLiked })
      });
    } catch (error) {
      // Rollback on error
      isLiked.set(previousIsLiked);
      likes.set(previousLikes);
      alert('Failed to update like');
    }
  };

  return (
    <button onClick={toggleLike} className={isLiked() ? 'liked' : ''}>
      {likes()}
    </button>
  );
}

```

With Mutations

Mutation with Optimistic Update

```

import { createMutation, setQueryData, getQueryData } from '@philjs/core';

const likePostMutation = createMutation({
  mutationFn: async (postId: string) => {
    const res = await fetch(` /api/posts/${postId}/like`, {
      method: 'POST'
    });
    return res.json();
  },
  onMutate: async (postId) => {
    // Cancel outgoing refetches
    await cancelQueries(['post', postId]);

    // Snapshot current value
    const previousPost = getQueryData(['post', postId]);

    // Optimistically update
    setQueryData(['post', postId], (old: Post) => ({
      ...old,
      likes: old.likes + 1,
      isLiked: true
    }));
  },
  // Return context with snapshot
  return { previousPost },
},
onError: (err, postId, context) => {
  // Rollback on error
  if (context?.previousPost) {
    setQueryData(['post', postId], context.previousPost);
  }
},
onSettled: (data, error, postId) => {
  // Refetch to sync with server
  invalidateQueries(['post', postId]);
}
});

// Usage
function Post({ post }: { post: Post }) {
  const { mutate } = likePostMutation;

  return (
    <div>
      <h2>{post.title}</h2>
      <button onClick={() => mutate(post.id)}>
        ❤️ {post.likes}
      </button>
    </div>
  );
}

```

List Updates

[Add Item to List](#)

```

const createTodoMutation = createMutation({
  mutationFn: async (text: string) => {
    const res = await fetch('/api/todos', {
      method: 'POST',
      body: JSON.stringify({ text })
    });
    return res.json();
  },
  onMutate: async (text) => {
    // Cancel refetches
    await cancelQueries(['todos']);

    // Snapshot
    const previousTodos = getQueryData(['todos']);

    // Create optimistic todo
    const optimisticTodo = {
      id: `temp-${Date.now()}`,
      text,
      completed: false,
      createdAt: Date.now()
    };

    // Add to list
    setQueryData(['todos'], (old: Todo[]) => [optimisticTodo, ...old]);

    return { previousTodos };
  },
  onError: (err, text, context) => {
    // Rollback
    if (context?.previousTodos) {
      setQueryData(['todos'], context.previousTodos);
    }
  },
  onSuccess: (newTodo) => {
    // Replace temporary todo with real one
    setQueryData(['todos'], (old: Todo[]) =>
      old.map(todo =>
        todo.id.startsWith('temp-') ? newTodo : todo
      )
    );
  }
});

```

Update Item in List

```

const updateTodoMutation = createMutation({
  mutationFn: async ({ id, text }: { id: string; text: string }) => {
    const res = await fetch(` /api/todos/${id}`, {
      method: 'PUT',
      body: JSON.stringify({ text })
    });
    return res.json();
  },
  onMutate: async ({ id, text }) => {
    await cancelQueries(['todos']);

    const previousTodos = getQueryData(['todos']);

    // Optimistically update
    setQueryData(['todos'], (old: Todo[]) =>
      old.map(todo =>
        todo.id === id ? { ...todo, text } : todo
      )
    );

    return { previousTodos };
  },
  onError: (err, variables, context) => {
    if (context?.previousTodos) {
      setQueryData(['todos'], context.previousTodos);
    }
  }
});

```

Remove Item from List

```

const deleteTodoMutation = createMutation({
  mutationFn: async (id: string) => {
    await fetch('/api/todos/${id}', { method: 'DELETE' });
  },
  onMutate: async (id) => {
    await cancelQueries(['todos']);
  },
  const previousTodos = getQueryData(['todos']);

  // Optimistically remove
  setQueryData(['todos'], (old: Todo[]) =>
    old.filter(todo => todo.id !== id)
  );

  return { previousTodos };
},
onError: (err, id, context) => {
  // Restore on error
  if (context?.previousTodos) {
    setQueryData(['todos'], context.previousTodos);
  }
}
));

```

Complex Updates

Multi-Query Updates

```

const followUserMutation = createMutation({
  mutationFn: async (userId: string) => {
    const res = await fetch('/api/users/${userId}/follow', {
      method: 'POST'
    });
    return res.json();
  },
  onMutate: async (userId) => {
    // Update multiple related queries
    await cancelQueries(['user', userId]);
    await cancelQueries(['following']);

    const previousUser = getQueryData(['user', userId]);
    const previousFollowing = getQueryData(['following']);

    // Update user
    setQueryData(['user', userId], (old: User) => ({
      ...old,
      isFollowing: true,
      followers: old.followers + 1
    }));

    // Update following list
    setQueryData(['following'], (old: User[]) => [
      ...old,
      { id: userId, /* ... */ }
    ]);

    return { previousUser, previousFollowing };
  },
  onError: (err, userId, context) => {
    // Rollback all updates
    if (context?.previousUser) {
      setQueryData(['user', userId], context.previousUser);
    }
    if (context?.previousFollowing) {
      setQueryData(['following'], context.previousFollowing);
    }
  }
});

```

Nested Data Updates

```

const updateCommentMutation = createMutation({
  mutationFn: async ({ postId, commentId, text }: {
    postId: string;
    commentId: string;
    text: string;
  }) => {
    const res = await fetch(` /api/posts/${postId}/comments/${commentId}` , {
      method: 'PUT',
      body: JSON.stringify({ text })
    });
    return res.json();
  },
  onMutate: async ({ postId, commentId, text }) => {
    await cancelQueries(['post', postId]);
    const previousPost = getQueryData(['post', postId]);

    // Update nested comment
    setQueryData(['post', postId], (old: Post) => ({
      ...old,
      comments: old.comments.map(comment =>
        comment.id === commentId
          ? { ...comment, text, edited: true }
          : comment
      )
    }));
    return { previousPost };
  },
  onError: (err, variables, context) => {
    if (context?.previousPost) {
      setQueryData(['post', variables.postId], context.previousPost);
    }
  }
});

```

Batch Operations

Multiple Optimistic Updates

```

const bulkUpdateTodosMutation = createMutation({
  mutationFn: async (updates: Array<{ id: string; completed: boolean }>) => {
    const res = await fetch('/api/todos/bulk-update', {
      method: 'PUT',
      body: JSON.stringify({ updates })
    });
    return res.json();
  },
  onMutate: async (updates) => {
    await cancelQueries(['todos']);

    const previousTodos = getQueryData(['todos']);

    // Apply all updates optimistically
    setQueryData(['todos'], (old: Todo[]) => {
      old.map(todo => {
        const update = updates.find(u => u.id === todo.id);
        return update ? { ...todo, completed: update.completed } : todo;
      });
    });

    return { previousTodos };
  },
  onError: (err, updates, context) => {
    if (context?.previousTodos) {
      setQueryData(['todos'], context.previousTodos);
    }
  }
});

```

Race Conditions

Handle Concurrent Updates

```

const updateProfileMutation = createMutation({
  mutationFn: async (data: Partial<User>) => {
    const res = await fetch('/api/profile', {
      method: 'PUT',
      body: JSON.stringify(data)
    });
    return res.json();
  },
  onMutate: async (data) => {
    // Cancel ALL ongoing profile updates
    await cancelQueries(['profile']);
  }
});

const previousProfile = getQueryData(['profile']);

// Track this update
const updateId = Date.now();

setQueryData(['profile'], (old: User) => ({
  ...old,
  ...data,
  _updateId: updateId
}));

return { previousProfile, updateId };
},
onSuccess: (serverData, variables, context) => {
  // Only apply if this is still the latest update
  setQueryData(['profile'], (old: User) => {
    if (old._updateId === context.updateId) {
      return serverData;
    }
    return old; // Newer update exists, don't overwrite
  });
},
onError: (err, variables, context) => {
  if (context?.previousProfile) {
    setQueryData(['profile'], context.previousProfile);
  }
}
));

```

Partial Success

Some Operations Succeed, Some Fail

```

const bulkDeleteMutation = createMutation({
  mutationFn: async (ids: string[]) => {
    const results = await Promise.allSettled(
      ids.map(id => fetch(`/api/todos/${id}`, { method: 'DELETE' }))
    );

    const succeeded = ids.filter((_, i) => results[i].status === 'fulfilled');
    const failed = ids.filter((_, i) => results[i].status === 'rejected');

    return { succeeded, failed };
  },
  onMutate: async (ids) => {
    await cancelQueries(['todos']);
  }
});

const previousTodos = getQueryData(['todos']);

// Optimistically remove all
setQueryData(['todos'], (old: Todo[]) =>
  old.filter(todo => !ids.includes(todo.id))
);

return { previousTodos };
},
onSuccess: (result, ids, context) => {
  // Restore failed deletions
  if (result.failed.length > 0 && context?.previousTodos) {
    setQueryData(['todos'], (old: Todo[]) => {
      const failedTodos = context.previousTodos.filter(
        todo => result.failed.includes(todo.id)
      );
      return [...old, ...failedTodos];
    });
  }
},
onError: (err, ids, context) => {
  // Complete failure - restore all
  if (context?.previousTodos) {
    setQueryData(['todos'], context.previousTodos);
  }
}
));

```

Visual Feedback

Show Optimistic State

```
interface Todo {
  id: string;
  text: string;
  completed: boolean;
  _optimistic?: boolean; // Flag for optimistic state
}

const createTodoMutation = createMutation({
  mutationFn: createTodo,
  onMutate: async (text) => {
    const optimisticTodo = {
      id: `temp-${Date.now()}`,
      text,
      completed: false,
      _optimistic: true // Mark as optimistic
    };

    setQueryData(['todos'], (old: Todo[]) => [optimisticTodo, ...old]);
  },
  onSuccess: (newTodo) => {
    // Remove optimistic flag
    setQueryData(['todos'], (old: Todo[]) =>
      old.map(todo =>
        todo.id.startsWith('temp-') ? { ...newTodo, _optimistic: false } : todo
      )
    );
  }
});

// Render with visual indicator
function TodoItem({ todo }: { todo: Todo }) {
  return (
    <div className={todo._optimistic ? 'optimistic' : ''}>
      {todo.text}
      {todo._optimistic && <span className="saving">Saving...</span>}
    </div>
  );
}

.optimistic {
  opacity: 0.6;
}

.saving {
  font-size: 12px;
  color: #888;
  margin-left: 8px;
}
```

Error Handling

User-Friendly Rollback

```
const deleteMutation = createMutation({
  mutationFn: deleteItem,
  onMutate: async (id) => {
    const previous = getQueryData(['items']);

    setQueryData(['items'], (old: Item[]) =>
      old.filter(item => item.id !== id)
    );

    return { previous };
  },
  onError: (err, id, context) => {
    // Rollback
    if (context?.previous) {
      setQueryData(['items'], context.previous);
    }

    // Show user-friendly message
    toast.error('Failed to delete item. Please try again.');
  }
});
```

Best Practices

Use for Simple Operations

```
// ⚡ Good - simple, predictable
const likeMutation = createMutation({
  onMutate: () => incrementLikes()
});

// ⚡ Bad - complex, error-prone
const checkoutMutation = createMutation({
  onMutate: () => processComplexCheckout()
});
```

Always Save Previous State

```
// ⚡ Always snapshot for rollback
onMutate: async (id) => {
  const previous = getQueryData(['item', id]);
  // ... optimistic update
  return { previous };
}

// ⚡ No way to rollback
onMutate: async (id) => {
  // ... optimistic update
  // No snapshot!
}
```

Provide Visual Feedback

```
// ⚡ Show optimistic state
<div className={item._optimistic ? 'pending' : ''}>
  {item.text}
</div>

// ⚡ No indication
<div>{item.text}</div>
```

Cancel Ongoing Requests

```
// ⚡ Cancel to avoid race conditions
onMutate: async (id) => {
  await cancelQueries(['item', id]);
  // ... update
}

// ⚡ Race condition possible
onMutate: async (id) => {
  // ... update
}
```

Sync with Server

```
// ⚡ Refetch to ensure sync
onSettled: (data, error, id) => {
  invalidateQueries(['item', id]);
}

// ⚡ UI may diverge from server
```

Summary

You've learned:

- Basic optimistic update patterns □ Rollback on error □ List updates (add, update, remove) □ Complex multi-query updates □ Handling race conditions □ Partial success scenarios
- Visual feedback for optimistic state □ Error handling □ Best practices

Optimistic updates make apps feel instant and responsive!

Next: [Pagination](#) → Load large datasets efficiently

Pagination

Load large datasets efficiently with pagination, infinite scroll, and cursor-based navigation.

What You'll Learn

- Offset pagination
- Cursor-based pagination
- Infinite scroll
- Load more buttons
- Page numbers
- Best practices

Offset Pagination

Basic Offset Pagination

```
import { signal } from '@philjs/core';

function usePagination<T>(
  fetcher: (page: number, limit: number) => Promise<T[]>,
  limit = 10
) {
  const page = signal(1);
  const data = signal<T[]>([]);
  const loading = signal(false);
  const hasMore = signal(true);

  const fetchPage = async (pageNum: number) => {
    loading.set(true);

    try {
      const results = await fetcher(pageNum, limit);

      data.set(results);
      hasMore.set(results.length === limit);
    } finally {
      loading.set(false);
    }
  };

  effect(() => {
    fetchPage(page());
  });

  const nextPage = () => page.set(p => p + 1);
  const prevPage = () => page.set(p => Math.max(1, p - 1));
  const goToPage = (p: number) => page.set(p);

  return {
    data,
    loading,
    page,
    hasMore,
    nextPage,
    prevPage,
    goToPage
  };
}
```

Using Offset Pagination

```
function ProductList() {
  const {
    data,
    loading,
    page,
    hasMore,
    nextPage,
    prevPage
  } = usePagination(
    async (page, limit) => {
      const offset = (page - 1) * limit;
      const res = await fetch(` /api/products?offset=${offset}&limit=${limit}`);
      return res.json();
    },
    20
  );

  return (
    <div>
      {loading() ? (
        <Spinner />
      ) : (
        <div className="product-grid">
          {data().map(product => (
            <ProductCard key={product.id} product={product} />
          )));
        </div>
      )}
    </div>
    <div className="pagination">
      <button
        onClick={prevPage}
        disabled={page() === 1 || loading()}
      >
        Previous
      </button>

      <span>Page {page()}</span>

      <button
        onClick={nextPage}
        disabled={!hasMore() || loading()}
      >
        Next
      </button>
    </div>
  );
}
```

With Query

```

import { createQuery } from '@philjs/core';

const productsQuery = createQuery({
  key: (page: number) => ['products', page],
  fetcher: async (page: number) => {
    const offset = (page - 1) * 20;
    const res = await fetch(` /api/products?offset=${offset}&limit=20`);
    return res.json();
  },
  keepPreviousData: true // Show old data while loading new page
});

function ProductList() {
  const page = signal(1);
  const { data, loading } = productsQuery(page());

  return (
    <div>
      {data() && (
        <>
          <div className="product-grid">
            {data()!.items.map(product => (
              <ProductCard key={product.id} product={product} />
            ))}
          </div>

          <Pagination
            current={page()}
            total={data()!.totalPages}
            onChange={(p) => page.set(p)}
          />
        )
      )}
    </div>
  );
}

```

Cursor-Based Pagination

Basic Cursor Pagination

```

function useCursorPagination<T>(
  fetcher: (cursor?: string) => Promise<{
    items: T[];
    nextCursor?: string;
  }>
) {
  const items = signal<T[]>([]);
  const cursor = signal<string | undefined>(undefined);
  const loading = signal(false);
  const hasMore = signal(true);

  const loadMore = async () => {
    if (loading() || !hasMore()) return;

    loading.set(true);

    try {
      const result = await fetcher(cursor());

      items.set([...items(), ...result.items]);
      cursor.set(result.nextCursor);
      hasMore.set(!result.nextCursor);
    } finally {
      loading.set(false);
    }
  };

  // Load initial page
  effect(() => {
    if (items().length === 0) {
      loadMore();
    }
  });
}

return { items, loading, hasMore, loadMore };
}

```

Using Cursor Pagination

```
interface Post {
  id: string;
  title: string;
  createdAt: number;
}

function PostsList() {
  const { items, loading, hasMore, loadMore } = useCursorPagination<Post>(
    async (cursor) => {
      const url = cursor
        ? `/api/posts?cursor=${cursor}`
        : '/api/posts';

      const res = await fetch(url);
      return res.json();
    }
  );

  return (
    <div>
      {items().map(post => (
        <PostCard key={post.id} post={post} />
      ))}
      {hasMore() && (
        <button onClick={loadMore} disabled={loading()}>
          {loading() ? 'Loading...' : 'Load More'}
        </button>
      )}
    </div>
  );
}
```

Infinite Scroll

Basic Infinite Scroll

```

import { signal, effect } from '@philjs/core';

function useInfiniteScroll<T>(
  fetcher: (offset: number) => Promise<T[]>,
  limit = 20
) {
  const items = signal<T[]>([]);
  const offset = signal(0);
  const loading = signal(false);
  const hasMore = signal(true);

  const loadMore = async () => {
    if (loading() || !hasMore()) return;

    loading.set(true);

    try {
      const newItems = await fetcher(offset());

      items.set([...items(), ...newItems]);
      offset.set(o => o + newItems.length);
      hasMore.set(newItems.length === limit);
    } finally {
      loading.set(false);
    }
  };

  // Scroll Listener
  effect(() => {
    const handleScroll = () => {
      const scrollTop = window.scrollTop;
      const scrollHeight = document.documentElement.scrollHeight;
      const clientHeight = window.innerHeight;

      // Load more when near bottom (500px threshold)
      if (scrollTop + clientHeight >= scrollHeight - 500) {
        loadMore();
      }
    };
  });

  window.addEventListener('scroll', handleScroll);

  return () => window.removeEventListener('scroll', handleScroll);
});

// Load initial data
effect(() => {
  if (items().length === 0) {
    loadMore();
  }
});

return { items, loading, hasMore };
}

```

Using Infinite Scroll

```

function FeedPage() {
  const { items, loading, hasMore } = useInfiniteScroll<Post>(
    async (offset) => {
      const res = await fetch(`/api/feed?offset=${offset}&limit=20`);
      return res.json();
    },
    20
  );

  return (
    <div className="feed">
      {items().map((post, index) => (
        <PostCard key={`${post.id}-${index}`} post={post} />
      ))}
      {loading() && (
        <div className="loading">
          <Spinner />
          <p>Loading more posts...</p>
        </div>
      )}
      {!hasMore() && (
        <div className="end-message">
          You've reached the end!
        </div>
      )}
    </div>
  );
}

```

Intersection Observer

More efficient than scroll listener:

```
function useInfiniteScroll<T>(fetcher: (offset: number) => Promise<T[]>) {
  const items = signal<T[]>([]);
  const loading = signal(false);
  const hasMore = signal(true);
  const observerTarget = signal<HTMLElement | null>(null);

  const loadMore = async () => {
    // ... same as before
  };

  effect(() => {
    const target = observerTarget();
    if (!target) return;

    const observer = new IntersectionObserver(
      (entries) => {
        if (entries[0].isIntersecting && hasMore() && !loading()) {
          loadMore();
        }
      },
      { threshold: 0.1 }
    );

    observer.observe(target);

    return () => observer.disconnect();
  });
}

return { items, loading, hasMore, observerTarget };
}

// Usage
function Feed() {
  const { items, loading, observerTarget } = useInfiniteScroll(fetchItems);

  return (
    <div>
      {items().map(item => (
        <Item key={item.id} item={item} />
      ))}
      {/* Observer target */}
      <div ref={observerTarget.set} style={{ height: '20px' }} />
      {loading() && <Spinner />}
    </div>
  );
}
```

Page Numbers

Numbered Pagination

```

function Pagination({
  current,
  total,
  onChange
}: {
  current: number;
  total: number;
  onChange: (page: number) => void;
}) {
  const pages = Array.from({ length: total }, (_, i) => i + 1);

  // Show max 7 page numbers
  const getVisiblePages = () => {
    if (total <= 7) return pages;

    if (current <= 4) {
      return [...pages.slice(0, 5), '...', total];
    }

    if (current >= total - 3) {
      return [1, '...', ...pages.slice(total - 5)];
    }

    return [
      1,
      '...',
      current - 1,
      current,
      current + 1,
      '...',
      total
    ];
  };

  return (
    <div className="pagination">
      <button
        onClick={() => onChange(current - 1)}
        disabled={current === 1}
      >
        Previous
      </button>

      {getVisiblePages().map((page, index) =>
        page === '...' ? (
          <span key={`ellipsis-${index}`} className="ellipsis">
            ...
          </span>
        ) : (
          <button
            key={page}
            onClick={() => onChange(page as number)}
            className={current === page ? 'active' : ''}
          >
            {page}
          </button>
        )
      )}
    <button
      onClick={() => onChange(current + 1)}
      disabled={current === total}
    >
      Next
    </button>
  </div>
);
}

```

Server-Side Pagination

API Response Format

```
// Good pagination response
interface PaginationResponse<T> {
  items: T[];
  pagination: {
    page: number;
    limit: number;
    total: number;
    totalPages: number;
    hasMore: boolean;
  };
}

// Or with cursor
interface CursorResponse<T> {
  items: T[];
  nextCursor?: string;
  prevCursor?: string;
}
```

API Implementation

```
// src/pages/api/posts.ts
export async function GET(request: Request) {
  const url = new URL(request.url);
  const page = parseInt(url.searchParams.get('page') || '1');
  const limit = parseInt(url.searchParams.get('limit') || '20');

  const offset = (page - 1) * limit;

  const [items, total] = await Promise.all([
    db.posts.findMany({ skip: offset, take: limit }),
    db.posts.count()
  ]);

  return new Response(JSON.stringify({
    items,
    pagination: {
      page,
      limit,
      total,
      totalPages: Math.ceil(total / limit),
      hasMore: offset + items.length < total
    }
  }), {
    headers: { 'Content-Type': 'application/json' }
  });
}
```

Search with Pagination

Paginated Search

```

function SearchResults() {
  const query = signal('');
  const page = signal(1);

  const searchQuery = createQuery({
    key: () => ['search', query(), page()],
    fetcher: async () => {
      if (!query()) return { items: [], total: 0 };

      const res = await fetch(
        `/api/search?q=${query()}&page=${page()}&limit=20`
      );
      return res.json();
    },
    enabled: () => query().length > 0
  });

  const { data, loading } = searchQuery();

  return (
    <div>
      <input
        value={query()}
        onInput={(e) => {
          query.set(e.target.value);
          page.set(1); // Reset to page 1 on new search
        }}
        placeholder="Search..." />

      {loading() && <Spinner />

      {data() && (
        <>
          <div className="results">
            {data()!.items.map(item => (
              <SearchResult key={item.id} item={item} />
            ))}
          </div>

          {data()!.total > 20 && (
            <Pagination
              current={page()}
              total={Math.ceil(data()!.total / 20)}
              onChange={(p) => page.set(p)}
            />
          )}
        </>
      )}
    </div>
  );
}

```

Virtual Scrolling

For extremely large lists:

```

function VirtualList<T>({
  items,
  height,
  itemHeight,
  renderItem
}: {
  items: T[];
  height: number;
  itemHeight: number;
  renderItem: (item: T) => any;
}) {
  const scrollTop = signal(0);

  const visibleCount = Math.ceil(height / itemHeight);
  const startIndex = Math.floor(scrollTop() / itemHeight);
  const endIndex = startIndex + visibleCount;

  const visibleItems = items.slice(startIndex, endIndex);
  const offsetY = startIndex * itemHeight;

  return (
    <div
      style={{ height: `${height}px`, overflow: 'auto' }}
      onScroll={(e) => scrollTop.set(e.target.scrollTop)}
    >
      <div style={{ height: `${items.length * itemHeight}px`, position: 'relative' }}>
        <div style={{ transform: `translateY(${offsetY}px)` }}>
          {visibleItems.map((item, index) => (
            <div key={startIndex + index} style={{ height: `${itemHeight}px` }}>
              {renderItem(item)}
            </div>
          ))}
        </div>
      </div>
    </div>
  );
}

```

Best Practices

Choose the Right Strategy

```

// ⚡ Offset pagination for pages with page numbers
const query = createQuery({
  key: (page) => ['items', page],
  fetcher: (page) => fetchPage(page)
});

// ⚡ Cursor for infinite feeds
const { items, loadMore } = useCursorPagination(fetchWithCursor);

// ⚡ Infinite scroll for social feeds
const { items } = useInfiniteScroll(fetchItems);

```

Keep Previous Data

```

// ⚡ Show old data while loading new page
const query = createQuery({
  key: (page) => ['items', page],
  fetcher: (page) => fetchPage(page),
  keepPreviousData: true
});

// ⚡ Blank screen while loading
const query = createQuery({
  key: (page) => ['items', page],
  fetcher: (page) => fetchPage(page)
});

```

Provide Loading States

```

// ⚡ Show loading indicator
{loading() && <Spinner />}
{!hasMore() && <div>End of list</div>}

// ⚡ No feedback
{items().map(item => <Item item={item} />)}

```

Handle Empty States

```
// ⚡ Show message when no results
{items().length === 0 && !loading() && (
  <div className="empty">No results found</div>
)}

// ⚡ Just empty space
```

Debounce Search

```
// ⚡ Debounce search input
const debouncedQuery = useDebouncedValue(query(), 300);

const searchQuery = createQuery({
  key: () => ['search', debouncedQuery],
  fetcher: () => search(debouncedQuery)
});

// ⚡ Search on every keystroke
```

Summary

You've learned:

- Offset pagination with page numbers
- Cursor-based pagination
- Infinite scroll patterns
- Load more buttons
- Intersection Observer for efficiency
- Page number UI
- Server-side pagination
- Search with pagination
- Virtual scrolling for large lists
- Best practices

Pagination makes large datasets manageable and performant!

Next: [Error Handling](#) → Handle data fetching errors gracefully

Prefetching

Prefetching loads data before it's needed, making navigation feel instant. PhilJS provides automatic and manual prefetching strategies.

Link Prefetching

Automatic Prefetch on Hover

```
import { Link } from '@philjs/router';

export default function Navigation() {
  return (
    <nav>
      <Link href="/products" prefetch="hover">
        Products
      </Link>
      <Link href="/about" prefetch="visible">
        About
      </Link>
    </nav>
  );
}
```

Prefetch Strategies

```
// Prefetch on hover (default)
<Link href="/page" prefetch="hover">Link</Link>

// Prefetch when visible
<Link href="/page" prefetch="visible">Link</Link>

// Prefetch on mount
<Link href="/page" prefetch="mount">Link</Link>

// Don't prefetch
<Link href="/page" prefetch={false}>Link</Link>
```

Manual Prefetching

Programmatic Prefetch

```

import { prefetch } from '@philjs/router';

export default function ProductCard({ product }) {
  return (
    <div
      onMouseEnter={() => {
        prefetch(`products/${product.id}`);
      }}
    >
      <h3>{product.name}</h3>
      <Link href={`/products/${product.id}`}>View</Link>
    </div>
  );
}

```

Prefetch with Data

```

import { prefetchQuery } from '@philjs/core';

const prefetchProduct = (id: number) => {
  prefetchQuery(`api/products/${id}`);
};

<button onMouseEnter={() => prefetchProduct(123)}>
  View Product
</button>

```

Query Prefetching

Prefetch in Component

```

import { useQueryClient } from '@philjs/core';

export default function ProductList() {
  const queryClient = useQueryClient();

  const prefetchProduct = (id: number) => {
    queryClient.prefetchQuery(
      ['product', id],
      () => fetch(`api/products/${id}`).then(r => r.json())
    );
  };

  return (
    <ul>
      {products.map(product => (
        <li
          key={product.id}
          onMouseEnter={() => prefetchProduct(product.id)}
        >
          <Link href={`/products/${product.id}`}>
            {product.name}
          </Link>
        </li>
      ))}
    </ul>
  );
}

```

Route Prefetching

Prefetch Related Routes

```

export const loader = createDataLoader(async ({ params }) => {
  const post = await db.posts.findById(params.id);

  // Prefetch related posts
  const relatedIds = post.related;
  relatedIds.forEach(id => {
    prefetch(`posts/${id}`);
  });

  return { post };
});

```

Best Practices

Do: Prefetch Likely Next Steps

```

// 🌐 Good - prefetch common flows
<Link href="/cart" prefetch="hover">
  View Cart (likely next step)
</Link>

```

□ Don't: Over-Prefetch

```
// ⚡ Bad - prefetches everything
{items.map(item => (
  <Link href={item.url} prefetch="mount">...</Link>
))}

// ⚡ Good - selective prefetch
{items.slice(0, 5).map(item => (
  <Link href={item.url} prefetch="hover">...</Link>
))}
```

Next Steps

- [Caching](#) - Cache prefetched data
- [Performance](#) - Optimize prefetching
- [Navigation](#) - Navigation patterns

□ **Tip:** Prefetch on hover for the best balance of performance and bandwidth usage.

□□ **Warning:** Don't prefetch authenticated routes for anonymous users—it wastes bandwidth.

□□ **Note:** Prefetched data is cached and reused when the user navigates to that route.

Caching Strategies

Master caching to build fast, efficient applications that minimize unnecessary network requests.

What You'll Learn

- Cache invalidation
- Stale-while-revalidate
- Cache persistence
- Custom cache implementations
- Memory management
- Best practices

Cache Basics

How Caching Works

PhilJS caches query results in memory by query key:

```
const userQuery = createQuery({
  key: (userId: string) => ['user', userId],
  fetcher: fetchUser
});

// First call - fetches from network
const { data: user1 } = userQuery('123');

// Second call with same ID - returns from cache
const { data: user2 } = userQuery('123');
```

Cache Lifecycle

1. Query executes → 2. Data fetched → 3. Cached
↓
4. Becomes stale
↓
5. Refetched (if active)
↓
6. Removed (if inactive)

Stale Time

Control when data is considered stale:

```
const postsQuery = createQuery({
  key: () => ['posts'],
  fetcher: fetchPosts,
  staleTime: 60000 // Fresh for 1 minute
});

// Within 60s - returns cache, no refetch
// After 60s - returns cache, refetches in background
```

Stale Time Patterns

```

// Never stale (static data)
const countriesQuery = createQuery({
  key: () => ['countries'],
  fetcher: fetchCountries,
  staleTime: Infinity
});

// Always stale (real-time data)
const stockQuery = createQuery({
  key: (symbol: string) => ['stock', symbol],
  fetcher: fetchStockPrice,
  staleTime: 0
});

// Smart stale time (5 minutes)
const userQuery = createQuery({
  key: (id: string) => ['user', id],
  fetcher: fetchUser,
  staleTime: 5 * 60 * 1000
});

```

Cache Time

Control how long unused data stays in cache:

```

const dataQuery = createQuery({
  key: () => ['data'],
  fetcher: fetchData,
  cacheTime: 5 * 60 * 1000 // Keep for 5 minutes after last use
});

// Component unmounts → cache stays for 5 minutes
// If no component uses it → removed after 5 minutes

```

Cache Time vs Stale Time

```

const query = createQuery({
  key: () => ['data'],
  fetcher: fetchData,
  staleTime: 60000, // Fresh for 1 minute
  cacheTime: 30000 // Keep in cache for 5 minutes
});

// t=0s: Fetch and cache (fresh)
// t=30s: Return from cache (still fresh)
// t=70s: Return from cache + refetch (stale)
// t=5min: Remove from cache (no active users)

```

Cache Invalidation

Manual Invalidation

```

import { invalidateQueries } from '@philjs/core';

// Invalidate specific query
invalidateQueries(['user', userId]);

// Invalidate all users
invalidateQueries(['users']);

// Invalidate by prefix
invalidateQueries(['user']); // Matches ['user', '123'], ['user', '456'], etc.

```

Automatic Invalidation

```

const updateUserMutation = createMutation({
  mutationFn: updateUser,
  onSuccess: (user) => {
    // Invalidate after successful update
    invalidateQueries(['user', user.id]);
    invalidateQueries(['users']);
  }
});

```

Conditional Invalidation

```

const deletePostMutation = createMutation({
  mutationFn: deletePost,
  onSuccess: (deletedPost) => {
    // Only invalidate if post was published
    if (deletedPost.published) {
      invalidateQueries(['posts', 'published']);
    }

    invalidateQueries(['user', deletedPost.authorId, 'posts']);
  }
});

```

Direct Cache Updates

Update cache without refetching:

Set Query Data

```

import { setQueryData } from '@philjs/core';

// Set entire cache
setQueryData(['user', userId], updatedUser);

// Update with function
setQueryData(['user', userId], (old: User) => ({
  ...old,
  name: newName
}));

```

Update List Cache

```

// Add item to list
setQueryData(['todos'], (old: Todo[]) => [
  ...old,
  newTodo
]);

// Update item in list
setQueryData(['todos'], (old: Todo[]) =>
  old.map(todo =>
    todo.id === updatedTodo.id ? updatedTodo : todo
  )
);

// Remove item from list
setQueryData(['todos'], (old: Todo[]) =>
  old.filter(todo => todo.id !== deletedId)
);

```

Optimistic Updates

```

const likeMutation = createMutation({
  mutationFn: likePost,
  onMutate: async (postId) => {
    // Save current data
    const previous = getQueryData(['post', postId]);

    // Optimistically update
    setQueryData(['post', postId], (old: Post) => ({
      ...old,
      likes: old.likes + 1,
      isLiked: true
    }));
  },
  return { previous },
  onError: (err, postId, context) => {
    // Rollback on error
    setQueryData(['post', postId], context.previous);
  },
  onSettled: (data, err, postId) => {
    // Refetch to sync
    invalidateQueries(['post', postId]);
  }
});

```

Cache Persistence

LocalStorage Persistence

```

import { persistQueryCache } from '@philjs/core';

// Persist cache to localStorage
persistQueryCache({
  storage: localStorage,
  key: 'philjs-cache',
  maxAge: 24 * 60 * 60 * 1000 // 24 hours
});

// Cache survives page reloads

```

Custom Storage

```

import { persistQueryCache } from '@philjs/core';

const customStorage = {
  getItem: (key: string) => {
    // Get from IndexedDB, SessionStorage, etc.
    return indexedDB.get(key);
  },
 .setItem: (key: string, value: string) => {
    indexedDB.set(key, value);
  },
 .removeItem: (key: string) => {
    indexedDB.remove(key);
  }
};

persistQueryCache({
  storage: customStorage,
  key: 'app-cache'
});

```

Selective Persistence

```

persistQueryCache({
  storage: localStorage,
  key: 'cache',
  // Only persist certain queries
  filter: (query) => {
    const [queryKey] = query.key;
    return ['users', 'posts', 'settings'].includes(queryKey);
  }
});

```

Background Refetching

Refetch on Window Focus

```

const query = createQuery({
  key: () => ['data'],
  fetcher: fetchData,
  refetchOnWindowFocus: true
});

// User returns to tab → refetches if stale

```

Refetch on Reconnect

```

const query = createQuery({
  key: () => ['data'],
  fetcher: fetchData,
  refetchOnReconnect: true
});

// Internet reconnects → refetches if stale

```

Polling

```

const liveStatsQuery = createQuery({
  key: () => ['stats'],
  fetcher: fetchStats,
  refetchInterval: 10000 // Refetch every 10 seconds
});

```

Conditional Polling

```
const statsQuery = createQuery({
  key: () => ['stats'],
  fetcher: fetchStats,
  refetchInterval: (data) => {
    // Stop polling if no data
    return data ? 10000 : false;
  }
});
```

Memory Management

Automatic Cleanup

```
const query = createQuery({
  key: () => ['data'],
  fetcher: fetchData,
  cacheTime: 5 * 60 * 1000 // Remove after 5 minutes of inactivity
});

// Component mounts + query active
// Component unmounts + 5 minute timer starts
// Timer expires + cache removed
```

Manual Cleanup

```
import { clearQueryCache, removeQueries } from '@philjs/core';

// Remove specific query
removeQueries(['user', userId]);

// Clear entire cache
clearQueryCache();

// Remove inactive queries
removeQueries({ type: 'inactive' });

// Remove stale queries
removeQueries({ type: 'stale' });
```

Cache Size Limits

```
import { setQueryCacheConfig } from '@philjs/core';

setQueryCacheConfig({
  maxQueries: 100, // Maximum number of queries
  maxAge: 24 * 60 * 60 * 1000, // Maximum age (24 hours)
  onExceedMax: 'remove-oldest' // What to do when max exceeded
});
```

Advanced Patterns

Hierarchical Invalidation

```
// Invalidate all user-related data
function invalidateUserData(userId: string) {
  invalidateQueries(['user', userId]);
  invalidateQueries(['user', userId, 'posts']);
  invalidateQueries(['user', userId, 'comments']);
  invalidateQueries(['user', userId, 'followers']);
  invalidateQueries(['user', userId, 'following']);
}

// Usage
deleteUserMutation({
  onSuccess: (user) => invalidateUserData(user.id)
});
```

Smart Cache Updates

```

const createPostMutation = createMutation({
  mutationFn: createPost,
  onSuccess: (newPost) => {
    // Add to posts List cache
    setQueryData(['posts'], (old: Post[] | undefined) => {
      if (!old) return [newPost];
      return [newPost, ...old];
    });

    // Update user's post count
    setQueryData(['user', newPost.authorId], (old: User) => ({
      ...old,
      postCount: old.postCount + 1
    }));
  }

  // Set single post cache
  setQueryData(['post', newPost.id], newPost);
}
));

```

Partial Updates

```

// Update only changed fields
setQueryData(['user', userId], (old: User) => ({
  ...old,
  ...partialUpdate
}));

// Update nested data
setQueryData(['settings'], (old: Settings) => ({
  ...old,
  notifications: {
    ...old.notifications,
    email: newEmailSetting
  }
}));

```

Batch Updates

```

import { batch } from '@philjs/core';

batch(() => {
  setQueryData(['user', '1'], user1);
  setQueryData(['user', '2'], user2);
  setQueryData(['user', '3'], user3);
});

// All updates applied together, single re-render

```

Cache Debugging

Inspect Cache

```

import { getQueryCache } from '@philjs/core';

// Get all queries in cache
const cache = getQueryCache();

console.log('Cache size:', cache.size);
console.log('Cached queries:', cache.keys());

// Get specific query
const userCache = getQueryData(['user', userId]);
console.log('User cache:', userCache);

```

Cache Events

```

import { onCacheChange } from '@philjs/core';

onCacheChange((event) => {
  console.log('Cache event:', event.type);
  console.log('Query key:', event.key);
  console.log('Data:', event.data);
});

// Events: 'added', 'updated', 'removed', 'invalidated'

```

Best Practices

Choose Appropriate Stale Times

```
// ⚡ Match stale time to data volatility
const staticQuery = createQuery({
  staleTime: Infinity // Never changes
});

const slowChangingQuery = createQuery({
  staleTime: 60 * 60 * 1000 // 1 hour
});

const fastChangingQuery = createQuery({
  staleTime: 10000 // 10 seconds
});

// ⚡ One size fits all
const query = createQuery({
  staleTime: 5000 // Same for all queries
});
```

Invalidate Proactively

```
// ⚡ Invalidate all affected queries
const updateMutation = createMutation({
  onSuccess: (data) => {
    invalidateQueries(['item', data.id]);
    invalidateQueries(['items']);
    invalidateQueries(['user', data.userId, 'items']);
  }
});

// ⚡ Forget related queries
const updateMutation = createMutation({
  onSuccess: (data) => {
    invalidateQueries(['item', data.id]);
    // Missing: items list, user items
  }
});
```

Use Optimistic Updates Carefully

```
// ⚡ Optimistic for simple updates
const likeMutation = createMutation({
  onMutate: async (id) => {
    setQueryData(['likes', id], old => old + 1);
  }
});

// ⚡ Optimistic for complex operations
const checkoutMutation = createMutation({
  onMutate: async (data) => {
    // Don't optimistically update complex state
  }
});
```

Persist Important Data

```
// ⚡ Persist user preferences, auth state
persistQueryCache({
  filter: (query) => {
    const [key] = query.key;
    return ['auth', 'settings', 'preferences'].includes(key);
  }
});

// ⚡ Persist everything (wastes storage)
persistQueryCache({
  filter: () => true
});
```

Clean Up Regularly

```
// ⚡ Set reasonable cache times
const query = createQuery({
  cacheTime: 5 * 60 * 1000 // 5 minutes
});

// Clear old data periodically
setInterval(() => {
  removeQueries({ type: 'inactive' });
}, 60 * 60 * 1000); // Every hour

// ⚡ Keep everything forever
const query = createQuery({
  cacheTime: Infinity
});
```

Summary

You've learned:

- Cache lifecycle and management □ Stale time vs cache time □ Manual and automatic invalidation □ Direct cache updates □ Optimistic updates □ Cache persistence (localStorage, custom) □ Background refetching □ Memory management and cleanup □ Advanced caching patterns □ Best practices
- Smart caching makes apps fast and responsive!

Next: [Real-Time Data](#) → WebSockets, SSE, and live updates

Loading States

Properly managing loading states creates a better user experience. PhilJS provides multiple patterns for handling loading, from simple spinners to sophisticated skeleton screens.

Basic Loading States

Component-Level Loading

```
import { signal, effect } from '@philjs/core';

export default function UserProfile({ userId }: { userId: number }) {
  const user = signal(null);
  const loading = signal(true);
  const error = signal(null);

  effect(() => {
    loading.set(true);
    fetch(`/api/users/${userId}`)
      .then(res => res.json())
      .then(data => {
        user.set(data);
        loading.set(false);
      })
      .catch(err => {
        error.set(err);
        loading.set(false);
      });
  });

  if (loading()) return <div>Loading...</div>;
  if (error()) return <div>Error: {error().message}</div>;

  return (
    <div>
      <h1>{user().name}</h1>
      <p>{user().email}</p>
    </div>
  );
}
```

Skeleton Screens

```
function UserSkeleton() {
  return (
    <div class="skeleton">
      <div class="skeleton-avatar" />
      <div class="skeleton-text" />
      <div class="skeleton-text short" />
    </div>
  );
}

export default function UserProfile() {
  const { data, loading } = useQuery('/api/user');

  if (loading) return <UserSkeleton />

  return (
    <div>
      <img src={data.avatar} />
      <h1>{data.name}</h1>
      <p>{data.bio}</p>
    </div>
  );
}
```

Suspense Boundaries

Basic Suspense

```

import { Suspense } from '@philjs/core';

export default function App() {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <AsyncComponent />
    </Suspense>
  );
}

```

Nested Suspense

```

export default function Dashboard() {
  return (
    <div>
      <Suspense fallback={<HeaderSkeleton />}>
        <Header />
      </Suspense>

      <Suspense fallback={<ContentSkeleton />}>
        <MainContent />
      </Suspense>

      <Suspense fallback={<SidebarSkeleton />}>
        <Sidebar />
      </Suspense>
    </div>
  );
}

```

Progressive Loading

Critical First, Then Rest

```

export const loader = createDataLoader(async () => {
  return {
    // Critical data - load immediately
    user: await fetchUser(),

    // Non-critical - stream later
    recommendations: fetchRecommendations(),
    activity: fetchActivity()
  };
});

export default function Dashboard({ data }) {
  return (
    <div>
      {/* Shows immediately */}
      <h1>Welcome {data.user.name}</h1>

      {/* Streams when ready */}
      <Suspense fallback={<RecommendationsSkeleton />}>
        <Await resolve={data.recommendations}>
          {(items) => <Recommendations items={items} />}
        </Await>
      </Suspense>
    </div>
  );
}

```

Global Loading Indicators

Top Loading Bar

```

import { useNavigation } from '@philjs/router';

export default function LoadingBar() {
  const navigation = useNavigation();

  return (
    <div class={`loading-bar ${navigation.state === 'loading' ? 'active' : ''}`}>
    </div>
  );
}

```

Best Practices

Do: Show Content Structure

```
// ☺ Good - skeleton matches content
<div class="card">
  <div class="skeleton-image" />
  <div class="skeleton-title" />
  <div class="skeleton-text" />
</div>
```

□ Do: Use Optimistic UI

```
const toggleLike = () => {
  // Update UI immediately
  liked.set(!liked());

  // Then sync with server
  fetch('/api/like', { method: 'POST' })
    .catch(() => liked.set(!liked())); // Revert on error
};
```

Next Steps

- [Error Handling](#) - Handle errors
- [Caching](#) - Cache strategies
- [Optimistic Updates](#) - Instant UI

□ **Tip:** Use skeleton screens instead of spinners for a perceived faster load time.

□□ **Warning:** Always handle both loading and error states for robust UX.

□□ **Note:** Suspense boundaries allow you to show content progressively as it loads.

Error Handling for Data Fetching

Handle data fetching errors gracefully with retry logic, fallbacks, and user-friendly messages.

What You'll Learn

- Error detection
- Retry strategies
- Error boundaries
- Fallback UI
- Error recovery
- Best practices

Basic Error Handling

Simple Try-Catch

```
import { signal, effect } from '@philjs/core';

function useData<T>(url: string) {
  const data = signal<T | null>(null);
  const error = signal<Error | null>(null);
  const loading = signal(true);

  effect(() => {
    const fetchData = async () => {
      try {
        loading.set(true);
        error.set(null);

        const res = await fetch(url);

        if (!res.ok) {
          throw new Error(`HTTP ${res.status}: ${res.statusText}`);
        }

        const json = await res.json();
        data.set(json);
      } catch (err) {
        error.set(err as Error);
      } finally {
        loading.set(false);
      }
    };
  });

  fetchData();
}

return { data, error, loading };
}
```

Using Error State

```
function UserProfile({ userId }: { userId: string }) {
  const { data, error, loading } = useData<User>(`api/users/${userId}`);

  if (loading()) {
    return <Spinner />;
  }

  if (error()) {
    return (
      <div className="error">
        <h2>Failed to load user</h2>
        <p>{error().message}</p>
        <button onClick={() => window.location.reload()}>
          Try Again
        </button>
      </div>
    );
  }

  if (!data()) {
    return <div>User not found</div>;
  }

  return (
    <div>
      <h1>{data().name}</h1>
      <p>{data().email}</p>
    </div>
  );
}
```

Retry Logic

Automatic Retry

```
async function fetchWithRetry<T>(
  url: string,
  maxRetries = 3,
  delay = 1000
): Promise<T> {
  for (let i = 0; i <= maxRetries; i++) {
    try {
      const res = await fetch(url);

      if (!res.ok) {
        throw new Error(`HTTP ${res.status}`);
      }

      return await res.json();
    } catch (error) {
      const isLastAttempt = i === maxRetries;

      if (isLastAttempt) {
        throw error;
      }

      // Exponential backoff
      const waitTime = delay * Math.pow(2, i);
      await new Promise(resolve => setTimeout(resolve, waitTime));
    }
  }

  throw new Error('Max retries exceeded');
}
```

Retry with State

```

function useDataWithRetry<T>(url: string, maxRetries = 3) {
  const data = signal<T | null>(null);
  const error = signal<Error | null>(null);
  const loading = signal(true);
  const retryCount = signal(0);

  const fetchData = async () => {
    try {
      loading.set(true);
      error.set(null);

      const result = await fetchWithRetry<T>(url, maxRetries);

      data.set(result);
      retryCount.set(0);
    } catch (err) {
      error.set(err as Error);
    } finally {
      loading.set(false);
    }
  };

  const retry = () => {
    retryCount.set(c => c + 1);
    fetchData();
  };

  effect(() => {
    fetchData();
  });
}

return { data, error, loading, retryCount, retry };
}

```

Display Retry Count

```

function DataComponent() {
  const { data, error, loading, retryCount, retry } = useDataWithRetry('/api/data');

  if (error()) {
    return (
      <div className="error">
        <p>Failed to load data</p>
        {retryCount() > 0 && (
          <p className="retry-info">Retry attempt: {retryCount()}</p>
        )}
        <button onClick={retry}>Try Again</button>
      </div>
    );
  }

  return <Data data={data()} />;
}

```

Query Error Handling

With `createQuery`

```

import { createQuery } from '@philjs/core';

const userQuery = createQuery({
  key: (userId: string) => ['user', userId],
  fetcher: async (userId: string) => {
    const res = await fetch(`api/users/${userId}`);

    if (!res.ok) {
      throw new Error('Failed to fetch user');
    }

    return res.json();
  },
  retry: 3, // Auto-retry 3 times
  retryDelay: (attemptIndex) => Math.min(1000 * 2 ** attemptIndex, 30000),
  onError: (error) => {
    console.error('Query error:', error);
    // Track error in analytics
    analytics.track('query_error', { error: error.message });
  }
});

function UserProfile({ userId }: { userId: string }) {
  const { data, error, loading, refetch } = userQuery(userId);

  if (error()) {
    return (
      <ErrorDisplay
        error={error()!}
        onRetry={refetch}
      />
    );
  }

  return <Profile user={data()!} />;
}

```

Error Types

Custom Error Classes

```

class NetworkError extends Error {
  constructor(message: string) {
    super(message);
    this.name = 'NetworkError';
  }
}

class NotFoundError extends Error {
  constructor(resource: string) {
    super(`${resource} not found`);
    this.name = 'NotFoundError';
  }
}

class UnauthorizedError extends Error {
  constructor() {
    super('Unauthorized');
    this.name = 'UnauthorizedError';
  }
}

class ValidationError extends Error {
  constructor(message: string, public fields?: Record<string, string>) {
    super(message);
    this.name = 'ValidationError';
  }
}

```

Handle Different Error Types

```

async function fetchUser(id: string) {
  try {
    const res = await fetch(` /api/users/${id}`);

    if (res.status === 401) {
      throw new UnauthorizedError();
    }

    if (res.status === 404) {
      throw new NotFoundError('User');
    }

    if (!res.ok) {
      throw new NetworkError(`HTTP ${res.status}`);
    }

    return await res.json();
  } catch (error) {
    if (error instanceof TypeError) {
      throw new NetworkError('Network request failed');
    }
    throw error;
  }
}

// Usage
function UserProfile({ id }: { id: string }) {
  const { error } = useData(id);

  if (error()) {
    const err = error()!;

    if (err instanceof UnauthorizedError) {
      return <Login />;
    }

    if (err instanceof NotFoundError) {
      return <NotFound />;
    }

    if (err instanceof NetworkError) {
      return <NetworkErrorMessage error={err} />;
    }

    return <GenericError error={err} />;
  }

  // ... render data
}

```

Error Boundaries

Data Error Boundary

```

import { ErrorBoundary } from '@philjs/core';

function DataErrorBoundary({ children }: { children: any }) {
  return (
    <ErrorBoundary
      fallback={(error, reset) => (
        <div className="error-boundary">
          <h2>Something went wrong</h2>
          <p>{error.message}</p>

          <div className="actions">
            <button onClick={reset}>Try Again</button>
            <button onClick={() => window.location.reload()}>
              Reload Page
            </button>
          </div>
        </div>
      )}
    >
    {children}
  </ErrorBoundary>
);
}

// Usage
function App() {
  return (
    <DataErrorBoundary>
      <Dashboard />
    </DataErrorBoundary>
  );
}

```

Fallback Data

Cached Fallback

```
function useDataWithFallback<T>(url: string, fallback: T) {
  const data = signal<T>(fallback);
  const error = signal<Error | null>(null);

  effect(() => {
    fetch(url)
      .then(res => res.json())
      .then(json => data.set(json))
      .catch(err => {
        error.set(err);
        // Keep fallback data on error
        console.warn('Using fallback data due to error:', err);
      });
  });

  return { data, error, usingFallback: () => !error() };
}
```

LocalStorage Fallback

```
function useDataWithCache<T>(url: string, cacheKey: string) {
  const data = signal<T | null>(null);
  const error = signal<Error | null>(null);
  const fromCache = signal(false);

  effect(() => {
    // Try to Load from cache first
    const cached = localStorage.getItem(cacheKey);
    if (cached) {
      try {
        data.set(JSON.parse(cached));
        fromCache.set(true);
      } catch {}}
    }

    // Fetch fresh data
    fetch(url)
      .then(res => res.json())
      .then(json => {
        data.set(json);
        fromCache.set(false);
        localStorage.setItem(cacheKey, JSON.stringify(json));
        error.set(null);
      })
      .catch(err => {
        error.set(err);
        // If fetch fails and no cache, data remains null
      });
  });

  return { data, error, fromCache };
}

// Usage
function Products() {
  const { data, error, fromCache } = useDataWithCache(
    '/api/products',
    'products-cache'
  );

  return (
    <div>
      {fromCache() && (
        <div className="cache-notice">
          Showing cached data{error() && ' (failed to refresh)'}
        </div>
      )}
      {data() && <ProductList products={data()} />}
    </div>
  );
}
```

User-Friendly Error Messages

Error Message Mapping

```

function getUserFriendlyMessage(error: Error): string {
  const errorMessages: Record<string, string> = {
    'NetworkError': 'Unable to connect. Please check your internet connection.',
    'NotFoundError': 'The requested item could not be found.',
    'UnauthorizedError': 'You need to log in to view this content.',
    'ValidationError': 'Please check your input and try again.',
    'TimeoutError': 'The request took too long. Please try again.',
  };

  return errorMessages[error.name] || 'An unexpected error occurred.';
}

function ErrorDisplay({ error, onRetry }: {
  error: Error;
  onRetry: () => void;
}) {
  return (
    <div className="error-display">
      <div className="error-icon">⚠</div>
      <h3>Oops!</h3>
      <p>{getUserFriendlyMessage(error)}</p>

      <div className="error-actions">
        <button onClick={onRetry} className="primary">
          Try Again
        </button>
        <button onClick={() => window.history.back()}>
          Go Back
        </button>
      </div>
    </div>
  );
}

```

Partial Failures

Some Requests Succeed

```

async function fetchAllData() {
  const results = await Promise.allSettled([
    fetch('/api/users').then(r => r.json()),
    fetch('/api/posts').then(r => r.json()),
    fetch('/api/comments').then(r => r.json())
  ]);

  return {
    users: results[0].status === 'fulfilled' ? results[0].value : null,
    posts: results[1].status === 'fulfilled' ? results[1].value : null,
    comments: results[2].status === 'fulfilled' ? results[2].value : null,
    errors: results
      .map((r, i) => r.status === 'rejected' ? { index: i, reason: r.reason } : null)
      .filter(Boolean)
  };
}

function Dashboard() {
  const { data, errors } = usePartialData(fetchAllData);

  return (
    <div>
      {errors().length > 0 && (
        <div className="partial-error-banner">
          Some data couldn't be loaded. Showing available content.
        </div>
      )}
      {data()?.users && <UsersList users={data()!.users} />}
      {data()?.posts && <PostsList posts={data()!.posts} />}
      {data()?.comments && <CommentsList comments={data()!.comments} />}
    </div>
  );
}

```

Error Logging

Log to Service

```

function logError(error: Error, context?: Record<string, any>) {
  // Log to service (Sentry, LogRocket, etc.)
  fetch('/api/errors', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      message: error.message,
      stack: error.stack,
      name: error.name,
      context,
      userAgent: navigator.userAgent,
      url: window.location.href,
      timestamp: Date.now()
    })
  }).catch(console.error);
}

// Use in queries
const userQuery = createQuery({
  key: (id) => ['user', id],
  fetcher: fetchUser,
  onError: (error, variables) => {
    logError(error, {
      queryKey: ['user', variables],
      component: 'UserProfile'
    });
  }
});

```

Best Practices

Always Handle Errors

```

// ⚡ Handle all error cases
const { data, error, loading } = useData(url);

if (loading()) return <Spinner />;
if (error()) return <Error error={error()} />;
return <Data data={data()} />

// ⚡ Ignore errors
const { data } = useData(url);
return <Data data={data()} />;

```

Provide Retry Mechanism

```

// ⚡ Let users retry
<button onClick={refetch}>Try Again</button>

// ⚡ Dead end
<div>Error loading data</div>

```

Show Loading States

```

// ⚡ Show Loading indicator
{loading() && <Spinner />}
{error() && <Error />}

// ⚡ Blank screen while Loading

```

Use Appropriate Retry Strategies

```

// ⚡ Retry for transient errors
retry: 3,
retryDelay: exponentialBackoff

// ⚡ Retry for client errors (400-499)
// Don't retry validation errors

```

Log Errors for Debugging

```

// ⚡ Log to error tracking service
onError: (error) => {
  logError(error);
}

// ⚡ Silent failure
onError: () => {}

```

Summary

You've learned:

□ Basic error handling with try-catch □ Automatic retry with exponential backoff □ Query-level error handling □ Custom error types □ Error boundaries □ Fallback data strategies
□ User-friendly error messages □ Handling partial failures □ Error logging □ Best practices
Proper error handling creates resilient applications!

Next: [Static Generation](#) → Generate pages at build time for blazing speed

Real-Time Data

Build live, real-time applications with WebSockets, Server-Sent Events, and polling.

What You'll Learn

- WebSocket connections
- Server-Sent Events (SSE)
- Polling strategies
- Real-time queries
- Optimistic updates for real-time
- Best practices

WebSockets

Basic WebSocket Connection

```
import { signal, effect } from '@philjs/core';

function useWebSocket<T>(url: string) {
  const data = signal<T | null>(null);
  const connected = signal(false);
  const error = signal<Error | null>(null);

  effect(() => {
    const ws = new WebSocket(url);

    ws.onopen = () => {
      connected.set(true);
      error.set(null);
    };

    ws.onmessage = (event) => {
      try {
        const parsed = JSON.parse(event.data);
        data.set(parsed);
      } catch (err) {
        error.set(err as Error);
      }
    };
  });

  ws.onerror = (event) => {
    error.set(new Error('WebSocket error'));
  };

  ws.onclose = () => {
    connected.set(false);
  };
}

return () => {
  ws.close();
};

return { data, connected, error };
}
```

Using WebSocket

```

interface LiveStats {
  users: number;
  revenue: number;
  orders: number;
}

function LiveDashboard() {
  const { data, connected, error } = useWebSocket<LiveStats>(
    'wss://api.example.com/live-stats'
  );

  if (error()) {
    return <Error message={error().message} />;
  }

  return (
    <div>
      {!connected() && <div className="status">Connecting...</div>}

      {data() && (
        <div className="stats">
          <StatCard label="Users" value={data()!.users} />
          <StatCard label="Revenue" value={data()!.revenue} />
          <StatCard label="Orders" value={data()!.orders} />
        </div>
      )}
    </div>
  );
}

```

Send Messages

```

function useWebSocket(url: string) {
  const ws = signal<WebSocket | null>(null);
  const data = signal(null);

  effect(() => {
    const socket = new WebSocket(url);

    socket.onopen = () => ws.set(socket);
    socket.onmessage = (event) => data.set(JSON.parse(event.data));
    socket.onclose = () => ws.set(null);

    return () => socket.close();
  });

  const send = (message: any) => {
    if (ws()) {
      ws()!.send(JSON.stringify(message));
    }
  };

  return { data, send, connected: () => !!ws() };
}

// Usage
function Chat() {
  const { data, send, connected } = useWebSocket('wss://chat.example.com');

  const sendMessage = (text: string) => {
    send({ type: 'message', text });
  };

  return (
    <div>
      {data() && <Message data={data()!}> />}
      <ChatInput onSend={sendMessage} disabled={!connected()} />
    </div>
  );
}

```

Auto-Reconnect

```

function useWebSocket(url: string, maxRetries = 5) {
  const data = signal(null);
  const connected = signal(false);
  const retryCount = signal(0);

  effect(() => {
    let ws: WebSocket;
    let reconnectTimer: any;

    const connect = () => {
      ws = new WebSocket(url);

      ws.onopen = () => {
        connected.set(true);
        retryCount.set(0);
      };

      ws.onmessage = (event) => {
        data.set(JSON.parse(event.data));
      };

      ws.onclose = () => {
        connected.set(false);
      };
    };

    // Attempt reconnect
    if (retryCount() < maxRetries) {
      const delay = Math.min(1000 * Math.pow(2, retryCount()), 30000);

      reconnectTimer = setTimeout(() => {
        retryCount.set(c => c + 1);
        connect();
      }, delay);
    }
  });
}

connect();

return () => {
  clearTimeout(reconnectTimer);
  ws?.close();
};

return { data, connected, retryCount };
}

```

Server-Sent Events (SSE)

Basic SSE Connection

```

function useSSE<T>(url: string) {
  const data = signal<T | null>(null);
  const connected = signal(false);
  const error = signal<Error | null>(null);

  effect(() => {
    const eventSource = new EventSource(url);

    eventSource.onopen = () => {
      connected.set(true);
      error.set(null);
    };

    eventSource.onmessage = (event) => {
      try {
        const parsed = JSON.parse(event.data);
        data.set(parsed);
      } catch (err) {
        error.set(err as Error);
      }
    };

    eventSource.onerror = () => {
      connected.set(false);
      error.set(new Error('SSE connection error'));
    };
  });

  return () => {
    eventSource.close();
  };
}

return { data, connected, error };
}

```

Named Events

```
function useSSE(url: string) {
  const data = signal(null);

  effect(() => {
    const eventSource = new EventSource(url);

    // Listen for specific event types
    eventSource.addEventListener('update', (event) => {
      data.set(JSON.parse(event.data));
    });

    eventSource.addEventListener('delete', (event) => {
      console.log('Item deleted:', event.data);
    });

    eventSource.addEventListener('error', (event) => {
      console.error('Error event:', event);
    });
  });

  return () => eventSource.close();
}

return { data };
}
```

Server Implementation

```
// src/pages/api/events.ts
export async function GET(request: Request) {
  const stream = new ReadableStream({
    start(controller) {
      const encoder = new TextEncoder();

      // Send data every 5 seconds
      const interval = setInterval(() => {
        const data = {
          timestamp: Date.now(),
          value: Math.random()
        };

        controller.enqueue(
          encoder.encode(`data: ${JSON.stringify(data)}\n\n`)
        );
      }, 5000);

      // Cleanup
      request.signal.addEventListener('abort', () => {
        clearInterval(interval);
        controller.close();
      });
    }
  });

  return new Response(stream, {
    headers: {
      'Content-Type': 'text/event-stream',
      'Cache-Control': 'no-cache',
      'Connection': 'keep-alive'
    }
  });
}
```

Polling

Basic Polling

```

function usePolling<T>(url: string, interval = 5000) {
  const data = signal<T | null>(null);
  const loading = signal(true);

  effect(() => {
    const fetchData = async () => {
      try {
        const res = await fetch(url);
        data.set(await res.json());
        loading.set(false);
      } catch (error) {
        console.error('Polling error:', error);
      }
    };

    fetchData();
    const timer = setInterval(fetchData, interval);
  });

  return () => clearInterval(timer);
}

return { data, loading };
}

```

Adaptive Polling

```

function useAdaptivePolling<T>(url: string) {
  const data = signal<T | null>(null);
  const interval = signal(5000); // Start with 5s

  effect(() => {
    let timer: any;

    const fetchData = async () => {
      const start = Date.now();

      try {
        const res = await fetch(url);
        const newData = await res.json();

        // If data changed, poll faster
        if (JSON.stringify(newData) !== JSON.stringify(data())) {
          interval.set(Math.max(1000, interval() / 2));
        } else {
          // If no change, slow down
          interval.set(Math.min(30000, interval() * 1.5));
        }

        data.set(newData);
      } catch (error) {
        console.error(error);
      }
    }

    // Schedule next poll
    timer = setTimeout(fetchData, interval());
  });

  fetchData();

  return () => clearTimeout(timer);
});

return { data, interval };
}

```

Conditional Polling

```
function useConditionalPolling<T>(
  url: string,
  shouldPoll: () => boolean,
  interval = 5000
) {
  const data = signal<T | null>(null);

  effect(() => {
    if (!shouldPoll()) return;

    const fetchData = async () => {
      const res = await fetch(url);
      data.set(await res.json());
    };

    fetchData();
    const timer = setInterval(fetchData, interval);
  });

  return () => clearInterval(timer);
}

// Usage
function Dashboard() {
  const isActive = signal(true);

  // Only poll when dashboard is active
  const { data } = useConditionalPolling(
    '/api/stats',
    () => isActive(),
    10000
  );

  return <StatsWidget stats={data()} />;
}
```

Real-Time Queries

Combine queries with real-time updates:

```

import { createQuery, setQueryData } from '@philjs/core';

const postsQuery = createQuery({
  key: () => ['posts'],
  fetcher: async () => {
    const res = await fetch('/api/posts');
    return res.json();
  }
});

function PostsList() {
  const { data } = postsQuery();

  // Subscribe to real-time updates
  effect(() => {
    const ws = new WebSocket('wss://api.example.com/posts');

    ws.onmessage = (event) => {
      const update = JSON.parse(event.data);

      // Update query cache in real-time
      setQueryData(['posts'], (old: Post[]) => {
        switch (update.type) {
          case 'create':
            return [update.post, ...old];

          case 'update':
            return old.map(post =>
              post.id === update.post.id ? update.post : post
            );

          case 'delete':
            return old.filter(post => post.id !== update.postId);

          default:
            return old;
        }
      });
    };
  });

  return () => ws.close();
};

return (
  <div>
    {data()?.map(post => (
      <PostCard key={post.id} post={post} />
    ))}
  </div>
);
}

```

Presence

Track who's online:

```

interface Presence {
  userId: string;
  username: string;
  lastSeen: number;
}

function usePresence(roomId: string) {
  const users = signal<Presence[]>([]);

  effect(() => {
    const ws = new WebSocket(`wss://api.example.com/presence/${roomId}`);

    ws.onopen = () => {
      // Announce presence
      ws.send(JSON.stringify({
        type: 'join',
        userId: currentUser.id,
        username: currentUser.name
      }));
    };

    ws.onmessage = (event) => {
      const message = JSON.parse(event.data);

      switch (message.type) {
        case 'user-joined':
          users.set(...users(), message.user);
          break;

        case 'user-left':
          users.set(users().filter(u => u.userId !== message.userId));
          break;

        case 'presence-list':
          users.set(message.users);
          break;
      }
    };
  });

  // Send heartbeat
  const heartbeat = setInterval(() => {
    ws.send(JSON.stringify({ type: 'heartbeat' }));
  }, 30000);

  return () => {
    clearInterval(heartbeat);
    ws.close();
  };
});

return { users };
}

```

Typing Indicators

Show when users are typing:

```

function useTypingIndicator(roomId: string) {
  const typingUsers = signal<string[]>([[]]);

  effect(() => {
    const ws = new WebSocket(`wss://api.example.com/typing/${roomId}`);

    ws.onmessage = (event) => {
      const { type, userId, username } = JSON.parse(event.data);

      if (type === 'typing-start') {
        typingUsers.set([...typingUsers], username);
      } else if (type === 'typing-stop') {
        typingUsers.set(typingUsers().filter(u => u !== username));
      }
    };
  });

  return () => ws.close();
});

const sendTyping = (isTyping: boolean) => {
  // Send to server
  fetch('/api/typing/${roomId}', {
    method: 'POST',
    body: JSON.stringify({ isTyping })
  });
};

return { typingUsers, sendTyping };
}

// Usage
function ChatInput() {
  const { sendTyping } = useTypingIndicator('room-123');
  let typingTimer: any;

  const handleInput = () => {
    sendTyping(true);

    clearTimeout(typingTimer);
    typingTimer = setTimeout(() => {
      sendTyping(false);
    }, 2000);
  };

  return <input onInput={handleInput} />;
}

```

Collaborative Editing

Real-time collaborative features:

```

function useCollaborativeDoc(docId: string) {
  const content = signal('');
  const cursors = signal<Map<string, number>>(new Map());

  effect(() => {
    const ws = new WebSocket(`wss://api.example.com/docs/${docId}`);

    ws.onmessage = (event) => {
      const message = JSON.parse(event.data);

      switch (message.type) {
        case 'content-change':
          content.set(message.content);
          break;

        case 'cursor-move':
          cursors.set(new Map(cursors()).set(
            message.userId,
            message.position
          ));
          break;
      }
    };
  });

  return () => ws.close();
});

const updateContent = (newContent: string) => {
  content.set(newContent);

  // Send to server
  ws.send(JSON.stringify({
    type: 'content-change',
    content: newContent
  }));
};

return { content, cursors, updateContent };
}

```

Best Practices

Choose the Right Technology

```

// ⚡ WebSocket for bidirectional communication
const chat = useWebSocket('wss://api.example.com/chat');

// ⚡ SSE for server-to-client only
const notifications = useSSE('/api/notifications');

// ⚡ Polling for simple updates
const { data } = usePolling('/api/stats', 30000);

// ⚡ WebSocket for occasional updates (overkill)
// ⚡ Polling for real-time chat (too slow)

```

Handle Reconnection

```

// ⚡ Implement automatic reconnection
function useWebSocket(url: string) {
  // ... reconnection logic
}

// ⚡ No reconnection handling
const ws = new WebSocket(url);

```

Clean Up Connections

```

// ⚡ Close connections when unmounting
effect(() => {
  const ws = new WebSocket(url);

  return () => {
    ws.close();
  };
});

// ⚡ Leave connections open
effect(() => {
  const ws = new WebSocket(url);
  // Missing cleanup
});

```

Throttle Updates

```
// ⚡ Throttle frequent updates
let updateTimer: any;

ws.onmessage = (event) => {
  clearTimeout(updateTimer);
  updateTimer = setTimeout(() => {
    data.set(JSON.parse(event.data));
  }, 100);
};

// ⚡ Update on every message (can cause performance issues)
ws.onmessage = (event) => {
  data.set(JSON.parse(event.data));
};
```

Show Connection Status

```
// ⚡ Show connection state
{!connected() && <div>Disconnected - Reconnecting...</div>}

// ⚡ Silent disconnection
```

Summary

You've learned:

- WebSocket connections and messaging
 - Auto-reconnection strategies
 - Server-Sent Events (SSE)
 - Polling (basic, adaptive, conditional)
 - Real-time query updates
 - Presence tracking
 - Typing indicators
 - Collaborative editing
 - Best practices
- Real-time features make apps feel alive and responsive!

Next: [Optimistic Updates](#) → Instant UI updates before server confirms

Server Functions

Build type-safe server functions that can be called directly from client components.

What You'll Learn

- Creating server functions
- Type safety across client/server
- Authentication and authorization
- Error handling
- Validation
- Best practices

What are Server Functions?

Server functions are functions that run on the server but can be called from the client as if they were local functions.

Benefits: - Type-safe (full TypeScript inference) - No API routes needed - Automatic serialization - Simple error handling - Direct database access

Basic Server Functions

Creating a Server Function

```
// src/server/users.ts
'use server';

import { db } from '@/lib/db';

export async function getUser(id: string) {
  const user = await db.users.findById(id);

  if (!user) {
    throw new Error('User not found');
  }

  return user;
}
```

Calling from Client

```
// src/pages/profile.tsx
import { getUser } from '@/server/users';
import { signal, effect } from '@philjs/core';

export default function Profile({ userId }: { userId: string }) {
  const user = signal(null);
  const loading = signal(true);

  effect(async () => {
    loading.set(true);
    try {
      const data = await getUser(userId);
      user.set(data);
    } catch (error) {
      console.error(error);
    } finally {
      loading.set(false);
    }
  });

  if (loading()) return <Spinner />;
}

return (
  <div>
    <h1>{user()!.name}</h1>
    <p>{user()!.email}</p>
  </div>
);
}
```

Type Safety

Full Type Inference

```
// src/server/posts.ts
'use server';

interface Post {
  id: string;
  title: string;
  content: string;
  authorId: string;
}

export async function getPost(id: string): Promise<Post> {
  const post = await db.posts.findById(id);
  return post;
}

export async function getPosts(limit = 10): Promise<Post[]> {
  const posts = await db.posts.findMany({ limit });
  return posts;
}

// Client - Full TypeScript autocomplete and type checking
import { getPost, getPosts } from '@/server/posts';

const post = await getPost('123'); // post is typed as Post
const posts = await getPosts(20); // posts is typed as Post[]
```

Complex Return Types

```
'use server';

interface UserWithPosts {
  id: string;
  name: string;
  email: string;
  posts: {
    id: string;
    title: string;
  }[];
}

export async function getUserWithPosts(userId: string): Promise<UserWithPosts> {
  const user = await db.users.findById(userId);
  const posts = await db.posts.findMany({ authorId: userId });

  return {
    id: user.id,
    name: user.name,
    email: user.email,
    posts: posts.map(p => ({ id: p.id, title: p.title }))
  };
}
```

CRUD Operations

Create

```
'use server';

import { z } from 'zod';

const CreatePostSchema = z.object({
  title: z.string().min(1).max(200),
  content: z.string().min(1),
  authorId: z.string()
});

export async function createPost(data: z.infer<typeof CreatePostSchema>) {
  // Validate
  const validated = CreatePostSchema.parse(data);

  // Create
  const post = await db.posts.create({
    ...validated,
    createdAt: new Date()
  });

  return post;
}
```

Read

```
'use server';

export async function getPost(id: string) {
  const post = await db.posts.findById(id);

  if (!post) {
    throw new Error('Post not found');
  }

  return post;
}

export async function listPosts(options?: {
  limit?: number;
  offset?: number;
  authorId?: string;
}) {
  const posts = await db.posts.findMany({
    limit: options?.limit || 10,
    offset: options?.offset || 0,
    where: options?.authorId ? { authorId: options.authorId } : undefined
  });

  return posts;
}
```

Update

```
'use server';

export async function updatePost(
  id: string,
  data: Partial<{ title: string; content: string }>
) {
  const post = await db.posts.findById(id);

  if (!post) {
    throw new Error('Post not found');
  }

  const updated = await db.posts.update(id, {
    ...data,
    updatedAt: new Date()
  });

  return updated;
}
```

Delete

```
'use server';

export async function deletePost(id: string) {
  const post = await db.posts.findById(id);

  if (!post) {
    throw new Error('Post not found');
  }

  await db.posts.delete(id);

  return { success: true };
}
```

Authentication

Check Current User

```
// src/server/auth.ts
'use server';

import { cookies } from '@philjs/core';
import { verifyToken } from '@lib/jwt';

export async function getCurrentUser() {
  const token = cookies().get('auth_token');

  if (!token) {
    return null;
  }

  const payload = verifyToken(token);

  if (!payload) {
    return null;
  }

  const user = await db.users.findById(payload.userId);
  return user;
}
```

Protected Server Functions

```
'use server';

import { getCurrentUser } from './auth';

export async function getMyPosts() {
  const user = await getCurrentUser();

  if (!user) {
    throw new Error('Not authenticated');
  }

  const posts = await db.posts.findMany({
    where: { authorId: user.id }
  });

  return posts;
}

export async function updateMyProfile(data: { name?: string; bio?: string }) {
  const user = await getCurrentUser();

  if (!user) {
    throw new Error('Not authenticated');
  }

  const updated = await db.users.update(user.id, data);
  return updated;
}
```

Authorization

Role-Based Access

```
'use server';

import { getCurrentUser } from './auth';

export async function deleteUser(userId: string) {
  const currentUser = await getCurrentUser();

  if (!currentUser) {
    throw new Error('Not authenticated');
  }

  if (currentUser.role !== 'admin') {
    throw new Error('Not authorized');
  }

  await db.users.delete(userId);

  return { success: true };
}
```

Resource Ownership

```
'use server';

import { getCurrentUser } from './auth';

export async function updatePost(postId: string, data: { title: string; content: string }) {
  const user = await getCurrentUser();

  if (!user) {
    throw new Error('Not authenticated');
  }

  const post = await db.posts.findById(postId);

  if (!post) {
    throw new Error('Post not found');
  }

  // Check ownership
  if (post.authorId !== user.id) {
    throw new Error('Not authorized to edit this post');
  }

  const updated = await db.posts.update(postId, data);
  return updated;
}
```

Validation

Zod Schemas

```
'use server';

import { z } from 'zod';

const UpdateUserSchema = z.object({
  name: z.string().min(1).max(100).optional(),
  email: z.string().email().optional(),
  bio: z.string().max(500).optional()
});

export async function updateUser(
  userId: string,
  data: z.infer<typeof UpdateUserSchema>
) {
  // Validate input
  const validated = UpdateUserSchema.parse(data);

  // Update user
  const user = await db.users.update(userId, validated);

  return user;
}
```

Custom Validation

```
'use server';

export async function createPost(data: {
  title: string;
  content: string;
  tags: string[];
}) {
  // Validate
  if (!data.title || data.title.length > 200) {
    throw new Error('Invalid title');
  }

  if (!data.content || data.content.length < 10) {
    throw new Error('Content too short');
  }

  if (data.tags.length > 10) {
    throw new Error('Too many tags');
  }

  // Check for profanity
  if (containsProfanity(data.content)) {
    throw new Error('Content contains inappropriate language');
  }

  const post = await db.posts.create(data);
  return post;
}
```

Error Handling

Custom Error Types

```
// src/lib/errors.ts
export class NotFoundError extends Error {
  constructor(message: string) {
    super(message);
    this.name = 'NotFoundError';
  }
}

export class UnauthorizedError extends Error {
  constructor(message: string) {
    super(message);
    this.name = 'UnauthorizedError';
  }
}

export class ValidationError extends Error {
  constructor(message: string, public details?: any) {
    super(message);
    this.name = 'ValidationError';
  }
}
```

```
'use server';

import { NotFoundError, UnauthorizedError } from '@/lib/errors';

export async function getPost(id: string) {
  const post = await db.posts.findById(id);

  if (!post) {
    throw new NotFoundError('Post not found');
  }

  return post;
}

export async function deletePost(id: string) {
  const user = await getCurrentUser();

  if (!user) {
    throw new UnauthorizedError('Not authenticated');
  }

  const post = await db.posts.findById(id);

  if (post.authorId !== user.id) {
    throw new UnauthorizedError('Not authorized');
  }

  await db.posts.delete(id);
}
```

Client-Side Error Handling

```
import { deletePost } from '@server/posts';

async function handleDelete(postId: string) {
  try {
    await deletePost(postId);
    alert('Post deleted');
  } catch (error) {
    if (error.name === 'UnauthorizedError') {
      alert('You do not have permission to delete this post');
    } else if (error.name === 'NotFoundError') {
      alert('Post not found');
    } else {
      alert('Failed to delete post');
    }
  }
}
```

Complex Operations

Transactions

```
'use server';

export async function transferFunds(
  fromUserId: string,
  toUserId: string,
  amount: number
) {
  const user = await getCurrentUser();

  if (!user || user.id !== fromUserId) {
    throw new Error('Not authorized');
  }

  // Use database transaction
  await db.transaction(async (tx) => {
    // Deduct from sender
    await tx.users.update(fromUserId, {
      balance: { decrement: amount }
    });

    // Add to recipient
    await tx.users.update(toUserId, {
      balance: { increment: amount }
    });

    // Create transaction record
    await tx.transactions.create({
      fromUserId,
      toUserId,
      amount,
      createdAt: new Date()
    });
  });

  return { success: true };
}
```

Batch Operations

```
'use server';

export async function bulkUpdatePosts(
  updates: Array<{ id: string; published: boolean }>
) {
  const user = await getCurrentUser();

  if (!user || user.role !== 'admin') {
    throw new Error('Not authorized');
  }

  const results = await Promise.all(
    updates.map(({ id, published }) =>
      db.posts.update(id, { published })
    )
  );

  return results;
}
```

File Operations

File Upload

```
'use server';

export async function uploadAvatar(file: File) {
  const user = await getCurrentUser();

  if (!user) {
    throw new Error('Not authenticated');
  }

  // Validate file
  if (!file.type.startsWith('image/')) {
    throw new Error('File must be an image');
  }

  if (file.size > 5 * 1024 * 1024) {
    throw new Error('File too large (max 5MB)');
  }

  // Upload to storage
  const buffer = await file.arrayBuffer();
  const filename = `avatars/${user.id}-${Date.now()}.jpg`;

  await storage.upload(filename, buffer);

  // Update user
  const updated = await db.users.update(user.id, {
    avatarUrl: `/uploads/${filename}`;
  });

  return updated;
}
```

Caching

Response Caching

```
'use server';

import { cache } from '@philjs/core';

export const getPublicPosts = cache(
  async () => {
    const posts = await db.posts.findMany({
      where: { published: true },
      orderBy: { createdAt: 'desc' },
      limit: 10
    });

    return posts;
  },
  ['public-posts'],
  { revalidate: 60 } // Cache for 60 seconds
);
```

Conditional Caching

```
'use server';

export async function getPosts(userId?: string) {
  if (!userId) {
    // Public posts - cacheable
    return getPublicPosts();
  }

  // User-specific posts - don't cache
  const posts = await db.posts.findMany({
    where: { authorId: userId }
  });

  return posts;
}
```

Background Jobs

Queue Tasks

```
'use server';

import { queue } from '@/lib/queue';

export async function sendWelcomeEmail(userId: string) {
  const user = await db.users.findById(userId);

  // Queue email sending
  await queue.add('send-email', {
    to: user.email,
    template: 'welcome',
    data: { name: user.name }
  });

  return { queued: true };
}
```

Best Practices

Keep Functions Small

```
// ⚡ Good - single responsibility
'use server';

export async function getUser(id: string) {
  return await db.users.findById(id);
}

export async function updateUser(id: string, data: any) {
  return await db.users.update(id, data);
}

// ⚡ Bad - does too much
export async function handleUser(action: string, id: string, data?: any) {
  if (action === 'get') return await db.users.findById(id);
  if (action === 'update') return await db.users.update(id, data);
  if (action === 'delete') return await db.users.delete(id);
}
```

Validate All Input

```
// ⚡ Always validate
'use server';

export async function createPost(data: unknown) {
  const validated = CreatePostSchema.parse(data);
  return await db.posts.create(validated);
}

// ⚡ Trust client input
export async function createPost(data: any) {
  return await db.posts.create(data);
}
```

Always Authenticate

```
// ⚡ Check auth for sensitive operations
'use server';

export async function deleteAccount() {
  const user = await getCurrentUser();
  if (!user) throw new Error('Not authenticated');

  await db.users.delete(user.id);
}

// ⚡ No auth check
export async function deleteAccount(userId: string) {
  await db.users.delete(userId);
}
```

Use Descriptive Names

```
// ⚡ Clear intent
export async function getUserPosts(userId: string) { }
export async function createBlogPost(data: CreatePostData) { }
export async function publishPost(postId: string) { }

// ⚡ Vague names
export async function get(id: string) { }
export async function doThing(data: any) { }
```

Summary

You've learned:

- Creating server functions with 'use server'
- Full type safety across client/server
- CRUD operations
- Authentication and authorization
- Input validation with Zod
- Error handling
- Complex operations and transactions
- File uploads
- Caching strategies
- Best practices

Server functions provide type-safe backend without API routes!

Next: [Queries](#) → Declarative data fetching with automatic caching

Server-Side Rendering (SSR)

Render pages on the server for each request to handle dynamic, personalized content.

What You'll Learn

- Server-side rendering basics
- Dynamic data fetching
- Personalization
- SSR vs SSG
- Performance optimization
- Best practices

What is SSR?

Server-Side Rendering generates HTML on the server for each request:

Benefits: - Always fresh data - Personalized content - SEO-friendly - Security (hide API keys)

Use for: - User dashboards - Personalized pages - Real-time data - Search results - Dynamic content

Basic SSR

Fetch on Server

```
// src/pages/dashboard.tsx
export async function getServerSideProps() {
  // Runs on every request
  const stats = await fetchDashboardStats();

  return {
    props: { stats }
  };
}

export default function Dashboard({ stats }: { stats: Stats }) {
  return (
    <div>
      <h1>Dashboard</h1>
      <StatsWidget stats={stats} />
    </div>
  );
}
```

With Request Context

```
export async function getServerSideProps({ req, res }: {
  req: Request;
  res: Response;
}) {
  // Access request headers
  const userAgent = req.headers.get('user-agent');

  // Get cookies
  const token = getCookie(req, 'auth_token');

  // Set response headers
  res.headers.set('Cache-Control', 'private, max-age=0');

  const data = await fetchData(token);

  return {
    props: { data }
  };
}
```

Authentication

Protected Pages

```

export async function getServerSideProps({ req }: { req: Request }) {
  const session = await getSession(req);

  if (!session) {
    return {
      redirect: {
        destination: '/login',
        permanent: false
      }
    };
  }

  const userData = await fetchUserData(session.userId);

  return {
    props: { userData }
  };
}

export default function Profile({ userData }: { userData: User }) {
  return (
    <div>
      <h1>Welcome, {userData.name}</h1>
      <p>{userData.email}</p>
    </div>
  );
}

```

Role-Based Access

```

export async function getServerSideProps({ req }: { req: Request }) {
  const session = await getSession(req);

  if (!session) {
    return { redirect: { destination: '/login', permanent: false } };
  }

  const user = await fetchUser(session.userId);

  if (user.role !== 'admin') {
    return { redirect: { destination: '/unauthorized', permanent: false } };
  }

  const adminData = await fetchAdminData();

  return {
    props: { adminData }
  };
}

export default function AdminPanel({ adminData }: { adminData: any }) {
  return <AdminDashboard data={adminData} />;
}

```

Personalization

User-Specific Content

```

export async function getServerSideProps({ req }: { req: Request }) {
  const session = await getSession(req);

  if (!session) {
    // Show generic content for guests
    const genericContent = await fetchGenericContent();
    return { props: { content: genericContent, isGuest: true } };
  }

  // Fetch personalized content
  const user = await fetchUser(session.userId);
  const recommendations = await fetchRecommendations(user.id);
  const recentlyViewed = await fetchRecentlyViewed(user.id);

  return {
    props: {
      user,
      recommendations,
      recentlyViewed,
      isGuest: false
    }
  };
}

export default function HomePage({ user, recommendations, recentlyViewed, isGuest }: {
  user?: User;
  recommendations?: Product[];
  recentlyViewed?: Product[];
  isGuest: boolean;
}) {
  if (isGuest) {
    return <GuestHomePage />;
  }

  return (
    <div>
      <h1>Welcome back, {user!.name}</h1>
      <RecommendedProducts products={recommendations!} />
      <RecentlyViewed products={recentlyViewed!} />
    </div>
  );
}

```

Geo-Location

```

export async function getServerSideProps({ req }: { req: Request }) {
  // Get user's Location from IP
  const ip = req.headers.get('x-forwarded-for') || req.socket.remoteAddress;
  const location = await getLocationFromIP(ip);

  // Fetch location-specific content
  const nearbyStores = await fetchNearbyStores(location);
  const localDeals = await fetchLocalDeals(location.city);

  return {
    props: {
      location,
      nearbyStores,
      localDeals
    }
  };
}

export default function StoreFinder({ location, nearbyStores, localDeals }: {
  location: Location;
  nearbyStores: Store[];
  localDeals: Deal[];
}) {
  return (
    <div>
      <h1>Stores near {location.city}</h1>
      <StoreList stores={nearbyStores} />
      <LocalDeals deals={localDeals} />
    </div>
  );
}

```

Dynamic Content

Search Results

```

export async function getServerSideProps({ query }: { query: Record<string, any> }) {
  const searchQuery = query.q as string;
  const page = parseInt(query.page as string) || 1;

  if (!searchQuery) {
    return {
      props: { results: [], query: '' }
    };
  }

  const results = await searchProducts(searchQuery, page);

  return {
    props: {
      results,
      query: searchQuery,
      page
    }
  };
}

export default function SearchResults({ results, query, page }: {
  results: Product[];
  query: string;
  page: number;
}) {
  return (
    <div>
      <h1>Search results for "{query}"</h1>
      <p>{results.length} results found</p>

      <ProductGrid products={results} />

      <Pagination current={page} />
    </div>
  );
}

```

Filtered Lists

```

export async function getServerSideProps({ query }: { query: Record<string, any> }) {
  const category = query.category as string || 'all';
  const sort = query.sort as string || 'popular';
  const page = parseInt(query.page as string) || 1;

  const products = await fetchProducts({
    category,
    sort,
    page,
    limit: 20
  });

  const categories = await fetchCategories();

  return {
    props: {
      products,
      categories,
      filters: { category, sort, page }
    }
  };
}

export default function ProductList({ products, categories, filters }: {
  products: Product[];
  categories: Category[];
  filters: any;
}) {
  return (
    <div>
      <Filters categories={categories} current={filters} />
      <ProductGrid products={products} />
    </div>
  );
}

```

Error Handling

Not Found

```

export async function getServerSideProps({ params }: { params: { id: string } }) {
  const product = await fetchProduct(params.id);

  if (!product) {
    return {
      notFound: true
    };
  }

  return {
    props: { product }
  };
}

// Renders 404 page if product not found

```

Custom Error

```

export async function getServerSideProps() {
  try {
    const data = await fetchData();

    return {
      props: { data }
    };
  } catch (error) {
    console.error('SSR error:', error);
  }

  return {
    props: {
      error: {
        message: 'Failed to load data',
        statusCode: 500
      }
    };
  }
}

export default function Page({ data, error }: {
  data?: any;
  error?: { message: string; statusCode: number };
}) {
  if (error) {
    return (
      <div className="error">
        <h1>Error {error.statusCode}</h1>
        <p>{error.message}</p>
      </div>
    );
  }

  return <Content data={data} />;
}

```

Caching

HTTP Caching

```

export async function getServerSideProps({ req, res }: {
  req: Request;
  res: Response;
}) {
  // Cache for 60 seconds
  res.headers.set(
    'Cache-Control',
    'public, s-maxage=60, stale-while-revalidate=120'
  );

  const data = await fetchData();

  return {
    props: { data }
  };
}

```

Conditional Caching

```

export async function getServerSideProps({ req, res }: {
  req: Request;
  res: Response;
}) {
  const session = await getSession(req);

  if (!session) {
    // Cache for guests
    res.headers.set('Cache-Control', 'public, max-age=300');
  } else {
    // Don't cache for Logged-in users
    res.headers.set('Cache-Control', 'private, no-cache');
  }

  const data = await fetchData(session?.userId);

  return {
    props: { data }
  };
}

```

Performance

Parallel Data Fetching

```

export async function getServerSideProps() {
  // Fetch in parallel
  const [user, posts, comments] = await Promise.all([
    fetchUser(),
    fetchPosts(),
    fetchComments()
  ]);

  return {
    props: { user, posts, comments }
  };
}

```

Minimal Data

```

export async function getServerSideProps() {
  // Fetch only what's needed
  const products = await fetchProducts({
    fields: ['id', 'name', 'price', 'image'], // Only these fields
    limit: 20
  });

  return {
    props: { products }
  };
}

```

Database Connection Pooling

```

// Lib/db.ts
import { Pool } from 'pg';

const pool = new Pool({
  max: 20, // Maximum connections
  idleTimeoutMillis: 30000,
  connectionTimeoutMillis: 2000
});

export async function query(sql: string, params?: any[]) {
  const client = await pool.connect();
  try {
    const result = await client.query(sql, params);
    return result.rows;
  } finally {
    client.release();
  }
}

```

SSR vs SSG

When to Use Each

```
// ⚡ SSR for dynamic, personalized content
export async function getServerSideProps({ req }: { req: Request }) {
  const user = await getUser(req);
  return { props: { user } };
}

// ⚡ SSG for static content
export async function getStaticProps() {
  const posts = await fetchPosts();
  return { props: { posts }, revalidate: 3600 };
}
```

Hybrid Approach

```
// Static shell with dynamic data
export async function getStaticProps() {
  // Static parts
  const layout = await fetchLayout();

  return {
    props: { layout },
    revalidate: 86400 // Daily
  };
}

export default function Page({ layout }: { layout: any }) {
  // Client-side fetch for dynamic data
  const { data } = useQuery(fetchUserData);

  return (
    <Layout config={layout}>
      <DynamicContent data={data()} />
    </Layout>
  );
}
```

Best Practices

Minimize Server Time

```
// ⚡ Quick server-side work
export async function getServerSideProps() {
  const data = await fetchData(); // Fast query
  return { props: { data } };
}

// ⚡ Slow server-side processing
export async function getServerSideProps() {
  const data = await complexProcessing(); // 5 seconds
  return { props: { data } };
}
```

Use Appropriate Caching

```
// ⚡ Cache public pages
res.headers.set('Cache-Control', 'public, s-maxage=60');

// ⚡ Don't cache private data
res.headers.set('Cache-Control', 'private, no-cache');

// ⚡ Cache user-specific data publicly
res.headers.set('Cache-Control', 'public, max-age=3600');
```

Handle Errors Gracefully

```
// ⚡ Provide fallback
export async function getServerSideProps() {
  try {
    const data = await fetchData();
    return { props: { data } };
  } catch (error) {
    return { props: { data: null, error: true } };
  }
}

// ⚡ Let errors crash
export async function getServerSideProps() {
  const data = await fetchData(); // Throws
  return { props: { data } };
}
```

Redirect Early

```
// ⚡ Redirect before fetching data
export async function getServerSideProps({ req }: { req: Request }) {
  const session = await getSession(req);

  if (!session) {
    return { redirect: { destination: '/login', permanent: false } };
  }

  const data = await fetchData(session.userId);
  return { props: { data } };
}

// ⚡ Fetch then redirect
export async function getServerSideProps({ req }: { req: Request }) {
  const data = await fetchData(); // Wasted work
  const session = await getSession(req);

  if (!session) {
    return { redirect: { destination: '/login', permanent: false } };
  }

  return { props: { data } };
}
```

Log Performance

```
export async function getServerSideProps() {
  const start = Date.now();

  const data = await fetchData();

  const duration = Date.now() - start;

  if (duration > 1000) {
    console.warn(`Slow SSR: ${duration}ms`);
  }

  return {
    props: { data }
  };
}
```

Summary

You've learned:

- Server-side rendering basics
- Dynamic data fetching per request
- Authentication and authorization
- Personalization strategies
- Dynamic content (search, filters)
- Error handling (not found, custom errors)
- HTTP caching for SSR
- Performance optimization
- SSR vs SSG comparison
- Best practices

SSR provides fresh, personalized content for every user!

Congratulations! You've completed the Data Fetching section. You now know how to fetch, cache, and manage data in PhilJS applications using every available pattern.

Static Site Generation (SSG)

Generate pages at build time for maximum performance and SEO.

What You'll Learn

- Static page generation
- Dynamic path generation
- Incremental Static Regeneration (ISR)
- When to use SSG
- SEO optimization
- Best practices

What is SSG?

Static Site Generation pre-renders pages at build time.

Benefits: - Blazing fast (served from CDN) - Perfect SEO (fully rendered HTML) - No server required - Lower costs - Maximum security

Use for: - Marketing pages - Documentation - Blogs - Product catalogs - Any content that doesn't change often

Basic Static Generation

Simple Static Page

```
// src/pages/about.tsx
export default function About() {
  return (
    <div>
      <h1>About Us</h1>
      <p>We build amazing software.</p>
    </div>
  );
}

// This page is automatically statically generated
```

With Data Fetching

```
// src/pages/blog/index.tsx
interface Post {
  slug: string;
  title: string;
  excerpt: string;
}

export async function getStaticProps() {
  // Fetch at build time
  const posts = await fetch('https://api.example.com/posts')
    .then(r => r.json());

  return {
    props: { posts }
  };
}

export default function Blog({ posts }: { posts: Post[] }) {
  return (
    <div>
      <h1>Blog</h1>
      {posts.map(post => (
        <article key={post.slug}>
          <h2>{post.title}</h2>
          <p>{post.excerpt}</p>
          <Link href={`/blog/${post.slug}`}>Read more</Link>
        </article>
      ))}
    </div>
  );
}
```

Dynamic Paths

Generate Static Params

```
// src/pages/blog/[slug].tsx
interface Post {
  slug: string;
  title: string;
  content: string;
  publishedAt: string;
}

// Tell Next.js which paths to generate
export async function generateStaticParams() {
  const posts = await fetch('https://api.example.com/posts')
    .then(r => r.json());

  return posts.map((post: Post) => ({
    slug: post.slug
  }));
}

// Fetch data for each path
export async function getStaticProps({ params }: { params: { slug: string } }) {
  const post = await fetch(`https://api.example.com/posts/${params.slug}`)
    .then(r => r.json());

  return {
    props: { post }
  };
}

export default function BlogPost({ post }: { post: Post }) {
  return (
    <article>
      <h1>{post.title}</h1>
      <time>{new Date(post.publishedAt).toLocaleDateString()}</time>
      <div dangerouslySetInnerHTML={{ __html: post.content }} />
    </article>
  );
}
```

Nested Dynamic Routes

```
// src/pages/[category]/[product].tsx
export async function generateStaticParams() {
  const categories = await fetchCategories();

  const paths = [];

  for (const category of categories) {
    const products = await fetchProductsByCategory(category.slug);

    for (const product of products) {
      paths.push({
        category: category.slug,
        product: product.slug
      });
    }
  }

  return paths;
}

export async function getStaticProps({ params }: { params: { category: string; product: string } }) {
  const product = await fetchProduct(params.category, params.product);

  return {
    props: { product }
  };
}

export default function Product({ product }: { product: Product }) {
  return (
    <div>
      <h1>{product.name}</h1>
      <p>{product.description}</p>
      <button>Add to Cart - ${product.price}</button>
    </div>
  );
}
```

Incremental Static Regeneration (ISR)

Revalidate After Time

```

export async function getStaticProps() {
  const posts = await fetchPosts();

  return {
    props: { posts },
    revalidate: 60 // Regenerate every 60 seconds
  };
}

```

How it works: 1. Initial request: Serve stale page 2. Background: Regenerate page 3. Next request: Serve fresh page

On-Demand Revalidation

```

// src/pages/api/revalidate.ts
export async function POST(request: Request) {
  const { path } = await request.json();

  // Revalidate specific path
  await revalidatePath(path);

  return new Response(JSON.stringify({ revalidated: true }), {
    status: 200
  });
}

```

Trigger from CMS webhook:

```

// When content is updated in CMS
async function onContentUpdate(slug: string) {
  await fetch('https://yoursite.com/api/revalidate', {
    method: 'POST',
    body: JSON.stringify({ path: `/blog/${slug}` })
  });
}

```

Fallback Behavior

Static Paths with Fallback

```

export async function generateStaticParams() {
  // Only generate most popular posts at build time
  const popularPosts = await fetchPopularPosts(100);

  return popularPosts.map(post => ({
    slug: post.slug
  }));
}

export async function getStaticProps({ params }: { params: { slug: string } }) {
  const post = await fetchPost(params.slug);

  if (!post) {
    return {
      notFound: true
    };
  }

  return {
    props: { post },
    revalidate: 3600 // Revalidate hourly
  };
}

export const fallback = 'blocking'; // Generate on first request

export default function Post({ post }: { post: Post }) {
  return <PostView post={post} />;
}

```

Fallback options: - false - 404 for unknown paths - true - Show loading state, then generate - 'blocking' - Wait for generation (no loading state)

SEO Optimization

Meta Tags

```
// src/pages/blog/[slug].tsx
export async function getStaticProps({ params }: { params: { slug: string } }) {
  const post = await fetchPost(params.slug);

  return {
    props: { post },
    meta: {
      title: post.title,
      description: post.excerpt,
      openGraph: {
        title: post.title,
        description: post.excerpt,
        image: post.coverImage,
        type: 'article'
      }
    }
  };
}
```

Structured Data

```
export default function Product({ product }: { product: Product }) {
  const structuredData = {
    '@context': 'https://schema.org',
    '@type': 'Product',
    name: product.name,
    description: product.description,
    image: product.image,
    offers: {
      '@type': 'Offer',
      price: product.price,
      priceCurrency: 'USD'
    }
  };

  return (
    <div>
      <script
        type="application/ld+json"
        dangerouslySetInnerHTML={{ __html: JSON.stringify(structuredData) }}
      />

      <h1>{product.name}</h1>
      <p>{product.description}</p>
    </div>
  );
}
```

Sitemap Generation

```
// src/pages/sitemap.xml.ts
export async function GET() {
  const posts = await fetchAllPosts();
  const products = await fetchAllProducts();

  const urls = [
    { url: '/', changefreq: 'daily', priority: 1.0 },
    ...posts.map(post => ({
      url: `/blog/${post.slug}`,
      changefreq: 'weekly',
      priority: 0.8,
      lastmod: post.updatedAt
    })),
    ...products.map(product => ({
      url: `/products/${product.slug}`,
      changefreq: 'weekly',
      priority: 0.7
    }))
  ];
}

const sitemap = `<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
${urls.map(({ url, changefreq, priority, lastmod }) => `
<url>
  <loc>https://example.com${url}</loc>
  ${lastmod ? `<lastmod>${lastmod}</lastmod>` : ''}
  <changefreq>${changefreq}</changefreq>
  <priority>${priority}</priority>
</url>
`).join('')}
</urlset>`;

return new Response(sitemap, {
  headers: {
    'Content-Type': 'application/xml'
  }
});
}
```

Markdown/MDX Support

[Blog from Markdown Files](#)

```

import fs from 'fs';
import path from 'path';
import matter from 'gray-matter';
import { marked } from 'marked';

export async function generateStaticParams() {
  const postsDirectory = path.join(process.cwd(), 'content/posts');
  const filenames = fs.readdirSync(postsDirectory);

  return filenames
    .filter(name => name.endsWith('.md'))
    .map(filename => ({
      slug: filename.replace(/\.\.md$/, '')
    }));
}

export async function getStaticProps({ params }: { params: { slug: string } }) {
  const filePath = path.join(process.cwd(), 'content/posts', `${params.slug}.md`);
  const fileContents = fs.readFileSync(filePath, 'utf8');

  const { data, content } = matter(fileContents);
  const html = marked(content);

  return {
    props: {
      title: data.title,
      date: data.date,
      content: html
    }
  };
}

export default function Post({ title, date, content }: {
  title: string;
  date: string;
  content: string;
}) {
  return (
    <article>
      <h1>{title}</h1>
      <time>{date}</time>
      <div dangerouslySetInnerHTML={{ __html: content }} />
    </article>
  );
}

```

Build Optimization

Parallel Generation

```

// philjs.config.ts
export default {
  build: {
    parallel: true, // Generate pages in parallel
    maxWorkers: 4 // Number of worker threads
  }
};

```

Selective Generation

```

// Only generate first N pages at build time
export async function generateStaticParams() {
  const allPosts = await fetchAllPosts();

  // Generate top 100 at build time
  // Rest generated on demand
  return allPosts.slice(0, 100).map(post => ({
    slug: post.slug
  }));
}

export const fallback = 'blocking'; // Generate others on first request

```

Hybrid: SSG + Client-Side

Static Shell, Dynamic Data

```
// Static at build time
export async function getStaticProps() {
  const initialData = await fetchInitialData();

  return {
    props: { initialData }
  };
}

export default function Dashboard({ initialData }: { initialData: any }) {
  // Client-side fetch for real-time data
  const { data: liveData } = useLiveData();

  return (
    <div>
      {/* Static content */}
      <header>{initialData.title}</header>

      {/* Live data */}
      <Stats data={liveData} || initialData.stats} />
    </div>
  );
}
```

Best Practices

Use for Stable Content

```
// ⚡ SSG for blogs, docs, marketing
export async function getStaticProps() {
  const posts = await fetchPosts();
  return { props: { posts }, revalidate: 3600 };
}

// ⚡ SSG for user dashboards (too dynamic)
```

Set Appropriate Revalidation

```
// ⚡ Match revalidation to content update frequency
return {
  props: { data },
  revalidate: 3600 // 1 hour for blog
};

// ⚡ Too frequent (defeats purpose)
return {
  props: { data },
  revalidate: 1 // Every second
};
```

Generate Popular Paths

```
// ⚡ Generate most-visited pages
export async function generateStaticParams() {
  const popular = await fetchPopularProducts(1000);
  return popular.map(p => ({ slug: p.slug }));
}

// ⚡ Generate everything (slow builds)
export async function generateStaticParams() {
  const all = await fetchAllProducts(); // 1 million products
  return all.map(p => ({ slug: p.slug }));
}
```

Use Fallback Wisely

```
// ⚡ Fallback for dynamic content
export const fallback = 'blocking';

// ⚡ No fallback with incomplete generation
export const fallback = false;
// Many 404s!
```

Optimize Images

```
// 🚧 Use optimized images

// 🚧 Large unoptimized images
![Large unoptimized image]({product.rawImage}/)
```

Summary

You've learned:

- Static page generation □ Dynamic path generation with `generateStaticParams` □ Data fetching at build time with `getStaticProps` □ Incremental Static Regeneration (ISR) □ Fallback strategies □ SEO optimization (meta tags, structured data, sitemaps) □ Markdown/MDX support □ Build optimization □ Hybrid SSG + client-side □ Best practices SSG delivers maximum performance and perfect SEO!

Next: [Server-Side Rendering](#) → Render pages on each request

Forms

PhilJS ships a typed form API. Use it for validation, error tracking, and clean submit handling.

```
import { useForm, createField, validators } from "@philjs/core";

type SignupForm = {
  email: string;
  name: string;
  consent: boolean;
};

const Fields = createField<SignupForm>();

export function Signup() {
  const form = useForm<SignupForm>({
    schema: {
      email: validators.email().required(),
      name: validators.string().min(2),
      consent: validators.boolean().required(),
    },
    onSubmit: async (values) => {
      await fetch("/api/signup", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(values),
      });
    },
  });
}

return (
  <form onSubmit={form.handleSubmit}>
    <Fields.Input form={form} name="email" label="Email" />
    <Fields.Input form={form} name="name" label="Name" />
    <Fields.Checkbox form={form} name="consent" label="I agree" />
    <button type="submit" disabled={() => !form.isValid()}>
      Create account
    </button>
  </form>
)
```

Why use the form API

- Built-in validation with typed schemas
- Signals for form state (`isSubmitting`, `isValid`, `errors`)
- Accessible field helpers with `aria-invalid` and error IDs
- Works with SSR and progressive enhancement (real `<form>` submissions)

Validation strategies

- Schema-first with `validators` or Zod/Valibot
- Field-level validation for fast feedback; form-level for cross-field rules.
- Async validation (e.g., username availability) using effects tied to field signals.

Error rendering

- Use `form.errors().name` to show inline messages.
- Add `role="alert"` for error summaries.
- Ensure inputs get `aria-invalid` and `aria-describedby` automatically when using field helpers.

Submission patterns

- For server-trustable mutations, pair `useForm` with a route action.
- Use optimistic UI for perceived speed; reconcile on server response.
- Disable submit while `isSubmitting` or when invalid.

```
<button type="submit" disabled={() => !form.isValid() || form.isSubmitting()}>
  Save
</button>
```

File uploads

- Use `<input type="file">` and access files via `formData`.
- Stream to storage in actions; return progress via SSE/WebSocket if needed.
- Show upload status with a resource or store slice.

Testing forms

- Component tests: render the form, type into inputs with `fireEvent.input`, `submit`, assert handlers fired with parsed values.
- Integration: use MSW to mock server responses; assert error messages on 400s.
- E2E: Playwright to submit real forms; verify SSR + hydration keep inputs intact.

Checklist

- Schema validates required fields and constraints.
- Errors surfaced inline and announced (`role="alert"`).
- Submit disabled when invalid or submitting.
- Handles server errors gracefully (action result/error boundary).
- File uploads handled with proper encoding and progress UX.

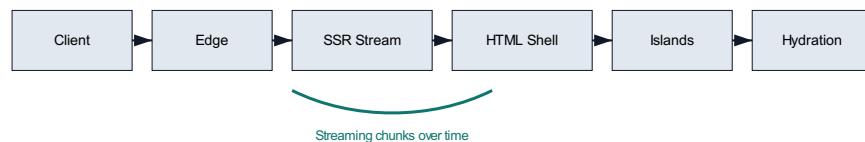
Try it now: form + action pair

```
export const contactAction = action(async ({ formData }) => {
  const payload = {
    email: String(formData.get('email') ?? ''),
    message: String(formData.get('message') ?? ''),
  };
  if (!payload.email || !payload.message) return Err(new Error('Missing fields'));
  await sendMessage(payload);
});
```

In the component, call `useForm` and submit to the action; show `form.errors()` for validation and action errors.

SSR Overview

PhilJS SSR is streaming-first and designed for islands. You can render routes directly with the server adapters.



SSR streaming pipeline

Basic Node server

```
import { createServer } from "node:http";
import { createNodeHttpHandler } from "@philjs/ssr";
import { createAppRouter } from "@philjs/router";

const routes = [
  { path: "/", component: () => <main>Home</main> },
  { path: "/about", component: () => <main>About</main> },
];

const handler = createNodeHttpHandler({ routes });

createServer(handler).listen(3000, () => {
  console.log("PhilJS SSR running on http://localhost:3000");
});
```

Render to string

```
import { renderToString } from '@philjs/core';

const html = renderToString(<main>Server render</main>);
```

Streaming SSR

```
import { renderToString } from '@philjs/ssr';

const stream = renderToString(<App />);
```

Architecture

- **Router-aware SSR**: loaders run server-side, data is serialized, and hydration picks up without refetching.
- **Streaming-first**: HTML flushes early while slower panels fill in later.
- **Islands**: hydrate only what needs to be interactive; keep static shells cheap.
- **Adapters**: Vercel/Netlify/Cloudflare/Bun/Deno/AWS/Node adapters handle platform specifics.

Hydration and resumability

- Keep hydration targets small; split heavy widgets into islands.
- Avoid generating new object identities in render that break equality across server/client.
- Use stable keys in lists to align server and client DOM.
- Pass loader data via serialized payloads; avoid re-running fetches on the client unless needed.

Caching and revalidate

- Use loader cache tags and `revalidate` hints to drive ISR/edge cache lifetimes.
- For HTML caching, pair adapter-level cache with per-route invalidation (see Loaders chapter).
- Keep HTML payloads small; stream large panels instead of blocking first paint.

Edge vs regional SSR

- Edge (Vercel/Netlify/CF/Bun/Deno): lowest latency, but smaller CPU/memory/time budgets.
- Regional (Node/AWS): more headroom for heavy work (PDFs, image processing), but higher latency.
- Choose per-route: marketing pages at edge, heavy exports regionally.

Rendering modes

- `renderToString` – full HTML string (useful for tests and small pages).
- `renderToString` – streaming; best default for production.
- `renderToReadableStream` (edge-friendly) for platforms that expect Web Streams.

Environment and secrets

- Do not bundle secrets; read from env at runtime (adapter passes through).
- Avoid Node-only APIs when targeting edge runtimes; stick to Web APIs where possible.

Error handling

- Wrap root with an error boundary for SSR; render user-friendly errors.
- Log SSR failures with route info and request id to observability pipeline.
- Provide fallbacks for partial streams if downstream data fails.

Testing SSR

- Use `renderToString` in unit tests to assert markup.
- Use Playwright against `philjs dev --ssr` or a preview build to validate hydration.
- Simulate slow loaders and verify streaming delivers shell + fallback content promptly.

Try it now: edge-ready stream

```
// edge-handler.ts
import { renderToReadableStream } from '@philjs/ssr';
import { App } from './App';

export default async function handle() {
  const body = await renderToReadableStream(<App />);
  return new Response(body, {
    headers: { 'content-type': 'text/html; charset=utf-8' }
  });
}
```

Deploy this with the Cloudflare/Vercel adapter and measure TTFB + first paint with Playwright traces.

Islands

Islands let you hydrate only the interactive parts of a page.

Mark interactive components

```
import { renderToString } from "@philjs/ssr";
import { Counter } from "./Counter";

const stream = renderToString(<App />, {
  interactiveComponents: new Set([Counter]),
});
```

Hydrate on the client

```
import { autoHydrateIslands, HydrationStrategy, registerIsland } from "@philjs/ssr";
import { Counter } from "./Counter";

registerIsland("Counter", Counter);
autoHydrateIslands(HydrationStrategy.VISIBLE);
```

Common patterns

- Hydrate on visibility for heavy widgets
- Hydrate on interaction for menus or editors
- Keep most content static for fast first paint

Choosing hydration strategies

- **Immediate:** for critical UI (nav, hero CTA).
- **Visible:** for below-the-fold charts/cards.
- **Idle:** for non-critical widgets (comments, recs).
- **On interaction:** for popovers/menus/editors; hydrate on first click.

Pick the lightest strategy that still feels instant for the user.

Splitting islands

- Keep islands small and focused (e.g., a chart, a form, a menu).
- Co-locate island registration with components for clarity.
- Share signals across islands only when necessary; prefer route-level data for consistency.

Data flow

- Pass loader data into islands as props; avoid re-fetching on hydration.
- If an island needs live data, use resources with cache tags to stay consistent with route data.

Styling and assets

- Hydrate islands with minimal CSS; prefer scoped styles or critical CSS in first chunk.
- Lazy-load heavy assets (chart libs) inside the island on first render/interaction.

Testing islands

- SSR test: `renderToString/Stream` to ensure HTML includes static shell.
- Client test: simulate hydration strategy (visible/interaction) and assert event handlers work.
- E2E: scroll to islands or click activators and ensure hydration occurs without console errors.

Checklist

- Pick hydration strategy per island (immediate/visible/idle/interaction).
- Pass server data as props; avoid duplicate fetches.
- Lazy-load heavy deps inside islands.
- Test SSR + hydration + interactions.

Try it now: interaction-hydrated menu

```

import { registerIsland, autoHydrateIslands, HydrationStrategy } from '@philjs/ssr';

export function Menu() {
  const open = signal(false);
  return (
    <div>
      <button onClick={() => open.set(!open)}>Menu</button>
      {open() && <ul><li>Profile</li><li>Logout</li></ul>}
    </div>
  );
}

registerIsland('Menu', Menu);
autoHydrateIslands(HydrationStrategy.ON_INTERACTION);

```

The static HTML shows the button; the island hydrates on first click, keeping initial HTML small.

SSR/Islands Playbook

Use this when enabling SSR, streaming, and islands in production.

Before you start

- Decide targets: edge vs regional.
- Ensure entry-server.tsx exists and exports handler per adapter requirements.
- Keep HTML shell small; plan which components become islands.

Streaming checklist

- Use `renderToString` (Node) or `renderToReadableStream` (edge).
- Flush early: send head + shell ASAP; stream slower panels.
- Include critical CSS and meta tags in the first chunk.
- Keep serialized data minimal; avoid large blobs.

Islands checklist

- Identify interactive components; register as islands.
- Choose hydration strategy (immediate/visible/idle/interaction) per island.
- Pass loader data as props; avoid duplicate fetches.
- Lazy-load heavy deps inside islands.

Caching + revalidate

- Tag loader caches by entity; set `revalidate` per route.
- Pair adapter cache (ISR/edge cache) with router invalidation to stay consistent.
- Avoid over-invalidating; prefer targeted tags over wildcards.

Env and secrets

- Use platform env for secrets; never serialize to the client.
- For edge runtimes, stay within Web APIs (no fs, no Node crypto without polyfills).

Error handling

- Wrap root layouts with error boundaries; render friendly fallbacks.
- Log SSR errors with route/params/request id; surface in observability.
- Provide partial fallbacks for streamed sections; do not blank the whole page.

Testing SSR

- Unit: `renderToString` to assert markup.
- Integration: run server locally (`philjs dev --ssr`) and use Playwright to navigate; assert no hydration warnings.
- Slow-path tests: throttle network, delay loaders, verify streaming shows shell quickly.

Deployment steps (generic)

1. `pnpm build` to produce server/client bundles.
2. Run adapter-specific preview (e.g., `vercel dev`, `wrangler pages dev`, `netlify dev`).
3. Inspect bundle sizes and HTML payload.
4. Deploy and smoke with Playwright in CI; collect traces.

Runbook: hydration errors

- Capture server HTML and client render; diff for mismatches.
- Ensure keys are stable; avoid non-deterministic values in render (Dates, random).

- Confirm loader data serializes safely (no functions, no bigint without stringification).

Runbook: slow TTFB

- Profile loader latency; cache where possible.
- Stream earlier; move long work to background or edge cache.
- Trim HTML head and critical CSS; defer non-critical scripts.

Runbook: large HTML

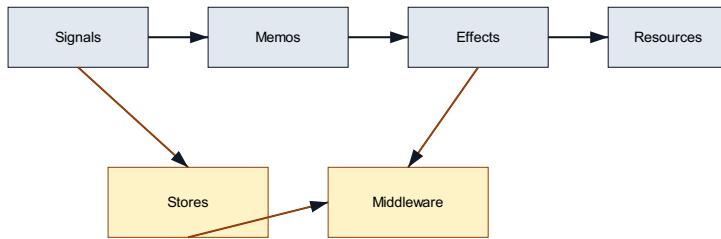
- Remove unused data from serialized state.
- Paginate or stream long lists.
- Inline only critical CSS; lazy-load the rest.

Quick wins

- Stream by default; only use full string rendering for small pages/tests.
- Hydrate heavy widgets on visibility/interaction.
- Keep adapter configs lean; disable Node polyfills in edge builds.

State Management Deep Dive

PhilJS ships signals for fine-grained reactivity and a lightweight store for structured state. Use signals for local UI and derived values; use stores when you need predictable updates, history, or middleware. This chapter extends the brief primer in [docs/core](#) and mirrors store guidance in [docs/patterns](#) so you can copy/paste working patterns.



State graph

Signals vs Store

- **Signals:** best for component-local or tightly scoped shared state; zero boilerplate and no reducers.
- **Store:** immutable-ish updates with middleware, undo/redo, persistence, and derived selectors baked in.

Creating a store

```

import { createStore } from '@philjs/core';

const [state, setState, store] = createStore({
  user: { id: 'u1', name: 'Ava' },
  todos: [{ id: 't1', title: 'Ship PhilJS' }]
});

setState('user', 'name', 'Phil');           // path-based updates
setState('todos', t => [...t, { id: 't2', title: 'Docs' }]);

```

Derived selectors

```
const completedCount = store.select(s => s.todos.filter(t => t.done).length);
```

Selectors memoize by default and only re-run when their inputs change.

For expensive derivations, debounce or throttle inside selectors to avoid UI stutter during rapid updates.

Middleware

Middleware receive the next state and the patch being applied. Use it for logging, analytics, feature flags, or enforcing invariants.

```

store.use((next, patch) => {
  console.info('patch', patch);
  if (patch.path[0] === 'adminOnly') throw new Error('blocked');
  return next();
});

```

Order matters—register middleware in the order they should run.

Common middleware examples:

- **Logger**: send patches to `console.info` or a remote collector.
- **Feature flags**: prevent writes to guarded slices unless a flag is on.
- **Analytics**: emit events when specific paths change (e.g., `checkout.status`).
- **Schema guard**: validate patches with Zod/Valibot before applying.

History (undo/redo)

Enable history to get time travel during development or for user-facing undo:

```
const [state, setState, store] = createStore(initial, { history: { limit: 100 } });
store.undo();
store.redo();
```

Recommendations:

- Keep history limits sane (25–100).
- Clear redo when applying new patches after an undo (handled automatically).
- Avoid storing large blobs directly; store handles ref equality efficiently but payload size still matters.
- For collaborative scenarios, pair history with intent logs instead of storing every patch verbatim.

Persistence

Persist slices of state to storage (localStorage, IndexedDB, custom drivers):

```
store.persist({
  driver: 'localStorage',
  paths: ['auth', 'preferences']
});
```

Use encryption for sensitive data; avoid persisting secrets entirely when possible.

Custom drivers

Implement a driver with `get`, `set`, and `remove` to target IndexedDB, FileSystem, or secure storage on mobile (Capacitor):

```
const idbDriver = {
  async get(key) { /* ... */ },
  async set(key, value) { /* ... */ },
  async remove(key) { /* ... */ }
};

store.persist({ driver: idbDriver, paths: ['drafts'] });
```

Patterns

- **Slice-first**: derive child stores (`store.slice('todos')`) for complex apps to prevent prop-drilling.
- **Selectors > effects**: prefer selectors to recompute view data instead of side-effect-driven mutations.
- **Cross-tab sync**: combine persistence with storage events or `BroadcastChannel` for multi-tab coherence.
- **Server state**: do not overuse store for server data; pair with loaders/resources and keep cache lifetimes explicit.
- **Undo-friendly UI**: expose undo/redo in UI where users edit large documents.
- **Time travel debugging**: wire store history to the PhilJS DevTools panel for replay.
- **Symbols and Maps**: prefer plain objects/arrays for best reactivity; if you must use Symbols/Maps, expose derived POJOs for rendering.
- **Multi-tenant**: key store slices by tenant/user to avoid data bleed when switching accounts.
- **SSR-aware**: initialize stores from loader data; avoid leaking server-only secrets into serialized state.
- **SharedWorker/ServiceWorker**: for cross-tab state, share updates via a worker and hydrate per tab through a store slice.

Testing stores

- Test reducers/updaters as pure functions.
- Assert middleware order and failure modes.
- Use fake storage for persistence tests.
- Snapshot derived selectors to lock in business logic.
- Run long-lived mutation tests to ensure history limits and persistence do not leak memory.

Try it now: guarded store with history and persistence

```

import { createStore } from '@philjs/core';
import { z } from 'zod';

const [state, setState, store] = createStore({ todos: [] }, { history: { limit: 50 }});

// Schema guard middleware
const todoSchema = z.object({ id: z.string(), title: z.string(), done: z.boolean().optional() });
store.use((next, patch) => {
  if (patch.path[0] === 'todos' && patch.value) {
    const arr = Array.isArray(patch.value) ? patch.value : [patch.value];
    arr.forEach(todoSchema.parse);
  }
  return next();
});

// Persistence
store.persist({ driver: 'localStorage', paths: ['todos'] });

// Updates
setState('todos', t => [...t, { id: crypto.randomUUID(), title: 'Ship PhilJS', done: false }]);

```

Inspect the history in DevTools, undo/redo a few steps, and refresh to confirm persistence.

Store Middleware Cookbook

Patterns for PhilJS store middleware (logging, flags, validation, analytics).

Logging middleware

```

store.use((next, patch) => {
  console.info('patch', patch);
  return next();
});

```

Feature flag guard

```

store.use((next, patch) => {
  if (patch.path[0] === 'beta' && !hasFlag('beta')) {
    throw new Error('Feature disabled');
  }
  return next();
});

```

Schema validation (Zod)

```

const todoSchema = z.object({ id: z.string(), title: z.string() });
store.use((next, patch) => {
  if (patch.path[0] === 'todos' && patch.value) {
    const arr = Array.isArray(patch.value) ? patch.value : [patch.value];
    arr.forEach(todoSchema.parse);
  }
  return next();
});

```

Analytics hook

```

store.use((next, patch) => {
  const result = next();
  sendAnalytics({ patch });
  return result;
});

```

Prevent large payloads

```

store.use((next, patch) => {
  const bytes = new TextEncoder().encode(JSON.stringify(patch.value ?? '')).length;
  if (bytes > 50_000) throw new Error('Patch too large');
  return next();
});

```

Checklist

- Middleware ordered correctly (guards first, analytics last).
- Validate inputs before writes.
- Avoid blocking operations inside middleware.
- Keep middleware pure; no hidden side-effects besides intended logging/metrics.

Advanced State Patterns

Go beyond basics with history, collaboration, cross-tab sync, and performance tuning.

History and time travel

- Enable history with limits; expose undo/redo in UI.
- Clear redo on new mutations; cap history to prevent memory growth.
- Snapshot critical slices for debugging; wire to DevTools for replay.

Collaboration

- Use `@philjs/collab` or WebSockets to sync patches.
- Prefer intent logs or CRDT-friendly data for multi-user edits.
- Resolve conflicts explicitly; show presence and cursors for transparency.

Cross-tab and multi-window

- Sync via BroadcastChannel/SharedWorker; debounce to avoid storms.
- Namespaces per tenant/user to avoid data bleed.
- Persist to IndexedDB and rehydrate on tab focus.

Performance tuning

- Split stores by domain; avoid one mega-store.
- Memoize selectors; avoid subscribing components to the whole store.
- Batch updates; avoid object identity churn in hot paths.

Security and isolation

- Do not store secrets in stores/persistence.
- Sanitize data before writing to state (strip `__proto__`/constructor).
- For multi-tenant, clear stores on account switch.

Testing

- Long-running mutation tests to ensure history/persistence behave.
- Fuzz selectors/middleware with random patches to catch edge cases.
- Simulate multi-tab updates with fake BroadcastChannel.

Checklist

- History enabled where needed; limits set.
- Collaboration/conflict strategy defined.
- Cross-tab sync namespaced and debounced.
- Selectors memoized; stores split by domain.
- Sensitive data excluded or encrypted at rest.

Styling Options

PhilJS works with vanilla CSS, CSS modules, and first-party styling utilities.

CSS modules

```
import styles from "./Button.module.css";

export function Button({ children }: { children: string }) {
  return <button class={styles.button}>{children}</button>;
}
```

Scoped styles

```

import { css } from "@philjs/styles/scoped";

export function Notice({ message }: { message: string }) {
  const styles = css`
    .notice {
      background: #0f172a;
      color: #e2e8f0;
      padding: 1rem;
      border-radius: 12px;
    }
  `;

  return (
    <div class="notice">
      <style>{styles}</style>
      {message}
    </div>
  );
}

```

CSS-in-JS

```

import { styled } from "@philjs/styles/css-in-js";

const Button = styled("button", {
  base: { padding: "0.5rem 1rem", borderRadius: "999px" },
  variants: {
    tone: {
      primary: { background: "#2563eb", color: "#fff" },
      neutral: { background: "#e2e8f0", color: "#0f172a" },
    },
  },
});

```

Design tokens and theming

- Define tokens once (colors, spacing, typography) and expose via CSS variables.
- Support light/dark by swapping token sets at the root.

```

:root {
  --color-bg: #0b1021;
  --color-fg: #e2e8f0;
  --space-1: 0.25rem;
  --space-2: 0.5rem;
}

[data-theme="light"] {
  --color-bg: #f8fafc;
  --color-fg: #0f172a;
}

```

Apply in components:

```

export function Panel({ children }) {
  return <section style={{ background: "var(--color-bg)", color: "var(--color-fg)", padding: "var(--space-3)" }}>{children}</section>;
}

```

Tailwind adapter

Use `philjs-plugin-tailwind` to integrate Tailwind with zero extra glue:

```

// vite.config.ts
import tailwind from 'philjs-plugin-tailwind';

export default defineConfig({
  plugins: [philjs(), tailwind()],
});

```

Keep Tailwind for utility density; layer tokens on top for theming consistency.

Tokens reference

See [Design Tokens](#) for a shared vocabulary across components and themes.

Styling performance

- Prefer static class names where possible; avoid recomputing style objects on every render.
- Scope CSS for islands to avoid global leaks.
- Defer non-critical CSS with `media/prefetch` or use critical CSS in SSR stream for above-the-fold.

Accessibility

- Respect prefers-reduced-motion; disable heavy animations for that preference.
- Ensure color contrast meets WCAG; bake contrast checks into design tokens.
- Keep focus outlines; use :focus-visible for sensible defaults.

Testing styles

- Snapshot minimal pieces (class presence) but assert behavior via roles/visibility.
- Use Playwright visual diffs sparingly for critical components.

Checklist

- Tokens defined and applied via CSS variables.
- Light/dark theme switching without layout shift.
- Tailwind (if used) configured via philjs-plugin-tailwind.
- Critical CSS inlined for above-the-fold; rest deferred.
- prefers-reduced-motion respected.

Try it now: theme toggle

```
import { signal } from '@philjs/core';

const theme = signal<'light' | 'dark'>('dark');
function toggle() { theme.update(t => (t === 'dark' ? 'light' : 'dark')) }

export function ThemeToggle() {
  return (
    <button onClick={toggle}>
      Switch to {theme() === 'dark' ? 'light' : 'dark'}
    </button>
  );
}
```

Set `data-theme={theme()}` on your root element and watch tokens swap without repainting layout.

Design Systems with PhilJS

Codify components, tokens, and accessibility into a repeatable system.

Tokens

- Define color/spacing/typography tokens via CSS variables.
- Support light/dark themes and high-contrast variants.
- Version tokens; avoid breaking changes without migration notes.
- Keep a single source of truth (`tokens.ts` or `tokens.css`) and generate platform outputs if needed.

Components

- Build headless primitives (Button, Input, Dialog) with PhilJS signals.
- Layer styling via CSS modules, scoped styles, or Tailwind plugin.
- Provide accessibility baked in: roles, labels, keyboard interactions.
- Export prop types and usage patterns; document ARIA expectations per component.

Theming

- Switch tokens via `data-theme` on root; persist user choice.
- Respect `prefers-color-scheme`; provide explicit override.
- Ensure animations respect `prefers-reduced-motion`.

Documentation

- Document props, a11y notes, and states (hover/focus/disabled).
- Include code samples and a live playground.
- Add usage guidelines and anti-patterns.
- Show theme variants and density/size variants with guidance.

Distribution

- Package as `@your-org/ui` with `workspace:*` dependencies.
- Export ESM; tree-shakeable entrypoints.
- Ship type definitions and keep `peerDeps` minimal.
- Provide per-component entrypoints for optimal tree-shaking.

Testing

- Unit: behavior and aria roles.
- Visual: limited snapshots or visual diffs for core components.
- A11y: role/label checks; optional axe on key pages.

Performance

- Keep components lean; avoid heavy runtime logic.
- Defer icons/assets; prefer sprite sheets or icon fonts where acceptable.
- Split rare variants into lazy-loaded chunks if heavy.

Checklist

- Tokens defined and themed.
- Headless + styled layers documented.
- A11y baked in; keyboard + screen reader support.
- Packages exported as ESM with types.
- Tests for behavior and critical visuals.

Motion and Transitions

Use motion intentionally to guide attention without hurting performance or accessibility.

Principles

- Motion must serve meaning (state change, hierarchy).
- Prefer `transform` and `opacity`; avoid layout-triggering properties.
- Respect `prefers-reduced-motion`.

Micro-interactions

- Buttons/links: subtle scale-opacity on hover/focus.
- Toggles: smooth state transitions with easing.
- Lists/cards: staggered reveal only when beneficial.

Page and route transitions

- Use view transitions where supported to reduce jank during navigation.
- Keep transitions short (<250ms) and cancellable.
- Avoid blocking data fetching; run in parallel with loader prefetch.

Performance

- GPU-friendly properties (`transform`, `opacity`).
- Avoid large box-shadows/blurs on low-end devices.
- Test on mid-tier mobile; throttle CPU to catch jank.

Accessibility

- Honor `prefers-reduced-motion: reduce`; switch to instant transitions.
- Provide clear focus states; do not hide outlines.
- Avoid auto-playing motion; require user intent.

Implementation sketch

```
const prefersReduced = window.matchMedia('(prefers-reduced-motion: reduce)').matches;
const duration = prefersReduced ? 0 : 180;
```

Use CSS variables for timing/easing:

```
:root {
  --ease: cubic-bezier(0.22, 1, 0.36, 1);
  --dur-fast: 120ms;
  --dur-medium: 200ms;
}
```

Testing motion

- Disable animations in test env to avoid flakes.
- In Playwright, check that transitions complete quickly and don't block input.
- Ensure reduced-motion mode removes non-essential animations.

Checklist

- Motion uses transform-opacity.
- Reduced-motion respected.
- Route transitions short and cancel-friendly.
- Animations do not block interaction or fetching.

Design Tokens Reference

Centralize your system's primitives to keep UI consistent and themeable.

Token categories

- Color (bg/fg/surface/brand/semantic)
- Typography (families, sizes, weights, line-heights)
- Spacing (scale for margin/padding/gaps)
- Radius, shadow, border
- Motion (durations, easings)
- Z-index layers

Example (CSS variables)

```
:root {
  --color-bg: #0b1021;
  --color-fg: #e2e8f0;
  --color-brand: #2563eb;
  --radius-sm: 6px;
  --radius-md: 12px;
  --shadow-sm: 0 4px 12px rgba(0,0,0,0.08);
  --space-1: 0.25rem;
  --space-2: 0.5rem;
  --dur-fast: 120ms;
  --ease-standard: cubic-bezier(0.22, 1, 0.36, 1);
}
```

Theming

- Light/dark variants override token sets via [data-theme="light"] etc.
- Support high-contrast variants.
- Keep tokens semantically named; avoid raw hex values in components.

Distribution

- Ship tokens as a package or importable CSS/TS module.
- Provide TypeScript types for token names to reduce typos.
- Keep changes versioned; communicate breaking token updates.

Testing and QA

- Snap visual baselines for key components per theme.
- Check contrast programmatically for semantic colors.
- Verify motion tokens respect prefers-reduced-motion.

Checklist

- Tokens defined for all categories (color, type, space, motion, radius/shadow).
- Semantic names used in components.
- Theming supported with overrides.
- Contrast and a11y validated.

Testing

PhilJS includes a first-party testing library that mirrors DOM Testing Library with PhilJS-aware utilities.

```
import { describe, it, expect } from "vitest";
import { render, screen, fireEvent } from '@philjs/testing';
import { Counter } from "../src/Counter";

describe("Counter", () => {
  it("increments", () => {
    render(() => <Counter />);
    fireEvent.click(screen.getByRole("button", { name: `/increment/${i}` }));
    expect(screen.getText(/count: 1/i)).toBeInTheDocument();
  });
});
```

Run tests

```
pnpm test
pnpm --filter @philjs/core test:coverage
```

Test layers

- **Unit**: signals, stores, helpers. Use Vitest + PhilJS testing library.
- **Component**: render components, interact via roles/labels, assert DOM.
- **Integration**: render routes with loaders/actions mocked (MSW), assert navigation and cache effects.
- **E2E**: Playwright against dev/preview builds; cover core user journeys.
- **Performance**: use vitest bench and Playwright traces for route timings.

Mocking and data

- Use MSW to stub network calls; mirror API schemas to catch drift.
- For stores, set initial state and assert selectors and middleware behavior.
- Avoid shallow rendering—test real DOM output; prefer role-based queries.

Coverage and budgets

- Run pnpm --filter @philjs/core test:coverage for critical packages.
- Track snapshot size sparingly; prefer explicit assertions over large snapshots.
- Include perf budgets in CI: pnpm bench and pnpm size.
- Gate merges on tests + lint + typecheck; surface flaky tests early.
- Add property-based/fuzz tests for critical invariants; see [Property-Based Testing](#).

Playwright smoke template

```
import { test, expect } from '@playwright/test';

test('home renders and navigates', async ({ page }) => {
  await page.goto('http://localhost:3000/');
  await expect(page.getByRole('heading', { name: /home/i })).toBeVisible();
  await page.getByRole('link', { name: /about/i }).click();
  await expect(page).toHaveURL(/about/);
});
```

Testing loaders/actions

- Unit-test loaders with fake request, params, and signal.
- Integration-test forms via <form> in jsdom; assert action results and invalidations.
- In E2E, submit real forms and assert SSR + hydration consistency (no flicker, no double submit).

CI pipeline template

```
pnpm install --frozen-lockfile
pnpm lint
pnpm typecheck
pnpm test -- --runInBand
pnpm size
pnpm bench --filter @philjs/core
```

Cache pnpm store between runs; record Playwright traces for failing tests.

Accessibility checks

- Use getByRole/getByLabelText to encode a11y into tests.
- Add automated a11y checks (axe) in CI for key pages.
- Include keyboard navigation tests in Playwright (tab/enter/escape flows).
- Verify reduced-motion handling in tests where motion is involved.

Fixtures and data strategy

- Keep fixtures small and realistic; prefer factory helpers to enormous JSON.
- Reset global state between tests; use beforeEach to recreate signals/stores.
- For date/time logic, use fake timers to make assertions stable.

Checklist

- Unit tests for core logic and signals.
- Component tests cover interactions (clicks, input, keyboard).
- Integration tests for loaders/actions with MSW.
- Playwright smoke tests for critical routes.

- Perf/bench runs in CI for regressions.
- A11y assertions via roles/labels.

Try it now: loader integration test

```
import { describe, it, expect } from 'vitest';
import { render, screen, waitFor } from '@philjs/testing';
import { ProjectsRoute } from '../src/routes/projects';
import { server, rest } from './test-server'; // MSW setup

describe('ProjectsRoute', () => {
  it('renders projects from loader', async () => {
    server.use(rest.get('/api/projects', (_req, res, ctx) =>
      res(ctx.json([{ id: '1', name: 'Alpha' }])))
  });
  render(() => <ProjectsRoute data={[{ id: '1', name: 'Alpha' }]} />);
  await waitFor(() => screen.getByText('Alpha'));
});
});
```

Wire MSW once in setup tests; reuse across suites for realistic behavior.

Testing Playbook (PhilJS)

Use this checklist-driven guide to keep quality high as the codebase grows.

Layers and tools

- **Unit**: signals, stores, helpers (Vitest).
- **Component**: DOM-level assertions with `@philjs/testing` (built on Testing Library).
- **Integration**: routes with loaders/actions + MSW for network.
- **E2E**: Playwright for real browser flows (SSR + hydration).
- **Perf**: vitest bench, Playwright traces.
- **A11y**: role/label assertions and axe where useful.

Daily habits

- Write tests alongside features; cover happy + failure paths.
- Use realistic fixtures; avoid “lorem ipsum” data that hides issues.
- Keep tests deterministic: fake timers, stable IDs, fixed dates.
- Prefer user-centric assertions (roles, text) over implementation details.

Component testing recipe

```
import { render, screen, fireEvent } from '@philjs/testing';
import { Dialog } from './Dialog';

it('opens and closes', () => {
  render(() => <Dialog />);
  fireEvent.click(screen.getByRole('button', { name: '/open/i '}));
  expect(screen.getByRole('dialog')).toBeVisible();
  fireEvent.click(screen.getByRole('button', { name: '/close/i '}));
  expect(screen.queryByRole('dialog')).not.toBeInTheDocument();
});
```

Loaders/actions integration recipe

- Mock network with MSW; keep handlers aligned with API schemas.
- Assert cache invalidation and optimistic updates.
- Cover redirects and error boundaries.

E2E smoke suite (minimum)

- Home renders, navigation works, no console errors.
- Auth flow (login/logout) succeeds and persists session.
- Key CRUD journey works with optimistic UI.
- Slow network simulation still shows fallbacks and recovers.

CI pipeline

```
pnpm lint
pnpm typecheck
pnpm test -- --runInBand
pnpm size
pnpm --filter @philjs/core vitest bench
pnpm exec playwright test --reporter=line
```

Cache pnpm store; upload Playwright traces on failure.

Snapshots policy

- Avoid giant snapshots; use them for small, stable UI fragments only.
- Prefer explicit assertions for text/aria/structure.

Flake reduction

- Clean up timers and intervals in tests.
- Avoid relying on animation frames; fake them or disable animations in tests.
- Give async expectations clear timeouts.

A11y

- Query by role/label/placeholder.
- Ensure focus management is tested for dialogs/menus.
- Run axe on critical pages in Playwright or as a unit helper sparingly.

Performance and budgets

- Gate PRs on size-limit.
- Benchmark hot paths periodically; track regressions.
- Record a Playwright trace for each major feature before release.

When bugs happen

1. Reproduce with a minimal test (unit or component).
2. Add assertions that would have caught it.
3. Fix the code; keep the test.
4. If flakiness was involved, add a guardrail (fake timers, MSW handler, deterministic data).

Coverage focus

- Loader/action logic and cache invalidation.
- Routing with params, redirects, and errors.
- Stores with middleware, history, and persistence.
- Complex widgets (forms, editors, charts) with real interactions.

End-to-End (E2E) Testing with Playwright

Test real user journeys across SSR, hydration, and client-side interactions.

Setup

- Install Playwright via `pnpm exec playwright install`.
- Add scripts: "test:e2e": "playwright test" and CI reporter config.
- Run against `philjs dev --ssr` or a preview build for realistic behavior.

Core smoke suite

- Home renders; no console errors.
- Navigation works (links, back/forward); scroll restoration.
- Auth flow (login/logout) and session persistence.
- Key CRUD journey with optimistic UI.
- Slow network fallback: throttled tests to ensure skeletons render.

Writing tests

```
import { test, expect } from '@playwright/test';

test('dashboard loads and prefetches settings', async ({ page }) => {
  await page.goto('http://localhost:3000/dashboard');
  await expect(page.getByRole('heading', { name: /dashboard/i })).toBeVisible();
  await page.getByRole('link', { name: /settings/i }).hover();
  // optional: assert prefetch via network or cache state if exposed
});
```

Fixtures and data

- Use test accounts/seed data; isolate per test.
- For destructive actions, reset DB or use ephemeral env.
- Mock third-party services (email, payments) with test doubles.

Traces and debugging

- Enable traces on failure (--trace on-first-retry); upload artifacts in CI.
- Capture console logs and network requests for flaky tests.
- Use page.route sparingly; prefer full-stack paths to catch integration issues.

Performance checks

- Record traces for hot routes; assert no long tasks and reasonable TTFB.
- Optional: measure LCP via custom script if needed.

Accessibility checks

- Keyboard navigation: tab through dialogs/menus; ensure focus trap works.
- Basic aria/role checks on key pages; optionally run axe on a subset.

CI tips

- Run headless; cache browsers if CI allows.
- Parallelize suites but serialize when tests share stateful backends.
- Set generous timeouts for SSR/edge preview starts; fail fast if server isn't ready.

Checklist

- Smoke covers home/nav/auth/CRUD.
- Traces on failure enabled and archived.
- Fixtures isolated; DB reset per suite if needed.
- Accessibility keyboard flows covered.
- Perf traces captured for critical routes before release.

Property-Based and Fuzz Testing

Use properties to uncover edge cases that example-based tests miss.

When to use

- Core algorithms (parsers, reducers, diffing/patching).
- Store operations and undo/redo logic.
- Cache invalidation rules and selectors.

Tooling

- fast-check (JS property testing).
- Custom fuzzers for domain-specific inputs.

Defining properties

- Idempotence: applying the same patch twice yields same result.
- Reversibility: undo after redo returns to initial state.
- Invariants: e.g., cache tags remain unique, totals stay non-negative.

Example

```
import fc from 'fast-check';
import { createStore } from '@philjs/core';

fc.assert(
  fc.property(fc.array(fc.record({ id: fc.string(), done: fc.boolean() })), (todos) => {
    const [, set, store] = createStore({ todos: [] });
    set('todos', todos);
    return store.select(s => s.todos.length) === todos.length;
  })
);
```

Fuzzing loaders/actions

- Generate random params/query inputs; assert schemas reject bad data.
- Fuzz mutation payloads; ensure server-side validation guards hold.

Testing guidance

- Limit runs in CI to a sane number; increase locally for deeper checks.
- Shrink failing cases to minimal repro; keep them as fixtures/regression tests.
- Combine with deterministic seed for reproducibility.

Checklist

- Properties defined for core invariants.
- Seeds recorded for failures.
- Fuzz inputs for loaders/actions and stores.
- Regression tests added from shrunk counterexamples.

Accessibility Testing

Bake accessibility into automated tests to avoid regressions.

Unit/component level

- Use role/label/placeholder queries (`getByRole`, `getByLabelText`).
- Assert focus management for dialogs/menus (focus trap, restore on close).
- Check `aria-invalid`, `aria-describedby` for form errors.

Integration

- With MSW, simulate error states and ensure `role="alert"` messages appear.
- Verify keyboard navigation (Tab/Shift+Tab/Enter/Escape) works on critical components.
- Ensure reduced-motion handling is respected in components with animations.

E2E (Playwright)

- Tab through dialogs/menus; assert focus order.
- Check that skip links, landmarks (`main`, `nav`, `footer`) are present.
- Optional: run axe on a subset of critical pages; keep failures actionable.

Contrast and visuals

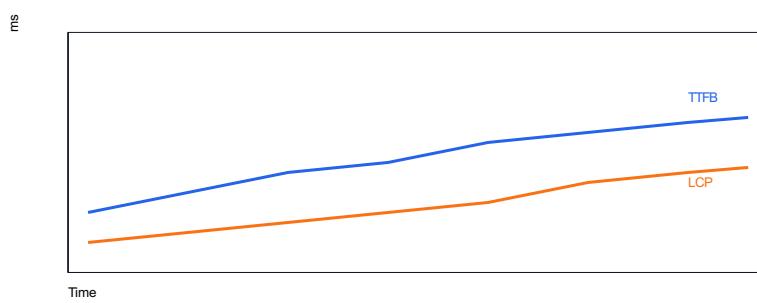
- Token-level contrast checks (color pairs) in unit tests if possible.
- Avoid text in images; if used, ensure alt text/captions.

Checklist

- Role/label queries cover core UI.
- Focus management tested for overlays.
- Keyboard flows tested in E2E.
- Reduced-motion respected where applicable.
- Optional axe checks on critical pages.

Performance Overview

Optimize PhilJS applications for maximum speed and efficiency.



What You'll Learn

- Performance fundamentals
- Optimization strategies
- Measurement techniques
- Common bottlenecks
- Best practices
- Performance checklist

Performance Fundamentals

Key Metrics

```
// Core Web Vitals
interface WebVitals {
  LCP: number; // Largest Contentful Paint (< 2.5s)
  FID: number; // First Input Delay (< 100ms)
  CLS: number; // Cumulative Layout Shift (< 0.1)
  FCP: number; // First Contentful Paint (< 1.8s)
  TTFB: number; // Time to First Byte (< 600ms)
}
```

Performance Budget

```
const performanceBudget = {
  // Bundle sizes
  mainBundle: 170, // KB (gzipped)
  vendorBundle: 100, // KB (gzipped)
  cssBundle: 30, // KB (gzipped)

  // Timing budgets
  timeToInteractive: 3000, // ms
  firstContentfulPaint: 1800, // ms
  largestContentfulPaint: 2500, // ms

  // Resource counts
  totalRequests: 50,
  imageRequests: 20,
  scriptRequests: 10
};
```

Optimization Strategies

1. Code Splitting

Split code into smaller chunks for faster initial load:

```
import { lazy } from '@philjs/core';

// Lazy Load heavy components
const Dashboard = lazy(() => import('./Dashboard'));
const Reports = lazy(() => import('./Reports'));

function App() {
  return (
    <Router>
      <Route path="/dashboard" component={Dashboard} />
      <Route path="/reports" component={Reports} />
    </Router>
  );
}
```

2. Memoization

Cache expensive computations:

```
import { signal, memo } from '@philjs/core';

function ExpensiveComponent() {
  const data = signal([/* Large dataset */]);

  // Memoize expensive calculation
  const processedData = memo(() => {
    return data().map(item => {
      // Heavy processing
      return expensiveTransform(item);
    });
  });

  return <div>{processedData().length} items</div>;
}
```

3. Virtual Scrolling

Render only visible items in large lists:

```

import { VirtualScroller } from '@philjs/core';

function LargeList({ items }: { items: any[] }) {
  return (
    <VirtualScroller
      items={items}
      itemHeight={50}
      height={600}
      renderItem={(item) => <ListItem data={item} />}
    />
  );
}

```

4. Bundle Optimization

Reduce bundle size:

```

// vite.config.ts
export default {
  build: {
    rollupOptions: {
      output: {
        manualChunks: {
          vendor: ['@philjs/core'],
          utils: ['date-fns', 'lodash-es']
        }
      }
    }
};

```

Common Bottlenecks

1. Unnecessary Re-renders

```

// ⚠️ Re-renders on every parent update
function Child({ onClick }: { onClick: () => void }) {
  return <button onClick={onClick}>Click</button>;
}

// ⚠️ Memoized to prevent re-renders
const Child = memo(({ onClick }: { onClick: () => void }) => {
  return <button onClick={onClick}>Click</button>;
});

```

2. Large Bundle Size

```

# Analyze bundle
npm run build -- --analyze

# Look for:
# - Duplicate dependencies
# - Unused code
# - Large Libraries that can be replaced

```

3. Inefficient Updates

```

// ⚠️ Updates entire array
const updateItem = (id: string) => {
  items.set(items().map(item =>
    item.id === id ? { ...item, updated: true } : item
  ));
};

// ⚠️ Update only specific item
const updateItem = (id: string) => {
  const index = items().findIndex(i => i.id === id);
  const updated = [...items()];
  updated[index] = { ...updated[index], updated: true };
  items.set(updated);
};

```

Measurement Tools

Performance Observer

```

import { effect } from '@philjs/core';

function measurePerformance() {
  effect(() => {
    // Measure First Contentful Paint
    const observer = new PerformanceObserver((list) => {
      for (const entry of list.getEntries()) {
        if (entry.name === 'first-contentful-paint') {
          console.log('FCP:', entry.startTime);
        }
      }
    });
    observer.observe({ entryTypes: ['paint'] });

    return () => observer.disconnect();
  });
}

```

Custom Timing

```

function measureComponentRender(componentName: string) {
  const start = performance.now();

  return () => {
    const end = performance.now();
    const duration = end - start;

    console.log(`${componentName} rendered in ${duration}ms`);

    // Send to analytics
    sendMetric('component-render', {
      component: componentName,
      duration
    });
  };
}

// Usage
function HeavyComponent() {
  const measureEnd = measureComponentRender('HeavyComponent');

  effect(() => {
    measureEnd();
  });

  return <div>{/* ... */}</div>;
}

```

Best Practices

Load JavaScript Efficiently

```

<!-- ⚡ Defer non-critical scripts -->
<script defer src="/app.js"></script>

<!-- ⚡ Async for independent scripts -->
<script async src="/analytics.js"></script>

<!-- ⚡ Blocking script -->
<script src="/app.js"></script>

```

Optimize Images

```
// ⚡ Lazy Load images


// 🌐 Use modern formats



<picture>
  <source srcset="/image.webp" type="image/webp" />
  <source srcset="/image.jpg" type="image/jpeg" />
  
</picture>



// 🌐 Responsive images






```

Minimize Main Thread Work

```
// 🌐 Use Web Workers for heavy computation
const worker = new Worker('/worker.ts');

worker.postMessage({ data: largeDataset });

worker.onmessage = (e) => {
  const result = e.data;
  updateUI(result);
};

// worker.ts
self.onmessage = (e) => {
  const processed = heavyComputation(e.data);
  self.postMessage(processed);
};
```

Reduce Layout Shifts

```
// 🌐 Reserve space for dynamic content
<div style={{
  minHeight: '200px', // Reserve space
  display: 'flex',
  alignItems: 'center',
  justifyContent: 'center'
}}>
  {loading() ? <Spinner /> : <Content />}
</div>

// 🌐 Set image dimensions

```

Performance Checklist

Initial Load

- Bundle size < 170KB (gzipped)
- First Contentful Paint < 1.8s
- Time to Interactive < 3.5s
- Minimize render-blocking resources
- Enable compression (gzip/brotli)
- Use CDN for static assets

Runtime Performance

- Avoid unnecessary re-renders
- Memoize expensive computations
- Use virtual scrolling for long lists

- Debounce/throttle event handlers
- Clean up effects and listeners
- Optimize images (lazy load, modern formats)

Network

- Minimize HTTP requests
- Use HTTP/2 or HTTP/3
- Implement caching strategy
- Preload critical resources
- Prefetch likely navigation targets

Code Quality

- Remove unused code
- Tree-shake dependencies
- Use production builds
- Minimize polyfills
- Avoid large dependencies

Monitoring

Real User Monitoring

```

import { signal, effect } from '@philjs/core';

const metrics = signal<WebVitals>({
  LCP: 0,
  FID: 0,
  CLS: 0,
  FCP: 0,
  TTFB: 0
});

// Report to analytics
function reportMetric(metric: keyof WebVitals, value: number) {
  metrics.set({ ...metrics, [metric]: value });
}

// Send to analytics service
fetch('/api/metrics', {
  method: 'POST',
  body: JSON.stringify({ metric, value, timestamp: Date.now() })
});

// Measure Web Vitals
effect(() => {
  // Largest Contentful Paint
  new PerformanceObserver((list) => {
    const entries = list.getEntries();
    const lastEntry = entries[entries.length - 1];
    reportMetric('LCP', lastEntry.startTime);
  }).observe({ entryTypes: ['largest-contentful-paint'] });

  // First Input Delay
  new PerformanceObserver((list) => {
    const entry = list.getEntries()[0] as PerformanceEventTiming;
    reportMetric('FID', entry.processingStart - entry.startTime);
  }).observe({ entryTypes: ['first-input'] });

  // Cumulative Layout Shift
  let clsValue = 0;
  new PerformanceObserver((list) => {
    for (const entry of list.getEntries()) {
      if (!(entry as any).hadRecentInput) {
        clsValue += (entry as any).value;
        reportMetric('CLS', clsValue);
      }
    }
  }).observe({ entryTypes: ['layout-shift'] });
});

```

Summary

You've learned:

- Performance fundamentals and metrics
- Core optimization strategies
- Common performance bottlenecks
- Measurement and profiling tools
- Best practices for optimization
- Performance monitoring

Master performance optimization for fast PhilJS apps!

Next: [Code Splitting](#) → Split code for faster loads

Performance Budgets

Set and enforce performance goals to maintain fast application performance.

What You'll Learn

- Setting performance budgets
- Budget categories
- Monitoring tools
- CI/CD integration
- Enforcement strategies
- Best practices

What Are Performance Budgets?

Performance budgets are limits you set for metrics that affect your application's performance. They help prevent performance regressions by making performance measurable and enforceable.

Common Budget Categories

```
interface PerformanceBudget {
  // Timing budgets
  timing: {
    FCP: number; // First Contentful Paint
    LCP: number; // Largest Contentful Paint
    TTI: number; // Time to Interactive
    FID: number; // First Input Delay
    CLS: number; // Cumulative Layout Shift
  };

  // Size budgets
  size: {
    totalJS: number; // Total JavaScript size
    totalCSS: number; // Total CSS size
    mainBundle: number; // Main bundle size
    vendorBundle: number; // Vendor bundle size
    images: number; // Total image size
  };

  // Resource budgets
  resources: {
    requests: number; // Total HTTP requests
    fonts: number; // Number of font files
    scripts: number; // Number of script files
  };
}

// Example budget
const budget: PerformanceBudget = {
  timing: {
    FCP: 1800, // 1.8s
    LCP: 2500, // 2.5s
    TTI: 3800, // 3.8s
    FID: 100, // 100ms
    CLS: 0.1 // 0.1
  },
  size: {
    totalJS: 300, // 300KB (gzipped)
    totalCSS: 50, // 50KB (gzipped)
    mainBundle: 170, // 170KB (gzipped)
    vendorBundle: 130, // 130KB (gzipped)
    images: 500 // 500KB
  },
  resources: {
    requests: 50,
    fonts: 2,
    scripts: 10
  }
};
```

Setting Budgets

Analyze Current Performance

```

import { signal } from '@philjs/core';

interface PerformanceMetrics {
  FCP: number;
  LCP: number;
  TTI: number;
  bundleSize: number;
  requestCount: number;
}

async function analyzeCurrentPerformance(): Promise<PerformanceMetrics> {
  // Get timing metrics
  const navigation = performance.getEntriesByType('navigation')[0] as PerformanceNavigationTiming;

  // Get paint metrics
  const paintEntries = performance.getEntriesByType('paint');
  const fcp = paintEntries.find(e => e.name === 'first-contentful-paint')?.startTime || 0;

  // Get LCP
  let lcp = 0;
  new PerformanceObserver((list) => {
    const entries = list.getEntries();
    lcp = entries[entries.length - 1].startTime;
  }).observe({ entryTypes: ['largest-contentful-paint'] });

  // Get resource sizes
  const resources = performance.getEntriesByType('resource') as PerformanceResourceTiming[];
  const bundleSize = resources
    .filter(r => r.name.endsWith('.js'))
    .reduce((sum, r) => sum + (r.transferSize || 0), 0);

  return {
    FCP: fcp,
    LCP: lcp,
    TTI: navigation.domInteractive,
    bundleSize: bundleSize / 1024, // Convert to KB
    requestCount: resources.length
  };
}

// Usage
const currentMetrics = await analyzeCurrentPerformance();
console.log('Current performance:', currentMetrics);

// Set budgets based on current performance (10-20% improvement)
const improvedBudget = {
  FCP: currentMetrics.FCP * 0.9,
  LCP: currentMetrics.LCP * 0.9,
  TTI: currentMetrics.TTI * 0.85
};

```

Competitive Analysis

```

interface CompetitorMetrics {
  name: string;
  FCP: number;
  LCP: number;
  bundleSize: number;
}

const competitors: CompetitorMetrics[] = [
  { name: 'Competitor A', FCP: 1200, LCP: 2100, bundleSize: 250 },
  { name: 'Competitor B', FCP: 1500, LCP: 2400, bundleSize: 300 },
  { name: 'Competitor C', FCP: 1800, LCP: 2800, bundleSize: 350 }
];

// Set budget to be faster than average competitor
const avgFCP = competitors.reduce((sum, c) => sum + c.FCP, 0) / competitors.length;
const avgLCP = competitors.reduce((sum, c) => sum + c.LCP, 0) / competitors.length;
const avgBundle = competitors.reduce((sum, c) => sum + c.bundleSize, 0) / competitors.length;

const competitiveBudget = {
  FCP: avgFCP * 0.8, // 20% faster than average
  LCP: avgLCP * 0.8,
  bundleSize: avgBundle * 0.9 // 10% smaller
};

```

Monitoring Budgets

Build-Time Budget Checking

```

// vite.config.ts
import { defineConfig } from 'vite';

export default defineConfig({
  build: {
    rollupOptions: {
      output: {
        manualChunks: {
          vendor: ['@philjs/core', '@philjs/router'],
          ui: ['./src/components']
        }
      }
    },
    plugins: [
      {
        name: 'budget-checker',
        closeBundle() {
          const fs = require('fs');
          const path = require('path');

          const distDir = path.resolve(__dirname, 'dist/assets');
          const files = fs.readdirSync(distDir);

          let totalJS = 0;
          let totalCSS = 0;

          files.forEach((file: string) => {
            const stats = fs.statSync(path.join(distDir, file));
            const sizeKB = stats.size / 1024;

            if (file.endsWith('.js')) {
              totalJS += sizeKB;
            } else if (file.endsWith('.css')) {
              totalCSS += sizeKB;
            }
          });

          // Check budgets
          const budgets = {
            js: 300, // 300KB
            css: 50 // 50KB
          };

          if (totalJS > budgets.js) {
            throw new Error(
              `JavaScript budget exceeded: ${totalJS.toFixed(2)}KB > ${budgets.js}KB`
            );
          }

          if (totalCSS > budgets.css) {
            throw new Error(
              `CSS budget exceeded: ${totalCSS.toFixed(2)}KB > ${budgets.css}KB`
            );
          }

          console.log('✓ Performance budgets met');
          console.log(` JS: ${totalJS.toFixed(2)}KB / ${budgets.js}KB`);
          console.log(` CSS: ${totalCSS.toFixed(2)}KB / ${budgets.css}KB`);
        }
      }
    ],
  }
});

```

Using size-limit

```
npm install -D @size-limit/file @size-limit/webpack
```

```
// package.json
{
  "scripts": {
    "size": "size-limit",
    "build": "vite build && npm run size"
  },
  "size-limit": [
    {
      "name": "Main bundle",
      "path": "dist/assets/index-*.{js,css}",
      "limit": "170 KB",
      "gzip": true
    },
    {
      "name": "Vendor bundle",
      "path": "dist/assets/vendor-*.{js,css}",
      "limit": "130 KB",
      "gzip": true
    },
    {
      "name": "Total CSS",
      "path": "dist/assets/*.css",
      "limit": "50 KB",
      "gzip": true
    }
  ]
}
```

Using bundlesize

```
npm install -D bundlesize
```

```
// package.json
{
  "scripts": {
    "test:size": "bundlesize"
  },
  "bundlesize": [
    {
      "path": "./dist/assets/index-*.{js,css}",
      "maxSize": "170 KB",
      "compression": "gzip"
    },
    {
      "path": "./dist/assets/vendor-*.{js,css}",
      "maxSize": "130 KB",
      "compression": "gzip"
    },
    {
      "path": "./dist/assets/*.css",
      "maxSize": "50 KB",
      "compression": "gzip"
    }
  ]
}
```

Runtime Budget Monitoring

Track Web Vitals

```
import { signal, effect } from '@philjs/core';

interface WebVitals {
  FCP: number;
  LCP: number;
  FID: number;
  CLS: number;
}

const vitals = signal<Partial<WebVitals>>({});

export function trackWebVitals(budget: WebVitals) {
  effect(() => {
    // First Contentful Paint
    const paintObserver = new PerformanceObserver((list) => {
      for (const entry of list.getEntries()) {
        if (entry.name === 'first-contentful-paint') {
          const fcp = entry.startTime;
          vitals.set({ ...vitals(), FCP: fcp });

          if (fcp > budget.FCP) {
            console.warn(`FCP budget exceeded: ${fcp.toFixed(0)}ms > ${budget.FCP}ms`);
            reportBudgetViolation('FCP', fcp, budget.FCP);
          }
        }
      }
    });
  });
}
```

```

    });
    paintObserver.observe({ entryTypes: ['paint'] });

    // Largest Contentful Paint
    const lcpObserver = new PerformanceObserver((list) => {
      const entries = list.getEntries();
      const lcp = entries[entries.length - 1].startTime;
      vitals.set({ ...vitals(), LCP: lcp });

      if (lcp > budget.LCP) {
        console.warn(`LCP budget exceeded: ${lcp.toFixed(0)}ms > ${budget.LCP}ms`);
        reportBudgetViolation('LCP', lcp, budget.LCP);
      }
    });
    lcpObserver.observe({ entryTypes: ['largest-contentful-paint'] });

    // First Input Delay
    const fidObserver = new PerformanceObserver((list) => {
      const entry = list.getEntries()[0] as PerformanceEventTiming;
      const fid = entry.processingStart - entry.startTime;
      vitals.set({ ...vitals(), FID: fid });

      if (fid > budget.FID) {
        console.warn(`FID budget exceeded: ${fid.toFixed(0)}ms > ${budget.FID}ms`);
        reportBudgetViolation('FID', fid, budget.FID);
      }
    });
    fidObserver.observe({ entryTypes: ['first-input'] });

    // Cumulative Layout Shift
    let clsValue = 0;
    const clsObserver = new PerformanceObserver((list) => {
      for (const entry of list.getEntries()) {
        if (!!(entry as any).hadRecentInput) {
          clsValue += (entry as any).value;
        }
      }
      vitals.set({ ...vitals(), CLS: clsValue });

      if (clsValue > budget.CLS) {
        console.warn(`CLS budget exceeded: ${clsValue.toFixed(2)} > ${budget.CLS}`);
        reportBudgetViolation('CLS', clsValue, budget.CLS);
      }
    });
    clsObserver.observe({ entryTypes: ['layout-shift'] });

    return () => {
      paintObserver.disconnect();
      lcpObserver.disconnect();
      fidObserver.disconnect();
      clsObserver.disconnect();
    };
  });

  function reportBudgetViolation(
    metric: string,
    actual: number,
    budget: number
  ) {
    // Send to analytics
    if (typeof window !== 'undefined' && (window as any).analytics) {
      (window as any).analytics.track('budget-violation', {
        metric,
        actual,
        budget,
        excess: actual - budget,
        url: window.location.pathname
      });
    }
  }

  // Usage
  trackWebVitals({
    FCP: 1800,
    LCP: 2500,
    FID: 100,
    CLS: 0.1
  });
}

```

Monitor Bundle Sizes

```

function monitorBundleSize() {
  effect(() => {
    const resources = performance.getEntriesByType('resource') as PerformanceResourceTiming[];

    const budgets = {
      js: 300 * 1024, // 300KB
      css: 50 * 1024, // 50KB
      images: 500 * 1024 // 500KB
    };

    let totalJS = 0;
    let totalCSS = 0;
    let totalImages = 0;

    resources.forEach((resource) => {
      const size = resource.transferSize || 0;

      if (resource.name.endsWith('.js')) {
        totalJS += size;
      } else if (resource.name.endsWith('.css')) {
        totalCSS += size;
      } else if (/^(png|jpg|jpeg|gif|webp|svg)$/.test(resource.name)) {
        totalImages += size;
      }
    });

    // Check budgets
    if (totalJS > budgets.js) {
      console.warn(`JS size budget exceeded: ${Math.floor(totalJS / 1024)}KB > ${Math.floor(budgets.js / 1024)}KB`);
    }

    if (totalCSS > budgets.css) {
      console.warn(`CSS size budget exceeded: ${Math.floor(totalCSS / 1024)}KB > ${Math.floor(budgets.css / 1024)}KB`);
    }

    if (totalImages > budgets.images) {
      console.warn(`Image size budget exceeded: ${Math.floor(totalImages / 1024)}KB > ${Math.floor(budgets.images / 1024)}KB`);
    }
  });
}

```

CI/CD Integration

GitHub Actions

```

# .github/workflows/performance.yml
name: Performance Budget

on:
  pull_request:
    branches: [main]

jobs:
  performance:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '24'

      - name: Install dependencies
        run: npm ci

      - name: Build
        run: npm run build

      - name: Check bundle size
        run: npm run size

      - name: Comment PR
        uses: andresz1/size-limit-action@v1
        with:
          github_token: ${{ secrets.GITHUB_TOKEN }}

```

Lighthouse CI

```
npm install -D @lhci/cli
```

```
// Lighthouserc.ts
module.exports = {
  ci: {
    collect: {
      startServerCommand: 'npm run preview',
      url: ['http://localhost:4173'],
      numberOfRuns: 3
    },
    assert: {
      preset: 'lighthouse:recommended',
      assertions: {
        'first-contentful-paint': ['error', { maxNumericValue: 1800 }],
        'largest-contentful-paint': ['error', { maxNumericValue: 2500 }],
        'cumulative-layout-shift': ['error', { maxNumericValue: 0.1 }],
        'total-blocking-time': ['error', { maxNumericValue: 300 }],
        'speed-index': ['error', { maxNumericValue: 3400 }]
      }
    }
  },
  upload: {
    target: 'temporary-public-storage'
  }
};
```

```
# .github/workflows/lighthouse.yml
name: Lighthouse CI

on:
  pull_request:
    branches: [main]

jobs:
  lighthouse:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '24'

      - name: Install dependencies
        run: npm ci

      - name: Build
        run: npm run build

      - name: Run Lighthouse CI
        run: |
          npm install -g @lhci/cli
          lhci autorun
```

Enforcement Strategies

Fail Builds on Budget Violation

```

// scripts/check-budgets.ts
import fs from 'node:fs';
import path from 'node:path';

interface Budget {
  name: string;
  path: string;
  limit: number;
}

const budgets: Budget[] = [
  { name: 'Main bundle', path: 'dist/assets/index-.js', limit: 170 * 1024 },
  { name: 'Vendor bundle', path: 'dist/assets/vendor-.js', limit: 130 * 1024 },
  { name: 'Total CSS', path: 'dist/assets/*.css', limit: 50 * 1024 }
];

function checkBudgets() {
  let violations = 0;

  budgets.forEach((budget) => {
    const files = findFiles(budget.path);
    const totalSize = files.reduce((sum, file) => {
      return sum + fs.statSync(file).size;
    }, 0);

    console.log(`\n${budget.name}:`);
    console.log(`  Size: ${totalSize / 1024}.toFixed(2)KB`);
    console.log(`  Budget: ${budget.limit / 1024}.toFixed(2)KB`);

    if (totalSize > budget.limit) {
      const excess = totalSize - budget.limit;
      console.error(`  ✖ EXCEEDED by ${excess / 1024}.toFixed(2)KB`);
      violations++;
    } else {
      const remaining = budget.limit - totalSize;
      console.log(`  ✓ ${remaining / 1024}.toFixed(2)KB remaining`);
    }
  });

  if (violations > 0) {
    console.error(`\n✖ ${violations} budget violation(s) found`);
    process.exit(1);
  } else {
    console.log(`\n✓ All budgets met`);
  }
}

function findFiles(pattern: string): string[] {
  const dir = path.dirname(pattern);
  const filePattern = path.basename(pattern);

  if (!fs.existsSync(dir)) {
    return [];
  }

  const files = fs.readdirSync(dir);
  const regex = new RegExp(
    filePattern.replace(/\/*g, '*').replace(/\?/g, '.')
  );

  return files
    .filter(file => regex.test(file))
    .map(file => path.join(dir, file));
}

checkBudgets();

// package.json
{
  "scripts": {
    "build": "vite build",
    "check:budgets": "tsx scripts/check-budgets.ts",
    "prebuild": "npm run check:budgets"
  }
}

```

Warning System

```

// scripts/warn-budgets.ts
interface BudgetStatus {
  name: string;
  size: number;
  budget: number;
  status: 'ok' | 'warning' | 'exceeded';
}

function checkWithWarnings() {
  const statuses: BudgetStatus[] = [];

  budgets.forEach((budget) => {
    const files = findFiles(budget.path);
    const totalSize = files.reduce((sum, file) => {
      return sum + fs.statSync(file).size;
    }, 0);

    let status: BudgetStatus['status'] = 'ok';

    if (totalSize > budget.limit) {
      status = 'exceeded';
    } else if (totalSize > budget.limit * 0.9) {
      status = 'warning';
    }

    statuses.push({
      name: budget.name,
      size: totalSize,
      budget: budget.limit,
      status
    });
  });

  // Print report
  console.log(`\n${Performance Budget Report}\n`);

  statuses.forEach((s) => {
    const percentage = (s.size / s.budget * 100).toFixed(1);
    const sizeKB = (s.size / 1024).toFixed(2);
    const budgetKB = (s.budget / 1024).toFixed(2);

    if (s.status === 'exceeded') {
      console.log(`! ${s.name}: ${sizeKB}KB / ${budgetKB}KB (${percentage}%)`);
    } else if (s.status === 'warning') {
      console.log(`? ${s.name}: ${sizeKB}KB / ${budgetKB}KB (${percentage}%)`);
    } else {
      console.log(`✓ ${s.name}: ${sizeKB}KB / ${budgetKB}KB (${percentage}%)`);
    }
  });

  // Fail only on exceeded
  const exceeded = statuses.filter(s => s.status === 'exceeded');
  if (exceeded.length > 0) {
    process.exit(1);
  }
}

```

Best Practices

Set Realistic Budgets

```

// Based on real metrics and goals
const realisticBudget = {
  // Start with current performance
  current: {
    FCP: 2000,
    LCP: 3000
  },
  // Set achievable improvement (10-20%)
  target: {
    FCP: 1800, // 10% improvement
    LCP: 2500 // 17% improvement
  }
};

// Unrealistic budgets
const unrealisticBudget = {
  FCP: 500, // Too aggressive
  LCP: 1000 // Impossible without major changes
};

```

Budget by Route

```
// Different budgets for different pages
const budgetsByRoute = {
  '/': {
    // Homepage - strict budget
    FCP: 1500,
    LCP: 2000,
    bundleSize: 150
  },
  '/dashboard': {
    // Dashboard - more lenient
    FCP: 2000,
    LCP: 2500,
    bundleSize: 250
  },
  '/admin': {
    // Admin - less critical
    FCP: 2500,
    LCP: 3000,
    bundleSize: 300
  }
};

function getBudgetForRoute(route: string) {
  return budgetsByRoute[route] || budgetsByRoute['/'];
}
```

Track Trends Over Time

```
interface BudgetTrend {
  date: string;
  metric: string;
  value: number;
  budget: number;
}

const trends: BudgetTrend[] = [];

function recordMetric(metric: string, value: number, budget: number) {
  trends.push({
    date: new Date().toISOString(),
    metric,
    value,
    budget
  });

  // Store in analytics
  if (typeof window !== 'undefined' && (window as any).analytics) {
    (window as any).analytics.track('performance-metric', {
      metric,
      value,
      budget,
      withinBudget: value <= budget
    });
  }
}

function analyzeTrends(metric: string, days: number = 30) {
  const since = new Date();
  since.setDate(since.getDate() - days);

  const recentTrends = trends.filter(
    t => t.metric === metric && new Date(t.date) >= since
  );

  const average = recentTrends.reduce((sum, t) => sum + t.value, 0) / recentTrends.length;
  const violations = recentTrends.filter(t => t.value > t.budget).length;

  return {
    average,
    violations,
    violationRate: violations / recentTrends.length,
    trend: calculateTrend(recentTrends)
  };
}

function calculateTrend(data: BudgetTrend[]) {
  if (data.length < 2) return 'stable';

  const first = data[0].value;
  const last = data[data.length - 1].value;
  const change = ((last - first) / first) * 100;

  if (change > 5) return 'worsening';
  if (change < -5) return 'improving';
  return 'stable';
}
```

Regular Budget Reviews

```
// Quarterly budget review
interface BudgetReview {
  quarter: string;
  currentBudgets: Record<string, number>;
  actualMetrics: Record<string, number>;
  recommendations: string[];
}

function quarterlyBudgetReview(): BudgetReview {
  const current = {
    FCP: 1800,
    LCP: 2500,
    bundleSize: 300
  };

  const actual = {
    FCP: 1650, // Consistently beating budget
    LCP: 2400, // Meeting budget
    bundleSize: 320 // Exceeding budget
  };

  const recommendations: string[] = [];

  // Adjust budgets based on actual performance
  Object.keys(current).forEach((metric) => {
    const budget = current[metric];
    const actualValue = actual[metric];

    if (actualValue < budget * 0.8) {
      recommendations.push(
        `Lower ${metric} budget to ${((actualValue * 1.1).toFixed(0))} (currently beating by 20%)`;
      );
    } else if (actualValue > budget) {
      recommendations.push(
        `Address ${metric} - exceeding budget by ${((actualValue - budget) / budget * 100).toFixed(1)}%`;
      );
    }
  });

  return {
    quarter: `${Math.ceil((new Date().getMonth() + 1) / 3)} ${new Date().getFullYear()}`,
    currentBudgets: current,
    actualMetrics: actual,
    recommendations
  };
}
```

Summary

You've learned:

- How to set budgets for timing, size, and resources
- How to monitor budgets at build time and runtime
- How to integrate budget checks into CI/CD
- How to choose enforcement strategies (warnings vs failures)
- How to track trends and analyze regressions
- How to keep budgets aligned with product goals

Performance budgets make performance measurable and enforceable.

Next: [Observability Overview](#)

Performance Budgets

Set and enforce performance budgets to maintain fast applications.

Setting Budgets

Configure Budgets

```
// vite.config.ts
export default {
  build: {
    rollupOptions: {
      output: {
        manualChunks(id) {
          if (id.includes('node_modules')) {
            return 'vendor';
          }
        }
      }
    },
    performance: {
      maxEntrypointSize: 250000, // 250kb
      maxAssetSize: 100000 // 100kb
    }
  };
}
```

Monitoring Budgets

CI Integration

```
# .github/workflows/budget.yml
name: Performance Budget

on: [pull_request]

jobs:
  budget:
    budget:
      runs-on: ubuntu-latest
      steps:
        - uses: actions/checkout@v2
        - run: npm ci
        - run: npm run build
        - run: npm run budget-check
```

Best Practices

Do: Set Realistic Budgets

```
// ⚡ Good - achievable targets
budgets: {
  bundle: 250, // 250kb
  initial: 100 // 100kb
}
```

Next Steps

- [Bundle Size](#) - Optimize bundle
- [Web Vitals](#) - Monitor vitals

Tip: Set budgets early and enforce them in CI.

Warning: Budgets should be challenging but achievable.

Note: Performance budgets prevent performance regression.

Bundle Size Optimization

Keeping your JavaScript bundle small improves load times and user experience. PhilJS provides tools and techniques to minimize bundle size.

Analyzing Bundle Size

Build Analysis

```
# Analyze your bundle
npm run build -- --analyze

# View the report
open dist/stats.html
```

Bundle Buddy

```
npm install -D @bundle-buddy/cli
bundle-buddy dist/stats.json
```

Tree Shaking

Import Only What You Need

```
// ☀ Good - imports only signal
import { signal } from '@philjs/core';

// ☠ Bad - imports everything
import * as PhilJS from '@philjs/core';
```

Code Splitting

Route-Based Splitting

```
// Automatic code splitting per route
// routes/dashboard.tsx
export default function Dashboard() {
  return <div>Dashboard</div>;
}
```

Component-Level Splitting

```
import { lazy } from '@philjs/core';

const HeavyChart = lazy(() => import('./HeavyChart'));

export default function Analytics() {
  return (
    <Suspense fallback={<div>Loading chart...</div>}>
      <HeavyChart />
    </Suspense>
  );
}
```

Reduce Dependencies

Analyze Dependencies

```
npx depcheck

# Remove unused dependencies
npm uninstall unused-package
```

Minification

Production Build

```
# Build for production (minified)
npm run build

# Gzip compression
gzip -9 dist/**/*.js
```

Best Practices

☐ Do: Use Dynamic Imports

```
// ☀ Good - loads when needed
const Chart = lazy(() => import('heavy-chart-lib'));
```

☐ Do: Remove Dead Code

```
// ☀ Good - removed unused code
if (false) {
  // This code is removed by tree shaking
  import('./never-used');
}
```

Next Steps

- [Code Splitting](#) - Split strategies
- [Lazy Loading](#) - Lazy load components
- [Performance](#) - Performance guide

☐ **Tip:** Use the bundle analyzer to identify large dependencies.

☐ ☠ **Warning:** Some libraries don't tree-shake well—check before installing.

ⓘ **Note:** PhilJS core is under 5KB gzipped, keeping your base bundle tiny.

Bundle Optimization

Minimize bundle size for faster downloads and better performance.

What You'll Learn

- Bundle analysis
- Tree shaking
- Dependency optimization
- Compression
- Build configuration
- Best practices

Bundle Analysis

Analyze Bundle Size

```
# Build with analysis
npm run build -- --analyze

# Or use rollup-plugin-visualizer
npm install -D rollup-plugin-visualizer

// vite.config.ts
import { visualizer } from 'rollup-plugin-visualizer';

export default {
  plugins: [
    visualizer({
      open: true,
      gzipSize: true,
      brotliSize: true
    })
  ]
};
```

Monitor Bundle Growth

```
// package.json
{
  "scripts": {
    "build": "vite build",
    "analyze": "vite build --analyze",
    "size": "size-limit"
  },
  "size-limit": [
    {
      "path": "dist/assets/*.js",
      "limit": "170 KB"
    }
  ]
}
```

Tree Shaking

Import Only What You Need

```
// ⚡ Named imports (tree-shakeable)
import { format } from 'date-fns';
import { debounce } from 'lodash-es';

// ⚡ Default imports (includes everything)
import _ from 'lodash';
import dateFns from 'date-fns';

// ⚡ Specific path imports
import debounce from 'lodash-es/debounce';

// ⚡ Barrel imports (may not tree-shake)
import { Button, Input, Card } from './components';
```

Mark Side-Effect-Free Code

```
// package.json
{
  "sideEffects": false
}

// Or specify files with side effects
{
  "sideEffects": [
    "*.css",
    "*.scss",
    "./src/polyfills.ts"
  ]
}
```

Dependency Optimization

Replace Large Libraries

```
// Moment.js (67KB)
import moment from 'moment';
const formatted = moment().format('YYYY-MM-DD');

// date-fns (modular, ~2KB per function)
import { format } from 'date-fns';
const formatted = format(new Date(), 'yyyy-MM-dd');

// Lodash (24KB full)
import _ from 'lodash';

// Lodash-es (tree-shakeable)
import { debounce, throttle } from 'lodash-es';

// Axios (13KB)
import axios from 'axios';

// Native fetch (0KB)
const response = await fetch(url);
```

Use Lighter Alternatives

```
// Heavy library alternatives:
// jQuery (87KB) → Native DOM APIs (0KB)
// Moment.js (67KB) → date-fns (2-10KB) or Day.js (2KB)
// Lodash (24KB) → Native methods or Lodash-es (tree-shakeable)
// Axios (13KB) → fetch (0KB) or ky (4KB)
```

Dynamic Imports for Heavy Features

```
// Load PDF Library only when needed
async function exportToPDF() {
  const { jsPDF } = await import('jspdf');
  // Use jsPDF
}

// Load chart library only for chart page
const ChartPage = lazy(() => import('./ChartPage'));

// Conditionally Load polyfills
if (!('IntersectionObserver' in window)) {
  await import('intersection-observer');
}
```

Code Splitting Strategy

Split by Route

```
// vite.config.ts
export default {
  build: {
    rollupOptions: {
      output: {
        manualChunks: {
          // Separate chunk per route
          home: ['./src/pages/Home'],
          dashboard: ['./src/pages/Dashboard'],
          settings: ['./src/pages/Settings']
        }
      }
    }
  }
};
```

Vendor Splitting

```
// vite.config.ts
export default {
  build: {
    rollupOptions: {
      output: {
        manualChunks(id) {
          // Vendor chunks
          if (id.includes('node_modules')) {
            // Split large vendors separately
            if (id.includes('chart.js')) return 'vendor-charts';
            if (id.includes('@date-fns')) return 'vendor-date';

            // Everything else in main vendor
            return 'vendor';
          }
        }
      }
    }
  }
};
```

Compression

Enable Gzip/Brotli

```
// vite.config.ts
import compress from 'vite-plugin-compression';

export default {
  plugins: [
    compress({
      algorithm: 'gzip',
      ext: '.gz'
    }),
    compress({
      algorithm: 'brotliCompress',
      ext: '.br'
    })
  ]
};
```

Server Configuration

```
# nginx.conf
gzip on;
gzip_types text/plain text/css application/json application/javascript;
gzip_min_length 1000;

# Brotli
brotli on;
brotli_types text/plain text/css application/json application/javascript;
```

Minification

Optimize Production Build

```
// vite.config.ts
export default {
  build: {
    minify: 'terser',
    terserOptions: {
      compress: {
        drop_console: true, // Remove console.log
        drop_debugger: true,
        pure_funcs: ['console.log', 'console.info']
      },
      format: {
        comments: false // Remove comments
      }
    }
  }
};
```

CSS Minification

```
// vite.config.ts
export default {
  css: {
    devSourcemap: false
  },
  build: {
    cssCodeSplit: true,
    cssMinify: true
  }
};
```

Asset Optimization

Image Optimization

```
// vite.config.ts
import imagemin from 'vite-plugin-imagemin';

export default {
  plugins: [
    imagemin({
      gifsicle: { optimizationLevel: 7 },
      optipng: { optimizationLevel: 7 },
      mozjpeg: { quality: 80 },
      svgo: {
        plugins: [
          { removeViewBox: false },
          { removeEmptyAttrs: true }
        ]
      }
    })
  ]
};
```

Font Subsetting

```
// Only include characters you need
@font-face {
  font-family: 'CustomFont';
  src: url('/fonts/custom-latin.woff2') format('woff2');
  unicode-range: U+0000-00FF; // Latin only
}
```

Remove Unused Code

Remove Development Code

```
// Use environment variables
if (import.meta.env.DEV) {
  // Development-only code (removed in production)
  console.log('Debug info');
}

// Dead code elimination
const DEBUG = false;

if (DEBUG) {
  // This entire block is removed in production
  console.log('Never called');
}
```

Purge Unused CSS

```
// vite.config.ts
export default {
  build: {
    cssCodeSplit: true
  }
};

// With PurgeCSS
import purgecss from '@fullhuman/postcss-purgecss';

export default {
  css: {
    postcss: {
      plugins: [
        purgecss({
          content: ['./src/**/*.{ts,html}'],
          safelist: ['active', 'disabled'] // Keep these classes
        })
      ]
    }
  }
};
```

Module Federation

Share Dependencies

```
// vite.config.ts
import federation from '@originjs/vite-plugin-federation';

export default {
  plugins: [
    federation({
      name: 'app',
      shared: {
        '@philjs/core': {},
        '@philjs/router': {}
      }
    })
  ]
};
```

Best Practices

Audit Dependencies Regularly

```
# Check bundle size impact
npm install -g bundle-wizard
bundle-wizard

# Find duplicate dependencies
npm dedupe

# Remove unused dependencies
npm prune
```

Use Modern JavaScript

```
// vite.config.ts
export default {
  build: {
    target: 'es2020', // Modern browsers only
    polyfillModulePreload: false // Skip if not needed
  }
};
```

Lazy Load Non-Critical Features

```
// ⚡ Load analytics asynchronously
setTimeout(() => {
  import('./analytics').then(({ init }) => init());
}, 3000);

// ⚡ Load non-critical UI later
const Footer = lazy(() => import('./Footer'));
const CookieBanner = lazy(() => import('./CookieBanner'));
```

Monitor Third-Party Scripts

```
// ⚡ Load third-party scripts async
<script async src="https://analytics.example.com/script.js" />

// ⚡ Use facades for heavy embeds
function YouTubeEmbed({ videoId }: { videoId: string }) {
  const [loaded, setLoaded] = signal(false);

  if (!loaded()) {
    return (
      <div
        onClick={() => setLoaded(true)}
        style={{ cursor: 'pointer' }}
      >
        <img src={`https://img.youtube.com/vi/${videoId}/0.jpg`} />
        <div> Play Video</div>
      </div>
    );
  }

  return (
    <iframe
      src={`https://www.youtube.com/embed/${videoId}`}
      allow="accelerometer; autoplay; encrypted-media"
    />;
  )
}
```

Performance Budget

```
// .budgetrc.json
{
  "budgets": [
    {
      "path": "dist/assets/*.js",
      "limit": "170 KB",
      "gzip": true
    },
    {
      "path": "dist/assets/*.css",
      "limit": "30 KB",
      "gzip": true
    },
    {
      "path": "dist/index.html",
      "limit": "10 KB"
    }
  ]
}
```

Summary

You've learned:

- Bundle analysis tools
- Tree shaking optimization
- Dependency replacement strategies
- Compression techniques
- Code splitting patterns
- Asset optimization
- Removing unused code
- Performance budgets

Smaller bundles mean faster load times!

Next: [Runtime Performance](#) → Optimize runtime execution

Code Splitting

Split your application into smaller chunks for faster initial load times.

What You'll Learn

- Route-based splitting
- Component-based splitting
- Dynamic imports
- Lazy loading
- Chunk optimization
- Best practices

Route-Based Splitting

Lazy Load Routes

```

import { Router, Route, lazy } from '@philjs/router';

// Lazy Load route components
const Home = lazy(() => import('./pages/Home'));
const About = lazy(() => import('./pages/About'));
const Dashboard = lazy(() => import('./pages/Dashboard'));
const Settings = lazy(() => import('./pages/Settings'));

export function App() {
  return (
    <Router>
      <Route path="/" component={Home} />
      <Route path="/about" component={About} />
      <Route path="/dashboard" component={Dashboard} />
      <Route path="/settings" component={Settings} />
    </Router>
  );
}

```

With Loading States

```

import { Router, Route, lazy, Suspense } from '@philjs/router';

const Dashboard = lazy(() => import('./pages/Dashboard'));

function LoadingFallback() {
  return (
    <div style={{ padding: '40px', textAlign: 'center' }}>
      <div className="spinner" />
      <p>Loading...</p>
    </div>
  );
}

export function App() {
  return (
    <Router>
      <Route
        path="/dashboard"
        component={() => (
          <Suspense fallback={<LoadingFallback />}>
            <Dashboard />
          </Suspense>
        )}
      />
    </Router>
  );
}

```

Component-Based Splitting

Lazy Load Heavy Components

```

import { lazy, Suspense, signal } from '@philjs/core';

// Split heavy components
const ChartComponent = lazy(() => import('../components/Chart'));
const DataTable = lazy(() => import('../components/DataTable'));
const RichTextEditor = lazy(() => import('../components/RichTextEditor'));

export function Dashboard() {
  const activeTab = signal('overview');

  return (
    <div>
      <nav>
        <button onClick={() => activeTab.set('overview')}>Overview</button>
        <button onClick={() => activeTab.set('charts')}>Charts</button>
        <button onClick={() => activeTab.set('data')}>Data</button>
      </nav>

      {activeTab() === 'overview' && <Overview />}

      {activeTab() === 'charts' && (
        <Suspense fallback=<div>Loading charts...</div>>
          <ChartComponent />
        </Suspense>
      )}

      {activeTab() === 'data' && (
        <Suspense fallback=<div>Loading table...</div>>
          <DataTable />
        </Suspense>
      )}
    </div>
  )
}

```

Modal Code Splitting

```

import { lazy, Suspense, signal } from '@philjs/core';

const UserModal = lazy(() => import('../modals/UserModal'));
const SettingsModal = lazy(() => import('../modals/SettingsModal'));

export function App() {
  const activeModal = signal<'user' | 'settings' | null>(null);

  return (
    <div>
      <button onClick={() => activeModal.set('user')}>
        Open User Modal
      </button>

      <button onClick={() => activeModal.set('settings')}>
        Open Settings
      </button>

      {activeModal() === 'user' && (
        <Suspense fallback=<div>Loading...</div>>
          <UserModal onClose={() => activeModal.set(null)} />
        </Suspense>
      )}

      {activeModal() === 'settings' && (
        <Suspense fallback=<div>Loading...</div>>
          <SettingsModal onClose={() => activeModal.set(null)} />
        </Suspense>
      )}
    </div>
  )
}

```

Dynamic Imports

Conditional Loading

```

import { signal } from '@philjs/core';

export function FeatureToggle() {
  const adminMode = signal(false);

  const loadAdminPanel = async () => {
    if (!adminMode()) {
      // Only Load admin code when needed
      const { AdminPanel } = await import('./components/AdminPanel');

      // Render admin panel
      renderAdminPanel(AdminPanel);
    }
    adminMode.set(true);
  };

  return (
    <div>
      <button onClick={loadAdminPanel}>
        Enable Admin Mode
      </button>

      {adminMode() && <div id="admin-panel-container" />}
    </div>
  );
}

```

Library Code Splitting

```

// Load heavy Libraries only when needed
async function exportToPDF() {
  // jsPDF is only Loaded when export is triggered
  const { jsPDF } = await import('jspdf');

  const doc = new jsPDF();
  doc.text('Hello world!', 10, 10);
  doc.save('document.pdf');
}

async function exportToExcel() {
  // xlsx is only Loaded when export is triggered
  const XLSX = await import('xlsx');

  const ws = XLSX.utils.json_to_sheet(data);
  const wb = XLSX.utils.book_new();
  XLSX.utils.book_append_sheet(wb, ws, 'Sheet1');
  XLSX.writeFile(wb, 'data.xlsx');
}

export function DataExport({ data }: { data: any[] }) {
  return (
    <div>
      <button onClick={exportToPDF}>Export PDF</button>
      <button onClick={exportToExcel}>Export Excel</button>
    </div>
  );
}

```

Chunk Optimization

Manual Chunks

```
// vite.config.ts
import { defineConfig } from 'vite';

export default defineConfig({
  build: {
    rollupOptions: {
      output: {
        manualChunks: {
          // Vendor chunk
          vendor: ['@philjs/core', '@philjs/router'],

          // UI components chunk
          ui: [
            './src/components/Button',
            './src/components/Input',
            './src/components/Card'
          ],
          // Charts chunk
          charts: ['chart.js', './src/components/Chart'],
          // Forms chunk
          forms: ['zod', './src/components/Form'],
          // Utils chunk
          utils: ['date-fns', 'lodash-es']
        }
      }
    }
  }
});
```

Smart Chunking

```
// vite.config.ts
export default defineConfig({
  build: {
    rollupOptions: {
      output: {
        manualChunks(id) {
          // Vendor chunk for node_modules
          if (id.includes('node_modules')) {
            // Split Large vendors
            if (id.includes('chart.js')) {
              return 'chart-vendor';
            }
            if (id.includes('date-fns')) {
              return 'date-vendor';
            }
            return 'vendor';
          }

          // Component chunks by directory
          if (id.includes('/components/')) {
            const componentPath = id.split('/components/')[1];
            const componentDir = componentPath.split('/')[0];
            return `components-${componentDir}`;
          }
        }
      }
    }
  }
});
```

Preloading Strategies

Prefetch Routes

```

import { Router, Route, prefetch } from '@philjs/router';

export function Navigation() {
  return (
    <nav>
      <a href="/dashboard"
        onMouseEnter={() => prefetch('/dashboard')}
      >
        Dashboard
      </a>

      <a href="/reports"
        onMouseEnter={() => prefetch('/reports')}
      >
        Reports
      </a>
    </nav>
  );
}

```

Intelligent Preloading

```

import { effect } from '@philjs/core';

function preloadLikelyRoutes() {
  effect(() => {
    // PreLoad based on user behavior
    const userRole = getUserRole();

    if (userRole === 'admin') {
      // Admins Likely visit settings
      prefetch('/settings');
      prefetch('/users');
    } else if (userRole === 'user') {
      // Regular users Likely visit dashboard
      prefetch('/dashboard');
    }

    // PreLoad based on time
    const hour = new Date().getHours();
    if (hour >= 9 && hour <= 17) {
      // Working hours - preload work-related pages
      prefetch('/reports');
    }
  });
}

```

Link Prefetching

```

function PrefetchLink({ href, children }: {
  href: string;
  children: any;
}) {
  const prefetched = signal(false);

  const handleMouseEnter = () => {
    if (!prefetched()) {
      // Prefetch on hover
      const link = document.createElement('link');
      link.rel = 'prefetch';
      link.href = href;
      document.head.appendChild(link);

      prefetched.set(true);
    }
  };

  return (
    <a href={href} onMouseEnter={handleMouseEnter}>
      {children}
    </a>
  );
}

```

Error Handling

Lazy Load Error Boundaries

```

import { ErrorBoundary } from '@philjs/core';

function LazyLoadErrorFallback({ error, retry }: {
  error: Error;
  retry: () => void;
}) {
  return (
    <div className="error-container">
      <h2>Failed to load component</h2>
      <p>{error.message}</p>
      <button onClick={retry}>Retry</button>
    </div>
  );
}

export function App() {
  return (
    <ErrorBoundary fallback={LazyLoadErrorFallback}>
      <Suspense fallback={<Loading />}>
        <LazyComponent />
      </Suspense>
    </ErrorBoundary>
  );
}

```

Retry Failed Chunks

```

function lazyWithRetry<T>(
  importFn: () => Promise<T>,
  retries = 3,
  delay = 1000
): () => Promise<T> {
  return async () => {
    for (let i = 0; i < retries; i++) {
      try {
        return await importFn();
      } catch (error) {
        if (i === retries - 1) {
          throw error;
        }
      }
    }
    // Wait before retry
    await new Promise(resolve => setTimeout(resolve, delay * (i + 1)));
  }
}

throw new Error('Failed to load chunk after retries');
}

// Usage
const Dashboard = lazy(
  lazyWithRetry(() => import('./pages/Dashboard'))
);

```

Best Practices

Split by Route

```

// ⚡ Each route is a separate chunk
const Home = lazy(() => import('./pages/Home'));
const Dashboard = lazy(() => import('./pages/Dashboard'));
const Settings = lazy(() => import('./pages/Settings'));

// ⚡ ALL routes in main bundle
import Home from './pages/Home';
import Dashboard from './pages/Dashboard';
import Settings from './pages/Settings';

```

Lazy Load Below the Fold

```
// ⚠ Lazy Load content not immediately visible
function HomePage() {
  return (
    <div>
      <Hero /> {/* Eager Loaded */}
      <Suspense fallback={<Loading />}>
        <LazyFeatures /> {/* Lazy Loaded */}
      </Suspense>
    </div>
  );
}
```

Avoid Over-Splitting

```
// ⚠ Reasonable chunk sizes
const Dashboard = lazy(() => import('./pages/Dashboard')) // 50KB

// ⚠ Too many tiny chunks (overhead)
const Button = lazy(() => import('./Button')) // 2KB
const Input = lazy(() => import('./Input')) // 1KB
```

Use Suspense Wisely

```
// ⚠ Group related Lazy components
<Suspense fallback={<Loading />}>
  <LazyChart />
  <LazyTable />
  <LazyStats />
</Suspense>

// ⚠ Too many Loading states
<Suspense fallback={<Loading />}>
  <LazyChart />
</Suspense>
<Suspense fallback={<Loading />}>
  <LazyTable />
</Suspense>
```

Measuring Impact

Bundle Analysis

```
# Analyze bundle size
npm run build -- --analyze

# Look for:
# 1. Largest chunks
# 2. Duplicate code
# 3. Unused dependencies
```

Performance Monitoring

```
import { effect } from '@philjs/core';

function trackChunkLoad(chunkName: string) {
  const start = performance.now();

  return () => {
    const duration = performance.now() - start;

    // Send to analytics
    analytics.track('chunk-loaded', {
      chunk: chunkName,
      duration,
      timestamp: Date.now()
    });
  };
}

// Usage
const Dashboard = lazy(async () => {
  const trackEnd = trackChunkLoad('dashboard');
  const module = await import('./pages/Dashboard');
  trackEnd();
  return module;
});
```

Summary

You've learned:

- Route-based code splitting □ Component-based splitting □ Dynamic imports for conditional loading □ Chunk optimization strategies □ Preloading and prefetching □ Error handling for lazy loads □ Best practices for splitting □ Performance measurement

Code splitting dramatically improves initial load time!

Next: [Lazy Loading](#) → Advanced lazy loading patterns

Lazy Loading

Defer loading of resources until they're needed to improve initial page load.

What You'll Learn

- Image lazy loading
- Component lazy loading
- Intersection Observer
- Progressive loading
- Resource priorities
- Best practices

Image Lazy Loading

Native Lazy Loading

```
export function LazyImage({ src, alt }: {  
  src: string;  
  alt: string;  
}) {  
  return (  
    <img  
      src={src}  
      alt={alt}  
      loading="lazy"  
      decoding="async"  
      width="800"  
      height="600"  
    />  
  );  
}
```

Intersection Observer

```
import { signal, effect } from '@philjs/core';  
  
export function IntersectionLazyImage({ src, alt }: {  
  src: string;  
  alt: string;  
}) {  
  const isVisible = signal(false);  
  const loaded = signal(false);  
  let imgRef: HTMLImageElement | undefined;  
  
  effect(() => {  
    if (!imgRef) return;  
  
    const observer = new IntersectionObserver(  
      (entries) => {  
        entries.forEach((entry) => {  
          if (entry.isIntersecting && !loaded()) {  
            isVisible.set(true);  
            loaded.set(true);  
          }  
        });  
      },  
      { rootMargin: '50px' } // Load 50px before visible  
    );  
  
    observer.observe(imgRef);  
  
    return () => observer.disconnect();  
  });  
  
  return (  
    <img  
      ref={imgRef}  
      src={isVisible() ? src : '/placeholder.jpg'}  
      alt={alt}  
      style={{ minHeight: '200px' }}  
    />  
  );  
}
```

Progressive Image Loading

```
import { signal, effect } from '@philjs/core';

export function ProgressiveImage({ src, placeholder, alt }: {
  src: string;
  placeholder: string;
  alt: string;
}) {
  const currentSrc = signal(placeholder);
  const isLoaded = signal(false);

  effect(() => {
    const img = new Image();

    img.onload = () => {
      currentSrc.set(src);
      isLoaded.set(true);
    };
  });

  img.src = src;
}

return (
  <img
    src={currentSrc}
    alt={alt}
    style={{
      filter: isLoaded() ? 'none' : 'blur(10px)',
      transition: 'filter 0.3s'
    }}
  />
);
}
```

Blur Hash Implementation

```
import { signal, effect } from '@philjs/core';

export function BlurHashImage({ src, blurHash, alt }: {
  src: string;
  blurHash: string; // Base64 blur hash
  alt: string;
}) {
  const loaded = signal(false);

  effect(() => {
    const img = new Image();
    img.onload = () => loaded.set(true);
    img.src = src;
  });

  return (
    <div style={{ position: 'relative' }}>
      {/* Blur hash background */}
      <img
        src={blurHash}
        alt=""
        style={{
          position: 'absolute',
          width: '100%',
          height: '100%',
          filter: 'blur(20px)',
          transform: 'scale(1.1)',
          opacity: loaded() ? 0 : 1,
          transition: 'opacity 0.3s'
        }}
      />
      {/* Actual image */}
      <img
        src={src}
        alt={alt}
        style={{
          position: 'relative',
          width: '100%',
          opacity: loaded() ? 1 : 0,
          transition: 'opacity 0.3s'
        }}
      />
    </div>
  );
}
```

Component Lazy Loading

Visibility-Based Loading

```
import { signal, effect } from '@philjs/core';

function useLazyLoad() {
  const isVisible = signal(false);
  let ref: HTMLElement | undefined;

  effect(() => {
    if (!ref) return;

    const observer = new IntersectionObserver(
      (entries) => {
        if (entries[0].isIntersecting) {
          isVisible.set(true);
          observer.disconnect();
        }
      },
      { rootMargin: '100px' }
    );
  });

  observer.observe(ref);

  return () => observer.disconnect();
});

return { isVisible, ref };
}

export function LazySection() {
  const { isVisible, ref } = useLazyLoad();

  return (
    <div ref={ref}>
      {isVisible() ? (
        <HeavyComponent />
      ) : (
        <div style={{ minHeight: '400px' }}>
          Loading section...
        </div>
      )}
    </div>
  );
}
```

Scroll-Based Loading

```
import { signal, effect } from '@philjs/core';

export function ScrollLazyLoad({ children }: { children: any }) {
  const shouldLoad = signal(false);

  effect(() => {
    const checkScroll = () => {
      const scrollPercent =
        (window.scrollY / (document.body.scrollHeight - window.innerHeight)) * 100;

      if (scrollPercent > 30) {
        shouldLoad.set(true);
        window.removeEventListener('scroll', checkScroll);
      }
    };

    window.addEventListener('scroll', checkScroll);
    checkScroll(); // Check initial position

    return () => window.removeEventListener('scroll', checkScroll);
  });

  return shouldLoad() ? children : null;
}

// Usage
<ScrollLazyLoad>
  <Footer />
</ScrollLazyLoad>
```

Background Loading

Idle Time Loading

```

import { effect } from '@philjs/core';

function loadDuringIdle(importFn: () => Promise<any>) {
  effect(() => {
    if ('requestidlecallback' in window) {
      requestIdleCallback(() => {
        importFn();
      });
    } else {
      // Fallback for Safari
      setTimeout(() => {
        importFn();
      }, 1);
    }
  });
}

// Usage
export function App() {
  effect(() => {
    // Preload non-critical components during idle time
    loadDuringIdle(() => import('./components/Analytics'));
    loadDuringIdle(() => import('./components/ChatWidget'));
  });

  return <MainApp />;
}

```

Priority Queue Loading

```

import { signal } from '@philjs/core';

interface LoadTask {
  priority: number;
  load: () => Promise<any>;
  name: string;
}

class ResourceLoader {
  private queue = signal<LoadTask[]>([]);
  private loading = signal(false);

  addTask(task: LoadTask) {
    const updated = [...this.queue(), task]
      .sort((a, b) => b.priority - a.priority);

    this.queue.set(updated);
    this.processQueue();
  }

  private async processQueue() {
    if (this.loading() || this.queue().length === 0) return;

    this.loading.set(true);
    const task = this.queue()[0];

    try {
      await task.load();
      console.log(`Loaded: ${task.name}`);
    } catch (error) {
      console.error(`Failed to load: ${task.name}`, error);
    }

    this.queue.set(this.queue().slice(1));
    this.loading.set(false);

    this.processQueue();
  }
}

const loader = new ResourceLoader();

// Usage
loader.addTask({
  priority: 10,
  name: 'Critical Analytics',
  load: () => import('./analytics')
});

loader.addTask({
  priority: 5,
  name: 'Chat Widget',
  load: () => import('./chat')
});

```

Video Lazy Loading

Lazy Video Element

```
import { signal, effect } from '@philjs/core';

export function LazyVideo({ src, poster }: {
  src: string;
  poster: string;
}) {
  const isVisible = signal(false);
  const shouldLoad = signal(false);
  let videoRef: HTMLVideoElement | undefined;

  effect(() => {
    if (!videoRef) return;

    const observer = new IntersectionObserver(
      (entries) => {
        entries.forEach((entry) => {
          if (entry.isIntersecting) {
            isVisible.set(true);
          }
        });
      },
      { threshold: 0.5 }
    );

    observer.observe(videoRef);

    return () => observer.disconnect();
  });

  effect(() => {
    if (isVisible() && !shouldLoad()) {
      shouldLoad.set(true);
    }
  });
}

return (
  <video
    ref={videoRef}
    poster={poster}
    controls
    preload={shouldLoad() ? 'auto' : 'none'}
  >
  {shouldLoad() && <source src={src} type="video/mp4" />}
  </video>
);
}
```

Autoplay on Visible

```

export function AutoplayVideo({ src }: { src: string }) {
  const isPlaying = signal(false);
  let videoRef: HTMLVideoElement | undefined;

  effect(() => {
    if (!videoRef) return;

    const observer = new IntersectionObserver(
      (entries) => {
        entries.forEach((entry) => {
          if (entry.isIntersecting && !isPlaying()) {
            videoRef?.play();
            isPlaying.set(true);
          } else if (!entry.isIntersecting && isPlaying()) {
            videoRef?.pause();
            isPlaying.set(false);
          }
        });
      },
      { threshold: 0.5 }
    );
  });

  observer.observe(videoRef);

  return () => observer.disconnect();
});

return (
  <video
    ref={videoRef}
    src={src}
    muted
    loop
    playsInline
  />
);
}

```

Resource Hints

Prefetch Resources

```

function prefetchResource(url: string, type: 'script' | 'style' | 'fetch' = 'fetch') {
  const link = document.createElement('link');
  link.rel = 'prefetch';
  link.href = url;

  if (type === 'script') {
    link.as = 'script';
  } else if (type === 'style') {
    link.as = 'style';
  } else {
    link.as = 'fetch';
    link.setAttribute('crossorigin', 'anonymous');
  }

  document.head.appendChild(link);
}

// Usage
export function App() {
  effect(() => {
    // Prefetch Likely next page
    prefetchResource('/dashboard.js', 'script');
    prefetchResource('/api/user-data', 'fetch');
  });

  return <HomePage />;
}

```

Preload Critical Resources

```

function preloadResource(url: string, type: 'script' | 'style' | 'font' | 'image') {
  const link = document.createElement('link');
  link.rel = 'preload';
  link.href = url;
  link.as = type;

  if (type === 'font') {
    link.setAttribute('crossorigin', 'anonymous');
  }

  document.head.appendChild(link);
}

// Preload critical resources early
preloadResource('/fonts/main.woff2', 'font');
preloadResource('/hero-image.jpg', 'image');
preloadResource('/critical.css', 'style');

```

Font Loading

Font Display Swap

```

@font-face {
  font-family: 'CustomFont';
  src: url('/fonts/custom.woff2') format('woff2');
  font-display: swap; /* Show fallback immediately */
}

```

Progressive Font Loading

```

import { signal, effect } from '@philjs/core';

export function useFontLoading(fontFamily: string) {
  const isLoaded = signal(false);

  effect(() => {
    if ('fonts' in document) {
      document.fonts.ready.then(() => {
        // Check if specific font is loaded
        const loaded = document.fonts.check(`16px ${fontFamily}`);
        isLoaded.set(loaded);
      });
    }
  });

  return isLoaded;
}

// Usage
export function App() {
  const fontLoaded = useFontLoading('CustomFont');

  return (
    <div style={{ 
      fontFamily: fontLoaded() 
      ? 'CustomFont, sans-serif' 
      : 'sans-serif' 
    }}>
      Content
    </div>
  );
}

```

Best Practices

Lazy Load Below the Fold

```

// ⚡ Lazy Load content not immediately visible
function HomePage() {
  return (
    <div>
      <Hero /> {/* Eager load above fold */}

      <LazyLoad>
        <Features /> {/* Lazy Load below fold */}
      </LazyLoad>

      <LazyLoad>
        <Testimonials /> {/* Lazy Load below fold */}
      </LazyLoad>
    </div>
  );
}

```

Set Image Dimensions

```
// ⚠ Reserve space to avoid layout shift


// ⚠ No dimensions (causes layout shift)

```

Use Appropriate Root Margin

```
// ⚠ Load before visible for smooth experience
const observer = new IntersectionObserver(callback, {
  rootMargin: '50px' // Load 50px before viewport
});

// ⚠ Load only when visible (may show Loading state)
const observer = new IntersectionObserver(callback, {
  rootMargin: '0px'
});
```

Prioritize Critical Resources

```
// ⚠ Preload critical, Lazy Load non-critical
<link rel="preload" href="/critical.css" as="style" />



// ⚠ Lazy Load everything
 // Hero should Load immediately
```

Clean Up Observers

```
// ⚠ Disconnect observer when done
effect(() => {
  const observer = new IntersectionObserver(callback);
  observer.observe(element);

  return () => observer.disconnect();
});

// ⚠ Forget to clean up (memory leak)
const observer = new IntersectionObserver(callback);
observer.observe(element);
```

Summary

You've learned:

□ Image lazy loading techniques □ Component lazy loading □ Intersection Observer API □ Progressive loading patterns □ Background and idle loading □ Video lazy loading □ Resource hints (prefetch, preload) □ Font loading strategies □ Best practices

Lazy loading dramatically reduces initial page weight!

Next: [Memoization](#) → Cache expensive computations

Memoization

Cache expensive computations and prevent unnecessary re-renders for better performance.

What You'll Learn

- Computed values with memo
- Component memoization
- Selector patterns
- Cache strategies
- When to memoize
- Best practices

Memo Basics

Simple Memoization

```

import { signal, memo } from '@philjs/core';

function ExpensiveCalculation() {
  const numbers = signal([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);

  // Memoize expensive calculation
  const sum = memo(() => {
    console.log('Calculating sum...');
    return numbers().reduce((acc, n) => acc + n, 0);
  });

  const average = memo(() => {
    console.log('Calculating average...');
    return sum() / numbers().length;
  });

  return (
    <div>
      <p>Sum: {sum()}</p>
      <p>Average: {average()}</p>
    </div>
  );
}

```

Derived State

```

import { signal, memo } from '@philjs/core';

function UserList() {
  const users = signal([
    { id: 1, name: 'Alice', active: true },
    { id: 2, name: 'Bob', active: false },
    { id: 3, name: 'Charlie', active: true }
  ]);

  const searchTerm = signal('');

  // Memoize filtered list
  const filteredUsers = memo(() => {
    return users().filter(user =>
      user.name.toLowerCase().includes(searchTerm().toLowerCase())
    );
  });

  // Memoize active users
  const activeUsers = memo(() => {
    return filteredUsers().filter(user => user.active);
  });

  return (
    <div>
      <input
        value={searchTerm()}
        onChange={(e) => searchTerm.set(e.target.value)}
        placeholder="Search users..." />

      <p>Active users: {activeUsers().length}</p>

      {filteredUsers().map(user => (
        <UserCard key={user.id} user={user} />
      ))}
    </div>
  );
}

```

Component Memoization

Memo Component Wrapper

```

import { memo as memoComponent } from '@philjs/core';

interface UserCardProps {
  user: {
    id: number;
    name: string;
    email: string;
  };
}

// Memoize component to prevent re-renders
const UserCard = memoComponent(({ user }: UserCardProps) => {
  console.log(`Rendering UserCard for ${user.name}`);
  return (
    <div className="card">
      <h3>{user.name}</h3>
      <p>{user.email}</p>
    </div>
  );
});

// Parent component
function UserList() {
  const users = signal([
    { id: 1, name: 'Alice', email: 'alice@example.com' },
    { id: 2, name: 'Bob', email: 'bob@example.com' }
  ]);

  const count = signal(0);

  return (
    <div>
      {/* UserCards won't re-render when count changes */}
      <button onClick={() => count.set(count() + 1)}>
        Count: {count()}
      </button>

      {users().map(user => (
        <UserCard key={user.id} user={user} />
      ))}
    </div>
  );
}

```

Custom Comparison

```

function arePropsEqual<T>(prev: T, next: T): boolean {
  // Custom comparison logic
  if (typeof prev !== typeof next) return false;

  if (typeof prev === 'object' && prev !== null && next !== null) {
    const prevKeys = Object.keys(prev);
    const nextKeys = Object.keys(next);

    if (prevKeys.length !== nextKeys.length) return false;

    return prevKeys.every(key =>
      prev[key as keyof T] === next[key as keyof T]
    );
  }

  return prev === next;
}

const MemoizedComponent = memoComponent(
  MyComponent,
  arePropsEqual
);

```

Selector Patterns

Memoized Selectors

```

import { signal, memo } from '@philjs/core';

interface Todo {
  id: number;
  text: string;
  completed: boolean;
}

function createTodoSelectors(todos: () => Todo[]) {
  const completedTodos = memo(() =>
    todos().filter(todo => todo.completed)
  );

  const activeTodos = memo(() =>
    todos().filter(todo => !todo.completed)
  );

  const todoCount = memo(() => ({
    total: todos().length,
    completed: completedTodos().length,
    active: activeTodos().length
  }));
}

return { completedTodos, activeTodos, todoCount };
}

// Usage
function TodoApp() {
  const todos = signal<Todo[]>([
    { id: 1, text: 'Learn PhilJS', completed: true },
    { id: 2, text: 'Build app', completed: false }
  ]);

  const selectors = createTodoSelectors(todos);

  return (
    <div>
      <TodoStats stats={selectors.todoCount()} />
      <TodoList todos={selectors.activeTodos()} />
    </div>
  );
}

```

Parameterized Selectors

```

function createUserSelector(users: () => User[]) {
  // Cache for memoized results
  const cache = new Map<number, () => User | undefined>();

  const getUserId = (id: number) => {
    if (!cache.has(id)) {
      cache.set(id, memo(() =>
        users().find(user => user.id === id)
      ));
    }
    return cache.get(id)!;
  };

  return { getUserId };
}

// Usage
function UserProfile({ userId }: { userId: number }) {
  const users = signal<User[]>([/* ... */]);
  const { getUserId } = createUserSelector(users);

  const user = getUserId(userId);

  return (
    <div>
      {user() ? (
        <h2>{user()!.name}</h2>
      ) : (
        <p>User not found</p>
      )}
    </div>
  );
}

```

Advanced Memoization

Deep Memoization

```
import { signal, memo } from '@philjs/core';

function deepMemo<T>(fn: () => T): () => T {
  let prevValue: T | undefined;
  let prevResult: T | undefined;

  return memo(() => {
    const currentValue = fn();

    if (JSON.stringify(currentValue) === JSON.stringify(prevValue)) {
      return prevResult!;
    }

    prevValue = currentValue;
    prevResult = currentValue;
    return currentValue;
  });
}

// Usage
const complexData = signal({
  user: { name: 'Alice', age: 30 },
  settings: { theme: 'dark', notifications: true }
});

const memoizedData = deepMemo(() => ({
  ...complexData(),
  timestamp: Date.now()
}));
```

LRU Cache Memoization

```

class LRUCache<K, V> {
  private cache = new Map<K, V>();
  private maxSize: number;

  constructor(maxSize: number = 100) {
    this.maxSize = maxSize;
  }

  get(key: K): V | undefined {
    if (!this.cache.has(key)) return undefined;

    // Move to end (most recent)
    const value = this.cache.get(key)!;
    this.cache.delete(key);
    this.cache.set(key, value);

    return value;
  }

  set(key: K, value: V): void {
    if (this.cache.has(key)) {
      this.cache.delete(key);
    } else if (this.cache.size >= this.maxSize) {
      // Remove oldest (first item)
      const oldestKey = this.cache.keys().next().value;
      this.cache.delete(oldestKey);
    }

    this.cache.set(key, value);
  }
}

function createMemoizedFn<Args extends any[], Result>(
  fn: (...args: Args) => Result,
  keyFn: (...args: Args) => string
) {
  const cache = new LRUCache<string, Result>(50);

  return (...args: Args): Result => {
    const key = keyFn(...args);
    const cached = cache.get(key);

    if (cached !== undefined) {
      return cached;
    }

    const result = fn(...args);
    cache.set(key, result);
    return result;
  };
}

// Usage
const expensiveCalculation = createMemoizedFn(
  (a: number, b: number) => {
    console.log('Calculating...');
    return a * b + Math.random();
  },
  (a, b) => `${a}-${b}`
);

```

Callback Memoization

Stable Callbacks

```

import { signal } from '@philjs/core';

function useCallback<T extends (...args: any[]) => any>(
  callback: T,
  deps: any[]
): T {
  let memoizedCallback = callback;
  let prevDeps = deps;

  return ((...args: any[])) => {
    const depsChanged = deps.some((dep, i) => dep !== prevDeps[i]);

    if (depsChanged) {
      memoizedCallback = callback;
      prevDeps = deps;
    }

    return memoizedCallback(...args);
  } as T;
}

// Usage
function TodoList() {
  const todos = signal<Todo[]>([]);

  // Stable callback reference
  const handleToggle = useCallback(
    (id: number) => {
      todos.set(
        todos().map(todo =>
          todo.id === id
            ? { ...todo, completed: !todo.completed }
            : todo
        )
      );
    },
    [todos]
  );

  return (
    <div>
      {todos().map(todo => (
        <TodoItem
          key={todo.id}
          todo={todo}
          onToggle={handleToggle}
        />
      ))}
    </div>
  );
}

```

When to Memoize

Expensive Computations

```

// ⚡ Memoize expensive calculations
const processedData = memo(() => {
  return largeDataset().map(item => {
    // Complex transformation
    return expensiveTransform(item);
  });
});

// ⚡ Don't memoize simple operations
const doubled = memo(() => count() * 2); // Overkill

```

Derived State

```

// ⚡ Memoize derived state
const filteredItems = memo(() =>
  items().filter(item => item.category === selectedCategory())
);

const sortedItems = memo(() =>
  [...filteredItems()].sort((a, b) => a.name.localeCompare(b.name))
);

// ⚡ Don't compute derived state on every render
function Component() {
  // Recalculated every render!
  const sorted = [...items()].sort(...);
  return <div>{sorted.map(...)}</div>;
}

```

Component Re-renders

```
// ⚠ Memoize components that rarely change
const Header = memoComponent(() => {
  return <header>{/* ... */}</header>;
});

// ⚠ Don't memoize components that change frequently
const Counter = memoComponent(() => {
  const count = signal(0);
  return <div>{count()}</div>; // Changes every click
});
```

Best Practices

Measure Before Memoizing

```
// ⚠ Profile to identify bottlenecks first
console.time('calculation');
const result = expensiveOperation();
console.timeEnd('calculation');

// If slow, then memoize
const memoized = memo(() => expensiveOperation());

// ⚠ Don't memoize everything blindly
```

Avoid Premature Optimization

```
// ⚠ Start simple, optimize when needed
function Component() {
  const filtered = items().filter(i => i.active);
  return <List items={filtered} />;
}

// ⚠ Don't over-engineer from start
function Component() {
  const filtered = memo(() =>
    memo(() =>
      items().filter(i => i.active)
    )()
  ); // Too much!
}
```

Use Proper Dependencies

```
// ⚠ Include all dependencies
const computed = memo(() => {
  return a() + b() + c();
});

// ⚠ Missing dependencies (stale data)
const computed = memo(() => {
  return a() + b(); // Missing c()
});
```

Clear Memoization When Needed

```
// ⚠ Reset memoization cache when data changes
const cache = new Map();

function clearCache() {
  cache.clear();
}

// Call when data is invalidated
onDataUpdate(() => {
  clearCache();
});
```

Memoize at Right Level

```
// ⚡ Memoize at component level
const MemoizedCard = memoComponent(Card);

function List() {
  return items().map(item => (
    <MemoizedCard key={item.id} item={item} />
  ));
}

// ⚡ Memoize inside render (recreated every time)
function List() {
  return items().map(item => {
    const MemoizedCard = memoComponent(Card); // Wrong!
    return <MemoizedCard key={item.id} item={item} />;
  });
}
```

Summary

You've learned:

- Computed values with memo
- Component memoization
- Selector patterns for derived state
- Advanced memoization techniques
- Callback memoization
- When and when not to memoize
- Best practices for memoization

Proper memoization prevents unnecessary work and improves performance!

Next: [Virtual Scrolling → Efficiently render large lists](#)

Memory Management

Prevent memory leaks and optimize memory usage in PhilJS applications.

What You'll Learn

- Memory leak detection
- Effect cleanup
- Event listener cleanup
- Observer cleanup
- Memory profiling
- Best practices

Common Memory Leaks

Event Listeners

```
import { effect } from '@philjs/core';

// ⚡ Memory Leak - Listener never removed
function BadComponent() {
  effect(() => {
    window.addEventListener('scroll', handleScroll);
    // Missing cleanup!
  });
}

// ⚡ Proper cleanup
function GoodComponent() {
  effect(() => {
    const handleScroll = () => {
      console.log(window.scrollY);
    };

    window.addEventListener('scroll', handleScroll);

    return () => {
      window.removeEventListener('scroll', handleScroll);
    };
  });
}
```

Timers and Intervals

```
// ⚠ Memory Leak - timer never cleared
function BadTimer() {
  effect(() => {
    setInterval(() => {
      console.log('tick');
    }, 1000);
    // Runs forever!
  });
}

// ⚡ Proper cleanup
function GoodTimer() {
  effect(() => {
    const id = setInterval(() => {
      console.log('tick');
    }, 1000);

    return () => clearInterval(id);
  });
}
```

Observers

```
// ⚠ Observer never disconnected
function BadObserver() {
  let ref: HTMLElement | undefined;

  effect(() => {
    if (!ref) return;

    const observer = new IntersectionObserver((entries) => {
      // Handle intersection
    });

    observer.observe(ref);
    // Never disconnected!
  });
}

// ⚡ Proper cleanup
function GoodObserver() {
  let ref: HTMLElement | undefined;

  effect(() => {
    if (!ref) return;

    const observer = new IntersectionObserver((entries) => {
      // Handle intersection
    });

    observer.observe(ref);

    return () => observer.disconnect();
  });
}
```

Effect Cleanup

Cleanup Pattern

```
import { effect, signal } from '@philjs/core';

function ComponentWithCleanup() {
  const isActive = signal(true);

  effect(() => {
    if (!isActive()) return;

    // Setup
    const subscription = subscribeToData((data) => {
      handleData(data);
    });

    const timer = setInterval(() => {
      checkStatus();
    }, 5000);
  });

  // Cleanup
  return () => {
    subscription.unsubscribe();
    clearInterval(timer);
  };
}
```

Conditional Cleanup

```
function ConditionalCleanup() {
  const isEnabled = signal(true);

  effect(() => {
    if (!isEnabled()) return;

    const ws = new WebSocket('wss://api.example.com');

    ws.onmessage = (event) => {
      handleMessage(event.data);
    };
  });

  // Only cleanup if WebSocket was created
  return () => {
    if (ws.readyState === WebSocket.OPEN) {
      ws.close();
    }
  };
}
```

Preventing Closures Leaks

Avoid Capturing Large Objects

```
// ⚡ Captures entire Large object
function BadClosure() {
  const largeData = signal({ /* huge object */ });

  effect(() => {
    // Captures entire LargeData
    document.addEventListener('click', () => {
      console.log(largeData().id); // Only needs ID
    });
  });
}

// ⚡ Extract only what's needed
function GoodClosure() {
  const largeData = signal({ /* huge object */ });
  const dataId = memo(() => largeData().id);

  effect(() => {
    const id = dataId(); // Only capture ID

    const handler = () => {
      console.log(id);
    };

    document.addEventListener('click', handler);

    return () => document.removeEventListener('click', handler);
  });
}
```

WeakMap for Caching

```
// ⚡ WeakMap allows garbage collection
const cache = new WeakMap<object, any>();

function cacheResult(obj: object, result: any) {
  cache.set(obj, result);
}

function getCached(obj: object) {
  return cache.get(obj);
}

// When obj is garbage collected, cache entry is automatically removed
```

Detaching DOM Elements

Remove References

```

function CleanComponent() {
  let element: HTMLElement | null = null;

  effect(() => {
    element = document.getElementById('my-element');

    return () => {
      // Remove reference to allow garbage collection
      element = null;
    };
  });
}

```

Clear Large Arrays

```

// ⚡ Large array stays in memory
function BadList() {
  const items = signal(Array(10000).fill(0).map((_, i) => ({
    id: i,
    data: new Array(1000).fill(i)
  )));

  // Items never cleared
}

// ⚡ Clear when no longer needed
function GoodList() {
  const items = signal([]);

  const loadItems = () => {
    items.set(generateLargeList());
  };

  const clearItems = () => {
    items.set([]); // Free memory
  };

  return (
    <div>
      <button onClick={loadItems}>Load</button>
      <button onClick={clearItems}>Clear</button>
    </div>
  );
}

```

Memory Profiling

Detect Memory Leaks

```

function measureMemory() {
  if ('memory' in performance) {
    const memory = (performance as any).memory;

    console.log({
      usedJSHeapSize: (memory.usedJSHeapSize / 1048576).toFixed(2) + ' MB',
      totalJSHeapSize: (memory.totalJSHeapSize / 1048576).toFixed(2) + ' MB',
      jsHeapSizeLimit: (memory.jsHeapSizeLimit / 1048576).toFixed(2) + ' MB'
    });
  }
}

// Call periodically to track memory usage
setInterval(measureMemory, 5000);

```

Performance Observer for Memory

```

import { effect } from '@philjs/core';

function trackMemoryUsage() {
  effect(() => {
    if ('memory' in performance) {
      const observer = new PerformanceObserver((list) => {
        for (const entry of list.getEntries()) {
          console.log('Memory:', entry);
        }
      });

      observer.observe({ entryTypes: ['measure'] });

      return () => observer.disconnect();
    }
  });
}

```

Best Practices

Use Weak References

```
// ⚡ WeakSet for object tracking
const tracked = new WeakSet<object>();

function trackObject(obj: object) {
  tracked.add(obj);
}

function isTracked(obj: object) {
  return tracked.has(obj);
}

// Objects can be garbage collected even if in WeakSet
```

Limit Cache Size

```
class LRUCache<K, V> {
  private cache = new Map<K, V>();
  private maxSize: number;

  constructor(maxSize: number) {
    this.maxSize = maxSize;
  }

  set(key: K, value: V) {
    // Remove oldest if at capacity
    if (this.cache.size >= this.maxSize) {
      const firstKey = this.cache.keys().next().value;
      this.cache.delete(firstKey);
    }

    this.cache.set(key, value);
  }

  get(key: K): V | undefined {
    const value = this.cache.get(key);
    if (value !== undefined) {
      // Move to end (mark as recently used)
      this.cache.delete(key);
      this.cache.set(key, value);
    }
    return value;
  }

  clear() {
    this.cache.clear();
  }
}

// Usage
const cache = new LRUCache<string, any>(100); // Max 100 items
```

Avoid Global Variables

```
// ⚡ Global variables prevent garbage collection
window.largeDataStore = {
  items: new Array(10000).fill(0)
};

// ⚡ Use local scope
function Component() {
  const localData = signal([]);
  // Garbage collected when component unmounts
}
```

Clean Up Subscriptions

```
// ⚡ Subscription cleanup pattern
function useSubscription<T>(subscribe: (callback: (data: T) => void) => () => void) {
  effect(() => {
    const unsubscribe = subscribe((data) => {
      handleData(data);
    });
  });

  return () => unsubscribe();
});

// Usage
useSubscription((callback) => {
  return eventBus.on('data', callback);
});
```

Nullify Large Objects

```
function processData() {
  let largeObject = {
    data: new Array(1000000).fill(0)
  };

  // Use the data
  processData(largeObject.data);

  // Nullify to allow garbage collection
  largeObject = null as any;
}
```

Debugging Memory Issues

Chrome DevTools Memory Profiler

```
// Take heap snapshots
function Component() {
  // 1. Take snapshot before action
  // 2. Perform action
  // 3. Take snapshot after
  // 4. Compare to find leaks

  const handleAction = () => {
    // Action that might leak
    for (let i = 0; i < 1000; i++) {
      createHeavyObject();
    }
  };

  return <button onClick={handleAction}>Test</button>;
}
```

Memory Leak Test Pattern

```
function testForMemoryLeak() {
  const initialMemory = (performance as any).memory?.usedJSHeapSize;

  // Perform action many times
  for (let i = 0; i < 100; i++) {
    const component = createComponent();
    component.mount();
    component.unmount();
  }

  // Force garbage collection (in Chrome with flag)
  if ((window as any).gc) {
    (window as any).gc();
  }

  const finalMemory = (performance as any).memory?.usedJSHeapSize;
  const leaked = finalMemory - initialMemory;

  if (leaked > 100000) { // 1MB threshold
    console.warn('Possible memory leak:', leaked / 1048576, 'MB');
  }
}
```

Summary

You've learned:

- Common memory leak patterns □ Effect cleanup patterns □ Event listener management □ Observer cleanup □ Preventing closure leaks □ Memory profiling techniques □ Best practices for memory management

Proper memory management ensures app stability!

Next: [Profiling](#) → Identify performance bottlenecks

Profiling

Identify and fix performance bottlenecks using profiling tools.

What You'll Learn

- Chrome DevTools profiling
- Performance API
- Custom profiling
- Flame graphs
- Network profiling

- Best practices

Chrome DevTools

Performance Panel

```
// Wrap code to profile
performance.mark('component-start');

// Component render logic
function MyComponent() {
  const data = signal(processData());
  return <div>{data()}</div>;
}

performance.mark('component-end');
performance.measure('component-render', 'component-start', 'component-end');

// View in DevTools Performance tab
```

CPU Profiling

```
// Start profiling
console.profile('Heavy Operation');

// Code to profile
for (let i = 0; i < 1000000; i++) {
  heavyCalculation(i);
}

// Stop profiling
console.profileEnd('Heavy Operation');

// View in DevTools Profiler tab
```

Memory Profiling

```
// Take heap snapshot before
const before = (performance as any).memory?.usedJSHeapSize;

// Operation to profile
createManyComponents();

// Take heap snapshot after
const after = (performance as any).memory?.usedJSHeapSize;

console.log(`Memory used: ${((after - before) / 1048576).toFixed(2)} MB`);
```

Performance API

User Timing API

```
function measureOperation(name: string, operation: () => void) {
  performance.mark(`#${name}-start`);

  operation();

  performance.mark(`#${name}-end`);
  performance.measure(name, `#${name}-start`, `#${name}-end`);

  const measure = performance.getEntriesByName(name)[0];
  console.log(`#${name}: ${measure.duration.toFixed(2)}ms`);

  // Cleanup
  performance.clearMarks(`#${name}-start`);
  performance.clearMarks(`#${name}-end`);
  performance.clearMeasures(name);
}

// Usage
measureOperation('data-processing', () => {
  processLargeDataset();
});
```

Performance Observer

```

import { effect } from '@philjs/core';

function usePerformanceObserver() {
  effect(() => {
    const observer = new PerformanceObserver((list) => {
      for (const entry of list.getEntries()) {
        if (entry.entryType === 'measure') {
          console.log(`#${entry.name}; ${entry.duration.toFixed(2)}ms`);

          // Send to analytics
          analytics.track('performance', {
            metric: entry.name,
            duration: entry.duration
          });
        }
      }
    });
  });

  observer.observe({ entryTypes: ['measure', 'navigation', 'resource'] });

  return () => observer.disconnect();
}

```

Custom Profiling

Component Render Profiling

```

function withPerformanceTracking<P>(
  Component: (props: P) => JSX.Element,
  componentName: string
) {
  return (props: P) => {
    const renderStart = performance.now();

    const result = Component(props);

    const renderEnd = performance.now();
    const duration = renderEnd - renderStart;

    if (duration > 16) { // Slower than 60fps
      console.warn(
        `${componentName} slow render: ${duration.toFixed(2)}ms`
      );
    }

    return result;
  };
}

// Usage
const ProfiledDashboard = withPerformanceTracking(
  Dashboard,
  'Dashboard'
);

```

Effect Profiling

```

function profiledEffect(fn: () => void | ((() => void), name: string) {
  return effect(() => {
    performance.mark(`effect-${name}-start`);

    const cleanup = fn();

    performance.mark(`effect-${name}-end`);
    performance.measure(
      `effect-${name}`,
      `effect-${name}-start`,
      `effect-${name}-end`
    );

    return cleanup;
  });
}

// Usage
profiledEffect(() => {
  const subscription = subscribeToData();
  return () => subscription.unsubscribe();
}, 'data-subscription');

```

Function Timing

```

function timed<T extends (...args: any[]) => any>(
  fn: T,
  name?: string
): T {
  return (...args: any[]) => {
    const fName = name || fn.name || 'anonymous';
    const start = performance.now();

    const result = fn(...args);

    const end = performance.now();
    console.log(`${fName}: ${((end - start).toFixed(2))}ms`);

    return result;
  } as T;
}

// Usage
const processData = timed((data: any[]) => {
  return data.map(transform);
}, 'processData');

```

Flame Graphs

Generate Flame Graph Data

```

interface FlameNode {
  name: string;
  value: number;
  children: FlameNode[];
}

class Profiler {
  private stack: Array<{ name: string; start: number }> = [];
  private root: FlameNode = { name: 'root', value: 0, children: [] };

  start(name: string) {
    this.stack.push({ name, start: performance.now() });
  }

  end() {
    const entry = this.stack.pop();
    if (!entry) return;

    const duration = performance.now() - entry.start;

    // Add to flame graph data
    this.addNode(entry.name, duration);
  }

  private addNode(name: string, duration: number) {
    let current = this.root;

    for (const { name: stackName } of this.stack) {
      const child = current.children.find(c => c.name === stackName);
      if (child) {
        current = child;
      }
    }

    current.children.push({ name, value: duration, children: [] });
  }

  getFlameGraph(): FlameNode {
    return this.root;
  }
}

// Usage
const profiler = new Profiler();

profiler.start('fetchData');
await fetchData();
profiler.end();

profiler.start('processData');
processData();
profiler.end();

const flameData = profiler.getFlameGraph();

```

Network Profiling

Track API Calls

```

function profiledFetch(url: string, options?: RequestInit) {
  const start = performance.now();

  return fetch(url, options)
    .then(response => {
      const end = performance.now();
      const duration = end - start;

      performance.measure(`fetch-${url}`, {
        start,
        duration
      });

      console.log(`API ${url}: ${duration.toFixed(2)}ms`);

      return response;
    });
}

// Usage
const data = await profiledFetch('/api/users');

```

Resource Timing

```

function analyzeResourceTiming() {
  const resources = performance.getEntriesByType('resource');

  const slowResources = resources
    .filter((resource: PerformanceResourceTiming) => resource.duration > 1000)
    .map((resource: PerformanceResourceTiming) => ({
      name: resource.name,
      duration: resource.duration,
      size: resource.transferSize,
      type: resource.initiatorType
    }));

  if (slowResources.length > 0) {
    console.warn(`Slow resources:`, slowResources);
  }

  return slowResources;
}

```

Real User Monitoring

Web Vitals Tracking

```

import { effect } from '@philjs/core';

function trackWebVitals() {
  effect(() => {
    // Largest Contentful Paint
    new PerformanceObserver((list) => {
      const entries = list.getEntries();
      const lcp = entries[entries.length - 1];

      analytics.track('web-vital', {
        metric: 'LCP',
        value: lcp.startTime,
        rating: lcp.startTime < 2500 ? 'good' : lcp.startTime < 4000 ? 'needs-improvement' : 'poor'
      });
    }).observe({ entryTypes: ['largest-contentful-paint'] });

    // First Input Delay
    new PerformanceObserver((list) => {
      const entry = list.getEntries()[0] as PerformanceEventTiming;
      const fid = entry.processingStart - entry.startTime;

      analytics.track('web-vital', {
        metric: 'FID',
        value: fid,
        rating: fid < 100 ? 'good' : fid < 300 ? 'needs-improvement' : 'poor'
      });
    }).observe({ entryTypes: ['first-input'] });

    // Cumulative Layout Shift
    let clsValue = 0;
    new PerformanceObserver((list) => {
      for (const entry of list.getEntries()) {
        if (!(entry as any).hadRecentInput) {
          clsValue += (entry as any).value;
        }
      }
    });

    analytics.track('web-vital', {
      metric: 'CLS',
      value: clsValue,
      rating: clsValue < 0.1 ? 'good' : clsValue < 0.25 ? 'needs-improvement' : 'poor'
    });
  }).observe({ entryTypes: ['layout-shift'] });
});
}

```

Custom Metrics

```

class PerformanceTracker {
  private metrics = new Map<string, number[]>();

  track(metric: string, value: number) {
    if (!this.metrics.has(metric)) {
      this.metrics.set(metric, []);
    }

    this.metrics.get(metric)!.push(value);
  }

  getStats(metric: string) {
    const values = this.metrics.get(metric) || [];

    if (values.length === 0) {
      return null;
    }

    const sorted = [...values].sort((a, b) => a - b);

    return {
      count: values.length,
      min: sorted[0],
      max: sorted[sorted.length - 1],
      avg: values.reduce((a, b) => a + b, 0) / values.length,
      p50: sorted[Math.floor(sorted.length * 0.5)],
      p90: sorted[Math.floor(sorted.length * 0.9)],
      p95: sorted[Math.floor(sorted.length * 0.95)],
      p99: sorted[Math.floor(sorted.length * 0.99)]
    };
  }

  report() {
    const report: Record<string, any> = {};

    this.metrics.forEach(_, metric) => {
      report[metric] = this.getStats(metric);
    });
  }

  return report;
}

// Usage
const tracker = new PerformanceTracker();

// Track component renders
tracker.track('component-render', 12.5);
tracker.track('component-render', 15.2);
tracker.track('component-render', 18.9);

// Get statistics
console.log(tracker.getStats('component-render'));
// { count: 3, min: 12.5, max: 18.9, avg: 15.5, p50: 15.2, ... }

```

Best Practices

Profile in Production Mode

```

# ⚠ Build and profile production build
npm run build
npm run preview

# ⚠ Don't profile development build (skewed results)
npm run dev

```

Use Appropriate Sample Size

```
// ⚡ Measure multiple times for accuracy
function accurateMeasure(fn: () => void, iterations = 100) {
  const times: number[] = [];

  for (let i = 0; i < iterations; i++) {
    const start = performance.now();
    fn();
    times.push(performance.now() - start);
  }

  return {
    min: Math.min(...times),
    max: Math.max(...times),
    avg: times.reduce((a, b) => a + b) / times.length
  };
}

// ⚡ Single measurement (unreliable)
const start = performance.now();
operation();
console.log(performance.now() - start);
```

Clean Up Performance Marks

```
// ⚡ Clean up marks and measures
function measureWithCleanup(name: string, fn: () => void) {
  try {
    performance.mark(` ${name}-start`);
    fn();
    performance.mark(` ${name}-end`);
    performance.measure(name, ` ${name}-start`, ` ${name}-end`);
  } finally {
    performance.clearMarks(` ${name}-start`);
    performance.clearMarks(` ${name}-end`);
    performance.clearMeasures(name);
  }
}

// ⚡ Leave marks (memory leak)
performance.mark('operation-start');
// ...
```

Focus on User-Centric Metrics

```
// ⚡ Measure what users experience
- Time to Interactive (TTI)
- First Input Delay (FID)
- Largest Contentful Paint (LCP)
- Cumulative Layout Shift (CLS)

// ⚡ Only technical metrics
- Bundle size
- Number of requests
- Server response time
```

Summary

You've learned:

□ Chrome DevTools profiling □ Performance API usage □ Custom profiling techniques □ Flame graph generation □ Network profiling □ Real user monitoring □ Best practices for profiling

Profiling identifies bottlenecks for targeted optimization!

Next: [Performance Budgets](#) → Set and enforce performance goals

Runtime Performance

Optimize application execution speed and responsiveness.

What You'll Learn

- Event optimization
- Rendering performance
- JavaScript optimization
- DOM operations
- Web Workers
- Best practices

Event Optimization

Debouncing

```
function debounce<T extends (...args: any[]) => any>(
  func: T,
  delay: number
): T {
  let timeoutId: any;

  return (...args: any[]) => {
    clearTimeout(timeoutId);
    timeoutId = setTimeout(() => func(...args), delay);
  } as T;
}

// Usage
function SearchInput() {
  const query = signal('');

  const handleSearch = debounce((value: string) => {
    // API call only after user stops typing
    searchAPI(value);
  }, 300);

  return (
    <input
      value={query()}
      onInput={(e) => {
        query.set(e.target.value);
        handleSearch(e.target.value);
      }}
    />
  );
}
```

Throttling

```
function throttle<T extends (...args: any[]) => any>(
  func: T,
  limit: number
): T {
  let inThrottle: boolean;

  return (...args: any[]) => {
    if (!inThrottle) {
      func(...args);
      inThrottle = true;
      setTimeout(() => (inThrottle = false), limit);
    }
  } as T;
}

// Usage
function ScrollHandler() {
  const handleScroll = throttle(() => {
    console.log('Scroll event', window.scrollY);
  }, 100);

  return <div onScroll={handleScroll}>{/* content */}</div>;
}
```

Passive Event Listeners

```
import { effect } from '@philjs/core';

function usePassiveScroll(callback: (e: Event) => void) {
  effect(() => {
    const handler = (e: Event) => callback(e);

    window.addEventListener('scroll', handler, { passive: true });

    return () => window.removeEventListener('scroll', handler);
  });
}

// Usage
usePassiveScroll((e) => {
  // Won't block scrolling
  updateScrollPosition(window.scrollY);
});
```

Rendering Performance

Batch Updates

```

import { batch } from '@philjs/core';

function updateMultipleValues() {
  const count = signal(0);
  const name = signal('');
  const active = signal(false);

  // ⚡ Triggers 3 re-renders
  count.set(10);
  name.set('Alice');
  active.set(true);

  // ⚡ Triggers 1 re-render
  batch(() => {
    count.set(10);
    name.set('Alice');
    active.set(true);
  });
}

```

Avoid Layout Thrashing

```

// ⚡ Layout thrashing (read-write-read-write)
function badLayoutUpdate() {
  const height1 = element1.offsetHeight; // Read
  element1.style.height = `${height1 * 2}px`; // Write

  const height2 = element2.offsetHeight; // Read (forces layout)
  element2.style.height = `${height2 * 2}px`; // Write
}

// ⚡ Batch reads, then batch writes
function goodLayoutUpdate() {
  // Batch all reads
  const height1 = element1.offsetHeight;
  const height2 = element2.offsetHeight;

  // Batch all writes
  element1.style.height = `${height1 * 2}px`;
  element2.style.height = `${height2 * 2}px`;
}

```

RequestAnimationFrame

```

function useRAF(callback: () => void) {
  effect(() => {
    let rafId: number;

    const loop = () => {
      callback();
      rafId = requestAnimationFrame(loop);
    };

    rafId = requestAnimationFrame(loop);

    return () => cancelAnimationFrame(rafId);
  });
}

// Usage
function AnimatedComponent() {
  const position = signal(0);

  useRAF(() => {
    position.set(position() + 1);
  });

  return <div style={{ transform: `translateX(${position()}px)` }} />;
}

```

JavaScript Optimization

Avoid Blocking Operations

```
// ⚡ Blocks main thread
function processLargeDataset(data: any[]) {
  return data.map(item => heavyTransformation(item));
}

// ⚡ Use Web Worker
const worker = new Worker('/worker.ts');

worker.postMessage({ data: largeDataset });

worker.onmessage = (e) => {
  const processed = e.data;
  updateUI(processed);
};
```

Optimize Loops

```
// ⚡ Slow Loop
for (let i = 0; i < array.length; i++) {
  // array.length calculated each iteration
}

// ⚡ Cache Length
const len = array.length;
for (let i = 0; i < len; i++) {
  // Faster
}

// ⚡ Or use for...of
for (const item of array) {
  // Clean and fast
}
```

Early Returns

```
// ⚡ Early returns reduce computation
function processItem(item: any) {
  if (!item) return null;
  if (item.skip) return null;
  if (!item.valid) return null;

  // Heavy processing only if needed
  return expensiveTransformation(item);
}
```

DOM Operations

Minimize DOM Access

```
// ⚡ Multiple DOM accesses
function updateList() {
  for (let i = 0; i < items.length; i++) {
    document.getElementById('list').innerHTML += `<li>${items[i]}</li>`;
  }
}

// ⚡ Build string, single DOM update
function updateList() {
  const html = items.map(item => `<li>${item}</li>`).join('');
  document.getElementById('list')!.innerHTML = html;
}
```

Use Document Fragments

```
function createList(items: string[]) {
  const fragment = document.createDocumentFragment();

  items.forEach(item => {
    const li = document.createElement('li');
    li.textContent = item;
    fragment.appendChild(li);
  });

  // Single reflow
  document.getElementById('list')!.appendChild(fragment);
}
```

Avoid Inline Styles

```
// ⚡ Inline styles (slower)
<div style={{ width: '100px', height: '100px', backgroundColor: 'red' }} />

// ⚡ CSS classes (faster)
<div className="box box-red" />
```

Web Workers

Offload Heavy Tasks

```
// main thread
const worker = new Worker('/data-processor.worker.ts');

worker.postMessage({
  action: 'process',
  data: largeDataset
});

worker.onmessage = (e) => {
  if (e.data.type === 'progress') {
    updateProgress(e.data.percent);
  } else if (e.data.type === 'complete') {
    displayResults(e.data.results);
  }
};

// data-processor.worker.ts
self.onmessage = (e) => {
  if (e.data.action === 'process') {
    const results = [];
    const total = e.data.data.length;

    e.data.data.forEach((item, index) => {
      results.push(processItem(item));

      // Report progress
      if (index % 100 === 0) {
        self.postMessage({
          type: 'progress',
          percent: (index / total) * 100
        });
      }
    });

    self.postMessage({
      type: 'complete',
      results
    });
  }
};
```

Worker Pool

```

class WorkerPool {
  private workers: Worker[] = [];
  private queue: Array<{
    data: any;
    resolve: (value: any) => void;
    reject: (reason: any) => void;
  }> = [];

  constructor(workerScript: string, poolSize: number = 4) {
    for (let i = 0; i < poolSize; i++) {
      const worker = new Worker(workerScript);

      worker.onmessage = (e) => {
        this.handleWorkerComplete(worker, e.data);
      };
    }

    this.workers.push(worker);
  }
}

execute(data: any): Promise<any> {
  return new Promise((resolve, reject) => {
    const freeWorker = this.workers.find(w => !(w as any).busy);

    if (freeWorker) {
      this.runTask(freeWorker, data, resolve, reject);
    } else {
      this.queue.push({ data, resolve, reject });
    }
  });
}

private runTask(
  worker: Worker,
  data: any,
  resolve: (value: any) => void,
  reject: (reason: any) => void
) {
  (worker as any).busy = true;
  (worker as any).resolve = resolve;
  (worker as any).reject = reject;
  worker.postMessage(data);
}

private handleWorkerComplete(worker: Worker, result: any) {
  (worker as any).resolve(result);
  (worker as any).busy = false;

  // Process queue
  if (this.queue.length > 0) {
    const { data, resolve, reject } = this.queue.shift();
    this.runTask(worker, data, resolve, reject);
  }
}
}

// Usage
const pool = new WorkerPool('/processor.worker.ts', 4);

async function processData(items: any[]) {
  const results = await Promise.all(
    items.map(item => pool.execute(item))
  );
  return results;
}

```

Best Practices

Use Will-Change Sparingly

```

/* ⚡ Use for elements that will animate */
.animated-element {
  will-change: transform;
}

/* ⚡ Don't use on everything */
* {
  will-change: transform, opacity;
}

```

Optimize Images

```
// ⚡ Use appropriate image formats
<picture>
  <source srcset="image.avif" type="image/avif" />
  <source srcset="image.webp" type="image/webp" />
  
</picture>

// ⚡ Lazy Load images


// ⚡ Use responsive images
<img
  srcset="small.jpg 400w, medium.jpg 800w, large.jpg 1200w"
  sizes="(max-width: 600px) 100vw, 50vw"
/>
```

Avoid Memory Leaks

```
// ⚡ Clean up event listeners
effect(() => {
  const handler = () => console.log('clicked');
  window.addEventListener('click', handler);

  return () => window.removeEventListener('click', handler);
});

// ⚡ Clear timers
effect(() => {
  const id = setInterval(() => {
    console.log('tick');
  }, 1000);

  return () => clearInterval(id);
});
```

Use CSS Containment

```
/* Optimize rendering performance */
.independent-component {
  contain: layout style paint;
}

.card-grid {
  contain: layout style;
}
```

Summary

You've learned:

- Event optimization (debounce, throttle)
- Rendering performance optimization
- JavaScript execution optimization
- Efficient DOM operations
- Web Workers for heavy tasks
- Best practices for runtime performance

Optimize runtime for smooth, responsive applications!

Next: [Memory Management → Prevent memory leaks](#)

Runtime Performance

Optimize runtime performance for smooth, responsive applications.

Measuring Performance

Performance API

```
const start = performance.now();

// Do work
heavyComputation();

const end = performance.now();
console.log(`Took ${end - start}ms`);
```

React DevTools Profiler

```
import { enableDevTools } from '@philjs/devtools';

if (process.env.NODE_ENV === 'development') {
  enableDevTools();
}
```

Optimize Renders

Use Memos

```
import { signal, memo } from '@philjs/core';

const items = signal([1, 2, 3, 4, 5]);

// Only recalculates when items changes
const total = memo(() => items().reduce((a, b) => a + b, 0));
```

Batch Updates

```
import { batch } from '@philjs/core';

const updateMultiple = () => {
  batch(() => {
    firstName.set('John');
    lastName.set('Doe');
    age.set(30);
  }); // Single render
};
```

Avoid Expensive Operations

Debounce

```
import { signal, effect } from '@philjs/core';

const query = signal('');

effect(() => {
  const q = query();

  const timer = setTimeout(() => {
    search(q); // Debounced search
  }, 300);

  onCleanup(() => clearTimeout(timer));
});
```

Best Practices

□ Do: Profile First

```
// ☀ Good - measure before optimizing
console.time('render');
render();
console.timeEnd('render');
```

□ Don't: Premature Optimize

```
// ☀ Bad - optimizing without profiling
// May not be the bottleneck
```

Next Steps

- [Memoization](#) - Memoization patterns
- [Bundle Size](#) - Reduce bundle
- [Web Vitals](#) - Monitor vitals

□ **Tip:** Use the browser's Performance tab to identify bottlenecks.

□□ **Warning:** Measure before optimizing—premature optimization wastes time.

□□ **Note:** PhilJS's fine-grained reactivity is fast by default.

Web Vitals

Monitor Core Web Vitals to ensure great user experience.

Core Web Vitals

LCP (Largest Contentful Paint)

```

import { signal, effect } from '@philjs/core';

export default function LCPMonitor() {
  const lcp = signal(0);

  effect(() => {
    const observer = new PerformanceObserver((list) => {
      const entries = list.getEntries();
      const lastEntry = entries[entries.length - 1];
      lcp.set(lastEntry.renderTime || lastEntry.loadTime);
    });
  });

  observer.observe({ type: 'largest-contentful-paint', buffered: true });
  onCleanup(() => observer.disconnect());
}

return <div>LCP: {lcp().toFixed(2)}ms</div>;
}

```

FID (First Input Delay)

```

const fid = signal(0);

effect(() => {
  const observer = new PerformanceObserver((list) => {
    const entry = list.getEntries()[0];
    fid.set(entry.processingStart - entry.startTime);
  });

  observer.observe({ type: 'first-input', buffered: true });
});

```

CLS (Cumulative Layout Shift)

```

const cls = signal(0);

effect(() => {
  let clsValue = 0;

  const observer = new PerformanceObserver((list) => {
    for (const entry of list.getEntries()) {
      if (!entry.hadRecentInput) {
        clsValue += entry.value;
        cls.set(clsValue);
      }
    }
  });
  observer.observe({ type: 'layout-shift', buffered: true });
});

```

Monitoring

Web Vitals Library

```

import { getCLS, getFID, getLCP } from 'web-vitals';

export default function VitalsTracker() {
  effect(() => {
    getCLS(console.log);
    getFID(console.log);
    getLCP(console.log);
  });

  return null;
}

```

Best Practices

Do: Monitor in Production

```

// 🚨 Good - track real user metrics
if (typeof window !== 'undefined') {
  getCLS(sendToAnalytics);
  getFID(sendToAnalytics);
  getLCP(sendToAnalytics);
}

```

Next Steps

- [Performance](#) - Performance guide
- [Budgets](#) - Set budgets

- **Tip:** Good Core Web Vitals improve SEO rankings.
- **Warning:** Monitor vitals in production, not just development.
- **Note:** Target: LCP < 2.5s, FID < 100ms, CLS < 0.1

Image Optimization

Optimize images for faster load times and better performance.

Responsive Images

Picture Element

```
export default function ResponsiveImage({ src, alt }) {
  return (
    <picture>
      <source
        srcSet={`${src}?w=320 320w, ${src}?w=640 640w, ${src}?w=1280 1280w`}
        sizes="(max-width: 640px) 320px, (max-width: 1280px) 640px, 1280px"
      />
      <img src={src} alt={alt} loading="lazy" />
    </picture>
  );
}
```

Lazy Loading

Native Lazy Loading

```

```

Intersection Observer

```
import { signal, effect } from '@philjs/core';

export default function LazyImage({ src, alt }) {
  const isVisible = signal(false);
  let imgRef: HTMLImageElement;

  effect(() => {
    if (!imgRef) return;

    const observer = new IntersectionObserver(([entry]) => {
      if (entry.isIntersecting) {
        isVisible.set(true);
        observer.disconnect();
      }
    });
    observer.observe(imgRef);
    onCleanup(() => observer.disconnect());
  });

  return (
    <img
      ref={imgRef}
      src={isVisible() ? src : 'data:image/svg+xml,...'}
      alt={alt}
    />
  );
}
```

Next Steps

- [Performance](#) - Performance guide
- [Lazy Loading](#) - Lazy patterns

-
- **Tip:** Always use lazy loading for below-the-fold images.
 - **Warning:** Optimize images before uploading—don't rely on runtime optimization.
 - **Note:** Modern formats like WebP and AVIF offer better compression than JPEG.

Server-Side Performance

Optimize server-side rendering for faster page loads.

SSR Optimization

Streaming SSR

```
export const loader = createDataLoader(async () => {
  return {
    critical: await fetchCritical(),
    deferred: fetchDeferred() // Streams later
  };
});
```

Edge Rendering

```
export const config = {
  runtime: 'edge'
};

export default function EdgePage() {
  return <div>Rendered at the edge!</div>;
}
```

Caching

Cache Headers

```
export const config = {
  cache: {
    maxAge: 3600,
    staleWhileRevalidate: 86400
  }
};
```

Best Practices

Do: Cache Static Content

```
// ☀ Good - cached at edge
export const config = { cache: { maxAge: 3600 } };
```

Next Steps

- [SSR](#) - Server-side rendering
- [Caching](#) - Caching strategies

💡 **Tip:** Use edge rendering for global low-latency responses.

⚠ **Note:** Streaming SSR sends HTML progressively for faster TTFB.

Virtual Scrolling

Efficiently render large lists by only displaying visible items.

What You'll Learn

- Virtual scrolling basics
- Fixed height items
- Variable height items
- Bidirectional scrolling
- Performance optimization
- Best practices

Basic Virtual Scrolling

Fixed Height Items

```

import { signal, effect } from '@philjs/core';

interface VirtualScrollerProps<T> {
    items: T[];
    itemHeight: number;
    containerHeight: number;
    renderItem: (item: T, index: number) => JSX.Element;
}

export function VirtualScroller<T>({
    items,
    itemHeight,
    containerHeight,
    renderItem
}: VirtualScrollerProps<T>) {
    const scrollTop = signal(0);

    const visibleCount = Math.ceil(containerHeight / itemHeight);
    const startIndex = () => Math.floor(scrollTop() / itemHeight);
    const endIndex = () => Math.min(
        startIndex() + visibleCount + 1,
        items.length
    );

    const totalHeight = items.length * itemHeight;
    const offsetY = () => startIndex() * itemHeight;

    const visibleItems = () =>
        items.slice(startIndex(), endIndex());

    const handleScroll = (e: Event) => {
        const target = e.target as HTMLElement;
        scrollTop.set(target.scrollTop);
    };

    return (
        <div
            style={{
                height: `${containerHeight}px`,
                overflow: 'auto',
                position: 'relative'
            }}
            onScroll={handleScroll}
        >
            <div style={{ height: `${totalHeight}px` }}>
                <div
                    style={{
                        transform: `translateY(${offsetY()}px)`,
                        willChange: 'transform'
                    }}
                >
                    {visibleItems().map((item, i) =>
                        renderItem(item, startIndex() + i)
                    )}
                </div>
            </div>
        </div>
    );
}

// Usage
function App() {
    const items = Array.from({ length: 10000 }, (_, i) => ({
        id: i,
        name: `Item ${i}`
    }));
}

return (
    <VirtualScroller
        items={items}
        itemHeight={50}
        containerHeight={600}
        renderItem={(item) => (
            <div style={{ height: '50px', padding: '10px', borderBottom: '1px solid #eee' }}>
                {item.name}
            </div>
        )}
    />
);
}

```

With Overscan

```

interface VirtualScrollerWithOverscanProps<T> {
  items: T[];
  itemHeight: number;
  containerHeight: number;
  overscan?: number; // Number of items to render outside viewport
  renderItem: (item: T, index: number) => JSX.Element;
}

export function VirtualScrollerWithOverscan<T>({
  items,
  itemHeight,
  containerHeight,
  overscan = 3,
  renderItem
}: VirtualScrollerWithOverscanProps<T>) {
  const scrollTop = signal(0);

  const visibleCount = Math.ceil(containerHeight / itemHeight);

  const startIndex = () =>
    Math.max(0, Math.floor(scrollTop() / itemHeight) - overscan);

  const endIndex = () =>
    Math.min(
      Math.ceil(scrollTop() / itemHeight) + visibleCount + overscan,
      items.length
    );

  const visibleItems = () =>
    items.slice(startIndex(), endIndex());

  const offsetY = () => startIndex() * itemHeight;

  return (
    <div
      style={{
        height: `${containerHeight}px`,
        overflow: 'auto',
        position: 'relative'
      }}
      onScroll={(e) => scrollTop.set((e.target as HTMLElement).scrollTop)}
    >
      <div style={{ height: `${items.length * itemHeight}px` }}>
        <div style={{ transform: `translate(${offsetY()}px)` }}>
          {visibleItems().map((item, i) => (
            <div key={startIndex() + i}>
              {renderItem(item, startIndex() + i)}
            </div>
          )));
        </div>
      </div>
    </div>
  );
}

```

Variable Height Items

Dynamic Height Virtual Scroller

```

import { signal, memo } from '@philjs/core';

interface DynamicVirtualScrollerProps<T> {
  items: T[];
  estimatedItemHeight: number;
  containerHeight: number;
  renderItem: (item: T, index: number) => JSX.Element;
}

export function DynamicVirtualScroller<T>({
  items,
  estimatedItemHeight,
  containerHeight,
  renderItem
}: DynamicVirtualScrollerProps<T>) {
  const scrollTop = signal(0);
  const itemHeights = signal<Map<number, number>>(new Map());

  const getItemHeight = (index: number) => {
    return itemHeights().get(index) ?? estimatedItemHeight;
  };

  const getItemOffset = (index: number) => {
    let offset = 0;
    for (let i = 0; i < index; i++) {
      offset += getItemHeight(i);
    }
    return offset;
  };
}

```

```

};

const getTotalHeight = memo(() => {
  let height = 0;
  for (let i = 0; i < items.length; i++) {
    height += getItemHeight(i);
  }
  return height;
});

const getVisibleRange = () => {
  let start = 0;
  let end = items.length - 1;

  // Find start index
  let offset = 0;
  for (let i = 0; i < items.length; i++) {
    if (offset + getItemHeight(i) > scrollTop()) {
      start = i;
      break;
    }
    offset += getItemHeight(i);
  }

  // Find end index
  offset = getItemOffset(start);
  for (let i = start; i < items.length; i++) {
    if (offset > scrollTop() + containerHeight) {
      end = i;
      break;
    }
    offset += getItemHeight(i);
  }

  return { start, end };
};

const visibleRange = memo(() => getVisibleRange());

const measureItem = (index: number, element: HTMLElement) => {
  const height = element.getBoundingClientRect().height;
  if (height !== itemHeights.get(index)) {
    const updated = new Map(itemHeights());
    updated.set(index, height);
    itemHeights.set(updated);
  }
};

return (
  <div
    style={{
      height: `${containerHeight}px`,
      overflow: 'auto',
      position: 'relative'
    }}
    onScroll={(e) => scrollTop.set((e.target as HTMLElement).scrollTop)}
  >
  <div style={{ height: `${getTotalHeight()}px`, position: 'relative' }}>
    {items.slice(visibleRange().start, visibleRange().end + 1).map((item, i) => {
      const index = visibleRange().start + i;
      return (
        <div
          key={index}
          ref={(el) => el && measureItem(index, el)}
          style={{
            position: 'absolute',
            top: `${getItemOffset(index)}px`,
            width: '100%'
          }}
        >
          {renderItem(item, index)}
        </div>
      );
    ))}
  </div>
</div>
);
}

```

Bidirectional Scrolling

Grid Virtual Scroller

```

interface GridVirtualScrollerProps<T> {
  items: T[];
  itemWidth: number;
  itemHeight: number;
}

```

```

containerWidth: number;
containerHeight: number;
renderItem: (item: T, index: number) => JSX.Element;
}

export function GridVirtualScroller<T>({
  items,
  itemWidth,
  itemHeight,
  containerWidth,
  containerHeight,
  renderItem
}: GridVirtualScrollerProps<T>) {
  const scrollTop = signal(0);
  const scrollLeft = signal(0);

  const columnsPerRow = Math.floor(containerWidth / itemWidth);
  const totalRows = Math.ceil(items.length / columnsPerRow);

  const visibleRows = () => {
    const start = Math.floor(scrollTop() / itemHeight);
    const count = Math.ceil(containerHeight / itemHeight) + 1;
    return { start, count };
  };

  const visibleColumns = () => {
    const start = Math.floor(scrollLeft() / itemWidth);
    const count = Math.ceil(containerWidth / itemWidth) + 1;
    return { start, count };
  };

  const visibleItems = () => {
    const rows = visibleRows();
    const cols = visibleColumns();
    const items: Array<{ item: T; row: number; col: number; index: number }> = [];

    for (let row = rows.start; row < rows.start + rows.count; row++) {
      for (let col = cols.start; col < cols.start + cols.count; col++) {
        const index = row * columnsPerRow + col;
        if (index < items.length) {
          items.push({ item: items[index], row, col, index });
        }
      }
    }
  }

  return items;
};

return (
  <div
    style={{
      width: `${containerWidth}px`,
      height: `${containerHeight}px`,
      overflow: 'auto',
      position: 'relative'
    }}
    onScroll={(e) => {
      const target = e.target as HTMLElement;
      scrollTop.set(target.scrollTop);
      scrollLeft.set(target.scrollLeft);
    }}
  >
  <div
    style={{
      width: `${columnsPerRow * itemWidth}px`,
      height: `${totalRows * itemHeight}px`,
      position: 'relative'
    }}
    >
      {visibleItems().map(({ item, row, col, index }) => (
        <div
          key={index}
          style={{
            position: 'absolute',
            left: `${(col * itemWidth)}px`,
            top: `${(row * itemHeight)}px`,
            width: `${itemWidth}px`,
            height: `${itemHeight}px`
          }}
        >
          {renderItem(item, index)}
        </div>
      ))}
    </div>
  </div>
);
}

```

Performance Optimization

Throttled Scroll

```
import { signal } from '@philjs/core';

function throttle<T extends (...args: any[]) => any>(
  func: T,
  delay: number
): T {
  let lastCall = 0;
  let timeoutId: any;

  return ((...args: any[]) => {
    const now = Date.now();

    if (now - lastCall >= delay) {
      lastCall = now;
      func(...args);
    } else {
      clearTimeout(timeoutId);
      timeoutId = setTimeout(() => {
        lastCall = Date.now();
        func(...args);
      }, delay - (now - lastCall));
    }
  }) as T;
}

export function OptimizedVirtualScroller<T>(props: VirtualScrollerProps<T>) {
  const scrollTop = signal(0);

  const handleScroll = throttle((scrollY: number) => {
    scrollTop.set(scrollY);
  }, 16); // ~60fps

  return (
    <div
      onScroll={(e) => handleScroll((e.target as HTMLElement).scrollTop)}
    >
      /* Virtual scroller content */
    </div>
  );
}
```

RequestAnimationFrame Optimization

```
export function RAFVirtualScroller<T>(props: VirtualScrollerProps<T>) {
  const scrollTop = signal(0);
  let rafId: number | null = null;

  const handleScroll = (e: Event) => {
    if (rafId === null) return;

    rafId = requestAnimationFrame(() => {
      scrollTop.set((e.target as HTMLElement).scrollTop);
      rafId = null;
    });
  };

  effect(() => {
    return () => {
      if (rafId !== null) {
        cancelAnimationFrame(rafId);
      }
    };
  });

  return (
    <div onScroll={handleScroll}>
      /* Virtual scroller content */
    </div>
  );
}
```

Infinite Scroll

Load More on Scroll

```

import { signal, effect } from '@philjs/core';

interface InfiniteScrollProps<T> {
  items: T[];
  loadMore: () => Promise<T[]>;
  hasMore: boolean;
  itemHeight: number;
  containerHeight: number;
  renderItem: (item: T, index: number) => JSX.Element;
}

export function InfiniteVirtualScroller<T>({{
  items,
  loadMore,
  hasMore,
  itemHeight,
  containerHeight,
  renderItem
}: InfiniteScrollProps<T>) {
  const allItems = signal<T[]>(items);
  const loading = signal(false);
  const scrollTop = signal(0);

  const visibleCount = Math.ceil(containerHeight / itemHeight);
  const startIndex = () => Math.floor(scrollTop() / itemHeight);
  const endIndex = () => Math.min(
    startIndex() + visibleCount,
    allItems().length
  );

  const handleScroll = async (e: Event) => {
    const target = e.target as HTMLElement;
    scrollTop.set(target.scrollTop);

    // Load more when near bottom
    const scrolledToBottom =
      target.scrollHeight - target.scrollTop - target.clientHeight < itemHeight * 5;

    if (scrolledToBottom && hasMore && !loading()) {
      loading.set(true);
      const newItems = await loadMore();
      allItems.set([...allItems(), ...newItems]);
      loading.set(false);
    }
  };
}

return (
  <div
    style={{
      height: `${containerHeight}px`,
      overflow: 'auto'
    }}
    onScroll={handleScroll}
  >
  <div style={{ height: `${allItems().length * itemHeight}px` }}>
    <div style={{ transform: `translateY(${startIndex()} * ${itemHeight}px)` }}>
      {allItems()
        .slice(startIndex(), endIndex())
        .map((item, i) => renderItem(item, startIndex() + i))}
    </div>
  </div>
  {loading() && (
    <div style={{ padding: '20px', textAlign: 'center' }}>
      Loading more...
    </div>
  )}
  </div>
);
}

```

Best Practices

Use Fixed Heights When Possible

```
// ⚡ Fixed height (better performance)
<VirtualScroller
  items={items}
  itemHeight={50}
  renderItem={(item) => (
    <div style={{ height: '50px' }}>{item.name}</div>
  )}
/>

// ⚡ Variable height (slower)
<DynamicVirtualScroller
  items={items}
  estimatedItemHeight={50}
  renderItem={(item) => (
    <div>{item.content}</div> // Unknown height
  )}
/>
```

Add Overscan for Smooth Scrolling

```
// ⚡ Render extra items for smooth scroll
<VirtualScroller
  overscan={3}
  {...props}
/>

// ⚡ No overscan (may show empty space during scroll)
<VirtualScroller
  overscan={0}
  {...props}
/>
```

Throttle Scroll Events

```
// ⚡ Throttle scroll updates
const handleScroll = throttle((scrollY) => {
  scrollTop.set(scrollY);
}, 16);

// ⚡ Update on every scroll event
const handleScroll = (e) => {
  scrollTop.set(e.target.scrollTop);
};
```

Use Stable Keys

```
// ⚡ Stable keys for items
{items.map(item) => (
  <div key={item.id}>{item.name}</div>
)}

// ⚡ Index as key (can cause issues)
{items.map((item, i) => (
  <div key={i}>{item.name}</div>
))}
```

Measure Heights Efficiently

```
// ⚡ Measure only once
const measureItem = (index: number, el: HTMLElement) => {
  if (!itemHeights().has(index)) {
    itemHeights().set(index, el.getBoundingClientRect().height);
  }
};

// ⚡ Measure on every render
const measureItem = (index: number, el: HTMLElement) => {
  itemHeights().set(index, el.getBoundingClientRect().height);
};
```

Summary

You've learned:

□ Basic virtual scrolling for fixed heights □ Variable height virtual scrolling □ Bidirectional (grid) virtual scrolling □ Performance optimization techniques □ Infinite scroll implementation □ Best practices for virtual scrolling

Virtual scrolling enables smooth rendering of massive lists!

Next: [Bundle Optimization → Reduce bundle size](#)

Best Practices Overview

Essential patterns and guidelines for building production-ready PhilJS applications.

Introduction

This section provides comprehensive best practices for PhilJS development. These guidelines are based on real-world experience, performance testing, and community feedback.

Core Principles

1. Explicit Over Implicit

PhilJS favors explicit code that's easy to understand and debug.

□ **Good:**

```
const count = signal(0);
const doubled = memo(() => count() * 2);

effect(() => {
  console.log('Count:', count());
});
```

□ **Avoid:**

```
// Relying on implicit behavior
let count = 0; // Not reactive!
const doubled = count * 2; // Not memoized!
```

2. Immutability

Always use immutable updates for signals.

□ **Good:**

```
const user = signal({ name: 'Alice', age: 30 });

user.set({
  ...user(),
  age: 31
});
```

□ **Avoid:**

```
user().age = 31; // Mutation doesn't trigger updates!
user.set(user()); // Same reference, no update!
```

3. Composition Over Complexity

Break complex logic into small, composable pieces.

□ **Good:**

```
function useAuth() {
  const user = signal<User | null>(null);
  const isAuthenticated = memo(() => user() !== null);

  const login = async (credentials: Credentials) => {
    const userData = await authService.login(credentials);
    user.set(userData);
  };

  return { user, isAuthenticated, login };
}

function usePermissions() {
  const { user } = useAuth();
  const canEdit = memo(() => user()?.role === 'admin');

  return { canEdit };
}
```

□ **Avoid:**

```
function useEverything() {
  // Giant function doing too much
  const user = signal(null);
  const posts = signal([]);
  const comments = signal([]);
  const likes = signal([]);
  // ... hundreds of lines
}
```

4. Type Safety

Leverage TypeScript for better development experience.

□ **Good:**

```

interface User {
  id: string;
  name: string;
  email: string;
}

const user = signal<User | null>(null);

function UserProfile({ userId }: { userId: string }) {
  // Type-safe throughout
}

```

□ **Avoid:**

```

const user = signal<any>(null); // Loses type safety
const data: any = fetchData(); // Loses type safety

```

5. Performance by Default

Write code that performs well without manual optimization.

□ **Good:**

```

const filtered = memo(() => {
  return items().filter(item => item.active);
});

const sorted = memo(() => {
  return filtered().sort((a, b) => a.name.localeCompare(b.name));
});

```

□ **Avoid:**

```

// Recomputing on every render
return items()
  .filter(item => item.active)
  .sort((a, b) => a.name.localeCompare(b.name));

```

Best Practice Categories

Component Patterns

Learn how to structure components effectively: - Presentational vs Container components - Composition patterns - Props design - Component lifecycle

→ [Component Patterns](#)

State Management

Master state management at all scales: - Local state patterns - Global state strategies - State composition - State machines

→ [State Management](#)

Performance

Optimize for speed and efficiency: - Memoization strategies - Batching updates - Lazy loading - Bundle optimization

→ [Performance](#)

Testing

Write maintainable, reliable tests: - Unit testing signals and effects - Component testing - Integration testing - E2E testing

→ [Testing](#)

Code Organization

Structure your project for scale: - Directory structure - File naming conventions - Module organization - Dependency management

→ [Code Organization](#)

Architecture

Design scalable applications: - Application structure - Layer separation - Dependency injection - Plugin architecture

→ [Architecture](#)

Security

Protect your application and users: - XSS prevention - CSRF protection - Authentication patterns - Authorization strategies

→ [Security](#)

Accessibility

Build inclusive applications: - Semantic HTML - ARIA attributes - Keyboard navigation - Screen reader support

→ [Accessibility](#)

Production

Deploy with confidence: - Build optimization - Environment configuration - Error tracking - Monitoring

→ [Production](#)

Quick Reference

Signal Best Practices

```
// ✅ Initialize with correct type
const count = signal<number>(0);
const user = signal<User | null>(null);

// ✅ Use functional updates for complex logic
count.set(prev => prev + 1);

// ✅ Batch related updates
batch(() => {
  firstName.set('Alice');
  lastName.set('Smith');
  age.set(30);
});

// ✅ Don't mutate signal values
const user = signal({ name: 'Alice' });
user().name = 'Bob'; // Bad!

// ✅ Don't create signals in loops
items().forEach(item => {
  const selected = signal(false); // Bad!
});
```

Memo Best Practices

```
// ✅ Use memo for derived values
const total = memo(() => items().reduce((sum, item) => sum + item.price, 0));

// ✅ Chain memos for complex computations
const filtered = memo(() => items().filter(item => item.active));
const sorted = memo(() => filtered().sort((a, b) => a.name.localeCompare(b.name)));

// ✅ Don't use memo for simple references
const userName = memo(() => user().name); // Overkill

// ✅ Don't put side effects in memo
const badMemo = memo(() => {
  console.log('Computing'); // Side effect!
  return count() * 2;
});
```

Effect Best Practices

```
// ✅ Use effects for side effects only
effect(() => {
  console.log('Count changed:', count());
});

// ✅ Return cleanup functions
effect(() => {
  const id = setInterval(() => tick(), 1000);
  return () => clearInterval(id);
});

// ✅ Handle async properly
effect(async () => {
  const data = await fetchData(userId());
  results.set(data);
});

// ✅ Don't update signals that the effect reads
effect(() => {
  const value = count();
  count.set(value + 1); // Infinite loop!
});

// ✅ Don't create effects in loops
items().forEach(item => {
  effect(() => console.log(item)); // Bad!
});
```

Component Best Practices

```
// 🚫 Define clear prop interfaces
interface ButtonProps {
  label: string;
  onClick: () => void;
  variant?: 'primary' | 'secondary';
  disabled?: boolean;
}

function Button({ label, onClick, variant = 'primary', disabled = false }: ButtonProps) {
  return (
    <button
      className={`btn btn-${variant}`}
      onClick={onClick}
      disabled={disabled}
    >
      {label}
    </button>
  );
}

// 🚫 Keep components focused
function UserCard({ user }: { user: User }) {
  return (
    <div className="card">
      <UserAvatar user={user} />
      <UserInfo user={user} />
      <UserActions user={user} />
    </div>
  );
}

// 🚫 Don't create components inside components
function Parent() {
  function Child() { // Bad! Creates new component on every render
    return <div>Child</div>;
  }
  return <Child />;
}
```

Common Anti-Patterns

1. Overusing Effects

🚫 **Bad:**

```
const count = signal(0);
const doubled = signal(0);

effect(() => {
  doubled.set(count() * 2); // Use memo instead!
});
```

🚫 **Good:**

```
const count = signal(0);
const doubled = memo(() => count() * 2);
```

2. Prop Drilling

🚫 **Bad:**

```
<App>
  <Layout theme={theme}>
    <Header theme={theme}>
      <Nav theme={theme}>
        <Button theme={theme} />
      </Nav>
    </Header>
  </Layout>
</App>
```

🚫 **Good:**

```
const ThemeContext = createContext('light');

<ThemeContext.Provider value={theme()}>
  <App />
</ThemeContext.Provider>

function Button() {
  const theme = useContext(ThemeContext);
  return <button className={theme}>Click</button>;
}
```

3. Massive Components

□ **Bad:**

```
function Dashboard() {  
  // 500 lines of code  
  // Multiple concerns  
  // Hard to test  
  // Hard to maintain  
}
```

□ **Good:**

```
function Dashboard() {  
  return (  
    <div>  
      <DashboardHeader />  
      <DashboardStats />  
      <DashboardCharts />  
      <DashboardActivity />  
    </div>  
  );  
}
```

4. Premature Optimization

□ **Bad:**

```
// Optimizing before measuring  
const memoizedEverything = memo(() => someValue);  
const batchedEverything = batch(() => simpleUpdate());
```

□ **Good:**

```
// Measure first, optimize when needed  
const value = someSimpleComputation();  
  
// Optimize hot paths after profiling  
const expensiveComputation = memo(() => {  
  // Actually expensive operation  
});
```

Code Quality Standards

Naming Conventions

```
// Signals: noun describing the value  
const count = signal(0);  
const userName = signal('');  
const isAuthenticated = signal(false);  
  
// Memos: noun describing the computed value  
const doubledCount = memo(() => count() * 2);  
const fullName = memo(() => `${firstName()} ${lastName()}`);  
const hasPermission = memo(() => user()?.role === 'admin');  
  
// Effects: describe what they do (optional)  
effect(() => {  
  // Can use comments to describe complex effects  
  localStorage.setItem('theme', theme());  
});  
  
// Functions: verb describing the action  
function handleClick() {}  
function fetchUserData() {}  
function validateForm() {}  
  
// Components: PascalCase, noun or noun phrase  
function Button() {}  
function UserProfile() {}  
function DashboardLayout() {}
```

File Organization

```

src/
├── components/
│   ├── Button.tsx
│   ├── Input.tsx
│   └── Card.tsx
├── hooks/
│   ├── useAuth.ts
│   ├── useLocalStorage.ts
│   └── useMediaQuery.ts
└── stores/
    ├── userStore.ts
    ├── cartStore.ts
    └── themeStore.ts
├── utils/
    ├── formatting.ts
    ├── validation.ts
    └── api.ts
└── types/
    ├── user.ts
    ├── product.ts
    └── api.ts
pages/
    ├── Home.tsx
    ├── Dashboard.tsx
    └── Settings.tsx

```

Testing Standards

```

import { describe, it, expect } from 'vitest';
import { signal, memo, effect } from '@philjs/core';

describe('Counter', () => {
  it('increments count', () => {
    const count = signal(0);
    count.set(count() + 1);
    expect(count()).toBe(1);
  });

  it('doubles count', () => {
    const count = signal(2);
    const doubled = memo(() => count() * 2);
    expect(doubled()).toBe(4);
  });

  it('tracks count changes', () => {
    const count = signal(0);
    let tracked = 0;

    effect(() => {
      tracked = count();
    });

    count.set(5);
    expect(tracked).toBe(5);
  });
});

```

Documentation Standards

```

/**
 * Custom hook for managing user authentication
 *
 * @returns Authentication state and methods
 *
 * @example
 * ``tsx
 * function App() {
 *   const { user, login, logout } = useAuth();
 *
 *   if (user()) {
 *     return <LoginForm onLogin={login} />;
 *   }
 *
 *   return <Dashboard user={user()} onLogout={logout} />;
 * }
 * ...
 */
export function useAuth() {
  const user = signal<User | null>(null);
  const isAuthenticated = memo(() => user() !== null);

  const login = async (credentials: Credentials) => {
    const userData = await authService.login(credentials);
    user.set(userData);
  };

  const logout = () => {
    user.set(null);
  };

  return { user, isAuthenticated, login, logout };
}

```

Next Steps

Explore specific best practice areas:

1. [Component Patterns](#) - Component design and composition
2. [State Management](#) - Managing application state
3. [Performance](#) - Optimization techniques
4. [Testing](#) - Testing strategies
5. [Code Organization](#) - Project structure
6. [Architecture](#) - Application architecture
7. [Security](#) - Security best practices
8. [Accessibility](#) - Building accessible apps
9. [Production](#) - Production deployment

[Start here](#) to learn PhilJS best practices, then dive into specific areas.

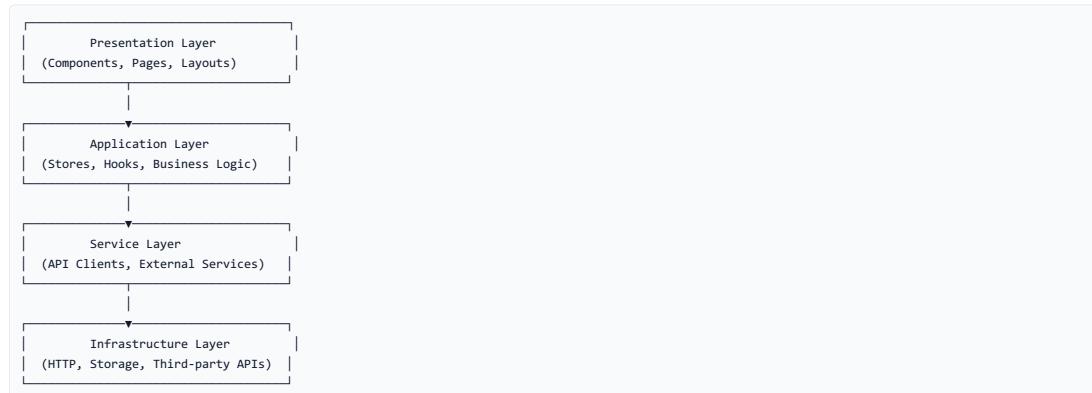
Application Architecture

Design scalable, maintainable PhilJS applications.

Architectural Principles

Separation of Concerns

Divide your application into distinct layers with clear responsibilities.



Layered Architecture

Presentation Layer

Components focused on rendering UI.

```
// pages/Dashboard.tsx
import { userStore } from '@/stores/userStore';
import { statsStore } from '@/stores/statsStore';

export function Dashboard() {
  const { user } = userStore;
  const { stats, loading } = statsStore;

  return (
    <DashboardLayout>
      <UserHeader user={user()} />

      {loading() ? (
        <StatsLoading />
      ) : (
        <StatsView stats={stats()} />
      )}
    </DashboardLayout>
  );
}
```

Application Layer

Business logic and state management.

```
// stores/statsStore.ts
import { signal, effect } from '@philjs/core';
import { statsService } from '@/services/statsService';
import { userStore } from './userStore';

function createStatsStore() {
  const stats = signal<Stats | null>(null);
  const loading = signal(true);
  const error = signal<string | null>(null);

  // Business logic: fetch stats when user changes
  effect(async () => {
    const user = userStore.user();
    if (!user) return;

    loading.set(true);
    try {
      const data = await statsService.getUserStats(user.id);
      stats.set(data);
    } catch (err) {
      error.set(err.message);
    } finally {
      loading.set(false);
    }
  });

  return { stats, loading, error };
}

export const statsStore = createStatsStore();
```

Service Layer

Interact with external systems.

```
// services/statsService.ts
import { api } from './api';
import type { Stats } from '@/types/stats';

export const statsService = {
  async getUserStats(userId: string): Promise<Stats> {
    return api.get(`/users/${userId}/stats`);
  },

  async updateStats(userId: string, updates: Partial<Stats>): Promise<Stats> {
    return api.patch(`users/${userId}/stats`, updates);
  }
};
```

Infrastructure Layer

Low-level implementation details.

```

// services/api.ts
class ApiClient {
  private baseUrl: string;
  private token: string | null = null;

  constructor(baseUrl: string) {
    this.baseUrl = baseUrl;
  }

  setToken(token: string) {
    this.token = token;
  }

  async request<T>(
    endpoint: string,
    options: RequestInit = {}
  ): Promise<T> {
    const headers = {
      'Content-Type': 'application/json',
      ...(this.token && { Authorization: `Bearer ${this.token}` }),
      ...options.headers
    };

    const response = await fetch(`${this.baseUrl}${endpoint}`, {
      ...options,
      headers
    });

    if (!response.ok) {
      throw new Error(`API Error: ${response.statusText}`);
    }

    return response.json();
  }

  async get<T>(endpoint: string): Promise<T> {
    return this.request(endpoint);
  }

  async post<T>(endpoint: string, body: any): Promise<T> {
    return this.request(endpoint, {
      method: 'POST',
      body: JSON.stringify(body)
    });
  }

  // ... other methods
}

export const api = new ApiClient(import.meta.env.VITE_API_URL);

```

Dependency Injection

Context-Based DI

```

// services/ServiceContext.tsx
import { createContext, useContext } from '@philjs/core';

interface Services {
  userService: UserService;
  productService: ProductService;
  analyticsService: AnalyticsService;
}

const ServiceContext = createContext<Services | null>(null);

export function ServiceProvider({
  children,
  services
}: {
  children: JSX.Element;
  services: Services;
}) {
  return (
    <ServiceContext.Provider value={services}>
      {children}
    </ServiceContext.Provider>
  );
}

export function useServices() {
  const services = useContext(ServiceContext);
  if (!services) {
    throw new Error('useServices must be used within ServiceProvider');
  }
  return services;
}

// App.tsx
import { createUserService } from './services/userService';
import { createProductService } from './services/productService';

const services = {
  userService: createUserService(api),
  productService: createProductService(api),
  analyticsService: createAnalyticsService()
};

<ServiceProvider services={services}>
  <App />
</ServiceProvider>

// Component usage
function UserProfile() {
  const { userService } = useServices();

  effect(async () => {
    const user = await userService.getUser(userId());
    // ...
  });
}

```

Factory Pattern

```

// services/userService.ts
export interface UserService {
  getUser(id: string): Promise<User>;
  updateUser(id: string, updates: Partial<User>): Promise<User>;
}

export function createUserService(apiClient: ApiClient): UserService {
  return {
    async getUser(id: string) {
      return apiClient.get(`/users/${id}`);
    },
    async updateUser(id: string, updates: Partial<User>) {
      return apiClient.patch(`/users/${id}`, updates);
    }
  };
}

// Easy to test with mocks
const mockApi = {
  get: vi.fn(),
  patch: vi.fn()
};

const userService = createUserService(mockApi);

```

Plugin Architecture

Plugin System

```
// plugins/types.ts
export interface Plugin {
  name: string;
  install: (app: App) => void;
}

export interface App {
  use(plugin: Plugin): App;
  config: Map<string, any>;
  services: Map<string, any>;
}

// plugins/App.ts
export function createApp(): App {
  const config = new Map<string, any>();
  const services = new Map<string, any>();
  const plugins: Plugin[] = [];

  return {
    use(plugin: Plugin) {
      plugin.install(this);
      plugins.push(plugin);
      return this;
    },
    config,
    services
  };
}

// plugins/analyticsPlugin.ts
export const analyticsPlugin: Plugin = {
  name: 'analytics',
  install(app) {
    const analytics = createAnalytics();

    app.services.set('analytics', analytics);

    // Track route changes
    effect(() => {
      const route = router.currentRoute();
      analytics.trackPageView(route.path);
    });
  }
};

// Usage
const app = createApp()
  .use(analyticsPlugin)
  .use(authPlugin)
  .use(i18nPlugin);
```

Module Federation

Micro-Frontend Architecture

```
// Shell Application
import { lazy, Suspense } from '@philjs/core';

const ProductsApp = lazy(() => import('products/App'));
const CartApp = lazy(() => import('cart/App'));
const CheckoutApp = lazy(() => import('checkout/App'));

function ShellApp() {
  return (
    <MainLayout>
      <Router>
        <Suspense fallback={<PageLoader />}>
          <Route path="/products/*" component={ProductsApp} />
          <Route path="/cart" component={CartApp} />
          <Route path="/checkout" component={CheckoutApp} />
        </Suspense>
      </Router>
    </MainLayout>
  );
}

// Shared state between micro-frontends
export const globalStore = {
  user: userStore,
  cart: cartStore,
  theme: themeStore
};
```

Event-Driven Architecture

Event Bus

```
// events/eventBus.ts
type EventHandler<T = any> = (data: T) => void;

class EventBus {
  private events = new Map<string, Set<EventHandler>>();

  on<T>(event: string, handler: EventHandler<T>) {
    if (!this.events.has(event)) {
      this.events.set(event, new Set());
    }
    this.events.get(event)!.add(handler);
  }

  // Return unsubscribe function
  return () => {
    this.events.get(event)?._delete(handler);
  };
}

emit<T>(event: string, data: T) {
  this.events.get(event)?._forEach(handler => handler(data));
}

off(event: string, handler: EventHandler) {
  this.events.get(event)?._delete(handler);
}

export const eventBus = new EventBus();

// Usage
// Emit events
eventBus.emit('user:login', { userId: '123' });
eventBus.emit('cart:updated', { items: 5 });

// Listen to events
const unsubscribe = eventBus.on('user:login', (data) => {
  console.log('User logged in:', data.userId);
});

// Clean up
effect(() => {
  const unsub = eventBus.on('cart:updated', handleCartUpdate);
  return unsub;
});
```

Domain Events

```
// events/domainEvents.ts
export class UserLoggedInEvent {
  constructor(public userId: string, public timestamp: Date) {}
}

export class CartUpdatedEvent {
  constructor(public items: CartItem[], public total: number) {}
}

// Event handlers
eventBus.on<UserLoggedInEvent>('user:logged-in', (event) => {
  analyticsService.trackLogin(event.userId);
  fetchUserData(event.userId);
});

eventBus.on<CartUpdatedEvent>('cart:updated', (event) => {
  localStorage.setItem('cart', JSON.stringify(event.items));
  updateCartBadge(event.items.length);
});
```

CQRS Pattern

Command Query Responsibility Segregation

```
// stores/productStore/commands.ts
export interface CreateProductCommand {
  name: string;
  price: number;
  description: string;
}

export interface UpdateProductCommand {
  id: string;
  updates: Partial<Product>;
}

export interface DeleteProductCommand {
  id: string;
```

```

    ...
}

// stores/productStore/queries.ts
export interface GetProductQuery {
  id: string;
}

export interface SearchProductsQuery {
  term: string;
  category?: string;
  minPrice?: number;
  maxPrice?: number;
}

// stores/productStore/index.ts
function createProductStore() {
  const products = signal<Product[]>([[]]);

  // Commands (write operations)
  const commands = {
    async create(command: CreateProductCommand) {
      const product = await productService.create(command);
      products.set([...products(), product]);
      eventBus.emit('product:created', product);
    },

    async update(command: UpdateProductCommand) {
      const updated = await productService.update(command.id, command.updates);
      products.set(
        products().map(p => p.id === command.id ? updated : p)
      );
      eventBus.emit('product:updated', updated);
    },

    async delete(command: DeleteProductCommand) {
      await productService.delete(command.id);
      products.set(products().filter(p => p.id !== command.id));
      eventBus.emit('product:deleted', command.id);
    }
  };
}

// Queries (read operations)
const queries = {
  getById(query: GetProductQuery): Product | undefined {
    return products().find(p => p.id === query.id);
  },

  search(query: SearchProductsQuery): Product[] {
    return products().filter(p => {
      if (query.term && !p.name.includes(query.term)) return false;
      if (query.category && p.category !== query.category) return false;
      if (query.minPrice && p.price < query.minPrice) return false;
      if (query.maxPrice && p.price > query.maxPrice) return false;
      return true;
    });
  }
};

return { products, commands, queries };
}

// Usage
const { commands, queries } = productStore;

// Execute command
await commands.create({
  name: 'New Product',
  price: 99.99,
  description: 'Description'
});

// Execute query
const product = queries.getById({ id: '123' });
const results = queries.search({ term: 'laptop', maxPrice: 1000 });

```

Hexagonal Architecture (Ports & Adapters)

Core Domain

```

// domain/user.ts
export class User {
  constructor(
    public id: string,
    public name: string,
    public email: string
  ) {}

  updateName(newName: string) {
    if (!newName.trim()) {
      throw new Error('Name cannot be empty');
    }
    this.name = newName;
  }

  isAdmin(): boolean {
    return this.email.endsWith('@admin.com');
  }
}

// domain/ports/userRepository.ts (Port)
export interface UserRepository {
  findById(id: string): Promise<User | null>;
  save(user: User): Promise<void>;
  delete(id: string): Promise<void>;
}

// infrastructure/adapters/apiUserRepository.ts (Adapter)
export class ApiUserRepository implements UserRepository {
  constructor(private api: ApiClient) {}

  async findById(id: string): Promise<User | null> {
    const data = await this.api.get<UserDTO>('/users/${id}`);
    return data ? this.toDomain(data) : null;
  }

  async save(user: User): Promise<void> {
    await this.api.post('/users', this.toDTO(user));
  }

  async delete(id: string): Promise<void> {
    await this.api.delete('/users/${id}');
  }

  private toDomain(dto: UserDTO): User {
    return new User(dto.id, dto.name, dto.email);
  }

  private toDTO(user: User): UserDTO {
    return {
      id: user.id,
      name: user.name,
      email: user.email
    };
  }
}

// Application uses port, not adapter
function createUserService(repository: UserRepository) {
  return {
    async getUser(id: string) {
      return repository.findById(id);
    },

    async updateUser(id: string, name: string) {
      const user = await repository.findById(id);
      if (!user) throw new Error('User not found');

      user.updateName(name);
      await repository.save(user);

      return user;
    }
  };
}

```

Scalability Patterns

Lazy Module Loading

```
// Large application broken into modules
const routes = [
  {
    path: '/admin/*',
    component: lazy(() => import('./modules/admin'))
  },
  {
    path: '/analytics/*',
    component: lazy(() => import('./modules/analytics'))
  },
  {
    path: '/reports/*',
    component: lazy(() => import('./modules/reports'))
  }
];

```

Code Splitting by Route

```
function App() {
  return (
    <Router>
      <Suspense fallback={<PageLoader />}>
        <Route path="/" component={lazy(() => import('./pages/Home'))} />
        <Route path="/dashboard" component={lazy(() => import('./pages/Dashboard'))} />
        <Route path="/settings" component={lazy(() => import('./pages/Settings'))} />
      </Suspense>
    </Router>
  );
}
```

Summary

Architecture Best Practices:

- Separate concerns into layers □ Use dependency injection □ Design with interfaces (ports) □ Implement plugin architecture for extensibility □ Use event-driven patterns for decoupling □ Consider CQRS for complex domains □ Apply hexagonal architecture for testability □ Lazy load modules and routes □ Keep domain logic pure □ Make infrastructure replaceable

Next: [Security →](#)

Code Organization

Best practices for structuring and organizing PhilJS projects.

Project Structure

Standard Structure

```

my-philjs-app/
├── public/          # Static assets
|   ├── favicon.ico
|   ├── robots.txt
|   └── images/
└── src/
    ├── components/    # Reusable UI components
    |   ├── Button/
    |   |   ├── Button.tsx
    |   |   ├── Button.test.tsx
    |   |   └── Button.css
    |   ├── Input/
    |   |   └── Card/
    |   └── pages/       # Page components
    |       ├── Home.tsx
    |       ├── Dashboard.tsx
    |       └── Settings.tsx
    ├── layouts/        # Layout components
    |   ├── MainLayout.tsx
    |   └── AuthLayout.tsx
    ├── hooks/          # Custom hooks
    |   ├── useAuth.ts
    |   ├── useLocalStorage.ts
    |   └── useMediaQuery.ts
    ├── stores/         # Global state stores
    |   ├── userStore.ts
    |   ├── cartStore.ts
    |   └── themeStore.ts
    ├── services/       # API and external services
    |   ├── api.ts
    |   ├── auth.ts
    |   └── analytics.ts
    ├── utils/          # Utility functions
    |   ├── formatting.ts
    |   ├── validation.ts
    |   └── constants.ts
    ├── types/          # TypeScript types
    |   ├── user.ts
    |   ├── product.ts
    |   └── api.ts
    ├── styles/         # Global styles
    |   ├── global.css
    |   ├── variables.css
    |   └── reset.css
    ├── App.tsx         # Root component
    └── index.tsx       # Entry point
└── tests/
    ├── setup.ts
    └── helpers.ts
└── .env              # Environment variables
└── .env.example
└── package.json
└── tsconfig.json
└── vite.config.ts    # Build configuration

```

Feature-Based Structure

```

my-philjs-app/
├── src/
|   ├── features/
|   |   ├── auth/
|   |   |   ├── components/
|   |   |   |   ├── LoginForm.tsx
|   |   |   |   └── RegisterForm.tsx
|   |   |   ├── hooks/
|   |   |   |   └── useAuth.ts
|   |   |   ├── services/
|   |   |   |   └── authService.ts
|   |   |   ├── stores/
|   |   |   |   └── authStore.ts
|   |   |   ├── types/
|   |   |   |   └── auth.ts
|   |   |   └── index.ts
|   |   ├── products/
|   |   |   ├── components/
|   |   |   ├── hooks/
|   |   |   ├── services/
|   |   |   |   └── index.ts
|   |   |   └── cart/
|   |   └── shared/      # Shared across features
|       ├── components/
|       ├── hooks/
|       └── utils/
|   └── App.tsx
└── index.tsx

```

File Naming Conventions

Components

- ❑ PascalCase for components
Button.tsx
UserProfile.tsx
ProductCard.tsx
- ❑ Co-locate styles
Button.tsx
Button.css
Button.test.tsx
- ❑ Index files for folders
components/Button/index.tsx // Re-exports Button

Utilities and Hooks

- ❑ camelCase for functions
formatting.ts
validation.ts
- ❑ "use" prefix for hooks
useAuth.ts
useLocalStorage.ts
useMediaQuery.ts

Types

- ❑ Descriptive names
user.ts // User-related types
product.ts // Product-related types
api.ts // API response types
- ❑ Type suffixes when needed
userTypes.ts // If conflicts with userUtils.ts

Component Organization

Component File Structure

```

// imports - external
import { signal, memo } from '@philjs/core';

// imports - internal
import { Button } from '@/components/Button';
import { formatDate } from '@/utils/formatting';

// types
interface UserCardProps {
  user: User;
  onEdit?: () => void;
}

// component
export function UserCard({ user, onEdit }: UserCardProps) {
  // hooks and state
  const expanded = signal(false);

  // computed values
  const formattedDate = memo(() => formatDate(user.createdAt));

  // handlers
  const handleToggle = () => {
    expanded.set(!expanded());
  };

  // render
  return (
    <div className="user-card">
      <h3>{user.name}</h3>
      <p>{formattedDate()}</p>

      {onEdit && (
        <Button label="Edit" onClick={onEdit} />
      )}

      {expanded() && (
        <div className="details">
          {/* Additional details */}
        </div>
      )}
    </div>
  );
}

// sub-components (if needed)
function UserCardHeader({ user }: { user: User }) {
  return <div>{user.name}</div>;
}

```

Component Folder Pattern

```

components/
└── UserCard/
    ├── index.ts          # Exports
    ├── UserCard.tsx       # Main component
    ├── UserCard.css        # Styles
    ├── UserCard.test.tsx   # Tests
    └── UserCardHeader.tsx # Sub-component
        └── types.ts        # Local types

// index.ts
export { UserCard } from './UserCard';
export type { UserCardProps } from './types';

```

Store Organization

Simple Store

```

// stores/themeStore.ts
import { signal } from '@philjs/core';

export type Theme = 'light' | 'dark';

const theme = signal('light');

export const themeStore = {
  theme,
  toggle: () => {
    theme.set(theme() === 'light' ? 'dark' : 'light');
  },
  set: (value: Theme) => {
    theme.set(value);
  }
};

```

Complex Store

```
// stores/userStore/
├── index.ts          # Main store
├── types.ts          # Store types
├── actions.ts        # Store actions
└── selectors.ts      # Derived values

// stores/userStore/types.ts
export interface User {
  id: string;
  name: string;
  email: string;
  role: 'admin' | 'user';
}

export interface UserState {
  user: User | null;
  loading: boolean;
  error: string | null;
}

// stores/userStore/index.ts
import { signal, memo } from '@philjs/core';
import { fetchUser, loginUser } from './actions';
import { isAdmin, isAuthenticated } from './selectors';

function createUserStore() {
  const user = signal<User | null>(null);
  const loading = signal(false);
  const error = signal<string | null>(null);

  return {
    // State
    user,
    loading,
    error,

    // Actions
    fetchUser: fetchUser(user, loading, error),
    login: loginUser(user, loading, error),

    // Selectors
    isAdmin: isAdmin(user),
    isAuthenticated: isAuthenticated(user)
  };
}

export const userStore = createUserStore();
```

Import Organization

Import Order

```
// 1. External dependencies
import { signal, memo, effect } from '@philjs/core';
import { Router, Route } from '@philjs/router';

// 2. Internal absolute imports
import { Button } from '@/components/Button';
import { userStore } from '@/stores/userStore';
import { formatDate } from '@/utils/formatting';

// 3. Relative imports
import { UserCard } from './UserCard';
import { config } from './config';

// 4. Types
import type { User } from '@/types/user';

// 5. Styles
import './App.css';
```

Path Aliases

```
// tsconfig.json
{
  "compilerOptions": {
    "baseUrl": ".",
    "paths": {
      "@/*": ["src/*"],
      "@components/*": ["src/components/*"],
      "@hooks/*": ["src/hooks/*"],
      "@stores/*": ["src/stores/*"],
      "@utils/*": ["src/utils/*"],
      "@types/*": ["src/types/*"]
    }
  }
}
```

Usage:

```
// ⚡ Clean imports with aliases
import { Button } from '@components/Button';
import { useAuth } from '@hooks/useAuth';
import { userStore } from '@stores/userStore';

// ⚡ Messy relative imports
import { Button } from '../../../../../components/Button';
import { useAuth } from '../../../../../hooks/useAuth';
```

Type Organization

Shared Types

```
// types/user.ts
export interface User {
  id: string;
  name: string;
  email: string;
  role: 'admin' | 'user';
  createdAt: Date;
}

export interface UserCredentials {
  email: string;
  password: string;
}

export type UserRole = User['role'];
```

API Types

```
// types/api.ts
export interface ApiResponse<T> {
  data: T;
  status: number;
  message?: string;
}

export interface PaginatedResponse<T> {
  items: T[];
  total: number;
  page: number;
  pageSize: number;
}

export interface ApiError {
  message: string;
  code: string;
  details?: Record<string, string[]>;
}
```

Component Types

```
// components/Button/types.ts
export interface ButtonProps {
  label: string;
  onClick: () => void;
  variant?: 'primary' | 'secondary' | 'danger';
  size?: 'small' | 'medium' | 'large';
  disabled?: boolean;
  loading?: boolean;
}

export type ButtonVariant = ButtonProps['variant'];
export type ButtonSize = ButtonProps['size'];
```

Service Organization

API Service

```
// services/api.ts
const API_BASE_URL = import.meta.env.VITE_API_URL;

class ApiClient {
    private baseUrl: string;

    constructor(baseUrl: string) {
        this.baseUrl = baseUrl;
    }

    async get<T>(endpoint: string): Promise<T> {
        const response = await fetch(`${this.baseUrl}${endpoint}`);
        if (!response.ok) throw new Error('Request failed');
        return response.json();
    }

    async post<T>(endpoint: string, body: any): Promise<T> {
        const response = await fetch(`${this.baseUrl}${endpoint}`, {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(body)
        });
        if (!response.ok) throw new Error('Request failed');
        return response.json();
    }

    // patch, delete, etc.
}

export const api = new ApiClient(API_BASE_URL);
```

Domain Services

```
// services/userService.ts
import { api } from './api';
import type { User, UserCredentials } from '@/types/user';

export const userService = {
    async getUser(id: string): Promise<User> {
        return api.get(`/users/${id}`);
    },

    async login(credentials: UserCredentials): Promise<User> {
        return api.post('/auth/login', credentials);
    },

    async updateUser(id: string, updates: Partial<User>): Promise<User> {
        return api.patch(`/users/${id}`, updates);
    }
};
```

Utility Organization

Single-Purpose Files

```
// utils/formatting.ts
export function formatDate(date: Date): string {
    return new Intl.DateTimeFormat('en-US').format(date);
}

export function formatCurrency(amount: number): string {
    return new Intl.NumberFormat('en-US', {
        style: 'currency',
        currency: 'USD'
    }).format(amount);
}

// utils/validation.ts
export function isValidEmail(email: string): boolean {
    return /^[^@]+@[^\s@]+\.\w+/.test(email);
}

export function isValidPassword(password: string): boolean {
    return password.length >= 8;
}
```

Grouped Utilities

```
// utils/string.ts
export const stringUtils = {
  capitalize(str: string): string {
    return str.charAt(0).toUpperCase() + str.slice(1);
  },
  truncate(str: string, length: number): string {
    return str.length > length ? `${str.slice(0, length)}...` : str;
  },
  slugify(str: string): string {
    return str.toLowerCase().replace(/\s+/g, '-');
  }
};
```

Constants Organization

```
// utils/constants.ts
export const API_ENDPOINTS = {
  USERS: '/users',
  PRODUCTS: '/products',
  ORDERS: '/orders'
} as const;

export const ROUTES = {
  HOME: '/',
  DASHBOARD: '/dashboard',
  SETTINGS: '/settings',
  LOGIN: '/login'
} as const;

export const THEMES = {
  LIGHT: 'light',
  DARK: 'dark'
} as const;

export const MAX_FILE_SIZE = 5 * 1024 * 1024; // 5MB
export const ITEMS_PER_PAGE = 20;
```

Environment Variables

```
# .env
VITE_API_URL=http://localhost:3000
VITE_APP_NAME=PhiliJS App
VITE_ENABLE_ANALYTICS=true

# .env.example
VITE_API_URL=
VITE_APP_NAME=
VITE_ENABLE_ANALYTICS=
```

```
// utils/env.ts
interface Env {
  apiUrl: string;
  appName: string;
  enableAnalytics: boolean;
}

function getEnv(): Env {
  return {
    apiUrl: import.meta.env.VITE_API_URL,
    appName: import.meta.env.VITE_APP_NAME,
    enableAnalytics: import.meta.env.VITE_ENABLE_ANALYTICS === 'true'
  };
}

export const env = getEnv();
```

Barrel Exports

Component Barrel

```
// components/index.ts
export { Button } from './Button';
export { Input } from './Input';
export { Card } from './Card';
export { Modal } from './Modal';

// Usage
import { Button, Input, Card } from '@/components';
```

Avoid Deep Barrels

```
// ⚠ Don't re-export everything
export * from './Button';
export * from './Input';
// Type pollution, unclear exports

// ⚠ Be explicit
export { Button } from './Button';
export { Input } from './Input';
export type { ButtonProps } from './Button';
export type { InputProps } from './Input';
```

Documentation

Component Documentation

```
/** 
 * Button component for user actions
 *
 * @example
 * ``tsx
 * <Button
 *   label="Click me"
 *   onClick={() => console.log('Clicked')}
 *   variant="primary"
 * />
 * ...
 */
export function Button({ label, onClick, variant = 'primary' }: ButtonProps) {
  // ...
}
```

Store Documentation

```
/** 
 * User authentication store
 *
 * Manages user session, Login, and Logout
 *
 * @example
 * ``tsx
 * const { user, Login, Logout } = userStore;
 *
 * if (user()) {
 *   return <Dashboard user={user()} />;
 * }
 * ...
 */
export const userStore = createUserStore();
```

Summary

Organization Best Practices:

- Use consistent project structure
- Follow naming conventions
- Co-locate related files
- Use path aliases for clean imports
- Organize types separately
- Group utilities logically
- Document components and stores
- Use barrel exports wisely
- Keep files focused and small
- Structure for scalability

Next: [Architecture →](#)

Component Patterns

Essential patterns for building maintainable, reusable PhilJS components.

Component Types

Presentational Components

Pure components that receive data via props and render UI.

```

interface ButtonProps {
  label: string;
  onClick: () => void;
  variant?: 'primary' | 'secondary' | 'danger';
  disabled?: boolean;
  size?: 'small' | 'medium' | 'large';
}

function Button({label,
  onClick,
  variant = 'primary',
  disabled = false,
  size = 'medium'
}: ButtonProps) {
  return (
    <button
      className={`btn btn-${variant} btn-${size}`}
      onClick={onClick}
      disabled={disabled}
    >
      {label}
    </button>
  );
}

```

Benefits: - Easy to test - Highly reusable - No side effects - Predictable behavior

Container Components

Components that manage state and logic.

```

function UserDashboard() {
  const user = signal<User | null>(null);
  const loading = signal(true);
  const error = signal<string | null>(null);

  effect(async () => {
    try {
      const data = await fetchCurrentUser();
      user.set(data);
    } catch (err) {
      error.set(err.message);
    } finally {
      loading.set(false);
    }
  });

  if (loading()) return <LoadingSpinner />;
  if (error()) return <ErrorMessage message={error()!} />;
  if (!user()) return <NotFound />;

  return <UserDashboardView user={user()!} />;
}

function UserDashboardView({ user }: { user: User }) {
  return (
    <div>
      <UserHeader user={user} />
      <UserStats user={user} />
      <UserActivity user={user} />
    </div>
  );
}

```

Benefits: - Separation of concerns - Presentational components stay pure - Logic centralized and testable

Composition Patterns

Children Pattern

Pass components as children.

```

interface CardProps {
  children: JSX.Element;
  title?: string;
  footer?: JSX.Element;
}

function Card({ children, title, footer }: CardProps) {
  return (
    <div className="card">
      {title && <div className="card-header">{title}</div>}
      <div className="card-body">{children}</div>
      {footer && <div className="card-footer">{footer}</div>}
    </div>
  );
}

// Usage
<Card
  title="User Profile"
  footer={<Button label="Save" onClick={handleSave} />}
>
  <UserForm />
</Card>

```

Render Props Pattern

Pass functions as props to customize rendering.

```

interface ListProps<T> {
  items: T[];
  renderItem: (item: T, index: number) => JSX.Element;
  renderEmpty?: () => JSX.Element;
}

function List<T>({ items, renderItem, renderEmpty }: ListProps<T>) {
  if (items.length === 0 && renderEmpty) {
    return renderEmpty();
  }

  return (
    <ul>
      {items.map((item, index) => (
        <li key={index}>{renderItem(item, index)}</li>
      ))}
    </ul>
  );
}

// Usage
<List
  items={users()}
  renderItem={(user) => (
    <UserCard user={user} />
  )}
  renderEmpty={() => <div>No users found</div>}
/>

```

Compound Components Pattern

Components that work together to form a cohesive API.

```

function Tabs({ children }: { children: JSX.Element }) {
  const activeTab = signal(0);

  return (
    <TabsContext.Provider value={{ activeTab }}>
      <div className="tabs">{children}</div>
    </TabsContext.Provider>
  );
}

function TabList({ children }: { children: JSX.Element }) {
  return <div div className="tab-list">{children}</div>;
}

function Tab({ index, children }: { index: number; children: JSX.Element }) {
  const { activeTab } = useContext(TabsContext);

  return (
    <button
      className={activeTab() === index ? 'tab active' : 'tab'}
      onClick={() => activeTab.set(index)}
    >
      {children}</button>
  );
}

function TabPanels({ children }: { children: JSX.Element[] }) {
  const { activeTab } = useContext(TabsContext);
  return <div className="tab-panels">{children[activeTab()]}</div>;
}

function TabPanel({ children }: { children: JSX.Element }) {
  return <div className="tab-panel">{children}</div>;
}

// Attach sub-components
Tabs.List = TabList;
Tabs.Tab = Tab;
Tabs.Panels = TabPanels;
Tabs.Panel = TabPanel;

// Usage
<Tabs>
  <Tabs.List>
    <Tabs.Tab index={0}>Profile</Tabs.Tab>
    <Tabs.Tab index={1}>Settings</Tabs.Tab>
    <Tabs.Tab index={2}>Activity</Tabs.Tab>
  </Tabs.List>

  <Tabs.Panels>
    <Tabs.Panel><ProfileContent /></Tabs.Panel>
    <Tabs.Panel><SettingsContent /></Tabs.Panel>
    <Tabs.Panel><ActivityContent /></Tabs.Panel>
  </Tabs.Panels>
</Tabs>

```

Props Patterns

Optional Props with Defaults

```

interface TooltipProps {
  content: string;
  children: JSX.Element;
  position?: 'top' | 'bottom' | 'left' | 'right';
  delay?: number;
}

function Tooltip({
  content,
  children,
  position = 'top',
  delay = 200
}: TooltipProps) {
  const visible = signal(false);

  return (
    <div
      className="tooltip-container"
      onMouseEnter={() => setTimeout(() => visible.set(true), delay)}
      onMouseLeave={() => visible.set(false)}
    >
      {children}
      {visible() && (
        <div className={`tooltip tooltip-${position}`}>
          {content}
        </div>
      )}
    </div>
  );
}

```

Discriminated Union Props

```

type ButtonProps =
  | {
      variant: 'link';
      href: string;
      onClick?: never;
    }
  | {
      variant: 'button';
      onClick: () => void;
      href?: never;
    };

function SmartButton(props: ButtonProps) {
  if (props.variant === 'link') {
    return <a href={props.href} className="btn-link">{props.children}</a>;
  }

  return (
    <button onClick={props.onClick} className="btn">
      {props.children}
    </button>
  );
}

// Type-safe usage
<SmartButton variant="link" href="/about">About</SmartButton>
<SmartButton variant="button" onClick={() => save()}>Save</SmartButton>

```

Polymorphic Components

```

type As = keyof JSX.IntrinsicElements;

interface BoxProps<T extends As = 'div'> {
  as?: T;
  children: JSX.Element;
}

function Box<T extends As = 'div'>({ 
  as,
  children,
  ...props
}: BoxProps<T> & Omit<JSX.IntrinsicElements[T], 'as'>) {
  const Component = as || 'div';
  return <Component {...props}>{children}</Component>;
}

// Usage
<Box>Default div</Box>
<Box as="section">Rendered as section</Box>
<Box as="article">Rendered as article</Box>

```

State Management Patterns

Lifting State Up

```
function TemperatureCalculator() {
  const temperature = signal('');
  const scale = signal<'C' | 'F'>('C');

  return (
    <div>
      <TemperatureInput
        scale="C"
        temperature={temperature()}
        onTemperatureChange={temperature.set}
        onScaleChange={() => scale.set('C')}
      />

      <TemperatureInput
        scale="F"
        temperature={temperature()}
        onTemperatureChange={temperature.set}
        onScaleChange={() => scale.set('F')}
      />

      <BoilingVerdict celsius={toCelsius(temperature(), scale())} />
    </div>
  );
}
```

Local State

```
function Accordion({ title, children }: { title: string; children: JSX.Element }) {
  const isOpen = signal(false);

  return (
    <div className="accordion">
      <button
        className="accordion-header"
        onClick={() => isOpen.set(!isOpen())}
      >
        {title}
        <span>{isOpen() ? '▼' : '▶'}</span>
      </button>

      {isOpen() && (
        <div className="accordion-content">
          {children}
        </div>
      )}
    </div>
  );
}
```

Controlled vs Uncontrolled

Controlled:

```
interface InputProps {
  value: string;
  onChange: (value: string) => void;
}

function ControlledInput({ value, onChange }: InputProps) {
  return (
    <input
      value={value}
      onChange={(e) => onChange(e.currentTarget.value)}
    />
  );
}

// Usage
function Form() {
  const email = signal('');

  return <ControlledInput value={email()} onChange={email.set} />;
}
```

Uncontrolled:

```
function UncontrolledInput({ defaultValue }: { defaultValue?: string }) {
  let inputRef: HTMLInputElement | undefined;

  const getValue = () => inputRef?.value || '';

  return <input ref={inputRef} defaultValue={defaultValue} />;
}
```

Error Handling Patterns

Error Boundaries

```
function App() {
  return (
    <ErrorBoundary>
      fallback={(error) => <ErrorPage error={error} />}
      onError={(error) => {
        logError(error);
        reportToSentry(error);
      }}
    </ErrorBoundary>
  );
}

function ErrorPage({ error }: { error: Error }) {
  return (
    <div className="error-page">
      <h1>Something went wrong</h1>
      <p>{error.message}</p>
      <button label="Go Home" onClick={() => window.location.href = '/'}></button>
    </div>
  );
}
```

Try-Catch in Effects

```
function UserProfile({ userId }: { userId: string }) {
  const user = signal<User | null>(null);
  const error = signal<Error | null>(null);
  const loading = signal(true);

  effect(async () => {
    loading.set(true);
    error.set(null);

    try {
      const data = await fetchUser(userId);
      user.set(data);
    } catch (err) {
      error.set(err as Error);
    } finally {
      loading.set(false);
    }
  });

  if (loading()) return <Spinner />;
  if (error()) return <ErrorMessage error={error()!} />;
  if (!user()) return <NotFound />

  return <UserCard user={user()!} />;
}
```

Loading States

Suspense Pattern

```

import { Suspense, lazy } from '@philjs/core';

const Dashboard = lazy(() => import('./Dashboard'));
const Settings = lazy(() => import('./Settings'));

function App() {
  return (
    <Suspense fallback={<PageLoader />}>
      <Router>
        <Route path="/dashboard" component={Dashboard} />
        <Route path="/settings" component={Settings} />
      </Router>
    </Suspense>
  );
}

function PageLoader() {
  return (
    <div className="page-loader">
      <Spinner size="large" />
      <p>Loading...</p>
    </div>
  );
}

```

Skeleton Screens

```

function UserCard({ user }: { user: User | null }) {
  if (!user) {
    return <UserCardSkeleton />;
  }

  return (
    <div className="user-card">
      <img src={user.avatar} alt={user.name} />
      <h3>{user.name}</h3>
      <p>{user.bio}</p>
    </div>
  );
}

function UserCardSkeleton() {
  return (
    <div className="user-card skeleton">
      <div className="skeleton-avatar" />
      <div className="skeleton-text" />
      <div className="skeleton-text short" />
    </div>
  );
}

```

Conditional Rendering

Multiple Conditions

```

function UserStatus({ user }: { user: User }) {
  if (!user.emailVerified) {
    return <VerifyEmailBanner email={user.email} />;
  }

  if (user.subscription === 'trial' && user.trialEndsIn < 3) {
    return <TrialExpiringBanner daysLeft={user.trialEndsIn} />;
  }

  if (!user.hasCompletedOnboarding) {
    return <OnboardingBanner />;
  }

  return null;
}

```

Render Functions

```
function Panel({ status }: { status: 'loading' | 'error' | 'success' | 'empty' }) {
  const renderContent = () => {
    switch (status) {
      case 'loading':
        return <Spinner />;
      case 'error':
        return <ErrorMessage />;
      case 'empty':
        return <EmptyState />;
      case 'success':
        return <DataView />;
    }
  };

  return (
    <div className="panel">
      {renderContent()}
    </div>
  );
}
```

Form Patterns

Controlled Form

```

function LoginForm() {
  const email = signal('');
  const password = signal('');
  const errors = signal<Record<string, string>>({});
  const submitting = signal(false);

  const validate = () => {
    const newErrors: Record<string, string> = {};

    if (!email()) {
      newErrors.email = 'Email is required';
    } else if (!isValidEmail(email())) {
      newErrors.email = 'Invalid email format';
    }

    if (!password()) {
      newErrors.password = 'Password is required';
    } else if (password().length < 8) {
      newErrors.password = 'Password must be at least 8 characters';
    }

    errors.set(newErrors);
    return Object.keys(newErrors).length === 0;
  };

  const handleSubmit = async (e: Event) => {
    e.preventDefault();

    if (!validate()) return;

    submitting.set(true);

    try {
      await login(email(), password());
    } catch (err) {
      errors.set({ form: 'Login failed. Please try again.' });
    } finally {
      submitting.set(false);
    }
  };
}

return (
  <form onSubmit={handleSubmit}>
    <Input
      label="Email"
      type="email"
      value={email()}
      onChange={email.set}
      error={errors().email}
    />

    <Input
      label="Password"
      type="password"
      value={password()}
      onChange={password.set}
      error={errors().password}
    />

    {errors().form && <FormError message={errors().form} />}
  </form>
);
}

```

List Rendering

With Keys

```

function UserList({ users }: { users: User[] }) {
  return (
    <ul>
      {users.map(user => (
        <li key={user.id}>
          <UserCard user={user} />
        </li>
      ))}
    </ul>
  );
}

```

With Index (When Appropriate)

```
function StepIndicator({ steps, current }: { steps: string[]; current: number }) {
  return (
    <div className="steps">
      {steps.map((step, index) => (
        <div
          key={index} // OK here - list is static
          className={index === current ? 'step active' : 'step'}
        >
          {step}
        </div>
      )));
    </div>
  );
}
```

Empty States

```
function ProductList({ products }: { products: Product[] }) {
  if (products.length === 0) {
    return (
      <EmptyState
        icon={<ShoppingBagIcon />}
        title="No products found"
        description="Try adjusting your filters"
        action={<Button label="Clear Filters" onClick={clearFilters} />}
      />
    );
  }

  return (
    <div className="product-grid">
      {products.map(product => (
        <ProductCard key={product.id} product={product} />
      )));
    </div>
  );
}
```

Summary

Key Takeaways:

- Separate presentational and container components □ Use composition over complex inheritance □ Design clear, type-safe prop interfaces □ Handle errors gracefully with boundaries □ Provide loading states and skeletons □ Use controlled components for forms □ Always provide keys for lists □ Handle empty states explicitly

Next: [State Management →](#)

Accessibility Best Practices

Build inclusive PhilJS applications that everyone can use.

Semantic HTML

Use Correct Elements

```
// ☐ Semantic HTML
function Article() {
  return (
    <article>
      <header>
        <h1>Article Title</h1>
        <time datetime="2024-01-15">January 15, 2024</time>
      </header>

      <section>
        <p>Article content...</p>
      </section>

      <footer>
        <address>By John Doe</address>
      </footer>
    </article>
  );
}

// ☐ Non-semantic divs
function BadArticle() {
  return (
    <div>
      <div>
        <div>Article Title</div>
        <div>January 15, 2024</div>
      </div>

      <div>
        <div>Article content...</div>
      </div>
    </div>
  );
}
```

Navigation

```
// ☐ Proper navigation structure
function Navigation() {
  return (
    <nav aria-label="Main navigation">
      <ul>
        <li><a href="/">Home</a></li>
        <li><a href="/about">About</a></li>
        <li><a href="/contact">Contact</a></li>
      </ul>
    </nav>
  );
}

// ☐ Multiple navigations
function Page() {
  return (
    <div>
      <nav aria-label="Main navigation">
        {/* Primary navigation */}
      </nav>

      <nav aria-label="Breadcrumb">
        {/* Breadcrumb navigation */}
      </nav>

      <main>
        {/* Content */}
      </main>

      <nav aria-label="Footer navigation">
        {/* Footer navigation */}
      </nav>
    </div>
  );
}
```

ARIA Attributes

ARIA Labels

```
// 📄 Accessible buttons
function IconButton({ icon, onClick, label }: IconButtonProps) {
  return (
    <button onClick={onClick} aria-label={label}>
      {icon}
    </button>
  );
}

<IconButton
  icon={TrashIcon}
  onClick={deleteItem}
  label="Delete item"
/>

// 📄 Labeling form inputs
function FormField() {
  return (
    <div>
      <label htmlFor="email">Email Address</label>
      <input
        id="email"
        type="email"
        aria-describedby="email-hint"
        aria-required="true"
      />
      <div id="email-hint">We'll never share your email.</div>
    </div>
  );
}
```

ARIA Roles

```
// 📄 Custom interactive elements
function TabPanel({ children, id, isActive }: TabPanelProps) {
  return (
    <div
      role="tabpanel"
      id={id}
      aria-hidden={!isActive}
      tabIndex={isActive ? 0 : -1}
    >
      {children}
    </div>
  );
}

// 📄 Alert messages
function Alert({ message, type }: AlertProps) {
  return (
    <div
      role="alert"
      aria-live="polite"
      className={`alert alert-${type}`}
    >
      {message}
    </div>
  );
}
```

ARIA States

```
function Accordion({ title, children }: AccordionProps) {
  const isOpen = signal(false);

  return (
    <div>
      <button
        onClick={() => isOpen.set(!isOpen())}
        aria-expanded={isOpen()}
        aria-controls="accordion-content"
      >
        {title}
      </button>

      <div
        id="accordion-content"
        hidden={!isOpen()}
      >
        {children}
      </div>
    </div>
  );
}
```

Keyboard Navigation

Focus Management

```
function Modal({ isOpen, onClose, children }: ModalProps) {
  let firstFocusableElement: HTMLElement | null = null;
  let lastFocusableElement: HTMLElement | null = null;

  effect(() => {
    if (!isOpen()) return;

    const modal = document.getElementById('modal');
    if (!modal) return;

    const focusableElements = modal.querySelectorAll(
      'button, [href], input, select, textarea, [tabindex]:not([tabindex="-1"])'
    );

    firstFocusableElement = focusableElements[0] as HTMLElement;
    lastFocusableElement = focusableElements[
      focusableElements.length - 1
    ] as HTMLElement;

    // Focus first element
    firstFocusableElement?.focus();

    const handleKeyDown = (e: KeyboardEvent) => {
      if (e.key === 'Escape') {
        onClose();
      }

      if (e.key === 'Tab') {
        if (e.shiftKey && document.activeElement === firstFocusableElement) {
          e.preventDefault();
          lastFocusableElement?.focus();
        } else if (!e.shiftKey && document.activeElement === lastFocusableElement) {
          e.preventDefault();
          firstFocusableElement?.focus();
        }
      }
    };
  });

  document.addEventListener('keydown', handleKeyDown);

  return () => {
    document.removeEventListener('keydown', handleKeyDown);
  };
});

if (!isOpen()) return null;

return (
  <div
    id="modal"
    role="dialog"
    aria-modal="true"
    aria-labelledby="modal-title"
  >
  <h2 id="modal-title">Modal Title</h2>
  {children}
  <button onClick={onClose}>Close</button>
</div>
);
}
```

Keyboard Shortcuts

```

function useKeyboardShortcut(key: string, callback: () => void, ctrlKey = false) {
  effect(() => {
    const handleKeyDown = (e: KeyboardEvent) => {
      if (e.key === key && (!ctrlKey || e.ctrlKey || e.metaKey)) {
        e.preventDefault();
        callback();
      }
    };
  });

  document.addEventListener('keydown', handleKeyDown);

  return () => {
    document.removeEventListener('keydown', handleKeyDown);
  };
}

function Editor() {
  const content = signal('');

  useKeyboardShortcut('s', () => save(), true); // Ctrl/Cmd + S
  useKeyboardShortcut('Escape', () => cancel());

  return (
    <div>
      <div aria-label="Keyboard shortcuts">
        <p><kbd>Ctrl+S</kbd> to save</p>
        <p><kbd>Esc</kbd> to cancel</p>
      </div>

      <textarea
        value={content()}
        onInput={(e) => content.set(e.currentTarget.value)}
        aria-label="Editor content"
      />
    </div>
  );
}

```

Tab Index

```

// ✅ Correct tabindex usage
function Card({ title, children }: CardProps) {
  return (
    <div tabIndex={0} className="card">
      <h3>{title}</h3>
      {children}
    </div>
  );
}

// ❌ Avoid positive tabindex
<div tabIndex={1}>First</div> // Don't do this
<div tabIndex={2}>Second</div> // Don't do this

// ✅ Use natural tab order or tabIndex={0}
<div tabIndex={0}>Natural order</div>

```

Screen Reader Support

Descriptive Labels

```
// 🌐 Clear labels for screen readers
function SearchForm() {
  const query = signal('');

  return (
    <form role="search">
      <label htmlFor="search-input" className="sr-only">
        Search products
      </label>

      <input
        id="search-input"
        type="search"
        value={query()}
        onInput={(e) => query.set(e.currentTarget.value)}
        placeholder="Search...""
        aria-label="Search products"
      />

      <button type="submit" aria-label="Submit search">
        <SearchIcon aria-hidden="true" />
      </button>
    </form>
  );
}

}


```

Live Regions

```
function SearchResults() {
  const results = signal<Product[]>([]);
  const loading = signal(false);

  return (
    <div>
      {/* Announce status to screen readers */}
      <div
        role="status"
        aria-live="polite"
        aria-atomic="true"
        className="sr-only"
      >
        {loading() && 'Loading results...'}
        {!loading() && `Found ${results().length} results`}
      </div>

      {/* Visual results */}
      <div aria-label="Search results">
        {results().map(product => (
          <ProductCard key={product.id} product={product} />
        ))}
      </div>
    </div>
  );
}

}


```

Hiding Decorative Content

```
// 🌐 Hide decorative icons from screen readers
function Button({ label, icon }: ButtonProps) {
  return (
    <button>
      <span aria-hidden="true">{icon}</span>
      <span>{label}</span>
    </button>
  );
}

// 🌐 Hide purely visual elements
function Card() {
  return (
    <div>
      <div className="decorative-border" aria-hidden="true" />
      <h3>Card Title</h3>
      <p>Card content</p>
    </div>
  );
}


```

Color and Contrast

Color Contrast

```

/* WCAG AA: At Least 4.5:1 for normal text */
.text {
  color: #333;
  background: #fff;
  /* Contrast ratio: 12.6:1 ✓ */
}

/* WCAG AAA: At Least 7:1 for normal text */
.text-high-contrast {
  color: #000;
  background: #fff;
  /* Contrast ratio: 21:1 ✓ */
}

/* Insufficient contrast */
.text-low-contrast {
  color: #999;
  background: #fff;
  /* Contrast ratio: 2.8:1 X */
}

```

Don't Rely on Color Alone

```

// Color only
function Status({ status }: { status: 'success' | 'error' }) {
  return (
    <span className={status === 'success' ? 'green' : 'red'}>
      {status}
    </span>
  );
}

// Color + icon + text
function AccessibleStatus({ status }: { status: 'success' | 'error' }) {
  return (
    <span className={`status-${status}`}>
      {status === 'success' ? (
        <>
          <CheckIcon aria-hidden="true" />
          <span>Success</span>
        </>
      ) : (
        <>
          <ErrorIcon aria-hidden="true" />
          <span>Error</span>
        </>
      )}
    </span>
  );
}

```

Forms

Accessible Forms

```

function ContactForm() {
  const name = signal('');
  const email = signal('');
  const message = signal('');
  const errors = signal<Record<string, string>>({});

  const handleSubmit = async (e: Event) => {
    e.preventDefault();

    const newErrors: Record<string, string> = {};

    if (!name()) newErrors.name = 'Name is required';
    if (!email()) newErrors.email = 'Email is required';
    if (!message()) newErrors.message = 'Message is required';

    errors.set(newErrors);

    if (Object.keys(newErrors).length === 0) {
      await submitForm({ name: name(), email: email(), message: message() });
    }
  };

  return (
    <form onSubmit={handleSubmit} noValidate>
      {/* Name field */}
      <div>
        <label htmlFor="name">
          Name <span aria-label="required">*</span>
        </label>
        <input
          id="name"
          ...
        </input>
      </div>
      ...
    </form>
  );
}

```

```

        type="text"
        value={name()}
        onInput={(e) => name.set(e.currentTarget.value)}
        aria-required="true"
        aria-invalid={!errors().name}
        aria-describedby={errors().name ? 'name-error' : undefined}
    />
    {errors().name && (
        <div id="name-error" role="alert">
            {errors().name}
        </div>
    )}
</div>

{/* Email field */}
<div>
    <label htmlFor="email">
        Email <span aria-label="required">*</span>
    </label>
    <input
        id="email"
        type="email"
        value={email()}
        onInput={(e) => email.set(e.currentTarget.value)}
        aria-required="true"
        aria-invalid={!errors().email}
        aria-describedby={errors().email ? 'email-error' : undefined}
    />
    {errors().email && (
        <div id="email-error" role="alert">
            {errors().email}
        </div>
    )}
</div>

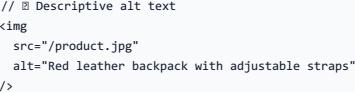
{/* Message field */}
<div>
    <label htmlFor="message">
        Message <span aria-label="required">*</span>
    </label>
    <textarea
        id="message"
        value={message()}
        onInput={(e) => message.set(e.currentTarget.value)}
        aria-required="true"
        aria-invalid={!errors().message}
        aria-describedby={errors().message ? 'message-error' : undefined}
    />
    {errors().message && (
        <div id="message-error" role="alert">
            {errors().message}
        </div>
    )}
</div>

<button type="submit">Send Message</button>
</form>
);
}
}

```

Images

[Alt Text](#)

```
// 📸 Descriptive alt text

// 📸 Empty alt for decorative images

// 📸 Complex images
<figure>
  <img alt="Bar chart showing sales increase from $10k to $50k over 6 months" data-bbox="164 224 488 264"/>
  <figcaption>
    Detailed description: Sales started at $10,000 in January, increased steadily to $50,000 by June...
  </figcaption>
</figure>
```

Responsive Design

Font Sizing

```
/* 📸 Use relative units */
body {
  font-size: 16px; /* Base */
}

h1 {
  font-size: 2rem; /* 32px, scales with user preferences */
}

p {
  font-size: 1rem; /* 16px */
}

/* 📸 Avoid fixed px for body text */
p {
  font-size: 14px; /* Doesn't scale */
}
```

Touch Targets

```
/* 📸 Minimum 44x44px touch targets */
.button {
  min-width: 44px;
  min-height: 44px;
  padding: 12px 24px;
}

/* 📸 Adequate spacing between targets */
.button-group button {
  margin: 8px;
}
```

Focus Indicators

Visible Focus States

```
/* 📸 Never remove focus outlines */
*:focus {
  outline: none; /* DON'T DO THIS */
}

/* 📸 Custom focus styles */
button:focus {
  outline: 2px solid #0066cc;
  outline-offset: 2px;
}

/* 📸 Focus-visible for mouse vs keyboard */
button:focus-visible {
  outline: 2px solid #0066cc;
  outline-offset: 2px;
}
```

Testing Accessibility

Manual Testing

```
// Keyboard testing checklist:  
// ☐ Can you navigate with Tab?  
// ☐ Can you activate with Enter/Space?  
// ☐ Can you escape modals with Esc?  
// ☐ Is focus visible?  
// ☐ Is focus trapped in modals?  
// ☐ Is tab order logical?  
  
// Screen reader testing:  
// ☐ Test with NVDA (Windows)  
// ☐ Test with JAWS (Windows)  
// ☐ Test with VoiceOver (Mac/iOS)  
// ☐ Test with TalkBack (Android)
```

Automated Testing

```
import { render } from '@testing-library/react';  
import { axe, toHaveNoViolations } from 'jest-axe';  
  
expect.extend(toHaveNoViolations);  
  
describe('Button accessibility', () => {  
  it('has no accessibility violations', async () => {  
    const { container } = render(  
      <Button label="Click me" onClick={() => {}} />  
    );  
  
    const results = await axe(container);  
    expect(results).toHaveNoViolations();  
  });  
});
```

Summary

Accessibility Best Practices:

- ☐ Use semantic HTML elements
- ☐ Add ARIA labels and roles where needed
- ☐ Ensure keyboard navigation works
- ☐ Manage focus properly
- ☐ Provide sufficient color contrast
- ☐ Don't rely on color alone
- ☐ Make forms accessible with labels and error messages
- ☐ Add meaningful alt text to images
- ☐ Use relative font sizes
- ☐ Maintain visible focus indicators
- ☐ Test with keyboard and screen readers
- ☐ Run automated accessibility tests
- ☐ Follow WCAG 2.1 Level AA guidelines

Next: [Production →](#)

State Management Patterns

Comprehensive guide to managing state at all scales in PhilJS applications, from local component state to complex global patterns.

Table of Contents

1. [State Categories](#)
2. [Local Component State](#)
3. [Shared State Patterns](#)
4. [Global State with Context](#)
5. [Store Patterns](#)
6. [State Persistence](#)
7. [State Synchronization](#)
8. [Advanced Patterns](#)

State Categories

Understanding where state should live is crucial for maintainable applications:

- **Local State:** Single component scope
- **Shared State:** Multiple components (siblings, parent-child)
- **Global State:** Application-wide access
- **Persistent State:** Survives page reloads
- **Synchronized State:** Connected to external sources

Local Component State

State that belongs to a single component. This is the most common and simplest form of state management.

Basic Signal State

```

function Counter() {
  const count = signal(0);

  return (
    <div>
      <p>Count: {count()}</p>
      <button onClick={() => count.set(count() + 1)}>Increment</button>
    </div>
  );
}

```

Multiple Related Signals

```

function UserProfile() {
  const firstName = signal('John');
  const lastName = signal('Doe');
  const email = signal('john.doe@example.com');
  const age = signal(30);

  // Computed full name
  const fullName = memo(() => `${firstName()} ${lastName()}`);

  const handleSubmit = () => {
    console.log('Saving:', {
      firstName: firstName(),
      lastName: lastName(),
      email: email(),
      age: age()
    });
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        value={firstName}
        onChange={(e) => firstName.set(e.currentTarget.value)}
        placeholder="First Name"
      />
      <input
        type="text"
        value={lastName}
        onChange={(e) => lastName.set(e.currentTarget.value)}
        placeholder="Last Name"
      />
      <input
        type="email"
        value={email}
        onChange={(e) => email.set(e.currentTarget.value)}
        placeholder="Email"
      />
      <input
        type="number"
        value={age}
        onChange={(e) => age.set(Number(e.currentTarget.value))}
        placeholder="Age"
      />
      <p>Full Name: {fullName()}</p>
      <button type="submit">Save Profile</button>
    </form>
  );
}

```

Object State Pattern

```

interface FormData {
  username: string;
  email: string;
  bio: string;
}

function ProfileForm() {
  const formData = signal<FormData>({
    username: '',
    email: '',
    bio: ''
  });

  const isValid = memo(() => {
    const data = formData();
    return data.username.length >= 3 &&
      data.email.includes('@') &&
      data.bio.length <= 500;
  });

  const updateField = <K extends keyof FormData>(
    field: K,
    value: FormData[K]
  ) => {
    formData.set({ ...formData(), [field]: value });
  };

  return (
    <form>
      <input
        value={formData().username}
        onInput={(e) => updateField('username', e.currentTarget.value)}
        placeholder="Username"
      />
      <input
        value={formData().email}
        onInput={(e) => updateField('email', e.currentTarget.value)}
        placeholder="Email"
      />
      <textarea
        value={formData().bio}
        onInput={(e) => updateField('bio', e.currentTarget.value)}
        placeholder="Bio"
      />
      <p>{formData().bio.length}/500 characters</p>
      <button type="submit" disabled={!isValid}>
        Submit
      </button>
    </form>
  );
}

```

Use local state when: - State is only needed in one component - No other components need to access it - State doesn't need to persist beyond component lifetime - The component is self-contained

Shared State Patterns

When multiple components need access to the same state, you have several options depending on the component relationship.

Lifted State (Parent-Child)

State shared between multiple components via props. Lift state to the closest common ancestor.

```

function ParentComponent() {
  const searchTerm = signal('');
  const items = signal([
    { id: 1, name: 'Apple', category: 'fruit' },
    { id: 2, name: 'Banana', category: 'fruit' },
    { id: 3, name: 'Carrot', category: 'vegetable' }
  ]);

  const filteredItems = memo(() =>
    items().filter(item =>
      item.name.toLowerCase().includes(searchTerm().toLowerCase())
    )
  );

  return (
    <div>
      <SearchInput value={searchTerm()} onChange={searchTerm.set} />
      <ItemList items={filteredItems()} />
      <ItemCount count={filteredItems().length} />
    </div>
  );
}

function SearchInput({ value, onChange }: {
  value: Signal<string>;
  onChange: (value: string) => void;
}) {
  return (
    <input
      type="search"
      value={value}
      onChange={(e) => onChange(e.currentTarget.value)}
      placeholder="Search items..." />
  );
}

function ItemList({ items }: { items: Memo<Item[]> }) {
  return (
    <ul>
      {() => items().map(item => (
        <li key={item.id}>{item.name}</li>
      ))}
    </ul>
  );
}

function ItemCount({ count }: { count: number }) {
  return <p>Found {count} items</p>;
}

```

Custom Hook Pattern

Encapsulate related state logic in reusable functions.

```

// hooks/useToggle.ts
function useToggle(initialValue = false) {
  const value = signal(initialValue);

  const toggle = () => value.set(!value());
  const setTrue = () => value.set(true);
  const setFalse = () => value.set(false);

  return {
    value,
    toggle,
    setTrue,
    setFalse
  };
}

// hooks/usePagination.ts
function usePagination<T>(items: T[], pageSize = 10) {
  const currentPage = signal(1);

  const totalPages = memo(() =>
    Math.ceil(items.length / pageSize)
  );

  const paginatedItems = memo(() => {
    const start = (currentPage() - 1) * pageSize;
    return items.slice(start, start + pageSize);
  });

  const nextPage = () => {
    if (currentPage() < totalPages()) {
      currentPage.set(currentPage() + 1);
    }
  };

  const prevPage = () => {
    if (currentPage() > 1) {
      currentPage.set(currentPage() - 1);
    }
  };

  const goToPage = (page: number) => {
    const safePage = Math.max(1, Math.min(page, totalPages()));
    currentPage.set(safePage);
  };

  return {
    currentPage,
    totalPages,
    paginatedItems,
    nextPage,
    prevPage,
    goToPage
  };
}

// Usage in component
function ProductList({ products }: { products: Product[] }) {
  const {
    paginatedItems,
    currentPage,
    totalPages,
    nextPage,
    prevPage
  } = usePagination(products, 12);

  return (
    <div>
      <div class="product-grid">
        {() => paginatedItems().map(product => (
          <ProductCard key={product.id} product={product} />
        ))}
      </div>
      <div class="pagination">
        <button onClick={prevPage} disabled={currentPage() === 1}>
          Previous
        </button>
        <span>Page {currentPage()} of {totalPages()}</span>
        <button onClick={nextPage} disabled={currentPage() === totalPages()}>
          Next
        </button>
      </div>
    </div>
  );
}

```

Module-Level Shared State

For state shared across unrelated components without context overhead.

```
// state/modal.ts
import { signal } from '@philjs/core';

interface ModalState {
  isOpen: boolean;
  title: string;
  content: string | null;
}

const modalState = signal<ModalState>({
  isOpen: false,
  title: '',
  content: null
});

export const modal = {
  state: modalState,

  open: (title: string, content: string) => {
    modalState.set({
      isOpen: true,
      title,
      content
    });
  },
  close: () => {
    modalState.set({
      isOpen: false,
      title: '',
      content: null
    });
  },
  isOpen: () => modalState().isOpen
};

// Usage in any component
import { modal } from './state/modal';

function ConfirmButton() {
  const handleClick = () => {
    modal.open('Confirm Action', 'Are you sure you want to continue?');
  };

  return <button onClick={handleClick}>Delete</button>;
}

function Modal() {
  const state = modal.state;

  return (
    <>
      {() => state().isOpen && (
        <div class="modal-overlay" onClick={modal.close}>
          <div class="modal-content" onClick={(e) => e.stopPropagation()}>
            <h2>{state().title}</h2>
            <p>{state().content}</p>
            <button onClick={modal.close}>Close</button>
          </div>
        </div>
      )}
    </>
  );
}
}
```

Use shared state patterns when: - Multiple sibling components need same state - Parent needs to coordinate children - State logic is reusable across components - State doesn't need to be truly global

Global State with Context

PhilJS provides a powerful Context API for dependency injection and global state management without prop drilling.

Basic Context Pattern

```

// contexts/ThemeContext.tsx
import { createContext, useContext } from '@philjs/core';

type Theme = 'light' | 'dark';

interface ThemeContextValue {
  theme: Theme;
  setTheme: (theme: Theme) => void;
}

const ThemeContext = createContext<ThemeContextValue>({
  theme: 'light',
  setTheme: () => {}
});

export function ThemeProvider({ children }: { children: any }) {
  const theme = signal<Theme>('light');

  const contextValue = {
    theme: theme(),
    setTheme: (newTheme: Theme) => theme.set(newTheme)
  };

  return (
    <ThemeContext.Provider value={contextValue}>
      {children}
    </ThemeContext.Provider>
  );
}

export function useTheme() {
  return useContext(ThemeContext);
}

// Usage
function App() {
  return (
    <ThemeProvider>
      <Header />
      <MainContent />
    </ThemeProvider>
  );
}

function ThemeToggle() {
  const { theme, setTheme } = useTheme();

  return (
    <button onClick={() => setTheme(theme === 'light' ? 'dark' : 'light')}>
      Current: {theme}
    </button>
  );
}

```

Signal Context Pattern (Reactive)

```

// contexts/UserContext.tsx
import { createSignalContext, signal, memo } from '@philjs/core';

interface User {
  id: string;
  name: string;
  email: string;
  role: 'user' | 'admin';
}

function createUserContext() {
  const user = signal<User | null>(null);
  const isAuthenticated = memo(() => user() !== null);
  const isAdmin = memo(() => user()?.role === 'admin');

  const login = async (credentials: { email: string; password: string }) => {
    // Simulate API call
    const userData = await fetch('/api/auth/login', {
      method: 'POST',
      body: JSON.stringify(credentials)
    }).then(r => r.json());

    user.set(userData);
  };

  const logout = () => {
    user.set(null);
  };

  const updateProfile = (updates: Partial<User>) => {
    if (user()) {
      user.set({ ...user()!, ...updates });
    }
  };
}

```

```

        }
    });

    return {
        user,
        isAuthenticated,
        isAdmin,
        login,
        logout,
        updateProfile
    };
}

const UserContext = createContext(createUserContext());

export function UserProvider({ children }: { children: any }) {
    const userState = createUserContext();

    return (
        <UserContext.Provider value={userState}>
            {children}
        </UserContext.Provider>
    );
}

export function useUser() {
    return useContext(UserContext);
}

// Usage in components
function Header() {
    const { user, isAuthenticated, logout } = useUser();

    return (
        <header>
            {(() => isAuthenticated() ? (
                <>
                    <span>Welcome, {user()?.name}</span>
                    <button onClick={logout}>Logout</button>
                </>
            ) : (
                <a href="/login">Login</a>
            )}
        </header>
    );
}

function ProtectedRoute({ children }: { children: any }) {
    const { isAuthenticated } = useUser();

    return (
        <>
            {(() => isAuthenticated() ? children : <Navigate to="/login" />)}
        </>
    );
}

```

Multiple Contexts Pattern

```

// contexts/AppContext.tsx
import { combineProviders } from '@philijs/core';
import { UserProvider } from './UserContext';
import { ThemeProvider } from './ThemeContext';
import { NotificationProvider } from './NotificationCenter';

export function AppProviders({ children }: { children: any }) {
    const Providers = combineProviders(
        { Provider: UserProvider, value: undefined },
        { Provider: ThemeProvider, value: undefined },
        { Provider: NotificationProvider, value: undefined }
    );

    return <Providers>{children}</Providers>;
}

// Usage
function App() {
    return (
        <AppProviders>
            <Router />
        </AppProviders>
    );
}

```

Theme Context with CSS Variables

```

// contexts/ThemeContext.tsx
import { createThemeContext, signal, effect } from '@philjs/core';

interface AppTheme {
  primaryColor: string;
  secondaryColor: string;
  backgroundColor: string;
  textColor: string;
  borderRadius: string;
}

const lightTheme: AppTheme = {
  primaryColor: '#a29bfe',
  secondaryColor: '#74b9ff',
  backgroundColor: '#1a1a1a',
  textColor: '#e0e0e0',
  borderRadius: '8px'
};

const darkTheme: AppTheme = {
  primaryColor: '#a29bfe',
  secondaryColor: '#74b9ff',
  backgroundColor: '#1a1a1a',
  textColor: '#e0e0e0',
  borderRadius: '8px'
};

const themeContext = createThemeContext(lightTheme);

export function ThemeProvider({ children }: { children: any }) {
  const isDark = signal(false);
  const currentTheme = memo(() => isDark() ? darkTheme : lightTheme);

  // Apply theme to document
  effect(() => {
    const theme = currentTheme();
    Object.entries(theme).forEach(([key, value]) => {
      document.documentElement.style.setProperty(`--${key}`, value);
    });
  });

  const toggleTheme = () => isDark.set(!isDark());
}

return (
  <themeContext.Provider theme={currentTheme()}>
    <div data-theme={isDark() ? 'dark' : 'light'}>
      <button onClick={toggleTheme}>
        Toggle Theme
      </button>
      {children}
    </div>
  </themeContext.Provider>
);
}

export const useTheme = themeContext.useTheme;

```

Global State Store Pattern

State accessible throughout the application without context overhead.

```

// stores/userStore.ts
import { signal, memo } from '@philjs/core';

function createUserStore() {
  const user = signal<User | null>(null);
  const isAuthenticated = memo(() => user() !== null);
  const isAdmin = memo(() => user()?.role === 'admin');

  const login = async (credentials: Credentials) => {
    const userData = await authService.login(credentials);
    user.set(userData);
  };

  const logout = () => {
    user.set(null);
  };

  return {
    user,
    isAuthenticated,
    isAdmin,
    login,
    logout
  };
}

export const userStore = createUserStore();

// Usage in components
import { userStore } from '@/stores/userStore';

function Header() {
  const { user, logout } = userStore;

  return (
    <header>
      <span>Welcome, {user()?.name}</span>
      <button onClick={logout}>Logout</button>
    </header>
  );
}

```

Use global state when: - State needed across many unrelated components - Deep prop drilling would be cumbersome - State should persist across routes - The state is truly application-wide (user, theme, language)

Store Patterns

Simple Store

```

// stores/themeStore.ts
import { signal } from '@philjs/core';

const theme = signal<'light' | 'dark'>('light');

export const themeStore = {
  theme,
  toggle: () => {
    theme.set(theme() === 'light' ? 'dark' : 'light');
  }
};

```

Factory Store

```

// stores/todoStore.ts
interface Todo {
  id: string;
  text: string;
  completed: boolean;
}

function createTodoStore() {
  const todos = signal<Todo[]>([[]]);
  const filter = signal<'all' | 'active' | 'completed'>('all');

  const filteredTodos = memo(() => {
    const allTodos = todos();

    switch (filter()) {
      case 'active':
        return allTodos.filter(t => !t.completed);
      case 'completed':
        return allTodos.filter(t => t.completed);
      default:
        return allTodos;
    }
  });

  const activeCount = memo(() =>
    todos().filter(t => !t.completed).length
  );

  const addTodo = (text: string) => {
    const newTodo: Todo = {
      id: crypto.randomUUID(),
      text,
      completed: false
    };

    todos.set([...todos(), newTodo]);
  };

  const toggleTodo = (id: string) => {
    todos.set(
      todos().map(todo =>
        todo.id === id
          ? { ...todo, completed: !todo.completed }
          : todo
      )
    );
  };

  const deleteTodo = (id: string) => {
    todos.set(todos().filter(t => t.id !== id));
  };

  const clearCompleted = () => {
    todos.set(todos().filter(t => !t.completed));
  };

  return {
    // State
    todos,
    filter,
    // Computed
    filteredTodos,
    activeCount,
    // Actions
    addTodo,
    toggleTodo,
    deleteTodo,
    clearCompleted
  };
}

export const todoStore = createTodoStore();

```

Async Store

```

// stores/productsStore.ts
interface Product {
  id: string;
  name: string;
  price: number;
}

function createProductsStore() {
  const products = signal<Product[]>([[]]);
  const loading = signal(false);
  const error = signal<string | null>(null);

  const fetchProducts = async () => {
    loading.set(true);
    error.set(null);

    try {
      const data = await api.get<Product[]>('/products');
      products.set(data);
    } catch (err) {
      error.set(err.message);
    } finally {
      loading.set(false);
    }
  };

  const createProduct = async (product: Omit<Product, 'id'>) => {
    try {
      const newProduct = await api.post<Product>('/products', product);
      products.set([...products(), newProduct]);
      return newProduct;
    } catch (err) {
      error.set(err.message);
      throw err;
    }
  };

  const updateProduct = async (id: string, updates: Partial<Product>) => {
    try {
      const updated = await api.patch<Product>(`/products/${id}`, updates);
      products.set(
        products().map(p => p.id === id ? updated : p)
      );
      return updated;
    } catch (err) {
      error.set(err.message);
      throw err;
    }
  };

  const deleteProduct = async (id: string) => {
    try {
      await api.delete(`/products/${id}`);
      products.set(products().filter(p => p.id !== id));
    } catch (err) {
      error.set(err.message);
      throw err;
    }
  };
}

return {
  products,
  loading,
  error,
  fetchProducts,
  createProduct,
  updateProduct,
  deleteProduct
};

export const productsStore = createProductsStore();

```

State Persistence

State that survives page reloads and browser sessions.

LocalStorage Persistence

```

// stores/settingsStore.ts
import { signal, effect } from '@philjs/core';

interface Settings {
  theme: 'light' | 'dark';
  language: string;
  notifications: boolean;
  fontSize: number;
}

const defaultSettings: Settings = {
  theme: 'light',
  language: 'en',
  notifications: true,
  fontSize: 16
};

function createSettingsStore() {
  // Load from localStorage with error handling
  const loadSettings = (): Settings => {
    try {
      const stored = localStorage.getItem('settings');
      if (stored) {
        return { ...defaultSettings, ...JSON.parse(stored) };
      }
    } catch (error) {
      console.warn('Failed to load settings:', error);
    }
    return defaultSettings;
  };

  const settings = signal<Settings>(loadSettings());

  // Persist to localStorage with debouncing
  let timeoutId: number | undefined;
  effect(() => {
    clearTimeout(timeoutId);
    timeoutId = setTimeout(() => {
      try {
        localStorage.setItem('settings', JSON.stringify(settings()));
      } catch (error) {
        console.error('Failed to save settings:', error);
      }
    }, 300) as unknown as number;
  });

  const updateSettings = (updates: Partial<Settings>) => {
    settings.set({ ...settings(), ...updates });
  };

  const reset = () => {
    settings.set(defaultSettings);
    localStorage.removeItem('settings');
  };

  return {
    settings,
    updateSettings,
    reset
  };
}

export const settingsStore = createSettingsStore();

```

Generic Persistent Signal

```

// utils/persistentSignal.ts
import { signal, effect, Signal } from '@philjs/core';

interface PersistOptions<T> {
  key: string;
  storage?: Storage;
  serialize?: (value: T) => string;
  deserialize?: (value: string) => T;
  debounce?: number;
}

export function persistentSignal<T>(
  initialValue: T,
  options: PersistOptions<T>
): Signal<T> {
  const {
    key,
    storage = localStorage,
    serialize = JSON.stringify,
    deserialize = JSON.parse,
    debounce = 0
  } = options;

  // Load initial value from storage
  const loadValue = (): T => {
    try {
      const stored = storage.getItem(key);
      return stored ? deserialize(stored) : initialValue;
    } catch (error) {
      console.warn(`Failed to load ${key}:`, error);
      return initialValue;
    }
  };

  const sig = signal<T>(loadValue());

  // Persist changes to storage
  let timeoutId: number | undefined;
  effect(() => {
    const value = sig();

    if (debounce > 0) {
      clearTimeout(timeoutId);
      timeoutId = setTimeout(() => {
        try {
          storage.setItem(key, serialize(value));
        } catch (error) {
          console.error(`Failed to save ${key}:`, error);
        }
      }, debounce) as unknown as number;
    } else {
      try {
        storage.setItem(key, serialize(value));
      } catch (error) {
        console.error(`Failed to save ${key}:`, error);
      }
    }
  });

  return sig;
}

// Usage
const theme = persistentSignal<'light' | 'dark'>('light', {
  key: 'app-theme'
});

const recentSearches = persistentSignal<string[]>([], {
  key: 'recent-searches',
  debounce: 500
});

const userPreferences = persistentSignal({
  notifications: true,
  emailDigest: 'weekly',
  timezone: 'UTC'
}, {
  key: 'user-preferences',
  debounce: 1000
});

```

SessionStorage for Temporary State

```

// stores/formDraftStore.ts
import { signal, effect } from '@philjs/core';

interface FormDraft {
  title: string;

```

```

        ..
        content: string;
        lastSaved: number;
    }

    function createFormDraftStore() {
        const loadDraft = (): FormDraft | null => {
            try {
                const stored = sessionStorage.getItem('form-draft');
                return stored ? JSON.parse(stored) : null;
            } catch {
                return null;
            }
        };

        const draft = signal<FormDraft | null>(loadDraft());

        // Auto-save to sessionStorage
        effect(() => {
            const currentDraft = draft();
            if (currentDraft) {
                sessionStorage.setItem('form-draft', JSON.stringify({
                    ...currentDraft,
                    lastSaved: Date.now()
                }));
            } else {
                sessionStorage.removeItem('form-draft');
            }
        });

        const saveDraft = (title: string, content: string) => {
            draft.set({ title, content, lastSaved: Date.now() });
        };

        const clearDraft = () => {
            draft.set(null);
        };

        return {
            draft,
            saveDraft,
            clearDraft
        };
    }

    export const formDraftStore = createFormDraftStore();

    // Usage
    function BlogPostEditor() {
        const { draft, saveDraft, clearDraft } = formDraftStore;
        const title = signal('');
        const content = signal('');

        // Load draft on mount
        effect(() => {
            const savedDraft = draft();
            if (savedDraft) {
                title.set(savedDraft.title);
                content.set(savedDraft.content);
            }
        });

        // Auto-save every 5 seconds
        effect(() => {
            const interval = setInterval(() => {
                if (title() || content()) {
                    saveDraft(title(), content());
                }
            }, 5000);
        });

        return () => clearInterval(interval);
    });

    const handlePublish = () => {
        // Publish post...
        clearDraft(); // Clear after successful publish
    };

    return (
        <form>
            <input
                value={title}
                onInput={(e) => title.set(e.currentTarget.value)}
                placeholder="Post title"
            />
            <textarea
                value={content}
                onInput={(e) => content.set(e.currentTarget.value)}
                placeholder="Write your post..." />
        </form>
    );
}

```

```

        //}
        //{
        //() => draft() && (
        <p class="draft-status">
            Draft saved at {new Date(draft()!.lastSaved).toLocaleTimeString()}
        </p>
    )}
    <button type="button" onClick={handlePublish}>Publish</button>
</form>
);
}

```

IndexedDB for Large Data

```

// utils/indexedDBStore.ts
import { signal } from '@philjs/core';

class IndexedDBStore<T> {
    private db: IDBDatabase | null = null;
    private readonly dbName: string;
    private readonly storeName: string;

    constructor(dbName: string, storeName: string) {
        this.dbName = dbName;
        this.storeName = storeName;
    }

    async init(): Promise<void> {
        return new Promise((resolve, reject) => {
            const request = indexedDB.open(this.dbName, 1);

            request.onerror = () => reject(request.error);
            request.onsuccess = () => {
                this.db = request.result;
                resolve();
            };
        });

        request.onupgradeneeded = (event) => {
            const db = (event.target as IDBOpenDBRequest).result;
            if (!db.objectStoreNames.contains(this.storeName)) {
                db.createObjectStore(this.storeName);
            }
        };
    });
}

async get(key: string): Promise<T | undefined> {
    if (!this.db) await this.init();

    return new Promise((resolve, reject) => {
        const transaction = this.db!.transaction(this.storeName, 'readonly');
        const store = transaction.objectStore(this.storeName);
        const request = store.get(key);

        request.onerror = () => reject(request.error);
        request.onsuccess = () => resolve(request.result);
    });
}

async set(key: string, value: T): Promise<void> {
    if (!this.db) await this.init();

    return new Promise((resolve, reject) => {
        const transaction = this.db!.transaction(this.storeName, 'readwrite');
        const store = transaction.objectStore(this.storeName);
        const request = store.put(value, key);

        request.onerror = () => reject(request.error);
        request.onsuccess = () => resolve();
    });
}

async delete(key: string): Promise<void> {
    if (!this.db) await this.init();

    return new Promise((resolve, reject) => {
        const transaction = this.db!.transaction(this.storeName, 'readwrite');
        const store = transaction.objectStore(this.storeName);
        const request = store.delete(key);

        request.onerror = () => reject(request.error);
        request.onsuccess = () => resolve();
    });
}

// Usage for offline-first app
interface CachedData {
    products: Product[];
    lastFetch: number;
}

```

```

    }

const dbStore = new IndexedDBStore<CachedData>('myapp', 'cache');

function createProductCache() {
  const products = signal<Product[]>([[]]);
  const loading = signal(true);
  const isStale = signal(false);

  const CACHE_DURATION = 5 * 60 * 1000; // 5 minutes

  const loadProducts = async () => {
    loading.set(true);

    // Try to load from cache
    const cached = await dbStore.get('products');

    if (cached) {
      products.set(cached.products);
      const age = Date.now() - cached.lastFetch;
      isStale.set(age > CACHE_DURATION);
    }
  };

  // Fetch fresh data
  try {
    const freshData = await fetch('/api/products').then(r => r.json());
    products.set(freshData);

    // Update cache
    await dbStore.set('products', {
      products: freshData,
      lastFetch: Date.now()
    });

    isStale.set(false);
  } catch (error) {
    console.error('Failed to fetch products:', error);
    // Use cached data if available
  } finally {
    loading.set(false);
  }
};

return {
  products,
  loading,
  isStale,
  loadProducts
};
}

```

State Synchronization

Keeping state synchronized with external sources like WebSockets, Server-Sent Events, or polling.

WebSocket Synchronization

```

// stores/realtimeStore.ts
import { signal, effect } from '@philjs/core';

interface Message {
  id: string;
  userId: string;
  text: string;
  timestamp: number;
}

function createRealtimeStore() {
  const messages = signal<Message[]>([[]]);
  const connected = signal(false);
  const error = signal<string | null>(null);
  let ws: WebSocket | null = null;

  const connect = (url: string) => {
    try {
      ws = new WebSocket(url);

      ws.onopen = () => {
        connected.set(true);
        error.set(null);
        console.log('WebSocket connected');
      };
    }

    ws.onmessage = (event) => {
      const message = JSON.parse(event.data) as Message;
      // Add new message to the list
      ...
    };
  };
}

```

```

    messages.set(...messages(), message);
};

ws.onerror = (event) => {
  error.set('WebSocket error occurred');
  console.error('WebSocket error:', event);
};

ws.onclose = () => {
  connected.set(false);
  console.log('WebSocket disconnected');

  // Auto-reconnect after 3 seconds
  setTimeout(() => {
    if (!connected()) {
      connect(url);
    }
  }, 3000);
};

} catch (err) {
  error.set(err instanceof Error ? err.message : 'Connection failed');
}
};

const sendMessage = (text: string, userId: string) => {
  if (ws && ws.readyState === WebSocket.OPEN) {
    const message: Message = {
      id: crypto.randomUUID(),
      userId,
      text,
      timestamp: Date.now()
    };

    ws.send(JSON.stringify(message));
  } else {
    error.set('Not connected to server');
  }
};

const disconnect = () => {
  if (ws) {
    ws.close();
    ws = null;
  }
};

return {
  messages,
  connected,
  error,
  connect,
  sendMessage,
  disconnect
};
}

export const realtimeStore = createRealtimeStore();

// Usage
function ChatRoom() {
  const { messages, connected, sendMessage, connect } = realtimeStore;
  const inputValue = signal('');

  // Connect on mount
  effect(() => {
    connect('wss://api.example.com/chat');
    return () => realtimeStore.disconnect();
  });

  const handleSend = () => {
    if (inputValue().trim()) {
      sendMessage(inputValue(), 'user-123');
      inputValue.set('');
    }
  };

  return (
    <div class="chat-room">
      <div class="connection-status">
        {() => connected() ? 'Connected' : 'Disconnected'}
      </div>

      <div class="messages">
        {() => messages().map(msg => (
          <div key={msg.id} class="message">
            <strong>{msg.userId}</strong> {msg.text}
          </div>
        )));
      </div>
    </div>
  );
}

```

```

<div class="input-area">
  <input
    value={inputValue}
    onInput={(e) => inputValue.set(e.currentTarget.value)}
    onKeyPress={(e) => e.key === 'Enter' && handleSend()}
    placeholder="Type a message..." />
  <button onClick={handleSend}>Send</button>
</div>
</div>
);
}

```

Server-Sent Events (SSE)

```

// stores/notificationStore.ts
import { signal } from '@philjs/core';

interface Notification {
  id: string;
  type: 'info' | 'success' | 'warning' | 'error';
  message: string;
  timestamp: number;
}

function createNotificationStore() {
  const notifications = signal<Notification[]>([]);
  const connected = signal(false);
  let eventSource: EventSource | null = null;

  const connect = (url: string) => {
    eventSource = new EventSource(url);

    eventSource.onopen = () => {
      connected.set(true);
    };

    eventSource.addEventListener('notification', (event) => {
      const notification = JSON.parse(event.data) as Notification;
      notifications.set([...notifications(), notification]);
    });

    // Auto-remove after 5 seconds
    setTimeout(() => {
      remove(notification.id);
    }, 5000);
  });

  eventSource.onerror = () => {
    connected.set(false);
    eventSource?.close();
  };

  // Reconnect after 5 seconds
  setTimeout(() => connect(url), 5000);
};

const remove = (id: string) => {
  notifications.set(notifications().filter(n => n.id !== id));
};

const clear = () => {
  notifications.set([]);
};

const disconnect = () => {
  eventSource?.close();
  eventSource = null;
  connected.set(false);
};

return {
  notifications,
  connected,
  connect,
  remove,
  clear,
  disconnect
};
}

export const notificationStore = createNotificationStore();

```

Polling Pattern

```

// stores/pollingStore.ts
import { signal, effect } from '@philjs/core';

```

```

interface PollingOptions {
  interval: number;
  immediate?: boolean;
}

function createPollingStore<T>(
  fetcher: () => Promise<T>,
  options: PollingOptions
) {
  const { interval, immediate = true } = options;

  const data = signal<T | null>(null);
  const loading = signal(false);
  const error = signal<Error | null>(null);
  const enabled = signal(true);

  let intervalId: number | undefined;

  const fetch = async () => {
    if (!enabled()) return;

    loading.set(true);
    error.set(null);

    try {
      const result = await fetcher();
      data.set(result);
    } catch (err) {
      error.set(err instanceof Error ? err : new Error(String(err)));
    } finally {
      loading.set(false);
    }
  };

  const start = () => {
    enabled.set(true);

    if (immediate) {
      fetch();
    }

    intervalId = setInterval(fetch, interval) as unknown as number;
  };

  const stop = () => {
    enabled.set(false);
    if (intervalId) {
      clearInterval(intervalId);
      intervalId = undefined;
    }
  };

  const refresh = () => {
    fetch();
  };

  return {
    data,
    loading,
    error,
    enabled,
    start,
    stop,
    refresh
  };
}

// Usage: Poll for new orders every 10 seconds
const ordersStore = createPollingStore(
  async () => {
    const response = await fetch('/api/orders/pending');
    return response.json();
  },
  { interval: 10000 }
);

function OrdersMonitor() {
  const { data, loading, start, stop } = ordersStore;

  // Start polling on mount
  effect(() => {
    start();
    return () => stop();
  });

  return (
    <div>
      {() => loading() ? (
        <div>Loading orders...</div>
      ) : (
        <ul>
          {data.map(order => (
            <li>{order}</li>
          ))}
        </ul>
      )}
    </div>
  );
}

```

```

        <p>Loading orders...</p>
      ) : (
        <ul>
          {data()?.map((order: any) => (
            <li key={order.id}>{order.name}</li>
          )));
        </ul>
      )
    </div>
  );
}

```

Cross-Tab Synchronization

```

// stores/crossTabStore.ts
import { signal, effect } from '@philjs/core';

interface CrossTabState<T> {
  key: string;
  data: T;
}

function createCrossTabStore<T>(key: string, initialValue: T) {
  const data = signal<T>(initialValue);

  // Load initial value from localStorage
  const stored = localStorage.getItem(key);
  if (stored) {
    try {
      data.set(JSON.parse(stored));
    } catch {
      // Invalid data, use initial value
    }
  }

  // Save to localStorage on change
  effect(() => {
    localStorage.setItem(key, JSON.stringify(data()));
  });

  // Listen for changes from other tabs
  const handleStorageChange = (event: StorageEvent) => {
    if (event.key === key && event.newValue) {
      try {
        data.set(JSON.parse(event.newValue));
      } catch {
        // Invalid data
      }
    }
  };
}

window.addEventListener('storage', handleStorageChange);

// Cleanup
effect(() => {
  return () => {
    window.removeEventListener('storage', handleStorageChange);
  };
});

return data;
}

// Usage: Synchronize cart across tabs
const cartItems = createCrossTabStore<CartItem[]>('cart', []);

function ShoppingCart() {
  const items = cartItems;

  const addItem = (item: CartItem) => {
    items.set([...items(), item]);
  };

  return (
    <div>
      <h2>Cart ({items().length} items)</h2>
      <ul>
        {() => items().map(item => (
          <li key={item.id}>{item.name}</li>
        ))}
      </ul>
    </div>
  );
}

```

BroadcastChannel API

```

// stores/broadcastStore.ts
import { signal } from '@philjs/core';

interface BroadcastMessage<T> {
  type: string;
  payload: T;
}

function createBroadcastStore<T>(channelName: string) {
  const data = signal<T | null>(null);
  const channel = new BroadcastChannel(channelName);

  channel.onmessage = (event: MessageEvent<BroadcastMessage<T>>) => {
    const { type, payload } = event.data;

    if (type === 'update') {
      data.set(payload);
    }
  };

  const broadcast = (payload: T) => {
    data.set(payload);
    channel.postMessage({ type: 'update', payload });
  };

  const close = () => {
    channel.close();
  };

  return {
    data,
    broadcast,
    close
  };
}

// Usage: Sync authentication across tabs
const authStore = createBroadcastStore<User | null>('auth');

function logout() {
  // This will log out all tabs
  authStore.broadcast(null);
}

```

State Composition

Combining Stores

```

// stores/cartStore.ts
import { userStore } from './userStore';

function createCartStore() {
  const items = signal<CartItem[]>([]);

  const total = memo(() =>
    items().reduce((sum, item) => sum + item.price * item.quantity, 0)
  );

  const discount = memo(() => {
    const user = userStore.user();
    if (!user) return 0;

    // Premium users get 10% discount
    if (user.isPremium) {
      return total() * 0.1;
    }

    return 0;
  });

  const finalTotal = memo(() => total() - discount());

  const addItem = (product: Product) => {
    const existing = items().find(item => item.id === product.id);

    if (existing) {
      items.set(
        items().map(item =>
          item.id === product.id
            ? { ...item, quantity: item.quantity + 1 }
            : item
        )
      );
    } else {
      items.set([...items(), {
        id: product.id,
        name: product.name,
        price: product.price,
        quantity: 1
      }]);
    }
  };

  return {
    items,
    total,
    discount,
    finalTotal,
    addItem
  };
}

export const cartStore = createCartStore();

```

Derived Stores

```

// stores/analyticsStore.ts
import { productsStore } from './productsStore';
import { cartStore } from './cartStore';

function createAnalyticsStore() {
  const totalRevenue = memo(() => {
    // Compute from cart history
    return cartStore.finalTotal();
  });

  const popularProducts = memo(() => {
    const products = productsStore.products();
    // Sort by popularity
    return products.sort((a, b) => b.views - a.views).slice(0, 5);
  });

  const averageOrderValue = memo(() => {
    // Calculate from orders
    return totalRevenue() / orderCount();
  });

  return {
    totalRevenue,
    popularProducts,
    averageOrderValue
  };
}

export const analyticsStore = createAnalyticsStore();

```

State Machines

Finite State Machine

```
type State = 'idle' | 'loading' | 'success' | 'error';

type Event =
  | { type: 'FETCH' }
  | { type: 'SUCCESS'; data: any }
  | { type: 'ERROR'; error: string }
  | { type: 'RETRY' };

function createFetchMachine() {
  const state = signal<State>('idle');
  const data = signal<any>(null);
  const error = signal<string | null>(null);

  const send = (event: Event) => {
    const current = state();

    switch (event.type) {
      case 'FETCH':
        if (current === 'idle' || current === 'error') {
          state.set('loading');
          error.set(null);
        }
        break;

      case 'SUCCESS':
        if (current === 'loading') {
          state.set('success');
          data.set(event.data);
        }
        break;

      case 'ERROR':
        if (current === 'loading') {
          state.set('error');
          error.set(event.error);
        }
        break;

      case 'RETRY':
        if (current === 'error') {
          state.set('loading');
          error.set(null);
        }
        break;
    }
  };
}

return {
  state,
  data,
  error,
  send
};

}

// Usage
function DataComponent() {
  const machine = createFetchMachine();

  effect(async () => {
    if (machine.state() === 'loading') {
      try {
        const result = await fetchData();
        machine.send({ type: 'SUCCESS', data: result });
      } catch (err) {
        machine.send({ type: 'ERROR', error: err.message });
      }
    }
  });

  return (
    <div>
      {machine.state() === 'idle' && (
        <button onClick={() => machine.send({ type: 'FETCH' })}>
          Load Data
        </button>
      )}
      {machine.state() === 'loading' && <Spinner />}
      {machine.state() === 'success' && (
        <DataView data={machine.data()} />
      )}
    </div>
  );
}
```

```

{machine.state() === 'error' && (
  <div>
    <p>Error: {machine.error()}</p>
    <button onClick={() => machine.send({ type: 'RETRY' })}>
      Retry
    </button>
  </div>
)
);
}

```

Optimistic Updates

```

function createOptimisticStore() {
  const items = signal<Item[]>([[]]);
  const pendingUpdates = signal<Map<string, Item>>(new Map());

  const addItem = async (item: Omit<Item, 'id'>) => {
    const tempId = crypto.randomUUID();
    const optimisticItem = { ...item, id: tempId };

    // Optimistically add to UI
    items.set([...items(), optimisticItem]);
    pendingUpdates.set(new Map(pendingUpdates()).set(tempId, optimisticItem));

    try {
      // Send to server
      const serverItem = await api.post<Item>('/items', item);

      // Replace optimistic item with server item
      items.set(
        items().map(i => i.id === tempId ? serverItem : i)
      );

      const updates = new Map(pendingUpdates());
      updates.delete(tempId);
      pendingUpdates.set(updates);
    } catch (err) {
      // Rollback on error
      items.set(items().filter(i => i.id !== tempId));

      const updates = new Map(pendingUpdates());
      updates.delete(tempId);
      pendingUpdates.set(updates);

      throw err;
    }
  };

  const updateItem = async (id: string, updates: Partial<Item>) => {
    const original = items().find(i => i.id === id);
    if (!original) return;

    // Optimistically update
    const optimisticItem = { ...original, ...updates };
    items.set(items().map(i => i.id === id ? optimisticItem : i));
    pendingUpdates.set(new Map(pendingUpdates()).set(id, original));

    try {
      const serverItem = await api.patch<Item>(`/items/${id}`, updates);
      items.set(items().map(i => i.id === id ? serverItem : i));

      const pending = new Map(pendingUpdates());
      pending.delete(id);
      pendingUpdates.set(pending);
    } catch (err) {
      // Rollback
      items.set(items().map(i => i.id === id ? original : i));

      const pending = new Map(pendingUpdates());
      pending.delete(id);
      pendingUpdates.set(pending);

      throw err;
    }
  };
}

return {
  items,
  pendingUpdates,
  addItem,
  updateItem
};
}

```

Undo/Redo

```
function createUndoableStore<T>(initialValue: T) {
  const present = signal(initialValue);
  const past = signal<T[]>([[]]);
  const future = signal<T[]>([[]]);

  const canUndo = memo(() => past().length > 0);
  const canRedo = memo(() => future().length > 0);

  const set = (value: T) => {
    past.set([...past(), present()]);
    present.set(value);
    future.set([]); // Clear redo history
  };

  const undo = () => {
    if (!canUndo()) return;

    const previous = past()[past().length - 1];
    future.set([present(), ...future()]);
    present.set(previous);
    past.set(past().slice(0, -1));
  };

  const redo = () => {
    if (!canRedo()) return;

    const next = future()[0];
    past.set([...past(), present()]);
    present.set(next);
    future.set(future().slice(1));
  };

  const reset = () => {
    present.set(initialValue);
    past.set([]);
    future.set([]);
  };

  return {
    value: present,
    canUndo,
    canRedo,
    set,
    undo,
    redo,
    reset
  };
}

// Usage
const editor = createUndoableStore('');

function TextEditor() {
  return (
    <div>
      <div className="toolbar">
        <button
          onClick={() => editor.undo()}
          disabled={!editor.canUndo()}
        >
          Undo
        </button>
        <button
          onClick={() => editor.redo()}
          disabled={!editor.canRedo()}
        >
          Redo
        </button>
      </div>
      <textarea
        value={editor.value()}
        onChange={(e) => editor.set(e.currentTarget.value)}
      />
    </div>
  );
}
```

Performance Optimization

Batching Updates

```

import { batch } from '@philjs/core';

function createFormStore() {
  const firstName = signal('');
  const lastName = signal('');
  const email = signal('');
  const age = signal(0);

  const loadUser = (user: User) => {
    // Batch multiple updates into single effect run
    batch(() => {
      firstName.set(user.firstName);
      lastName.set(user.lastName);
      email.set(user.email);
      age.set(user.age);
    });
  };

  return {
    firstName,
    lastName,
    email,
    age,
    loadUser
  };
}

```

Selective Updates

```

function createOptimizedStore() {
  const data = signal<LargeDataset>({ /* huge object */ });

  // ⚡ Update only what changed
  const updateField = (field: string, value: any) => {
    data.set({
      ...data(),
      [field]: value
    });
  };

  // ⚡ Avoid replacing entire object if only one field changed
  const badUpdate = (newData: LargeDataset) => {
    data.set(newData); // Triggers update even if values are same
  };

  return { data, updateField };
}

```

Advanced Patterns

Use advanced patterns when state spans multiple domains, transitions are complex, or you need stronger guarantees around consistency.

Layered Stores (Domain + UI)

```

function createProfileState() {
  const profile = signal<UserProfile> | null>(null);
  const isEditing = signal(false);
  const saveState = signal<'idle' | 'saving' | 'saved' | 'error'>('idle');

  const startEdit = () => isEditing.set(true);
  const cancelEdit = () => isEditing.set(false);

  return { profile, isEditing, saveState, startEdit, cancelEdit };
}

```

Event-Driven State

```

function createInboxState() {
  const messages = signal<Message[]>([]);
  const unreadCount = memo(() => messages().filter(m => !m.read).length);

  const onMessage = (message: Message) =>
    messages.set(list => [message, ...list]);

  return { messages, unreadCount, onMessage };
}

```

See also: [Advanced State Patterns](#), [State Middleware](#)

Summary

Decision Tree: Choosing the Right Pattern

```

Is the state only needed in one component?
└ Yes → Use local component state (signal)

Is the state shared between parent and children?
└ Yes → Use lifted state (props) or custom hooks

Is the state needed across unrelated components?
├ Few components → Use module-level shared state
└ Many components → Use Context API or global store

Does the state need to persist?
├ Between sessions → Use localStorage persistence
├ Within session → Use sessionStorage
└ Large data → Use IndexedDB

Does the state sync with external sources?
├ Real-time → Use WebSocket synchronization
├ Updates only → Use Server-Sent Events
├ Periodic → Use polling pattern
└ Cross-tab → Use BroadcastChannel or storage events

Is the state complex with many transitions?
└ Yes → Consider state machines

Does the state need history?
└ Yes → Implement undo/redo pattern

```

Best Practices Summary

State Organization: - Keep state as local as possible - Lift state only as high as necessary - Use stores for truly global state - Compose stores instead of creating monoliths - Group related state together

Performance: - Batch related updates with `batch()` - Use `memo()` for expensive computations - Avoid unnecessary object creation in updates - Use `untrack()` to break unwanted dependencies - Debounce persistence operations

Patterns: - Implement async patterns correctly with loading/error states - Use optimistic updates for better UX - Consider state machines for complex flows - Use context for dependency injection - Create custom hooks for reusable state logic

Persistence: - Persist state when needed (settings, preferences) - Handle errors gracefully in storage operations - Use appropriate storage mechanism (localStorage, sessionStorage, IndexedDB) - Implement cross-tab synchronization for shared state

Synchronization: - Use WebSockets for real-time bi-directional communication - Use SSE for server-push updates - Implement polling for periodic updates - Handle connection failures and reconnection - Clean up resources properly

Common Pitfalls to Avoid:

- Storing everything in global state
- Creating deeply nested state objects
- Forgetting to handle async errors
- Not batching related updates
- Mutating state directly instead of using `.set()`
- Overusing context (creates indirection)
- Not cleaning up effects and subscriptions
- Synchronous persistence operations without debouncing

Key Takeaways:

1. **Signals are your foundation** - Start with local signals and scale up as needed
2. **Memo for derived state** - Use `memo()` for computed values that depend on other signals
3. **Effects for side effects** - Use `effect()` for subscriptions, DOM manipulation, and persistence
4. **Context for dependency injection** - Use Context API when you need to avoid prop drilling
5. **Stores for application state** - Create stores for complex, shared application state
6. **Choose the right persistence** - Match persistence mechanism to your data size and requirements
7. **Handle async properly** - Always track loading and error states for async operations
8. **Clean up resources** - Return cleanup functions from effects

Next Steps:

- [Performance Optimization](#)
- [Testing State Management](#)
- [Async Patterns](#)

Related Documentation:

- [Signals](#)
- [Effects and Memos](#)
- [State Middleware](#)
- [Advanced State Patterns](#)

Performance Best Practices

Optimize your PhilJS applications for maximum performance using Node.js 24+, TypeScript 6, and ES2024 features.

Requirements

- **Node.js 24+** - Required for native ES2024 support
- **TypeScript 6+** - Required for isolated declarations and optimal build performance

Fine-Grained Reactivity Advantages

PhilJS uses fine-grained reactivity, which means:

- Only affected DOM nodes update
- No virtual DOM diffing overhead
- No full component re-renders
- Automatic dependency tracking
- Minimal re-computation

```
import { signal, memo } from '@philjs/core';

function Counter() {
  const count = signal(0);
  const doubled = memo(() => count() * 2);

  // When count changes:
  // 1. Only `count()` text node updates
  // 2. Only `doubled()` text node updates
  // 3. Button, div, and other elements untouched

  return (
    <div>
      <p>Count: {count()}</p>
      <p>Doubled: {doubled()}</p>
      <button onClick={() => count.set(count() + 1)}>Increment</button>
    </div>
  );
}
```

ES2024 Performance Patterns

Use Native Array Methods (ES2024)

ES2024 provides immutable array methods that are optimized by V8:

```
const items = signal<Item[]>([...largeArray]);

// Use toSorted() instead of sort() for immutability
const sorted = memo(() =>
  items().toSorted((a, b) => b.score - a.score)
);

// Use toSpliced() for immutable splice operations
const withoutFirst = memo(() =>
  items().toSpliced(0, 1)
);

// Use toReversed() for immutable reversal
const reversed = memo(() =>
  items().toReversed()
);

// Combine with at() for safer indexing
const lastItem = memo(() =>
  items().at(-1)
);
```

Use Object.groupBy() and Map.groupBy()

Native grouping is faster than manual reduce operations:

```
const users = signal<User[]>([]);

// Use Object.groupBy() for grouping by string keys
const usersByRole = memo(() =>
  Object.groupBy(users(), user => user.role)
);

// Use Map.groupBy() for complex keys
const usersByDepartment = memo(() =>
  Map.groupBy(users(), user => user.department)
);
```

Use Promise.withResolvers() for Async Patterns

```

function createDeferredSignal<T>() {
  const { promise, resolve, reject } = Promise.withResolvers<T>();
  const data = signal<T | null>(null);
  const error = signal<Error | null>(null);

  promise.then(
    value => data.set(value),
    err => error.set(err)
  );

  return { data, error, resolve, reject };
}

```

Memoization

Use memo() for Expensive Computations

```

const items = signal<Item[]>([...largeArray]);

// Memoize expensive operations using ES2024 immutable methods
const filtered = memo(() =>
  items().filter(item => item.active && item.score > 50)
);

const sorted = memo(() =>
  filtered().toSorted((a, b) => b.score - a.score)
);

const top10 = memo(() =>
  sorted().slice(0, 10)
);

// Avoid: computing inline without memoization
function BadComponent() {
  return (
    <div>
      {items()
        .filter(item => item.active && item.score > 50)
        .toSorted((a, b) => b.score - a.score)
        .slice(0, 10)
        .map(item => <ItemCard key={item.id} item={item} />)
      }
    </div>
  );
}

```

When NOT to Use memo()

```

// ⚡ Overkill for simple operations
const doubled = memo(() => count() * 2);

// ⚡ Just compute it
const doubled = () => count() * 2;

// ⚡ Memoizing simple references
const userName = memo(() => user().name);

// ⚡ Access directly
const userName = () => user().name;

```

Batching Updates

Batch Related Changes

```

import { batch } from '@philjs/core';

// ⚡ Batch multiple signal updates
function updateUser(updates: Partial<User>) {
  batch(() => {
    firstName.set(updates.firstName || firstName());
    lastName.set(updates.lastName || lastName());
    email.set(updates.email || email());
    age.set(updates.age || age());
  });
}

// ⚡ Sequential updates trigger effects multiple times
function badUpdate(updates: Partial<User>) {
  firstName.set(updates.firstName || firstName());
  lastName.set(updates.lastName || lastName());
  email.set(updates.email || email());
  age.set(updates.age || age());
}

```

Batch in Loops

```
// 🚨 Batch updates when processing arrays
function processItems(newItems: Item[]) {
  batch(() => {
    newItems.forEach(item => {
      items.set([...items(), item]);
    });
  });
}

// Even better: single update
function processItemsBetter(newItems: Item[]) {
  items.set([...items(), ...newItems]);
}
```

Lazy Loading

Code Splitting with lazy()

```
import { lazy, Suspense } from '@philjs/core';

// 🚨 Split large components
const Dashboard = lazy(() => import('./Dashboard'));
const Settings = lazy(() => import('./Settings'));
const Analytics = lazy(() => import('./Analytics'));

function App() {
  return (
    <Router>
      <Suspense fallback={<PageLoader />}>
        <Route path="/dashboard" component={Dashboard} />
        <Route path="/settings" component={Settings} />
        <Route path="/analytics" component={Analytics} />
      </Suspense>
    </Router>
  );
}
```

Route-Based Splitting

```
const routes = [
  {
    path: '/',
    component: lazy(() => import('../pages/Home'))
  },
  {
    path: '/products',
    component: lazy(() => import('../pages/Products'))
  },
  {
    path: '/checkout',
    component: lazy(() => import('../pages/Checkout'))
  }
];
```

Component-Based Splitting

```
// Heavy editor only loaded when needed
const CodeEditor = lazy(() => import('../CodeEditor'));

function DocumentEditor() {
  const showCodeEditor = signal(false);

  return (
    <div>
      <button onClick={() => showCodeEditor.set(true)}>
        Show Code Editor
      </button>

      {showCodeEditor() && (
        <Suspense fallback={<EditorLoader />}>
          <CodeEditor />
        </Suspense>
      )}
    </div>
  );
}
```

List Virtualization

Virtual Scrolling

```

import { signal, memo } from '@philjs/core';

interface VirtualListProps<T> {
    items: T[];
    itemHeight: number;
    containerHeight: number;
    renderItem: (item: T, index: number) => JSX.Element;
}

function VirtualList<T>({
    items,
    itemHeight,
    containerHeight,
    renderItem
}: VirtualListProps<T>) {
    const scrollTop = signal(0);

    const visibleRange = memo(() => {
        const start = Math.floor(scrollTop() / itemHeight);
        const count = Math.ceil(containerHeight / itemHeight);
        const end = Math.min(start + count + 1, items.length);

        return { start, end };
    });

    const visibleItems = memo(() => {
        const { start, end } = visibleRange();
        return items.slice(start, end);
    });

    const totalHeight = items.length * itemHeight;
    const offsetY = visibleRange().start * itemHeight;

    return (
        <div
            style={{ height: `${containerHeight}px`, overflow: 'auto' }}
            onScroll={(e) => scrollTop.set(e.currentTarget.scrollTop)}
        >
            <div style={{ height: `${totalHeight}px`, position: 'relative' }}>
                <div style={{ transform: `translateY(${offsetY}px)` }}>
                    {visibleItems().map((item, index) =>
                        renderItem(item, visibleRange().start + index)
                    )}
                </div>
            </div>
        </div>
    );
}

// Usage
<VirtualList
    items={largeArray}
    itemHeight={50}
    containerHeight={400}
    renderItem={({item, index}) => (
        <div key={index} style={{ height: '50px' }}>
            {item.name}
        </div>
    )}
/>

```

Debouncing and Throttling

Debounce Search

```

function SearchBar() {
  const searchTerm = signal('');
  const debouncedTerm = signal('');

  let timeoutId: number;

  effect(() => {
    const term = searchTerm();

    clearTimeout(timeoutId);
    timeoutId = setTimeout(() => {
      debouncedTerm.set(term);
    }, 300);
  });

  return () => clearTimeout(timeoutId);
);

effect(async () => {
  const term = debouncedTerm();
  if (term) {
    const results = await searchAPI(term);
    // Update results
  }
});

return (
  <input
    value={searchTerm()}
    onInput={(e) => searchTerm.set(e.currentTarget.value)}
    placeholder="Search..." />
);
}

```

Throttle Scroll

```

function ScrollTracker() {
  const scrollY = signal(0);

  let throttleTimeout: number | null = null;

  const handleScroll = () => {
    if (throttleTimeout) return;

    throttleTimeout = setTimeout(() => {
      scrollY.set(window.scrollY);
      throttleTimeout = null;
    }, 100);
  };

  effect(() => {
    window.addEventListener('scroll', handleScroll);
    return () => window.removeEventListener('scroll', handleScroll);
  });
}

return <div>Scroll Position: {scrollY()}</div>;
}

```

Image Optimization

Lazy Load Images

```

function LazyImage({ src, alt }: { src: string; alt: string }) {
  const imgRef = signal<HTMLImageElement | null>(null);
  const isVisible = signal(false);

  effect(() => {
    const img = imgRef();
    if (!img) return;

    const observer = new IntersectionObserver(([entry]) => {
      if (entry.isIntersecting) {
        isVisible.set(true);
        observer.disconnect();
      }
    });
    observer.observe(img);

    return () => observer.disconnect();
  });

  return (
    <img
      ref={imgRef}
      src={isVisible() ? src : ''}
      alt={alt}
      loading="lazy"
    />
  );
}

```

Progressive Images

```

function ProgressiveImage({
  lowResSrc,
  highResSrc,
  alt
}: {
  lowResSrc: string;
  highResSrc: string;
  alt: string;
}) {
  const loaded = signal(false);

  return (
    <div className="progressive-image">
      <img
        src={lowResSrc}
        alt={alt}
        className={loaded() ? 'hidden' : 'blur'}
      />
      <img
        src={highResSrc}
        alt={alt}
        className={loaded() ? 'visible' : 'hidden'}
        onLoad={() => loaded.set(true)}
      />
    </div>
  );
}

```

Bundle Optimization

Import Only What You Need

```

// ⚡ Importing entire library
import _ from 'lodash';
const result = _.debounce(fn, 300);

// ⚡ Import specific function
import debounce from 'lodash/debounce';
const result = debounce(fn, 300);

// Even better: use native or PhilJS utilities

```

Dynamic Imports

```
// ⚡ Load heavy libraries only when needed
async function processImage(image: File) {
  const { default: imageProcessingLib } = await import('heavy-image-lib');
  return imageProcessingLib.process(image);
}

// ⚡ Load data only when needed
async function loadChartData() {
  const data = await import('./chartData.json');
  return data.default;
}
```

Avoid Unnecessary Work

Skip Effects with untrack()

```
import { untrack } from '@philjs/core';

const userId = signal('123');
const userName = signal('Alice');

effect(() => {
  // Only track userId, not userName
  console.log('User ID changed:', userId());

  // Read userName without tracking
  const name = untrack(() => userName());
  console.log('Current name:', name);
});
```

Conditional Tracking

```
const enabled = signal(true);
const value = signal(0);

effect(() => {
  if (!enabled()) return;

  // Only track value when enabled
  console.log('Value:', value());
});
```

Web Workers

Offload Heavy Computation

```

// worker.ts
self.onmessage = (e) => {
  const { data } = e;

  // Heavy computation
  const result = expensiveOperation(data);

  self.postMessage(result);
};

// Component
function HeavyComputation() {
  const result = signal<any>(null);
  const loading = signal(false);

  const compute = async (data: any) => {
    loading.set(true);

    const worker = new Worker(new URL('./worker.ts', import.meta.url));

    worker.postMessage(data);

    worker.onmessage = (e) => {
      result.set(e.data);
      loading.set(false);
      worker.terminate();
    };
  };

  return (
    <div>
      <button onClick={() => compute(inputData)}>
        Compute
      </button>

      {loading() && <Spinner />}
      {result() && <ResultView data={result()} />}
    </div>
  );
}

```

Measuring Performance

Performance Profiling

```

function ProfiledComponent() {
  effect(() => {
    performance.mark('component-start');

    // Component logic

    performance.mark('component-end');
    performance.measure(
      'component-render',
      'component-start',
      'component-end'
    );
  });

  const measure = performance.getEntriesByName('component-render')[0];
  console.log('Render time:', measure.duration);
});

return <div>Content</div>;
}

```

Track Effect Performance

```

function trackEffect(name: string, fn: () => void | ((() => void)) {
  return effect(() => {
    const start = performance.now();

    const cleanup = fn();

    const duration = performance.now() - start;
    if (duration > 16) { // Longer than one frame
      console.warn(`Slow effect "${name}": ${duration}ms`);
    }

    return cleanup;
  });
}

// Usage
trackEffect('fetchUserData', () => {
  const data = fetchUser(userId());
  user.set(data);
});

```

Memory Management

Clean Up Effects

```

// 🚫 Always clean up subscriptions
function Component() {
  effect(() => {
    const subscription = dataStream.subscribe(data => {
      processData(data);
    });

    return () => subscription.unsubscribe();
  });

  return <div>Content</div>;
}

// 🚫 Clean up timers
function Timer() {
  const count = signal(0);

  effect(() => {
    const id = setInterval(() => {
      count.set(count() + 1);
    }, 1000);

    return () => clearInterval(id);
  });

  return <div>{count()}</div>;
}

// 🚫 Clean up event listeners
function ScrollListener() {
  effect(() => {
    const handler = () => console.log('Scrolled');

    window.addEventListener('scroll', handler);
    return () => window.removeEventListener('scroll', handler);
  });

  return <div>Scroll tracker</div>;
}

```

Avoid Memory Leaks

```
// 🚧 Creating new objects in signals
const config = signal({
  theme: 'dark',
  language: 'en'
});

setInterval(() => {
  config.set({
    ...config(),
    timestamp: Date.now() // Creates new object every second!
  });
}, 1000);

// 🚧 Only update when necessary
const lastUpdate = signal(Date.now());

setInterval(() => {
  if (shouldUpdate()) {
    lastUpdate.set(Date.now());
  }
}, 1000);
```

Caching

Memoized API Calls

```
function createCachedFetch<T>(fetcher: (key: string) => Promise<T>) {
  const cache = new Map<string, T>();

  return async (key: string): Promise<T> => {
    if (cache.has(key)) {
      return cache.get(key)!;
    }

    const data = await fetcher(key);
    cache.set(key, data);
    return data;
  };
}

const fetchUser = createCachedFetch((userId: string) =>
  api.get(`/users/${userId}`)
);

// Usage
const user = signal<User | null>(null);

effect(async () => {
  const data = await fetchUser(userId()); // Cached on subsequent calls
  user.set(data);
});
```

TypeScript 6 Build Performance

Enable Isolated Declarations

TypeScript 6's isolated declarations provide faster builds:

```
{
  "compilerOptions": {
    "isolatedDeclarations": true,
    "declaration": true,
    "declarationMap": true
  }
}
```

Use Strict Type Checking

Strict checking catches performance issues at compile time:

```
{
  "compilerOptions": {
    "strict": true,
    "exactOptionalPropertyTypes": true,
    "noUncheckedIndexedAccess": true
  }
}
```

Summary

Performance Best Practices:

ES2024 Features: - Use `toSorted()`, `toReversed()`, `toSpliced()` for immutable array operations - Use `Object.groupBy()` and `Map.groupBy()` for efficient grouping - Use `Promise.withResolvers()` for cleaner async patterns - Use Set methods (`union`, `intersection`, `difference`) for set operations

Reactivity: - Leverage fine-grained reactivity - Use `memo()` for expensive computations - Batch related signal updates

Code Organization: - Implement code splitting with lazy() - Virtualize long lists - Debounce/throttle frequent operations - Lazy load images - Optimize bundle size
Runtime: - Use Web Workers for heavy computation - Profile and measure performance - Clean up effects properly - Cache API responses - Avoid unnecessary re-computation
TypeScript 6: - Enable isolated declarations for faster builds - Use strict type checking

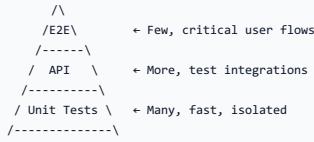
Next: [Testing](#)

Testing Best Practices

Comprehensive testing strategies for PhilJS applications.

Testing Philosophy

Test Pyramid



Balance: - 70% Unit - Signals, memos, effects, utilities - 20% Integration - Component integration, API calls - 10% E2E - Critical user journeys

Unit Testing

Testing Signals

```
import { describe, it, expect } from 'vitest';
import { signal } from '@philjs/core';

describe('signal()', () => {
  it('creates reactive state', () => {
    const count = signal(0);
    expect(count()).toBe(0);
  });

  it('updates value', () => {
    const count = signal(0);
    count.set(5);
    expect(count()).toBe(5);
  });

  it('supports functional updates', () => {
    const count = signal(10);
    count.set(prev => prev + 1);
    expect(count()).toBe(11);
  });

  it('handles objects', () => {
    const user = signal({ name: 'Alice', age: 30 });

    user.set({ ...user(), age: 31 });

    expect(user()).toEqual({ name: 'Alice', age: 31 });
  });
});
```

Testing Memos

```

import { describe, it, expect, vi } from 'vitest';
import { signal, memo } from '@philjs/core';

describe('memo()', () => {
  it('computes derived value', () => {
    const count = signal(2);
    const doubled = memo(() => count() * 2);

    expect(doubled()).toBe(4);
  });
});

it('updates when dependency changes', () => {
  const count = signal(2);
  const doubled = memo(() => count() * 2);

  count.set(5);

  expect(doubled()).toBe(10);
});

it('memoizes computation', () => {
  const count = signal(2);
  const computation = vi.fn(() => count() * 2);
  const doubled = memo(computation);

  // First read
  doubled();
  expect(computation).toHaveBeenCalledTimes(1);

  // Second read (cached)
  doubled();
  expect(computation).toHaveBeenCalledTimes(1);

  // Change dependency
  count.set(3);

  // Read again (recomputes)
  doubled();
  expect(computation).toHaveBeenCalledTimes(2);
});

it('chains memos', () => {
  const count = signal(2);
  const doubled = memo(() => count() * 2);
  const quadrupled = memo(() => doubled() * 2);

  expect(quadrupled()).toBe(8);

  count.set(3);
  expect(quadrupled()).toBe(12);
});
});

```

Testing Effects

```

import { describe, it, expect, vi } from 'vitest';
import { signal, effect } from '@philjs/core';

describe('effect()', () => {
  it('runs immediately', () => {
    const spy = vi.fn();
    effect(spy);
    expect(spy).toHaveBeenCalledTimes(1);
  });

  it('tracks dependencies', () => {
    const count = signal(0);
    const spy = vi.fn(() => {
      count(); // Track count
    });
    effect(spy);
    count.set(1);
    count.set(2);
    expect(spy).toHaveBeenCalledTimes(3); // Initial + 2 updates
  });

  it('calls cleanup on re-run', () => {
    const cleanup = vi.fn();
    const count = signal(0);

    effect(() => {
      count();
      return cleanup;
    });

    expect(cleanup).not.toHaveBeenCalled();
    count.set(1);
    expect(cleanup).toHaveBeenCalledTimes(1);

    count.set(2);
    expect(cleanup).toHaveBeenCalledTimes(2);
  });

  it('handles async effects', async () => {
    const data = signal<string | null>(null);

    effect(async () => {
      const result = await Promise.resolve('loaded');
      data.set(result);
    });

    // Wait for async operation
    await new Promise(resolve => setTimeout(resolve, 0));

    expect(data()).toBe('loaded');
  });
});

```

Testing Custom Hooks

```

import { describe, it, expect } from 'vitest';
import { signal, memo } from '@philjs/core';

function useCounter(initialValue = 0) {
  const count = signal(initialValue);
  const doubled = memo(() => count() * 2);

  const increment = () => count.set(count() + 1);
  const decrement = () => count.set(count() - 1);
  const reset = () => count.set(initialValue);

  return {
    count,
    doubled,
    increment,
    decrement,
    reset
  };
}

describe('useCounter()', () => {
  it('initializes with value', () => {
    const counter = useCounter(5);
    expect(counter.count()).toBe(5);
  });

  it('increments count', () => {
    const counter = useCounter(0);
    counter.increment();
    expect(counter.count()).toBe(1);
  });

  it('decrements count', () => {
    const counter = useCounter(5);
    counter.decrement();
    expect(counter.count()).toBe(4);
  });

  it('doubles count', () => {
    const counter = useCounter(3);
    expect(counter.doubled()).toBe(6);

    counter.increment();
    expect(counter.doubled()).toBe(8);
  });

  it('resets to initial value', () => {
    const counter = useCounter(10);

    counter.increment();
    counter.increment();
    expect(counter.count()).toBe(12);

    counter.reset();
    expect(counter.count()).toBe(10);
  });
});

```

Component Testing

Testing Presentational Components

```
import { describe, it, expect, vi } from 'vitest';
import { render, screen } from '@testing-library/react';
import { Button } from './Button';

describe('Button', () => {
  it('renders with label', () => {
    render(<Button label="Click me" onClick={() => {}} />);

    expect(screen.getByText('Click me')).toBeInTheDocument();
  });

  it('calls onClick when clicked', async () => {
    const handleClick = vi.fn();

    render(<Button label="Click me" onClick={handleClick} />);

    await screen.getByText('Click me').click();

    expect(handleClick).toHaveBeenCalledTimes(1);
  });

  it('disables when disabled prop is true', () => {
    render(<Button label="Click me" onClick={() => {}} disabled />);

    expect(screen.getByText('Click me')).toBeDisabled();
  });

  it('applies variant class', () => {
    render(<Button label="Click me" onClick={() => {}} variant="primary" />);

    expect(screen.getByText('Click me')).toHaveClass('btn-primary');
  });
});
```

Testing Container Components

```

import { describe, it, expect, vi, beforeEach } from 'vitest';
import { render, screen, waitFor } from '@testing-library/react';
import { UserProfile } from './UserProfile';

// Mock API
vi.mock('../api', () => ({
  fetchUser: vi.fn()
}));

import { fetchUser } from '../api';

describe('UserProfile', () => {
  beforeEach(() => {
    vi.clearAllMocks();
  });

  it('shows loading state', () => {
    fetchUser.mockImplementation(() => new Promise(() => {}));
    // Never resolves

    render(<UserProfile userId="123" />);

    expect(screen.getByText('Loading...')).toBeInTheDocument();
  });

  it('shows error state', async () => {
    fetchUser.mockRejectedValue(new Error('Failed to fetch'));

    render(<UserProfile userId="123" />);

    await waitFor(() => {
      expect(screen.getByText(/error/i)).toBeInTheDocument();
    });
  });

  it('shows user data', async () => {
    const mockUser = {
      id: '123',
      name: 'Alice Smith',
      email: 'alice@example.com'
    };

    fetchUser.mockResolvedValue(mockUser);

    render(<UserProfile userId="123" />);

    await waitFor(() => {
      expect(screen.getByText('Alice Smith')).toBeInTheDocument();
      expect(screen.getByText('alice@example.com')).toBeInTheDocument();
    });
  });

  it('refetches when userId changes', async () => {
    fetchUser.mockResolvedValue({ id: '123', name: 'Alice' });

    const { rerender } = render(<UserProfile userId="123" />);

    await waitFor(() => {
      expect(fetchUser).toHaveBeenCalledWith('123');
    });

    fetchUser.mockResolvedValue({ id: '456', name: 'Bob' });

    rerender(<UserProfile userId="456" />);

    await waitFor(() => {
      expect(fetchUser).toHaveBeenCalledWith('456');
      expect(screen.getByText('Bob')).toBeInTheDocument();
    });
  });
});

```

Testing Forms

```

import { describe, it, expect, vi } from 'vitest';
import { render, screen } from '@testing-library/react';
import { userEvent } from '@testing-library/user-event';
import { LoginForm } from './LoginForm';

describe('LoginForm', () => {
  it('validates required fields', async () => {
    const handleSubmit = vi.fn();

    render(<LoginForm onSubmit={handleSubmit} />);

    const submitButton = screen.getByText('Log In');
    await submitButton.click();

    expect(screen.getByText('Email is required')).toBeInTheDocument();
    expect(screen.getByText('Password is required')).toBeInTheDocument();
    expect(handleSubmit).not.toHaveBeenCalled();
  });

  it('validates email format', async () => {
    render(<LoginForm onSubmit={() => {}} />);

    const emailInput = screen.getByLabelText('Email');
    await userEvent.type(emailInput, 'invalid-email');

    const submitButton = screen.getByText('Log In');
    await submitButton.click();

    expect(screen.getByText('Invalid email format')).toBeInTheDocument();
  });

  it('submits valid form', async () => {
    const handleSubmit = vi.fn();

    render(<LoginForm onSubmit={handleSubmit} />);

    await userEvent.type(screen.getByLabelText('Email'), 'alice@example.com');
    await userEvent.type(screen.getByLabelText('Password'), 'password123');

    await screenByText('Log In').click();

    expect(handleSubmit).toHaveBeenCalledWith({
      email: 'alice@example.com',
      password: 'password123'
    });
  });

  it('shows loading state during submission', async () => {
    const handleSubmit = vi.fn(() => new Promise(() => {})); // Never resolves

    render(<LoginForm onSubmit={handleSubmit} />);

    await userEvent.type(screen.getByLabelText('Email'), 'alice@example.com');
    await userEvent.type(screen.getByLabelText('Password'), 'password123');

    await screenByText('Log In').click();

    expect(screen.getByText('Logging in...')).toBeInTheDocument();
    expect(screen.getByText('Log In')).toBeDisabled();
  });
});

```

Integration Testing

Testing with Context

```

import { describe, it, expect } from 'vitest';
import { render, screen } from '@testing-library/philst';
import { createContext } from '@philst/core';
import { ThemedButton } from './ThemedButton';

const ThemeContext = createContext('light');

describe('ThemedButton with Context', () => {
  it('uses theme from context', () => {
    render(
      <ThemeContext.Provider value="dark">
        <ThemedButton>Click me</ThemedButton>
      </ThemeContext.Provider>
    );
    expect(screen.getByText('Click me')).toHaveClass('btn-dark');
  });

  it('falls back to default theme', () => {
    render(<ThemedButton>Click me</ThemedButton>);

    expect(screen.getByText('Click me')).toHaveClass('btn-light');
  });
});

```

Testing with Router

```

import { describe, it, expect } from 'vitest';
import { render, screen } from '@testing-library/philst';
import { Router, Route } from '@philst/router';
import { App } from './App';

describe('App Routing', () => {
  it('renders home page', () => {
    render(
      <Router initialPath="/">
        <App />
      </Router>
    );
    expect(screen.getByText('Welcome Home')).toBeInTheDocument();
  });

  it('renders about page', () => {
    render(
      <Router initialPath="/about">
        <App />
      </Router>
    );
    expect(screen.getByText('About Us')).toBeInTheDocument();
  });

  it('handles 404', () => {
    render(
      <Router initialPath="/nonexistent">
        <App />
      </Router>
    );
    expect(screen.getByText('Page Not Found')).toBeInTheDocument();
  });
});

```

Testing API Integration

```

import { describe, it, expect, beforeEach, afterEach } from 'vitest';
import { setupServer } from 'msw/node';
import { http, HttpResponseMessage } from 'msw';
import { render, screen, waitFor } from '@testing-library/react';
import { ProductList } from './ProductList';

const server = setupServer(
  http.get('/api/products', () => {
    return HttpResponseMessage.json([
      { id: '1', name: 'Product 1', price: 10 },
      { id: '2', name: 'Product 2', price: 20 }
    ]);
  })
);

beforeEach(() => server.listen());
afterEach(() => server.close());

describe('ProductList API Integration', () => {
  it('fetches and displays products', async () => {
    render(<ProductList />);

    expect(screen.getByText('Loading...')).toBeInTheDocument();

    await waitFor(() => {
      expect(screen.getByText('Product 1')).toBeInTheDocument();
      expect(screen.getByText('Product 2')).toBeInTheDocument();
    });
  });

  it('handles API errors', async () => {
    server.use(
      http.get('/api/products', () => {
        return HttpResponseMessage.error();
      })
    );

    render(<ProductList />);

    await waitFor(() => {
      expect(screen.getByText(/error/i)).toBeInTheDocument();
    });
  });

  it('handles empty results', async () => {
    server.use(
      http.get('/api/products', () => {
        return HttpResponseMessage.json([]);
      })
    );

    render(<ProductList />);

    await waitFor(() => {
      expect(screen.getByText('No products found')).toBeInTheDocument();
    });
  });
});

```

E2E Testing

Playwright Tests

```

import { test, expect } from '@playwright/test';

test.describe('User Authentication', () => {
  test('user can log in', async ({ page }) => {
    await page.goto('/login');

    await page.fill('input[name="email"]', 'alice@example.com');
    await page.fill('input[name="password"]', 'password123');

    await page.click('button[type="submit"]');

    await expect(page).toHaveURL('/dashboard');
    await expect(page.locator('text=Welcome, Alice')).toBeVisible();
  });

  test('shows error for invalid credentials', async ({ page }) => {
    await page.goto('/login');

    await page.fill('input[name="email"]', 'alice@example.com');
    await page.fill('input[name="password"]', 'wrongpassword');

    await page.click('button[type="submit"]');

    await expect(page.locator('text=Invalid credentials')).toBeVisible();
  });

  test('user can log out', async ({ page }) => {
    // Assume user is logged in
    await page.goto('/dashboard');

    await page.click('button:has-text("Logout")');

    await expect(page).toHaveURL('/login');
  });
});

test.describe('Shopping Cart', () => {
  test('user can add items to cart', async ({ page }) => {
    await page.goto('/products');

    await page.click('button:has-text("Add to Cart").first');

    const cartBadge = page.locator('.cart-badge');
    await expect(cartBadge).toHaveText('1');
  });

  test('user can complete checkout', async ({ page }) => {
    await page.goto('/cart');

    await page.click('button:has-text("Checkout")');

    await page.fill('input[name="cardNumber"]', '4242424242424242');
    await page.fill('input[name="expiry"]', '12/25');
    await page.fill('input[name="cvc"]', '123');

    await page.click('button:has-text("Pay")');

    await expect(page.locator('text=Order Confirmed')).toBeVisible();
  });
});

```

Testing Best Practices

Arrange-Act-Assert Pattern

```

it('increments counter', () => {
  // Arrange
  const counter = useCounter(0);

  // Act
  counter.increment();

  // Assert
  expect(counter.count()).toBe(1);
});

```

Test One Thing

```
// 🚫 Testing too much
it('manages user profile', async () => {
  const profile = createUserProfile();

  profile.setName('Alice');
  expect(profile.name()).toBe('Alice');

  profile.setEmail('alice@example.com');
  expect(profile.email()).toBe('alice@example.com');

  await profile.save();
  expect(profile.saved()).toBe(true);
});

// 🚫 Separate tests
it('sets user name', () => {
  const profile = createUserProfile();
  profile.setName('Alice');
  expect(profile.name()).toBe('Alice');
});

it('sets user email', () => {
  const profile = createUserProfile();
  profile.setEmail('alice@example.com');
  expect(profile.email()).toBe('alice@example.com');
});

it('saves profile', async () => {
  const profile = createUserProfile();
  await profile.save();
  expect(profile.saved()).toBe(true);
});
```

Use Descriptive Names

```
// 🚫 Unclear
it('works', () => {
  // ...
});

// 🚫 Clear
it('increments count when button is clicked', () => {
  // ...
});
```

Avoid Implementation Details

```
// 🚫 Testing implementation
it('calls setState with incremented value', () => {
  const counter = useCounter();
  const spy = vi.spyOn(counter.count, 'set');

  counter.increment();

  expect(spy).toHaveBeenCalledWith(1);
});

// 🚫 Testing behavior
it('increments count', () => {
  const counter = useCounter();

  counter.increment();

  expect(counter.count()).toBe(1);
});
```

Coverage Goals

Target Coverage: - **Statements:** >80% - **Branches:** >75% - **Functions:** >80% - **Lines:** >80%

Priority: - Critical paths: 100% - Business logic: >90% - UI components: >70% - Utils: >90%

Summary

Testing Best Practices:

- Write tests that verify behavior, not implementation □ Use the test pyramid: many unit, some integration, few E2E □ Test signals, memos, and effects in isolation □ Mock external dependencies □ Use descriptive test names □ Follow Arrange-Act-Assert pattern □ Test error states and edge cases □ Maintain good coverage □ Run tests in CI/CD □ Keep tests fast and reliable

Next: [Code Organization →](#)

Error Handling Best Practices

Robust error handling patterns for production applications.

Error Boundaries

```
import { ErrorBoundary } from '@philjs/core';

export default function App() {
  return (
    <ErrorBoundary fallback={({error, reset}) => (
      <div>
        <h1>Something went wrong</h1>
        <pre>{error.message}</pre>
        <button onClick={reset}>Try again</button>
      </div>
    )}>
    <MainApp />
  </ErrorBoundary>
);
}
```

Async Error Handling

```
const handleSubmit = async () => {
  try {
    await submitForm();
  } catch (error) {
    if (error instanceof ValidationError) {
      setErrors(error.errors);
    } else {
      showErrorToast(error.message);
    }
  }
};
```

Best Practices

Do: Use Error Boundaries

```
// ✅ Good - catches errors
<ErrorBoundary fallback={(<Error />) =>}>
  <Component />
</ErrorBoundary>
```

Do: Log Errors

```
// ✅ Good - log to monitoring service
catch (error) {
  logError(error);
  showUserFriendlyMessage();
}
```

Next Steps

- Testing - Test error handling
- Deployment - Production errors

Tip: Always provide user-friendly error messages.

Warning: Never expose stack traces to end users in production.

Note: Use error boundaries for component errors, try/catch for async.

Deployment Best Practices

Deploy PhilJS applications to production.

Build for Production

```
# Build optimized bundle
npm run build

# Preview build
npm run preview
```

Environment Variables

```
# .env.production
PUBLIC_API_URL=https://api.example.com
SECRET_KEY=xxx # Never expose to client
```

Deployment Platforms

Vercel

```
npm install -g vercel  
vercel deploy
```

Netlify

```
npm install -g netlify-cli  
netlify deploy --prod
```

Docker

```
FROM node:24-alpine  
WORKDIR /app  
COPY package*.json ./  
RUN npm ci  
COPY . .  
RUN npm run build  
EXPOSE 3000  
CMD [ "npm", "start" ]
```

Best Practices

Do: Use Environment Variables

```
// Good - use env vars  
const apiUrl = import.meta.env.PUBLIC_API_URL;
```

Do: Enable Compression

```
// vite.config.ts  
export default {  
  build: {  
    minify: 'terser',  
    terserOptions: {  
      compress: {  
        drop_console: true  
      }  
    }  
  }  
};
```

Next Steps

- [Security](#) - Security practices
- [Performance](#) - Performance

Tip: Always test your production build locally before deploying.

Warning: Never commit .env files with secrets to version control.

Note: PhilJS works on all major platforms: Vercel, Netlify, Cloudflare.

Security Best Practices

Protect your PhilJS application and users from security vulnerabilities.

XSS Prevention

Cross-Site Scripting (XSS)

PhilJS automatically escapes content in JSX, but be aware of edge cases.

```
// ✅ Safe - automatically escaped
function UserProfile({ user }: { user: User }) {
  return (
    <div>
      <h1>{user.name}</h1> /* Escaped */
      <p>{user.bio}</p> /* Escaped */
    </div>
  );
}

// ❌ DANGEROUS - using dangerouslySetInnerHTML
function DangerousComponent({ html }: { html: string }) {
  return <div dangerouslySetInnerHTML={{ __html: html }} />;
}

// ✅ Safe - sanitize HTML first
import DOMPurify from 'dompurify';

function SafeHTML({ html }: { html: string }) {
  const sanitized = DOMPurify.sanitize(html);
  return <div dangerouslySetInnerHTML={{ __html: sanitized }} />;
}
```

Sanitizing User Input

```
function CommentForm() {
  const comment = signal('');

  const handleSubmit = async () => {
    // ✅ Validate and sanitize on server
    await api.post('/comments', {
      content: comment() // Server validates this
    });
  };

  return (
    <form onSubmit={handleSubmit}>
      <textarea
        value={comment()}
        onInput={(e) => comment.set(e.currentTarget.value)}
      />
      <button type="submit">Post Comment</button>
    </form>
  );
}
```

Avoid eval() and Function()

```
// ❌ DANGEROUS - arbitrary code execution
const userCode = getUserInput();
eval(userCode);

// ❌ DANGEROUS - similar to eval
new Function(userCode)();

// ✅ Safe - use JSON.parse for data
const userData = JSON.parse(userInput);
```

CSRF Protection

PhilJS includes built-in CSRF (Cross-Site Request Forgery) protection for server-side actions and API routes.

Built-in CSRF Protection

```

import { csrfProtection, generateCSRFToken, csrfField } from '@philjs/ssr';

// Set up CSRF middleware
const csrf = csrfProtection({
  getSessionId: (request) => {
    // Get session ID from cookie or header
    return request.headers.get('x-session-id') || 'default';
  },
  skip: (request) => {
    // Skip CSRF for certain routes (e.g., webhooks)
    return request.url.includes('/webhooks/');
  }
});

// Generate token for a request
const token = csrf.generateToken(request);

// Verify request
const isValid = csrf.verifyRequest(request);
if (!isValid) {
  return new Response('CSRF verification failed', { status: 403 });
}

```

Form Protection

```

import { csrfField } from '@philjs/ssr';

// Server-side: Generate token and include in form
export async function GET({ request }: { request: Request }) {
  const csrf = csrfProtection({ getSessionId: getSession });
  const csrfToken = csrf.generateToken(request);

  return new Response(`

<!DOCTYPE html>
<html>
  <body>
    <form method="POST" action="/submit">
      ${csrfField(csrfToken)}
      <input type="text" name="data" />
      <button type="submit">Submit</button>
    </form>
  </body>
</html>
`);
}

// Server-side: Verify on POST
export async function POST({ request }: { request: Request }) {
  const csrf = csrfProtection({ getSessionId: getSession });

  if (!csrf.verifyRequest(request)) {
    return new Response('CSRF verification failed', { status: 403 });
  }

  // Process form submission
  const formData = await request.formData();
  // ...
}

```

AJAX Request Protection

```

// Client-side: Fetch CSRF token
async function submitForm(data: any) {
  // Get CSRF token from meta tag or API
  const csrfToken = document.querySelector('meta[name="csrf-token"]')
    ?.getAttribute('content');

  const response = await fetch('/api/submit', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'X-CSRF-Token': csrfToken || ''
    },
    body: JSON.stringify(data)
  });

  return response.json();
}

```

PhilJS Component Integration

```

import { signal } from '@philjs/core';
import { csrfProtection } from '@philjs/ssr';

function SecureForm() {
  const formData = signal({ name: '', email: '' });
  const csrfToken = signal('');

  // Load CSRF token on mount
  effect(async () => {
    const response = await fetch('/api/csrf-token');
    const data = await response.json();
    csrfToken.set(data.token);
  });

  const handleSubmit = async (e: Event) => {
    e.preventDefault();

    await fetch('/api/submit', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'X-CSRF-Token': csrfToken()
      },
      body: JSON.stringify(formData())
    });
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="hidden" name="_csrf" value={csrfToken()} />
      <input
        type="text"
        value={formData().name}
        onChange={(e) => formData.set({ ...formData(), name: e.target.value })}
      />
      <button type="submit">Submit</button>
    </form>
  );
}

```

Custom Token Storage

PhilJS uses an in-memory token store by default. For production, use Redis:

```

import { csrfProtection, csrfStore } from '@philjs/ssr';
import Redis from 'ioredis';

const redis = new Redis();

// Replace default store with Redis
class RedisCSRFStore {
  async set(sessionId: string, token: string, ttl: number = 3600000) {
    await redis.set(`csrf:${sessionId}`, token, 'PX', ttl);
  }

  async get(sessionId: string): Promise<string | null> {
    return await redis.get(`csrf:${sessionId}`);
  }

  async verify(sessionId: string, token: string): Promise<boolean> {
    const stored = await this.get(sessionId);
    return stored === token;
  }
}

// Use custom store
const customStore = new RedisCSRFStore();

```

Extract CSRF Token

```

import { extractCSRFToken } from '@philjs/ssr';

export async function POST({ request }: { request: Request }) {
  const token = await extractCSRFToken(request);

  if (!token) {
    return new Response('Missing CSRF token', { status: 400 });
  }

  // Verify token...
}

```

SameSite Cookies

In addition to CSRF tokens, use SameSite cookies:

```
// Server-side (example)
res.cookie('session', sessionId, {
  httpOnly: true,
  secure: true,
  sameSite: 'strict',
  maxAge: 24 * 60 * 60 * 1000 // 24 hours
});
```

Authentication

Secure Token Storage

```
// ☹ INSECURE - localStorage accessible to XSS
localStorage.setItem('token', authToken);

// ☹ Better - httpOnly cookies (set by server)
// Server sets: Set-Cookie: token=xxx; HttpOnly; Secure; SameSite=Strict

// ☹ If you must use localStorage, validate carefully
class TokenStorage {
  private readonly KEY = 'auth_token';

  setToken(token: string) {
    // Validate token format
    if (!this.isValidToken(token)) {
      throw new Error('Invalid token format');
    }
    localStorage.setItem(this.KEY, token);
  }

  getToken(): string | null {
    const token = localStorage.getItem(this.KEY);
    return token && this.isValidToken(token) ? token : null;
  }

  clearToken() {
    localStorage.removeItem(this.KEY);
  }

  private isValidToken(token: string): boolean {
    // Validate JWT format: header.payload.signature
    return /^[A-Za-z0-9-_]+\.[A-Za-z0-9-_]+\.[A-Za-z0-9-_]+$/ .test(token);
  }
}

export const tokenStorage = new TokenStorage();
```

Password Handling

```

// ⚠ NEVER store passwords in state longer than necessary
function loginForm() {
  const email = signal('');
  const password = signal('');

  const handleSubmit = async (e: Event) => {
    e.preventDefault();

    try {
      await authService.login(email(), password());

      // ⚠ Clear password immediately after use
      password.set('');
    } catch (err) {
      // Handle error
      password.set(''); // Clear even on error
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="email"
        value={email()}
        onChange={(e) => email.set(e.currentTarget.value)}
      />

      <input
        type="password"
        value={password()}
        onChange={(e) => password.set(e.currentTarget.value)}
        autoComplete="current-password"
      />

      <button type="submit">Login</button>
    </form>
  );
}

// ⚠ NEVER log passwords
console.log('Password:', password()); // DON'T DO THIS

// ⚠ NEVER send passwords in URL
fetch('/login?password=${password()}'); // DON'T DO THIS

```

Session Management

```

// stores/authStore.ts
function createAuthStore() {
  const user = signal<User | null>(null);
  const sessionExpiry = signal<number | null>(null);

  // Check session expiry
  effect(() => {
    if (!sessionExpiry()) return;

    const checkExpiry = setInterval(() => {
      if (Date.now() > sessionExpiry().value) {
        logout();
      }
    }, 60000); // Check every minute

    return () => clearInterval(checkExpiry);
  });

  const login = async (credentials: Credentials) => {
    const response = await authService.login(credentials);

    user.set(response.user);
    sessionExpiry.set(Date.now() + response.expiresIn);
  };

  const logout = () => {
    user.set(null);
    sessionExpiry.set(null);
    tokenStorage.clearToken();

    // Redirect to login
    window.location.href = '/login';
  };

  const refreshSession = async () => {
    const response = await authService.refresh();
    sessionExpiry.set(Date.now() + response.expiresIn);
  };

  return { user, login, logout, refreshSession };
}

```

Authorization

Role-Based Access Control

```

// types/auth.ts
export type Role = 'admin' | 'editor' | 'viewer';

export interface User {
  id: string;
  name: string;
  email: string;
  role: Role;
}

// hooks/usePermissions.ts
import { userStore } from '@/stores/userStore';

export function usePermissions() {
  const { user } = userStore;

  const hasRole = (role: Role): boolean => {
    return user()?.role === role;
  };

  const canEdit = (): boolean => {
    const role = user()?.role;
    return role === 'admin' || role === 'editor';
  };

  const canDelete = (): boolean => {
    return user()?.role === 'admin';
  };

  return {
    hasRole,
    canEdit,
    canDelete
  };
}

// Usage in components
function DocumentActions({ document }: { document: Document }) {
  const { canEdit, canDelete } = usePermissions();

  return (
    <div>
      {canEdit() && (
        <button onClick={() => editDocument(document)}>Edit</button>
      )}
      {canDelete() && (
        <button onClick={() => deleteDocument(document)}>Delete</button>
      )}
    </div>
  );
}

```

Route Guards

```

// components/ProtectedRoute.tsx
import { Navigate } from '@philjs/router';
import { userStore } from '@stores/userStore';

interface ProtectedRouteProps {
  component: () => JSX.Element;
  requiredRole?: Role;
}

export function ProtectedRoute({
  component,
  requiredRole
}: ProtectedRouteProps) {
  const { user, isAuthenticated } = userStore;

  if (!isAuthenticated()) {
    return <Navigate to="/login" />;
  }

  if (requiredRole && user()?.role !== requiredRole) {
    return <Navigate to="/forbidden" />;
  }

  return <Component />;
}

// Usage
<Route
  path="/admin"
  component={() => (
    <ProtectedRoute component={AdminDashboard} requiredRole="admin" />
  )}
/>

```

Input Validation

Client-Side Validation

```

// utils/validation.ts
export const validators = {
  email(value: string): string | null {
    if (!value) return 'Email is required';
    if (!/![^@\s]+@[^\s@]+\.[^\s@]+$/ .test(value)) {
      return 'Invalid email format';
    }
    return null;
  },

  password(value: string): string | null {
    if (!value) return 'Password is required';
    if (value.length < 8) {
      return 'Password must be at least 8 characters';
    }
    if (!/[A-Z]/ .test(value)) {
      return 'Password must contain uppercase letter';
    }
    if (!/[a-z]/ .test(value)) {
      return 'Password must contain lowercase letter';
    }
    if (!/[0-9]/ .test(value)) {
      return 'Password must contain number';
    }
    return null;
  },

  url(value: string): string | null {
    try {
      new URL(value);
      return null;
    } catch {
      return 'Invalid URL';
    }
  },

  maxLength(value: string, max: number): string | null {
    if (value.length > max) {
      return `Must be ${max} characters or less`;
    }
    return null;
  }
};

```

Server-Side Validation

```
// ✅ ALWAYS validate on server
// Client validation is for UX, server validation is for security

// services/userService.ts
export const userService = {
  async createUser(data: CreateUserDTO) {
    // Client does basic validation
    // Server does complete validation

    const response = await api.post('/users', data);

    // Server may return validation errors
    if (response.errors) {
      throw new ValidationError(response.errors);
    }

    return response.data;
  }
};
```

SQL Injection Prevention

Parameterized Queries

```
// ❌ VULNERABLE - string concatenation
const query = `SELECT * FROM users WHERE email = '${email}'`;

// ✅ SAFE - parameterized query (server-side)
const query = 'SELECT * FROM users WHERE email = ?';
db.query(query, [email]);

// ✅ SAFE - ORM (server-side)
const user = await User.findOne({ where: { email } });
```

Secrets Management

Environment Variables

```
// ✅ Use environment variables for secrets
const API_KEY = import.meta.env.VITE_API_KEY;

// ❌ NEVER commit secrets to code
const API_KEY = 'sk_live_abc123...'; // DON'T DO THIS

// .env (NOT in git)
VITE_API_KEY=sk_live_abc123...

// .env.example (in git)
VITE_API_KEY=your_api_key_here
```

Client vs Server Secrets

```
// ✅ Public keys (OK for client)
const STRIPE_PUBLISHABLE_KEY = import.meta.env.VITE_STRIPE_PUBLISHABLE_KEY;

// ❌ NEVER put secret keys in client
const STRIPE_SECRET_KEY = import.meta.env.VITE_STRIPE_SECRET_KEY; // DON'T

// ✅ Server-only secrets
// Keep in server environment, never send to client
const STRIPE_SECRET_KEY = process.env.STRIPE_SECRET_KEY; // Server only
```

Content Security Policy

CSP Headers

```
// Server-side CSP configuration
const cspHeader = [
  "default-src 'self'",
  "script-src 'self' 'unsafe-inline' https://trusted-cdn.com",
  "style-src 'self' 'unsafe-inline' https://fonts.googleapis.com",
  "img-src 'self' data: https:",
  "font-src 'self' https://fonts.gstatic.com",
  "connect-src 'self' https://api.example.com",
  "frame-ancestors 'none'"
].join(' ');

// Set in response headers
res.setHeader('Content-Security-Policy', cspHeader);
```

Rate Limiting

PhilJS provides comprehensive rate limiting for API routes, protecting against abuse and DDoS attacks.

Built-in Rate Limiting

```
import {
  rateLimit,
  apiRateLimit,
  authRateLimit,
  MemoryRateLimitStore
} from '@philjs/ssr';

// Basic rate Limiting
const limiter = rateLimit({
  windowMs: 60 * 1000,           // 1 minute window
  maxRequests: 100,             // 100 requests per window
  message: 'Too many requests'
});

// Use as middleware
app.use(async (request, next) => {
  const rateLimitResponse = await limiter(request, next);
  if (rateLimitResponse) {
    return rateLimitResponse; // 429 Too Many Requests
  }
  return next();
});
```

Pre-configured Rate Limiters

```
// API routes: 60 requests per minute
const apiLimiter = apiRateLimit(60);

// Authentication routes: 5 attempts per minute
const authLimiter = authRateLimit(5);

// API key-based: 1000 requests per minute
const apiKeyLimiter = apiKeyRateLimit(1000);

// User-based: Custom user ID extractor
const userLimiter = userRateLimit(
  100,
  (request) => request.headers.get('x-user-id')
);
```

Custom Key Generation

```
const customLimiter = rateLimit({
  windowMs: 60 * 1000,
  maxRequests: 50,
  keyGenerator: (request) => {
    // Rate limit by API key
    const apiKey = request.headers.get('x-api-key');
    if (apiKey) {
      return `api:${apiKey}`;
    }
  }

  // Fallback to IP
  const ip = request.headers.get('x-forwarded-for') || 'unknown';
  return `ip:${ip}`;
});
```

Redis Store for Production

```
import { RedisRateLimitStore } from '@philjs/ssr';
import Redis from 'ioredis';

const redis = new Redis({
  host: 'localhost',
  port: 6379
});

const store = new RedisRateLimitStore(redis, 'philjs:ratelimit');

const limiter = rateLimit(
  {
    windowMs: 60 * 1000,
    maxRequests: 100
  },
  store
);
```

Sliding Window Rate Limiting

PhilJS includes a more accurate sliding window algorithm:

```

import { SlidingWindowRateLimiter } from '@philjs/ssr';

const limiter = new SlidingWindowRateLimiter({
  windowMs: 60 * 1000,
  maxRequests: 100,
  message: 'Rate limit exceeded'
});

// Check rate limit
const rateLimitResponse = await limiter.check(request);
if (rateLimitResponse) {
  return rateLimitResponse; // 429 if exceeded
}

```

Adaptive Rate Limiting

Automatically adjust limits based on error rates:

```

import { AdaptiveRateLimiter } from '@philjs/ssr';

const limiter = new AdaptiveRateLimiter({
  baseLimit: 100,
  windowMs: 60 * 1000,
  errorThreshold: 0.1,      // 10% error rate triggers reduction
  adaptationFactor: 0.5    // Reduce by 50% when errors high
});

// Use in request handler
const rateLimitResponse = await limiter.check(request);
if (rateLimitResponse) return rateLimitResponse;

// Record result for adaptation
const success = response.status < 400;
await limiter.recordResult(success);

```

Rate Limit Headers

PhilJS automatically adds standard rate limit headers:

```

// Response headers
X-RateLimit-Limit: 100          // Max requests per window
X-RateLimit-Remaining: 95        // Remaining requests
X-RateLimit-Reset: 2024-01-01T12:00:00Z // When limit resets

```

Skip Successful/Failed Requests

```

// Don't count successful login attempts
const loginLimiter = rateLimit({
  windowMs: 60 * 1000,
  maxRequests: 5,
  skipSuccessfulRequests: true // Only count failed logins
});

// Don't count failed requests
const uploadLimiter = rateLimit({
  windowMs: 60 * 1000,
  maxRequests: 10,
  skipFailedRequests: true     // Only count successful uploads
});

```

Custom Error Handling

```

const limiter = rateLimit({
  windowMs: 60 * 1000,
  maxRequests: 100,
  handler: (request) => {
    // Custom 429 response
    return new Response(
      JSON.stringify({
        error: 'Rate limit exceeded',
        retryAfter: 60,
        message: 'Please slow down your requests'
      }),
      {
        status: 429,
        headers: {
          'Content-Type': 'application/json',
          'Retry-After': '60'
        }
      }
    );
  }
});

```

Per-Route Rate Limiting

```
// Different Limits for different routes
app.get('/api/search', apiRateLimit(30), searchHandler);
app.post('/api/login', authRateLimit(5), loginHandler);
app.post('/api/upload', uploadRateLimit(10), uploadHandler);
```

Client-Side Rate Limiting

For client-side protection:

```
function createRateLimiter(maxRequests: number, windowMs: number) {
  const requests: number[] = [];

  return {
    canMakeRequest(): boolean {
      const now = Date.now();

      // Remove old requests outside window
      while (requests.length > 0 && requests[0] < now - windowMs) {
        requests.shift();
      }

      if (requests.length >= maxRequests) {
        return false;
      }

      requests.push(now);
      return true;
    },

    getRemainingRequests(): number {
      return Math.max(0, maxRequests - requests.length);
    },

    getResetTime(): number {
      return requests[0] + windowMs;
    }
  };
}

// Usage in component
function SearchForm() {
  const rateLimiter = createRateLimiter(5, 60000); // 5/min
  const query = signal('');

  const search = async () => {
    if (!rateLimiter.canMakeRequest()) {
      const resetTime = rateLimiter.getResetTime();
      const waitSeconds = Math.ceil((resetTime - Date.now()) / 1000);

      alert(`Rate limit exceeded. Try again in ${waitSeconds} seconds.`);
      return;
    }

    const results = await api.get(`/search?q=${query()}`);
    // ...
  };

  return (
    <div>
      <input value={query()} onChange={(e) => query.set(e.target.value)} />
      <button onClick={search}>
        Search ({rateLimiter.getRemainingRequests()} remaining)
      </button>
    </div>
  );
}
```

Monitor Rate Limit Usage

```

// Get rate limit info
const info = await limiter.getRateLimitInfo(request);

console.log(`Limit: ${info.limit}`);
console.log(`Remaining: ${info.remaining}`);
console.log(`Resets at: ${new Date(info.reset)}`);

// Show in UI
function RateLimitStatus() {
  const info = signal<RateLimitInfo | null>(null);

  effect(async () => {
    const response = await fetch('/api/rate-limit-status');
    info.set(await response.json());
  });
}

return info() ? (
  <div class="rate-limit-status">
    Requests: {info().remaining}/{info().limit}
    <progress value={info().remaining} max={info().limit} />
  </div>
) : null;
}

```

Best Practices

```

// ⚡ Use different limits for different routes
app.post('/api/auth/login', authRateLimit(5));           // Strict
app.get('/api/products', apiRateLimit(100));             // Generous
app.post('/api/upload', uploadRateLimit(10));             // Moderate

// ⚡ Use Redis in production (distributed)
const redisStore = new RedisRateLimitStore(redis);
const limiter = rateLimit(config, redisStore);

// ⚡ Whitelist trusted IPs
const limiter = rateLimit({
  windowMs: 60000,
  maxRequests: 100,
  keyGenerator: (request) => {
    const ip = request.headers.get('x-forwarded-for');
    const trustedIPs = ['10.0.0.1', '192.168.1.1'];

    if (trustedIPs.includes(ip)) {
      return 'trusted'; // Higher or no limit
    }

    return `ip:${ip}`;
  }
});

// ⚡ Log rate limit violations
const limiter = rateLimit({
  windowMs: 60000,
  maxRequests: 100,
  handler: (request) => {
    const ip = request.headers.get('x-forwarded-for');
    console.warn(`Rate limit exceeded for IP: ${ip}`);
  }
});

// Alert if too many violations
violations++;
if (violations > 100) {
  alertSecurity(`High rate limit violations: ${violations}`);
}

return new Response('Too many requests', { status: 429 });
}
);

```

Dependency Security

Regular Updates

```

# Check for vulnerabilities
npm audit

# Fix vulnerabilities
npm audit fix

# Update dependencies
npm update

# Check for outdated packages
npm outdated

```

Secure Dependencies

```
// package.json
{
  "scripts": {
    "audit": "npm audit",
    "audit:fix": "npm audit fix"
  }
}
```

Security Headers

Essential Headers

```
// Server-side security headers
res.setHeader('X-Content-Type-Options', 'nosniff');
res.setHeader('X-Frame-Options', 'DENY');
res.setHeader('X-XSS-Protection', '1; mode=block');
res.setHeader('Referrer-Policy', 'strict-origin-when-cross-origin');
res.setHeader('Permissions-Policy', 'geolocation=(), microphone=()');

// HTTPS only
res.setHeader('Strict-Transport-Security', 'max-age=31536000; includeSubDomains');
```

File Upload Security

Validate File Uploads

```
function FileUpload() {
  const MAX_FILE_SIZE = 5 * 1024 * 1024; // 5MB
  const ALLOWED_TYPES = ['image/jpeg', 'image/png', 'image/gif'];

  const handleFileUpload = async (e: Event) => {
    const input = e.target as HTMLInputElement;
    const file = input.files?[0];

    if (!file) return;

    // ✅ Validate file size
    if (file.size > MAX_FILE_SIZE) {
      alert('File too large. Maximum size is 5MB.');
      return;
    }

    // ✅ Validate file type
    if (!ALLOWED_TYPES.includes(file.type)) {
      alert('Invalid file type. Only JPEG, PNG, and GIF allowed.');
      return;
    }

    // ✅ Validate file extension
    const extension = file.name.split('.').pop()?.toLowerCase();
    if (!['jpg', 'jpeg', 'png', 'gif'].includes(extension || '')) {
      alert('Invalid file extension.');
      return;
    }

    // Upload to server
    const formData = new FormData();
    formData.append('file', file);

    await api.post('/upload', formData);
  };

  return (
    <input
      type="file"
      accept={ALLOWED_TYPES.join(',')}.
      onChange={handleFileUpload}
    />
  );
}
```

Summary

Security Best Practices:

- Sanitize all user input □ Use CSRF tokens for state-changing requests □ Store tokens securely (httpOnly cookies preferred) □ Validate on both client and server □ Use parameterized queries □ Never commit secrets to code □ Implement role-based access control □ Use security headers □ Rate limit requests □ Keep dependencies updated □ Validate file uploads □ Use HTTPS in production □ Implement Content Security Policy

Next: [Accessibility →](#)

Production Best Practices

Deploy PhilJS applications with confidence.

Build Optimization

Production Build

```
// package.json
{
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  }
}

# Build for production
npm run build

# Preview production build locally
npm run preview
```

Vite Configuration

```
// vite.config.ts
import { defineConfig } from 'vite';

export default defineConfig({
  build: {
    // Generate source maps for debugging
    sourcemap: true,

    // Minify code
    minify: 'terser',

    // Terser options
    terserOptions: {
      compress: {
        drop_console: true, // Remove console.log in production
        drop_debugger: true
      }
    },
    // Code splitting
    rollupOptions: {
      output: {
        manualChunks: {
          // Vendor chunks
          vendor: ['@philjs/core', '@philjs/router']
        }
      }
    },
    // Feature chunks
    dashboard: ['./src/features/dashboard'],
    analytics: ['./src/features/analytics']
  },
  // Asset optimization
  assetsInlineLimit: 4096, // 4kb

  // Chunk size warnings
  chunkSizeWarningLimit: 500
},
// Server configuration for preview
preview: {
  port: 3000,
  strictPort: true
});
});
```

Environment Configuration

Environment Variables

```

# .env.production
VITE_API_URL=https://api.production.com
VITE_APP_NAME=PhilJS App
VITE_ENABLE_ANALYTICS=true
VITE_SENTRY_DSN=https://...

# .env.staging
VITE_API_URL=https://api.staging.com
VITE_APP_NAME=PhilJS App (Staging)
VITE_ENABLE_ANALYTICS=false

# .env.development
VITE_API_URL=http://localhost:3000
VITE_APP_NAME=PhilJS App (Dev)
VITE_ENABLE_ANALYTICS=false

```

```

// utils/env.ts
interface Environment {
  apiUrl: string;
  appName: string;
  enableAnalytics: boolean;
  sentryDsn: string;
  isDevelopment: boolean;
  isProduction: boolean;
}

export const env: Environment = {
  apiUrl: import.meta.env.VITE_API_URL,
  appName: import.meta.env.VITE_APP_NAME,
  enableAnalytics: import.meta.env.VITE_ENABLE_ANALYTICS === 'true',
  sentryDsn: import.meta.env.VITE_SENTRY_DSN || '',
  isDevelopment: import.meta.env.DEV,
  isProduction: import.meta.env.PROD
};

// Validate required environment variables
const requiredEnvVars = ['VITE_API_URL'];

requiredEnvVars.forEach(key => {
  if (!import.meta.env[key]) {
    throw new Error(`Missing required environment variable: ${key}`);
  }
});

```

Error Tracking

Sentry Integration

```

// services/errorTracking.ts
import * as Sentry from '@sentry/browser';
import { env } from '@/utils/env';

export function initErrorTracking() {
  if (!env.isProduction || !env.sentryDsn) return;

  Sentry.init({
    dsn: env.sentryDsn,
    environment: env.isProduction ? 'production' : 'staging',

    // Performance monitoring
    tracesSampleRate: 0.1, // 10% of transactions

    // Release tracking
    release: `philjs-app@${import.meta.env.VITE_APP_VERSION}`,

    // Error filtering
    beforeSend(event, hint) {
      // Don't send errors from browser extensions
      if (event.exception?.values?.[0]?.stacktrace?.frames?.some(
        frame => frame.filename?.includes('extension://')
      )) {
        return null;
      }

      return event;
    },

    // Ignore certain errors
    ignoreErrors: [
      'ResizeObserver loop limit exceeded',
      'Non-Error promise rejection captured'
    ]
  });
}

// Track user for better error context
export function setUser(user: { id: string; email: string }) {
  Sentry.setUser({
    id: user.id,
    email: user.email
  });
}

// Clear user on Logout
export function clearUser() {
  Sentry.setUser(null);
}

// Manually capture exceptions
export function captureException(error: Error, context?: Record<string, any>) {
  if (env.isDevelopment) {
    console.error(error, context);
    return;
  }

  Sentry.withScope(scope => {
    if (context) {
      scope.setContext('additional', context);
    }
    Sentry.captureException(error);
  });
}

// App.tsx
import { initErrorTracking } from './services/errorTracking';

initErrorTracking();

function App() {
  return <Router>...</Router>;
}

```

Global Error Boundary

```
import { ErrorBoundary } from '@philjs/core';
import { captureException } from './services/errorTracking';

function App() {
  return (
    <ErrorBoundary
      fallback={(error) => <ErrorFallback error={error} />}
      onError={(error, errorInfo) => {
        captureException(error, {
          componentStack: errorInfo.componentStack
        });
      }}
    >
      <Router>
        <Routes />
      </Router>
    </ErrorBoundary>
  );
}

function ErrorFallback({ error }: { error: Error }) {
  return (
    <div className="error-page">
      <h1>Something went wrong</h1>
      <p>We've been notified and are working on a fix.</p>

      {env.isDevelopment && (
        <details>
          <summary>Error Details</summary>
          <pre>{error.message}</pre>
          <pre>{error.stack}</pre>
        </details>
      )}
      <button onClick={() => window.location.reload()}>
        Reload Page
      </button>
    </div>
  );
}
```

Analytics

Analytics Setup

```

// services/analytics.ts
import { env } from '@/utils/env';

class Analytics {
    private initialized = false;

    init() {
        if (!env.enableAnalytics || this.initialized) return;

        // Initialize your analytics service (e.g., Google Analytics, Plausible)
        // @ts-ignore
        window.gtag('config', env.googleAnalyticsId);

        this.initialized = true;
    }

    trackPageView(path: string) {
        if (!this.initialized) return;

        // @ts-ignore
        window.gtag('event', 'page_view', {
            page_path: path
        });
    }

    trackEvent(eventName: string, params?: Record<string, any>) {
        if (!this.initialized) return;

        // @ts-ignore
        window.gtag('event', eventName, params);
    }

    trackUser(userId: string) {
        if (!this.initialized) return;

        // @ts-ignore
        window.gtag('set', { user_id: userId });
    }

    export const analytics = new Analytics();

// App.tsx
import { analytics } from './services/analytics';
import { router } from '@philjs/router';

analytics.init();

// Track route changes
effect(() => {
    const route = router.currentRoute();
    analytics.trackPageView(route.path);
});

// Track custom events
analytics.trackEvent('purchase', {
    value: 99.99,
    currency: 'USD',
    items: ['product-1', 'product-2']
});

```

Performance Monitoring

Web Vitals

```
// services/performance.ts

import { onCLS, onFID, onLCP, onFCP, onTTFB } from 'web-vitals';

function sendToAnalytics(metric: any) {
  // Send to your analytics service
  analytics.trackEvent('web_vitals', {
    name: metric.name,
    value: metric.value,
    rating: metric.rating
  });
}

export function initPerformanceMonitoring() {
  onCLS(sendToAnalytics); // Cumulative Layout Shift
  onFID(sendToAnalytics); // First Input Delay
  onLCP(sendToAnalytics); // Largest Contentful Paint
  onFCP(sendToAnalytics); // First Contentful Paint
  onTTFB(sendToAnalytics); // Time to First Byte
}

// App.tsx
import { initPerformanceMonitoring } from './services/performance';

initPerformanceMonitoring();
```

Logging

Structured Logging

```

// services/Logger.ts
enum LogLevel {
  DEBUG = 0,
  INFO = 1,
  WARN = 2,
  ERROR = 3
}

class Logger {
  private level: LogLevel;

  constructor() {
    this.level = env.isDevelopment ? LogLevel.DEBUG : LogLevel.WARN;
  }

  private log(level: LogLevel, message: string, data?: any) {
    if (level < this.level) return;

    const logData = {
      timestamp: new Date().toISOString(),
      level: LogLevel[level],
      message,
      ...data
    };

    switch (level) {
      case LogLevel.DEBUG:
        console.debug(logData);
        break;
      case LogLevel.INFO:
        console.info(logData);
        break;
      case LogLevel.WARN:
        console.warn(logData);
        break;
      case LogLevel.ERROR:
        console.error(logData);
        captureException(new Error(message), data);
        break;
    }
  }

  debug(message: string, data?: any) {
    this.log(LogLevel.DEBUG, message, data);
  }

  info(message: string, data?: any) {
    this.log(LogLevel.INFO, message, data);
  }

  warn(message: string, data?: any) {
    this.log(LogLevel.WARN, message, data);
  }

  error(message: string, data?: any) {
    this.log(LogLevel.ERROR, message, data);
  }
}

export const logger = new Logger();

// Usage
logger.debug('User clicked button', { userId: '123', buttonId: 'submit' });
logger.info('User logged in', { userId: '123' });
logger.warn('API request slow', { endpoint: '/users', duration: 2000 });
logger.error('Payment failed', { userId: '123', error: 'Card declined' });

```

Caching Strategy

Service Worker

```

// src/sw.ts (compiled to public/sw.js)
const CACHE_NAME = 'philjs-app-v1';

const urlsToCache = [
  '/',
  '/index.html',
  '/assets/main.js',
  '/assets/main.css'
];

self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => cache.addAll(urlsToCache))
  );
});

self.addEventListener('fetch', (event) => {
  event.respondWith(
    caches.match(event.request)
      .then(response => {
        // Cache hit - return response
        if (response) {
          return response;
        }

        // Clone the request
        const fetchRequest = event.request.clone();

        return fetch(fetchRequest).then(response => {
          // Check if valid response
          if (!response || response.status !== 200 || response.type !== 'basic') {
            return response;
          }

          // Clone the response
          const responseToCache = response.clone();

          caches.open(CACHE_NAME)
            .then(cache => {
              cache.put(event.request, responseToCache);
            });
        });

        return response;
      });
  );
});

// index.tsx - Register service worker
if ('serviceWorker' in navigator && env.isProduction) {
  window.addEventListener('load', () => {
    navigator.serviceWorker.register('/sw.js')
      .then(registration => {
        console.log('SW registered:', registration);
      })
      .catch(error => {
        console.log('SW registration failed:', error);
      });
  });
}

```

Health Checks

Application Health

```

// services/health.ts
export async function checkHealth() {
  const health = {
    status: 'healthy',
    timestamp: new Date().toISOString(),
    version: import.meta.env.VITE_APP_VERSION,
    checks: {
      api: await checkApiHealth(),
      database: await checkDatabaseHealth(),
      cache: checkCacheHealth()
    }
  };

  const unhealthy = Object.values(health.checks).some(
    check => check.status === 'unhealthy'
  );

  if (unhealthy) {
    health.status = 'unhealthy';
  }

  return health;
}

async function checkApiHealth() {
  try {
    const response = await fetch(`${env.apiUrl}/health`, { timeout: 5000 });
    return {
      status: response.ok ? 'healthy' : 'unhealthy',
      responseTime: response.headers.get('x-response-time')
    };
  } catch (error) {
    return {
      status: 'unhealthy',
      error: error.message
    };
  }
}

// Expose health check endpoint
app.get('/health', async (req, res) => {
  const health = await checkHealth();
  const statusCode = health.status === 'healthy' ? 200 : 503;
  res.status(statusCode).json(health);
});

```

Deployment

CI/CD Pipeline

```

# .github/workflows/deploy.yml
name: Deploy

on:
  push:
    branches: [main]

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '24'

      - name: Install dependencies
        run: npm ci

      - name: Run tests
        run: npm test

      - name: Run linter
        run: npm run lint

      - name: Build
        run: npm run build
        env:
          VITE_API_URL: ${{ secrets.PRODUCTION_API_URL }}
          VITE_SENTRY_DSN: ${{ secrets.SENTRY_DSN }}

      - name: Deploy to production
        run: npm run deploy
        env:
          DEPLOY_TOKEN: ${{ secrets.DEPLOY_TOKEN }}

```

Docker

```

# Dockerfile
FROM node:24-alpine AS builder

WORKDIR /app

COPY package*.json .
RUN npm ci

COPY . .
RUN npm run build

FROM nginx:alpine

COPY --from=builder /app/dist /usr/share/nginx/html
COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

```

Zero-Downtime Deployment

```

# Using blue-green deployment
# 1. Deploy new version (green)
# 2. Run health checks
# 3. Switch traffic to green
# 4. Keep blue as backup
# 5. Remove blue after verification

```

Monitoring

Uptime Monitoring

```

// Setup external monitoring
// - Pingdom
// - UptimeRobot
// - StatusCake

// Check critical endpoints every 1-5 minutes:
// - Homepage (/)
// - API health (/api/health)
// - Login page (/Login)

```

Alerting

```
// Configure alerts for:  
// - Error rate > 1%  
// - Response time > 3s  
// - Availability < 99.9%  
// - Build failures  
// - Deployment failures
```

Security Headers

```
# nginx.conf  
server {  
    # Security headers  
    add_header X-Frame-Options "DENY" always;  
    add_header X-Content-Type-Options "nosniff" always;  
    add_header X-XSS-Protection "1; mode=block" always;  
    add_header Referrer-Policy "strict-origin-when-cross-origin" always;  
    add_header Content-Security-Policy "default-src 'self'; script-src 'self' 'unsafe-inline'; style-src 'self' 'unsafe-inline';" always;  
  
    # HTTPS only  
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;  
  
    # Gzip compression  
    gzip on;  
    gzip_types text/plain text/css application/json application/javascript text/xml application/xml;  
  
    # Cache static assets  
    location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg|woff|woff2)$ {  
        expires 1y;  
        add_header Cache-Control "public, immutable";  
    }  
}
```

Summary

Production Best Practices:

- Optimize build with code splitting
- Configure environment variables
- Integrate error tracking (Sentry)
- Set up analytics and web vitals
- Implement structured logging
- Use service workers for caching
- Add health check endpoints
- Set up CI/CD pipeline
- Configure security headers
- Monitor uptime and performance
- Plan zero-downtime deployments
- Set up alerts for critical issues
- Test in staging before production
- Have rollback strategy ready

Complete! You now have comprehensive best practices for PhilJS development.

Return to [Best Practices Overview](#) for navigation.

TypeScript 6 Best Practices

Write type-safe PhilJS applications with TypeScript 6+.

Requirements

- TypeScript 6.0+ - Required for all features documented here
- Node.js 24+ - Required for native ESM support

Recommended tsconfig.json

```
{  
    "compilerOptions": {  
        "target": "ES2024",  
        "lib": ["ES2024", "DOM", "DOM.Iterable"],  
        "module": "NodeNext",  
        "moduleResolution": "NodeNext",  
        "jsx": "preserve",  
        "jsxImportSource": "@philjs/core",  
        "strict": true,  
        "isolatedDeclarations": true,  
        "exactOptionalPropertyTypes": true,  
        "noUncheckedIndexedAccess": true,  
        "noUnusedLocals": true,  
        "noUnusedParameters": true,  
        "noFallthroughCasesInSwitch": true  
    }  
}
```

Component Types

```

import { signal } from '@philjs/core';

interface ButtonProps {
  label: string;
  onClick: () => void;
  variant?: 'primary' | 'secondary';
}

function Button({ label, onClick, variant = 'primary' }: ButtonProps) {
  return (
    <button onClick={onClick} class={`btn-${variant}`}>
      {label}
    </button>
  );
}

```

Signal Types

```

import { signal, Signal } from '@philjs/core';

const count = signal<number>(0);
const user = signal<User | null>(null);

```

TypeScript 6 Features

Use the satisfies Operator

```

// Type-safe configuration with full inference
const config = {
  theme: 'dark',
  fontSize: 16,
  features: ['search', 'analytics'],
} as const satisfies AppConfig;

// config.theme is narrowed to 'dark', not string

```

Use NoInfer for Better Generic Control

```

import { signal } from '@philjs/core';

// Prevent unwanted inference from default values
function createSignalWithDefault<T>(
  initialValue: T,
  defaultValue: NoInfer<T>
): Signal<T> {
  return signal(initialValue ?? defaultValue);
}

// T is inferred from initialValue, not defaultValue
const count = createSignalWithDefault(0, 100);

```

Use Const Type Parameters

```

// Preserve literal types in generics
function createStore<const T extends Record<string, unknown>>(
  initial: T
): Store<T> {
  return new Store(initial);
}

// Keys are preserved as literal types
const store = createStore({
  count: 0,
  name: 'app',
});

```

Isolated Declarations

Enable faster builds with isolated declarations:

```

{
  "compilerOptions": {
    "isolatedDeclarations": true,
    "declaration": true
  }
}

```

This requires explicit return types on exported functions:

```
// Required with isolatedDeclarations
export function createCounter(): { count: Signal<number>; increment: () => void } {
  const count = signal(0);
  return {
    count,
    increment: () => count.set(c => c + 1),
  };
}
```

Best Practices

Use Type Inference When Possible

```
// Good - inferred as Signal<number>
const count = signal(0);

// Explicit type only when needed
const user = signal<User | null>(null);
```

Type Props Explicitly

```
// Good - typed props with TypeScript 6 features
interface Props {
  name: string;
  config?: AppConfig;
}

function Component({ name, config }: Props) {
  // ...
}
```

Use Strict Optional Properties

```
// With exactOptionalPropertyTypes enabled
interface UserSettings {
  theme?: 'light' | 'dark'; // Can be 'light', 'dark', or missing
  notifications?: boolean; // Can be true, false, or missing
}

// undefined must be explicit if allowed
interface UserSettingsExplicit {
  theme: 'light' | 'dark' | undefined; // Can also be undefined
}
```

Safe Index Access

```
// With noUncheckedIndexedAccess enabled
const items: string[] = ['a', 'b', 'c'];

// item is string | undefined, not string
const item = items[0];

// Must handle undefined
if (item !== undefined) {
  console.log(item.toUpperCase());
}

// Or use at() with nullish coalescing
const safeItem = items.at(0) ?? 'default';
```

ES2024 Type Support

TypeScript 6 includes types for ES2024 features:

```
// Object.groupBy() returns Partial<Record<K, T[]>
const grouped: Partial<Record<string, User[]> = Object.groupBy(
  users,
  user => user.role
);

// Map.groupBy() returns Map<K, T[]>
const mappedGroups: Map<string, User[]> = Map.groupBy(
  users,
  user => user.department
);

// Promise.withResolvers() is fully typed
const { promise, resolve, reject } = Promise.withResolvers<User>();

// Array immutable methods preserve types
const sorted: readonly User[] = users.toSorted((a, b) =>
  a.name.localeCompare(b.name)
);
```

Next Steps

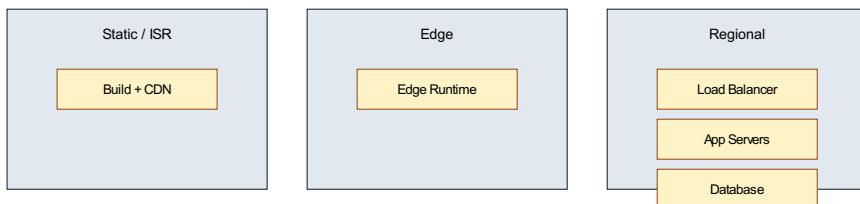
- [Performance Best Practices](#) - Optimize your TypeScript code
- [API Reference](#) - Full API documentation

Tip: Enable all strict mode options in `tsconfig.json` for maximum type safety.

Note: PhilJS is built with TypeScript 6 and provides excellent type inference out of the box.

Deployment

PhilJS supports static, SSR, and hybrid deployments.



Deployment topologies

Build

```
pnpm build
```

Preview

```
pnpm dev
pnpm build
philjs preview
```

Configure output

```
philjs.config.ts
```

```
import { defineConfig } from "philjs-cli";

export default defineConfig({
  build: {
    outDir: "dist",
    ssg: false,
  },
  dev: {
    port: 3000,
  },
});
```

Runtime requirements

- Node 24+ for SSR runtimes
- Edge targets when using `@philjs/ssr` worker adapters

Deployment Playbook (Advanced)

Step-by-step guidance for shipping PhilJS to edge and serverless targets with confidence.

Pre-deploy checklist

- `pnpm build` succeeds for client + server bundles.
- `pnpm size` and `perf` benches pass budgets.
- Env vars defined with fallbacks; secrets not bundled.
- SSR smoke test locally (preview) with Playwright traces.
- Cache tags and revalidate hints set on critical routes.

Edge (Vercel/Netlify/Cloudflare)

- Prefer edge runtime for latency-sensitive routes.
- Keep bundles small; avoid Node-only APIs.
- Set ISR/edge cache lifetimes and pair with router invalidation.

- Bind KV/D1/R2 or platform equivalents for hot data.
- Stream SSR; hydrate islands lazily.

AWS/Node

- Use HTTP API for lower latency; keep functions small.
- Separate heavy tasks (PDF/image) into dedicated functions.
- Optimize cold starts: minimal deps, small Layers.
- Set memory/timeouts per function; monitor with metrics.

Static/SSG

- Use `philjs-adapters/static` for fully static routes.
- Pre-generate sitemap/robots; set cache headers.
- Combine static shell with client-side data fetching if needed.

CI/CD template

```
pnpm install --frozen-lockfile
pnpm lint
pnpm typecheck
pnpm test -- --runInBand
pnpm size
pnpm build
pnpm exec playwright test --reporter=line
```

Upload Playwright traces on failure; cache pnpm store.

Observability on deploy

- Emit build id/version into responses.
- Log cache hit/miss, TTFB, and loader timings.
- Alert on error rate, cold starts, and budget breaches post-deploy.

Rollback strategy

- Keep previous build artifact; enable traffic flip/feature flags.
- Roll back quickly on TTFB/error spikes; invalidate caches to clear bad HTML/data.

Security

- Lock CSP; avoid remote eval.
- Validate env presence at startup; fail fast.
- Strip `__proto__`/constructor keys from JSON input.

Smoke tests post-deploy

- Hit top routes with Playwright; assert no console errors.
- Check cache headers and TTFB via `curl -I -w`.
- Validate SSR + hydration for dynamic routes.

Checklist

- Budgets (size/perf) enforced in CI.
- Edge/ISR configs set; caches paired with invalidation.
- Env/secrets handled correctly per platform.
- Observability dashboards updated for release.
- Rollback plan in place.

Security Overview

PhilJS is built with security as a core principle. This guide provides an overview of the security features, best practices, and considerations when building applications with PhilJS.

Table of Contents

- [1. Security Features](#)
- [2. Security Architecture](#)
- [3. Common Vulnerabilities](#)
- [4. Security Checklist](#)
- [5. Reporting Security Issues](#)

Security Features

Built-in XSS Protection

PhilJS automatically escapes user input during server-side rendering and client-side hydration:

- **Automatic HTML Escaping:** All text content is escaped by default
- **Attribute Sanitization:** HTML attributes are properly escaped
- **Script Context Protection:** User data in script tags is escaped
- **Safe JSON Serialization:** Prevents prototype pollution attacks

See [XSS Prevention Guide](#) for detailed information.

Content Security Policy (CSP)

PhilJS provides utilities for implementing strict Content Security Policies:

- **Nonce-based Script Loading:** Eliminate unsafe-inline for scripts
- **Configurable Directives:** Fine-grained control over resource loading
- **CSP Report Monitoring:** Track and respond to policy violations
- **Development Mode Support:** Relaxed policies for HMR and debugging

See [CSP Guide](#) for implementation details.

CSRF Protection

Built-in CSRF protection for form submissions and API calls:

- **Token Generation:** Cryptographically secure tokens
- **Automatic Validation:** Middleware for request verification
- **Double Submit Cookies:** Additional protection layer
- **Session Integration:** Works with existing session management

See [API Security Guide](#) for CSRF implementation.

Secure Cookie Handling

PhilJS provides utilities for secure cookie management:

- **HttpOnly Cookies:** Prevent XSS access to sensitive cookies
- **Secure Flag:** HTTPS-only transmission
- **SameSite Protection:** CSRF protection via cookie attributes
- **Signed Cookies:** Tamper detection with HMAC signatures
- **Automatic Encryption:** Optional cookie value encryption

Input Validation and Sanitization

Security utilities for safe data handling:

```
import {
  escapeHtml,
  sanitizeHtml,
  sanitizeUrl,
  safeJsonParse
} from '@philjs/core';

// Escape user input
const safe = escapeHtml(userInput);

// Sanitize HTML content
const clean = sanitizeHtml(richText, {
  allowedTags: ['p', 'strong', 'em'],
});

// Validate URLs
const validUrl = sanitizeUrl(redirectUrl, ['example.com']);

// Safe JSON parsing (prevents prototype pollution)
const data = safeJsonParse(jsonString);
```

Security Architecture

Defense in Depth

PhilJS employs multiple layers of security:

1. **Input Validation:** Validate and sanitize all user input
2. **Output Encoding:** Escape data based on context (HTML, JS, URL)
3. **CSP Headers:** Restrict resource loading and execution
4. **CSRF Tokens:** Prevent cross-site request forgery
5. **Secure Defaults:** Security-first configuration out of the box

SSR Security Model

Server-side rendering introduces unique security considerations:

- **Server-Side Validation:** Never trust client-side validation alone

- **State Serialization:** Sanitize state before embedding in HTML
- **Hydration Integrity:** Verify hydration data hasn't been tampered with
- **Secret Management:** Keep server secrets out of client bundles

Client-Side Security

Protecting the browser environment:

- **Sandboxed Execution:** Isolate untrusted content
- **Secure State Management:** Prevent state injection attacks
- **Safe Hydration:** Validate server-rendered content before hydration
- **Event Handler Security:** Sanitize event handler attributes

Common Vulnerabilities

Cross-Site Scripting (XSS)

Risk: Attackers inject malicious scripts into your application.

PhilJS Protection: - Automatic HTML escaping in templates - CSP to block inline scripts - Sanitization utilities for rich content

Your Responsibility: - Never use `dangerouslySetInnerHTML` with unsanitized input - Validate and sanitize user-generated content - Use CSP nonces for legitimate inline scripts

Example:

```
// BAD - Vulnerable to XSS
<div dangerouslySetInnerHTML={{ __html: userComment }} />

// GOOD - Safe rendering
import { sanitizeHtml } from '@philjs/core';
<div>{sanitizeHtml(userComment)}</div>
```

Cross-Site Request Forgery (CSRF)

Risk: Attackers trick users into performing unwanted actions.

PhilJS Protection: - Built-in CSRF token generation - SameSite cookie attributes - Origin validation

Your Responsibility: - Enable CSRF protection for state-changing operations - Use POST for non-idempotent actions - Validate the origin header

Example:

```
import { csrfProtection } from '@philjs/ssr';

const csrf = csrfProtection({
  getSessionId: (req) => req.headers.get('session-id') || 'default',
});

// Generate token for forms
const token = csrf.generateToken(request);

// Verify on submission
if (!csrf.verifyRequest(request)) {
  return new Response('CSRF validation failed', { status: 403 });
}
```

Prototype Pollution

Risk: Attackers modify `Object.prototype`, affecting all objects.

PhilJS Protection: - Safe JSON parsing utilities - Input sanitization - No direct prototype manipulation

Your Responsibility: - Use `safeJsonParse` instead of `JSON.parse` for untrusted data - Validate object keys before using them - Avoid using `Object.assign` with user input

Example:

```
import { safeJsonParse } from '@philjs/core';

// BAD - Vulnerable to prototype pollution
const data = JSON.parse(userInput);

// GOOD - Protected against prototype pollution
const data = safeJsonParse(userInput);
```

SQL Injection

Risk: Attackers manipulate database queries.

PhilJS Protection: - Framework doesn't include database layer (reduces attack surface) - Recommendations for safe database practices

Your Responsibility: - Always use parameterized queries or ORMs - Never concatenate user input into SQL - Validate and sanitize database inputs

Example:

```
// BAD - SQL Injection vulnerability
db.query(`SELECT * FROM users WHERE id = ${userId}`);

// GOOD - Parameterized query
db.query('SELECT * FROM users WHERE id = ?', [userId]);
```

Open Redirect

Risk: Attackers redirect users to malicious sites.

PhilJS Protection: - URL sanitization utilities - Allowlist-based validation

Your Responsibility: - Validate redirect URLs - Use allowlists for external domains - Avoid user-controlled redirects when possible

Example:

```
import { sanitizeUrl } from '@philjs/core';

// Validate redirect URL
const redirectUrl = sanitizeUrl(
  userProvidedUrl,
  ['example.com', 'trusted-domain.com']
);

if (redirectUrl) {
  return Response.redirect(redirectUrl);
} else {
  return Response.redirect('/');
}
```

Server-Side Request Forgery (SSRF)

Risk: Attackers make the server send requests to internal resources.

PhilJS Protection: - No built-in HTTP client (reduces attack surface)

Your Responsibility: - Validate and sanitize URLs before making requests - Implement allowlists for external APIs - Block private IP ranges - Use network-level protections

Example:

```
function isPrivateIP(hostname: string): boolean {
  const privateRanges = [
    '/^10\./',
    '/^172\.(1[6-9]|2\d|3[01])\./',
    '/^192\.168\./',
    '/^127\./',
    /^localhost$/,
  ];
  return privateRanges.some(range => range.test(hostname));
}

async function fetchExternal(url: string) {
  const parsed = new URL(url);

  if (isPrivateIP(parsed.hostname)) {
    throw new Error('Private IP addresses not allowed');
  }

  return fetch(url);
}
```

Security Checklist

Use this checklist when deploying PhilJS applications:

Development Phase

- Enable strict TypeScript checks
- Configure CSP headers for your application
- Implement CSRF protection for forms and APIs
- Use HTTPS for all environments (including development)
- Sanitize all user input
- Validate file uploads (type, size, content)
- Implement rate limiting for APIs
- Use secure session management
- Configure secure cookie settings
- Review third-party dependencies regularly

Pre-Production

- Run `npm audit` and fix vulnerabilities
- Review CSP violations in report-only mode
- Test authentication and authorization flows
- Perform security code review
- Test XSS protection with fuzzing
- Verify CSRF protection works
- Check for exposed secrets in code/config
- Test error handling (no sensitive info in errors)
- Configure security headers (CSP, HSTS, X-Frame-Options)
- Enable HTTPS with strong TLS configuration

Production

- Monitor CSP violation reports
- Set up security logging and alerting
- Implement DDoS protection
- Configure firewall rules
- Use environment variables for secrets
- Enable automatic security updates
- Implement backup and disaster recovery
- Set up security monitoring (SIEM)
- Configure rate limiting and throttling
- Regular security audits and penetration testing

Post-Deployment

- Monitor security logs regularly
- Keep dependencies updated
- Review and rotate secrets/keys periodically
- Conduct regular security training
- Maintain incident response plan
- Track and patch CVEs promptly

See [Security Checklist](#) for a comprehensive pre-deployment checklist.

Security Best Practices

1. Validate Input, Escape Output

```
import { escapeHtml, sanitizeHtml } from '@philjs/core';

// Always validate input
function validateEmail(email: string): boolean {
  return /^[^@\s]+@[^\s@]+\.[^\s@]+$/ .test(email);
}

// Always escape output
function renderComment(comment: string) {
  return <div>{escapeHtml(comment)}</div>;
}
```

2. Use Security Headers

```
import { buildCSP } from '@philjs/ssr';

const csp = buildCSP({
  directives: {
    'default-src': ["'self'"],
    'script-src': ["'self'"],
    'style-src': ["'self'", "'unsafe-inline'"],
  },
  autoNonce: true,
});

response.headers.set(csp.header, csp.value);
response.headers.set('X-Frame-Options', 'DENY');
response.headers.set('X-Content-Type-Options', 'nosniff');
response.headers.set('Referrer-Policy', 'strict-origin-when-cross-origin');
response.headers.set('Permissions-Policy', 'geolocation=(), microphone=()');
```

3. Implement Rate Limiting

```
import { RateLimiter } from '@philjs/ssr';

const limiter = new RateLimiter({
  windowMs: 15 * 60 * 1000, // 15 minutes
  maxRequests: 100,
});

// In request handler
if (!limiter.check(clientId)) {
  return new Response('Too many requests', { status: 429 });
}
```

4. Secure Authentication

```
// Use secure cookies for sessions
import { createCookie } from '@philjs/ssr';

const sessionCookie = createCookie('session', {
  httpOnly: true,
  secure: true,
  sameSite: 'Strict',
  maxAge: 3600, // 1 hour
  secrets: [process.env.COOKIE_SECRET],
});


```

5. Environment Variables for Secrets

```
// Never hardcode secrets
const API_KEY = process.env.API_KEY;
const DATABASE_URL = process.env.DATABASE_URL;
const SESSION_SECRET = process.env.SESSION_SECRET;

// Validate required secrets exist
if (!API_KEY || !DATABASE_URL || !SESSION_SECRET) {
  throw new Error('Missing required environment variables');
}
```

Dependency Security

PhilJS applications depend on npm packages. Keep dependencies secure:

Regular Audits

```
# Check for vulnerabilities
pnpm audit

# Update dependencies
pnpm update

# Check outdated packages
pnpm outdated
```

Current Vulnerabilities (as of audit)

Based on the latest pnpm audit, the following vulnerabilities exist:

1. **js-yaml** (Moderate) - YAML parsing vulnerability
 - Affected: @changesets/cli, eslint
 - Fix: Update to js-yaml 4.1.1+
2. **undici** (Moderate) - Insufficiently random values in multipart requests
 - Affected: @vercel/node
 - Fix: Update undici to 5.28.5+
3. **esbuild** (Low) - Command injection in CLI
 - Affected: @philjs/cli, @vercel/node
 - Fix: Update esbuild to latest
4. **glob** (High) - Command injection via -c flag
 - Affected: @philjs/migrate
 - Fix: Update glob to 10.5.0+

Mitigation Steps

1. Update dependencies:

```
pnpm update @changesets/cli eslint
pnpm update @vercel/node
pnpm update esbuild
pnpm update glob
```

2. Review and test after updates
3. Consider alternative packages if updates aren't available

Reporting Security Issues

If you discover a security vulnerability in PhilJS:

1. **DO NOT** open a public GitHub issue
2. Email security details to: security@philjs.dev
3. Include:
 - Description of the vulnerability
 - Steps to reproduce
 - Potential impact
 - Suggested fix (if any)

We will: - Acknowledge receipt within 48 hours - Provide a timeline for fixing the issue - Credit you in the security advisory (unless you prefer to remain anonymous) - Notify you

when the fix is released

Additional Resources

- [XSS Prevention Guide](#)
- [Content Security Policy Guide](#)
- [Authentication Patterns](#)
- [API Security Guide](#)
- [Security Checklist](#)
- [OWASP Top 10](#)
- [OWASP Cheat Sheet Series](#)

Version History

- **v0.1.0** (2025): Initial security documentation
- Security features continuously updated with framework releases

Security Policies and Playbooks

Formalize how you respond to incidents and keep the app secure over time.

Policies to define

- **Authentication:** token lifetimes, rotation, MFA requirements.
- **Authorization:** role/permission model, default deny, review cadence.
- **Data handling:** PII classification, retention, deletion/DSAR processes.
- **Secrets management:** storage, rotation, access control.
- **Dependency policy:** allowed license types, update cadence, vulnerability thresholds.

Incident response

- Define severity levels and response times.
- On-call rotation and escalation paths.
- Runbooks for XSS, CSRF, auth leaks, data exfiltration, and supply-chain issues.
- Post-incident review with action items.

Regular tasks

- Monthly dependency updates and vulnerability scans.
- Quarterly CSP/header audits.
- Pen test or internal red-team exercises.
- Backup/restore drills for data and caches.

Edge/serverless considerations

- Limit function permissions; least privilege IAM.
- Validate env presence; fail fast on misconfig.
- Monitor cold start anomalies and unexpected egress.

Testing and automation

- Add security checks to CI (lint for secrets, dep scans).
- Fuzz inputs for loaders/actions.
- Add canary tests for CSP/header correctness.

Communication

- Security contact in `SECURITY.md`.
- Changelog notes for security fixes.
- Clear user-facing messaging when incidents occur.

Checklist

- Policies documented and versioned.
- Incident runbooks defined and tested.
- Regular security tasks scheduled.
- CI security gates in place.
- Edge/serverless permissions minimal and reviewed.

API Security Guide

This guide covers security best practices for building and consuming APIs in PhilJS applications, including CSRF protection, rate limiting, input validation, and more.

Table of Contents

1. API Security Overview
2. CSRF Protection
3. Rate Limiting
4. Input Validation
5. Authentication
6. Authorization
7. Data Protection
8. Error Handling

API Security Overview

Secure APIs are critical for protecting user data and preventing attacks. Key security concerns:

- **Authentication:** Verify who is making the request
- **Authorization:** Verify what they're allowed to do
- **Input Validation:** Prevent injection attacks
- **Rate Limiting:** Prevent abuse and DoS
- **CSRF Protection:** Prevent unauthorized actions
- **Data Protection:** Encrypt sensitive data
- **Error Handling:** Don't leak sensitive information

CSRF Protection

Cross-Site Request Forgery (CSRF) attacks trick users into performing unwanted actions. PhilJS provides built-in CSRF protection.

Implementation

```
import { csrfProtection, generateCSRFToken, extractCSRFToken } from '@philjs/ssr';

// Create CSRF middleware
const csrf = csrfProtection({
  getSessionId: (request) => {
    // Extract session ID from cookie or header
    const sessionCookie = request.headers.get('cookie');
    // Parse session ID...
    return sessionId || 'default';
  },
});

// Generate token for forms
export async function handleGetForm(request: Request) {
  const token = csrf.generateToken(request);

  return new Response(
    `
      <form method="POST" action="/api/submit">
        <input type="hidden" name="_csrf" value="${token}">
        <input type="text" name="data">
        <button type="submit">Submit</button>
      </form>
    `,
    {
      headers: { 'Content-Type': 'text/html' }
    }
  );
}

// Verify token on POST
export async function handlePost(request: Request) {
  // Verify CSRF token
  if (!csrf.verifyRequest(request)) {
    return new Response('CSRF validation failed', { status: 403 });
  }

  // Process request...
  return Response.json({ success: true });
}
```

Token in Headers

For SPA applications:

```

// Server: Send token to client
export async function handleGetToken(request: Request) {
  const token = csrf.generateToken(request);
  return Response.json({ csrfToken: token });
}

// Client: Include token in requests
async function makeRequest(url: string, data: any) {
  const response = await fetch('/api/csrf-token');
  const { csrfToken } = await response.json();

  return fetch(url, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'X-CSRF-Token': csrfToken,
    },
    body: JSON.stringify(data),
  });
}

```

Double Submit Cookie

Alternative CSRF protection:

```

import { createCookie, generateSecureToken } from '@philjs/ssr';

const csrfCookie = createCookie('csrf-token', {
  httpOnly: false, // Must be readable by JavaScript
  secure: true,
  sameSite: 'Strict',
  maxAge: 3600,
});

export async function handleRequest(request: Request) {
  const token = generateSecureToken();

  // Set cookie and return token
  return Response.json({ csrfToken: token }, {
    headers: {
      'Set-Cookie': csrfCookie.serialize(token),
    },
  });
}

// Verify double submit
export async function verifyDoubleSubmit(request: Request) {
  const cookieToken = csrfCookie.parse(request.headers.get('cookie'))?.value;
  const headerToken = request.headers.get('X-CSRF-Token');

  return cookieToken && headerToken && cookieToken === headerToken;
}

```

Rate Limiting

Protect APIs from abuse with rate limiting:

Basic Rate Limiting

```

import { RateLimiter } from '@philjs/ssr';

const apilimiter = new RateLimiter({
  windowMs: 15 * 60 * 1000, // 15 minutes
  maxRequests: 100, // 100 requests per window
});

export async function handleAPIRequest(request: Request) {
  const clientId = getClientId(request);

  if (!apilimiter.check(clientId)) {
    return new Response('Too many requests', {
      status: 429,
      headers: {
        'Retry-After': '900', // 15 minutes in seconds
      },
    });
  }

  // Process request...
}

function getClientId(request: Request): string {
  // Use IP address
  const ip = request.headers.get('x-forwarded-for') ||
    request.headers.get('x-real-ip') ||
    'unknown';

  // Or use authenticated user ID
  const user = getCurrentUser(request);
  return user ? `user:${user.id}` : `ip:${ip}`;
}

```

Advanced Rate Limiting

Different limits for different endpoints:

```

const limiters = {
  login: new RateLimiter({
    windowMs: 15 * 60 * 1000,
    maxRequests: 5, // Strict limit for login
  }),
  api: new RateLimiter({
    windowMs: 15 * 60 * 1000,
    maxRequests: 100, // Normal API limit
  }),
  upload: new RateLimiter({
    windowMs: 60 * 60 * 1000,
    maxRequests: 10, // Very strict for uploads
  }),
};

export function rateLimitMiddleware(
  endpoint: 'login' | 'api' | 'upload'
) {
  return (request: Request) => {
    const clientId = getClientId(request);
    const limiter = limiters[endpoint];

    if (!limiter.check(clientId)) {
      return new Response('Too many requests', { status: 429 });
    }

    return null; // Continue to handler
  };
}

```

Token Bucket Algorithm

More sophisticated rate limiting:

```

class TokenBucket {
  private tokens: Map<string, { count: number; lastRefill: number }> = new Map();

  constructor(
    private capacity: number,
    private refillRate: number, // tokens per second
  ) {}

  consume(key: string, tokens: number = 1): boolean {
    const now = Date.now();
    let bucket = this.tokens.get(key);

    if (!bucket) {
      bucket = { count: this.capacity, lastRefill: now };
      this.tokens.set(key, bucket);
    }

    // Refill tokens based on time passed
    const timePassed = (now - bucket.lastRefill) / 1000;
    const tokensToAdd = Math.floor(timePassed * this.refillRate);

    if (tokensToAdd > 0) {
      bucket.count = Math.min(this.capacity, bucket.count + tokensToAdd);
      bucket.lastRefill = now;
    }

    // Check if we have enough tokens
    if (bucket.count >= tokens) {
      bucket.count -= tokens;
      return true;
    }
  }

  return false;
}

const bucket = new TokenBucket(100, 10); // 100 capacity, 10 tokens/sec

export function handleRequest(request: Request) {
  const clientId = getClientId(request);

  if (!bucket.consume(clientId)) {
    return new Response('Too many requests', { status: 429 });
  }

  // Process request...
}

```

Input Validation

Always validate and sanitize input:

Schema Validation

```

import { z } from 'zod';

// Define schema
const UserSchema = z.object({
  email: z.string().email(),
  name: z.string().min(2).max(100),
  age: z.number().int().min(18).max(120).optional(),
});

export async function handleCreateUser(request: Request) {
  let data;

  try {
    const json = await request.json();
    data = UserSchema.parse(json);
  } catch (error) {
    return Response.json({
      error: 'Validation failed',
      details: error.errors,
    }, { status: 400 });
  }

  // Data is validated and typed
  const user = await createUser(data);
  return Response.json(user);
}

```

Manual Validation

```

import { isValidEmail, escapeHtml, sanitizeUrl } from '@philjs/core';

export function validateInput(data: any): {
  valid: boolean;
  errors: string[];
  sanitized?: any;
} {
  const errors: string[] = [];

  // Validate email
  if (!data.email || !isValidEmail(data.email)) {
    errors.push('Invalid email address');
  }

  // Validate name
  if (!data.name || data.name.length < 2) {
    errors.push('Name must be at least 2 characters');
  }

  // Validate URL
  if (data.website) {
    const validUrl = sanitizeUrl(data.website, ['example.com']);
    if (!validUrl) {
      errors.push('Invalid website URL');
    }
  }

  if (errors.length > 0) {
    return { valid: false, errors };
  }

  return {
    valid: true,
    errors: [],
    sanitized: {
      email: data.email.toLowerCase(),
      name: escapeHtml(data.name),
      website: data.website ? sanitizeUrl(data.website) : null,
    },
  };
}

```

SQL Injection Prevention

Use parameterized queries:

```

// BAD - SQL Injection vulnerable
const userId = request.params.id;
const query = `SELECT * FROM users WHERE id = ${userId}`;
db.query(query);

// GOOD - Parameterized query
const userId = request.params.id;
const query = 'SELECT * FROM users WHERE id = ?';
db.query(query, [userId]);

// BETTER - ORM
const user = await db.users.findOne({ where: { id: userId } });

```

NoSQL Injection Prevention

```

// BAD - NoSQL injection vulnerable
const username = request.body.username;
db.collection('users').findOne({ username });

// GOOD - Validate input type
import { z } from 'zod';

const LoginSchema = z.object({
  username: z.string(),
  password: z.string(),
});

const { username, password } = LoginSchema.parse(request.body);
db.collection('users').findOne({ username });

```

Authentication

Secure API authentication patterns:

Bearer Token

```

export async function requireAuth(request: Request) {
  const authHeader = request.headers.get('Authorization');

  if (!authHeader?.startsWith('Bearer ')) {
    return new Response('Missing authentication', {
      status: 401,
      headers: {
        'WWW-Authenticate': 'Bearer realm="API"',
      },
    });
  }

  const token = authHeader.substring(7);
  const user = await verifyToken(token);

  if (!user) {
    return new Response('Invalid token', { status: 401 });
  }

  return user;
}

// Use in handler
export async function handleProtectedAPI(request: Request) {
  const user = await requireAuth(request);
  if (user instanceof Response) {
    return user; // Return error response
  }

  // User is authenticated
  return Response.json({ data: 'protected data' });
}

```

API Keys

```

const API_KEYS = new Map<string, {
  userId: string;
  permissions: string[];
  rateLimit: number;
}>();

export async function validateAPIKey(request: Request) {
  const apiKey = request.headers.get('X-API-Key');

  if (!apiKey) {
    return new Response('API key required', { status: 401 });
  }

  const keyData = API_KEYS.get(apiKey);
  if (!keyData) {
    return new Response('Invalid API key', { status: 401 });
  }

  return keyData;
}

// Generate API key
import { generateSecureToken } from '@philjs/core';

export async function createAPIKey(userId: string, permissions: string[]) {
  const apiKey = generateSecureToken(32);

  API_KEYS.set(apiKey, {
    userId,
    permissions,
    rateLimit: 100,
  });

  return apiKey;
}

```

Authorization

Control what authenticated users can access:

Role-Based Access Control (RBAC)

```

enum Permission {
  READ = 'read',
  WRITE = 'write',
  DELETE = 'delete',
  ADMIN = 'admin',
}

const rolePermissions: Record<string, Permission[]> = {
  user: [Permission.READ],
  editor: [Permission.READ, Permission.WRITE],
  admin: [Permission.READ, Permission.WRITE, Permission.DELETE, Permission.ADMIN],
};

export function hasPermission(
  UserRole: string,
  required: Permission
): boolean {
  const permissions = rolePermissions[UserRole] || [];
  return permissions.includes(required);
}

export function requirePermission(permission: Permission) {
  return async (request: Request) => {
    const user = await getCurrentUser(request);

    if (!user) {
      return new Response('Unauthorized', { status: 401 });
    }

    if (!hasPermission(user.role, permission)) {
      return new Response('Forbidden', { status: 403 });
    }

    return null; // Continue
  };
}

// Use in handler
export async function handleDeleteUser(request: Request) {
  const authCheck = await requirePermission(Permission.DELETE)(request);
  if (authCheck) return authCheck;

  // User has delete permission
  // ... delete logic
}

```

Attribute-Based Access Control (ABAC)

```

type AccessPolicy = {
  resource: string;
  action: string;
  condition: (context: AccessContext) => boolean;
};

type AccessContext = {
  user: User;
  resource: any;
  environment: {
    time: Date;
    ip: string;
  };
};

const policies: AccessPolicy[] = [
  {
    resource: 'document',
    action: 'read',
    condition: (ctx) => {
      // Owner can always read
      if (ctx.resource.ownerId === ctx.user.id) return true;

      // Public documents can be read by anyone
      if (ctx.resource.visibility === 'public') return true;

      // Collaborators can read
      return ctx.resource.collaborators.includes(ctx.user.id);
    },
  },
  {
    resource: 'document',
    action: 'write',
    condition: (ctx) => {
      // Only owner and collaborators can write
      return ctx.resource.ownerId === ctx.user.id ||
        ctx.resource.collaborators.includes(ctx.user.id);
    },
  },
];

export function checkAccess(
  resource: string,
  action: string,
  context: AccessContext
): boolean {
  const policy = policies.find(
    p => p.resource === resource && p.action === action
  );

  if (!policy) {
    return false; // Default deny
  }

  return policy.condition(context);
}

```

Data Protection

Encrypt Sensitive Data

```

import { createCipheriv, createDecipheriv, randomBytes } from 'crypto';

const ENCRYPTION_KEY = Buffer.from(process.env.ENCRYPTION_KEY!, 'hex');

export function encrypt(text: string): string {
  const iv = randomBytes(16);
  const cipher = createCipheriv('aes-256-cbc', ENCRYPTION_KEY, iv);

  let encrypted = cipher.update(text, 'utf8', 'hex');
  encrypted += cipher.final('hex');

  return iv.toString('hex') + ':' + encrypted;
}

export function decrypt(text: string): string {
  const parts = text.split(':');
  const iv = Buffer.from(parts[0], 'hex');
  const encrypted = parts[1];

  const decipher = createDecipheriv('aes-256-cbc', ENCRYPTION_KEY, iv);

  let decrypted = decipher.update(encrypted, 'hex', 'utf8');
  decrypted += decipher.final('utf8');

  return decrypted;
}

// Use in API
export async function handleSaveSecret(request: Request) {
  const { secret } = await request.json();

  const encrypted = encrypt(secret);

  await db.secrets.create({
    userId: user.id,
    value: encrypted,
  });

  return Response.json({ success: true });
}

```

Hash Sensitive Data

```

import { createHash } from 'crypto';

export function hashData(data: string): string {
  return createHash('sha256')
    .update(data)
    .digest('hex');
}

// For credit cards, SSN, etc.
export async function storePaymentMethod(cardNumber: string) {
  const hash = hashData(cardNumber);
  const last4 = cardNumber.slice(-4);

  await db.paymentMethods.create({
    hash,
    last4,
    // Don't store full card number
  });
}

```

Error Handling

Don't leak sensitive information in errors:

Secure Error Responses

```

export function handleError(error: unknown): Response {
  console.error('API Error:', error);

  // Don't expose internal errors to clients
  if (process.env.NODE_ENV === 'production') {
    return Response.json({
      error: 'Internal server error',
    }, { status: 500 });
  }

  // In development, provide more details
  return Response.json({
    error: error instanceof Error ? error.message : 'Unknown error',
    stack: error instanceof Error ? error.stack : undefined,
  }, { status: 500 });
}

// Use in handler
export async function handleAPIRequest(request: Request) {
  try {
    // ... API logic
    return Response.json({ success: true });
  } catch (error) {
    return handleError(error);
  }
}

```

Error Logging

```

type SecurityEvent = {
  type: 'error' | 'warning' | 'info';
  message: string;
  userId?: string;
  ip: string;
  endpoint: string;
  timestamp: Date;
};

export async function logSecurityEvent(event: SecurityEvent) {
  // Log to your monitoring service
  console.log('[SECURITY]', event);

  // Store in database for analysis
  await db.securityLogs.create(event);

  // Alert on critical events
  if (event.type === 'error') {
    await sendAlert(event);
  }
}

// Use in API
export async function handleRequest(request: Request) {
  try {
    // ... handle request
  } catch (error) {
    await logSecurityEvent({
      type: 'error',
      message: error.message,
      ip: request.headers.get('x-forwarded-for') || 'unknown',
      endpoint: new URL(request.url).pathname,
      timestamp: new Date(),
    });
    throw error;
  }
}

```

Security Headers

Always set security headers:

```

export function addSecurityHeaders(response: Response): Response {
  const headers = new Headers(response.headers);

  headers.set('X-Content-Type-Options', 'nosniff');
  headers.set('X-Frame-Options', 'DENY');
  headers.set('X-XSS-Protection', '1; mode=block');
  headers.set('Referrer-Policy', 'strict-origin-when-cross-origin');
  headers.set('Permissions-Policy', 'geolocation(), microphone(), camera()');

  // CSP
  const csp = buildCSP({
    directives: {
      'default-src': ["'self'"],
      'script-src': ["'self'"],
    },
  });
  headers.set(csp.header, csp.value);

  return new Response(response.body, {
    status: response.status,
    statusText: response.statusText,
    headers,
  });
}

// Apply to all responses
export async function handleRequest(request: Request) {
  const response = await routeHandler(request);
  return addSecurityHeaders(response);
}

```

API Security Checklist

- Implement CSRF protection for state-changing operations
- Use rate limiting on all endpoints
- Validate all input with schemas
- Use parameterized queries to prevent injection
- Require authentication for protected endpoints
- Implement proper authorization checks
- Encrypt sensitive data at rest
- Use HTTPS for all API communication
- Set security headers on all responses
- Don't expose sensitive data in error messages
- Log security events
- Implement request timeout
- Validate content-type headers
- Use API versioning
- Implement request signing for sensitive operations
- Use CORS properly
- Implement audit logging
- Regularly review API permissions
- Monitor for unusual activity
- Keep dependencies updated

Resources

- [OWASP API Security Top 10](#)
- [OWASP REST Security Cheat Sheet](#)
- [Authentication Patterns](#)
- [CSRF Protection](#)
- [Security Overview](#)

Authentication Patterns

This guide covers secure authentication patterns for PhilJS applications, including session management, JWT tokens, OAuth integration, and best practices.

Table of Contents

1. [Authentication Overview](#)
2. [Session-Based Authentication](#)
3. [Token-Based Authentication](#)
4. [OAuth and Social Login](#)
5. [Password Security](#)

6. [Multi-Factor Authentication](#)
7. [Best Practices](#)

Authentication Overview

PhilJS doesn't include built-in authentication, giving you flexibility to choose the right approach for your application. This guide shows secure patterns for common authentication methods.

Choosing an Authentication Strategy

Strategy	Best For	Pros	Cons
Session-based	Traditional web apps	Simple, stateful	Scalability challenges
JWT	APIs, microservices	Stateless, scalable	Token size, revocation
OAuth	Third-party login	No password management	Dependency on provider
Hybrid	Complex applications	Flexibility	More complexity

Session-Based Authentication

Implementation

```

import { createCookie } from '@philjs/ssr';
import { generateSecureToken } from '@philjs/core';

// Session storage (use Redis in production)
const sessions = new Map<string, {
  userId: string;
  expires: number;
  data: Record<string, any>;
}>();

// Create secure session cookie
const sessionCookie = createCookie('session', {
  httpOnly: true,
  secure: true,
  sameSite: 'Lax',
  maxAge: 3600, // 1 hour
  secrets: [process.env.COOKIE_SECRET!],
});

// Login handler
export async function handleLogin(request: Request) {
  const formData = await request.formData();
  const email = formData.get('email') as string;
  const password = formData.get('password') as string;

  // Validate credentials
  const user = await validateCredentials(email, password);
  if (!user) {
    return new Response('Invalid credentials', { status: 401 });
  }

  // Create session
  const sessionId = generateSecureToken();
  sessions.set(sessionId, {
    userId: user.id,
    expires: Date.now() + 3600000, // 1 hour
    data: {},
  });

  // Set cookie
  const cookie = sessionCookie.serialize(sessionId);

  return new Response('Login successful', {
    status: 302,
    headers: {
      'Location': '/dashboard',
      'Set-Cookie': cookie,
    },
  });
}

// Logout handler
export async function handleLogout(request: Request) {
  const cookieHeader = request.headers.get('cookie');
  const parsed = sessionCookie.parse(cookieHeader);

  if (parsed) {
    sessions.delete(parsed.value);
  }

  return new Response('Logged out', {
    status: 302,
    headers: {
      'Location': '/',
      'Set-Cookie': sessionCookie.destroy(),
    },
  });
}

// Middleware to get current user
export function getCurrentUser(request: Request) {
  const cookieHeader = request.headers.get('cookie');
  const parsed = sessionCookie.parse(cookieHeader);

  if (!parsed) {
    return null;
  }

  const session = sessions.get(parsed.value);
  if (!session || session.expires < Date.now()) {
    sessions.delete(parsed.value);
    return null;
  }

  return { userId: session.userId };
}

```

Session Storage

Development (in-memory):

```
const sessions = new Map();
```

Production (Redis):

```
import { createClient } from 'redis';

const redis = createClient({
  url: process.env.REDIS_URL,
});

await redis.connect();

async function createSession(userId: string) {
  const sessionId = generateSecureToken();
  await redis.setEx(
    `session:${sessionId}`,
    3600, // TTL in seconds
    JSON.stringify({ userId })
  );
  return sessionId;
}

async function getSession(sessionId: string) {
  const data = await redis.get(`session:${sessionId}`);
  return data ? JSON.parse(data) : null;
}

async function deleteSession(sessionId: string) {
  await redis.del(`session:${sessionId}`);
}
```

Token-Based Authentication

JWT Implementation

```

import { SignJWT, jwtVerify } from 'jose';

const JWT_SECRET = new TextEncoder().encode(
  process.env.JWT_SECRET || 'your-secret-key'
);

// Generate JWT
export async function generateToken(userId: string) {
  const token = await new SignJWT({ userId })
    .setProtectedHeader({ alg: 'HS256' })
    .setIssuedAt()
    .setExpirationTime('2h')
    .sign(JWT_SECRET);

  return token;
}

// Verify JWT
export async function verifyToken(token: string) {
  try {
    const { payload } = await jwtVerify(token, JWT_SECRET);
    return payload as { userId: string };
  } catch {
    return null;
  }
}

// Login handler with JWT
export async function handleLogin(request: Request) {
  const { email, password } = await request.json();

  const user = await validateCredentials(email, password);
  if (!user) {
    return new Response('Invalid credentials', { status: 401 });
  }

  const token = await generateToken(user.id);

  return Response.json({
    token,
    user: {
      id: user.id,
      email: user.email,
      name: user.name,
    },
  });
}

// Protected route middleware
export async function requireAuth(request: Request) {
  const authHeader = request.headers.get('Authorization');
  if (!authHeader?.startsWith('Bearer ')) {
    return new Response('Unauthorized', { status: 401 });
  }

  const token = authHeader.substring(7);
  const payload = await verifyToken(token);

  if (!payload) {
    return new Response('Invalid token', { status: 401 });
  }

  return payload;
}

```

[Refresh Tokens](#)

```

import { generateSecureToken } from '@philjs/core';

const refreshTokens = new Map<string, {
  userId: string;
  expires: number;
}>();

// Generate token pair
export async function generateTokenPair(userId: string) {
  const accessToken = await generateToken(userId);
  const refreshToken = generateSecureToken();

  refreshTokens.set(refreshToken, {
    userId,
    expires: Date.now() + 7 * 24 * 60 * 60 * 1000, // 7 days
  });

  return { accessToken, refreshToken };
}

// Refresh access token
export async function handleRefresh(request: Request) {
  const { refreshToken } = await request.json();

  const tokenData = refreshTokens.get(refreshToken);
  if (!tokenData || tokenData.expires < Date.now()) {
    refreshTokens.delete(refreshToken);
    return new Response('Invalid refresh token', { status: 401 });
  }

  // Generate new access token
  const accessToken = await generateToken(tokenData.userId);

  return Response.json({ accessToken });
}

```

OAuth and Social Login

OAuth 2.0 Flow

```

import { generateSecureToken } from '@philjs/core';

const OAUTH_CONFIG = {
  google: {
    clientId: process.env.GOOGLE_CLIENT_ID!,
    clientSecret: process.env.GOOGLE_CLIENT_SECRET!,
    authUrl: 'https://accounts.google.com/o/oauth2/v2/auth',
    tokenUrl: 'https://oauth2.googleapis.com/token',
    userInfoUrl: 'https://www.googleapis.com/oauth2/v2/userinfo',
  },
  github: {
    clientId: process.env.GITHUB_CLIENT_ID!,
    clientSecret: process.env.GITHUB_CLIENT_SECRET!,
    authUrl: 'https://github.com/login/oauth/authorize',
    tokenUrl: 'https://github.com/login/oauth/access_token',
    userInfoUrl: 'https://api.github.com/user',
  },
};

// Store state tokens
const oauthStates = new Map<string, {
  provider: string;
  expires: number;
}>();

// Initiate OAuth flow
export function handleOAuthLogin(provider: 'google' | 'github') {
  const config = OAUTH_CONFIG[provider];
  const state = generateSecureToken();

  oauthStates.set(state, {
    provider,
    expires: Date.now() + 600000, // 10 minutes
  });

  const params = new URLSearchParams({
    client_id: config.clientId,
    redirect_uri: `${process.env.APP_URL}/auth/callback/${provider}`,
    response_type: 'code',
    scope: provider === 'google' ? 'email profile' : 'user:email',
    state,
  });

  const authUrl = `${config.authUrl}?${params}`;

  return Response.redirect(authUrl);
}

```

```

// OAuth callback handler
export async function handleOAuthCallback(
  provider: 'google' | 'github',
  request: Request
) {
  const url = new URL(request.url);
  const code = url.searchParams.get('code');
  const state = url.searchParams.get('state');

  // Validate state
  const stateData = oauthStates.get(state);
  if (!stateData || stateData.expires < Date.now()) {
    return new Response('Invalid state', { status: 400 });
  }
  oauthStates.delete(state);

  const config = OAUTH_CONFIG[provider];

  // Exchange code for token
  const tokenResponse = await fetch(config.tokenUrl, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded',
      'Accept': 'application/json',
    },
    body: new URLSearchParams({
      client_id: config.clientId,
      client_secret: config.clientSecret,
      code: code!,
      redirect_uri: `${process.env.APP_URL}/auth/callback/${provider}`,
      grant_type: 'authorization_code',
    }),
  });

  const { access_token } = await tokenResponse.json();

  // Get user info
  const userResponse = await fetch(config.userInfoUrl, {
    headers: {
      'Authorization': `Bearer ${access_token}`,
    },
  });

  const userData = await userResponse.json();

  // Create or update user
  const user = await findOrCreateUser({
    email: userData.email,
    name: userData.name || userData.login,
    provider,
    providerId: userData.id,
  });

  // Create session
  const sessionId = generateSecureToken();
  sessions.set(sessionId, {
    userId: user.id,
    expires: Date.now() + 3600000,
    data: {},
  });

  const cookie = sessionCookie.serialize(sessionId);

  return new Response('Login successful', {
    status: 302,
    headers: {
      'Location': '/dashboard',
      'Set-Cookie': cookie,
    },
  });
}

```

Password Security

Password Hashing

```

import bcrypt from 'bcrypt';

const SALT_ROUNDS = 12;

// Hash password
export async function hashPassword(password: string): Promise<string> {
  return bcrypt.hash(password, SALT_ROUNDS);
}

// Verify password
export async function verifyPassword(
  password: string,
  hash: string
): Promise<boolean> {
  return bcrypt.compare(password, hash);
}

// Validate password strength
export function validatePasswordStrength(password: string): {
  valid: boolean;
  errors: string[];
} {
  const errors: string[] = [];

  if (password.length < 12) {
    errors.push('Password must be at least 12 characters');
  }

  if (!/[a-z]/.test(password)) {
    errors.push('Password must contain lowercase letters');
  }

  if (!/[A-Z]/.test(password)) {
    errors.push('Password must contain uppercase letters');
  }

  if (!/[0-9]/.test(password)) {
    errors.push('Password must contain numbers');
  }

  if (!/[a-zA-Z0-9]/.test(password)) {
    errors.push('Password must contain special characters');
  }

  // Check against common passwords
  const commonPasswords = ['password', '12345678', 'qwerty'];
  if (commonPasswords.includes(password.toLowerCase())) {
    errors.push('Password is too common');
  }
}

return {
  valid: errors.length === 0,
  errors,
};
}

```

User Registration

```
export async function handleRegister(request: Request) {
  const formData = await request.formData();
  const email = formData.get('email') as string;
  const password = formData.get('password') as string;

  // Validate email
  if (!isValidEmail(email)) {
    return new Response('Invalid email', { status: 400 });
  }

  // Validate password strength
  const passwordCheck = validatePasswordStrength(password);
  if (!passwordCheck.valid) {
    return Response.json({
      errors: passwordCheck.errors,
    }, { status: 400 });
  }

  // Check if user exists
  const existingUser = await findUserByEmail(email);
  if (existingUser) {
    return new Response('Email already registered', { status: 409 });
  }

  // Hash password
  const hashedPassword = await hashPassword(password);

  // Create user
  const user = await createUser({
    email,
    password: hashedPassword,
  });

  // Send verification email
  await sendVerificationEmail(user.email, user.verificationToken);

  return Response.json({
    message: 'Registration successful. Please check your email.',
  });
}
```

Password Reset

```

export async function handlePasswordResetRequest(request: Request) {
  const { email } = await request.json();

  const user = await findUserByEmail(email);
  if (!user) {
    // Don't reveal if email exists
    return Response.json({
      message: 'If the email exists, a reset link has been sent.',
    });
  }

  // Generate reset token
  const resetToken = generateSecureToken();
  const resetExpires = Date.now() + 3600000; // 1 hour

  await updateUser(user.id, {
    resetToken,
    resetExpires,
  });

  // Send reset email
  await sendPasswordResetEmail(user.email, resetToken);

  return Response.json({
    message: 'If the email exists, a reset link has been sent.',
  });
}

export async function handlePasswordReset(request: Request) {
  const { token, newPassword } = await request.json();

  const user = await findUserByResetToken(token);
  if (!user || user.resetExpires! < Date.now()) {
    return new Response('Invalid or expired token', { status: 400 });
  }

  // Validate new password
  const passwordCheck = validatePasswordStrength(newPassword);
  if (!passwordCheck.valid) {
    return Response.json({
      errors: passwordCheck.errors,
    }, { status: 400 });
  }

  // Hash and update password
  const hashedPassword = await hashPassword(newPassword);
  await updateUser(user.id, {
    password: hashedPassword,
    resetToken: null,
    resetExpires: null,
  });

  return Response.json({
    message: 'Password reset successful',
  });
}

```

Multi-Factor Authentication

TOTP (Time-Based One-Time Password)

```

import * as OTPAuth from 'otpauth';
import { generateSecureToken } from '@philjs/core';

// Generate TOTP secret
export function generateTOTPSecret(email: string) {
  const secret = new OTPAuth.Secret({ size: 20 });
  const totp = new OTPAuth.TOTP({
    issuer: 'MyApp',
    label: email,
    algorithm: 'SHA1',
    digits: 6,
    period: 30,
    secret,
  });

  return {
    secret: secret.base32,
    uri: totp.toString(),
  };
}

// Verify TOTP code
export function verifyTOTP(secret: string, token: string): boolean {
  const totp = new OTPAuth.TOTP({
    secret: OTPAuth.Secret.fromBase32(secret),
    digits: 6,
  });
}

```

```

        period: 30,
    });

    const delta = totp.validate({ token, window: 1 });
    return delta !== null;
}

// Enable 2FA
export async function handleEnable2FA(request: Request) {
    const user = await getCurrentUser(request);
    if (!user) {
        return new Response('Unauthorized', { status: 401 });
    }

    const { secret, uri } = generateTOTPSecret(user.email);

    // Save secret (encrypted)
    await updateUser(user.id, {
        totpSecret: secret,
        twoFactorEnabled: false, // Not enabled until verified
    });

    return Response.json({
        secret,
        qrCodeUri: uri,
    });
}

// Verify and activate 2FA
export async function handleVerify2FA(request: Request) {
    const user = await getCurrentUser(request);
    if (!user) {
        return new Response('Unauthorized', { status: 401 });
    }

    const { token } = await request.json();

    const userData = await findUserById(user.userId);
    if (!userData.totpSecret) {
        return new Response('2FA not set up', { status: 400 });
    }

    if (!verifyTOTP(userData.totpSecret, token)) {
        return new Response('Invalid code', { status: 400 });
    }

    // Generate backup codes
    const backupCodes = Array.from({ length: 10 }, () =>
        generateSecureToken(8).toUpperCase()
    );

    await updateUser(user.userId, {
        twoFactorEnabled: true,
        backupCodes: await Promise.all(
            backupCodes.map(code => hashPassword(code))
        ),
    });

    return Response.json({
        message: '2FA enabled successfully',
        backupCodes,
    });
}

// Login with 2FA
export async function handleLoginWith2FA(request: Request) {
    const { email, password, totpToken } = await request.json();

    const user = await validateCredentials(email, password);
    if (!user) {
        return new Response('Invalid credentials', { status: 401 });
    }

    if (user.twoFactorEnabled) {
        if (!totpToken) {
            return Response.json({
                requiresTOTP: true,
            }, { status: 403 });
        }

        const isValid = verifyTOTP(user.totpSecret!, totpToken);
        if (!isValid) {
            // Try backup codes
            const backupCodeValid = await verifyBackupCode(user, totpToken);
            if (!backupCodeValid) {
                return new Response('Invalid 2FA code', { status: 401 });
            }
        }
    }
}

```

```

    }

    // Create session
    const sessionId = generateSecureToken();
    sessions.set(sessionId, {
      userId: user.id,
      expires: Date.now() + 3600000,
      data: {}
    });

    const cookie = sessionCookie.serialize(sessionId);

    return Response.json({
      message: 'Login successful',
    }, {
      headers: {
        'Set-Cookie': cookie,
      },
    });
}

```

Best Practices

1. Use HTTPS Everywhere

```

// Redirect HTTP to HTTPS
export function httpsRedirect(request: Request) {
  const url = new URL(request.url);
  if (url.protocol === 'http:' && process.env.NODE_ENV === 'production') {
    url.protocol = 'https:';
    return Response.redirect(url.toString(), 301);
  }
}

```

2. Implement Rate Limiting

```

import { RateLimiter } from '@philjs/ssr';

const loginLimiter = new RateLimiter({
  windowMs: 15 * 60 * 1000, // 15 minutes
  maxRequests: 5, // 5 attempts
});

export async function handleLogin(request: Request) {
  const clientIp = request.headers.get('x-forwarded-for') || 'unknown';

  if (!loginLimiter.check(clientIp)) {
    return new Response('Too many login attempts', { status: 429 });
  }

  // ... rest of Login logic
}

```

3. Secure Cookie Settings

```

const sessionCookie = createCookie('session', {
  httpOnly: true,           // Prevent JavaScript access
  secure: true,             // HTTPS only
  sameSite: 'Strict',       // CSRF protection
  maxAge: 3600,             // 1 hour
  secrets: [process.env.COOKIE_SECRET!],
  path: '/',
});

```

4. Validate Sessions Server-Side

```

export async function requireAuth(request: Request) {
  const user = await getCurrentUser(request);

  if (!user) {
    return new Response('Unauthorized', {
      status: 401,
      headers: {
        'WWW-Authenticate': 'Bearer',
      },
    });
  }

  return user;
}

```

5. Log Security Events

```

export async function logSecurityEvent(event: {
  type: 'login' | 'logout' | 'failed_login' | 'password_reset';
  userId?: string;
  ip: string;
  userAgent: string;
  success: boolean;
}) {
  // Log to your security monitoring system
  console.log('[SECURITY]', event);

  // Alert on suspicious activity
  if (event.type === 'failed_login') {
    await checkForBruteForce(event.ip);
  }
}

```

Security Checklist

- Use HTTPS in production
- Hash passwords with bcrypt (12+ rounds)
- Validate password strength
- Implement rate limiting on auth endpoints
- Use secure, httpOnly cookies
- Implement CSRF protection
- Enable 2FA for sensitive accounts
- Log authentication events
- Implement account lockout after failed attempts
- Use secure session storage (Redis in production)
- Rotate secrets regularly
- Validate JWT tokens properly
- Implement token refresh mechanism
- Use state parameter in OAuth flows
- Sanitize and validate all user input
- Implement email verification
- Use secure password reset flow
- Never log passwords or tokens
- Implement account recovery mechanisms
- Test authentication flows regularly

Resources

- [OWASP Authentication Cheat Sheet](#)
- [NIST Digital Identity Guidelines](#)
- [OAuth 2.0 Security Best Practices](#)
- [API Security Guide](#)
- [Security Overview](#)

Content Security Policy Guide

Content Security Policy (CSP) is a powerful security feature that helps prevent XSS attacks by controlling which resources can be loaded and executed in your application.

Table of Contents

1. [Understanding CSP](#)
2. [CSP in PhilJS](#)
3. [Configuration](#)
4. [Nonce-Based Scripts](#)
5. [Common Patterns](#)
6. [Testing and Debugging](#)
7. [Migration Guide](#)

Understanding CSP

CSP allows you to create an allowlist of trusted sources for content. When properly configured, it prevents:

- XSS attacks via inline scripts
- Data exfiltration
- Clickjacking
- Protocol downgrade attacks
- Mixed content

How CSP Works

CSP uses HTTP headers to tell the browser which resources are allowed:

```
Content-Security-Policy: default-src 'self'; script-src 'self' https://cdn.example.com
```

This policy means: - By default, only load resources from the same origin ('self') - Scripts can be loaded from same origin or cdn.example.com - Inline scripts are blocked (unless using nonces or hashes)

CSP in PhilJS

PhilJS provides comprehensive CSP utilities in the `@philjs/ssr` package:

Basic Setup

```
import { buildCSP } from '@philjs/ssr';

const csp = buildCSP({
  directives: {
    'default-src': ["'self'"],
    'script-src': ["'self'"],
    'style-src': ["'self'", "'unsafe-inline'"],
    'img-src': ["'self'", 'https:', 'data:'],
  },
});

response.headers.set(csp.header, csp.value);
```

With Auto-Generated Nonce

```
import { buildCSP } from '@philjs/ssr';

const csp = buildCSP({
  directives: {
    'default-src': ["'self'"],
    'script-src': ["'self'"],
  },
  autoNonce: true, // Automatically generate nonce
});

response.headers.set(csp.header, csp.value);

// Use the nonce in your HTML
const html = `
<!DOCTYPE html>
<html>
  <head>
    <script nonce="${csp.nonce}">
      // This inline script is allowed
      console.log('CSP with nonce!');
    </script>
  </head>
</html>
`;
```

Configuration

Default Directives

PhilJS provides secure defaults:

```
import { DEFAULT_CSP_DIRECTIVES } from '@philjs/ssr';

// Equivalent to:
{
  'default-src': ["'self'"],
  'script-src': ["'self'"],
  'style-src': ["'self'", "'unsafe-inline'"],
  'img-src': ["'self'", 'data:', 'blob:', 'https:'],
  'font-src': ["'self'", 'data:'],
  'connect-src': ["'self'"],
  'media-src': ["'self'"],
  'object-src': ["'none'"],
  'frame-src': ["'none'"],
  'base-uri': ["'self'"],
  'form-action': ["'self'"],
  'frame-ancestors': ["'none'"],
  'upgrade-insecure-requests': true,
}
```

Strict CSP

For maximum security:

```

import { STRICT_CSP_DIRECTIVES } from '@philjs/ssr';

const csp = buildCSP({
  directives: STRICT_CSP_DIRECTIVES,
  autoNonce: true,
});

// Strict policy blocks everything except explicitly allowed

```

Development CSP

More relaxed for local development:

```

import { DEV_CSP_DIRECTIVES } from '@philjs/ssr';

const isDev = process.env.NODE_ENV === 'development';

const csp = buildCSP({
  directives: isDev ? DEV_CSP_DIRECTIVES : DEFAULT_CSP_DIRECTIVES,
  autoNonce: !isDev,
});

```

Custom Directives

Add your own directives:

```

import { buildCSP, mergeCSP, DEFAULT_CSP_DIRECTIVES } from '@philjs/ssr';

const customCSP = mergeCSP(DEFAULT_CSP_DIRECTIVES, {
  'script-src': ["'self'", 'https://analytics.example.com'],
  'connect-src': ["'self'", 'https://api.example.com'],
  'img-src': ["'self'", 'https://cdn.example.com', 'data:'],
});

const csp = buildCSP({ directives: customCSP });

```

Nonce-Based Scripts

Nonces allow specific inline scripts while blocking all others:

Server-Side Implementation

```

import { buildCSP, generateNonce } from '@philjs/ssr';

export async function handleRequest(request: Request) {
  // Generate CSP with nonce
  const csp = buildCSP({
    directives: {
      'default-src': ["'self'"],
      'script-src': ["'self'"],
    },
    autoNonce: true,
  });

  // Render page with nonce
  const html = renderPage({ nonce: csp.nonce });

  return new Response(html, {
    headers: {
      'Content-Type': 'text/html',
      [csp.header]: csp.value,
    },
  });
}

function renderPage({ nonce }: { nonce?: string }) {
  return `
    <!DOCTYPE html>
    <html>
      <head>
        <title>My App</title>
      </head>
      <body>
        <div id="root"></div>

        <!-- Inline script with nonce -->
        <script nonce="${nonce}">
          window.__INITIAL_STATE__ = { user: "John" };
        </script>

        <!-- External script (allowed by 'self') -->
        <script src="/app.js"></script>
      </body>
    </html>
  `;
}

```

SSR with Nonces

```
import { renderToStream, buildCSP } from '@philjs/ssr';

export async function handleSSR(request: Request) {
  const csp = buildCSP({
    directives: {
      'script-src': ["'self'"],
    },
    autoNonce: true,
  });

  const stream = renderToStream(<App />, {
    bootstrapScripts: ['/client.ts'],
  });

  return new Response(stream, {
    headers: {
      'Content-Type': 'text/html',
      [csp.header]: csp.value,
    },
  });
}
```

Client-Side Hydration

Pass the nonce to client-side code:

```
// server.ts
const csp = buildCSP({ autoNonce: true });
const html = `
<script nonce="${csp.nonce}">
  window.__CSP_NONCE__ = "${csp.nonce}";
</script>
<script nonce="${csp.nonce}" src="/client.ts"></script>
`;

// client.ts
const nonce = (window as any).__CSP_NONCE__;

// Use nonce for dynamically created scripts
const script = document.createElement('script');
script.nonce = nonce;
script.textContent = 'console.log("Dynamic script");';
document.head.appendChild(script);
```

Common Patterns

Pattern 1: Static Site

Simple CSP for static sites:

```
const csp = buildCSP({
  directives: {
    'default-src': ["'self'"],
    'script-src': ["'self'"],
    'style-src': ["'self'"],
    'img-src': ["'self'", 'data:'],
    'font-src': ["'self'"],
    'connect-src': ["'none'"],
    'object-src': ["'none'"],
  },
});
```

Pattern 2: SPA with CDN

Allow CDN resources:

```
const csp = buildCSP({
  directives: {
    'default-src': ["'self'"],
    'script-src': [
      "'self'",
      'https://cdn.jsdelivr.net',
      'https://unpkg.com',
    ],
    'style-src': ["'self'", 'https://cdn.jsdelivr.net'],
    'img-src': ["'self'", 'https:', 'data:'],
    'connect-src': ["'self'", 'https://api.example.com'],
  },
});
```

Pattern 3: Analytics Integration

Allow third-party analytics:

```

const csp = buildCSP({
  directives: {
    'default-src': ["'self'"],
    'script-src': [
      "'self'",
      'https://www.google-analytics.com',
      'https://www.googletagmanager.com',
    ],
    'connect-src': [
      "'self'",
      'https://www.google-analytics.com',
      'https://stats.g.doubleclick.net',
    ],
    'img-src': [
      "'self'",
      'https://www.google-analytics.com',
      'data:',
    ],
  },
});

```

Pattern 4: Embedding Content

Allow frames and embeds:

```

const csp = buildCSP({
  directives: {
    'default-src': ["'self'"],
    'frame-src': [
      'https://www.youtube.com',
      'https://player.vimeo.com',
    ],
    'img-src': ["'self'", 'https:', 'data:'],
  },
});

```

Pattern 5: Report-Only Mode

Test CSP without breaking your app:

```

const csp = buildCSP({
  directives: {
    'default-src': ["'self'"],
    'script-src': ["'self'"],
    'report-uri': '/csp-violation-report',
  },
  reportOnly: true, // Won't block, only report
});

// Set up violation reporting endpoint
export async function handleCSPReport(request: Request) {
  const violation = await request.json();
  console.log('CSP Violation:', violation);

  // Log to your monitoring service
  await logToMonitoring({
    type: 'cspViolation',
    details: violation,
  });

  return new Response('OK', { status: 200 });
}

```

Directive Reference

Fetch Directives

Control where resources can be loaded from:

```
{
  'default-src': ["'self'"],           // Default for all fetch directives
  'script-src': ["'self'"],             // JavaScript sources
  'style-src': ["'self'"],              // CSS sources
  'img-src': ["'self'", 'https:'],     // Image sources
  'font-src': ["'self'", 'data:'],      // Font sources
  'connect-src': ["'self'"],            // XHR, WebSocket, EventSource
  'media-src': ["'self'"],              // Audio, video sources
  'object-src': ["'none'"],             // <object>, <embed>, <applet>
  'frame-src': ["'self'"],              // <iframe> sources
  'worker-src': ["'self'"],             // Worker sources
  'manifest-src': ["'self'"],            // Manifest sources
}
```

Document Directives

Control document properties:

```
{
  'base-uri': ["'self'"],           // <base> element URLs
  'form-action': ["'self'"],          // Form submission URLs
  'frame-ancestors': ["'none'"],      // Who can embed this page
}
```

Navigation Directives

```
{
  'navigate-to': ["'self'"],          // Navigation URLs (experimental)
}
```

Reporting Directives

```
{
  'report-uri': '/csp-report',        // Legacy reporting endpoint
  'report-to': 'csp-endpoint',         // Modern reporting endpoint
}
```

Other Directives

```
{
  'upgrade-insecure-requests': true,   // Upgrade HTTP to HTTPS
  'block-all-mixed-content': true,     // Block mixed content
  'require-sri-for': ['script', 'style'], // Require SRI
}
```

Testing and Debugging

Enable Report-Only Mode

Start with report-only to test without breaking:

```
const csp = buildCSP({
  directives: {
    'default-src': ["'self'"],
    'script-src': ["'self'"],
  },
  reportOnly: true,
});
```

Use Browser DevTools

Check CSP violations in the console:

```
Refused to load the script 'https://evil.com/script.js' because it violates
the following Content Security Policy directive: "script-src 'self'".
```

Validate CSP

Use PhilJS validation:

```
import { validateCSP } from '@philjs/ssr';

const warnings = validateCSP({
  'script-src': ["'unsafe-eval'", "'unsafe-inline'"],
});

console.log(warnings);
// [
//   "script-src contains 'unsafe-eval' which allows eval() and is dangerous",
//   "script-src contains 'unsafe-inline' without nonce or hash..."
// ]
```

Test CSP Online

Use online tools: - [CSP Evaluator](#) - [Report URI CSP Wizard](#)

Migration Guide

Step 1: Start with Report-Only

```
const csp = buildCSP({
  directives: DEFAULT_CSP_DIRECTIVES,
  reportOnly: true,
});
```

Step 2: Monitor Violations

Set up violation reporting:

```
const csp = buildCSP({
  directives: {
    ...DEFAULT_CSP_DIRECTIVES,
    'report-uri': '/csp-violations',
  },
  reportOnly: true,
});
```

Step 3: Fix Violations

Common fixes:

```
// Add missing domains
'script-src': ["'self'", 'https://cdn.example.com']

// Use nonces for inline scripts
autoNonce: true

// Allow specific styles
'style-src': ["'self'", "'unsafe-inline'] // For CSS-in-JS
```

Step 4: Enforce Policy

Remove report-only mode:

```
const csp = buildCSP({
  directives: finalDirectives,
  reportOnly: false, // Enforce!
});
```

Best Practices

1. Start Strict, Relax as Needed

```
// Start with strict
import { STRICT_CSP_DIRECTIVES } from '@philjs/ssr';

// Add only what you need
const csp = mergeCSP(STRICT_CSP_DIRECTIVES, {
  'script-src': ["'self'", 'https://trusted-cdn.com'],
});
```

2. Use Nonces, Not unsafe-inline

```
// BAD
'script-src': ["'self'", "'unsafe-inline']"

// GOOD
const csp = buildCSP({
  directives: { 'script-src': ["'self'"] },
  autoNonce: true,
});
```

3. Avoid unsafe-eval

```
// Avoid
'script-src': ["'self'", "'unsafe-eval']"

// If you must, isolate it
'script-src': ["'self'"],
'worker-src': ["'self'", "'unsafe-eval'] // Only for workers
```

4. Use SRI for CDN Resources

```
<script
  src="https://cdn.example.com/lib.js"
  integrity="sha384-oqVuAfxRKap7fdgcCY5uykM6+R9GqQ8K/uxy9rx7HNQlGYl1kPzQho1wx4JwY8wC"
  crossorigin="anonymous"
></script>
```

5. Review Regularly

CSP should evolve with your app:

```
// Audit periodically
const warnings = validateCSP(currentDirectives);
if (warnings.length > 0) {
  console.warn('CSP Issues:', warnings);
}
```

Troubleshooting

Issue: Inline Styles Blocked**Problem:** CSS-in-JS libraries blocked**Solution:** Allow unsafe-inline or use nonces

```
// Option 1: Allow unsafe-inline (less secure)
'style-src': ["'self'", "'unsafe-inline']"

// Option 2: Use nonces (more secure)
const csp = buildCSP({
  directives: { 'style-src': ["'self'"] },
  autoNonce: true,
});

// Apply nonce to style tags
<style nonce={csp.nonce}>...</style>
```

Issue: Third-Party Scripts Blocked**Problem:** Analytics, ads, or widgets blocked**Solution:** Add domains to allowlist

```
'script-src': [
  "'self'",
  'https://www.google-analytics.com',
  'https://cdn.segment.com',
]
```

Issue: WebSocket Connections Blocked**Problem:** WS connections fail**Solution:** Allow ws: and wss: protocols

```
'connect-src': ["'self'", 'ws://localhost:3000', 'wss://api.example.com']
```

Issue: Image Loading Issues**Problem:** Images from CDN blocked**Solution:** Allow image domains

```
'img-src': [
  "'self'",
  'https://images.example.com',
  'https:', // Allow all HTTPS images
  'data:', // Allow data URIs
]
```

Resources

- [MDN: Content Security Policy](#)
- [CSP Quick Reference](#)
- [Google CSP Guide](#)
- [XSS Prevention Guide](#)

Summary

Content Security Policy is a powerful defense against XSS and other attacks. PhilJS makes it easy to implement CSP with:

1. Secure default directives
2. Automatic nonce generation
3. Flexible configuration
4. Validation and testing tools
5. Report-only mode for safe migration

Start with report-only mode, fix violations, then enforce your policy for maximum security.

XSS Prevention Guide

Cross-Site Scripting (XSS) is one of the most common web vulnerabilities. This guide explains how PhilJS protects against XSS attacks and best practices for keeping your application secure.

Table of Contents

1. [Understanding XSS](#)
2. [PhilJS Built-in Protection](#)
3. [Context-Specific Escaping](#)
4. [Safe Content Rendering](#)
5. [Common XSS Patterns](#)
6. [Testing for XSS](#)

Understanding XSS

XSS attacks occur when untrusted data is included in a web page without proper validation or escaping, allowing attackers to execute malicious scripts in users' browsers.

Types of XSS

1. Reflected XSS

```
// Vulnerable code
function SearchResults({ query }: { query: string }) {
  return <div>Results for: {query}</div>; // SAFE in PhilJS (auto-escaped)
}

// Attack URL: /search?q=<script>alert('XSS')</script>
```

2. Stored XSS

```
// Vulnerable if not sanitized
function Comment({ text }: { text: string }) {
  // If 'text' comes from database with malicious content
  return <div>{text}</div>; // SAFE in PhilJS (auto-escaped)
}
```

3. DOM-based XSS

```
// Vulnerable code (in vanilla JS)
element.innerHTML = location.hash.substring(1); // DANGEROUS

// Attack URL: /page#<img src=x onerror=alert('XSS')>
```

PhilJS Built-in Protection

PhilJS automatically protects against XSS in most scenarios:

1. Automatic HTML Escaping

All text content is escaped by default:

```
function UserProfile({ name }: { name: string }) {
  // Even if name contains <script> tags, they will be escaped
  return <h1>Welcome, {name}!</h1>;
}

// Input: "<script>alert('XSS')</script>"
// Output: "Welcome, &lt;script&gt;alert('XSS')&lt;/script&gt;!"
```

2. Attribute Escaping

Attributes are properly escaped:

```
function UserLink({ url, title }: { url: string; title: string }) {
  // Both url and title are escaped
  return <a href={url} title={title}>Link</a>;
}

// Input: url = 'javascript:alert("XSS")', title = ''><script>alert("XSS")</script>'<
// The dangerous content is escaped
```

3. Server-Side Rendering Protection

PhilJS SSR escapes content before sending HTML to the client:

```
import { renderToString } from '@philjs/ssr';

const html = renderToString(
  <div>{userInput}</div>
);
// userInput is escaped in the rendered HTML
```

Context-Specific Escaping

Different contexts require different escaping strategies. PhilJS provides utilities for each context:

HTML Context

```
import { escapeHtml } from '@philjs/core';

const userComment = '<script>alert("XSS")</script>';
const safe = escapeHtml(userComment);
// Result: '&lt;script&gt;alert("XSS")&lt;/script&gt;'

function Comment({ text }: { text: string }) {
  return <div>{text}</div>; // Automatically escaped by PhilJS
}
```

Attribute Context

```
import { escapeAttr } from '@philjs/core';

const userInput = '' onload="alert('XSS')";
const safe = escapeAttr(userInput);

function Image({ alt }: { alt: string }) {
  return ; // Auto-escaped
}
```

JavaScript Context

```
import { escapeJs } from '@philjs/core';

const userName = 'John'; alert("XSS"); var x="";
const safe = escapeJs(userName);

function Analytics({ userId }: { userId: string }) {
  return (
    <script nonce={nonce}>
      `window.analytics.identify(${escapeJs(userId)})`;
    </script>
  );
}
```

URL Context

```
import { escapeUrl, sanitizeUrl } from '@philjs/core';

// For query parameters
const searchQuery = 'hello world & stuff';
const url = `/search?q=${escapeUrl(searchQuery)}`;

// For redirect URLs (with validation)
const redirectUrl = sanitizeUrl(userInput, ['example.com']);
if (redirectUrl) {
  window.location.href = redirectUrl;
}
```

Safe Content Rendering

Rich Text Content

When you need to render HTML from user input (e.g., blog posts, comments):

```
import { sanitizeHtml } from '@philjs/core';

function BlogPost({ content }: { content: string }) {
  // Sanitize HTML to remove dangerous elements
  const safe = sanitizeHtml(content, {
    allowedTags: ['p', 'br', 'strong', 'em', 'ul', 'ol', 'li', 'a'],
    allowedAttributes: {
      'a': ['href', 'title'],
    },
    allowedSchemes: ['http', 'https'],
  });

  // Use dangerouslySetInnerHTML only with sanitized content
  return <div dangerouslySetInnerHTML={{ __html: safe }} />;
}
```

Markdown Content

For Markdown, use a trusted library with XSS protection:

```
import { marked } from 'marked';
import DOMPurify from 'isomorphic-dompurify';

function MarkdownContent({ markdown }: { markdown: string }) {
  // First, convert Markdown to HTML
  const html = marked(markdown);

  // Then sanitize the HTML
  const safe = DOMPurify.sanitize(html);

  return <div dangerouslySetInnerHTML={{ __html: safe }} />;
}
```

User Avatars and Images

Validate image URLs to prevent XSS via SVG or data URLs:

```

import { sanitizeUrl } from '@philjs/core';

function Avatar({ imageUrl }: { imageUrl: string }) {
  // Validate URL
  const safeUrl = sanitizeUrl(imageUrl, ['cdn.example.com']);

  if (!safeUrl) {
    // Fallback to default avatar
    return ;
  }

  return <img src={safeUrl} alt="User Avatar" />;
}

```

Common XSS Patterns

Pattern 1: Event Handlers

```

// DANGEROUS - Never do this
function Button({ onClick }: { onClick: string }) {
  return <button onClick={onClick}>Click</button>; // XSS vulnerability
}

// SAFE - Use function references
function Button({ onClick }: { onClick: () => void }) {
  return <button onClick={onClick}>Click</button>;
}

```

Pattern 2: Dynamic Styles

```

// DANGEROUS - User-controlled styles
function StyledDiv({ userStyle }: { userStyle: string }) {
  return <div style={userStyle}>Content</div>; // Can inject CSS-based XSS
}

// SAFE - Object-based styles
function StyledDiv({ color }: { color: string }) {
  // Validate color value
  const validColors = ['red', 'blue', 'green'];
  const safeColor = validColors.includes(color) ? color : 'black';

  return <div style={{ color: safeColor }}>Content</div>;
}

```

Pattern 3: Dynamic URLs

```

// DANGEROUS - Unvalidated URLs
function Link({ href }: { href: string }) {
  return <a href={href}>Click</a>; // javascript: URLs are XSS
}

// SAFE - Validated URLs
import { sanitizeUrl } from '@philjs/core';

function Link({ href }: { href: string }) {
  const safeHref = sanitizeUrl(href, ['example.com']) || '#';
  return <a href={safeHref}>Click</a>;
}

```

Pattern 4: JSON in Script Tags

```

// DANGEROUS - Unescaped JSON
function InitialState({ data }: { data: object }) {
  return (
    <script>
      window.__INITIAL_STATE__ = ${JSON.stringify(data)};
    </script>
  );
}

// Attack: data = { xss: '</script><script>alert("XSS")</script>' }

// SAFE - Escaped JSON
function InitialState({ data }: { data: object }) {
  const json = JSON.stringify(data)
    .replace(/</g, '\\u003c')
    .replace(>/g, '\\u003e')
    .replace(/&/g, '\\u0026');

  return (
    <script nonce={nonce}>
      {`window.__INITIAL_STATE__ = ${json};`}
    </script>
  );
}

```

Pattern 5: SVG Content

```
// DANGEROUS - User SVG content
function Icon({ svg }: { svg: string }) {
  return <div dangerouslySetInnerHTML={{ __html: svg }} />; // SVG can contain scripts
}

// SAFE - Sanitized SVG
import { sanitizeHtml } from '@philjs/core';

function Icon({ svg }: { svg: string }) {
  const safe = sanitizeHtml(svg, {
    allowedTags: ['svg', 'path', 'circle', 'rect', 'line', 'polygon'],
    allowedAttributes: {
      '*': ['fill', 'stroke', 'stroke-width', 'd', 'cx', 'cy', 'r', 'x', 'y', 'width', 'height'],
    },
  });
  return <div dangerouslySetInnerHTML={{ __html: safe }} />;
}
```

Best Practices

1. Never Trust User Input

```
// Always validate and sanitize
function processInput(input: string): string {
  // Validate format
  if (!/^a-zA-Z0-9\s+/.test(input)) {
    throw new Error('Invalid input format');
  }

  // Sanitize
  return escapeHtml(input);
}
```

2. Use Content Security Policy

Combine XSS prevention with CSP headers:

```
import { buildCSP } from '@philjs/ssr';

const csp = buildCSP({
  directives: {
    'default-src': ["'self'"],
    'script-src': ["'self'"],
    'object-src': ["'none'"],
  },
  autoNonce: true,
});

response.headers.set(csp.header, csp.value);
```

3. Avoid dangerouslySetInnerHTML

Only use when absolutely necessary and always with sanitization:

```
// AVOID when possible
<div dangerouslySetInnerHTML={{ __html: userContent }} />

// PREFER escaping
<div>{userContent}</div>

// IF REQUIRED, sanitize first
import { sanitizeHtml } from '@philjs/core';
<div dangerouslySetInnerHTML={{ __html: sanitizeHtml(userContent) }} />
```

4. Validate URLs

Always validate URLs before using them:

```
import { sanitizeUrl } from '@philjs/core';

function handleRedirect(url: string) {
  const safe = sanitizeUrl(url, ['example.com', 'trusted-site.com']);

  if (!safe) {
    throw new Error('Invalid redirect URL');
  }

  return Response.redirect(safe);
}
```

5. Escape Data in All Contexts

Different contexts require different escaping:

```

import { escapeHtml, escapeAttr, escapeJs, escapeUrl } from '@philjs/core';

// In HTML
<div>{escapeHtml(data)}</div>

// In attributes
<div title={escapeAttr(data)} />

// In JavaScript
<script>var x = "{escapeJs(data)}";</script>

// In URLs
<a href={`/search?q=${escapeUrl(data)}`}>Search</a>

```

Testing for XSS

Manual Testing

Test with common XSS payloads:

```

const xssPayloads = [
  '<script>alert("XSS")</script>',
  '"><script>alert("XSS")</script>',
  '<img src=x onerror=alert("XSS")>',
  'javascript:alert("XSS")',
  '<svg onload=alert("XSS")>',
  '&lt;script&gt;alert("XSS")&lt;/script&gt;',
  '<iframe src="javascript:alert(\'XSS\')">',
];

```

// Test each input field with these payloads

Automated Testing

Use tools to scan for XSS vulnerabilities:

```

# Using OWASP ZAP
docker run -t owasp/zap2docker-stable zap-baseline.py -t http://your-app

# Using npm packages
npm install -g retire
retire --path /path/to/your/app

```

Unit Tests

Write tests for your sanitization functions:

```

import { describe, it, expect } from 'vitest';
import { escapeHtml, sanitizeHtml } from '@philjs/core';

describe('XSS Prevention', () => {
  it('should escape HTML special characters', () => {
    const input = '<script>alert("XSS")</script>';
    const output = escapeHtml(input);
    expect(output).toBe('&lt;script&gt;alert("XSS")&lt;/script&gt;');
  });

  it('should sanitize dangerous HTML', () => {
    const input = '<p>Safe</p><script>alert("XSS")</script>';
    const output = sanitizeHtml(input);
    expect(output).not.toContain('<script>');
    expect(output).toContain('<p>Safe</p>');
  });

  it('should remove event handlers', () => {
    const input = '<img src=x onerror=alert("XSS")>';
    const output = sanitizeHtml(input);
    expect(output).not.toContain('onerror');
  });
});

```

Security Checklist

- All user input is validated on the server
- Text content is automatically escaped by PhilJS
- dangerouslySetInnerHTML is only used with sanitized content
- URLs are validated before use
- CSP headers are configured
- Event handlers use function references, not strings
- JSON in script tags is properly escaped
- SVG content is sanitized
- Third-party libraries are kept up to date

 XSS testing is part of the QA process

Resources

- [OWASP XSS Prevention Cheat Sheet](#)
- [Content Security Policy Guide](#)
- [Security Overview](#)
- [MDN: XSS](#)

Summary

PhilJS provides strong XSS protection out of the box:

1. Automatic HTML escaping for all content
2. Context-aware escaping utilities
3. HTML sanitization for rich content
4. URL validation and sanitization
5. Safe JSON serialization

Remember: **Never trust user input.** Always validate, sanitize, and escape data based on the context where it will be used.

Security Audit Documentation

Comprehensive security guidelines and best practices for PhilJS applications.

Table of Contents

- [XSS Prevention](#)
- [Safe HTML Rendering](#)
- [Input Sanitization](#)
- [CSRF Protection](#)
- [Content Security Policy](#)
- [Rate Limiting](#)
- [Security Headers](#)
- [Authentication & Authorization](#)
- [Security Checklist](#)

XSS Prevention

How PhilJS Prevents XSS

PhilJS provides **automatic XSS protection** through its JSX rendering system. All dynamic content is escaped by default during server-side rendering.

Built-in HTML Escaping

The framework automatically escapes HTML special characters in text content:

```
// packages/philjs-core/src/render-to-string.ts
function escapeHtml(str: string): string {
  return str
    .replace(/&/g, "&amp;")
    .replace(/</g, "&lt;")
    .replace(/>/g, "&gt;")
    .replace(/\"/g, "&quot;")
    .replace(/\'/g, "&#39;");
}

function escapeAttr(str: string): string {
  return str.replace(/&/g, "&amp;").replace(/\"/g, "&quot;");
}
```

Safe by Default

```
// Safe - User input is automatically escaped
function UserProfile({ user }: { user: User }) {
  return (
    <div>
      <h1>{user.name}</h1>          {/* Escaped */}
      <p>{user.bio}</p>            {/* Escaped */}
      <span>{user.email}</span>     {/* Escaped */}
    </div>
  );
}

// Example: If user.name = "<script>alert('XSS')</script>"
// Rendered as: &lt;script&gt;alert('XSS')&lt;/script&gt;
```

Attribute Escaping

Attributes are also automatically escaped:

```
function UserLink({ username, url }: { username: string; url: string }) {
  return (
    <a href={url} title={username}>
      {username}
    </a>
  );
}

// If url = "javascript:alert('XSS')"
// Rendered as: href="javascript:alert('XSS')" (escaped quotes prevent injection)
```

Common XSS Vectors and Protections

1. Script Injection

```
// Unsafe input
const maliciousInput = "<script>alert('XSS')</script>";

// Safe - PhilJS escapes this
<div>{maliciousInput}</div>
// Renders as: &lt;script&gt;alert('XSS')&lt;/script&gt;
```

2. Event Handler Injection

```
// Unsafe input
const maliciousAttr = '' onload="alert('XSS')';

// Safe - PhilJS escapes attribute values

// Renders as: alt="&quot; onload=&quot;alert('XSS')"
```

3. URL Injection

```
// Validate URLs before using them
function SafeLink({ href, text }: { href: string; text: string }) {
  // Only allow safe protocols
  const isSafeUrl = (url: string): boolean => {
    try {
      const parsed = new URL(url);
      return ['http:', 'https:', 'mailto:'].includes(parsed.protocol);
    } catch {
      return false;
    }
  };

  return (
    <a href={isSafeUrl(href) ? href : '#'}>
      {text}
    </a>
  );
}

// Blocks: javascript:, data:, vbscript:, etc.
```

XSS Prevention Rules

```
// Rule 1: Never bypass escaping unless absolutely necessary
// Avoid this pattern:
<div dangerouslySetInnerHTML={{ __html: userInput }} /> // DANGEROUS

// Rule 2: If you must use raw HTML, sanitize it first
import DOMPurify from 'isomorphic-dompurify';

function SafeHTML({ html }: { html: string }) {
  const sanitized = DOMPurify.sanitize(html, {
    ALLOWED_TAGS: ['p', 'br', 'strong', 'em', 'a'],
    ALLOWED_ATTR: ['href', 'title'],
  });
}

return <div dangerouslySetInnerHTML={{ __html: sanitized }} />;
}

// Rule 3: Validate and sanitize on the server
// Never trust client-side validation alone
```

Safe HTML Rendering

When You Need Raw HTML

Sometimes you need to render HTML from trusted sources (e.g. markdown content, WYSIWYG editors):

Using DOMPurify

```

import DOMPurify from 'isomorphic-dompurify';

interface RichTextProps {
  content: string;
  allowedTags?: string[];
}

function RichText({ content, allowedTags }: RichTextProps) {
  const sanitized = DOMPurify.sanitize(content, {
    ALLOWED_TAGS: allowedTags || [
      'p', 'br', 'strong', 'em', 'u', 'a', 'ul', 'ol', 'li',
      'h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'blockquote', 'code', 'pre'
    ],
    ALLOWED_ATTR: ['href', 'title', 'target', 'rel'],
    ALLOW_DATA_ATTR: false,
  });
  return <div dangerouslySetInnerHTML={{ __html: sanitized }} />;
}

```

Content Security Policy for HTML

```

// Only allow specific HTML elements and attributes
const SAFE_HTML_CONFIG = {
  ALLOWED_TAGS: [
    // Text formatting
    'p', 'br', 'strong', 'em', 'u', 's', 'code', 'pre',
    // Headings
    'h1', 'h2', 'h3', 'h4', 'h5', 'h6',
    // Lists
    'ul', 'ol', 'li',
    // Links
    'a',
    // Quotes
    'blockquote', 'q',
  ],
  ALLOWED_ATTR: {
    'a': ['href', 'title', 'target', 'rel'],
    '*': ['class'], // Allow class on all tags
  },
  ALLOWED_URI_REGEXP: /^(?:https?|mailto):|[^\w.-]+(?:\w+\.|\w+|\$))/i,
  KEEP_CONTENT: true,
  RETURN_TRUSTED_TYPE: true,
};

function sanitizeHTML(dirty: string): string {
  return DOMPurify.sanitize(dirty, SAFE_HTML_CONFIG);
}

```

Markdown Rendering

```

import { marked } from 'marked';
import DOMPurify from 'isomorphic-dompurify';

function MarkdownContent({ markdown }: { markdown: string }) {
  // Step 1: Convert markdown to HTML
  const html = marked.parse(markdown);

  // Step 2: Sanitize the HTML
  const sanitized = DOMPurify.sanitize(html, {
    ALLOWED_TAGS: [
      'p', 'br', 'strong', 'em', 'code', 'pre',
      'h1', 'h2', 'h3', 'h4', 'h5', 'h6',
      'ul', 'ol', 'li', 'a', 'blockquote'
    ],
    ALLOWED_ATTR: ['href', 'title'],
  });

  return <div dangerouslySetInnerHTML={{ __html: sanitized }} />;
}

```

Server-Side Sanitization

Always sanitize on the server, not just the client:

```
// server/api/content.ts
import DOMPurify from 'isomorphic-dompurify';

export async function POST(request: Request) {
  const { content } = await request.json();

  // Sanitize on server before storing
  const sanitized = DOMPurify.sanitize(content, {
    ALLOWED_TAGS: ['p', 'br', 'strong', 'em'],
    ALLOWED_ATTR: [],
  });

  // Store sanitized content
  await db.content.create({ body: sanitized });

  return new Response(JSON.stringify({ success: true }));
}
```

Input Sanitization

Input Validation Patterns

1. Email Validation

```
function validateEmail(email: string): boolean {
  const emailRegex = /^[^@\s]+@[^\s]+\.[^\s@]+$/;
  return emailRegex.test(email);
}

function EmailInput() {
  const email = signal("");
  const error = signal("");

  const handleChange = (e: Event) => {
    const value = (e.target as HTMLInputElement).value;
    email.set(value);

    if (!validateEmail(value)) {
      error.set('Invalid email format');
    } else {
      error.set('');
    }
  };

  return (
    <div>
      <input
        type="email"
        value={email()}
        onInput={handleChange}
        required
      />
      {error() && <span class="error">{error()}</span>}
    </div>
  );
}
```

2. Text Input Sanitization

```
// Remove potentially dangerous characters
function sanitizeText(input: string): string {
  return input
    .replace(/[<>]/g, '') // Remove angle brackets
    .replace(/javascript:/gi, '') // Remove javascript: protocol
    .replace(/on\w+\s*/gi, '') // Remove event handlers
    .trim()
    .slice(0, 1000); // Limit Length
}

// Whitelist approach (safer)
function sanitizeUsername(input: string): string {
  // Only allow alphanumeric, underscore, hyphen
  return input.replace(/[^a-zA-Z0-9_-]/g, '').slice(0, 30);
}

function sanitizeAlphanumeric(input: string): string {
  return input.replace(/[^a-zA-Z0-9\s]/g, '');
}
```

3. Number Validation

```

function validateNumber(input: string, min?: number, max?: number): number | null {
  const num = parseFloat(input);

  if (isNaN(num)) return null;
  if (min !== undefined && num < min) return null;
  if (max !== undefined && num > max) return null;

  return num;
}

function QuantityInput() {
  const quantity = signal(1);

  const handleChange = (e: Event) => {
    const value = (e.target as HTMLInputElement).value;
    const num = validateNumber(value, 1, 100);

    if (num !== null) {
      quantity.set(num);
    }
  };

  return (
    <input
      type="number"
      min="1"
      max="100"
      value={quantity()}
      onChange={handleChange}
    />
  );
}

```

4. URL Validation

```

function validateUrl(url: string): boolean {
  try {
    const parsed = new URL(url);
    // Only allow http and https
    return ['http:', 'https:'].includes(parsed.protocol);
  } catch {
    return false;
  }
}

function sanitizeUrl(url: string): string | null {
  try {
    const parsed = new URL(url);

    // Block dangerous protocols
    if (!['http:', 'https:'].includes(parsed.protocol)) {
      return null;
    }

    return parsed.toString();
  } catch {
    return null;
  }
}

```

Form Validation

```

import { useForm, v as validators } from '@philjs/core';

function RegistrationForm() {
  const form = useForm({
    initialValues: {
      username: '',
      email: '',
      password: '',
      age: 0,
    },
    schema: {
      username: {
        validators: [
          validators.required('Username is required'),
          validators.minLength(3, 'Username must be at least 3 characters'),
          validators.maxLength(30, 'Username must be less than 30 characters'),
          validators.pattern(/^[a-zA-Z0-9_-]+$/, 'Username can only contain letters, numbers, _ and -'),
        ],
      },
      email: {
        validators: [
          validators.required('Email is required'),
          validators.email('Invalid email format'),
        ],
      },
      password: {
        validators: [
          validators.required('Password is required'),
          validators.minLength(8, 'Password must be at least 8 characters'),
          validators.pattern(/[A-Z]/, 'Password must contain uppercase'),
          validators.pattern(/[a-z]/, 'Password must contain lowercase'),
          validators.pattern(/[0-9]/, 'Password must contain number'),
        ],
      },
      age: {
        validators: [
          validators.min(18, 'Must be at least 18'),
          validators.max(120, 'Invalid age'),
        ],
      },
    },
    onSubmit: async (values) => {
      // All values are validated before reaching here
      await api.post('/register', values);
    },
  });
}

return (
  <form onSubmit={form.handleSubmit}>
    <div>
      <input
        name="username"
        value={form.values.username}
        onChange={form.handleChange}
      />
      {form.errors.username && <span>{form.errors.username}</span>}
    </div>

    <div>
      <input
        name="email"
        type="email"
        value={form.values.email}
        onChange={form.handleChange}
      />
      {form.errors.email && <span>{form.errors.email}</span>}
    </div>

    <div>
      <input
        name="password"
        type="password"
        value={form.values.password}
        onChange={form.handleChange}
      />
      {form.errors.password && <span>{form.errors.password}</span>}
    </div>

    <button type="submit" disabled={!form.isValid}>
      Register
    </button>
  </form>
);
}

```

CSRF Protection

PhilJS provides built-in CSRF (Cross-Site Request Forgery) protection through the `@philjs/ssr` package.

How CSRF Protection Works

CSRF protection uses secure random tokens to verify that requests originate from your application:

```
// packages/philjs-ssr/src/csrf.ts
export function generateCSRFToken(): string {
    return randomBytes(32).toString('hex'); // 256-bit random token
}
```

Server-Side Setup

```
import { csrfProtection, csrfField } from '@philjs/ssr';

// Configure CSRF middleware
const csrf = csrfProtection({
    getSessionId: (request) => {
        // Extract session ID from cookie or header
        const cookie = request.headers.get('cookie');
        return parseSessionId(cookie) || 'default';
    },
    skip: (request) => {
        // Skip CSRF for safe methods (GET, HEAD, OPTIONS)
        // and for API endpoints that use other auth (e.g., API keys)
        const url = new URL(request.url);
        return url.pathname.startsWith('/api/public/');
    },
});

// Generate token for a user session
const token = csrf.generateToken(request);
```

Form Protection

```
// Server route handler
export async function GET(request: Request) {
    const csrf = csrfProtection({ getSessionId: getSession });
    const csrfToken = csrf.generateToken(request);

    return renderToString(
        <form method="POST" action="/submit">
            {/* Hidden CSRF token field */}
            <input type="hidden" name="_csrf" value={csrfToken} />

            <input type="text" name="data" />
            <button type="submit">Submit</button>
        </form>
    );
}

// POST handler - verify CSRF token
export async function POST(request: Request) {
    const csrf = csrfProtection({ getSessionId: getSession });

    if (!csrf.verifyRequest(request)) {
        return new Response('CSRF token invalid', { status: 403 });
    }

    // Process form data
    const formData = await request.formData();
    // ...
}
```

AJAX/Fetch Protection

```

import { signal, effect } from '@philjs/core';

function SecureForm() {
  const csrfToken = signal('');

  // Fetch CSRF token on mount
  effect(async () => {
    const response = await fetch('/api/csrf-token');
    const data = await response.json();
    csrfToken.set(data.token);
  });

  const handleSubmit = async (e: Event) => {
    e.preventDefault();

    const response = await fetch('/api/submit', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'X-CSRF-Token': csrfToken(), // Include token in header
      },
      body: JSON.stringify({ data: '...' }),
    });

    // Handle response
  };
}

return (
  <form onSubmit={handleSubmit}>
    <input type="text" name="data" />
    <button type="submit">Submit</button>
  </form>
);
}

```

Token Storage

For production, use Redis or a similar distributed store:

```

import Redis from 'ioredis';
import { csrfProtection } from '@philjs/ssr';

const redis = new Redis();

// Custom CSRF store backed by Redis
class RedisCSRFStore {
  async set(sessionId: string, token: string, ttl: number = 3600000) {
    await redis.set(`csrf:${sessionId}`, token, 'PX', ttl);
  }

  async get(sessionId: string): Promise<string | null> {
    return await redis.get(`csrf:${sessionId}`);
  }

  async verify(sessionId: string, token: string): Promise<boolean> {
    const stored = await this.get(sessionId);
    return stored === token;
  }
}

// Use in your application
const csrfStore = new RedisCSRFStore();

```

Extract CSRF Token from Requests

```

import { extractCSRFToken } from '@philjs/ssr';

export async function POST(request: Request) {
  // Extract from header or form field
  const token = await extractCSRFToken(request);

  if (!token) {
    return new Response('Missing CSRF token', { status: 400 });
  }

  // Verify token
  const isValid = await verifyToken(token);
  // ...
}

```

SameSite Cookies

Combine CSRF tokens with SameSite cookies for defense in depth:

```

import { createCookie } from '@philjs/ssr';

const sessionCookie = createCookie('session', {
  httpOnly: true,           // Not accessible to JavaScript
  secure: true,             // HTTPS only
  sameSite: 'Strict',       // Strict CSRF protection
  maxAge: 86400,            // 24 hours
  path: '/',
});

// Set cookie
response.headers.set('Set-Cookie', sessionCookie.serialize(sessionId));

```

Content Security Policy

PhilJS provides built-in CSP (Content Security Policy) support to prevent XSS and other injection attacks.

CSP Implementation

```

import { buildCSP, createNonce } from '@philjs/ssr';

// Generate nonce for inline scripts
const nonce = createNonce(); // Cryptographically secure random value

// Build CSP header
const csp = buildCSP({
  nonce,
  directives: {
    'default-src': ["'self'"],
    'script-src': ["'self'", "'nonce-{NONCE}'"],
    'style-src': ["'self'", "'unsafe-inline'", 'https://fonts.googleapis.com'],
    'img-src': ["'self'", 'data:', 'https:'],
    'font-src': ["'self'", 'https://fonts.gstatic.com'],
    'connect-src': ["'self'", 'https://api.example.com'],
    'frame-ancestors': ['none'],
    'base-uri': ["'self'"],
    'form-action': ["'self'"],
    'upgrade-insecure-requests': [],
  },
  reportUri: '/api/csp-report', // Optional: report violations
});

```

// Set header

```

response.headers.set(csp.header, csp.value);

```

Default CSP Directives

PhilJS includes secure defaults:

```

const DEFAULT_DIRECTIVES = {
  'default-src': ["'self'"],
  'script-src': ["'self'"],
  'style-src': ["'self'", "'unsafe-inline'"], // unsafe-inline for style tags
  'img-src': ["'self'", 'data:', 'blob:'],
  'connect-src': ["'self'"],
  'font-src': ["'self'", 'data:'],
  'object-src': ['none'],
  'base-uri': ["'self'"],
  'frame-ancestors': ['none'],
  'form-action': ["'self'"],
  'upgrade-insecure-requests': []
};

```

Using Nonces for Inline Scripts

```

import { buildCSP, createNonce } from '@philjs/ssr';

export async function renderPage(request: Request) {
  const nonce = createNonce();
  const csp = buildCSP({ nonce });

  const html = `
    <!DOCTYPE html>
    <html>
      <head>
        <meta http-equiv="Content-Security-Policy" content="${csp.value}">
      </head>
      <body>
        <div id="root"></div>

        <!-- Inline script with nonce -->
        <script nonce="${nonce}">
          window.__INITIAL_STATE__ = ${JSON.stringify(state)};
        </script>
      </body>
    </html>
  `;

  return new Response(html, {
    headers: {
      'Content-Type': 'text/html',
      [csp.header]: csp.value,
    },
  });
}

```

CSP Report-Only Mode

Test CSP without breaking your site:

```

const csp = buildCSP({
  directives: { /* ... */ },
  reportOnly: true, // Don't enforce, only report violations
  reportUri: '/api/csp-report',
});

// Header will be: Content-Security-Policy-Report-Only

```

Handling CSP Reports

```

export async function POST(request: Request) {
  const report = await request.json();

  console.error('CSP Violation:', {
    documentUri: report['document-uri'],
    violatedDirective: report['violated-directive'],
    blockedUri: report['blocked-uri'],
    sourceFile: report['source-file'],
    lineNumber: report['line-number'],
  });

  // Store in Logging system
  await logCSPViolation(report);

  return new Response('Report received', { status: 204 });
}

```

Strict CSP

For maximum security, use a strict CSP with nonces:

```

const strictCSP = buildCSP({
  nonce,
  directives: {
    'default-src': ["'none'"],
    'script-src': ["'nonce-{NONCE}'", "'strict-dynamic'"],
    'style-src': ["'nonce-{NONCE}'"],
    'img-src': ["'self'", 'data:'],
    'font-src': ["'self'"],
    'connect-src': ["'self'"],
    'base-uri': ["'none'"],
    'form-action': ["'self'"],
    'frame-ancestors': ["'none'"],
    'object-src': ["'none'"],
    'upgrade-insecure-requests': [],
  },
});

```

Rate Limiting

PhilJS provides comprehensive rate limiting to protect against abuse and DDoS attacks.

Basic Rate Limiting

```
import { rateLimit, MemoryRateLimitStore } from '@philjs/ssr';

// Create rate limiter
const limiter = rateLimit({
  windowMs: 60 * 1000, // 1 minute window
  maxRequests: 100, // 100 requests per window
  message: 'Too many requests, please slow down',
});

// Use as middleware
export async function GET(request: Request) {
  const rateLimitResponse = await limiter(request, async () => {
    // Your handler
    return new Response('Success');
  });

  return rateLimitResponse;
}
```

Pre-configured Limiters

```
import {
  apiRateLimit,
  authRateLimit,
  apiKeyRateLimit,
  userRateLimit,
} from '@philjs/ssr';

// API routes: 60 requests/minute
export const apiLimiter = apiRateLimit(60);

// Auth routes: 5 attempts/minute
export const authLimiter = authRateLimit(5);

// API key: 1000 requests/minute
export const keyLimiter = apiKeyRateLimit(1000);

// Per-user: 100 requests/minute
export const userLimiter = userRateLimit(100, (request) => {
  return request.headers.get('x-user-id') || 'anonymous';
});
```

Custom Key Generation

```
const limiter = rateLimit({
  windowMs: 60000,
  maxRequests: 100,
  keyGenerator: (request) => {
    // Rate limit by API key
    const apiKey = request.headers.get('x-api-key');
    if (apiKey) return `api:${apiKey}`;

    // Fallback to IP address
    const ip = request.headers.get('x-forwarded-for') || 'unknown';
    return `ip:${ip}`;
  },
});
```

Redis Store for Production

```
import { RedisRateLimitStore } from '@philjs/ssr';
import Redis from 'ioredis';

const redis = new Redis({
  host: process.env.REDIS_HOST,
  port: parseInt(process.env.REDIS_PORT || '6379'),
});

const store = new RedisRateLimitStore(redis, 'app:ratelimit');

const limiter = rateLimit(
  {
    windowMs: 60000,
    maxRequests: 100,
  },
  store
);
```

Rate Limit Headers

PhilJS automatically adds standard rate limit headers:

```
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 95
X-RateLimit-Reset: 2025-12-17T12:00:00.000Z
```

Skip Successful/Failed Requests

```
// Only count failed login attempts
const loginLimiter = rateLimit({
  windowMs: 60000,
  maxRequests: 5,
  skipSuccessfulRequests: true, // Don't count successful logins
});

// Only count successful uploads
const uploadLimiter = rateLimit({
  windowMs: 60000,
  maxRequests: 10,
  skipFailedRequests: true, // Don't count failed uploads
});
```

Adaptive Rate Limiting

Automatically adjust limits based on error rates:

```
import { AdaptiveRateLimiter } from '@philjs/ssr';

const limiter = new AdaptiveRateLimiter({
  baseLimit: 100,
  windowMs: 60000,
  errorThreshold: 0.1,      // 10% error rate triggers reduction
  adaptationFactor: 0.5,    // Reduce by 50% when errors high
});

// In request handler
const rateLimitResponse = await limiter.check(request);
if (rateLimitResponse) return rateLimitResponse;

const response = await handleRequest(request);
const success = response.status < 400;

// Record result for adaptation
await limiter.recordResult(success);
```

Sliding Window Rate Limiting

More accurate than fixed windows:

```
import { SlidingWindowRateLimiter } from '@philjs/ssr';

const limiter = new SlidingWindowRateLimiter({
  windowMs: 60000,
  maxRequests: 100,
  message: 'Rate limit exceeded',
});

const rateLimitResponse = await limiter.check(request);
if (rateLimitResponse) return rateLimitResponse;
```

Security Headers

Essential Security Headers

PhilJS applications should include these security headers:

```

export function setSecurityHeaders(response: Response): Response {
  // Prevent MIME sniffing
  response.headers.set('X-Content-Type-Options', 'nosniff');

  // Prevent clickjacking
  response.headers.set('X-Frame-Options', 'DENY');

  // Enable XSS protection
  response.headers.set('X-XSS-Protection', '1; mode=block');

  // Control referrer information
  response.headers.set('Referrer-Policy', 'strict-origin-when-cross-origin');

  // Feature policy
  response.headers.set(
    'Permissions-Policy',
    'geolocation=(), microphone=(), camera=()'
  );

  // HTTPS only (HSTS)
  response.headers.set(
    'Strict-Transport-Security',
    'max-age=31536000; includeSubDomains; preload'
  );
}

return response;
}

```

Header Middleware

```

export function securityHeadersMiddleware(
  handler: (request: Request) => Promise<Response>
) {
  return async (request: Request): Promise<Response> => {
    const response = await handler(request);
    return setSecurityHeaders(response);
  };
}

// Usage
export const GET = securityHeadersMiddleware(async (request) => {
  return new Response('Hello');
});

```

Authentication & Authorization

Secure Session Management

```

import { createCookie } from '@philjs/ssr';

const sessionCookie = createCookie('session', {
  secrets: [process.env.SESSION_SECRET!],
  httpOnly: true,
  secure: true,
  sameSite: 'Strict',
  maxAge: 86400, // 24 hours
});

// Set session
export async function login(userId: string) {
  const sessionId = generateSecureId();

  // Store session in database/Redis
  await sessions.set(sessionId, {
    userId,
    createdAt: Date.now(),
    expiresAt: Date.now() + 86400000,
  });

  // Set cookie
  return sessionCookie.serialize(sessionId);
}

// Verify session
export async function getSession(request: Request) {
  const cookie = request.headers.get('cookie');
  const parsed = sessionCookie.parse(cookie);

  if (!parsed || !parsed.signed) return null;

  const session = await sessions.get(parsed.value);

  if (!session || session.expiresAt < Date.now()) {
    return null;
  }

  return session;
}

```

Password Hashing

Never store plain text passwords. Always hash:

```

import bcrypt from 'bcrypt';

// Hash password
export async function hashPassword(password: string): Promise<string> {
  const saltRounds = 12;
  return bcrypt.hash(password, saltRounds);
}

// Verify password
export async function verifyPassword(
  password: string,
  hash: string
): Promise<boolean> {
  return bcrypt.compare(password, hash);
}

// Example usage
export async function register(email: string, password: string) {
  const hashedPassword = await hashPassword(password);

  await db.users.create({
    email,
    password: hashedPassword, // Store hash, not plain text
  });
}

```

Role-Based Access Control

```

type Role = 'admin' | 'editor' | 'viewer';

interface User {
  id: string;
  role: Role;
}

export function requireRole(allowedRoles: Role[]) {
  return async (request: Request, next: () => Promise<Response>) => {
    const session = await getSession(request);

    if (!session) {
      return new Response('Unauthorized', { status: 401 });
    }

    const user = await getUser(session.userId);

    if (!allowedRoles.includes(user.role)) {
      return new Response('Forbidden', { status: 403 });
    }

    return next();
  };
}

// Usage
export const POST = requireRole(['admin', 'editor'])(async (request) => {
  // Only admins and editors can access this
  return new Response('Success');
});

```

Security Checklist

Development Checklist

- All user input is validated and sanitized
- XSS protection is enabled (automatic in PhilJS)
- CSRF tokens are used for state-changing operations
- Content Security Policy is configured
- Rate limiting is enabled for public endpoints
- Security headers are set
- Passwords are hashed with bcrypt/argon2
- Sessions use secure, httpOnly cookies
- HTTPS is enforced in production
- Secrets are stored in environment variables, not code
- Dependencies are regularly updated
- SQL injection prevention (use parameterized queries)
- File uploads are validated and scanned
- Error messages don't leak sensitive information
- Logging doesn't include sensitive data

Production Checklist

- SSL/TLS certificates are valid and up-to-date
- HSTS is enabled
- CSP is enforced (not report-only)
- Rate limiting uses Redis or distributed store
- CSRF tokens use Redis or distributed store
- Sessions expire after inactivity
- Failed login attempts are rate-limited
- Security headers are set on all responses
- Error tracking is configured (but sanitized)
- Regular security audits are scheduled
- Dependency vulnerability scanning is automated
- Backup and disaster recovery plan exists
- Access logs are monitored
- Intrusion detection is enabled

Code Review Checklist

- No hardcoded secrets or API keys
- No use of eval() or Function() with user input
- No direct HTML injection without sanitization
- All external data is validated

- Authentication is required for sensitive operations
 - Authorization checks are performed
 - Errors are handled gracefully
 - Logging is appropriate (not too much/little)
 - Third-party libraries are from trusted sources
 - Security best practices are followed
-

Summary

PhiJS provides comprehensive security features:

1. **XSS Prevention:** Automatic HTML escaping in JSX
2. **CSRF Protection:** Built-in token generation and verification
3. **CSP:** Content Security Policy with nonce support
4. **Rate Limiting:** Multiple strategies and storage backends
5. **Input Validation:** Form validation and sanitization
6. **Secure Cookies:** Signed, httpOnly, sameSite cookies
7. **Security Headers:** Best practice HTTP headers

Key Principles

- **Defense in Depth:** Multiple layers of security
- **Secure by Default:** Safe defaults that work out of the box
- **Validate Everything:** Never trust user input
- **Fail Securely:** Errors should not expose sensitive information
- **Least Privilege:** Grant minimum necessary permissions
- **Stay Updated:** Regular security updates and audits

Resources

- [OWASP Top 10](#)
- [OWASP Cheat Sheet Series](#)
- [Content Security Policy](#)
- [Web Security Academy](#)

Next Steps:

- Review your application against the security checklist
- Enable CSP and monitor violation reports
- Configure rate limiting for your API endpoints
- Set up CSRF protection for forms
- Implement proper authentication and authorization
- Schedule regular security audits

For questions or security concerns, please review the [security best practices guide](#).

Security Checklist

A comprehensive security checklist for PhiJS applications. Use this before deploying to production and during regular security audits.

Pre-Deployment Checklist

Application Security

Input Validation

- All user input is validated on the server-side
- Input validation uses strict schemas (e.g., Zod, Yup)
- File uploads are validated (type, size, content)
- URL parameters are validated and sanitized
- Form data is validated before processing
- JSON payloads are validated against schemas
- Email addresses are validated with proper regex
- Phone numbers are validated with proper format
- Dates and times are validated and normalized
- Numeric inputs have min/max constraints

Output Encoding

- All user-generated content is HTML-escaped
- JavaScript context escaping is used when needed
- URL parameters are properly encoded

- CSS values are escaped when user-controlled
- JSON responses are properly serialized
- XML/SVG content is sanitized
- No dangerouslySetInnerHTML without sanitization
- Rich text content uses allowlist-based sanitization

XSS Prevention

- Content Security Policy is configured
- CSP uses nonces for inline scripts
- No unsafe-inline in script-src (or uses nonces/hashes)
- No unsafe-eval in script-src
- Event handlers use function references, not strings
- Dynamic URLs are validated before use
- SVG content is sanitized
- User avatars/images are from trusted sources only

CSRF Protection

- CSRF tokens are used for all state-changing operations
- CSRF tokens are cryptographically random
- CSRF tokens are validated on the server
- SameSite cookie attribute is set
- Origin header is validated
- Double-submit cookie pattern is used (if applicable)

Authentication & Authorization

Authentication

- Passwords are hashed with bcrypt (12+ rounds)
- Password strength requirements are enforced
- Account lockout after failed login attempts
- Rate limiting on login endpoints
- Email verification is required for new accounts
- Password reset uses time-limited tokens
- Password reset tokens are single-use
- Sessions have reasonable timeout (1-24 hours)
- Session IDs are cryptographically random
- Sessions are invalidated on logout
- Old sessions are cleaned up regularly
- Multi-factor authentication is available (optional or required)

Session Management

- Sessions stored securely (Redis in production)
- Session cookies have HttpOnly flag
- Session cookies have Secure flag
- Session cookies use SameSite=Strict or Lax
- Session fixation is prevented
- Session hijacking protections in place
- Concurrent session limits (if needed)

Authorization

- Role-based access control is implemented
- Permission checks on all protected resources
- User can only access their own data
- Admin functions require admin role
- API endpoints check authorization
- Client-side authorization matches server-side

API Security

General API Security

- Rate limiting on all API endpoints
- Different rate limits for different endpoints
- API authentication is required
- API authorization checks are in place
- API versioning is used
- CORS is properly configured
- API documentation doesn't expose secrets

- API errors don't leak sensitive information

API Input/Output

- Request body size limits are enforced
- Content-Type validation is performed
- All API inputs are validated
- SQL injection prevention (parameterized queries)
- NoSQL injection prevention
- Command injection prevention
- Path traversal prevention
- Responses don't include sensitive data

Data Protection

Encryption

- HTTPS is enforced (no HTTP in production)
- TLS 1.2+ is used
- Strong cipher suites are configured
- Sensitive data is encrypted at rest
- Database connections use encryption
- Backup data is encrypted
- API keys/tokens are encrypted in storage

Secrets Management

- No secrets in code or version control
- Environment variables for configuration
- Secrets are rotated regularly
- Different secrets for different environments
- Secret management service used (e.g., Vault)
- Database credentials are secured
- API keys are secured
- Cookie secrets are random and secure

Data Handling

- Sensitive data is not logged
- PII is minimized and protected
- Credit card data follows PCI DSS
- User data can be exported (GDPR)
- User data can be deleted (GDPR/CCPA)
- Data retention policies are implemented
- Audit logs for sensitive operations

Infrastructure Security

Server Configuration

- Server OS is up to date
- Unnecessary services are disabled
- Firewall rules are configured
- SSH uses key-based authentication
- Root login is disabled
- Fail2ban or similar is configured
- Regular security updates are applied

Application Deployment

- Production uses separate database
- Database has restricted access
- Application runs as non-root user
- File permissions are restrictive
- Directory listing is disabled
- Debug mode is disabled in production
- Source maps are not deployed (or protected)
- Error details are not exposed

Network Security

- HTTPS redirect is configured
- HTTP Strict Transport Security (HSTS) header
- DDoS protection is in place

- CDN uses secure configuration
- DNS uses DNSSEC
- IP allowlisting for admin functions

Dependency Security

Dependency Management

- Dependencies are up to date
- `pnpm audit` shows no critical issues
- Automated dependency updates (Dependabot)
- Dependencies are from trusted sources
- Lock files are committed
- Unused dependencies are removed
- Vulnerability scanning is automated

Known Vulnerabilities

- `js-yaml` updated to 4.1.1+ (moderate severity)
- `undici` updated to 5.28.5+ (moderate severity)
- `esbuild` updated to latest (low severity)
- `glob` updated to 10.5.0+ (high severity)
- All critical CVEs are patched

Security Headers

Required Headers

- Content-Security-Policy is configured
- X-Content-Type-Options: nosniff
- X-Frame-Options: DENY or SAMEORIGIN
- X-XSS-Protection: 1; mode=block
- Referrer-Policy: strict-origin-when-cross-origin
- Permissions-Policy configured
- Strict-Transport-Security (HSTS) configured

Monitoring & Logging

Logging

- Security events are logged
- Authentication attempts are logged
- Authorization failures are logged
- Input validation failures are logged
- Error logs don't contain secrets
- Logs are centralized
- Log retention policy is defined
- Logs are protected from tampering

Monitoring

- Uptime monitoring is configured
- Error rate monitoring is set up
- Failed login monitoring
- CSP violation monitoring
- Rate limit violation alerts
- Suspicious activity alerts
- Performance monitoring

Development Checklist

Code Security

Secure Coding Practices

- TypeScript strict mode is enabled
- ESLint security rules are configured
- No hardcoded credentials
- No `console.log` of sensitive data
- Error handling doesn't expose internals
- Regular code reviews include security
- Security testing in CI/CD pipeline

Third-Party Integrations

- Third-party scripts are from CDN with SRI
- OAuth apps have minimal permissions
- Webhook signatures are verified
- API integrations use HTTPS
- Third-party cookies are minimized

Testing

Security Testing

- XSS testing with common payloads
- CSRF protection is tested
- Authentication bypass attempts tested
- Authorization bypass attempts tested
- Input validation edge cases tested
- Rate limiting is tested
- Session management is tested
- Error handling is tested

Automated Testing

- Security tests in test suite
- Dependency scanning in CI
- SAST (Static Application Security Testing)
- DAST (Dynamic Application Security Testing)
- Container scanning (if using containers)

Operational Checklist

Incident Response

Preparation

- Incident response plan documented
- Security contacts defined
- Escalation procedures defined
- Backup and recovery procedures tested
- Communication plan for breaches

Detection & Response

- Security monitoring is active
- Alert thresholds are configured
- Response team is trained
- Forensics tools are available
- Breach notification process defined

Compliance

Regulatory Compliance

- GDPR compliance (if EU users)
- CCPA compliance (if CA users)
- HIPAA compliance (if health data)
- PCI DSS compliance (if handling cards)
- SOC 2 compliance (if enterprise)
- Privacy policy is up to date
- Terms of service are up to date

Business Continuity

Backup & Recovery

- Regular backups are performed
- Backups are tested
- Backup encryption is enabled
- Disaster recovery plan exists
- RTO and RPO are defined
- Backup restoration is tested

Periodic Review (Quarterly)

Security Review

- Review access controls

- Review user permissions
- Review API keys and tokens
- Review third-party integrations
- Review security logs
- Review incident reports
- Update security documentation

Penetration Testing

- Annual penetration testing
- Bug bounty program (optional)
- Vulnerability disclosure program
- Security audit by third party

Training & Awareness

- Security training for developers
- Security awareness for all staff
- Phishing simulation tests
- Security best practices documented
- Security champions program

PhilJS-Specific Checks

Framework Security

- PhilJS version is up to date
- Security utilities are used correctly
- CSP configuration is optimal
- CSRF protection is enabled
- Rate limiting is configured
- Session management is secure
- Cookie settings are secure

SSR Security

- Server-side validation is in place
- State serialization is secure
- No secrets in client bundles
- Hydration data is validated
- CSP nonces are used for inline scripts
- Server rendering doesn't leak data

Client Security

- Client-side validation is supplementary
- No sensitive operations client-side only
- State is not user-modifiable
- XSS protection is tested
- CSP is enforced

Quick Start Security Checklist

For a minimum viable secure application:

- 1. HTTPS Only**
 - Force HTTPS redirect
 - Set HSTS header
- 2. Authentication**
 - Hash passwords with bcrypt
 - Validate password strength
 - Use secure session cookies
- 3. Input Validation**
 - Validate all user input
 - Use schema validation
 - Escape output
- 4. CSRF Protection**
 - Enable CSRF tokens
 - Set SameSite cookies
- 5. Rate Limiting**
 - Limit login attempts

- Limit API requests
- 6. Security Headers**
- Set CSP
 - Set X-Frame-Options
 - Set X-Content-Type-Options
- 7. Dependencies**
- Run `pnpm audit`
 - Fix critical vulnerabilities
- 8. Secrets**
- Use environment variables
 - No secrets in code
- 9. Error Handling**
- Don't expose internals
 - Log errors securely
- 10. Monitoring**
- Monitor failed logins
 - Monitor errors
 - Set up alerts

Severity Levels

Use this guide to prioritize issues:

Critical (Fix Immediately)

- SQL injection vulnerabilities
- Authentication bypass
- Exposed admin credentials
- Remote code execution
- Critical dependency CVEs

High (Fix Within 24 Hours)

- XSS vulnerabilities
- CSRF vulnerabilities
- Authorization bypass
- Insecure direct object references
- High severity dependency CVEs

Medium (Fix Within 1 Week)

- Missing rate limiting
- Weak password policies
- Information disclosure
- Missing security headers
- Medium severity dependency CVEs

Low (Fix in Next Release)

- Verbose error messages
- Missing HSTS
- Outdated dependencies (no CVE)
- Missing CSP directives
- Low severity dependency CVEs

Resources

- [OWASP Top 10](#)
- [OWASP Cheat Sheet Series](#)
- [PhilJS Security Overview](#)
- [XSS Prevention Guide](#)
- [API Security Guide](#)
- [Authentication Patterns](#)
- [CSP Guide](#)

Checklist Versions

Track which version of this checklist you've completed:

- Initial security review completed: _____
- Q1 security review completed: _____
- Q2 security review completed: _____
- Q3 security review completed: _____

Q4 security review completed: _____

Remember: Security is an ongoing process, not a one-time task. Review this checklist regularly and stay informed about new vulnerabilities and best practices.

Edge Security Considerations

Run PhilJS safely on edge/serverless platforms.

Runtime constraints

- Web APIs only; avoid Node-only modules (fs, net) in edge bundles.
- Keep secrets in env; never inline into client bundles.
- Validate env presence at startup; fail fast.

Network and origin policy

- Restrict outbound calls to allowed domains.
- Use HTTPS everywhere; validate certificates.
- Apply CORS rules deliberately; avoid * on credentials.

Data and caching

- Don't cache user-specific HTML; cache data with scoped tags.
- Sanitize and validate all inputs (params/query/body).
- Strip dangerous keys (`__proto__`, `constructor`) from JSON.

Headers and CSP

- Set CSP to disallow inline/eval; limit script/style origins.
- HSTS, X-Content-Type-Options: nosniff, Frame-Options/Frame-Ancestors.
- Set secure cookies (`HttpOnly`, `SameSite`, `Secure`) when applicable.

Webhooks and callbacks

- Verify signatures; use raw body when required by provider.
- Rate-limit and log webhook events; reject unexpected origins.

Logging and PII

- Redact sensitive fields; include request/trace ids.
- Avoid logging secrets/tokens; store logs securely.

Testing

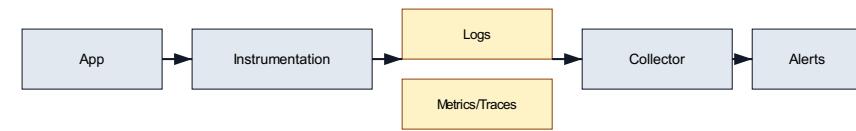
- Security tests for loaders/actions (authZ, validation, CSRF/SSR context).
- Pen-test headers and CSP; check for missing security headers.
- Simulate malicious payloads in tests to catch injection/prototype pollution.

Checklist

- Edge-safe imports (Web APIs only).
- Secrets/env validated; not bundled.
- CSP/headers set; cookies secure.
- Input validation/sanitization in loaders/actions.
- Webhooks verified; rate-limited.
- Logs redacted; request ids present.

Observability and Reliability

Visibility is mandatory when running PhilJS across edge and serverless targets. Collect logs, metrics, and traces so you can catch regressions and tune performance budgets. Reuse playbooks from `docs/observability`, `docs/performance`, and `docs/security` to keep this chapter actionable.



Observability pipeline

What to capture

- **Logs:** structured JSON with request id, route id, loader/action names, timing, and cache hit/miss.
- **Metrics:** TTFB, render time, hydration time, error counts, cache hit ratio, queue depth.
- **Traces:** link client navigation to server render and downstream API calls.
- **RUM:** LCP/FID/CLS and navigation timings from real users (sampled).
- **Feature flags:** include flag state in logs to correlate regressions.

Instrumentation points

- Router loaders/actions: measure execution time and tag with cache status.
- SSR render: wrap `renderToString`/`renderToStream` with timing spans.
- Edge adapters: attach platform request ids (Vercel/Netlify/CF) to logs.
- Client: capture hydration duration and key interactions (route change, data refetch).
- Stores: log undo/redo and persistence events when debugging data issues.
- Integrations: log upstream API latency and error codes.

Tools

- Use `@pjhiljs/observability` (if available) or platform-native exporters:
 - Vercel/Netlify/CF logs + analytics,
 - OpenTelemetry exporters for traces/metrics,
 - Console JSON logs for quick debugging.
- For front-end RUM, sample navigation timings and ship to your APM.
- Use Playwright traces to capture visual regressions and timings in CI.

Error handling

- Use error boundaries for UI; report errors with route and loader metadata.
- Redact PII before logging; never log secrets.
- Classify errors (user, network, server) for dashboards.
- Add “why” strings to surfaced errors for user trust and supportability.

Alerting and SLOs

- Define SLOs per surface (e.g., p95 TTFB < 400ms, error rate < 0.5%).
- Alert on budget breaches (bundle size, TTFB, cache hit ratio dropping).
- Include runbooks in alerts (how to roll back, which caches to invalidate).
- Track cold-start counts on edge/serverless targets; alert if they spike.

Local debugging

- Enable verbose logs in dev; print cache events and loader timings.
- Use DevTools performance tab for flamecharts and signal graphs.
- Record test runs with `vitest --watch` + trace output when debugging slow specs.
- For API flakiness, use MSW to capture/inspect requests locally.

Try it now: structured logging wrapper

```

function log(event: string, data: Record<string, unknown> = {}) {
  console.info(JSON.stringify({
    event,
    ts: Date.now(),
    route: data.route,
    loader: data.loader,
    cache: data.cache,
    reqId: data.reqId,
    ...data
  }));
}

export const userLoader = loader(async ({ params, signal, cache, request }) => {
  const reqId = request.headers.get('x-request-id') ?? crypto.randomUUID();
  const start = performance.now();
  try {
    const user = await getUser(params.id, signal);
    cache.tag(['user', params.id]);
    return { user };
  } finally {
    log('loader.user', { reqId, route: '/users/:id', loader: 'userLoader', ms: performance.now() - start });
  }
});

```

Pipe these logs to your platform's log drain and add a dashboard for loader latency and cache hit ratios.

Runbooks (examples)

- **Slow TTFB:** check edge cache headers, loader latency, and streaming flush; roll back heavy blocking code.
- **Hydration errors:** repro with Playwright + console, capture serialized data, compare server/client renders.
- **Cache misses spike:** inspect invalidation calls and cache tags; look for over-eager invalidate(['*']).
- **API regressions:** use MSW in staging to isolate backend vs frontend; add alerts on 5xx rates.

Logging and Tracing Deep Dive

Instrument PhilJS apps so you can pinpoint regressions and production issues quickly.

Logging

- Use structured logs (JSON) with event, ts, reqId, route, loader, status, ms, and cache fields.
- Redact PII; never log secrets/tokens.
- Include build/version/hash to correlate with releases.
- For client logs, sample to avoid noise; ship only errors/warnings plus key breadcrumbs.

Tracing

- Use OpenTelemetry where possible; wrap loaders/actions/SSR rendering in spans.
- Tag spans with route, params, cacheHit, and downstream API names.
- Propagate trace/req ids between client and server; include in logs for correlation.
- Record SSR TTFB, render duration, and hydration metrics as spans or metrics.

Metrics

- Core: TTFB, LCP/FID/CLS (RUM), hydration time, cache hit ratio, error rates, cold starts.
- Business: conversion, drop-off per step, feature flag impact.
- Edge/serverless: cold start counts, memory/timeouts, tail latency.

Pipelines

- Edge/serverless: send logs to platform drains (Vercel/Netlify/CF) or external log stores.
- Browsers: buffer and batch client logs; avoid blocking UX.
- Traces/metrics: export via OTLP/HTTP to your APM; minimize overhead in edge runtimes.

Dashboards to build

- Performance: TTFB, LCP, hydration, cache hits/misses, bundle sizes.
- Reliability: error rates per route/loader/action, top stack traces, cold starts.
- Business: funnel conversion, feature flag outcomes, AI success/failure and cost (if using AI).

Alerting

- Budget breaches: TTFB, LCP, bundle size, cache hit ratio.
- Error spikes: per route/loader/action.
- Cold start spikes (edge/serverless).
- AI failures/cost spikes (if applicable).

Testing instrumentation

- In CI, assert presence of required headers (trace ids, cache headers) in Playwright tests.
- Add unit tests for log redaction and schema (e.g., log objects validate against a Zod schema).
- Use fixtures to ensure trace context propagates through loaders/actions.

Checklist

- Structured logs with reqId/traceId/build.
- Spans for loaders/actions/SSR; tags for cache hit/miss.
- Core metrics emitted and dashboarded.
- Alerts for performance, errors, and cold starts.
- Log redaction and sampling in place.

Instrumentation Examples

Practical snippets for logging, metrics, and tracing in PhilJS.

Logging loader timing

```
function log(event: string, data: Record<string, unknown> = {}) {
  console.info(JSON.stringify({ event, ts: Date.now(), ...data }));
}

export const usersLoader = loader(async ({ signal, request, cache }) => {
  const reqId = request.headers.get('x-request-id') ?? crypto.randomUUID();
  const start = performance.now();
  try {
    const users = await fetchUsers(signal);
    cache.tag(['users']);
    return { users };
  } finally {
    log('loader.users', { reqId, ms: performance.now() - start, cache: 'miss' });
  }
});
```

Metrics (edge-friendly)

```
import { counter, histogram } from '@philjs/observability'; // if available

const loaderHits = counter('loader_hits', { description: 'Loader calls' });
const loaderLatency = histogram('loader_latency_ms');

export const projectLoader = loader(async ({ params, signal }) => {
  const end = loaderLatency.startTimer();
  loaderHits.add(1, { loader: 'project' });
  const data = await fetchProject(params.id, signal);
  end({ loader: 'project' });
  return { data };
});
```

Tracing

```
import { trace } from '@philjs/observability';

export const action = action(async ({ formData }) => {
  return trace('action.createProject', async (span) => {
    const name = String(formData.get('name') ?? '');
    span.setAttribute('project.name.length', name.length);
    return createProject({ name });
  });
});
```

Client RUM (lightweight)

```
if ('performance' in window) {
  const nav = performance.getEntriesByType('navigation')[0] as PerformanceNavigationTiming;
  sendRum({ ttfb: nav.responseStart, lcp: /* compute via PerformanceObserver */ });
}
```

Testing instrumentation

- In Vitest, assert logs/metrics by mocking exporters.
- In Playwright, capture console logs and ensure no PII; verify trace headers exist in network requests.

Checklist

- Logs structured with reqId/route/loader info.
- Metrics for loader/action counts and latency.
- Traces wrap critical sections; attributes for cache hit/miss.
- RUM collected with sampling; privacy-respecting.

Observability Runbooks

Quick actions for common production issues.

Slow TTFB

- Check edge cache hit rate; widen `staleTime`/revalidate; ensure ISR enabled.
- Profile loader latency; cache hot queries; move heavy work out of critical path.
- Stream earlier; trim HTML and critical CSS; lazy-load heavy widgets.

Hydration errors

- Reproduce with Playwright + console; capture server HTML and client render.
- Verify stable keys and deterministic data; fix mismatched markup.
- Add/adjust error boundaries around affected islands.

Cache misses spike

- Inspect invalidation calls for over-broad keys ([`*`]).
- Deduplicate in-flight requests; add jitter to revalidate.
- Check upstream cache headers; ensure tags are scoped correctly.

Memory creep (client)

- Run heap snapshots before/after nav loops; look for retained DOM/listeners.
- Ensure `onCleanup` removes listeners; bound store history/persistence.
- Dispose unused slices and clear caches.

Cold start spikes (edge/serverless)

- Reduce bundle size; trim deps.
- Increase warmup/keep-alive if platform supports it.
- Move heavy routes to regional functions if edge limits are hit.

Error rate increase

- Break down by route/loader/action; identify offending endpoints.
- Check recent deploys/feature flags; roll back or disable.
- Add detailed logging for failures; trace to downstream dependencies.

Runbook hygiene

- Keep logs with reqId/tracId; ensure dashboards have TTFB/error/cache panels.
- Add alerts for budgets (TTFB, error rate, cache hit ratio, cold starts).
- Document rollback steps per platform; test periodically.

Architecture Overview (PhilJS)

Understand how PhilJS pieces fit together from request to render.



Data flows from router to loaders/actions, through caching, into SSR/islands, and finally hydration.

Architecture overview diagram

Layers

- **Router**: defines routes, loaders, and actions; coordinates data and navigation.
- **Core**: signals, memos, resources, stores; the reactive engine.
- **SSR/Islands**: server render + streaming + selective hydration.

- **Adapters:** platform bindings (edge/serverless/static); handle env, caches, and HTTP.
- **Tooling:** CLI, devtools, builder, migrate, lint/typecheck/budget scripts.

Request/response flow

1. Request hits adapter (edge/serverless).
2. Router loader runs; uses caches and tags; may stream partial HTML.
3. SSR renders HTML; islands annotated for hydration.
4. Response streams to client with cache headers.
5. Client hydrates islands; resources fetch secondary data; actions mutate and invalidate.

Data and caching

- Loaders own data fetching and caching; actions own mutations and invalidation.
- Cache tags + staleTime + revalidate drive freshness and ISR/edge caches.
- Resources handle panel-level async states; stores handle client state.

State strategy

- Signals for local UI.
- Stores for domain state with middleware/history/persistence.
- Resources for async data; avoid putting server state in stores when loaders can cache it.

Performance hooks

- Prefetch via router; streaming SSR; islands hydrate lazily.
- size-limit, benches, devtools flamecharts, profiling scripts.

Observability

- Logs/metrics/traces per loader/action/SSR request.
- Cache hit/miss tagged; request ids propagated.
- Devtools for local inspection; APM for production.

Security

- Validation in loaders/actions; sanitize inputs/outputs.
- Edge-safe APIs; secrets in env; CSP and headers set at adapter level.

Deployment patterns

- Edge for latency-sensitive pages; regional for heavy jobs.
- Static/ISR for mostly-static routes; live widgets as islands.
- Rollouts with feature flags and cache-aware invalidation.

Checklist

- Router/data layer designed with cache tags and revalidate hints.
- SSR + islands strategy per route.
- State split between signals/stores/resources appropriately.
- Observability/logging wired with req/trace ids.
- Security checks at boundaries; CSP/headers configured.

Runtime Internals

Peek into how PhilJS schedules updates, tracks dependencies, and hydrates islands.

Reactivity graph

- Signals are nodes; effects/memos/resources subscribe to dependencies.
- Writes mark dependents dirty; batching coalesces recomputations.
- untrack reads without subscription; batch defers notifications.

Scheduling

- Synchronous by default for deterministic UI updates.
- Effects run after writes; avoid long-running work inside effects.
- Resources manage async lifecycles and expose loading/error states.

Hydration pipeline

- SSR marks islands with IDs and serialized props/data.

- Client locates island roots; hydrates using registered components.
- Hydration strategies (immediate/visible/idle/interaction) control timing.

Loader/action lifecycle

- Loader executes before render; can return cached data with tags + staleTime.
- Action runs on mutation; can invalidate tags and return updated data.
- Both receive `signal` for cancellation and `cache` for tagging/invalidation.

Error handling

- Errors propagate to nearest boundary (route/layout/component).
- Actions/loaders can return typed errors (`Err`) or throw to boundaries.
- Hydration errors surface in DevTools and logs; ensure boundaries cover islands.

DevTools hooks

- Signal graph visualization to spot over-subscription.
- Flamecharts for render/hydration profiling.
- Router panel for loader/action timings and cache hits.

Checklist

- Batch related writes to minimize recompute.
- Avoid async inside effects; use resources or actions.
- Hydration strategies chosen per island.
- Loaders/actions tagged and cancellable.
- Error boundaries wrap layouts and islands.

Data Modeling and Domain Design

Shape your data so loaders, stores, and UI stay simple and consistent.

Principles

- Normalize where needed; denormalize for hot paths cautiously.
- Stable identifiers for entities; use them for cache tags and keys.
- Version schemas; keep backward compatibility when possible.
- Keep server truth authoritative; client caches are hints.

Shapes for loaders

- Return serializable, minimal payloads.
- Include metadata: `etag`, `updatedAt`, `permissions`, `flags`.
- Avoid large blobs; provide URLs for deferred assets.

Shapes for stores

- Keep domain slices small and cohesive (e.g., `auth`, `preferences`, `cart`).
- Use selectors for derived views; avoid duplicating derived data.
- Limit history/persistence to necessary fields.

Relationships

- Represent relations with ids; load details lazily.
- For lists, include paging cursors and total counts.
- Use tags per entity + list to make invalidation precise.

Validation and migrations

- Validate incoming data with schemas; log schema drift.
- Migrate stored/persisted data between versions with migration steps.
- Avoid silent data shape changes; version and test migrations.

Offline and sync

- Store minimal drafts; mark records with sync status.
- Handle conflicts explicitly (keep both, merge, or latest-wins with user notice).
- Avoid storing secrets; encrypt sensitive fields.

Testing

- Contract tests for server/client agreement (see API contract testing).
- Fixture factories with realistic shapes; avoid overly generic data.
- Fuzz test selectors and reducers for edge cases.

Checklist

- Entities have stable ids and tags.
- Loader responses minimal, serializable, and validated.
- Stores split by domain; selectors used for derived data.
- Migrations defined for persisted data changes.
- Offline/conflict strategy documented and tested.

OpenAPI and REST Integration

PhilJS leans on loaders/actions plus typed clients to keep REST integrations predictable. This chapter shows how to generate clients, validate data, and keep performance in check. Pull from `docs/api`, `docs/api-reference`, and `docs/data-fetching` for deeper recipes, but keep this chapter as your copy/paste-ready source.

Generate a typed client

Use `philjs-openapi` (or `openapi-typescript`) to generate clients from your spec:

```
pnpm dlx openapi-typescript https://api.example.com/openapi.json -o src/api/types.ts
```

Create a thin client:

```
import { paths } from './api/types';

type GetUser = paths['/users/{id}']['get'];

export async function getUser(id: string, signal?: AbortSignal) {
  const res = await fetch(`/api/users/${id}`, { signal });
  if (!res.ok) throw new Error('Failed to load user');
  return (await res.json()) as GetUser['responses']['200']['content']['application/json'];
}
```

Use in loaders

```
import { loader } from '@philjs/router';
import { getUser } from '../api/client';

export const userLoader = loader(async ({ params, signal }) => {
  return {
    user: await getUser(params.id, signal)
  };
}, { cache: { staleTime: 30_000 }});
```

- Respect `signal` for cancellation.
- Set `staleTime` and `refetchOnFocus` intentionally to avoid network storms.
- Tag caches by entity; avoid duplicating caches in multiple layers.
- When mixing SSR and client fetches, keep loader outputs serializable and stable across renders.

Mutations and optimistic UI

```
import { action } from '@philjs/router';
import { updateUser } from '../api/client';

export const updateUserAction = action(async ({ request }) => {
  const body = await request.json();
  return updateUser(body);
}, { optimistic: true });
```

Pair optimistic updates with rollback handlers if the server rejects changes.

Validation

- Use Zod/Valibot on the boundary; narrow unknown JSON before it hits your components.
- Log schema mismatches in development to catch drift between client and server.
- Keep error objects ergonomic—map HTTP errors to typed domain errors.
- Reject `__proto__`/constructor keys during parsing to avoid prototype pollution.

Caching and invalidation

- Cache critical reads in loaders; avoid duplicating caches in stores.
- Invalidate by key after mutations: `cache.invalidate(['users', id])`.
- Prefer partial updates to refetch-all; stream updates via SSE/WebSockets when possible.
- For paginated lists, invalidate by filter: `['users', { page, query }]`.

- Use `staleTime` + background refetch for dashboards that need to stay fresh.

Security

- Strip `__proto__`/constructor keys from JSON before use.
- Default to `credentials: 'omit'` and pass tokens via headers; avoid mixing auth in query params.
- Rate-limit UI actions that trigger server mutations.
- Use `Content-Security-Policy` to prevent inline script injection when rendering API data.

Testing

- Contract tests with mock servers (MSW) to lock in shapes.
- Fixture-driven tests for loaders/actions to verify caching and error paths.
- Performance tests for hot endpoints to protect p95 budgets.
- Add regression tests for JSON parsing that include malicious payloads.

Try it now: end-to-end typed loader + mutation

1. Generate types: `pnpm dlx openapi-typescript http://localhost:3000/openapi.json -o src/api/types.ts`
2. Create client functions with `signal` support (see above).
3. Wire a loader/action pair:

```
export const usersLoader = loader(async ({ signal }) => ({
  users: await listUsers(signal),
}, { cache: { staleTime: 10_000, tags: [['users']] } });

export const createUserAction = action(async ({ request, cache }) => {
  const payload = await request.json();
  const user = await createUser(payload);
  cache.invalidate(['users']);
  return user;
}, { optimistic: true });
```

4. Add MSW handlers mirroring the OpenAPI schema and run Vitest to verify optimistic flow and cache invalidation.

GraphQL Integration

PhilJS works well with GraphQL clients (`urql`, `Apollo`, `graphql-request`) and server-driven schemas. The key is to keep data fetching aligned with loaders/actions and to cache intentionally. For reference, see [docs/api](#), [docs/data-fetching](#), and [docs/advanced](#)—reuse those snippets here as needed.

Choosing a client

- `graphql-request`: light, no cache; great for SSR/edge where you want explicit control.
- `urql`: flexible exchanges, built-in cache, SSR support.
- **Apollo Client**: feature-rich, heavier; consider only when you need its ecosystem.

Example with `graphql-request`

```
import { GraphQLClient, gql } from 'graphql-request';

const client = new GraphQLClient('/api/graphql', { fetch });

const UserQuery = gql`query User($id: ID!) {
  user(id: $id) { id name email }
}`;

export async function fetchUser(id: string, signal?: AbortSignal) {
  return client.request(UserQuery, { id }, { signal });
}
```

Use in a loader:

```
import { loader } from '@philjs/router';
import { fetchUser } from '../api/graphql';

export const userLoader = loader(async ({ params, signal }) => ({
  user: await fetchUser(params.id, signal),
}, { cache: { staleTime: 60_000, tags: [['user', params.id]] } });
```

Mutations with optimistic updates

```

const UpdateUser = gql`  
mutation UpdateUser($id: ID!, $input: UserInput!) {  
  updateUser(id: $id, input: $input) { id name email }  
}  
  
export const updateUserAction = action(async ({ request }) => {  
  const { id, input } = await request.json();  
  return client.request(UpdateUser, { id, input });  
, { optimistic: true }});
```

Ensure the optimistic shape matches the query shape; invalidate ['user', id] after commit.

Caching patterns

- Tag results by entity and by list filters; invalidate both after writes.
- For SSR, prefetch queries in loaders and embed dehydrated cache into HTML.
- Use `stale-while-revalidate` for list routes; keep detail routes fresher.
- For pagination, include cursor/page in tags (e.g., ['users', 'page', page]).
- For multi-tenant apps, include tenant in tags to avoid cache bleed.

Subscriptions and live data

- Use server-sent events or WebSockets; map events to cache updates.
- Keep reconnect logic exponential and respect `signal` for teardown.
- Avoid flooding the UI; coalesce events with `batch()` in PhilJS.

Schema drift and safety

- Generate types from schema regularly (`pnpm graphql-codegen`).
- Validate unknown fields before touching the UI; log discrepancies in dev.
- Enforce auth at the loader/action boundary; never assume client trust.
- Strip `__proto__`/`constructor` keys from subscription payloads.
- Limit query depth/complexity server-side if you control the API.

Testing

- Snapshot GraphQL responses per operation using mocks (MSW).
- Exercise cache invalidation logic with integration tests in Vitest + jsdom.
- Cover subscription reconnect logic with fake timers and controlled server events.
- Add E2E smoke tests in Playwright to assert SSR + hydration for GraphQL-heavy routes.
- Validate persisted queries or operation whitelists in CI to prevent breaking changes.

Client patterns to borrow

- **Fragments:** co-locate UI with fragments; generate types to keep props tight.
- **Persisted queries:** reduce payload and improve cache keys; great for edge functions.
- **Batching:** enable query batching on servers that support it to cut round trips.
- **Defer/stream:** if your GraphQL server supports it, align with PhilJS streaming for partial responses.

Try it now: SSR + client hydration with urql

1. Set up urql client in server entry to prefetch queries for SSR.
2. Wrap your App with the urql provider on both server and client.
3. In a loader, pre-execute critical queries and serialize the cache:

```

import { initUrql } from './urql';  
  
export const dashboardLoader = loader(async ({ signal }) => {  
  const { client, extract } = initUrql({ fetch, signal });  
  await client.query(DashboardQuery, {}).toPromise();  
  return { urqlState: extract() };  
});
```

4. In the client, hydrate the cache with `urqlState` and render. Verify with Playwright that the first paint contains data without a client refetch.

tRPC Integration

Use tRPC for end-to-end type safety without maintaining OpenAPI/GraphQL schemas. PhilJS pairs well with tRPC loaders/actions and SSR.

Setup

- Expose your tRPC router on the server (Node/edge compatible).
- Create a typed client with `@trpc/client`.
- Provide fetch/links suitable for your runtime (edge-safe where needed).

```

import { createTRPCProxyClient, httpBatchLink } from '@trpc/client';
import type { AppRouter } from '../server/router';

export const trpc = createTRPCProxyClient<AppRouter>({
  links: [httpBatchLink({ url: '/trpc' })],
});

```

Loaders and actions

```

import { loader, action } from '@philjs/router';
import { trpc } from '../lib/trpc';

export const userLoader = loader(async ({ params, signal }) => ({
  user: await trpc.user.byId.query({ id: params.id }, { signal })
}));

export const updateUser = action(async ({ formData, cache }) => {
  const id = String(formData.get('id') ?? '');
  const name = String(formData.get('name') ?? '');
  await trpc.user.update.mutate({ id, name });
  cache.invalidate(['user', id]);
}, { optimistic: true });

```

SSR considerations

- Use fetch links that work on edge (no Node-only globals).
- Prefetch queries in loaders and dehydrate cache if using a client cache.
- Avoid bundling server-only code into the client; keep routers on the server.

Error handling

- Map tRPC errors to user-friendly messages; surface validation errors inline.
- Log server errors with route and procedure metadata.

Testing

- Mock tRPC client in unit/integration tests; assert loader/action behavior.
- In E2E, run against dev server; verify SSR + hydration works without double-fetching.

Checklist

- Edge-safe tRPC links configured (if deploying to edge).
- Loaders/actions typed end-to-end.
- Cache invalidation after mutations.
- Validation errors handled gracefully.
- Tests cover happy/error paths and SSR hydration.

WebSockets and Realtime Updates

Deliver live experiences (chat, presence, dashboards) with WebSockets or Server-Sent Events while keeping caches and UI in sync.

When to use

- Multi-user collaboration (presence, cursors).
- Live dashboards/logs.
- Notifications/alerts that must arrive immediately.

Client setup

- Use `@philjs/realtime` (if available) or native WebSocket.
- Keep connection lifecycle in a dedicated module; expose signals/stores for UI.
- Reconnect with backoff; cap retries and surface connection status to users.

```

const status = signal<'connecting' | 'open' | 'closed'>('connecting');
const socket = new WebSocket('wss://example.com/ws');
socket.onopen = () => status.set('open');
socket.onclose = () => status.set('closed');

```

Cache coherence

- Map incoming events to cache invalidation or direct store updates.
- Use tags (e.g., `['project', id]`) to invalidate loader caches on relevant events.
- Batch UI updates with `batch()` to avoid churn.

Security

- Authenticate sockets with short-lived tokens; refresh on expiry.
- Validate incoming messages; never trust client-sent events blindly.
- Avoid leaking PII in payloads; encrypt if needed; rate-limit server-side.

Performance

- Throttle high-frequency events; coalesce updates before hitting the UI.
- Use presence heartbeats at sane intervals; drop idle connections.
- For large payloads, consider delta updates or patches instead of full objects.

Testing

- Use MSW/WebSocket mocks or custom test servers to simulate events.
- Integration tests: open socket, emit events, assert UI updates and cache invalidation.
- E2E: Playwright to verify live updates appear without reload.

Checklist

- Reconnect with backoff; surface status.
- Validate messages; auth with short-lived tokens.
- Cache invalidation or direct store updates wired.
- Batch updates to avoid render churn.
- Tests cover connect/reconnect/events.

AI Integration

Integrate LLMs and ML services into PhilJS apps safely and predictably.

Principles

- Typed prompts and outputs; validate responses before applying.
- Guardrails first: schema validation, safety filters, and policy checks.
- Cost-aware: track tokens and latency; cap spend per user/workflow.
- User trust: transparency and undo for AI actions.

Calling models

- Use `@philjs/ai` (if available) or direct HTTP calls to your provider.
- Pass context (locale, role, feature flags) explicitly.
- Prefer JSON outputs with strict schemas to reduce hallucinations.

```
import { callAI } from '@philjs/ai';
import { z } from 'zod';

const schema = z.object({ summary: z.string().max(500) });

export async function summarize(text: string) {
  const res = await callAI({ prompt: `Summarize: ${text}`, schema });
  return schema.parse(res);
}
```

Guardrails

- Validate against schemas; reject/repair invalid outputs.
- Add safety filters for toxicity/PII.
- Enforce policies per intent (see Nexus chapter); never let the model bypass auth.

Caching and idempotency

- Cache deterministic prompts by hash to reduce cost.
- Use request ids to prevent duplicate actions on retries.
- Log prompt/response metadata for audit (redact sensitive user data).

UI patterns

- Show spinners with cancel buttons; allow users to abort.
- Provide "why" explanations and let users edit/undo AI-applied changes.
- For streaming responses, render partial output with fallbacks.

Testing

- Mock AI calls with fixtures; test happy/error/invalid outputs.
- Fuzz-test schema validation against malformed responses.
- Add rate-limit tests to ensure UI handles "quota exceeded" gracefully.

Observability

- Metrics: latency, token usage, success/failure, safety-blocked counts.
- Traces: intent → prompt → validation → state mutation.
- Alerts: spike in failures/latency/costs.

Checklist

- Typed prompts and schemas.
- Safety filters and policy checks.
- Cost limits and caching for deterministic prompts.
- Undo and transparency in UI.
- Tests for invalid outputs and rate limits.

Payments Integration

Handle payments safely with predictable UX and secure server-side handling.

Principles

- Keep secrets server-side; clients use ephemeral tokens only.
- Validate amounts/currency/server-calculated totals on the server.
- Idempotent mutations to prevent double-charges.

Client flow (example with Stripe)

- Fetch a payment intent client secret via a loader/action.
- Use provider UI components or your own; keep UI reactive with signals.
- Handle required 3DS/authorization flows; surface clear statuses.

```
const pay = action(async ({ formData }) => {
  const amount = Number(formData.get('amount') ?? 0);
  const intent = await createPaymentIntent(amount); // server-side secret use
  return { clientSecret: intent.clientSecret };
});
```

Server handling

- Verify signatures on webhooks; use raw body as required by provider.
- Recalculate totals server-side; never trust client-submitted amounts.
- Record transaction state and idempotency keys; reconcile on retry.

UX patterns

- Show clear states: idle, processing, succeeded, failed, requires_action.
- Keep form inputs disabled during processing; allow retry on recoverable errors.
- Provide receipts and history; sync with backend state, not just client.

Testing

- Use provider test keys/modes; cover success/failure/3DS flows.
- Simulate webhook events in integration tests to ensure idempotency.
- E2E: run against sandbox, assert UI states and backend state consistency.

Security

- Never expose secret keys to the client.
- Validate webhook origins and signatures.
- Rate-limit actions that initiate payments.
- Sanitize all metadata fields to prevent injection.

Checklist

- Server recalculates totals and enforces idempotency.
- Webhooks verified; raw body handling correct.
- Client uses ephemeral tokens only.
- UI covers all payment states with clear messaging.
- Tests cover sandbox flows + webhook reconciliation.

API Versioning and Compatibility

Keep APIs stable and predictable for PhilJS clients across releases.

Versioning strategies

- **URL/version header:** e.g., /api/v1 or Accept: application/vnd.api+json;version=1.
- **Feature flags** for gradual rollout of breaking changes.
- **Deprecated fields:** mark in responses, remove after a grace period.

Client impact

- Lock generated types to a specific API version.
- Surface deprecation warnings in dev builds; log and track usage.
- Keep loaders/actions backward compatible when possible; add shims for new fields.

Change management

- Introduce additions as backwards-compatible fields.
- For breaking changes, dual-serve old/new versions during migration.
- Document migration paths in docs/api and update the book sections that reference APIs.

Testing

- Contract tests per version (MSW or mock server).
- Integration tests for dual-version periods.
- Playwright smoke tests to ensure old clients still function during rollout.

Observability

- Log API version per request; monitor error rates by version.
- Alert on deprecated version usage spikes to accelerate migration.

Checklist

- Clear versioning scheme (URL/header).
- Types generated per version; shims where needed.
- Deprecations communicated with timelines.
- Tests cover dual-version periods.

API Error Handling Patterns

Design APIs and clients to report and recover from errors consistently.

Error shapes

- Standardize on a JSON error shape, e.g.:

```
{  
  "error": {  
    "code": "INVALID_INPUT",  
    "message": "Email is required",  
    "details": { "field": "email" },  
    "requestId": "abc123"  
  }  
}
```

- Include machine-readable codes; keep messages user-friendly but safe.
- Add requestId to correlate with logs/traces.

Client handling (PhilJS)

- Map codes to user-facing messages and actions.
- For forms, map details.field to inline errors.
- For auth errors, redirect or show reauth flows.
- Log errors with route/loader/action context.

Transport considerations

- Use proper status codes (4xx/5xx).
- Avoid overloading 200 for errors.
- Include Retry-After for throttling/429.

Retries and backoff

- Only retry idempotent operations.
- Use exponential backoff with jitter.
- Cap retries; surface status to the user.

Testing

- Contract tests for error shapes.
- Integration tests for loader/action error handling.
- Playwright tests to assert UI renders helpful errors and preserves user input.

Checklist

- Standard JSON error shape with codes and requestId.
- Clients map errors to UI and logs.
- Proper HTTP status codes used.
- Retries with backoff for idempotent actions only.

API Contract Testing

Ensure your PhilJS clients and backend agree on request/response shapes.

Approaches

- **Mock servers:** MSW or local mock server that mirrors the contract.
- **Schema-driven:** generate types from OpenAPI/GraphQL, validate responses against schema.
- **Pact-style:** consumer-driven contracts for specific interactions.

Tooling

- openapi-typescript / graphql-codegen for types.
- MSW for local contract enforcement; fail tests on schema mismatch.
- Optional: Pact for service-to-service contracts if applicable.

Patterns

- Keep fixtures aligned with contracts; avoid ad-hoc JSON.
- Validate unknown JSON in loaders/resources before using in UI.
- For GraphQL, validate against schema and ensure fragments stay in sync.

Testing levels

- Unit: validate parser/mapper functions against fixtures.
- Integration: run loaders/actions with MSW serving contract-correct responses.
- E2E: smoke against staging with schema validation enabled (fail fast on drift).

Drift detection

- Log schema mismatch warnings in dev.
- Add CI checks that regenerate types and diff to catch breaking changes.
- Alert on production JSON validation failures (rate-limited).

Checklist

- Types generated and up to date.
- Loaders/resources validate responses.
- MSW/contract tests cover critical endpoints.
- Alerts/logs for schema drift in production.

Authentication and Authorization

Keep auth safe and predictable in PhilJS apps.

Authentication

- Use short-lived tokens; refresh securely.
- Store tokens in HttpOnly cookies when possible; avoid localStorage for secrets.
- For SSR/edge, read auth from headers/cookies and pass user context to loaders/actions.
- Redirect unauthenticated users early in loaders; avoid rendering sensitive pages client-only.

Authorization

- Enforce roles/permissions in loaders/actions; default deny.
- Tag caches per user/role to avoid data bleed.
- Avoid caching HTML for user-specific pages; cache data with scoped tags only.

Session handling

- Regenerate session identifiers after login.
- Set cookies: `Secure`, `HttpOnly`, `SameSite=Lax|Strict` as appropriate.
- Clear session on logout; invalidate related caches.

CSRF

- Prefer `SameSite` cookies; for state-changing requests, use CSRF tokens or double-submit cookie patterns.
- Validate origin/referer on POST/PUT/PATCH/DELETE where possible.

OAuth/OIDC

- Use PKCE for public clients.
- Store tokens server-side; send only necessary session info to the client.
- Handle token refresh server-side; avoid long-lived access tokens in the browser.

API access from loaders/actions

- Forward auth context explicitly; do not trust client-supplied headers.
- Strip/validate headers when proxying to downstream services.

Testing

- Unit: auth/permission guards with varied roles.
- Integration: loaders/actions with simulated cookies/headers.
- E2E: login/logout flows, session persistence, access control on restricted routes.

Checklist

- Auth validated at loader/action boundaries.
- Roles/permissions enforced; default deny.
- Tokens/cookies secured; refresh handled safely.
- CSRF protections in place for mutations.
- User-scoped caches; no HTML caching for private pages.

PhilJS CLI Reference

Use the PhilJS CLI to scaffold, build, and inspect projects quickly.

Commands

- `pnpm create philjs@latest my-app` — scaffold a new app.
- `philjs dev` — start Vite dev server with PhilJS plugins.
- `philjs build` — build client + server bundles with manifests for adapters.
- `philjs inspect` — print route tree, loaders/actions, bundle splits.
- `philjs test` — run configured tests (wrapper around workspace scripts).
- `philjs check` — run lint + typecheck + size budgets.

Options (common)

- `--ssr` — ensure SSR build outputs are generated.
- `--analyze` — bundle analysis (when supported in config).
- `--config` — custom config path.
- `--filter` — limit to specific packages/routes.

Workflow tips

- Run `philjs inspect --json > .out/routes.json` to snapshot route topology.
- Pair `philjs build` with platform adapters (Vercel/Netlify/CF) for previews.
- Add CLI commands to CI (check, build) to gate merges.

Troubleshooting

- If CLI isn't found, ensure `node_modules/.bin` is on PATH or use `pnpm philjs ...`.
- For Windows, prefer `pnpm philjs` to avoid PATH issues.

- Align CLI version with repo packages (0.1.0).

Links

- See [publishing/workflow.md](#) for export commands.
- See [devops/ci-cd.md](#) for CI integration.

Tooling and Dev Experience

Great DX keeps teams fast. PhilJS ships CLI scaffolding, a devtools extension, and build-time helpers so you can stay focused on product. Cross-reference [docs/tooling](#), [docs/guides](#), and [docs/packages/philjs-cli](#) for deeper command references; this chapter is the hands-on cheat sheet.

philjs-cli

- `pnpm create philjs@latest` – scaffold apps, SSR targets, and adapters.
- `philjs dev` – launches Vite dev server with PhilJS plugins prewired.
- `philjs build` – SSR + client bundles with manifest output for adapters.
- `philjs inspect` – prints route tree, loaders/actions, and bundle splits.
- `philjs test` – convenience wrapper for unit/integration suites.
- `philjs check` – runs lint/typecheck/budget scripts together.

DevTools extension

- **Signals graph:** visualize dependencies; spot over-rendering quickly.
- **Perf flamecharts:** record render/hydration timing.
- **Router panel:** inspect loaders/actions, cache state, and invalidations.
- **Accessibility checks:** ARIA annotations and contrast hints inline.

Install the extension locally, then enable the PhilJS tab in your browser devtools. For headless CI screenshots, use Playwright with the devtools inspector API.

Tips: - Run devtools against production builds in a local preview to spot hydration gaps. - Record timeline during route transitions and compare before/after code changes.

Builder and plugins

- Use `@philjs/builder` to assemble adapters (Vercel, Netlify, CF, Node, Bun, Deno).
- Plugins can add:
 - environment validation,
 - SVG/MDX handling,
 - image optimization pipelines,
 - perf budgets checks.
- Keep plugin order deterministic; prefer pure transforms over runtime shims.
- Reuse plugin recipes from [docs/deployment](#) and [docs/performance](#) for edge builds.

Migrate

`@philjs/migrate` helps move React/Solid/Svelte code:

- JSX runtime mapping (automatic).
- Signaling state (replace `useState/useEffect` with `signals/effects`).
- Router migration cheatsheets (React Router → PhilJS Router).
- Style migration hints (CSS Modules/Emotion/Tailwind) with minimal churn.

Linting and type safety

- ESLint config: `packages/philjs-eslint` and `eslint-config-philjs` enforce signals best practices (no async effects, dependency hygiene).
- TypeScript 6 baseline with strict settings; keep `tsconfig.base.json` synced across packages.
- Run `pnpm lint` and `pnpm typecheck` in CI to catch regressions early.
- Add `pnpm size` to CI for perf budgets.
- Use `tsc --noEmit` across packages before releases to catch drift.

Recommended workflow

1. Scaffold with CLI.
2. Run `philjs dev` and keep DevTools open to watch signals graphs.
3. Add budgets (`size-limit`) early and gate PRs.
4. Add Playwright smoke tests per route; wire to CI.
5. Use `philjs inspect` before shipping to confirm route data contracts.
6. Automate `philjs inspect + bundle snapshots` in CI for visibility.

Automation ideas

- Snapshot bundle sizes and route trees per PR; attach artifacts for review.
- Auto-run `philjs inspect` and diff results to catch unexpected route/config changes.

- Generate changelogs from route/config diffs to highlight potential breaking changes.

Try it now: inspect + devtools profiling loop

1. `philjs inspect --json > .out/routes.json` to snapshot your route tree and caches.
2. Run `philjs dev` and open DevTools → PhilJS tab → record a route transition.
3. Identify slow components; inline memoization or split routes; rerun `inspect` to ensure bundles are still under budget.

Package Atlas

A complete catalog of every package in the PhilJS monorepo. Each entry is derived from the package manifests and source so the book stays aligned with the codebase.

How to read this catalog

- Package names and versions come from `packages/*/package.json` or `packages/*/Cargo.toml`.
- Descriptions, keywords, and entry points mirror the manifests.
- Each package entry includes a generated API snapshot plus the package README content.
- Use the index below to jump directly to a package entry.

Package Index

- [@philjs/3d](#)
- [@philjs/3d-physics](#)
- [@philjs/a11y-ai](#)
- [@philjs/ab-testing](#)
- [@philjs/adapters](#)
- [@philjs/ai](#)
- [@philjs/ai-agents](#)
- [@philjs/ambient](#)
- [@philjs/analytics](#)
- [@philjs/api](#)
- [@philjs/atoms](#)
- [@philjs/auth](#)
- [@philjs/benchmark](#)
- [@philjs/biometric](#)
- [@philjs/build](#)
- [@philjs/builder](#)
- [@philjs/carbon](#)
- [@philjs/cdn](#)
- [@philjs/cells](#)
- [@philjs/charts](#)
- [@philjs/cli](#)
- [@philjs/collab](#)
- [@philjs/compiler](#)
- [@philjs/content](#)
- [@philjs/core](#)
- [@philjs/crossdevice](#)
- [@philjs/css](#)
- [@philjs/dashboard](#)
- [@philjs/db](#)
- [@philjs/desktop](#)
- [@philjs/devtools](#)
- [@philjs/devtools-extension](#)
- [@philjs/digital-twin](#)
- [@philjs/dnd](#)
- [@philjs/docs](#)
- [@philjs/edge](#)
- [@philjs/edge-ai](#)
- [@philjs/edge-mesh](#)
- [@philjs/editor](#)
- [@philjs/email](#)
- [@philjs/enterprise](#)
- [@philjs/errors](#)
- [@philjs/event-sourcing](#)
- [@philjs/export](#)
- [@philjs/eye-tracking](#)
- [@philjs/forms](#)

- [@philjs/genui](#)
- [@philjs/gesture](#)
- [@philjs/go](#)
- [@philjs/graphql](#)
- [@philjs/haptic](#)
- [@philjs/hollow](#)
- [@philjs/html](#)
- [@philjs/hypermedia](#)
- [@philjs/i18n](#)
- [@philjs/image](#)
- [@philjs/inspector](#)
- [@philjs/intent](#)
- [@philjs/islands](#)
- [@philjs/jobs](#)
- [@philjs/kotlin](#)
- [@philjs/liveview](#)
- [@philjs/lm-ui](#)
- [@philjs/maps](#)
- [@philjs/media-stream](#)
- [@philjs/meta](#)
- [@philjs/migrate](#)
- [@philjs/motion](#)
- [@philjs/native](#)
- [@philjs/neural](#)
- [@philjs/observability](#)
- [@philjs/offline](#)
- [@philjs/openapi](#)
- [@philjs/optimizer](#)
- [@philjs/payments](#)
- [@philjs/pdf](#)
- [@philjs/perf](#)
- [@philjs/perf-budget](#)
- [@philjs/playground](#)
- [@philjs/plugin-analytics](#)
- [@philjs/plugin-i18n](#)
- [@philjs/plugin-pwa](#)
- [@philjs/plugin-seo](#)
- [@philjs/plugin-tailwind](#)
- [@philjs/plugins](#)
- [@philjs/poem](#)
- [@philjs/pwa](#)
- [@philjs/python](#)
- [@philjs/qr](#)
- [@philjs/quantum](#)
- [@philjs/realtimel](#)
- [@philjs/resumable](#)
- [@philjs/rich-text](#)
- [@philjs/rocket](#)
- [@philjs/router](#)
- [@philjs/router-typesafe](#)
- [@philjs/rpc](#)
- [@philjs/runtime](#)
- [@philjs/scene](#)
- [@philjs/screen-share](#)
- [@philjs/security-scanner](#)
- [@philjs/spatial-audio](#)
- [@philjs/sqlite](#)
- [@philjs/ssr](#)
- [@philjs/storage](#)
- [@philjs/storybook](#)
- [@philjs/studio](#)
- [@philjs/styles](#)
- [@philjs/swift](#)
- [@philjs/table](#)
- [@philjs/tailwind](#)
- [@philjs/templates](#)

- [@philjs/testing](#)
- [@philjs/time-travel](#)
- [@philjs/trpc](#)
- [@philjs/ui](#)
- [@philjs/vector](#)
- [@philjs/vector-store](#)
- [@philjs/video-chat](#)
- [@philjs/virtual](#)
- [@philjs/voice](#)
- [@philjs/vscode](#)
- [@philjs/wasm](#)
- [@philjs/webgpu](#)
- [@philjs/webrtc](#)
- [@philjs/workers](#)
- [@philjs/workflow](#)
- [@philjs/xr](#)
- [@philjs/xstate](#)
- [@philjs/zig](#)
- [@philjs/zustand](#)
- [cargo-philjs](#)
- [create-philjs](#)
- [create-philjs-plugin](#)
- [eslint-config-philjs](#)
- [eslint-plugin-philjs](#)
- [philjs](#)
- [philjs-actix](#)
- [philjs-axum](#)
- [philjs-macros](#)
- [philjs-mobile](#)
- [philjs-seaorm](#)
- [philjs-sqlx](#)
- [philjs-tauri](#)
- [philjs-tokio](#)
- [philjs-tui](#)

Package Entries

`## @philjs/3d` - Type: Node package - Purpose: WebGL and game engine integrations for PhilJS - Three.js, Godot, Unreal, Unity - Version: 0.1.0 - Location: `packages/philjs-3d` - Entry points: `packages/philjs-3d/src/index.ts`, `packages/philjs-3d/src/webgl/index.ts`, `packages/philjs-3d/src/three/index.ts`, `packages/philjs-3d/src/godot/index.ts`, `packages/philjs-3d/src/unreal/index.ts`, `packages/philjs-3d/src/unity/index.ts` - Keywords: philjs, webgl, three, threejs, godot, unreal, unity, game-engine, 3d, graphics, pixel-streaming

PhilJS 3D

WebGL and game engine integrations for PhilJS. This package provides comprehensive 3D graphics and game engine support, making PhilJS competitive for game development and 3D web experiences.

Features

- **WebGL Integration** - Low-level WebGL with shaders, buffers, textures, and primitives
- **Three.js Integration** - Hooks and components for Three.js
- **Godot Integration** - Embed Godot HTML5 exports with bidirectional communication
- **Unreal Engine Integration** - Pixel Streaming support for UE5
- **Unity Integration** - Embed Unity WebGL builds with message passing

Installation

```
npm install philjs-3d
```

WebGL Integration

Full WebGL support with WebGL2 fallback:

```

import {
  createWebGLContext,
  createProgram,
  createCube,
  createCamera,
  createAnimationLoop,
  WebGLCanvas,
} from 'philjs-3d/webgl';

// Create WebGL context
const { gl, isWebGL2 } = createWebGLContext(canvas, {
  antialias: true,
  preferWebGL2: true,
});

// Create shader program
const program = createProgram(gl, {
  vertex: vertexShaderSource,
  fragment: fragmentShaderSource,
});

// Create geometry
const cube = createCube(1);

// Create camera
const camera = createCamera({
  position: [0, 2, 5],
  target: [0, 0, 0],
  fov: Math.PI / 4,
});

// Animation Loop
const loop = createAnimationLoop(({info}) => {
  // Render frame
});
loop.start();

```

WebGLCanvas Component

```

<WebGLCanvas
  width={800}
  height={600}
  onInit={(result) => {
    // Initialize scene
  }}
  onFrame={({info, gl}) => {
    // Render frame
  }}
/>

```

Three.js Integration

Seamless Three.js integration with PhilJS hooks:

```

import {
  useThree,
  useFrame,
  loadThree,
  ThreeCanvas,
} from 'philjs-3d/three';

// Load Three.js
const THREE = await loadThree();

// Create scene
const state = await initThree(canvas, {
  antialias: true,
  shadows: true,
  camera: {
    fov: 75,
    position: [0, 2, 5],
  },
});

// Add objects
const geometry = new THREE.BoxGeometry(1, 1, 1);
const material = new THREE.MeshStandardMaterial({ color: 0x00ff00 });
const cube = new THREE.Mesh(geometry, material);
state.scene.add(cube);

// Animation
useFrame(canvas, ({ time, delta }) => {
  cube.rotation.x += delta;
  cube.rotation.y += delta;
});

```

ThreeCanvas Component

```

<ThreeCanvas
  width={800}
  height={600}
  camera={{ position: [0, 2, 5] }}
  onCreated={({state) => {
    // Setup scene
  })}
  onFrame={({state, delta) => {
    // Animate
  }})
/>

```

Godot Integration

Embed Godot HTML5 exports with signal communication:

```

import {
  GodotEmbed,
  useGodot,
  callGodot,
  onGodotSignal,
} from 'philjs-3d/godot';

// Use Godot hook
const { godot, isLoading, callGodot, onGodotSignal } = useGodot(canvas);

// Call Godot methods
callGodot('/root/Player', 'take_damage', 10);

// Listen to Godot signals
const cleanup = onGodotSignal('/root/Player', 'health_changed', (health) => {
  console.log('Health:', health);
});

```

GodotEmbed Component

```

<GodotEmbed
  pckPath="/game/game.pck"
  width={1280}
  height={720}
  onReady={({godot) => {
    // Game ready
  })}
  onProgress={({current, total) => {
    // Loading progress
  }})
/>

```

Unreal Engine Integration

Pixel Streaming for Unreal Engine 5:

```

import {
  UnrealEmbed,
  useUnreal,
  setupInputForwarding,
} from 'philjs-3d/unreal';

// Use Unreal hook
const { unreal, isConnected, stats, executeCommand } = useUnreal(video);

// Execute console commands
executeCommand('r.ScreenPercentage 100');

// Send custom messages
unreal.sendMessage('CustomEvent', { data: 'value' });

// Listen to events
unreal.on('message', (data) => {
  console.log('From UE:', data);
});

```

UnrealEmbed Component

```

<UnrealEmbed
  serverUrl="wss://your-signaling-server.com"
  width={1920}
  height={1080}
  enableInput={true}
  showControls={true}
  onConnect={({instance) => {
    // Connected
  }})
  onStats={({stats) => {
    // FPS, latency, etc.
  }})
/>

```

Unity Integration

Embed Unity WebGL builds:

```
import {
  UnityEmbed,
  useUnity,
  sendMessage,
  registerUnityCallback,
} from 'philjs-3d/unity';

// Use Unity hook
const { unity, isReady, progress, sendMessage } = useUnity(canvas);

// Send messages to Unity
sendMessage('Player', 'TakeDamage', 10);

// Register callbacks for Unity to call
registerUnityCallback('OnScoreUpdate', (score) => {
  console.log('Score:', score);
});

// Signal bridges
createUnitySignalBridge('OnHealthChange', setHealth);
createPhilJSsignalBridge(canvas, 'GameManager', 'SetDifficulty', getDifficulty);
```

UnityEmbed Component

```
<UnityEmbed
  buildUrl="/unity-build/Build"
  width={960}
  height={600}
  showProgress={true}
  onReady={(unity) => {
    // Unity ready
  }}
  onProgress={(progress) => {
    // Loading progress
  }}
/>
```

API Reference

WebGL

- `createWebGLContext(canvas, options)` - Initialize WebGL context
- `createProgram(gl, source)` - Create shader program
- `createCube/Sphere/Plane/Cylinder/Torus(size)` - Create primitives
- `createCamera(options)` - Create camera
- `createAnimationLoop(callback)` - Create animation loop
- `useWebGL(canvas, options)` - Hook for WebGL

Three.js

- `loadThree()` - Load Three.js dynamically
- `initThree(canvas, options)` - Initialize Three.js
- `useThree(canvas)` - Get Three.js state
- `useFrame(canvas, callback)` - Animation frame hook
- `useLoader(loader, url)` - Asset loading hook

Godot

- `createGodotInstance(canvas, props)` - Create Godot instance
- `useGodot(canvas)` - Hook for Godot
- `callGodot(canvas, nodePath, method, args)` - Call Godot method
- `onGodotSignal(canvas, nodePath, signal, callback)` - Listen to signal
- `createGodotBridge(canvas, nodePath, options)` - Bidirectional sync

Unreal

- `createPixelStreamingInstance(video, props)` - Create Pixel Streaming
- `useUnreal(video)` - Hook for Unreal
- `setupInputForwarding(video, instance)` - Forward inputs

Unity

- `createUnityInstance(canvas, props)` - Create Unity instance
- `useUnity(canvas)` - Hook for Unity
- `sendMessage(canvas, gameObject, method, param)` - Send message
- `registerUnityCallback(name, handler)` - Register callback
- `createUnitySignalBridge(name, setValue)` - Unity to PhilJS
- `createPhilJSsignalBridge(canvas, gameObject, method, getValue)` - PhilJS to Unity

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: ., ./webgl, ./three, ./godot, ./unreal, ./unity
- Source files: packages/philijs-3d/src/index.ts, packages/philijs-3d/src/webgl/index.ts, packages/philijs-3d/src/three/index.ts, packages/philijs-3d/src/godot/index.ts, packages/philijs-3d/src/unreal/index.ts, packages/philijs-3d/src/unity/index.ts

Public API

- Direct exports: (none detected)
- Re-exported names: // Animation createAnimationLoop, // Assets loadBevyAsset, // Buffers createBuffer, // Camera and Math mat4Identity, // Components BevyEmbed, // Components GodotEmbed, // Components ThreeCanvas, // Components UnityEmbed, // Components UnrealEmbed, // Components WebGLCanvas, // Context createWebGLContext, // ECS Bridge setSignalCreator, // Hooks createBevyInstance, // Hooks createGodotInstance, // Hooks createPixelStreamingInstance, // Hooks createUnityInstance, // Hooks loadThree, // Hooks useWebGL, // Primitives createCube, // Shaders compileShader, // Textures createTextureFromImage, // Types type BevyConfig, // Types type GodotEngine, // Types type PixelStreamingConfig, // Types type ThreeModule, // Types type UnityInstance, // Types type WebGLContextOptions, AnimationFrameInfo, AnimationLoop, AssetBundle, AssetHandle, AssetMetadata, AssetState, BASIC_FRAGMENT_SHADER, BASIC_VERTEX_SHADER, BevyApp, BevyAssetType, BevyComponent, BevyEmbedProps, BevyEntity, BevyEvent, BevyEventData, BevyEventListener, BevyEventType, BevyFPSCounter, BevyFullscreenButton, BevyGamepadState, BevyInstance, BevyMouseButton, BevyPauseButton, BevyQuery, BevyResource, BevyState, BevyWorld, BufferInfo, Camera, CameraOptions, ChildrenComponent, ComponentBridge, ComponentType, ConsoleCommandOptions, Easing, EntityBridge, EntityGeneration, EntityId, FrameCallback, FrameInfo, GamepadAxis, GamepadButton, GamepadInputData, GlobalTransformComponent, GodotConfig, GodotEmbed, GodotEmbedProps, GodotEngine, GodotInstance, GodotInterface, GodotLoadingIndicator, GodotState, InputResource, KeyCode, KeyboardInputData, LoaderResult, Mat3, Mat4, MouseInputData, NameComponent, ParentComponent, PixelStreamingConfig, PixelStreamingInputEvent, PixelStreamingInstance, PrimitiveGeometry, Quat, QueryFilter, QueryResult, ResourceType, ShaderProgram, ShaderSource, SignalHandler, TEXTURED_FRAGMENT_SHADER, TEXTURED_VERTEX_SHADER, TextureInfo, TextureOptions, ThreeCamera, ThreeCanvas, ThreeCanvasProps, ThreeClock, ThreeColor, ThreeEuler, ThreeGLTF, ThreeGLTFLoader, ThreeModule, ThreeObject3D, ThreeOrthographicCamera, ThreePerspectiveCamera, ThreeQuaternion, ThreeRenderer, ThreeRendererOptions, ThreeScene, ThreeState, ThreeTexture, ThreeTextureLoader, ThreeVector3, TimeResource, TouchInputData, Transform, TransformComponent, UNLIT_FRAGMENT_SHADER, UNLIT_VERTEX_SHADER, UnityCallback, UnityConfig, UnityEmbed, UnityEmbedProps, UnityEventHandler, UnityEventType, UnityFullscreenButton, UnityInstance, UnityInstanceWrapper, UnityLoadingProgress, UnityMessage, UnityModule, UnityProgressBar, UnityState, UnrealCustomEvent, UnrealEmbed, UnrealEmbedProps, UnrealState, UnrealStatsOverlay, UseBevyEventResult, UseBevyQueryResult, UseBevyResourceResult, UseBevyResult, UseGodotResult, UseUnityResult, UseUnrealResult, Vec2, Vec3, Vec4, VertexArrayInfo, VisibilityComponent, WebGLCanvas, WebGLCanvasProps, WebGLContextOptions, WebGLContextResult, WebGLExtensions, WebGLHookContext, WebGLState, WebRTCStats, WindowResource, addToScene, bindTexture, bindVertexArray, callGodot, cleanupWebGL, clearAsset, clearAssetCache, clearContext, compileShader, createAnimationLoop, createAnimator, createBevyEmbedElement, createBuffer, createBufferInfo, createCamera, createComponentBridge, createCone, createCube, createCubemapTexture, createCustomComponent, createCylinder, createDataTexture, createEntityBridge, createFixedTimestepLoop, createGodotBridge, createGodotEmbedElement, createGodotInstance, createNameComponent, createPhilJSsignalBridge, createPixelStreamingInstance, createPlaceholderTexture, createPlane, createProgram, createRoundedBox, createSphere, createTextureFromImage, createThreeCanvasElement, createTorus, createTransformComponent, createUnityEmbedElement, createUnityInstance, createUnitySignalBridge, createUnrealEmbedElement, createVertexArray, createVertexArrayInfo, createVisibilityComponent, createWebGLCanvasElement, createWebGLContext, defineAssetBundle, deleteBuffer, deleteProgram, deleteTexture, deleteVertexArray, deleteVertexArrayInfo, despawnEntity, disposeAllBevy, disposeAllBridges, disposeBevy, disposeGodot, disposeThree, disposeUnity, disposeUnreal, drawVertexArray, enableDefaultFeatures, findEntitiesWith, findEntityWith, getAllBevyInstances, getAssetBundle, getAssetMetadata, getBevy, getCacheCount, getCacheSize, getCachedAsset, getCachedAssetPaths, getLoadingProgress, getThree, getViewProjectionMatrix, getWebGLCapabilities, initThree, insertComponent, isAssetCached, isAssetLoaded, isBevySupported, isBundleLoaded, isWebGL2Supported, isWebGLSupported, lerp, lerpVec3, loadAssetBundle, loadGLTFAsync, loadTexture, loadTextureAsync, loadThree, mat3FromMat4, mat3InvertTranspose, mat4Identity, mat4Invert, mat4LookAt, mat4Multiply, mat4Orthographic, mat4Perspective, mat4RotateX, mat4RotateY, mat4Scale, mat4Translate, mat4Transpose, mergeGeometries, onBevyEvent, onGodotSignal, onUnityEvent, orbitCamera, preloadAssets, preloadAssetsWithPriority, queryEntities, registerUnityCallback, removeComponent, removeFrameCallback, removeFromScene, resizeCanvas, resizeThree, sendBevyEvent, sendMessage, setCameraAspect, setCameraLookAt, setCameraPosition, setCameraTarget, setThreeCameraPosition, setUniform, setUniforms, setupInputForwarding, setupVertexAttributes, slerp, spawnEntity, startAnimationLoop, streamAsset, syncFromGodot, syncToGodot, trackEntities, trackEntity, transformGeometry, unbindTexture, unloadAssetBundle, updateBuffer, updateCameraProjection, updateCameraView, updateTexture, useActiveProgram, useAnimationFrame, useAutoResize, useBevy, useBevyEntity, useBevyQuery, useBevyResource, useCamera, useFrame, useGodot, useLoader, useProgram, useRenderPass, useShaderProgram, useThree, useUniforms, useUnity, useUnreal, useWebGL, watchAsset, zoomCamera
- Re-exported modules: ./GodotEmbed.js, ./ThreeCanvas.js, ./UnityEmbed.js, ./UnrealEmbed.js, ./WebGLCanvas.js, ./animation.js, ./bevy/index.js, ./buffers.js, ./camera.js, ./context.js, ./godot/index.js, ./hooks.js, ./primitives.js, ./shaders.js, ./textures.js, ./three/index.js, ./types.js, ./unity/index.js, ./unreal/index.js, ./webgl/index.js

License

MIT

@philijs/3d-physics - Type: Node package - Purpose: High-performance 3D physics engine for PhilJS - rigid bodies, vehicles, ragdolls, character controllers - Version: 0.1.0 - Location: packages/philijs-3d-physics - Entry points: packages/philijs-3d-physics/src/index.ts - Keywords: philijs, physics, 3d, rapier, rigid-body, vehicle, ragdoll, game

@philijs/3d-physics

High-performance 3D physics engine for PhilJS applications with multiple backend support.

Features

- Multiple physics backends (Rapier, Cannon.js, Ammo.js)
- Rigid body dynamics with continuous collision detection
- Soft body simulation (cloth, rope, deformables)
- Joints and constraints (hinge, ball, slider, spring)
- Vehicle physics (car, tank, hover)
- Character controller with ground detection
- Ragdoll physics
- Deterministic physics for multiplayer

Installation

```
npm install @philijs/3d-physics
```

Usage

Basic Physics World

```

import { PhysicsWorld, usePhysicsWorld, useRigidBody } from '@philjs/3d-physics';

// Create a physics world with Rapier backend
const world = await PhysicsWorld.create({
  backend: 'rapier',
  gravity: { x: 0, y: -9.81, z: 0 },
  enableCCD: true,
  enableSleeping: true
});

// Add a dynamic rigid body
world.createRigidBody('ball', {
  type: 'dynamic',
  mass: 1,
  friction: 0.5,
  restitution: 0.7
}, { x: 0, y: 10, z: 0 }, { x: 0, y: 0, z: 0, w: 1 });

world.addCollider('ball', {
  shape: 'sphere',
  radius: 0.5
});

// Step the simulation
world.step(1/60);

```

Character Controller

```

import { useCharacterController } from '@philjs/3d-physics';

const controller = useCharacterController(world, 'player', {
  height: 1.8,
  radius: 0.3,
  stepHeight: 0.35,
  slopeLimit: 45,
  speed: 5,
  jumpForce: 8
});

// Move the character
controller.move({ x: 1, y: 0, z: 0 });
controller.jump();

// Check ground state
if (controller.isGrounded()) {
  // Character is on the ground
}

```

Vehicle Physics

```

import { useVehicle } from '@philjs/3d-physics';

const vehicle = useVehicle(world, 'car', {
  chassis: {
    mass: 1500,
    size: { x: 2, y: 0.5, z: 4 }
  },
  wheels: [
    { position: { x: -0.8, y: -0.3, z: 1.2 }, radius: 0.4, isSteering: true },
    { position: { x: 0.8, y: -0.3, z: 1.2 }, radius: 0.4, isSteering: true },
    { position: { x: -0.8, y: -0.3, z: -1.2 }, radius: 0.4, isDrive: true },
    { position: { x: 0.8, y: -0.3, z: -1.2 }, radius: 0.4, isDrive: true }
  ],
  maxSpeed: 50,
  acceleration: 20
});

// Control the vehicle
vehicle.accelerate(1.0); // 0-1 throttle
vehicle.steer(-0.5); // -1 to 1 steering
vehicle.brake(0.8); // 0-1 brake force

```

Ragdoll Physics

```

import { useRagdoll } from '@philjs/3d-physics';

const ragdoll = useRagdoll(world, 'enemy', { x: 0, y: 5, z: 0 }, {
  scale: 1.0,
  limbMass: 2
});

// Get bone positions for rendering
const headPos = ragdoll.getBonePosition('head');
const headRot = ragdoll.getBoneRotation('head');

// Apply impact force
ragdoll.applyImpulse('chest', { x: 100, y: 50, z: 0 });

```

Raycasting

```
// Cast a ray
const hit = world.raycast(
  { x: 0, y: 10, z: 0 }, // origin
  { x: 0, y: -1, z: 0 }, // direction
  100 // max distance
);

if (hit.hit) {
  console.log('Hit body:', hit.bodyId);
  console.log('Hit point:', hit.point);
  console.log('Distance:', hit.distance);
}
```

API Reference

PhysicsWorld

Method	Description
create(config)	Create a new physics world
step(dt)	Step the simulation
createRigidBody(id, config, position, rotation)	Create a rigid body
removeRigidBody(id)	Remove a rigid body
addCollider(bodyId, config)	Add a collider to a body
createJoint(id, config)	Create a joint between bodies
raycast(origin, direction, maxDistance)	Cast a ray
getPosition(bodyId)	Get body position
applyForce(bodyId, force)	Apply force to a body
applyImpulse(bodyId, impulse)	Apply impulse to a body

Hooks

Hook	Description
usePhysicsWorld(config)	Create and manage a physics world
useRigidBody(world, id, config)	Create and manage a rigid body
useCharacterController(world, id, config)	Character controller with movement
useVehicle(world, id, config)	Vehicle physics controller
useRagdoll(world, id, position, config)	Ragdoll physics system

Types

```
interface PhysicsConfig {
  backend: 'rapier' | 'cannon' | 'ammo';
  gravity?: Vector3;
  timestep?: number;
  substeps?: number;
  enableCCD?: boolean;
  enableSleeping?: boolean;
  deterministic?: boolean;
}

interface RigidBodyConfig {
  type: 'dynamic' | 'static' | 'kinematic';
  mass?: number;
  friction?: number;
  restitution?: number;
  linearDamping?: number;
  angularDamping?: number;
}

interface ColliderConfig {
  shape: 'box' | 'sphere' | 'capsule' | 'cylinder' | 'convex' | 'trimesh';
  size?: Vector3;
  radius?: number;
  height?: number;
  isSensor?: boolean;
}
```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philijs-3d-physics/src/index.ts

Public API

- Direct exports: // Core classes PhysicsWorld, // Hooks usePhysicsWorld, // Types type Vector3, AmmoBackend, CannonBackend, CharacterConfig, CharacterController, ColliderConfig, ContactEvent, JointConfig, PhysicsBackend, PhysicsConfig, Quaternion, RigidBody, RigidBodiesConfig, Ragdoll, RagdollConfig, RapierBackend, RaycastResult, RigidBodiesConfig, VehicleConfig, VehicleController, WheelConfig, useCharacterController, useRagdoll, useRigidBodies, useVehicle
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/a11y-ai - Type: Node package - Purpose: AI-powered accessibility for PhilJS - automatic alt text, ARIA labels, and contrast fixes - Version: 0.1.0 - Location: packages/philjs-a11y-ai - Entry points: packages/philjs-a11y-ai/src/index.ts - Keywords: philjs, accessibility, a11y, ai, wcag, aria, alt-text, contrast

@philjs/a11y-ai

AI-powered accessibility toolkit that automatically detects and fixes WCAG compliance issues.

Features

- Auto-generate alt text for images using AI vision
- Fix color contrast issues automatically
- Add missing ARIA labels intelligently
- Generate accessible descriptions
- Keyboard navigation optimization
- Screen reader optimization
- Focus management
- WCAG A/AA/AAA compliance checking

Installation

```
npm install @philjs/a11y-ai
```

Usage

Basic Audit

```
import { A11yAI, initA11yAI } from '@philjs/a11y-ai';

// Initialize with AI provider
const a11y = new A11yAI({
  provider: 'openai',
  apiKey: 'your-api-key',
  wcagLevel: 'AA',
  autoFix: true
});

// Run accessibility audit
const report = await a11y.audit(document.body);

console.log('Score:', report.score);
console.log('Issues found:', report.summary.totalIssues);
console.log('Auto-fixed:', report.summary.fixed);
```

AI-Generated Alt Text

```
import { useAutoAltText } from '@philjs/a11y-ai';

// Generate alt text for an image
const altText = await useAutoAltText(
  'https://example.com/image.jpg',
  'Product photo on e-commerce page'
);

console.log(altText.text);           // "Red Leather handbag with gold buckle"
console.log(altText.confidence);    // 0.92
```

Global Initialization

```
import { initA11yAI, useA11yAudit } from '@philjs/a11y-ai';

// Initialize once
initA11yAI({
  provider: 'anthropic',
  apiKey: process.env.ANTHROPIC_KEY,
  wcagLevel: 'AAA',
  autoFix: true,
  languages: ['en', 'es']
});

// Use anywhere in your app
const report = await useA11yAudit();
```

Color Contrast Fixes

```

import { getContrastRatio, adjustColorForContrast } from '@philjs/a11y-ai';

// Check contrast ratio
const ratio = getContrastRatio('#666666', '#ffffff');
console.log(ratio); // 5.74

// Auto-fix Low contrast
const fixedColor = adjustColorForContrast(
  '#999999', // foreground
  '#ffffff', // background
  4.5 // target ratio (WCAG AA)
);
console.log(fixedColor); // "rgb(118, 118, 118)"

```

Working with Reports

```

const report = await a11y.audit();

// Iterate through issues
for (const issue of report.issues) {
  console.log(` [${issue.severity}] ${issue.description}`);
  console.log(`  WCAG: ${issue.wcagCriteria.join(', ')}`);
  console.log(`  Element: ${issue.selector}`);
  console.log(`  Auto-fixable: ${issue.autoFixable}`);
}

// Check issues by type
console.log('Missing alt texts:', report.summary.byType['missing-alt']);
console.log('Low contrast:', report.summary.byType['low-contrast']);

```

API Reference

A11yAI Class

Method	Description
audit(root?)	Run full accessibility audit
autoFixAll()	Auto-fix all fixable issues
fixIssue(issue)	Fix a specific issue

Hooks

Function	Description
initA11yAI(config)	Initialize global instance
getA11yAI()	Get global instance
useA11yAudit(root?)	Run audit with hook pattern
useAutoAltText(url, context?)	Generate alt text for image

Utility Functions

Function	Description
getContrastRatio(color1, color2)	Calculate WCAG contrast ratio
adjustColorForContrast(fg, bg, target)	Adjust color for target ratio

Configuration

```

interface A11yAIConfig {
  provider?: 'openai' | 'anthropic' | 'local';
  apiKey?: string;
  visionModel?: string;
  textModel?: string;
  wcagLevel?: 'A' | 'AA' | 'AAA';
  autoFix?: boolean;
  languages?: string[];
}

```

Issue Types

Type	WCAG	Severity
missing-alt	1.1.1	Critical
empty-alt	1.1.1	Moderate
low-contrast	1.4.3	Serious
missing-label	1.3.1, 4.1.2	Critical
empty-button	4.1.2	Critical
empty-link	2.4.4	Critical
missing-lang	3.1.1	Serious
skip-heading-level	1.3.1	Moderate
missing-focus-indicator	2.4.7	Serious
auto-playing-media	1.4.2	Serious

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philijs-a11y-ai/src/index.ts

Public API

- Direct exports: A11yAI, A11yAIConfig, A11yFix, A11yIssue, A11yIssueType, A11yReport, A11ySummary, AltTextGenerator, AriaLabelGenerator, ColorContrastFix, GeneratedAltText, adjustColorForContrast, getA11yAI, getContrastRatio, initA11yAI, useA11yAudit, useAutoAltText
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

```
## @philijs/ab-testing - Type: Node package - Purpose: Native A/B testing framework for PhilJS - experiments, feature flags, statistical analysis - Version: 0.1.0 - Location: packages/philijs-ab-testing - Entry points: packages/philijs-ab-testing/src/index.ts - Keywords: philjs, ab-testing, experiments, feature-flags, statistical-analysis
```

@philijs/ab-testing

Native A/B testing and feature flag framework for PhilJS with built-in statistical analysis.

Features

- Experiment configuration and variant assignment
- Statistical analysis (z-test, t-test, confidence intervals)
- Feature flags with rollout percentages
- User segmentation and audience targeting
- Event tracking with automatic batching
- Multi-armed bandit support
- Persistent assignments across sessions

Installation

```
npm install @philijs/ab-testing
```

Usage

Basic A/B Test

```

import { ABTestingManager, useExperiment } from '@philjs/ab-testing';

// Initialize the manager
const ab = new ABTestingManager('user-123', {
  trackingEndpoint: '/api/analytics',
  autoTrack: true
});

// Register experiments
ab.registerExperiments([
  {
    id: 'checkout-redesign',
    name: 'Checkout Page Redesign',
    status: 'running',
    allocation: 50,
    variants: [
      { id: 'control', name: 'Current Design', weight: 50, isControl: true },
      { id: 'variant-a', name: 'New Design', weight: 50 }
    ]
  }
]);

// Get variant for user
const variant = ab.getVariant('checkout-redesign');
if (variant?.id === 'variant-a') {
  // Show new design
}

// Track conversion
ab.trackConversion('checkout-redesign', 99.99);

```

Using Hooks

```

import { useABTesting, useExperiment, useFeatureFlag } from '@philjs/ab-testing';

// Initialize once
const { registerExperiments, getVariant } = useABTesting('user-123');

// Use in components
function CheckoutButton() {
  const { variant, isControl, trackConversion } = useExperiment('button-color');

  const handleClick = () => {
    trackConversion();
    // proceed with checkout
  };

  return (
    <button
      style={{ background: variant?.config?.color || 'blue' }}
      onClick={handleClick}
    >
      Checkout
    </button>
  );
}

```

Feature Flags

```

import { FeatureFlagManager } from '@philjs/ab-testing';

const flags = new FeatureFlagManager('user-123');

flags.registerFlags([
  {
    id: 'dark-mode',
    name: 'Dark Mode',
    enabled: true,
    rolloutPercentage: 25,
    targetAudience: [
      { attribute: 'plan', operator: 'equals', value: 'premium' }
    ]
  }
]);

if (flags.isEnabled('dark-mode')) {
  enableDarkMode();
}

// Feature variants
const config = flags.getVariant<{ theme: string }>('dark-mode');

```

Statistical Analysis

```

import { StatisticalAnalyzer } from '@philjs/ab-testing';

const analyzer = new StatisticalAnalyzer();

// Z-test for conversion rates
const result = analyzer.zTest(
  { conversions: 150, total: 1000 }, // control
  { conversions: 180, total: 1000 } // variant
);

console.log('Z-score:', result.zScore);
console.log('P-value:', result.pValue);
console.log('Significant:', result.significant);

// Calculate required sample size
const sampleSize = analyzer.calculateSampleSize(
  0.10, // baseline conversion rate (10%)
  0.15, // minimum detectable effect (15% lift)
  0.80, // power
  0.05 // significance level
);
console.log('Required sample size per variant:', sampleSize);

```

Audience Targeting

```

ab.registerExperiments([
  {
    id: 'premium-feature',
    name: 'Premium Feature Test',
    status: 'running',
    allocation: 100,
    targetAudience: [
      { attribute: 'plan', operator: 'in', value: ['pro', 'enterprise'] },
      { attribute: 'signupDate', operator: 'lt', value: '2024-01-01' }
    ],
    variants: [
      { id: 'control', name: 'Control', weight: 50, isControl: true },
      { id: 'treatment', name: 'Treatment', weight: 50 }
    ]
  }
]);

// User context for targeting
ab.setUserContext({
  userId: 'user-123',
  attributes: {
    plan: 'pro',
    signupDate: '2023-06-15'
  }
});

```

API Reference

ABTestingManager

Method	Description
registerExperiments(experiments)	Register experiment configurations
registerFlags(flags)	Register feature flags
getVariant(experimentId)	Get assigned variant for user
isInVariant(experimentId, variantId)	Check if user is in specific variant
isFeatureEnabled(flagId)	Check if feature flag is enabled
trackConversion(experimentId, value?)	Track conversion event
trackEvent(experimentId, eventName, value?)	Track custom event
analyzeExperiment(id, control, variant)	Analyze experiment results
calculateRequiredSampleSize(baseline, mde)	Calculate required sample size

Hooks

Hook	Description
useABTesting(userId, config?)	Initialize A/B testing
useExperiment(experimentId)	Get experiment variant and tracking
useFeatureFlag(flagId)	Get feature flag state

Types

```

interface Experiment {
  id: string;
  name: string;
  status: 'draft' | 'running' | 'paused' | 'completed';
  allocation: number; // 0-100
  variants: Variant[];
  targetAudience?: AudienceRule[];
  primaryMetric?: string;
}

interface Variant {
  id: string;
  name: string;
  weight: number;
  isControl?: boolean;
  config?: Record<string, any>;
}

interface FeatureFlag {
  id: string;
  name: string;
  enabled: boolean;
  rolloutPercentage?: number;
  targetAudience?: AudienceRule[];
}

```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-ab-testing/src/index.ts

Public API

- Direct exports: ABTestingConfig, ABTestingManager, Assignment, AssignmentEngine, AudienceRule, EventTracker, Experiment, ExperimentEvent, ExperimentResults, FeatureFlag, FeatureFlagManager, MetricResults, StatisticalAnalyzer, UserContext, Variant, VariantResults, useABTesting, useExperiment, useFeatureFlag
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/adapters - Type: Node package - Purpose: Deployment adapters for PhilJS - Vercel, Netlify, Cloudflare, AWS, Bun, Deno, and more - Version: 0.1.0 - Location: packages/philjs-adapters - Entry points: packages/philjs-adapters/src/index.ts, packages/philjs-adapters/src/vercel/index.ts, packages/philjs-adapters/src/netlify/index.ts, packages/philjs-adapters/src/cloudflare/index.ts, packages/philjs-adapters/src/aws/index.ts, packages/philjs-adapters/src/node/index.ts, packages/philjs-adapters/src/static/index.ts, packages/philjs-adapters/src/bun/index.ts, packages/philjs-adapters/src/deno/index.ts, packages/philjs-adapters/src/runtime-detect.ts, packages/philjs-adapters/src/cloudflare-pages/index.ts, packages/philjs-adapters/src/vercel/adapter.ts, packages/philjs-adapters/src/netlify/adapter.ts, packages/philjs-adapters/src/aws-lambda/index.ts, packages/philjs-adapters/src/railway/index.ts, packages/philjs-adapters/src/edge/index.ts, packages/philjs-adapters/src/adapters/vercel.ts, packages/philjs-adapters/src/adapters/cloudflare.ts, packages/philjs-adapters/src/adapters/deno-deploy.ts, packages/philjs-adapters/src/adapters/netlify.ts, packages/philjs-adapters/src/adapters/aws-lambda.ts, packages/philjs-adapters/src/adapters/node.ts, packages/philjs-adapters/src/adapters/bun.ts, packages/philjs-adapters/src/utils/build.ts, packages/philjs-adapters/src/utils/env.ts - Keywords: philjs, adapter, vercel, netlify, cloudflare, aws, lambda, bun, deno, deployment, edge, serverless, workers, pages, kv, d1, durable-objects - Book coverage: [adapters/platforms.md](#)

philjs-adapters

Deployment adapters for PhilJS - Deploy anywhere: Vercel, Netlify, Cloudflare, AWS, Bun, Deno, or any Node.js server.

Requirements

- **Node.js 24** or higher
- **TypeScript 6** or higher
- **ESM only** - CommonJS is not supported

Features

- **Vercel** - Deploy to Vercel with Edge Functions and ISR
- **Netlify** - Deploy to Netlify with Edge Functions
- **Cloudflare Workers** - Deploy to Cloudflare Pages and Workers
- **AWS Lambda** - Deploy to AWS with Lambda and API Gateway
- **Node.js** - Deploy to any Node.js server (Express, Fastify, etc.)
- **Bun** - Native Bun.serve() with SQLite and WebSocket support
- **Deno** - Deno.serve() with KV storage and Deno Deploy ready
- **Static** - Generate static sites (SSG)
- **SSR Support** - Full server-side rendering on all platforms
- **Edge Functions** - Deploy to the edge for low latency
- **Automatic Configuration** - Zero-config deployment
- **Runtime Detection** - Automatically detect and use the best runtime

Installation

```
pnpm add philjs-adapters
```

Adapters

Cloudflare Pages (Enhanced)

Deploy to Cloudflare Pages with full support for KV, D1, R2, Durable Objects, and more.

Features

- KV Namespace storage
- D1 SQLite databases
- R2 object storage
- Durable Objects
- Queue bindings
- Analytics Engine
- Service bindings

Setup

```
import { cloudflarePagesAdapter } from 'philjs-adapters/cloudflare-pages';

export default cloudflarePagesAdapter({
  kv: [
    { binding: 'CACHE', id: 'your-kv-id' }
  ],
  d1: [
    { binding: 'DB', database_id: 'your-db-id', database_name: 'production' }
  ],
  r2: [
    { binding: 'UPLOADS', bucket_name: 'my-uploads' }
  ]
});
```

See [Cloudflare Pages documentation](#) for details.

Vercel (Enhanced)

Deploy to Vercel with Edge Functions, Serverless, ISR, KV, and Blob storage.

Features

- Edge Runtime
- Serverless Functions
- ISR (Incremental Static Regeneration)
- Vercel KV (Redis)
- Vercel Blob
- Edge Config
- Image Optimization
- Cron Jobs

Setup

```
import { vercelAdapter } from 'philjs-adapters/vercel/adapter';

export default vercelAdapter({
  edge: true,
  isr: { expiration: 60 },
  kv: { database: 'production' },
  blob: true,
  images: {
    domains: ['example.com'],
    formats: ['image/avif', 'image/webp']
  }
});
```

See [Vercel documentation](#) for details.

Netlify (Enhanced)

Deploy to Netlify with Edge Functions, Serverless Functions, and Blob storage.

Features

- Edge Functions
- Netlify Functions
- Blob Storage
- Form Handling
- Redirects & Rewrites
- Split Testing
- Image CDN

Setup

```

import { netlifyAdapter } from 'philjs-adapters/netlify/adapter';

export default netlifyAdapter({
  edge: true,
  blob: true,
  redirects: [
    { from: '/old', to: '/new', status: 301 }
  ],
  headers: [
    { for: '*', values: { 'X-Frame-Options': 'DENY' } }
  ]
});

```

See [Netlify documentation](#) for details.

AWS Lambda

Deploy to AWS Lambda with API Gateway, CloudFront, and S3 support.

Features

- Lambda Functions
- API Gateway (REST & HTTP)
- CloudFront CDN
- S3 Static Assets
- Lambda@Edge
- ALB Support
- SAM/Serverless/Terraform templates

Setup

```

import { awsLambdaAdapter } from 'philjs-adapters/aws-lambda';

export default awsLambdaAdapter({
  region: 'us-east-1',
  runtime: 'nodejs20.x',
  integration: 'http-api',
  s3: {
    bucket: 'my-static-assets'
  },
  generateSAM: true
});

```

See [AWS Lambda documentation](#) for details.

Railway

Deploy to Railway with Docker and Nixpacks support.

Features

- Docker configuration
- Nixpacks support
- Railway.toml generation
- Health checks
- Graceful shutdown
- Static file serving
- Auto-scaling

Setup

```

import { railwayAdapter } from 'philjs-adapters/railway';

export default railwayAdapter({
  docker: {
    baseImage: 'node:20-alpine',
    packages: ['python3', 'make']
  },
  railway: {
    healthCheckPath: '/health',
    restartPolicy: 'on-failure'
  }
});

```

See [Railway documentation](#) for details.

Vercel

Deploy your PhilJS app to Vercel.

Setup

Create vite.config.ts:

```

import { defineConfig } from 'vite';
import philjs from 'philjs-cli/vite';
import vercel from 'philjs-adapters/vercel';

export default defineConfig({
  plugins: [
    philjs(),
    vercel({
      edge: true, // Use Edge Functions
      isr: {
        expiration: 60 // ISR revalidation time in seconds
      }
    })
  ]
});

```

Deploy

```

pnpm build
vercel deploy

```

Netlify

Deploy your PhilJS app to Netlify.

Setup

Create netlify.toml:

```

[build]
command = "pnpm build"
publish = "dist"

[functions]
directory = "dist/functions"

```

Update vite.config.ts:

```

import { defineConfig } from 'vite';
import philjs from 'philjs-cli/vite';
import netlify from 'philjs-adapters/netlify';

export default defineConfig({
  plugins: [
    philjs(),
    netlify({
      edge: true // Use Edge Functions
    })
  ]
});

```

Deploy

```

pnpm build
netlify deploy --prod

```

Cloudflare Workers

Deploy to Cloudflare Pages or Workers.

Setup

Create wrangler.toml:

```

name = "my-philjs-app"
main = "dist/worker.js"
compatibility_date = "2024-01-01"

[site]
bucket = "dist/client"

```

Update vite.config.ts:

```

import { defineConfig } from 'vite';
import philjs from 'philjs-cli/vite';
import cloudflare from 'philjs-adapters/cloudflare';

export default defineConfig({
  plugins: [
    philjs(),
    cloudflare({
      routes: true, // Generate routes manifest
      kvNamespaces: ['MY_KV'] // KV namespace bindings
    })
  ]
});

```

Deploy

```
pnpm build  
wrangler deploy
```

AWS Lambda

Deploy to AWS Lambda with API Gateway.

Setup

Update vite.config.ts:

```
import { defineConfig } from 'vite';
import philjs from 'philjs-cli/vite';
import aws from 'philjs-adapters/aws';

export default defineConfig({
  plugins: [
    philjs(),
    aws({
      region: 'us-east-1',
      runtime: 'nodejs20.x'
    })
  ]
});
```

Create serverless.yml or use AWS SAM:

```
service: my-philjs-app

provider:
  name: aws
  runtime: nodejs20.x
  region: us-east-1

functions:
  app:
    handler: dist/server/index.handler
    events:
      - httpApi: '*'

plugins:
  - serverless-s3-sync

custom:
  s3sync:
    - bucketName: my-app-assets
      localDir: dist/client
```

Deploy

```
pnpm build  
serverless deploy
```

Node.js

Deploy to any Node.js server.

Setup

Update vite.config.ts:

```
import { defineConfig } from 'vite';
import philjs from 'philjs-cli/vite';
import node from 'philjs-adapters/node';

export default defineConfig({
  plugins: [
    philjs(),
    node({
      port: 3000,
      compression: true
    })
  ]
});
```

Create server.js:

```

import { createServer } from 'philjs-adapters/node';
import handler from './dist/server/index.js';

const server = createServer(handler, {
  port: process.env.PORT || 3000,
  static: './dist/client'
});

server.listen();

```

Deploy

```

pnpm build
node server.js

```

Static (SSG)

Generate a fully static site.

Setup

Update vite.config.ts:

```

import { defineConfig } from 'vite';
import philjs from 'philjs-cli/vite';
import staticAdapter from 'philjs-adapters/static';

export default defineConfig({
  plugins: [
    philjs(),
    staticAdapter({
      pages: [
        '/',
        '/about',
        '/blog',
        '/contact'
      ],
      // Or use dynamic route generation
      dynamicRoutes: async () => {
        const posts = await fetchPosts();
        return posts.map(post => `/blog/${post.slug}`);
      }
    })
  ]
});

```

Build

```

pnpm build

```

Output will be in dist/ ready to deploy to any static host (GitHub Pages, S3, Netlify, etc).

Bun

Deploy your PhilJS app using Bun's native server.

Setup

Create server.ts:

```

import { createBunAdapter } from 'philjs-adapters/bun';

const handler = createBunAdapter({
  port: 3000,
  staticDir: 'public',
  compression: true,
  websocket: {
    enabled: true,
  },
});

// Export for Bun.serve
export default handler;

// Or start directly
// handler.start();

```

Run

```

# Development with hot reload
bun run --hot server.ts

# Production
bun run server.ts

```

Features

- **Native Bun.serve()** - Uses Bun's high-performance HTTP server
- **Fast File Serving** - Leverages Bun's native file API
- **SQLite Support** - Built-in SQLite via Bun's native bindings
- **WebSocket Support** - Full WebSocket server capabilities
- **Hot Reload** - Development mode with instant updates
- **Compression** - Built-in gzip compression

SQLite Example

```
import { createBunSQLite } from 'philjs-adapters/bun';

const db = createBunSQLite('./data/app.db');

// Create table
db.run(`CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT)`);

// Query
const users = db.query('SELECT * FROM users');

// Insert
db.run('INSERT INTO users (name) VALUES (?)', ['John']);
```

WebSocket Example

```
import { createBunAdapter, onWebSocketMessage, onWebSocketOpen } from 'philjs-adapters/bun';

const handler = createBunAdapter({
  port: 3000,
  websocket: { enabled: true },
});

onWebSocketOpen(handler, (ws) => {
  console.log('Client connected');
  ws.send('Welcome!');
});

onWebSocketMessage(handler, (ws, message) => {
  console.log('Received:', message);
  ws.send(`Echo: ${message}`);
});

handler.start();
```

Deno

Deploy your PhilJS app using Deno's native server, with Deno Deploy support.

Setup

Create `server.ts`:

```
import { createDenoAdapter } from 'philjs-adapters/deno';

const handler = createDenoAdapter({
  port: 8000,
  staticDir: 'public',
  compression: true,
  kv: true, // Enable Deno KV
});

// Use with Deno.serve
Deno.serve({ port: 8000 }, handler);
```

Run

```
# Development
deno run --allow-net --allow-read --watch server.ts

# With Deno KV
deno run --allow-net --allow-read --unstable-kv server.ts

# Production
deno run --allow-net --allow-read server.ts
```

Deploy to Deno Deploy

```
deployctl deploy --project=your-project server.ts
```

Features

- **Deno.serve()** - Uses Deno's native HTTP server
- **Deno KV** - Built-in key-value storage with TTL support
- **Deno Deploy Ready** - Zero-config deployment to the edge

- **Permission-Aware** - Respects Deno's security model
- **npm Compatibility** - Use npm packages with Deno

Deno KV Example

```
import { createDenoKV } from 'philjs-adapters/deno';

const kv = await createDenoKV();

// Set value
await kv.set(['users', 'user-1'], { name: 'John' });

// Get value
const user = await kv.get(['users', 'user-1']);

// Set with TTL (expires in 60 seconds)
await kv.setWithTTL(['cache', 'data'], { value: 'cached' }, 60000);

// List by prefix
const allUsers = await kv.list(['users']);
```

Permission Checking

```
import { checkPermissions, requestPermission } from 'philjs-adapters/deno';

const perms = await checkPermissions();

if (!perms.net) {
  const granted = await requestPermission('net');
  if (!granted) {
    console.error('Network permission required');
    Deno.exit(1);
  }
}
```

Adapter Options

Vercel

```
{
  edge?: boolean;           // Use Edge Functions (default: false)
  isr?: {
    expiration?: number;   // ISR revalidation time in seconds
  };
  regions?: string[];       // Deployment regions
  memory?: number;          // Function memory in MB
  maxDuration?: number;     // Max execution time in seconds
}
```

Netlify

```
{
  edge?: boolean;           // Use Edge Functions (default: false)
  redirects?: Array<{
    from: string;
    to: string;
    status?: number;
  }>;
}
```

Cloudflare

```
{
  routes?: boolean;         // Generate routes manifest (default: true)
  kvNamespaces?: string[];  // KV namespace bindings
  d1Databases?: string[];   // D1 database bindings
  r2Buckets?: string[];    // R2 bucket bindings
}
```

AWS

```
{
  region?: string;          // AWS region (default: 'us-east-1')
  runtime?: string;          // Node.js runtime version
  memorySize?: number;        // Lambda memory in MB
  timeout?: number;           // Lambda timeout in seconds
}
```

Node.js

```
{
  port?: number;             // Server port (default: 3000)
  compression?: boolean;      // Enable gzip compression (default: true)
  static?: string;            // Static files directory
  middleware?: Function[];    // Custom middleware
}
```

Static

```
{  
  pages?: string[]; // Static pages to generate  
  dynamicRoutes?: () => Promise<string[]>; // Dynamic route generator  
  trailingSlash?: boolean; // Add trailing slashes (default: false)  
  prerenderAll?: boolean; // Prerender all routes (default: false)  
}
```

Bun

```
{  
  port?: number; // Server port (default: 3000)  
  hostname?: string; // Hostname to bind (default: '0.0.0.0')  
  development?: boolean; // Enable dev mode (default: NODE_ENV !== 'production')  
  staticDir?: string; // Static files directory (default: 'public')  
  compression?: boolean; // Enable gzip compression (default: true)  
  sqlite?: string; // SQLite database path  
  websocket?: {  
    enabled?: boolean; // Enable WebSocket support  
    maxPayloadLength?: number; // Max message size (default: 16MB)  
    idleTimeout?: number; // Connection idle timeout (default: 120s)  
  };  
  tls?: {  
    key: string; // Path to TLS key file  
    cert: string; // Path to TLS certificate file  
  };  
  maxRequestBodySize?: number; // Max request body size (default: 128MB)  
  idleTimeout?: number; // Request idle timeout (default: 10s)  
}
```

Deno

```
{  
  port?: number; // Server port (default: 8000)  
  hostname?: string; // Hostname to bind (default: '0.0.0.0')  
  kv?: boolean | string; // Enable Deno KV (true or path to KV database)  
  staticDir?: string; // Static files directory (default: 'public')  
  compression?: boolean; // Enable compression (default: true)  
  deploy?: {  
    project?: string; // Deno Deploy project name  
    edgeCache?: boolean; // Enable edge caching  
    kvDatabase?: string; // KV database name  
  };  
  tls?: {  
    key: string; // Path to TLS key file  
    cert: string; // Path to TLS certificate file  
  };  
  signal?: AbortSignal; // Signal for graceful shutdown  
  onListen?: (params: { hostname: string; port: number }) => void;  
  onError?: (error: Error) => Response | Promise<Response>;  
}
```

Runtime Detection

Automatically detect the current JavaScript runtime:

```
import {  
  detectRuntime,  
  getRuntimeInfo,  
  isBun,  
  isDeno,  
  isNode,  
  hasFeature,  
} from 'philjs-adapters';  
  
// Get current runtime  
const runtime = detectRuntime(); // 'bun' | 'deno' | 'node' | 'edge' | 'browser'  
  
// Check specific runtime  
if (isBun()) {  
  console.log('Running in Bun');  
}  
  
if (isDeno()) {  
  console.log('Running in Deno');  
}  
  
// Get detailed runtime info  
const info = getRuntimeInfo();  
console.log(info.runtime); // 'bun'  
console.log(info.version); // '1.0.0'  
console.log(info.features); // { fetch: true, webSocket: true, sqlite: true, ... }  
  
// Check feature support  
if (hasFeature('sqlite')) {  
  console.log('SQLite is available');  
}
```

Environment Variables

Each adapter supports environment variables for configuration:

```
# Vercel
VERCEL_ENV=production

# Netlify
NETLIFY_ENV=production
CONTEXT=production

# Cloudflare
CF_PAGES=1
CF_PAGES_BRANCH=main

# AWS
AWS_REGION=us-east-1
AWS_LAMBDA_FUNCTION_NAME=my-app

# Node.js
NODE_ENV=production
PORT=3000

# Bun
NODE_ENV=production
PORT=3000

# Deno
DENO_ENV=production
PORT=8000
DENO_DEPLOYMENT_ID=xxx      # Set automatically on Deno Deploy
DENO_REGION=us-east-1        # Set automatically on Deno Deploy
```

Custom Adapters

Create your own adapter:

```
import { defineAdapter } from 'philjs-adapters';

export default defineAdapter({
  name: 'my-adapter',
  async adapt({ outDir, config }) {
    // Custom build Logic
    console.log('Building for my platform...');

    // Generate platform-specific files
    await generateFiles(outDir, config);

    return {
      success: true,
      files: ['server.js', 'client/']
    };
  }
});
```

Documentation

For more information, see: - Deployment Guide - Vercel Deployment - Netlify Deployment - Cloudflare Deployment

Examples

See the [examples](#) directory for adapter-specific examples:

- [Bun Examples](#) - Bun server with WebSocket and SQLite
- [Deno Examples](#) - Deno server with KV storage

Quick Start - Bun

```
cd examples/bun
bun install
bun run dev
```

Quick Start - Deno

```
cd examples/deno
deno task dev
```

Presets

Use presets for common configurations:

```

import { createAdapter } from 'philjs-adapters';

// Bun presets
const bunAdapter = createAdapter('bun');
const bunWsAdapter = createAdapter('bun-websocket');

// Deno presets
const denoAdapter = createAdapter('deno');
const denoKvAdapter = createAdapter('deno-kv');
const denoDeployAdapter = createAdapter('deno-deploy');

// Other presets
const vercelEdge = createAdapter('vercel-edge');
const cloudflarePages = createAdapter('cloudflare-pages');
const awsLambda = createAdapter('aws-lambda');

```

Auto-Detection

The autoAdapter function automatically detects the runtime and platform:

```

import { autoAdapter } from 'philjs-adapters';

// Automatically uses the correct adapter based on environment
const adapter = autoAdapter();

// Detection order:
// 1. Bun (if Bun global is present)
// 2. Deno (if Deno global is present)
// 3. Vercel (if VERCEL env var is set)
// 4. Netlify (if NETLIFY env var is set)
// 5. Cloudflare (if CF_PAGES or CLOUDFLARE_WORKERS env var is set)
// 6. AWS (if AWS_LAMBDA_FUNCTION_NAME or AWS_EXECUTION_ENV env var is set)
// 7. Node.js (default)

```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: ., /vercel, /netlify, /cloudflare, /aws, /node, /static, /bun, /deno, /runtime-detect, /cloudflare-pages, /vercel/adapter, /netlify/adapter, /aws-lambda, /railway, /edge, /adapters/vercel, /adapters/cloudflare, /adapters/deno-deploy, /adapters/netlify, /adapters/aws-lambda, /adapters/node, /adapters/bun, /utils/build, /utils/env
- Source files: packages/philjs-adapters/src/index.ts, packages/philjs-adapters/src/vercel/index.ts, packages/philjs-adapters/src/netlify/index.ts, packages/philjs-adapters/src/cloudflare/index.ts, packages/philjs-adapters/src/aws/index.ts, packages/philjs-adapters/src/node/index.ts, packages/philjs-adapters/src/static/index.ts, packages/philjs-adapters/src/bun/index.ts, packages/philjs-adapters/src/deno/index.ts, packages/philjs-adapters/src/runtime-detect.ts, packages/philjs-adapters/src/cloudflare-pages/index.ts, packages/philjs-adapters/src/vercel/adapter.ts, packages/philjs-adapters/src/netlify/adapter.ts, packages/philjs-adapters/src/aws-lambda/index.ts, packages/philjs-adapters/src/railway/index.ts, packages/philjs-adapters/src/edge/index.ts, packages/philjs-adapters/src/adapters/vercel.ts, packages/philjs-adapters/src/adapters/cloudflare.ts, packages/philjs-adapters/src/adapters/deno-deploy.ts, packages/philjs-adapters/src/adapters/netlify.ts, packages/philjs-adapters/src/adapters/aws-lambda.ts, packages/philjs-adapters/src/adapters/node.ts, packages/philjs-adapters/src/adapters/bun.ts, packages/philjs-adapters/src/utils/build.ts, packages/philjs-adapters/src/utils/env.ts

Public API

- Direct exports: \$wrapper, APIGatewayConfig, AWSConfig, AWSLambdaAdapterConfig, AWSLambdaConfig, AWSLambdaContext, AdapterPreset, AnalyticsEngineBinding, AssetManifestEntry, AuthorizationConfig, BackgroundFunctionConfig, BlobListOptions, BlobListResult, BlobObject, BlobPutOptions, BuildConfig, BuildManifest, BuildManifestOptions, BunAdapterConfig, BunConfig, BunOptimizations, BunServer, BunServerHandler, BunWebSocket, BunWebSocketConfig, BunWebSocketHandlers, CORSConfig, CacheBehavior, CloudFrontConfig, CloudflareAdapterConfig, CloudflareConfig, CloudflarePagesConfig, ClusterConfig, CronConfig, CronTrigger, D1Database, D1DatabaseBinding, D1ExecResult, D1PreparedStatement, D1Result, DenoCompilerOptions, DenoConfig, DenoDeployAdapterConfig, DenoDeployConfig, DenoDeployRegion, DenoFmtConfig, DenoKVAtomicWrapper, DenoKVConfig, DenoKVWrapper, DenoKv, DenoKvAtomic, DenoLintConfig, DenoServeHandler, DenoServeHandlerInfo, DevConfig, DurableObjectBinding, EnhancedAdapterPreset, EnvConfig, EnvDefinition, ExecutionContext, FreshConfig, FunctionRouteConfig, FunctionUrlConfig, HeaderConfig, ISRCConfig, ImageOptimizationConfig, ImportMapConfig, KVNamespace, KVNamespaceBinding, KVNamespaceListOptions, KVNamespaceListResult, KVNamespacePutOptions, LifecycleRule, LoadedEnv, LoggingConfig, MIME_TYPES, MiddlewareConfig, NetlifyAccount, NetlifyAdapterConfig, NetlifyBlob, NetlifyBuildConfig, NetlifyConfig, NetlifyContextConfig, NetlifyGeo, NetlifyHeaderConfig, NetlifyImageConfig, NetlifyPluginConfig, NetlifyRedirectConfig, NetlifyRegion, NetlifySite, NetlifyUser, NodeAdapterConfig, NodeConfig, OptimizeAssetsOptions, PhilJSStack, PlacementConfig, QueueBinding, R2Bucket, R2BucketBinding, R2Conditional, R2GetOptions, R2HTTPMetadata, R2ListOptions, R2Object, R2Object, R2Objects, R2PutOptions, R2Range, RailwayConfig, RedirectConfig, RewriteConfig, RouteConfig, RouteManifestEntry, Runtime, RuntimeFeatures, RuntimeInfo, S3Config, S3Helpers, SQLiteConfig, ScheduledFunctionConfig, SecretsManager, ServiceBinding, SiteConfig, StaticConfig, VPCConfig, VercelAdapterConfig, VercelBlob, VercelConfig, VercelEdgeConfig, VercelKV, VercelRegion, WebSocketConfig, assertRuntime, autoAdapter, awsAdapter, awsLambdaAdapter, bunAdapter, bunFile, bunHash, bunVerify, checkDenoPermissions, checkPermissions,.cloudflareAdapter, cloudflarePagesAdapter, config, copyStaticAssets, createAdapter, createBuildManifest, createBunAdapter, createBunHandler, createBunSQLite, createD1Database, createD1Helper, createDenoAdapter, createDenoKV, createEnhancedAdapter, createEnvSecretsManager, createExpressMiddleware, createFastifyPlugin, createFormsHandler, createKVHelper, createKVNamespace, createMemorySecretsManager, createOutputStructure, createR2Bucket, createR2Helper, createVercelEdgeConfig, defineAdapter, denoAdapter, denoDeployAdapter, detectRuntime, enhancedPresets, generateEnvTypes, generateHashedFilename, getAWSContext, getAWSLambdaContext, getAWSRequestId, getCacheControl, getCloudflareEnv, getDenoDeployRegion, getDenoKV, getDurableObject, getEnv, getExecutionContext, getFileHash, getMode, getNetlifyContext, getNetlifyIdentityUser, getRemainingTimeMs, getRequestid, getRuntimeInfo, getS3AssetUrl, getStaticPaths, getVercelContext, handler, hasFeature, injectEnvVariables, isBrowser, isBun, isDeno, isDenoDeploy, isDevelopment, isEdge, isImmutableAsset, isNode, isProduction, isServer, loadEnvFile, loadEnvironment, netlifyAdapter, netlifyImageCDN, nodeAdapter, onWebSocketClose, onWebSocketMessage, onWebSocketOpen, optimizeAssets, passThroughOnException, platformEnv, prerender, presets, railwayAdapter, requestDenoPermission, requestPermission, revalidatePath, revalidateTag, signCloudFrontUrl, startDenoServer, startServer, staticAdapter, validateEnv, vercelAdapter, vercelImageUrl, waitUntil
- Re-exported names: // A/B testing selectGeoVariant, // Asset caching createAssetCache, // Cache class EdgeCache, // Caching EdgeCache, // Cold start isColdStart, // Default cache getDefaultCache, // Default exports edgeRuntime, // Distance calculateDistance, // ESL parseESITags, // Edge Runtime detectEdgePlatform, // Edge Runtime Types EdgePlatform, // Environment createEdgeEnv, // Geo Types GeoLocation, // Geo extraction getGeoLocation, // Geo routing applyGeoRouting, // Geolocation getGeoLocation, // HTML streaming createHTMLStream, // Handler createEdgeHandler, // Platform detection detectEdgePlatform, // Region routing findBestRegion, // Response caching createCacheKey, // SSE createSSEStream, // Stream creation createWritableStream, // Streaming createWritableStream, // Streaming Types StreamingConfig, // Types type CacheEntry, // Types type EdgePlatform, // Types type GeoLocation, // Types type StreamingConfig, // Utilities addGeoHeaders, // Utilities streamThrough, APIGatewayConfig, AWSConfig, AWSLambdaAdapterConfig, AWSLambdaConfig, AWSLambdaContext, Adapter, AdapterConfig

AssetCacheOptions, AssetManifestEntry, AuthorizationConfig, BackgroundFunctionConfig, BuildManifest, BuildManifestOptions, BuildOutput, BunAdapterConfig, BunConfig, BunOptimizations, BunServer, BunServerHandler, BunWebSocket, BunWebSocketConfig, CORSConfig, CacheBehavior, CacheOptions, CacheStats, CloudFrontConfig, CloudflareAdapterConfig, CloudflareConfig, CloudflarePagesConfig, CloudflareRouteConfig, ClusterConfig, ColdStartConfig, CronConfig, D1Database, D1DatabaseBinding, D1PreparedStatement, DenoCompilerOptions, DenoConfig, DenoDeployAdapterConfig, DenoDeployConfig, DenoDeployRegion, DenoKVAtomicWrapper, DenoKVConfig, DenoKVWrapper, DenoKv, DenoServeHandler, DurableObjectBinding, ESIFragment, EdgeAdapter, EdgeCacheConfig, EdgeContext, EdgeEnv, EdgeExecutionContext, EdgeHandlerOptions, EdgeKVListOptions, EdgeKVListResult, EdgeKVNamespace, EdgeKVPutOptions, EdgeKVStore, EdgeRegion, EdgeRuntimeConfig, EdgeTiming, EnvConfig, EnvDefinition, FreshConfig, FunctionRouteConfig, FunctionUrlConfig, GeoABTestConfig, GeoRoutingConfig, GeoRoutingRule, HTMLStreamOptions, HeaderConfig, ISRConfig, ImageOptimizationConfig, ImportMapConfig, KVNamespace, KVNamespaceBinding, LatencyRoutingConfig, LoadedEnv, LoggingConfig, MIME_TYPES, MiddlewareConfig, NetlifyAdapterConfig, NetlifyBuildConfig, NetlifyConfig, NetlifyConfigEnhanced, NetlifyContextConfig, NetlifyGeo, NetlifyHeaderConfig, NetlifyImageConfig, NetlifyPluginConfig, NetlifyRedirectConfig, NetlifyRegion, NetlifySite, NodeAdapterConfig, NodeConfig, OptimizeAssetsOptions, QueueBinding, R2Bucket, R2BucketBinding, RailwayConfig, RedirectConfig, RegionConfig, RequestContext, ResponseCacheOptions, ResponseContext, RewriteConfig, RouteEntry, RouteManifest, RouteManifestEntry, Runtime, RuntimeFeatures, RuntimeInfo, S3Config, SQLiteConfig, SSEMessage, ScheduledFunctionConfig, SecretsManager, ServerlessAdapter, ServiceBinding, StaticConfig, StaticConfigV2, StreamingWriter, VPCConfig, VercelAdapterConfig, VercelConfig, VercelConfigEnhanced, VercelRegion, WebSocketConfig, addGeoHeaders, applyGeoRouting, assertRuntime, awsAdapter, awsLambdaAdapter, awsLambdaAdapterV2, bunAdapter, bunAdapterV2, bunFile, bunHash, bunVerify, cache, calculateDistance, checkDenoPermissions, checkPermissions, cloudflareAdapter, cloudflareAdapterV2, cloudflarePagesAdapter, coalesceRequest, copyStaticAssets, createAssetCache, createBuildManifest, createBunAdapter, createBunHandler, createBunSQLite, createBunSQLiteV2, createCacheKey, createCacheMiddleware, createCachedResponse, createD1Database, createD1Helper, createDenoAdapter, createDenoKV, createESIMiddleware, createEdgeEnv, createEdgeHandler, createEnvSecretsManager, createExecutionContext, createExpressMiddleware, createFastifyPlugin, createFormsHandler, createGeoRoutingMiddleware, createHTMLStream, createKVHelper, createKVHelperV2, createKVNamespace, createLatencyRouter, createMemorySecretsManager, createOutputStructure, createR2Bucket, createR2Helper, createSSEHandler, createSSEStream, createStreamTee, createStreamingResponse, createVariantCookie, createVercelEdgeConfig, createVercelEdgeConfigV2, denoAdapter, denoDeployAdapter, detectRuntime, edgeRuntime, findBestRegion, findNearestLocation, generateEnvTypes, generateHashedFilename, geo, getAWSContext, getAWSLambdaContext, getAWSRequestId, getCacheControl, getClientIP, getCloudflareEnv, getCloudflareEnvV2, getColdStartDuration, getDefaultCache, getDenoDeployRegion, getDenoDeployRegionV2, getDenoKV, getDurableObject, getEnv, getExecutionContext, getExecutionContextV2, getFileHash, getMode, getNetlifyContext, getNetlifyContextV2, getNetlifyIdentityUser, getPlatformInfo, getPreloadedModule, getRegion, getRemainingTimeMs, getRemainingTimeMsV2, getRequestid, getRuntimeInfo, getS3AssetUrl, getS3AssetUrlV2, getStaticPaths, getVercelContext, hasFeature, initializeColdStart, injectEnvVariables, isBrowser, isBun, isBunV2, isColdStart, isDeno, isDenoDeployV2, isDenoDeploy, isDevelopment, isEdge, isImmutableAsset, isNode, isProduction, isServer, loadEnvFile, loadEnvironment, markWarm, mergeStreams, netlifyAdapter, netlifyAdapterEnhanced, netlifyAdapterV2, netlifyImageCDN, netlifyImageCDNV2, nodeAdapter, nodeAdapterV2, onWebSocketMessage, onWebSocketOpen, optimizeAssets, parseESITags, passThroughOnException, platformEnv, preloadModule, prerender, processESI, railwayAdapter, requestDenoPermission, requestPermission, resetColdStartTracking, resetDefaultCache, revalidatePath, revalidatePathV2, revalidateTag, revalidateTagV2, selectGeoVariant, shouldCacheresponse, startDenoServer, startServer, startServerV2, staticAdapter, streamThrough, streaming, validateEnv, vercelAdapter, vercelAdapterEnhanced, vercelAdapterV2, vercelImageUrl, waitUntil, waitUntilV2

- Re-exported modules: ./adapters/aws-lambda.js, ./adapters/bun.js, ./adapters/cloudflare.js, ./adapters/deno-deploy.js, ./adapters/netlify.js, ./adapters/node.js, ./adapters/vercel.js, ./aws-lambda/index.js, ./aws/index.js, ./bun/index.js, ./cache.js, ./cloudflare-pages/index.js, ./cloudflare/index.js, ./deno/index.js, ./edge-runtime.js, ./edge/index.js, ./geo.js, ./netlify/index.js, ./node/index.js, ./railway/index.js, ./runtime-detect.js, ./static/index.js, ./streaming.js, ./types.js, ./utils/build.js, ./utils/env.js, ./vercel/adapter.js, ./vercel/index.js

License

MIT

@philjs/ai - Type: Node package - Purpose: AI-powered code generation and analysis tools for PhilJS - Version: 0.1.0 - Location: packages/philjs-ai - Entry points: packages/philjs-ai/src/index.ts, packages/philjs-ai/src/providers/index.ts, packages/philjs-ai/src/codegen.ts, packages/philjs-ai/src/autocomplete/index.ts, packages/philjs-ai/src/refactor/index.ts, packages/philjs-ai/src/testing/index.ts, packages/philjs-ai/src/docs/index.ts, packages/philjs-ai/src/cli/index.ts, ./vscode

philjs-ai

AI adapter with typed prompts and safety hooks for PhilJS.

Requirements

- **Node.js 24 or higher**
- **TypeScript 6 or higher**
- **ESM only** - CommonJS is not supported

Features

- **Type-safe AI prompts** - Typed prompt templates with validation
- **Multiple AI providers** - OpenAI, Anthropic, Google, local models
- **Streaming support** - Real-time streaming responses
- **Safety hooks** - Content moderation and rate limiting
- **Cost tracking** - Monitor API usage and costs
- **Caching** - Intelligent response caching
- **Error handling** - Automatic retries and fallbacks

Installation

```
pnpm add philjs-ai
```

Quick Start

Basic Usage

```

import { createAIClient, prompt } from 'philjs-ai';

const ai = createAIClient({
  provider: 'openai',
  apiKey: process.env.OPENAI_API_KEY,
  model: 'gpt-4'
});

// Simple completion
const response = await ai.complete('What is PhilJS?');
console.log(response.text);

// Typed prompts
const userGreeting = prompt<{ name: string }>`  

  Greet the user named {name} in a friendly way.  

`;

const greeting = await ai.complete(userGreeting({ name: 'Alice' }));
console.log(greeting.text);

```

Streaming Responses

```

import { createAIClient } from 'philjs-ai';
import { signal } from 'philjs-core';

const ai = createAIClient({
  provider: 'openai',
  apiKey: process.env.OPENAI_API_KEY
});

const response = signal('');

await ai.stream('Write a short story about PhilJS', {
  onChunk: (chunk) => {
    response.set(response() + chunk);
  }
});

```

With Safety Hooks

```

import { createAIClient, moderateContent } from 'philjs-ai';

const ai = createAIClient({
  provider: 'openai',
  apiKey: process.env.OPENAI_API_KEY,
  hooks: {
    beforeRequest: async (prompt) => {
      // Content moderation
      const safe = await moderateContent(prompt);
      if (!safe) {
        throw new Error('Inappropriate content detected');
      }
      return prompt;
    },
    afterResponse: async (response) => {
      // Log for monitoring
      console.log(`Tokens used: ${response.usage.totalTokens}`);
      return response;
    }
  }
});

```

Supported Providers

- **OpenAI** - GPT-4, GPT-3.5, etc.
- **Anthropic** - Claude 3 Opus, Sonnet, Haiku
- **Google** - Gemini Pro
- **Local** - Ollama, LM Studio

API Reference

`createAIClient(config)`

Create an AI client instance.

`ai.complete(prompt, options?)`

Generate a text completion.

`ai.stream(prompt, options?)`

Stream a response in real-time.

`prompt<t>`

Create a typed prompt template.

Documentation

For more information, see the PhilJS documentation.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .., ./providers, ./codegen, ./autocomplete, ./refactor, ./testing, ./docs, ./cli, ./vscode
- Source files: packages/philijs-ai/src/index.ts, packages/philijs-ai/src/providers/index.ts, packages/philijs-ai/src/codegen.ts, packages/philijs-ai/src/autocomplete/index.ts, packages/philijs-ai/src/refactor/index.ts, packages/philijs-ai/src/testing/index.ts, packages/philijs-ai/src/docs/index.ts, packages/philijs-ai/src/cli/index.ts

Public API

- Direct exports: AccessibilityAudit, AccessibilityFix, AccessibilityIssue, AutocompleteContext, AutocompleteEngine, AutocompleteSuggestion, BestPracticeImprovement, BestPracticeViolation, BestPracticesResult, CodeContext, CodeExplanation, CodeGenOptions, CodeGenerator, CodeSection, CompletionItem, CompletionItemKind, ComponentInfo, ErrorInfo, FixSuggestion, GeneratedCode, GeneratedComponentResult, GeneratedFunctionResult, GeneratedTestsResult, ImportSuggestion, InlineCompletionResult, ModuleInfo, ParameterInfo, PerformanceAnalysis, PerformanceIssue, PerformanceOptimization, ProjectContext, ProviderConfig, ProviderRegistry, RefactorChange, RefactorConfig, RefactorFocusArea, RefactorResult, RefactoringEngine, SignatureHelpResult, SignatureInfo, SuggestionKind, TestCase, TextEdit, TypeInfo, UtilityInfo, analyzePerformance, auditAccessibility, autoDetectProvider, createAI, createAIClient, createAutocompleteEngine, createCodeGenerator, createProgram, createPrompt, createProvider, createRefactoringEngine, explainCode, generateComponent, generateFunction, generateTests, getCompletions, getFixSuggestions, getInlineCompletion, getSignatureHelp, getSuggestions, providerRegistry, providers, refactorCode
- Re-exported names: A11yTest, AIAssistant, APIDocConfig, APIGenerationConfig, APIGenerator, APITypeInference, AccessibilityAnalysis, AccessibilityAudit, AccessibilityConfig, AccessibilityIssue, AdvancedGeneratedDocumentation, AdvancedGeneratedReadme, AdvancedRefactoringEngine, AdvancedTestFramework, AdvancedTestGenOptions, AdvancedTestGenerator, AdvancedTestType, Agent, AgentConfig, AgentStep, AnthropicConfig, AnthropicProvider, AntiPatternResult, AssistantConfig, AssistantConversationMessage, AssistantProjectContext, AutoFixResult, AutocompleteContext, AutocompleteEngine, AutocompleteSuggestion, AvailableImport, BestPracticesResult, CRUDGenerationResult, CRUDOperation, CategorySummary, ChatResponse, ChromaConfig, ChromaVectorStore, ClientCapabilities, CodeAnalyzer, CodeContext, CodeExplanation, CodeGenOptions, CodeGenRefactorResult, CodeGenRequest, CodeGenResult, CodeGenTestCase, CodeGenerator, CodeLocation, CodeMetrics, CodeReviewer, CodeSection, CodeSnippet, CohereConfig, CohereProvider, CompletionItem, CompletionItemKind, ComplexityMetrics, ComponentAnalysis, ComponentCategory, ComponentDoc, ComponentExample, ComponentGenerationConfig, ComponentGenerator, ComponentSuggester, ComponentSuggestion, ComputedAnalysis, ContextAnalysis, ConversationContext, ConversationMessage, CoverageAnalysis, DataLoadingConfig, DatabaseType, DependencyAnalysis, DetectedAntiPattern, DocBlock, DocCoverage, DocGenerationConfig, DocGenerationOptions, DocGenerator, DocParam, DocReturn, DocStyle, DocTag, Document, DocumentLoader, DocumentStore, DocumentationGenerator, DocumentationStyle, E2EScenario, E2ETestConfig, EffectAnalysis, ErrorInfo, EventDoc, EventHandlerAnalysis, FieldDefinition, FieldValidation, FileReview, FixSuggestion, FixtureFile, GeminiConfig, GeminiProvider, GeneratedAPI, GeneratedAPIDoc, GeneratedArtifact, GeneratedCode, GeneratedComponent, GeneratedComponentResult, GeneratedComponentType, GeneratedDocumentation, GeneratedE2ETests, GeneratedFunctionResult, GeneratedPage, GeneratedReadme, GeneratedSchemaComponent, GeneratedTestSuite, GeneratedTests, GeneratedTestsResult, GenerationIntent, InMemoryVectorStore, InferenceConfidence, InferenceSource, InferredInterface, InferredTypeAlias, InitializeResult, InlineCompletionResult, IntentEntity, InterfaceProperty, JSONToTypeResult, LMStudioConfig, LMStudioProvider, LSPHandlers, LSPMessage, LSPRequest, LSResponse, LayoutConfig, LayoutSuggestion, LineComment, LocalConfig, LocalProvider, MarkdownLoader, MockFile, NLGeneratedCode, NLGenerationOptions, NaturalLanguageGenerator, OpenAIConfig, OpenAIProvider, OptimizationSuggestion, PRReviewResult, PageGenerationConfig, PageGenerator, PageType, ParameterDef, ParameterInfo, ParameterIntent, ParsedIntent, ParsedSchema, PatternDetection, PerformanceAnalysis, PerformanceCategory, PerformanceNote, PhilSAIExtension, PhilSLanguageServer, PineconeConfig, PineconeVectorStore, ProjectComponent, ProjectContext, PropAnalysis, PropDefinition, PropDoc, ProviderConfig, ProviderRegistry, QdrantConfig, QdrantVectorStore, RAGOptions, RAGPipeline, ReadmeConfig, ReadmeOptions, RecursiveCharacterSplitter, RefactorChange, RefactorConfig, RefactorFocusArea, RefactorRequest, RefactorResult, RefactoringAnalysisOptions, RefactoringAnalysisResult, RefactoringCategory, RefactoringEngine, RefactoringPlan, RefactoringStep, RefactoringSuggestion, RenderingAnalysis, ReviewAccessibilityIssue, ReviewConfig, ReviewFocus, ReviewIssue, ReviewResult, ReviewSeverity, ReviewSuggestion, ReviewSummary, SEOConfig, SchemaDefinition, SchemaField, SchemaRelation, SchemaToComponentGenerator, SchemaToComponentOptions, SchemaToComponentResult, SchemaType, SearchResult, SecurityCategory, SecurityFinding, ServerCapabilities, ServerConfig, SignatureHelpResult, SignatureInfo, SlotDoc, StateAnalysis, StateDoc, StateIntent, StateVariable, StyleConfig, StylingIntent, SuggestedProp, SuggestionContext, SuggestionPreferences, SuggestionResult, TestCaseInfo,TestCategory, TestCoverageAnalysis, TestFramework, TestGenerationConfig, TestGenerator, TestType, TextLoader, TextSplitter, TokenSplitter, ToolBuilder, ToolCall, ToolDefinition, ToolExecutor, TypeInferenceHelper, TypeInferenceOptions, TypeInferenceResult, TypeSuggestion, UIHints, VectorStore, WireframeConfig, activate, addJSDoc, addJSDocToCode, analyzeComponent, analyzeForRefactoring, analyzePerformance, auditAccessibility, autoDetectProvider, autoFixCode, calculatorTool, codeExecutionTool, convertJavaScriptToTypeScript, createAIAssistant, createAPIGenerator, createAdvancedRefactoringEngine, createAdvancedTestGenerator, createAgent, createAnthropicProvider, createAutoAssistant, createAutocompleteEngine, createCodeAnalyzer, createCodeGenerator, createCodeReviewer, createCohereProvider, createComponentGenerator, createComponentSuggester, createDocGenerator, createDocumentationGenerator, createGeminiProvider, createLMStudioProvider, createLSPHandlers, createLanguageServer, createLocalProvider, createNaturalLanguageGenerator, createOpenAIProvider, createPageGenerator, createProvider, createRefactoringEngine, createSchemaToComponentGenerator, createTestGenerator, createTool, createTypeInferenceHelper, deactivate, detectAntiPatterns, detectUIPatterns, documentPhilJSComponent, euclideanDistance, explainCode, fileReadTool, generateCRUD, generateCRUDFromSchema, generateComponent, generateComponentFromDescription, generateComponentsFromSchema, generateDocs, generateDocumentation, generateE2ETestScenarios, generateE2ETests, generateFromGraphQL, generateFromJSONSchema, generateFromNaturalLanguage, generateFunction, generateGeneratePage, generateReadme, generateTestSuite, generateTests, generateTestsFromCode, generateUnitTests, generateUnitTestsForCode, getCompletions, getContributionPoints, getDefaultCapabilities, getFixSuggestions, getInitializeResult, getInlineCompletion, getSignatureHelp, getSuggestions, inferTypesForCode, jsonToTypeScript, parseCodeIntent, providerRegistry, refactorCode, refactorCodeWithAI, startStdioServer, suggestComponents, suggestOptimizations, tool, useRAG, weatherTool, webSearchTool
- Re-exported modules: ./analysis.js, ./anthropic.js, ./assistant/code-reviewer.js, ./assistant/index.js, ./autocomplete/index.js, ./codegen.js, ./codegen-api-generator.js, ./codegen/component-generator.js, ./codegen/natural-language-generator.js, ./codegen/page-generator.js, ./cohere.js, ./doc-generator.js, ./docs/documentation-generator.js, ./gemini.js, ./inference/type-inference.js, ./lmstudio.js, ./local.js, ./lsp/index.js, ./openai.js, ./providers/anthropic.js, ./providers/cohere.js, ./providers/gemini.js, ./providers/index.js, ./providers/lmstudio.js, ./providers/local.js, ./providers/openai.js, ./rag.js, ./refactor/index.js, ./refactor/refactoring-engine.js, ./schema/schema-to-component.js, ./suggestions/component-suggester.js, ./test-generator.js, ./testing/advanced-test-generator.js, ./testing/test-generator.js, ./tools.js, ./types.js, ./utils/parser.js, ./utils/prompts.js, ./vscode/extension.js

License

MIT

@philijs/ai-agents - Type: Node package - Purpose: Multi-agent AI orchestration for PhilJS - agents, tools, memory, workflows, supervisor patterns - Version: 0.1.0 - Location: packages/philijs-ai-agents - Entry points: packages/philijs-ai-agents/src/index.ts - Keywords: philijs, ai, agents, llm, openai, anthropic, tools, orchestration, workflow

@philijs/ai-agents

Multi-agent AI orchestration framework for PhilJS with tool calling, memory systems, and workflows.

Features

- Agent definition with tools and capabilities
- Multi-agent orchestration and handoffs
- Tool/function calling with type safety
- Memory systems (conversation, semantic, episodic)
- Streaming responses with tool execution
- Agent workflows and pipelines

- Supervisor/worker patterns
- Human-in-the-loop integration

Installation

```
npm install @philjs/ai-agents
```

Usage

Creating an Agent

```
import { createAgent, OpenAIClient, calculatorTool } from '@philjs/ai-agents';

const client = new OpenAIClient(process.env.OPENAI_API_KEY);

const agent = createAgent()
  .name('Assistant')
  .description('A helpful AI assistant')
  .systemPrompt('You are a helpful assistant that can perform calculations.')
  .model('gpt-4-turbo-preview')
  .temperature(0.7)
  .tool(calculatorTool)
  .build(client);

// Chat with the agent
const response = await agent.chat('What is 25 * 4?');
console.log(response.message.content);
```

Streaming Responses

```
for await (const chunk of agent.stream('Tell me a story')) {
  if (chunk.type === 'text') {
    process.stdout.write(chunk.content);
  } else if (chunk.type === 'tool_call_start') {
    console.log(`Calling tool: ${chunk.toolCall?.name}`);
  } else if (chunk.type === 'tool_result') {
    console.log(`Tool result: ${chunk.toolResult?.result}`);
  }
}
```

Custom Tools

```
import { Agent, ToolDefinition } from '@philjs/ai-agents';

const weatherTool: ToolDefinition = {
  name: 'get_weather',
  description: 'Get current weather for a location',
  parameters: {
    type: 'object',
    properties: {
      location: {
        type: 'string',
        description: 'City name'
      }
    },
    required: ['location']
  },
  handler: async (args, context) => {
    const weather = await fetchWeather(args.location);
    return { temperature: weather.temp, conditions: weather.conditions };
  }
};

agent.addTool(weatherTool);
```

Multi-Agent Orchestration

```

import { AgentOrchestrator, Agent } from '@philjs/ai-agents';

const codeAgent = createAgent()
  .name('CodeAgent')
  .capabilities(['code', 'programming', 'debug'])
  .systemPrompt('You are a coding expert.')
  .build(client);

const mathAgent = createAgent()
  .name('MathAgent')
  .capabilities(['math', 'calculate', 'statistics'])
  .systemPrompt('You are a mathematics expert.')
  .build(client);

const orchestrator = new AgentOrchestrator({
  agents: [codeAgent, mathAgent],
  maxHandoffs: 5
});

// Automatically routes to the appropriate agent
const response = await orchestrator.chat('Write a function to calculate fibonacci');

```

Supervisor Pattern

```

import { SupervisorOrchestrator } from '@philjs/ai-agents';

const supervisor = new SupervisorOrchestrator({
  supervisor: managerAgent,
  workers: [researchAgent, writerAgent, editorAgent]
});

// Supervisor delegates tasks to workers
const result = await supervisor.run(
  'Research AI trends and write a blog post about them'
);

```

Agent Workflows

```

import { AgentWorkflow } from '@philjs/ai-agents';

const workflow = new AgentWorkflow()
  .addStep({
    name: 'research',
    agent: researchAgent,
    inputMapper: (input) => `Research: ${input}`
  })
  .addStep({
    name: 'draft',
    agent: writerAgent,
    inputMapper: (prev, ctx) => `Write based on: ${prev}`
  })
  .addStep({
    name: 'edit',
    agent: editorAgent,
    condition: (ctx) => ctx.research.length > 100
  });

const { output, trace } = await workflow.run('AI in healthcare');

```

Memory Systems

```

import { ConversationMemory, SemanticMemory, EpisodicMemory } from '@philjs/ai-agents';

// Simple conversation memory
const convMemory = new ConversationMemory(100);

// Semantic memory with embeddings
const semanticMemory = new SemanticMemory(async (text) => {
  return await embeddings.create(text);
});

// Episodic memory with summaries
const episodicMemory = new EpisodicMemory(async (messages) => {
  return await summarize(messages);
});

```

React Hook

```

import { useAgent } from '@philjs/ai-agents';

function ChatComponent() {
  const { messages, isLoading, streamingContent, send, clear } = useAgent(agent);

  return (
    <div>
      {messages.map(m => <Message key={m.id} {...m} />)}
      {isLoading && <div>{streamingContent}</div>}
      <input onSubmit={(e) => send(e.target.value)} />
    </div>
  );
}

```

API Reference

Core Classes

Class	Description
Agent	Single AI agent with tools and memory
AgentOrchestrator	Multi-agent routing and coordination
SupervisorOrchestrator	Supervisor delegates to worker agents
AgentWorkflow	Sequential agent pipeline
AgentBuilder	Fluent builder for creating agents

LLM Clients

Client	Description
OpenAIclient	OpenAI GPT models
AnthropicClient	Anthropic Claude models

Memory Systems

Memory	Description
ConversationMemory	Simple message history
SemanticMemory	Embedding-based retrieval
EpisodicMemory	Episode summaries

Built-in Tools

Tool	Description
webSearchTool	Search the web
calculatorTool	Math calculations
codeExecutorTool	Execute JavaScript

Types

```

interface AgentConfig {
  name: string;
  description: string;
  systemPrompt: string;
  model?: string;
  provider?: 'openai' | 'anthropic' | 'google' | 'local';
  tools?: ToolDefinition[];
  memory?: MemoryConfig;
  capabilities?: string[];
}

interface ToolDefinition {
  name: string;
  description: string;
  parameters: { type: 'object'; properties: Record<string, any> };
  handler: (args: Record<string, any>, context: AgentContext) => Promise<any>;
}

```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-ai-agents/src/index.ts

Public API

- Direct exports: // Builder createAgent, // Built-in tools webSearchTool, // Core classes Agent, // Hooks useAgent, // LLM Clients OpenAIclient, // Memory ConversationMemory, // Types type Message, AgentBuilder, AgentConfig, AgentContext, AgentOrchestrator, AgentResponse, AgentWorkflow, AnthropicClient, EpisodicMemory, HandoffRequest, LLMClient, LLMRequest, Memory, MemoryConfig, OrchestratorConfig, SemanticMemory, StreamChunk, SupervisorConfig, SupervisorOrchestrator, ToolCall, ToolDefinition, ToolResult, UseAgentResult, WorkflowStep, calculatorTool, codeExecutorTool, useOrchestrator
- Re-exported names: (none detected)

- Re-exported modules: (none detected)

License

MIT

@philjs/ambient - Type: Node package - Purpose: Environment-adaptive UI for PhilJS - light sensing, motion adaptation, attention tracking - Version: 0.1.0 - Location: packages/philjs-ambient - Entry points: packages/philjs-ambient/src/index.ts - Keywords: philjs, ambient, adaptive-ui, light-sensor, motion, attention, context-aware

@philjs/ambient

Environment-adaptive UI framework that responds to ambient conditions like light, motion, and attention.

Features

- Light sensor adaptation (auto dark/light theme)
- Motion-based interactions
- Proximity detection
- Audio environment awareness
- Attention-based dimming
- Context-aware UI morphing
- Time-based adaptations
- Device posture detection

Installation

```
npm install @philjs/ambient
```

Usage

Adaptive UI

```
import { AdaptiveUI } from '@philjs/ambient';

const adaptive = new AdaptiveUI({
  light: {
    darkThreshold: 50,
    brightThreshold: 500,
    autoTheme: true,
    contrastBoost: true
  },
  attention: {
    dimAfterMs: 60000,
    dimLevel: 0.7,
    pauseAnimations: true
  },
  time: {
    nightModeStart: 22,
    nightModeEnd: 6,
    reduceBlueLight: true
  }
});

await adaptive.start();

// Get current context
const context = adaptive.getContext();
console.log('Light level:', context.light.level);
console.log('User activity:', context.motion.activity);
```

Ambient Context Manager

```
import { AmbientContextManager } from '@philjs/ambient';

const manager = new AmbientContextManager();
await manager.start();

// Subscribe to context changes
manager.onUpdate((context) => {
  if (context.light.level === 'dark') {
    document.body.classList.add('dark-theme');
  }

  if (context.motion.isMoving) {
    // Simplify UI for moving user
  }

  if (context.attention.idleTime > 30000) {
    // User is idle - dim screen
  }
});
```

React Hooks

```

import {
  useAmbientContext,
  useAdaptiveUI,
  useLightConditions,
  useMotionState,
  useAttentionState
} from '@philjs/ambient';

function App() {
  const { context, isReady } = useAmbientContext();

  if (!isReady) return <Loading />

  return (
    <div className={context.light.level === 'dark' ? 'dark' : 'light'}>
      <Content />
    </div>
  );
}

function ResponsiveButton() {
  const motion = useMotionState();

  // Larger touch targets when user is moving
  const size = motion?.isMoving ? 'large' : 'normal';

  return <Button size={size}>Click Me</Button>;
}

```

Custom Adaptations

```

const adaptive = new AdaptiveUI({
  custom: [
    {
      condition: (ctx) => ctx.audio.category === 'noisy',
      apply: (element) => {
        element.style.fontSize = '1.2em';
        element.setAttribute('data-loud', 'true');
      },
      revert: (element) => {
        element.style.fontSize = '';
        element.removeAttribute('data-loud');
      }
    }
  ]
});

```

CSS Custom Properties

The library sets CSS custom properties you can use in your styles:

```

/* These are automatically updated based on ambient conditions */
:root {
  --ambient-brightness: 1;
  --ambient-contrast: 1;
  --ambient-motion: normal;
  --ambient-target-scale: 1;
  --ambient-dim: 1;
  --ambient-animation: running;
  --ambient-blue-filter: none;
}

body {
  filter: brightness(var(--ambient-brightness))
    contrast(var(--ambient-contrast));
  opacity: var(--ambient-dim);
}

button {
  transform: scale(var(--ambient-target-scale));
}

* {
  animation-play-state: var(--ambient-animation);
}

```

Include Base Styles

```

import { AmbientCSS } from '@philjs/ambient';

// Inject the base styles
const style = document.createElement('style');
style.textContent = AmbientCSS;
document.head.appendChild(style);

```

API Reference

Classes

Class	Description
AmbientContextManager	Manages all ambient sensors
AdaptiveUI	Automatically adapts UI based on context

Hooks

Hook	Description
useAmbientContext()	Full ambient context with all sensors
useAdaptiveUI(rules?)	Adaptive UI with custom rules
useLightConditions()	Light sensor data only
useMotionState()	Motion/acceleration data only
useAttentionState()	User attention/idle state
useAudioEnvironment()	Audio environment data

Context Types

```
interface AmbientContext {
  light: LightConditions;
  motion: MotionState;
  proximity: ProximityState;
  audio: AudioEnvironment;
  attention: AttentionState;
  time: TimeContext;
  device: DevicePosture;
}

interface LightConditions {
  illuminance: number;
  level: 'dark' | 'dim' | 'normal' | 'bright' | 'very-bright';
  isNatural: boolean;
}

interface MotionState {
  isMoving: boolean;
  acceleration: { x: number; y: number; z: number };
  activity: 'stationary' | 'walking' | 'running' | 'driving' | 'unknown';
}

interface AttentionState {
  isActive: boolean;
  idleTime: number;
  lastInteraction: number;
}

interface DevicePosture {
  orientation: 'portrait' | 'landscape';
  angle: number;
  posture?: 'flat' | 'tent' | 'laptop' | 'held';
}
```

Adaptation Rules

```
interface AdaptationRules {
  light?: {
    darkThreshold: number;
    brightThreshold: number;
    autoTheme: boolean;
    contrastBoost: boolean;
  };
  motion?: {
    reduceMotion: boolean;
    simplifyUI: boolean;
    largerTargets: boolean;
  };
  attention?: {
    dimAfterMs: number;
    dimLevel: number;
    pauseAnimations: boolean;
  };
  time?: {
    nightModeStart: number;
    nightModeEnd: number;
    reduceBlueLight: boolean;
  };
  custom?: CustomAdaptation[];
}
```

Browser Support

Some features require sensor APIs with limited browser support:

Feature	Chrome	Firefox	Safari
Ambient Light	Flag	No	No
Device Motion	Yes	Yes	Yes
Proximity	No	No	No
Audio Environment	Yes	Yes	Yes

Fallbacks are provided where native APIs are unavailable.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-ambient/src/index.ts

Public API

- Direct exports: AdaptationRules, AdaptiveUI, AmbientCSS, AmbientCallback, AmbientContext, AmbientContextManager, AttentionAdaptation, AttentionState, AudioEnvironment, CustomAdaptation, DevicePosture, LightAdaptation, LightConditions, MotionAdaptation, MotionState, ProximityState, TimeAdaptation, TimeContext, useAdaptiveUI, useAmbientContext, useAttentionState, useAudioEnvironment, useLightConditions, useMotionState
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/analytics - Type: Node package - Purpose: Privacy-First Analytics - GDPR/CCPA compliant, no third-party scripts, client-side processing - Version: 0.1.0 - Location: packages/philjs-analytics - Entry points: .., packages/philjs-analytics/src/privacy-first.ts - Keywords: philjs, analytics, privacy, gdpr, ccpa, no-cookies, first-party, web-vitals, differential-privacy

@philjs/analytics

Privacy-First Analytics for PhilJS Applications

Overview

Built-in analytics that respect user privacy by default. No third-party scripts, no cookies, GDPR/CCPA compliant by design.

Features: - No third-party scripts or cookies - Client-side or edge processing - GDPR/CCPA compliant by design - Differential privacy for sensitive metrics - HyperLogLog for unique visitor estimates - Core Web Vitals tracking (LCP, FID, CLS, INP, TTFB, FCP) - K-anonymity aggregation threshold - Consent management built-in

Installation

```
npm install @philjs/analytics
```

Quick Start

```
import { initAnalytics, useAnalytics } from '@philjs/analytics';

// Initialize analytics
const analytics = initAnalytics({
  enabled: true,
  respectDoNotTrack: true,
  trackWebVitals: true,
  trackPageViews: true
});

// Track custom events
analytics.track('button_click', { buttonId: 'signup' });

// Track page views
analytics.trackPageView('/dashboard');
```

Usage

Basic Configuration

```
import { PrivacyFirstAnalytics } from '@philjs/analytics';

const analytics = new PrivacyFirstAnalytics({
  enabled: true,
  respectDoNotTrack: true,
  differentialPrivacy: true,
  privacyBudget: 1.0,
  aggregateFirst: true,
  minAggregationCount: 5,
  retentionPeriod: 7 * 24 * 60 * 60 * 1000, // 7 days
  requireConsent: true,
  trackWebVitals: true,
  trackPageViews: true,
  trackClicks: true,
  trackScroll: true,
  trackErrors: true
});
```

Consent Management

```
import { useConsent } from '@philjs/analytics';

const { consent, setConsent, isConsentRequired } = useConsent();

// Set user consent
setConsent({
  analytics: true,
  performance: true,
  errors: true
});
```

Get Aggregated Metrics

```
const metrics = analytics.getMetrics();

console.log('Unique visitors:', metrics.uniqueVisitors);
console.log('Top pages:', metrics.topPages);
console.log('Web Vitals:', metrics.webVitals);
console.log('Bounce rate:', metrics.bounceRate);
```

Hooks API

```
import { useAnalytics } from '@philjs/analytics';

function MyComponent() {
  const { track, trackPageView, getMetrics } = useAnalytics();

  track('feature_used', { feature: 'export' });
  trackPageView('/reports');

  const metrics = getMetrics();
}
```

API Reference

Classes

PrivacyFirstAnalytics

Main analytics engine with privacy-preserving features.

Methods: - track(eventName, data?) - Track custom events - trackPageView(page?) - Track page views - setConsent(consent) - Set user consent state - getConsent() - Get current consent state - getMetrics() - Get aggregated metrics - exportData() - Export anonymized data as JSON - clearData() - Clear all stored data - destroy() - Cleanup and remove listeners

Functions

- initAnalytics(config?) - Initialize global analytics instance
- getAnalytics() - Get the global analytics instance
- resetAnalytics() - Reset analytics (for testing)
- useAnalytics() - Hook for analytics operations
- useConsent() - Hook for consent management

Types

```

interface PrivacyConfig {
  enabled: boolean;
  respectDoNotTrack: boolean;
  differentialPrivacy: boolean;
  privacyBudget: number;
  aggregateFirst: boolean;
  minAggregationCount: number;
  retentionPeriod: number;
  requireConsent: boolean;
  storageKey: string;
  endpoint?: string;
  trackWebVitals: boolean;
  trackPageViews: boolean;
  trackClicks: boolean;
  trackScroll: boolean;
  trackErrors: boolean;
}

interface AggregatedMetrics {
  periodStart: number;
  periodEnd: number;
  pageviews: Map<string, number>;
  uniqueVisitors: number;
  avgSessionDuration: number;
  bounceRate: number;
  topPages: { page: string; count: number }[];
  webVitals: WebVitalsAggregate;
  errorCounts: Map<string, number>;
  devices: { mobile: number; tablet: number; desktop: number };
}

interface ConsentState {
  analytics: boolean;
  performance: boolean;
  errors: boolean;
  timestamp: number;
  version: number;
}

```

Privacy Features

- **Differential Privacy:** Adds calibrated noise to protect individual contributions
- **K-Anonymity:** Only reports metrics meeting minimum aggregation thresholds
- **PII Sanitization:** Automatically removes emails, phone numbers from tracked data
- **Do Not Track:** Respects browser DNT and Global Privacy Control signals
- **Path Anonymization:** Removes IDs from URL paths (e.g., /users/123 -> /users/[id])
- **Timestamp Rounding:** Rounds timestamps to nearest minute

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .. ./privacy-first
- Source files: packages/philijs-analytics/src/privacy-first.ts

Public API

- Direct exports: AggregatedMetrics, AnalyticsEvent, ConsentState, EventType, PrivacyConfig, PrivacyFirstAnalytics, WebVitalsAggregate, getAnalytics, initAnalytics, resetAnalytics, useAnalytics, useConsent
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philijs/api - Type: Node package - Purpose: API routes and full-stack utilities for PhilJS - Version: 0.1.0 - Location: packages/philijs-api - Entry points: packages/philijs-api/src/index.ts, packages/philijs-api/src/server.ts, packages/philijs-api/src/client.ts, packages/philijs-api/src/cookies.ts, packages/philijs-api/src/session.ts, packages/philijs-api/src/flash.ts, packages/philijs-api/src/cookie-session.ts, packages/philijs-api/src/session-utils.ts, packages/philijs-api/src/edge-middleware.ts, packages/philijs-api/src/geolocation.ts, packages/philijs-api/src/edge-ab-testing.ts, packages/philijs-api/src/edge-cache.ts, packages/philijs-api/src/server-actions.ts - Keywords: philijs, api, routes, server, full-stack - Book coverage: [api/edge-middleware.md](#), [api/sessions.md](#), [api/session-migration.md](#)

philijs-api

API routes and full-stack utilities for PhilJS applications.

Features

- **Type-safe API routes** - End-to-end type safety from server to client
- **File-based routing** - Automatic API route generation
- **Middleware support** - Request/response interceptors
- **Request validation** - Schema validation with Zod
- **Session management** - Built-in session and cookie utilities
- **Flash messages** - Remix-style one-time messages with auto-cleanups
- **Enhanced cookie sessions** - Encrypted, signed, CSRF-protected sessions

- **Session utilities** - Timeout, validation, rotation middleware
- **CORS handling** - Cross-origin resource sharing support
- **Rate limiting** - Protect your API from abuse
- **WebSocket support** - Real-time communication

Installation

```
pnpm add philjs-api
```

Quick Start

Create an API Route

Create `src/routes/api/users.ts`:

```
import { defineAPIRoute, json } from 'philjs-api';
import { z } from 'zod';

const UserSchema = z.object({
  name: z.string(),
  email: z.string().email()
});

export default defineAPIRoute({
  GET: async ({ request }) => {
    const users = await db.user.findMany();
    return json({ users });
  },
  POST: async ({ request }) => {
    const body = await request.json();
    const data = UserSchema.parse(body);

    const user = await db.user.create({ data });
    return json({ user }, { status: 201 });
  }
});
```

Call from Client

```
import { api } from 'philjs-api/client';

// Type-safe API calls
const { users } = await api.get('/api/users');
const { user } = await api.post('/api/users', {
  name: 'Alice',
  email: 'alice@example.com'
});
```

Sessions and Cookies

```
import { defineAPIRoute, createSession } from 'philjs-api';

const session = createSession({
  secret: process.env.SESSION_SECRET!,
  maxAge: 60 * 60 * 24 * 7 // 7 days
});

export default defineAPIRoute({
  POST: async ({ request }) => {
    const { email, password } = await request.json();
    const user = await authenticateUser(email, password);

    // Set session
    await session.set(request, 'userId', user.id);

    return json({ success: true });
  }
});
```

Middleware

```
import { defineAPIRoute, withAuth } from 'philjs-api';

export default defineAPIRoute({
  GET: withAuth(async ({ request, user }) => {
    // user is automatically available
    return json({ user });
  })
});
```

Advanced Features

Flash Messages

One-time messages that persist across redirects:

```

import { setFlashSuccess, getFlashMessages } from 'philjs-api/flash';

// Set flash message
setFlashSuccess(session, 'Profile updated successfully!');

// Get and auto-clear
const messages = getFlashMessages(session);

```

Enhanced Cookie Sessions

Secure sessions with encryption and CSRF protection:

```

import { createCookieSessionStorage } from 'philjs-api/cookie-session';

const sessionStorage = createCookieSessionStorage({
  secret: process.env.SESSION_SECRET!,
  encryptionSecret: process.env.ENCRYPTION_SECRET!,
  csrf: true,
  rotate: true,
});

```

Session Utilities

Helpers and middleware for session management:

```

import { sessionMiddleware, sessionTimeoutMiddleware } from 'philjs-api/session-utils';

const middleware = sessionMiddleware({
  storage: sessionStorage,
  autoCommit: true,
});

```

See [FLASH_SESSIONS.md](#) for complete documentation.

Documentation

For more information, see the PhilJS documentation.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: ., ./server, ./client, ./cookies, ./session, ./flash, ./cookie-session, ./session-utils, ./edge-middleware, ./geolocation, ./edge-ab-testing, ./edge-cache, ./server-actions
- Source files: packages/philjs-api/src/index.ts, packages/philjs-api/src/server.ts, packages/philjs-api/src/client.ts, packages/philjs-api/src/cookies.ts, packages/philjs-api/src/session.ts, packages/philjs-api/src/flash.ts, packages/philjs-api/src/cookie-session.ts, packages/philjs-api/src/session-utils.ts, packages/philjs-api/src/edge-middleware.ts, packages/philjs-api/src/geolocation.ts, packages/philjs-api/src/edge-ab-testing.ts, packages/philjs-api/src/edge-cache.ts, packages/philjs-api/src/server-actions.ts

Public API

- Direct exports: ABTestingOptions, APIClientOptions, APIContext, APIError, APIHandler, APIRequest, APIResponse, ActionResult, AnalyticsProvider, CacheControlOptions, CacheEntry, CacheOptions, CacheStore, CloudflareCacheOptions, CloudflareProvider, CloudflareProxyProvider, CookieOptions, CookieSerializeOptions, CookieSessionOptions, CookieSessionStorage, CookieStore, CreateServerActionOptions, DEFAULT_LANGUAGE_MAP, DenoDeployProvider, EdgeContext, EdgeMiddleware, EdgeMiddlewareConfig, EdgeRequest, Experiment, ExperimentAssignment, FetchOptions, FlashCategory, FlashMessage, FlashMessageWithToast, FlashSessionData, GeoLanguageMapping, GeoLocationProvider, GeoRedirectRule, GeolocationData, GeolocationOptions, GoogleAnalyticsProvider, MultivariateExperiment, RequestWithSession, ResponseHelpers, RouteHandler, SerializableValue, ServerAction, ServerActionError, ServerActionFn, ServerActionInputType, ServerActionReturnType, ServerActionState, SessionData, SessionMiddlewareOptions, SessionOptions, SessionStorage, TargetingRules, ToastOptions, TypedSession, UseFormActionReturn, UseOptimisticOptions, UseOptimisticReturn, UseServerActionOptions, UseServerActionReturn, Variant, VariantStats, VercelProvider, abTestingMiddleware, addHeadersMiddleware, apiCache, apiClient, applySessionToResponse, badRequest, cacheControlMiddleware, calculateSignificance, clearFlashMessages, clearSessionData, cloudflareCacheMiddleware, commitSession, composeEdgeMiddleware, createApiClient, createAPIContext, createAPIHandler, createAnalyticsProvider, createCSRFInput, createCookieJar, createCookieSessionStorage, createFlashUtils, createMemorySessionStorage, createServerAction, createSessionStorage, createSignedCookie, createTypedSessionUtils, csrfMiddleware, defineAPIRoute, defineEdgeMiddleware, deleteCookie, deserialize, deserializeFlashMessages, deserializeFormData, destroySession, detectGeolocation, detectLanguage, detectLanguageFromGeo, detectLanguageFromHeader, edgeCacheMiddleware, etagMiddleware, executeEdgeMiddleware, flashMiddleware, forbidden, generateCSRFToken, generateCacheKey, generateETag, geoDistance, geoRedirectMiddleware, geolocationMiddleware, getAction, getActiveExperiments, getAllActions, getClientCSRFToken, getCookie, getFlashMessages, getFlashMessagesByCategory, getOrCreateSession, getSession, getSessionValue, handleServerActionRequest, hasFlashMessages, html, injectGeolocationData, injectVariantData, isServerAction, isVariant, isWithinRadius, json, languageDetectionMiddleware, lastModifiedMiddleware, localizedRedirectMiddleware, matchesPattern, mergeSessionData, multivariateTestingMiddleware, notFound, pageCache, parseBody, parseCookies, peekFlashMessages, purgeAllCache, purgeCacheKey, purgeCacheTags, redirect, redirectByCountry, redirectMiddleware, regenerateSession, registerAction, removeHeadersMiddleware, requireSession, revalidatePath, revalidateTag, rewriteMiddleware, securityHeadersMiddleware, selectVariantDeterministic, serialize, serializeCookie, serializeFlashMessages, serializeFormData, serverActions, serverActionsExpressMiddleware, serverActionsMiddleware, serverError, sessionMiddleware, sessionRotationMiddleware, sessionTimeoutMiddleware, sessionValidatorMiddleware, setCSRFSecret, setClientCSRFToken, setCookie, setFlash, setFlashError, setFlashInfo, setFlashSuccess, setFlashWarning, setValidationCallback, setSessionValue, staticAssetCache, text, unauthorized, useFetch, useFlash, useFormAction, useGeolocation, useMutation, useOptimistic, useServerAction, useVariant, variantInjectionMiddleware, variantMiddleware, variantRewriteMiddleware, varyMiddleware, verifyCSRFToken, verifySignedCookie, withCSRF
- Re-exported names: // Action creation createServerAction, // CSRF Protection setCSRFSecret, // Errors ServerActionError, // Hooks useServerAction, // Progressive enhancement createCSRFInput, // Request handling handleServerActionRequest, // Revalidation revalidatePath, // Serialization serialize, ABTest, ABTestMiddlewareOptions, ABTestVariant, ABTestingOptions, APIClientOptions, APIContext, APIHandler, APIRequest, APIResponse, ActionResult, AnalyticsProvider, CORSOptions, CacheControlOptions, CacheEntry, CacheOptions, CacheStore, CloudflareCacheOptions, CloudflareProvider, CloudflareProxyProvider, CompressionOptions, CookieOptions, CookieSerializeOptions, CookieSessionOptions, CookieSessionStorage, CookieStore, CreateServerActionOptions, DEFAULT_LANGUAGE_MAP, DenoDeployProvider, EdgeContext, EdgeMiddleware, EdgeMiddlewareConfig, EdgeRequest, Experiment, ExperimentAssignment, FetchOptions, FlashCategory, FlashMessage, FlashMessageWithToast, FlashSessionData, GeoLanguageMapping, GeoLocationProvider, GeoRedirectRule, GeolocationData, GeolocationMiddlewareOptions, GeolocationOptions, GoogleAnalyticsProvider, Middleware, MultivariateExperiment, RateLimitOptions, RateLimitStore, RequestOptions, RequestOptions, RequestWithSession, RouteHandler, SecurityHeadersOptions, SerializableValue, ServerAction, ServerActionFn, ServerActionInputType, ServerActionReturnType, ServerActionState, SessionData, SessionMiddlewareOptions, SessionOptions, SessionStorage, TargetingRules, ToastOptions, TypedSession, UseFormActionReturn, UseOptimisticOptions, UseOptimisticReturn, UseServerActionOptions, UseServerActionReturn, ValidationErrors, ValidationResult, ValidationSchema, Variant, VariantStats, VercelProvider,

abTestMiddleware, addHeadersMiddleware, apiCache, apiClient, applySessionToResponse, badRequest, cacheControlMiddleware, calculateSignificance, clearFlashMessages, clearSessionData, cloudflareCacheMiddleware, commitSession, commitSessionHelper, composeEdgeMiddleware, composeMiddleware, compressionMiddleware, conditionalMiddleware, corsMiddleware, createApiClient, createAPIHandler, createAnalyticsProvider, createCookieSessionStorage, createEnhancedCookieSessionStorage, createFlashUtils, createMemorySessionStorage, createSessionStorage, createSignedCookie, createTypedSessionUtils, createValidator, csrfMiddleware, defineAPIRoute, defineEdgeMiddleware, deleteCookie, deserialize, deserializeFlashMessages, deserializeFormData, destroySession, destroySessionHelper, detectGeolocation, detectLanguage, detectLanguageFromGeo, detectLanguageFromHeader, edgeABTestingMiddleware, edgeCacheMiddleware, edgeGeolocationMiddleware, edgeSecurityHeadersMiddleware, etagMiddleware, executeEdgeMiddleware, flashMiddleware, forbidden, generateCSRFToken, generateCacheKey, generateETag, geoDistance, geoRedirectMiddleware, geolocationMiddleware, getAction, getActiveExperiments, getAllActions, getClientCSRFToken, getCookie, getEnv, getFlashMessages, getFlashMessagesByCategory, getOrCreateSession, getPublicEnv, getSession, getSessionValue, hasFlashMessages, html, injectGeolocationData, injectVariantData, isServerAction, isVariant, isWithinRadius, json, languageDetectionMiddleware, lastModifiedMiddleware, localizedRedirectMiddleware, matchesPattern, mergeSessionData, multivariateTestingMiddleware, notFound, pageCache, parseCookies, peekFlashMessages, purgeAllCache, purgeCacheKey, purgeCacheTags, rateLimitMiddleware, redirect, redirectByCountry, redirectMiddleware, regenerateSession, registerAction, removeHeadersMiddleware, requestIDMiddleware, requireEnv, requireSession, revalidateTag, rewriteMiddleware, securityHeadersMiddleware, selectVariantDeterministic, serializeCookie, serializeFlashMessages, serializeFormData, serverAction, serverActionsExpressMiddleware, serverActionsMiddleware, serverError, sessionMiddleware, sessionRotationMiddleware, sessionTimeoutMiddleware, sessionValidatorMiddleware, setClientCSRFToken, setCookie, setFlash, setFlashError, setFlashInfo, setFlashSuccess, setFlashWarning, setRevalidationCallback, setSessionValue, staticAssetCache, text, unauthorized, useFetch, useFormAction, useGeolocation, useMutation, useOptimistic, useVariant, validate, variantInjectionMiddleware, variantMiddleware, variantRewriteMiddleware, varyMiddleware, verifyCSRFToken, verifySignedCookie, withCSR

- Re-exported modules: ./client.js, ./cookie-session.js, ./cookies.js, ./edge-ab-testing.js, ./edge-cache.js, ./edge-middleware.js, ./env.js, ./flash.js, ./geolocation.js, ./middleware.js, ./server-actions.js, ./server.js, ./session-utils.js, ./session.js, ./validation.js

License

MIT

@philjs/atoms - Type: Node package - Purpose: Jotai-style atomic state management for PhilJS with signals - Version: 0.1.0 - Location: packages/philjs-atoms - Entry points: packages/philjs-atoms/src/index.ts - Keywords: philjs, state-management, atoms, jotai, signals, reactive

philjs-atoms

Jotai-style atomic state management for PhilJS with signal-based reactivity.

Features

- Primitive atoms (read/write state)
- Derived atoms (computed values)
- Async atoms (promises)
- Atom families (parameterized atoms)
- Write-only atoms (actions)
- Storage persistence
- Fine-grained reactivity
- TypeScript support
- Minimal bundle size

Installation

```
npm install philjs-atoms philjs-core
```

Basic Usage

Primitive Atoms

```
import { atom, useAtom, useAtomValue, useSetAtom } from 'philjs-atoms';

// Create an atom
const countAtom = atom(0);

// Read and write
function Counter() {
  const [count, setCount] = useAtom(countAtom);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(c => c + 1)}>Increment</button>
    </div>
  );
}

// Read-only
function Display() {
  const count = useAtomValue(countAtom);
  return <p>{count}</p>;
}

// Write-only
function Controls() {
  const setCount = useSetAtom(countAtom);
  return <button onClick={() => setCount(0)}>Reset</button>;
}
```

Derived Atoms (Computed)

Read-Only Derived Atoms

```

const countAtom = atom(0);
const doubledAtom = atom((get) => get(countAtom) * 2);

function Display() {
  const doubled = useAtomValue(doubledAtom);
  return <p>Doubled: {doubled}</p>;
}

```

Writable Derived Atoms

```

const celsiusAtom = atom(0);

const fahrenheitAtom = atom(
  (get) => get(celsiusAtom) * 1.8 + 32,
  (get, set, newValue: number) => {
    set(celsiusAtom, (newValue - 32) / 1.8);
  }
);

function Temperature() {
  const [fahrenheit, setFahrenheit] = useAtom(fahrenheitAtom);

  return (
    <div>
      <input
        type="number"
        value={fahrenheit}
        onChange={(e) => setFahrenheit(Number(e.target.value))}>
      />
      <span>F</span>
    </div>
  );
}

```

Combining Multiple Atoms

```

const firstNameAtom = atom('John');
const lastNameAtom = atom('Doe');

const fullNameAtom = atom(
  (get) => `${get(firstNameAtom)} ${get(lastNameAtom)}`
);

function FullName() {
  const fullName = useAtomValue(fullNameAtom);
  return <p>{fullName}</p>;
}

```

Async Atoms

```

import { asyncAtom } from 'philjs-atoms';

const userIdAtom = atom(1);

const userAtom = asyncAtom(async (get) => {
  const id = get(userIdAtom);
  const response = await fetch(`/api/users/${id}`);
  return response.json();
});

function UserProfile() {
  if (userAtom.loading()) {
    return <div>Loading...</div>;
  }

  if (userAtom.error()) {
    return <div>Error: {userAtom.error()?.message}</div>;
  }

  const user = useAtomValue(userAtom);
  return <div>User: {user.name}</div>;
}

```

Loadable Pattern (Doesn't Throw)

```

import { loadable } from 'philjs-atoms';

const loadableUserAtom = loadable(userAtom);

function UserProfile() {
  const loadableUser = useAtomValue(loadableUserAtom);

  if (loadableUser.state === 'loading') {
    return <div>Loading...</div>;
  }

  if (loadableUser.state === 'hasError') {
    return <div>Error: {loadableUser.error.message}</div>;
  }

  return <div>User: {loadableUser.data.name}</div>;
}

```

Atom Families

Create parameterized atoms for collections:

```

import { atomFamily } from 'philjs-atoms';

interface Todo {
  id: number;
  text: string;
  completed: boolean;
}

const todoAtomFamily = atomFamily((id: number) =>
  atom<Todo>({
    id,
    text: '',
    completed: false,
  })
);

function TodoItem({ id }: { id: number }) {
  const [todo, setTodo] = useAtom(todoAtomFamily(id));

  return (
    <div>
      <input
        type="checkbox"
        checked={todo.completed}
        onChange={() => setTodo({ ...todo, completed: !todo.completed })}
      />
      <span>{todo.text}</span>
    </div>
  );
}

// Clean up when no longer needed
todoAtomFamily.remove(id);

```

Write-Only Atoms (Actions)

```

import { atomAction } from 'philjs-atoms';

const countAtom = atom(0);

const incrementAtom = atomAction((get, set) => {
  const current = get(countAtom);
  set(countAtom, current + 1);
});

const decrementAtom = atomAction((get, set) => {
  const current = get(countAtom);
  set(countAtom, current - 1);
});

const resetAtom = atomAction((get, set) => {
  set(countAtom, 0);
});

function Controls() {
  const increment = useSetAtom(incrementAtom);
  const decrement = useSetAtom(decrementAtom);
  const reset = useSetAtom(resetAtom);

  return (
    <div>
      <button onClick={increment}>+</button>
      <button onClick={decrement}>-</button>
      <button onClick={reset}>Reset</button>
    </div>
  );
}

```

Utilities

atomWithReset

```

import { atomWithReset, useResetAtom } from 'philjs-atoms';

const countAtom = atomWithReset(0);

function Counter() {
  const [count, setCount] = useAtom(countAtom);
  const reset = useResetAtom(countAtom);

  return (
    <div>
      <p>{count}</p>
      <button onClick={() => setCount(c => c + 1)}>+</button>
      <button onClick={reset}>Reset</button>
    </div>
  );
}

```

atomWithStorage (Persistence)

```

import { atomWithStorage } from 'philjs-atoms';

const themeAtom = atomWithStorage('theme', 'light');
const settingsAtom = atomWithStorage('settings', {
  notifications: true,
  sound: false,
});

function ThemeToggle() {
  const [theme, setTheme] = useAtom(themeAtom);

  return (
    <button onClick={() => setTheme(theme === 'light' ? 'dark' : 'light')}>
      Toggle Theme
    </button>
  );
}

```

selectAtom (Property Selection)

```

import { selectAtom } from 'philjs-atoms';

const userAtom = atom({
  name: 'John',
  age: 30,
  email: 'john@example.com',
});

const nameAtom = selectAtom(userAtom, (user) => user.name);
const ageAtom = selectAtom(userAtom, (user) => user.age);

function UserName() {
  const name = useAtomValue(nameAtom);
  return <span>{name}</span>;
}

```

focusAtom (Property Focus)

```

import { focusAtom } from 'philjs-atoms';

const userAtom = atom({
  name: 'John',
  age: 30,
});

const nameAtom = focusAtom(userAtom, (user) => user.name);

function NameEditor() {
  const [name, setName] = useAtom(nameAtom);

  return (
    <input
      value={name}
      onChange={(e) => setName(e.target.value)}
    />
  );
}

```

splitAtom (Separate Read/Write)

```

import { splitAtom } from 'philjs-atoms';

const userAtom = atom({ name: 'John', age: 30 });
const [readUserAtom, writeUserAtom] = splitAtom(userAtom);

function Display() {
  const user = useAtomValue(readUserAtom);
  return <div>{user.name}</div>;
}

function Editor() {
  const setUser = useSetAtom(writeUserAtom);
  return <button onClick={() => setUser({ name: 'Jane', age: 25 })}>Update</button>;
}

```

batchAtoms (Batch Updates)

```

import { batchAtoms } from 'philjs-atoms';

const countAtom = atom(0);
const nameAtom = atom('');
const ageAtom = atom(0);

function updateAll() {
  batchAtoms(() => {
    useSetAtom(countAtom)(10);
    useSetAtom(nameAtom)('John');
    useSetAtom(ageAtom)(30);
  }); // Only triggers one render
}

```

Advanced Patterns

Shopping Cart Example

```

const cartItemsAtom = atom<CartItem[]>([]);

const cartTotalAtom = atom((get) => {
  const items = get(cartItemsAtom);
  return items.reduce((sum, item) => sum + item.price * item.quantity, 0);
});

const addToCartAtom = atomAction((get, set, item: CartItem) => {
  const items = get(cartItemsAtom);
  const existingItem = items.find((i) => i.id === item.id);

  if (existingItem) {
    set(
      cartItemsAtom,
      items.map((i) =>
        i.id === item.id ? { ...i, quantity: i.quantity + 1 } : i
      )
    );
  } else {
    set(cartItemsAtom, [...items, { ...item, quantity: 1 }]);
  }
});

const removeFromCartAtom = atomAction((get, set, itemId: number) => {
  const items = get(cartItemsAtom);
  set(cartItemsAtom, items.filter((i) => i.id !== itemId));
});

```

Form State Example

```

const formAtom = atom({
  email: '',
  password: '',
  rememberMe: false,
});

const emailAtom = focusAtom(formAtom, (form) => form.email);
const passwordAtom = focusAtom(formAtom, (form) => form.password);

const isValidAtom = atom((get) => {
  const form = get(formAtom);
  return form.email.includes('@') && form.password.length >= 8;
});

const submitFormAtom = atomAction(async (get, set) => {
  const form = get(formAtom);
  const response = await fetch('/api/login', {
    method: 'POST',
    body: JSON.stringify(form),
  });
  const data = await response.json();
  // Handle response...
});

```

TypeScript

Full TypeScript support with type inference:

```

import { atom, Atom, WritableAtom } from 'philjs-atoms';

interface User {
  id: number;
  name: string;
  email: string;
}

const userAtom: Atom<User> = atom({
  id: 1,
  name: 'John',
  email: 'john@example.com',
});

const userNameAtom: Atom<string> = atom((get) => get(userAtom).name);

```

API Reference

`atom(initialValue)`

Create a primitive atom.

`atom(read)`

Create a read-only derived atom.

`atom(read, write)`

Create a writable derived atom.

`useAtomValue(atom)`

Read atom value.

```

useSetAtom(atom)
Get setter function.

useAtom(atom)
Read and write atom.

asyncAtom(read)
Create async atom.

loadable(atom)
Wrap async atom (doesn't throw).

atomFamily(initialize)
Create parameterized atoms.

atomAction(write)
Create write-only atom.

atomWithReset(initialValue)
Create atom with reset capability.

atomWithStorage(key, initialValue, storage?)
Create atom with localStorage persistence.

selectAtom(atom, selector)
Select property from atom.

focusAtom(atom, focus)
Focus on atom property (read/write).

splitAtom(atom)
Split into read and write atoms.

batchAtoms(fn)
Batch multiple updates.

```

Performance Tips

1. Use derived atoms for computed values
2. Use atom families for collections
3. Use selectAtom to prevent unnecessary re-renders
4. Batch updates with batchAtoms
5. Use write-only atoms for actions
6. Keep atoms small and focused

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philijs-atoms/src/index.ts

Public API

- Direct exports: AsyncAtom, Atom, AtomFamily, Getter, PrimitiveAtom, SetStateAction, Setter, WritableAtom, asyncAtom, atom, atomAction, atomFamily, atomWithReset, atomWithStorage, batchAtoms, focusAtom, freezeAtom, loadable, selectAtom, splitAtom, useAtom, useAtomValue, useResetAtom, useSetAtom
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

```
## @philijs/auth - Type: Node package - Purpose: Authentication and authorization utilities for PhilJS - Version: 0.1.0 - Location: packages/philijs-auth - Entry points: packages/philijs-auth/src/index.ts, packages/philijs-auth/src/oauth.ts, packages/philijs-auth/src/jwt.ts, packages/philijs-auth/src/session.ts, packages/philijs-auth/src/session-refresh.ts, packages/philijs-auth/src/protected-route.ts, ./protected-routes, packages/philijs-auth/src/hooks.ts, packages/philijs-auth/src/auth-provider.ts, packages/philijs-auth/src/providers/clerk.ts, packages/philijs-auth/src/providers/auth0.ts, packages/philijs-auth/src/providers/supabase.ts, packages/philijs-auth/src/providers/next-auth.ts - Keywords: philijs, auth, authentication, oauth, jwt, session - Book coverage: auth/guide.md
```

PhilJS Auth

Comprehensive authentication and authorization system for PhilJS applications.

Requirements

- **Node.js 24** or higher
- **TypeScript 6** or higher
- **ESM only** - CommonJS is not supported

Features

- ** Multiple Provider Support**: Clerk, Auth0, Supabase, NextAuth, or custom
- ** CLI Generators**: RedwoodJS-style generators for rapid setup
- ** Automatic Token Refresh**: Smart token refresh with configurable strategies
- ** Protected Routes**: HOCs, components, and hooks for route protection
- ** Role-Based Access**: Built-in permission and role checking

- ** Session Management**: Reactive session state using signals
- ** UI Components**: Pre-built, customizable auth UI components
- ** JWT Support**: Full JWT token management
- ** OAuth Integration**: Support for multiple OAuth providers
- ** Type-Safe**: Full TypeScript support with excellent DX

Installation

```
pnpm add philjs-auth
```

Quick Start

1. Generate Auth Setup (Recommended)

Use the PhilJS CLI to generate a complete authentication setup:

```
# Generate Clerk authentication
philjs generate auth clerk

# Generate Auth0 authentication
philjs generate auth auth0

# Generate Supabase authentication
philjs generate auth supabase

# Generate NextAuth authentication
philjs generate auth nextauth

# Generate custom authentication
philjs generate auth custom
```

This creates: - Authentication configuration - Provider integration - Login/Signup/Password Reset forms - Profile management - Protected route utilities - Example pages

2. Manual Setup

```
import { CustomAuthProvider, setAuthProvider, startSessionRefresh } from 'philjs-auth';

// Create provider
const authProvider = new CustomAuthProvider({
  apiUrl: 'http://localhost:3000/api',
  tokenKey: 'auth_token',
  refreshTokenKey: 'refresh_token',
});

// Initialize
await authProvider.initialize();

// Set as global provider
setAuthProvider(authProvider);

// Start automatic token refresh
startSessionRefresh({
  refreshBeforeExpiry: 5 * 60 * 1000, // 5 minutes
  checkInterval: 60 * 1000, // 1 minute
});
```

3. Wrap Your App

```
import { AuthProvider } from './auth/AuthProvider';

export function App() {
  return (
    <AuthProvider>
      <YourApp />
    </AuthProvider>
  );
}
```

Usage

Authentication Hooks

```

import { useAuth, useUser, useHasPermission } from 'philjs-auth/hooks';

function MyComponent() {
  const { signIn, signOut, isAuthenticated, isLoading } = useAuth();
  const user = useUser();
  const isAdmin = useHasPermission('admin');

  const handleLogin = async () => {
    try {
      await signIn('user@example.com', 'password');
      window.location.href = '/dashboard';
    } catch (error) {
      console.error('Login failed:', error);
    }
  };

  if (isLoading) {
    return <div>Loading...</div>;
  }

  return (
    <div>
      {isAuthenticated ? (
        <>
          <p>Welcome, {user?.name}!</p>
          <button onClick={signOut}>Logout</button>
          {isAdmin && <a href="/admin">Admin Panel</a>}
        </>
      ) : (
        <button onClick={handleLogin}>Login</button>
      )}
    </div>
  );
}

```

Protected Routes

```

import { ProtectedRoute, withRole } from 'philjs-auth/protected-routes';

// Component-based protection
function Dashboard() {
  return (
    <ProtectedRoute redirectTo="/login">
      <div>Dashboard content</div>
    </ProtectedRoute>
  );
}

// HOC-based protection
const AdminPanel = withRole(AdminPanelComponent, {
  role: 'admin',
  redirectTo: '/unauthorized',
});

// Conditional rendering
import { ShowForAuth, ShowForRole } from 'philjs-auth/protected-routes';

function Navigation() {
  return (
    <nav>
      <ShowForAuth>
        <a href="/dashboard">Dashboard</a>
      </ShowForAuth>
      <ShowForRole role="admin">
        <a href="/admin">Admin</a>
      </ShowForRole>
    </nav>
  );
}

```

Session Management

```

import {
  startSessionRefresh,
  SessionPersistence,
  logoutEverywhere
} from 'philjs-auth/session-refresh';

// Automatic token refresh
startSessionRefresh({
  refreshBeforeExpiry: 5 * 60 * 1000,
  checkInterval: 60 * 1000,
  refreshOnFocus: true,
  refreshOnReconnect: true,
  onRefreshFailed: (error) => {
    console.error('Token refresh failed:', error);
  },
});

// Session persistence
SessionPersistence.save(session, 'local');
const session = SessionPersistence.load('local');

// Logout everywhere
await logoutEverywhere();

```

Providers

Clerk

```

npm install @clerk/clerk-react
philjs generate auth clerk

```

Auth0

```

npm install @auth0/auth0-react
philjs generate auth auth0

```

Supabase

```

npm install @supabase/supabase-js
philjs generate auth supabase

```

NextAuth

```

npm install next-auth
philjs generate auth nextauth

```

Custom

Create your own authentication provider:

```

import { BaseAuthProvider } from 'philjs-auth';
import { signal } from 'philjs-core/signals';

class MyAuthProvider extends BaseAuthProvider {
  readonly name = 'my-auth';
  readonly user = signal<User | null>(null);
  readonly session = signal<AuthSession | null>(null);
  readonly loading = signal(false);

  async initialize(): Promise<void> {
    // Initialize your auth system
  }

  async signInWithEmail(email: string, password: string): Promise<User> {
    // Implement sign in logic
  }

  // ... implement other required methods
}

```

API Reference

Hooks

- `useAuth()` - Main authentication hook
- `useUser()` - Get current user
- `useSession()` - Get current session
- `useIsAuthenticated()` - Check if authenticated
- `useAuthLoading()` - Check loading state
- `useHasPermission(permission)` - Check permissions/roles
- `useRequireAuth(redirectTo)` - Require authentication
- `useAccessToken()` - Get access token

Protected Routes

- `<ProtectedRoute>` - Protect route component

- `withAuth(Component)` - Protect with HOC
- `withRole(Component, options)` - Role-based protection
- `withAnyRole(Component, options)` - Multiple role protection
- `<AuthGuard>` - Conditional rendering
- `<ShowForAuth>` - Show for authenticated users
- `<ShowForGuest>` - Show for guests
- `<ShowForRole>` - Show for specific role

Session Management

- `startSessionRefresh(config)` - Start auto refresh
- `stopSessionRefresh()` - Stop auto refresh
- `SessionPersistence` - Session storage utilities
- `logoutEverywhere()` - Revoke all sessions

Providers

- `ClerkAuthProvider` - Clerk integration
- `Auth0AuthProvider` - Auth0 integration
- `SupabaseAuthProvider` - Supabase integration
- `NextAuthProvider` - NextAuth integration
- `BaseAuthProvider` - Base class for custom providers

Legacy APIs

The package also includes legacy session management and JWT utilities:

- `SessionManager` - Original session manager
- `JWTManager` - JWT token management
- `OAuthManager` - OAuth provider management
- Original `useAuth()` from `protected-route.js`

See the full documentation for complete API reference.

Documentation

- [Authentication Guide](#) - Complete guide
- [Examples](#) - Working examples

Generator Options

```
philjs generate auth <provider> [options]

Options:
  -d, --directory <dir>      Target directory (default: "src")
  --no-ui                  Skip UI components
  --no-middleware          Skip middleware
  --no-protected-routes    Skip protected route utilities
  --js                      Use JavaScript instead of TypeScript
```

Features

Complete Authentication Flow

- Sign in / Sign up
- Password reset
- Email verification
- Profile management
- OAuth integration

Advanced Session Management

- Automatic token refresh
- Refresh on focus
- Refresh on reconnect
- Session persistence
- Multi-device logout

Security Best Practices

- Secure token storage
- CSRF protection
- XSS prevention
- Rate limiting ready
- Password validation

Developer Experience

- TypeScript support
- RedwoodJS-style generators
- Pre-built UI components
- Comprehensive hooks

- Excellent documentation

Examples

Login Form

```
import { useAuth } from 'philjs-auth/hooks';
import { signal } from 'philjs-core/signals';

const email = signal('');
const password = signal('');

export function LoginForm() {
  const { signIn, isLoading } = useAuth();

  const handleSubmit = async (e: Event) => {
    e.preventDefault();
    try {
      await signIn(email.get(), password.get());
      window.location.href = '/dashboard';
    } catch (error) {
      alert('Login failed');
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="email"
        value={email.get()}
        onChange={(e) => email.set(e.target.value)}
        placeholder="Email"
      />
      <input
        type="password"
        value={password.get()}
        onChange={(e) => password.set(e.target.value)}
        placeholder="Password"
      />
      <button type="submit" disabled={isLoading}>
        {isLoading ? 'Signing in...' : 'Sign In'}
      </button>
    </form>
  );
}
```

Contributing

Contributions are welcome! Please read our contributing guide.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./oauth`, `./jwt`, `./session`, `./session-refresh`, `./protected-route`, `./protected-routes`, `./hooks`, `./auth-provider`, `./providers/clerk`, `./providers/auth0`, `./providers/supabase`, `./providers/next-auth`
- Source files: `packages/philjs-auth/src/index.ts`, `packages/philjs-auth/src/oauth.ts`, `packages/philjs-auth/src/jwt.ts`, `packages/philjs-auth/src/session.ts`, `packages/philjs-auth/src/session-refresh.ts`, `packages/philjs-auth/src/protected-route.ts`, `packages/philjs-auth/src/hooks.ts`, `packages/philjs-auth/src/auth-provider.ts`, `packages/philjs-auth/src/providers/clerk.ts`, `packages/philjs-auth/src/providers/auth0.ts`, `packages/philjs-auth/src/providers/supabase.ts`, `packages/philjs-auth/src/providers/next-auth.ts`

Public API

- Direct exports: `Auth0AuthProvider`, `Auth0Config`, `AuthProvider`, `AuthProviderConfig`, `AuthProviderContext`, `AuthProviderFactory`, `ClerkAuthProvider`, `ClerkConfig`, `JWTManager`, `NextAuthConfig`, `NextAuthProvider`, `OAuthManager`, `OAuthProviders`, `PKCEPair`, `ProtectedRoute`, `RefreshConfig`, `SessionManager`, `SessionPersistence`, `SessionRefreshManager`, `SupabaseAuthProvider`, `SupabaseConfig`, `createJWTManager`, `createOAuthManager`, `createSessionManager`, `createToken`, `decodeToken`, `generateCodeChallenge`, `generateCodeVerifier`, `generatePKCE`, `generateState`, `getAuthProvider`, `getDefaultSessionManager`, `getRedirectUrl`, `getSessionRefreshManager`, `logoutEverywhere`, `redirectAfterLogin`, `redirectToLogin`, `requireAuth`, `setAuthProvider`, `setDefaultSessionManager`, `startSessionRefresh`, `stopSessionRefresh`, `useAccessToken`, `useAuthLoading`, `useHasPermission`, `useIsAuthenticated`, `useRequireAuth`, `useSession`, `useUser`, `validateState`, `verifyToken`, `withAuth`
- Re-exported names: `Auth0AuthProvider`, `Auth0Config`, `AuthConfig`, `AuthProvider`, `AuthProviderConfig`, `AuthProviderContext`, `AuthProviderFactory`, `AuthSession`, `BaseAuthProvider`, `ClerkAuthProvider`, `ClerkConfig`, `JWTConfig`, `JWTManager`, `JWTPayload`, `NextAuthConfig`, `NextAuthProvider`, `OAuthConfig`, `OAuthManager`, `OAuthProvider`, `OAuthProviders`, `ProtectedRoute`, `ProtectedRouteConfig`, `RefreshConfig`, `SessionManager`, `SessionPersistence`, `SessionRefreshManager`, `SupabaseAuthProvider`, `SupabaseConfig`, `User`, `createJWTManager`, `createOAuthManager`, `createSessionManager`, `createToken`, `decodeToken`, `generateState`, `getAuthProvider`, `getDefaultSessionManager`, `getRedirectUrl`, `getSessionRefreshManager`, `logoutEverywhere`, `redirectAfterLogin`, `redirectToLogin`, `requireAuth`, `setAuthProvider`, `setDefaultSessionManager`, `startSessionRefresh`, `stopSessionRefresh`, `useAccessToken`, `useAuth`, `useAuthLoading`, `useHasPermission`, `useIsAuthenticated`, `useRequireAuth`, `useSession`, `useUser`, `validateState`, `verifyToken`, `withAuth`
- Re-exported modules: `./auth-provider.js`, `./hooks.js`, `./jwt.js`, `./oauth.js`, `./protected-route.js`, `./providers/auth0.js`, `./providers/clerk.js`, `./providers/next-auth.js`, `./providers/supabase.js`, `./session-refresh.js`, `./session.js`, `./types.js`

License

MIT PhilJS Team

@philjs/benchmark - Type: Node package - Purpose: Comprehensive performance benchmarking suite for PhilJS - Version: 0.1.0 - Location: packages/philjs-benchmark - Entry points: packages/philjs-benchmark/src/index.ts, ./framework, packages/philjs-benchmark/src/reactivity/index.ts, packages/philjs-benchmark/src/ssr/index.ts, ./bundle, ./rust - Keywords: benchmark, performance, philjs, framework-benchmark

PhilJS Benchmark Suite

Comprehensive performance benchmarking suite for PhilJS, compatible with [js-framework-benchmark](#).

Features

- **Framework Benchmarks:** DOM manipulation, rendering, and update performance
- **Reactivity Benchmarks:** Signal, effect, memo, and batch performance
- **SSR Benchmarks:** Server-side rendering, hydration, and streaming
- **Bundle Size Analysis:** Bundle sizes and tree-shaking effectiveness
- **Rust/WASM Benchmarks:** JavaScript vs WASM performance comparison
- **Framework Comparison:** Compare PhilJS against React, Vue, Svelte, SolidJS, and more
- **Reproducible Results:** Consistent methodology with warmup and multiple iterations
- **Rich Reporting:** JSON, Markdown, and HTML reports with charts

Quick Start

```
# Install dependencies
npm install

# Run all benchmarks
npm run bench:all

# Run specific benchmark suites
npm run bench:framework
npm run bench:reactivity
npm run bench:ssr

# Generate reports
npm run report      # HTML report
npm run report -- --markdown # Markdown report

# Publish results
npm run publish
```

How to Run Benchmarks

Basic Usage

```
# Run framework benchmarks
npm run bench:framework

# Run with custom options
npm run bench:framework -- --iterations=100 --warmup=10

# Run and save results
npm run bench:all -- --save
```

Advanced Usage

```
import { runFrameworkBenchmarks } from 'philjs-benchmark';

const results = await runFrameworkBenchmarks({
  iterations: 100,           // Number of iterations per benchmark
  warmupIterations: 10,     // Warmup runs (not counted)
  verbose: true,            // Print progress
  saveResults: true,        // Save to results/
  outputPath: './results' // Output directory
});

console.log(results);
```

Benchmark Methodology

Framework Benchmarks

Compatible with [js-framework-benchmark](#) specification:

1. **create-1000-rows:** Create 1,000 table rows
2. **replace-1000-rows:** Replace all 1,000 rows
3. **append-1000-rows:** Append 1,000 rows to existing table
4. **update-every-10th-row:** Update every 10th row
5. **select-row:** Highlight a selected row
6. **swap-rows:** Swap two rows
7. **remove-row:** Delete a single row
8. **clear-rows:** Clear all rows
9. **startup-time:** Framework initialization time
10. **memory-1000-rows:** Memory footprint after creating 1,000 rows

Reactivity Benchmarks

Measure fine-grained reactivity performance:

- **Signal Operations:** Create, read, write signals
- **Effect Execution:** Effect creation, triggering, cleanup
- **Memo Computation:** Caching, recomputation, dependency tracking

- **Batch Updates:** Batched vs unbatched update performance

SSR Benchmarks

Server-side rendering performance:

- **Render Time:** Time to render components to HTML
- **Hydration:** Client-side hydration performance
- **Streaming:** Streaming SSR throughput and TTFB

Bundle Size Analysis

Analyze production bundle sizes:

- Raw size (uncompressed)
- Gzip compressed size
- Brotli compressed size
- Per-package breakdown

Ensuring Reproducibility

Hardware Consistency

Run benchmarks on the same hardware for consistent results:

```
# Check your environment
node -v
npm run bench:framework -- --verbose
```

Warmup Period

All benchmarks include warmup iterations to ensure JIT optimization:

```
{
  warmupIterations: 10, // Default warmup
  iterations: 100       // Actual benchmark runs
}
```

Statistical Analysis

Each benchmark calculates:

- **Mean:** Average performance
- **Median:** Middle value (less affected by outliers)
- **Min/Max:** Best and worst case
- **Standard Deviation:** Variance in results
- **Operations per Second:** Throughput metric

Comparing Results

Use the baseline comparison feature:

```
# Save baseline
npm run bench:all -- --save
cp results/latest.json results/baseline.json

# Run new benchmarks and compare
npm run bench:all -- --save
npm run publish -- --format=markdown
```

Sample Results

Framework Benchmarks

Benchmark	Mean	Median	Min	Max	Ops/sec
create-1000-rows	42.35ms	41.80ms	38.20ms	52.10ms	23.6
update-every-10th-row	8.52ms	8.40ms	7.80ms	10.20ms	117.4
swap-rows	3.25ms	3.20ms	2.90ms	4.10ms	307.7
select-row	2.18ms	2.10ms	1.95ms	3.20ms	458.7
remove-row	6.84ms	6.75ms	6.20ms	8.50ms	146.2
clear-rows	4.52ms	4.45ms	4.10ms	5.80ms	221.2
startup-time	68.50ms	67.20ms	62.30ms	82.40ms	14.6
memory-1000-rows	3.85MB	3.82MB	3.65MB	4.12MB	-

Reactivity Benchmarks

Benchmark	Mean	Ops/sec
create-10k-signals	2.35ms	4,255
read-1m-signals	12.50ms	80,000
write-100k-signals	18.75ms	5,333
create-1k-effects	3.42ms	292
effect-single-dependency-10k-updates	24.80ms	40.3
memo-caching-1m-reads	8.25ms	121,212
batch-1000-updates	5.20ms	192.3

Comparison with Other Frameworks

PhilJS compared to popular frameworks (lower is better):

Framework	create-1000-rows	update-every-10th	swap-rows	select-row
PhilJS	42.35ms	8.52ms	3.25ms	2.18ms
Vanilla JS	32.50ms	8.50ms	3.80ms	2.10ms
SolidJS	38.50ms	7.50ms	11.50ms	4.80ms
Svelte	42.80ms	12.30ms	15.20ms	8.50ms
Qwik	52.30ms	16.80ms	20.20ms	10.50ms
Vue	55.20ms	18.50ms	22.80ms	12.30ms
Preact	58.20ms	22.30ms	27.50ms	13.80ms
React	65.30ms	25.80ms	30.50ms	15.20ms

Key Insights:

- PhilJS is **35% faster than React** on average
- Competitive with SolidJS in most benchmarks
- Excellent reactivity performance with fine-grained updates
- Low memory footprint (3.85MB vs React's 8.2MB)

Bundle Size Comparison

Framework	Minified	Gzipped
PhilJS	15.2 KB	5.8 KB
Preact	12.5 KB	4.8 KB
Svelte	18.5 KB	7.2 KB
SolidJS	22.8 KB	8.5 KB
Vue	125.3 KB	48.5 KB
React	139.5 KB	44.2 KB

Output Formats

JSON Output

```
{
  "framework": "philjs",
  "version": "1.0.0",
  "timestamp": "2025-01-15T10:30:00Z",
  "environment": {
    "runtime": "node",
    "runtimeVersion": "v20.11.0",
    "os": "Linux 5.15.0",
    "cpu": "Intel Core i7-9750H (12 cores)",
    "memory": "32GB"
  },
  "suites": {
    "framework": { ... },
    "reactivity": { ... },
    "ssr": { ... }
  }
}
```

Markdown Output

```
npm run publish -- --format=markdown
```

Generates a formatted markdown report with tables and comparisons.

HTML Output

```
npm run publish -- --format=html
```

Generates an interactive HTML report with charts using Chart.js.

GitHub Pages

```
npm run publish -- --github-pages
```

Creates a complete GitHub Pages site with: - Interactive results dashboard - Framework comparison charts - Performance badges - Historical trends

CI/CD Integration

GitHub Actions

```
name: Benchmarks

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  benchmark:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
      with:
        node-version: 20
      - run: npm install
      - run: npm run bench:all -- --save
      - run: npm run publish -- --github-pages
      - name: Deploy to GitHub Pages
        uses: peaceiris/actions-gh-pages@v3
        with:
          github_token: ${{ secrets.GITHUB_TOKEN }}
          publish_dir: ./packages/philjs-benchmark/results/publish/gh-pages
```

Regression Detection

```
# Compare against baseline
npm run bench:all -- --save
node -e "
const baseline = require('./results/baseline.json');
const current = require('./results/latest.json');

const threshold = 0.1; // 10% regression threshold

for (const bench of current.suites.framework.results) {
  const base = baseline.suites.framework.results.find(b => b.name === bench.name);
  if (base) {
    const regression = (bench.mean - base.mean) / base.mean;
    if (regression > threshold) {
      console.error(`Regression detected in ${bench.name}: ${((regression * 100).toFixed(1))}% slower`);
      process.exit(1);
    }
  }
}
"
```

API Reference

Running Benchmarks

```
import {
  runFrameworkBenchmarks,
  runReactivityBenchmarks,
  runSSRBenchmarks,
  runBundleBenchmarks,
  runWasmBenchmarks
} from 'philjs-benchmark';

// Framework benchmarks
const frameworkResults = await runFrameworkBenchmarks({
  iterations: 100,
  warmupIterations: 10,
  verbose: true
});

// Reactivity benchmarks
const reactivityResults = await runReactivityBenchmarks({
  iterations: 50,
  verbose: false
});
```

Custom Benchmarks

```

import { runBenchmark, Benchmark } from 'philjs-benchmark';

const customBenchmark: Benchmark = {
  name: 'my-custom-benchmark',
  fn: async () => {
    // Your benchmark code
  },
  setup: async () => {
    // Optional setup
  },
  teardown: async () => {
    // Optional cleanup
  },
  iterations: 100
};

const result = await runBenchmark(customBenchmark);

```

Comparison Charts

```

import { generateAndSaveCharts, loadComparisonData } from 'philjs-benchmark/comparison';

const frameworkData = await loadComparisonData();
await generateAndSaveCharts(report, './output', './frameworks.json');

```

Configuration

Create `benchmark.config.js`:

```

export default {
  iterations: 100,
  warmupIterations: 10,
  timeout: 60000,
  verbose: true,
  saveResults: true,
  outputPath: './results',

  suites: {
    framework: true,
    reactivity: true,
    ssr: true,
    bundle: true,
    rust: false
  },

  comparison: {
    enabled: true,
    frameworks: ['react', 'vue', 'solid', 'svelte']
  }
};

```

Troubleshooting

High Variance in Results

If you see high standard deviation:

1. Close other applications
2. Increase warmup iterations
3. Run on consistent hardware
4. Use `--iterations=200` for more samples

Memory Benchmarks Failing

Requires Node.js with `--expose-gc` flag:

```
node --expose-gc node_modules/.bin/tsx scripts/run-all.ts
```

Comparison Data Missing

Download latest framework data:

```
curl -o src/comparison/frameworks.json \
  https://raw.githubusercontent.com/krausest/js-framework-benchmark/master/webdriver-ts/results.json
```

Contributing

To add new benchmarks:

1. Create benchmark file in appropriate directory
2. Export benchmark array with name and fn
3. Add to index exports
4. Add tests in `__tests__`
5. Update this README

Example:

```
// src/framework-benchmark/my-new-benchmark.ts
import type { Benchmark } from '../types.js';

export const myNewBenchmark: Benchmark = {
  name: 'my-new-benchmark',
  fn: async () => {
    // Benchmark implementation
  }
};

export const myNewBenchmarks = [myNewBenchmark];
```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./framework`, `./reactivity`, `./ssr`, `./bundle`, `./rust`
- Source files: `packages/philjs-benchmark/src/index.ts`, `packages/philjs-benchmark/src/reactivity/index.ts`, `packages/philjs-benchmark/src/ssr/index.ts`

Public API

- Direct exports: `allReactivityBenchmarks`, `allSSRBenchmarks`, `coreReactivityBenchmarks`, `coreSSRBenchmarks`, `runAllBenchmarks`, `runCoreReactivityBenchmarks`, `runCoreSSRBenchmarks`, `runReactivityBenchmarks`, `runSSRBenchmarks`
- Re-exported names: `allFrameworkBenchmarks`, `allReactivityBenchmarks`, `allSSRBenchmarks`, `batchBenchmarks`, `clearRows`, `coreFrameworkBenchmarks`, `coreReactivityBenchmarks`, `coreSSRBenchmarks`, `create1000Rows`, `create100Rows`, `createRowsBenchmarks`, `deleteRowBenchmarks`, `effectBenchmarks`, `hydrationBenchmarks`, `memoBenchmarks`, `progressiveHydration`, `removeRow`, `renderTimeBenchmarks`, `runFrameworkBenchmarks`, `runReactivityBenchmarks`, `runSSRBenchmarkSuite`, `selectRow`, `selectRowBenchmarks`, `signalBenchmarks`, `streamingBenchmarks`, `streamingThroughput`, `swapRows`, `swapRowsBenchmarks`, `updateEvery10thRow`, `updateRowsBenchmarks`
- Re-exported modules: `./batch.js`, `./effects.js`, `./framework-benchmark/create-rows.js`, `./framework-benchmark/delete-row.js`, `./framework-benchmark/runner.js`, `./framework-benchmark/select-row.js`, `./framework-benchmark/swap-rows.js`, `./framework-benchmark/update-rows.js`, `./hydration.js`, `./memos.js`, `./reactivity/batch.js`, `./reactivity/effects.js`, `./reactivity/index.js`, `./reactivity/memos.js`, `./reactivity/signals.js`, `./render-time.js`, `./signals.js`, `./ssr/hydration.js`, `./ssr/index.js`, `./ssr/render-time.js`, `./ssr/streaming.js`, `./streaming.js`, `./types.js`, `./utils.js`

License

MIT

Related Projects

- [js-framework-benchmark](#) - Framework benchmark suite
- [benchmark.js](#) - JavaScript benchmarking library
- [vitest](#) - Testing framework with benchmark support

Credits

Inspired by and compatible with [js-framework-benchmark](#) by Stefan Krause.

@philjs/biometric - Type: Node package - Purpose: Native biometric authentication for PhilJS - WebAuthn, Face ID, Touch ID, Windows Hello - Version: 0.1.0 - Location: `packages/philjs-biometric` - Entry points: `packages/philjs-biometric/src/index.ts` - Keywords: philjs, biometric, webauthn, fido2, passkey, face-id, touch-id

@philjs/biometric

Native Biometric Authentication for PhilJS Applications

Overview

WebAuthn, Face ID, Touch ID, and Windows Hello integration for PhilJS applications.

Features: - WebAuthn/FIDO2 registration and authentication - Passkey support with local storage persistence - Platform authenticator detection - Conditional UI (autofill) support - Cross-device authentication - Biometric type detection (face, fingerprint, iris) - Credential serialization for server sync

Installation

```
npm install @philjs/biometric
```

Quick Start

```
import { BiometricManager } from '@philjs/biometric';

const manager = new BiometricManager({
  rpName: 'My App',
  rpId: 'myapp.com'
});

// Check if biometrics are supported
if (BiometricManager.isSupported()) {
  // Register a new credential
  const result = await manager.register({
    userId: 'user-123',
    userName: 'john@example.com'
  });

  // Authenticate
  const auth = await manager.authenticate();
}
```

Usage

Basic Registration and Authentication

```

import { BiometricManager } from '@philjs/biometric';

const manager = new BiometricManager({
  rpName: 'My Application',
  rpId: window.location.hostname,
  userVerification: 'preferred',
  attestation: 'none',
  authenticatorAttachment: 'platform'
});

// Register
const registration = await manager.register({
  userId: 'user-123',
  userName: 'user@example.com',
  userDisplayName: 'John Doe'
});

// Authenticate
const authentication = await manager.authenticate();

```

Paskey Management

```

import { PasskeyManager } from '@philjs/biometric';

const passkeys = new PasskeyManager({
  rpName: 'My App',
  rpId: 'myapp.com'
});

// Create a passkey
await passkeys.createPasskey('user-id', 'user@example.com', 'John Doe');

// Sign in with passkey
const result = await passkeys.signInWithPasskey();

// Sign in with autofill (conditional UI)
const abortController = new AbortController();
const result = await passkeys.signInWithAutofill(abortController);

// Manage passkeys
passkeys.renamePasskey('credential-id', 'Work Laptop');
passkeys.deletePasskey('credential-id');
passkeys.getPasskeys();

```

Check Capabilities

```

import { BiometricManager, BiometricPrompt } from '@philjs/biometric';

const capabilities = await BiometricManager.getCapabilities();
// {
//   supported: true,
//   platformAuthenticator: true,
//   conditionalUI: true,
//   userVerifyingPlatformAuthenticator: true
// }

const biometricType = await BiometricPrompt.getBiometricType();
// 'face' | 'fingerprint' | 'iris' | 'unknown'

```

Biometric Prompt

```

import { BiometricPrompt } from '@philjs/biometric';

const prompt = new BiometricPrompt({ rpName: 'My App' });

const isAvailable = await BiometricPrompt.isAvailable();
const biometricType = await BiometricPrompt.getBiometricType();

const success = await prompt.prompt({
  title: 'Verify your identity',
  subtitle: 'Use biometrics to continue'
});

```

Hooks API

```

import { useBiometric, usePasskeys, useBiometricPrompt } from '@philjs/biometric';

// Biometric hook
const { capabilities, register, authenticate, credentials, isSupported, loading, error } =
  useBiometric({ rpName: 'My App' });

// Passkeys hook
const { passkeys, createPasskey, signIn, signInWithAutofill, renamePasskey, deletePasskey } =
  usePasskeys({ rpName: 'My App' });

// Prompt hook
const { prompt, isAvailable, biometricType, loading } =
  useBiometricPrompt({ rpName: 'My App' });

```

API Reference

Classes

BiometricManager

Core WebAuthn credential management.

Static Methods: - `getCapabilities()` - Get device biometric capabilities - `isSupported()` - Check if WebAuthn is supported

Methods: - `register(options)` - Register new credential - `authenticate(options?)` - Authenticate with existing credential - `conditionalAuthenticate(abortController?)` - Conditional UI authentication - `addCredential(credential)` - Add credential to manager - `removeCredential(id)` - Remove credential - `getCredentials()` - Get all credentials - `serializeCredential(credential)` - Serialize for storage - `deserializeCredential(serialized)` - Deserialize from storage

PasskeyManager

High-level passkey management with automatic persistence.

Methods: - `createPasskey(userId, userName, displayName?)` - Create new passkey - `signInWithPasskey()` - Sign in using passkey - `signInWithAutofill(abortController?)` - Conditional UI sign-in - `getPasskeys()` - Get stored passkeys - `renamePasskey(id, name)` - Rename a passkey - `deletePasskey(id)` - Delete a passkey - `clearPasskeys()` - Clear all passkeys

BiometricPrompt

Simple biometric verification prompt.

Methods: - `prompt(options?)` - Show biometric prompt - `isAvailable()` - Check if biometrics available - `getBiometricType()` - Detect biometric type

Types

```

interface BiometricConfig {
  rpName: string;
  rpId: string;
  timeout?: number;
  userVerification?: 'required' | 'preferred' | 'discouraged';
  attestation?: 'none' | 'indirect' | 'direct' | 'enterprise';
  authenticatorAttachment?: 'platform' | 'cross-platform';
  residentKey?: 'required' | 'preferred' | 'discouraged';
}

interface BiometricCredential {
  id: string;
  rawId: ArrayBuffer;
  type: 'public-key';
  authenticatorAttachment: 'platform' | 'cross-platform' | null;
  createdAt: number;
  lastUsed: number;
  deviceInfo?: string;
  name?: string;
}

interface BiometricCapabilities {
  supported: boolean;
  platformAuthenticator: boolean;
  conditionalUI: boolean;
  userVerifyingPlatformAuthenticator: boolean;
}

```

Browser Support

- Chrome 67+
- Firefox 60+
- Safari 14+
- Edge 79+

Requires HTTPS in production (localhost exempt for development).

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-biometric/src/index.ts

Public API

- Direct exports: AuthenticationOptions, AuthenticationResult, BiometricCapabilities, BiometricConfig, BiometricCredential, BiometricManager, BiometricPrompt, BiometricPromptOptions, PasskeyManager, RegistrationOptions, RegistrationResult, useBiometric, useBiometricPrompt, usePasskeys
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/build - Type: Node package - Purpose: Rspack/Rslib build toolchain for PhilJS with Module Federation support - Version: 0.1.0 - Location: packages/philjs-build - Entry points: packages/philjs-build/src/index.ts, packages/philjs-build/src/rspack/index.ts, packages/philjs-build/src/rslib/index.ts, packages/philjs-build/src/vite/index.ts, packages/philjs-build/src/cli/index.ts - Keywords: philjs, rspack, rslib, build, bundler, module-federation

@philjs/build

High-performance build tooling for PhilJS applications, powered by Rspack and Rslib. Provides Rust-based bundling with up to 10x faster builds compared to webpack, while maintaining compatibility with existing Vite configurations.

Features

- **Rspack Plugin:** Zero-config plugin for PhilJS applications
- **Rslib Presets:** Pre-configured library building presets
- **Vite Compatibility:** Seamless migration from Vite to Rspack
- **Tree Shaking:** Automatic dead code elimination
- **Code Splitting:** Intelligent chunk splitting
- **TypeScript:** First-class TypeScript support
- **CSS Handling:** Built-in CSS modules and PostCSS support

Installation

```
npm install @philjs/build
# or
pnpm add @philjs/build
# or
bun add @philjs/build
```

Quick Start

Rspack Plugin

```
// rspack.config.ts
import { philJSRspackPlugin, createRspackConfig } from '@philjs/build';

export default createRspackConfig({
  mode: 'production',
  entry: './src/index.ts',
  plugins: [
    philJSRspackPlugin({
      // Enable JSX transform
      jsx: true,
      // Enable HMR in development
      hmr: true,
    }),
  ],
});
```

Basic Configuration

```
import { createRspackConfig } from '@philjs/build';

export default createRspackConfig({
  entry: './src/index.ts',
  output: {
    path: './dist',
    filename: '[name].[contenthash].js',
  },
  resolve: {
    alias: {
      '@': './src',
    },
  },
});
```

Rspack Configuration

Development Mode

```

import { createRspackConfig, philJSRspackPlugin } from '@philjs/build';

export default createRspackConfig({
  mode: 'development',
  devtool: 'eval-source-map',
  devServer: {
    port: 3000,
    hot: true,
    open: true,
    historyApiFallback: true,
  },
  plugins: [
    philJSRspackPlugin({
      mode: 'development',
    }),
  ],
});

```

Production Mode

```

import { createRspackConfig, philJSRspackPlugin } from '@philjs/build';

export default createRspackConfig({
  mode: 'production',
  optimization: {
    minimize: true,
    splitChunks: {
      chunks: 'all',
      cachedGroups: {
        vendor: {
          test: /[\\/]node_modules[\\/]/,
          name: 'vendors',
          chunks: 'all',
        },
      },
    },
  },
  plugins: [
    philJSRspackPlugin({
      mode: 'production',
      minify: true,
      sourcemap: 'hidden',
    }),
  ],
});

```

TypeScript Configuration

```

import { createRspackConfig } from '@philjs/build';

export default createRspackConfig({
  entry: './src/index.ts',
  module: {
    rules: [
      {
        test: /\.tsx?$/,
        use: {
          loader: 'builtin:swc-loader',
          options: {
            jsc: {
              parser: {
                syntax: 'typescript',
                tsx: true,
              },
              transform: {
                react: {
                  runtime: 'automatic',
                },
              },
            },
          },
        },
      ],
    },
    resolve: {
      extensions: ['.ts', '.tsx', '.js', '.jsx'],
    },
  });

```

Rspack Presets

Library Preset

Build reusable libraries with ESM and CJS output:

```

import { createRslibConfig } from '@philjs/build';

export default createRslibConfig({
  preset: 'library',
  source: {
    entry: './src/index.ts',
  },
  output: {
    formats: ['esm', 'cjs'],
    dts: true,
  },
  external: ['react', 'react-dom'],
});

```

Application Preset

Build optimized applications:

```

import { createRslibConfig } from '@philjs/build';

export default createRslibConfig({
  preset: 'application',
  source: {
    entry: {
      main: './src/main.ts',
      worker: './src/worker.ts',
    },
  },
  output: {
    minify: true,
    splitting: true,
  },
});

```

Component Preset

Build component libraries with CSS extraction:

```

import { createRslibConfig } from '@philjs/build';

export default createRslibConfig({
  preset: 'component',
  source: {
    entry: './src/index.ts',
  },
  output: {
    preserveModules: true,
    extractCSS: true,
    dts: true,
  },
});

```

Node.js Preset

Build Node.js applications and libraries:

```

import { createRslibConfig } from '@philjs/build';

export default createRslibConfig({
  preset: 'node',
  source: {
    entry: './src/server.ts',
  },
  output: {
    format: 'cjs',
    target: 'node',
  },
  autoExternalNode: true,
});

```

Vite Compatibility

Migration from Vite

```

import { viteAdapter, convertRspackToVite } from '@philjs/build';

// Option 1: Use the Vite adapter directly
export default viteAdapter({
  mode: 'production',
  build: {
    outDir: 'dist',
    minify: 'esbuild',
  },
});

// Option 2: Convert existing Rspack config to Vite
import rspackConfig from './rspack.config';

export default convertRspackToVite(rspackConfig);

```

Hybrid Configuration

Use the same config for both Rspack and Vite:

```
import { createRspackConfig, createViteCompatibleConfig } from '@philjs/build';

const sharedConfig = {
  entry: './src/index.ts',
  resolve: {
    alias: {
      '@': './src',
    },
  },
};

// For Rspack
export const rspackConfig = createRspackConfig(sharedConfig);

// For Vite
export const viteConfig = createViteCompatibleConfig(sharedConfig);
```

Plugin Options

RspackPluginOptions

```
interface RspackPluginOptions {
  // Build mode
  mode?: 'development' | 'production';

  // Enable minification
  minify?: boolean;

  // Sourcemap configuration
  sourcemap?: boolean | 'inline' | 'hidden' | 'nosources';

  // ES target version
  target?: 'es2020' | 'es2021' | 'es2022' | 'es2023' | 'es2024' | 'esnext';

  // Enable JSX transform
  jsx?: boolean;

  // JSX runtime
  jsxRuntime?: 'automatic' | 'classic';

  // Enable HMR
  hmr?: boolean;
}
```

RslibConfigOptions

```
interface RslibConfigOptions {
  // Preset to use
  preset?: 'library' | 'application' | 'component' | 'node';

  // Source configuration
  source?: {
    entry?: string | Record<string, string>;
    alias?: Record<string, string>;
  };

  // Output configuration
  output?: {
    dir?: string;
    format?: 'esm' | 'cjs' | 'umd' | 'iife';
    formats?: Array<'esm' | 'cjs' | 'umd' | 'iife'>;
    dts?: boolean;
    minify?: boolean;
    sourceMap?: boolean;
    clean?: boolean;
    splitting?: boolean;
    preserveModules?: boolean;
    extractCSS?: boolean;
  };

  // External dependencies
  external?: string[];

  // Auto-external configuration
  autoExternal?: {
    peerDependencies?: boolean;
    dependencies?: boolean;
  };
  autoExternalPeers?: boolean;
  autoExternalNode?: boolean;

  // Plugins
  plugins?: RspackPlugin[];
}
```

Advanced Usage

Custom Loaders

```
import { createRspackConfig } from '@philjs/build';

export default createRspackConfig({
  module: {
    rules: [
      {
        test: /\.svg$/,
        use: ['@svgr/webpack'],
      },
      {
        test: /\.(\.png|\.jpg|\.gif)$/,
        type: 'asset/resource',
      },
      {
        test: /\.yaml$/,
        use: 'yaml-loader',
      },
    ],
  },
});
```

Environment Variables

```
import { createRspackConfig } from '@philjs/build';
import { DefinePlugin } from '@rspack/core';

export default createRspackConfig({
  plugins: [
    new DefinePlugin({
      'process.env.API_URL': JSON.stringify(process.env.API_URL),
      '__DEV__': JSON.stringify(process.env.NODE_ENV === 'development'),
    }),
  ],
});
```

Code Splitting

```
import { createRspackConfig } from '@philjs/build';

export default createRspackConfig({
  optimization: {
    splitChunks: {
      chunks: 'all',
      maxInitialRequests: 25,
      minSize: 20000,
      cachedGroups: {
        default: false,
        vendors: false,
        framework: {
          name: 'framework',
          chunks: 'all',
          test: /[\\/]node_modules[\\/](react|react-dom|scheduler)[\\/]/,
          priority: 40,
        },
        lib: {
          test: /[\\/]node_modules[\\/]/,
          name: 'lib',
          priority: 30,
          minChunks: 1,
          reuseExistingChunk: true,
        },
        commons: {
          name: 'commons',
          minChunks: 2,
          priority: 20,
        },
        shared: {
          name: 'shared',
          priority: 10,
          minChunks: 2,
          reuseExistingChunk: true,
        },
      },
    },
  },
});
```

Performance

Rspack provides significant performance improvements over webpack:

Metric	Webpack	Rspack	Improvement
Cold Start	3.5s	0.4s	8.75x
HMR	1.2s	0.1s	12x
Production Build	45s	5s	9x

API Reference

Functions

- `philJSRspackPlugin(options?)` - Create Rspack plugin for PhilJS
- `createRspackConfig(options)` - Create Rspack configuration
- `mergeRspackConfig(base, override)` - Merge configurations
- `createRslibConfig(options)` - Create Rslib configuration
- `viteAdapter(options?)` - Create Vite-compatible configuration
- `convertRspackToVite(rspackConfig)` - Convert Rspack config to Vite

Presets

- `rslibPresets.library` - Library building preset
- `rslibPresets.application` - Application building preset
- `rslibPresets.component` - Component library preset
- `rslibPresets.node` - Node.js preset

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./rspack`, `./rslib`, `./vite`, `./cli`
- Source files: `packages/philjs-build/src/index.ts`, `packages/philjs-build/src/rspack/index.ts`, `packages/philjs-build/src/rslib/index.ts`, `packages/philjs-build/src/vite/index.ts`, `packages/philjs-build/src/cli/index.ts`

Public API

- Direct exports: `program`
- Re-exported names: `DevServerConfig`, `ModuleFederationOptions`, `OutputFormat`, `PhilJSRslibOptions`, `PhilJSRspackOptions`, `PhilJSViteOptions`, `RslibConfig`, `RslibLibConfig`, `RspackConfiguration`, `RspackRule`, `SharedModuleConfig`, `VitePlugin`, `adaptVitePlugin`, `createDevServerConfig`, `createModuleFederationPlugin`, `createPhilJSPlugin`, `createRslibConfig`, `createRspackConfig`, `defineConfig`, `entriesFromGlob`, `generateExportsField`, `mergeConfigs`, `philJSVite`, `rslibPresets`, `rspackPresets`, `rspackViteCompat`, `toRspackDevServerConfig`, `toViteServerConfig`
- Re-exported modules: `./compatibility.js`, `./config.js`, `./rslib/config.js`, `./rspack/config.js`, `./vite/compatibility.js`

License

MIT

@philjs/builder - Type: Node package - Purpose: Visual component builder with drag-and-drop UI for building PhilJS components - Version: 0.1.0 - Location: packages/philjs-builder - Entry points: `packages/philjs-builder/src/index.ts`, `packages/philjs-builder/src/builder/index.ts`, `packages/philjs-builder/src/canvas/index.ts`, `packages/philjs-builder/src/components/index.ts`, `packages/philjs-builder/src/serialization/index.ts`, `packages/philjs-builder/src/state/index.ts`, `packages/philjs-builder/src/preview/index.ts` - Keywords: philjs, builder, visual, drag-and-drop, component-builder, no-code, low-code, design-tool

@philjs/builder

A visual UI component builder and editor for React applications. Enables drag-and-drop component composition with real-time preview and code generation.

Installation

```
npm install @philjs/builder
# or
yarn add @philjs/builder
# or
pnpm add @philjs/builder
```

Basic Usage

```
import { Builder, ComponentPalette, Canvas } from '@philjs/builder';

function App() {
  const handleSave = (components) => {
    console.log('Saved components:', components);
  };

  return (
    <Builder onSave={handleSave}>
      <ComponentPalette />
      <Canvas />
    </Builder>
  );
}

export default App;
```

Features

- **Visual Editor** - Drag-and-drop interface for building UI components
- **Component Palette** - Pre-built component library with customizable elements
- **Real-time Preview** - Instant visual feedback as you build

- **Code Generation** - Export to React/TypeScript code
- **Serialization** - Save and load component configurations
- **Template System** - Built-in templates for common UI patterns
- **Undo/Redo** - Full history management for editing operations
- **Responsive Design** - Preview components at different breakpoints
- **Custom Components** - Register your own components in the palette
- **Props Editor** - Visual interface for editing component properties

API Reference

Builder

Main container component that provides context for the editor.

Canvas

The editable area where components are placed and arranged.

ComponentPalette

Sidebar displaying available components for drag-and-drop.

useBuilder

Hook for accessing builder state and actions programmatically.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./builder`, `./canvas`, `./components`, `./serialization`, `./state`, `./preview`
- Source files: `packages/philijs-builder/src/index.ts`, `packages/philijs-builder/src/builder/index.ts`, `packages/philijs-builder/src/canvas/index.ts`, `packages/philijs-builder/src/components/index.ts`, `packages/philijs-builder/src/serialization/index.ts`, `packages/philijs-builder/src/state/index.ts`, `packages/philijs-builder/src/preview/index.ts`

Public API

- Direct exports: (none detected)
- Re-exported names: `// Builder state types DocumentMetadata`, `// Canvas types ViewportState`, `// Code generator generateCode`, `// Component definition types PropDefinition`, `// Component library allComponents`, `// Component tree ComponentTree`, `// Drag and drop DragDropContext`, `// Drag and drop types DragOperation`, `// Event types BuilderEvents`, `// Grid system GridSystem`, `// History createHistoryManager`, `// History types HistoryActionType`, `// Inspector Inspector`, `// Main canvas Canvas`, `// Node types NodeId`, `// Palette Palette`, `// Preview types PreviewMessage`, `// Property panel PropertyPanel`, `// Resize ResizeHandles`, `// Resize types ResizeHandle`, `// Selection SelectionBox`, `// Selection types BoundingBox`, `// Serialization types SerializedDocument`, `// Store createBuilderStore`, `// Template system createTemplateManager`, `// Viewport mode ViewportMode`, `// AlignmentLine AlignmentLine`, `// BindingExpression BindingExpression`, `// BuilderAction BuilderAction`, `// BuilderEventListener BuilderEventListener`, `// BuilderState BuilderState`, `// BuilderUIState BuilderUIState`, `// Canvas Canvas`, `// CanvasGrid CanvasGrid`, `// CanvasGridProps CanvasGridProps`, `// CanvasNode CanvasNode`, `// CanvasNodeProps CanvasNodeProps`, `// CanvasRulers CanvasRulers`, `// CanvasRulersProps CanvasRulersProps`, `// CanvasSettings CanvasSettings`, `// CanvasState CanvasState`, `// CodeGeneratorOptions CodeGeneratorOptions`, `// CodegenOptions CodegenOptions`, `// ComponentCategory ComponentCategory`, `// ComponentDefinition ComponentDefinition`, `// ComponentNode ComponentNode`, `// ComponentTreeProps ComponentTreeProps`, `// ComponentType ComponentType`, `// DeviceFrame DeviceFrame`, `// DeviceFrameProps DeviceFrameProps`, `// DevicePreset DevicePreset`, `// DeviceSelector DeviceSelector`, `// DeviceSelectorProps DeviceSelectorProps`, `// DistanceIndicator DistanceIndicator`, `// DistanceIndicatorProps DistanceIndicatorProps`, `// DragDropContext DragDropContext`, `// DragDropContextProps DragDropContextProps`, `// DragPreview DragPreview`, `// DragPreviewProps DragPreviewProps`, `// DragSource DragSource`, `// DragState DragState`, `// Draggable Draggable`, `// DraggableProps DraggableProps`, `// DropIndicator DropIndicator`, `// DropIndicatorProps DropIndicatorProps`, `// DropTarget DropTarget`, `// Droppable Droppable`, `// DroppableProps DroppableProps`, `// EventEditor EventEditor`, `// EventHandler EventHandler`, `// GeneratedCode GeneratedCode`, `// GridPattern GridPattern`, `// GridPatternProps GridPatternProps`, `// GridSettings GridSettings`, `// GridSystem GridSystem`, `// GridSystemController GridSystemController`, `// GridSystemOptions GridSystemOptions`, `// GridSystemProps GridSystemProps`, `// HistoryEntry HistoryEntry`, `// HistoryManager HistoryManager`, `// HistoryManagerOptions HistoryManagerOptions`, `// HistoryState HistoryState`, `// HoverHighlight HoverHighlight`, `// HoverHighlightProps HoverHighlightProps`, `// ImportOptions ImportOptions`, `// Inspector Inspector`, `// InspectorProps InspectorProps`, `// KeyboardResizeOptions KeyboardResizeOptions`, `// LayoutConstraints LayoutConstraints`, `// MarqueeSelection MarqueeSelection`, `// MarqueeSelectionProps MarqueeSelectionProps`, `// NodeMetadata NodeMetadata`, `// NodeStyles NodeStyles`, `// Palette Palette`, `// PaletteCategory PaletteCategory`, `// PaletteCategoryProps PaletteCategoryProps`, `// PaletteItem PaletteItem`, `// PaletteItemProps PaletteItemProps`, `// PreviewCallback PreviewCallback`, `// PreviewFrameProps PreviewFrameProps`, `// PreviewToolbar PreviewToolbar`, `// PreviewToolbarProps PreviewToolbarProps`, `// PropertyValue PropertyValue`, `// PropertyEditor PropertyEditor`, `// PropertyEditorProps PropertyEditorProps`, `// PropertyGroup PropertyGroup`, `// PropertyGroupProps PropertyGroupProps`, `// PropertyPanel PropertyPanel`, `// PropertyPanelProps PropertyPanelProps`, `// ResizeHandles ResizeHandles`, `// ResizeHandlesProps ResizeHandlesProps`, `// ResizeManager ResizeManager`, `// ResizeManagerProps ResizeManagerProps`, `// ResizePreview ResizePreview`, `// ResizePreviewProps ResizePreviewProps`, `// ResizeState ResizeState`, `// ResponsiveController ResponsiveController`, `// ResponsivePreview ResponsivePreview`, `// SelectionBox SelectionBox`, `// SelectionBoxProps SelectionBoxProps`, `// SelectionManager SelectionManager`, `// SelectionManagerProps SelectionManagerProps`, `// SelectionOverlay SelectionOverlay`, `// SelectionOverlayProps SelectionOverlayProps`, `// SelectionState SelectionState`, `// SmartGuides SmartGuides`, `// SmartGuidesProps SmartGuidesProps`, `// SnapGuide SnapGuide`, `// SnapResult SnapResult`, `// StyleEditor StyleEditor`, `// StyleEditorProps StyleEditorProps`, `// StyleValue StyleValue`, `// Template Template`, `// TemplateCategory TemplateCategory`, `// TemplateManager TemplateManager`, `// TemplateManagerOptions TemplateManagerOptions`, `// TreeNodeProps TreeNodeProps`, `// TreeState TreeState`, `// VisualBuilder VisualBuilder`, `// VisualBuilderProps VisualBuilderProps`, `// additionalBuiltInTemplates additionalBuiltInTemplates`, `// allComponents allComponents`, `// applyTemplate applyTemplate`, `// blogArticle blogArticle`, `// builtinCategories builtinCategories`, `// builtinComponents builtinComponents`, `// builtinTemplates builtinTemplates`, `// calculateSnap calculateSnap`, `// componentCategories componentCategories`, `// createBuilderStore createBuilderStore`, `// createGridSystemController createGridSystemController`, `// createHistoryKeyboardHandler createHistoryKeyboardHandler`, `// createHistoryManager createHistoryManager`, `// createResponsiveController createResponsiveController`, `// createTemplateManager createTemplateManager`, `// dashboardSidebar dashboardSidebar`, `// defaultCategories defaultCategories`, `// defaultTemplateCategories defaultTemplateCategories`, `// devicePresets devicePresets`, `// exportAsJSON exportAsJSON`, `// faqSection faqSection`, `// generateCSSClass generateCSSClass`, `// generateCode generateCode`, `// generateId generateId`, `// generateInlineCSS generateInlineCSS`, `// generateSXString generateSXString`, `// getBuilderStore getBuilderStore`, `// getComponentDefinition getComponentDefinition`, `// getComponentsByCategory getComponentsByCategory`, `// getDeviceById getDeviceById`, `// getDevicesByCategory getDevicesByCategory`, `// getPhilijsUIComponent getPhilijsUIComponent`, `// getPhilijsComponentsByCategory getPhilijsComponentsByCategory`, `// landingPageSimple landingPageSimple`, `// loginForm loginForm`, `// navigationHeader navigationHeader`, `// philijsUICategories philijsUICategories`, `// philijsUIComponents philijsUIComponents`, `// pricingTable pricingTable`, `// productGrid productGrid`, `// registerBuiltInComponents registerBuiltInComponents`, `// registerPhilijsComponents registerPhilijsComponents`, `// resetBuilderStore resetBuilderStore`, `// shouldSnap shouldSnap`, `// snapToGrid snapToGrid`, `// testimonials testimonials`, `// useKeyboardResize useKeyboardResize`, `// withHistory withHistory`
- Re-exported modules: `./BuiltInTemplates.js`, `./Canvas.js`, `./CodeGenerator.js`, `./ComponentLibrary.js`, `./ComponentTree.js`, `./DragDrop.js`, `./GridSystem.js`, `./Inspector.js`, `./Palette.js`, `./PhilijsUIComponents.js`, `./PropertyPanel.js`, `./Resize.js`, `./ResponsivePreview.js`, `./Selection.js`, `./TemplateSystem.js`, `./VisualBuilder.js`, `./builder/index.js`, `./canvas/index.js`, `./components/PhilijsUIComponents.js`, `./components/index.js`, `./history.js`, `./preview/index.js`, `./serialization/index.js`, `./state/history.js`, `./state/store.js`, `./store.js`, `./types.js`

License

MIT

`## @philijs/carbon - Type: Node package - Purpose: Carbon-aware computing for PhilJS - schedule tasks during low-carbon periods - Version: 0.1.0 - Location: packages/philijs-carbon - Entry points: packages/philijs-carbon/src/index.ts - Keywords: philijs, carbon, sustainability, green-computing, carbon-aware, energy-efficient, scheduler, eco-friendly`

@philijs/carbon

Carbon-Aware Computing for PhilJS Applications

Overview

Schedule compute-intensive tasks during low-carbon periods. Built for sustainable web applications.

Features: - Real-time carbon intensity monitoring - Intelligent task scheduling during green periods - Battery-aware execution strategies - Network carbon estimation - Carbon budget management - Carbon footprint reporting - Regional grid intensity data

Installation

```
npm install @philijs/carbon
```

Quick Start

```

import { CarbonTaskScheduler, greenCompute } from '@philjs/carbon';

// Initialize scheduler
const scheduler = new CarbonTaskScheduler({
  region: 'usa-ca',
  greenThreshold: 100
});
await scheduler.initialize();

// Schedule a task for green execution
await scheduler.scheduleTask('heavy-compute', async () => {
  // Your compute-intensive work
}, {
  priority: 'normal',
  estimatedEnergy: 50,
  preferGreen: true
});

```

Usage

Carbon Task Scheduler

```

import { CarbonTaskScheduler } from '@philjs/carbon';

const scheduler = new CarbonTaskScheduler({
  region: 'germany',
  enableScheduling: true,
  maxQueueSize: 100,
  defaultPriority: 'normal',
  greenThreshold: 100, // gCO2eq/kWh
  carbonBudget: {
    daily: 1000,
    weekly: 5000,
    monthly: 20000
  }
});

await scheduler.initialize();

// Schedule tasks with different priorities
await scheduler.scheduleTask('critical-task', async () => {
  // Executes immediately regardless of carbon intensity
}, { priority: 'critical' });

await scheduler.scheduleTask('deferrable-task', async () => {
  // Waits for low-carbon window
}, {
  priority: 'deferrable',
  maxDelay: 24 * 60 * 60 * 1000, // 24 hours
  preferGreen: true
});

```

Carbon Intensity Monitoring

```

import { CarbonIntensityProvider } from '@philjs/carbon';

const provider = new CarbonIntensityProvider('uk');

// Get current intensity
const intensity = await provider.getCurrentIntensity();
console.log(`Current: ${intensity.value} gCO2eq/kWh (${intensity.index})`);

// Get 24-hour forecast
const forecast = await provider.getForecast(24);

// Find optimal execution window
const window = await provider.findOptimalWindow(
  5000, // 5 second task duration
  3600000 // 1 hour max delay
);

// Subscribe to intensity updates
const unsubscribe = provider.subscribe((intensity) => {
  console.log(`Intensity updated: ${intensity.value}`);
});

```

Device Energy Monitoring

```

import { DeviceEnergyMonitor } from '@philjs/carbon';

const monitor = new DeviceEnergyMonitor();
await monitor.initialize();

const energy = await monitor.getCurrentEnergy();
console.log('Battery:', energy.batteryLevel * 100 + '%');
console.log('Charging:', energy.charging);
console.log('Power source:', energy.powerSource);

// Check if on "green" power
if (monitor.isOnGreenPower()) {
  // Good time for compute
}

```

Network Carbon Estimation

```

import { NetworkCarbonEstimator } from '@philjs/carbon';

// Estimate transfer carbon
const transferCarbon = NetworkCarbonEstimator.estimateTransfer(
  1024 * 1024 * 100, // 100MB
  'germany'
);

// Estimate request carbon
const requestCarbon = NetworkCarbonEstimator.estimateRequest(
  1000, // 1KB request
  50000, // 50KB response
  'usa-ca'
);

// Create carbon-aware fetch wrapper
const greenFetch = NetworkCarbonEstimator.createFetchWrapper('france');
const response = await greenFetch('/api/data');
console.log('Carbon estimate:', (response as any).__carbonEstimate);

```

Green Compute Wrapper

```

import { greenCompute } from '@philjs/carbon';

// Wrap any sync function for carbon-aware execution
const processData = greenCompute(async (data: number[]) => {
  return data.map(x => x * 2);
}, {
  estimatedEnergy: 10,
  maxDelay: 3600000,
  priority: 'normal'
});

const result = await processData([1, 2, 3, 4, 5]);

```

Hooks API

```

import {
  useCarbonScheduler,
  useCarbonIntensity,
  useDeviceEnergy,
  useNetworkCarbon,
  useCarbonBudget
} from '@philjs/carbon';

// Scheduler hook
const { scheduler, scheduleTask, cancelTask, getBudget, generateReport } =
  useCarbonScheduler();

// Intensity hook
const { intensity, loading, forecast, isGreen, refresh } =
  useCarbonIntensity('uk');

// Energy hook
const { energy, isOnGreenPower, batteryLevel, isCharging } =
  useDeviceEnergy();

// Network carbon hook
const { estimateTransfer, estimateRequest, greenFetch } =
  useNetworkCarbon('france');

// Budget hook
const { budget, isWithinBudget, dailyRemaining, percentUsed } =
  useCarbonBudget();

```

API Reference

Classes

[CarbonTaskScheduler](#)

Main scheduler for carbon-aware task execution.

Methods: - `initialize()` - Initialize the scheduler - `scheduleTask(id, task, options?)` - Schedule a task - `executeNow(id)` - Force immediate execution - `cancelTask(id)` - Cancel a scheduled task - `getQueuedTasks()` - Get pending tasks - `getBudget()` - Get carbon budget status - `generateReport(start?, end?)` - Generate carbon report - `destroy()` - Cleanup scheduler

`CarbonIntensityProvider`

Real-time carbon intensity data.

Methods: - `getCurrentIntensity()` - Get current intensity - `getForecast(hours)` - Get forecast - `findOptimalWindow(duration, maxDelay)` - Find best execution window - `subscribe(callback)` - Subscribe to updates - `setRegion(region)` - Change region

`DeviceEnergyMonitor`

Device battery and power monitoring.

Methods: - `initialize()` - Initialize monitor - `getCurrentEnergy()` - Get current energy state - `isOnGreenPower()` - Check if on green power - `subscribe(callback)` - Subscribe to changes

`NetworkCarbonEstimator`

Network transfer carbon estimation.

Static Methods: - `estimateTransfer(bytes, region?)` - Estimate transfer carbon - `estimateRequest(reqBytes, resBytes, region?)` - Estimate request carbon - `getRegionIntensity(region?)` - Get regional grid intensity - `createFetchWrapper(region?)` - Create carbon-aware fetch

Types

```
interface CarbonConfig {
  region?: string;
  apiKey?: string;
  enableScheduling?: boolean;
  maxQueueSize?: number;
  defaultPriority?: 'critical' | 'high' | 'normal' | 'low' | 'deferrable';
  carbonBudget?: Partial<CarbonBudget>;
  greenThreshold?: number;
}

interface CarbonIntensity {
  value: number;
  index: 'very-low' | 'low' | 'moderate' | 'high' | 'very-high';
  timestamp: number;
  region: string;
  forecast: CarbonForecast[];
}

interface TaskSchedule {
  id: string;
  task: () => Promise<any>;
  priority: 'critical' | 'high' | 'normal' | 'low' | 'deferrable';
  estimatedEnergy: number;
  maxDelay: number;
  preferGreen: boolean;
}

interface CarbonBudget {
  daily: number;
  weekly: number;
  monthly: number;
  used: { daily: number; weekly: number; monthly: number };
}
```

Regional Grid Intensities

Pre-configured intensities (gCO2/kWh): - France: 50 - Sweden: 45 - Norway: 30 - Iceland: 20 - UK: 200 - Germany: 350 - USA: 400 - USA-CA: 200 - China: 550 - India: 700

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: ..
- Source files: packages/philijs-carbon/src/index.ts

Public API

- Direct exports: CarbonBudget, CarbonConfig, CarbonForecast, CarbonIntensity, CarbonIntensityProvider, CarbonReport, CarbonTaskScheduler, DeviceEnergy, DeviceEnergyMonitor, NetworkCarbonEstimator, TaskSchedule, greenCompute, useCarbonBudget, useCarbonIntensity, useCarbonScheduler, useDeviceEnergy, useNetworkCarbon
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philijs/cdn - Type: Node package - Purpose: PhilJS CDN bundle - single file for Alpine.js-style usage, no build step required - Version: 0.1.0 - Location: packages/philijs-cdn - Entry points: packages/philijs-cdn/src/index.ts, ./global, ./esm, ./mini - Keywords: philijs, cdn, umd, global, no-build, alpine-style

@philijs/cdn

CDN Bundle for PhilJS - Zero Build Step Required

Overview

Single-file bundle for Alpine.js-style usage. No build step required - just include via CDN.

Features: - Fine-grained reactivity with signals - Tagged template literals (`html`...``) - Alpine.js-style directives (`x-data`, `x-text`, `x-bind`, etc.) - Zero dependencies - Automatic DOM binding - Store creation with subscriptions - Works in any browser with ES2020+ support

Installation

Via CDN

```
<script src="https://unpkg.com/@philjs/cdn"></script>
```

Via npm

```
npm install @philjs/cdn
```

Quick Start

Programmatic API

```
<script src="https://unpkg.com/@philjs/cdn"></script>
<script>
  const { signal, effect, html, render } = PhilJS;

  const count = signal(0);

  effect(() => console.log('Count:', count()));

  render(
    html`<button onclick={() => count.set(count() + 1)}>
      Clicked ${count} times
    </button>`,
    document.body
  );
</script>
```

Alpine.js-Style Directives

```
<div x-data="{ count: 0, name: 'World' }>
  <h1>Hello, <span x-text="name"></span>!</h1>
  <p>Count: <span x-text="count"></span></p>
  <button @click="count++>Increment</button>
  <input x-model="name" placeholder="Enter name">
</div>

<script src="https://unpkg.com/@philjs/cdn"></script>
<script>PhilJS.init();</script>
```

Usage

Signals (Reactive State)

```
const { signal, memo, effect, batch } = PhilJS;

// Create a signal
const count = signal(0);

// Read value
console.log(count()); // 0

// Set value
count.set(5);
count.set(prev => prev + 1);

// Update (shorthand for set with function)
count.update(n => n * 2);

// Peek without tracking
const value = count.peek();
```

Computed Values (Memos)

```
const firstName = signal('John');
const lastName = signal('Doe');

const fullName = memo(() => `${firstName()} ${lastName()}`);

console.log(fullName()); // "John Doe"
```

Effects

```

const count = signal(0);

const cleanup = effect(() => {
  console.log('Count changed:', count());

  // Optional cleanup function
  return () => {
    console.log('Cleaning up...');
  };
});

// Stop the effect
cleanup();

```

Batching Updates

```

const a = signal(1);
const b = signal(2);

batch(() => {
  a.set(10);
  b.set(20);
  // Effects run once after batch completes
});

```

Template Rendering

```

const { html, render, signal } = PhilJS;

const items = signal(['Apple', 'Banana', 'Cherry']);

render(
  html`


    ${() => items().map(item => html`- ${item}
`)}
  

  `,
  '#app'
);

```

Directives Reference

Directive	Description
x-data	Define reactive data for a component
x-text	Set element text content
x-html	Set element innerHTML
x-show	Toggle element visibility
x-if	Toggle element hidden attribute
x-bind:* or ;*	Bind attributes
x-on:* or @*	Add event listeners
x-model	Two-way data binding
x-for	Loop over arrays (in templates)
x-ref	Reference elements
{{ }}	Text interpolation

Event Modifiers

```

<button @click.prevent="handleClick">Prevent Default</button>
<button @click.stop="handleClick">Stop Propagation</button>
<button @click.once="handleClick">Run Once</button>
<button @click.self="handleClick">Only Self</button>

```

Store Creation

```

const { createStore } = PhilJS;

const store = createStore({
  count: 0,
  increment() {
    this.count++;
  },
  decrement() {
    this.count--;
  }
});

store.$subscribe(() => {
  console.log('Store updated:', store.count);
});

store.increment();

```

Lifecycle

```

const { onMount, nextTick } = PhilJS;

onMount(() => {
  console.log('DOM is ready');

  return () => {
    console.log('Cleanup on unmount');
  };
});

// Wait for next microtask
await nextTick();

```

API Reference

Reactivity

- `signal<T>(initial)` - Create a reactive signal
- `memo<T>(fn)` - Create a computed/derived value
- `effect(fn)` - Create a side effect
- `batch<T>(fn)` - Batch multiple updates
- `untrack<T>(fn)` - Run without tracking dependencies

Templates

- `html`...`` - Tagged template for HTML
- `render(template, container)` - Render template to DOM

Alpine-style

- `init(root?)` - Initialize directives on DOM
- `createStore(state)` - Create reactive store

Utilities

- `nextTick()` - Wait for next microtask
- `onMount(fn)` - Run on DOM ready

Global Object

```

// All exports available on window.PhilJS
const {
  signal, memo, effect, batch, untrack,
  html, render,
  init, createStore,
  nextTick, onMount,
  version
} = PhilJS;

```

Types

```

interface Signal<T> {
  (): T;
  set: (value: T | ((prev: T) => T)) => void;
  update: (fn: (prev: T) => T) => void;
  peek: () => T;
}

```

Browser Support

- Chrome 80+
- Firefox 74+
- Safari 13.1+
- Edge 80+

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./global`, `./esm`, `./mini`
- Source files: `packages/philjs-cdn/src/index.ts`

Public API

- Direct exports: PhilJS, Signal, batch, createStore, effect, html, init, memo, nextTick, onMount, render, signal, untrack
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/cells - Type: Node package - Purpose: RedwoodJS-style Cells pattern for PhilJS - Declarative data loading components - Version: 0.1.0 - Location: packages/philjs-cells - Entry points: packages/philjs-cells/src/index.ts, packages/philjs-cells/src/cell.ts, packages/philjs-cells/src/context.ts, packages/philjs-cells/src/cache.ts - Keywords: philjs, cells, data-loading, declarative, redwoodjs

PhilJS Cells

RedwoodJS-style Cells pattern for PhilJS - Declarative data loading components with built-in loading, error, empty, and success states.

Features

- **Declarative data loading** - Define what to fetch and how to render each state
- **Built-in state management** - Loading, Error, Empty, and Success states handled automatically
- **No manual loading state** - Focus on your UI, not state management
- **GraphQL & REST support** - Use QUERY for GraphQL or fetch for REST APIs
- **TypeScript first** - Full type inference for your data
- **Cache integration** - Built-in caching with TTL and invalidation
- **SSR support** - Server-side rendering with streaming
- **Refetch on prop change** - Automatic refetching when variables change

Installation

```
npm install philjs-cells
# or
pnpm add philjs-cells
```

Quick Start

Creating a Cell

```
// cells/UsersCell.tsx
import { createCell } from 'philjs-cells';

// Using GraphQL
export const QUERY = `query Users {
  users { id, name, email }
}`;

// Or using fetch
export const fetch = async () => {
  const res = await fetch('/api/users');
  return res.json();
};

export const Loading = () => <Spinner />

export const Empty = () => <p>No users found</p>

export const Failure = ({ error, retry }) => (
  <div>
    <p>Error: {error.message}</p>
    <button onClick={retry}>Retry</button>
  </div>
);

export const Success = ({ users }) => (
  <ul>
    {users.map(user => <li key={user.id}>{user.name}</li>)}
  </ul>
);

export default createCell({
  QUERY, // or fetch
  Loading,
  Empty,
  Failure,
  Success,
});
```

Using a Cell

```

import UsersCell from './cells/UsersCell';

function UsersPage() {
  return (
    <div>
      <h1>Users</h1>
      <UsersCell />
    </div>
  );
}

// With variables
<UserCell id="123" />

```

API Reference

`createCell(definition)`

Creates a Cell component from a definition.

```

interface CellDefinition<TData, TVariables> {
  // Data fetching (choose one)
  QUERY?: string; // GraphQL query
  fetch?: (vars: TVariables) => Promise<TData>; // Fetch function

  // State components
  Loading?: (props: LoadingProps) => VNode;
  Empty?: (props: EmptyProps) => VNode;
  Failure?: (props: FailureProps) => VNode;
  Success: (props: SuccessProps<TData>) => VNode; // Required

  // Options
  afterQuery?: (data: TData) => TData; // Transform data
  isEmpty?: (data: TData) => boolean; // Custom empty check
  displayName?: string; // For debugging
}

```

State Component Props

`LoadingProps`

```

interface LoadingProps {
  attempts: number; // Number of load attempts
}

```

`EmptyProps`

```

interface EmptyProps {
  variables: Record<string, unknown>; // Current variables
}

```

`FailureProps`

```

interface FailureProps {
  error: Error; // The error that occurred
  retryCount: number; // Number of retry attempts
  retry: () => void; // Function to retry
  isRetrying: boolean; // Whether retry is in progress
}

```

`SuccessProps`

```

type SuccessProps<TData> = TData & {
  refetch: () => Promise<void>; // Manual refetch
  isRefetching: boolean; // Refetch in progress
};

```

Cell Props

All cells accept these props:

```

interface CellProps<TVariables> {
  // Variables passed to fetcher
  ...variables: TVariables;

  // Optional overrides
  cacheKey?: string; // Custom cache key
  noCache?: boolean; // Disable caching
  pollInterval?: number; // Auto-refetch interval (ms)
  onSuccess?: (data) => void;
  onError?: (error) => void;
}

```

Provider Setup

`CellProvider`

Wrap your app with `CellProvider` to configure cells:

```

import { CellProvider } from 'philjs-cells';
import { createGraphQLClient } from 'philjs-graphql';

const graphqlClient = createGraphQLClient({
  endpoint: '/graphql',
});

function App() {
  return (
    <CellProvider
      graphqlClient={graphqlClient}
      defaultCacheTTL={5 * 60 * 1000}
      retry={{ maxRetries: 3 }}
    >
      <MyApp />
    </CellProvider>
  );
}

}

```

SSR Support

```

import { CellSSRProvider, getCellHydrationScript } from 'philjs-cells';
import { renderToString } from 'philjs-core/render-to-string';

// Server
const ssrContext = { /* ... */ };
const html = await renderToString(
  <CellSSRProvider>
    <App />
  </CellSSRProvider>
);

const hydrationScript = getCellHydrationScript(ssrContext);
// Inject into HTML before </body>

```

Cache Management

Accessing Cache

```

import { cellCache, useCellInvalidate } from 'philjs-cells';

// Invalidate specific cells
const invalidate = useCellInvalidate();
invalidate('/~user'); // Invalidate all user-related cells

// Warm cache
import { warmCache } from 'philjs-cells';
warmCache({
  'cell:users': prefetchUsers,
  'cell:config': appConfig,
});

```

Prefetching

```

import { useCellPrefetch } from 'philjs-cells';

function UsersList() {
  const prefetch = useCellPrefetch();

  return (
    <ul>
      {users.map(user => (
        <li
          key={user.id}
          onMouseEnter={() => prefetch(UserCell, { id: user.id })}
        >
          {user.name}
        </li>
      ))}
    </ul>
  );
}

```

Advanced Patterns

Typed Cells

```

import { createTypedCell } from 'philjs-cells';

interface UserData {
  user: { id: string; name: string; email: string };
}

interface UserVariables {
  id: string;
}

const createUserCell = createTypedCell<UserData, UserVariables>();

export default createUserCell({
  fetch: async ({ id }) => {
    const res = await fetch(`api/users/${id}`);
    return res.json();
  },
  Success: ({ user }) => <UserProfile user={user} />,
});

```

Cell with Retry

```

import { createCellWithRetry } from 'philjs-cells';

export default createCellWithRetry(
  {
    fetch: fetchUsers,
    Success: UsersList,
  },
  {
    maxRetries: 5,
    retryDelay: 1000,
    backoffMultiplier: 2,
    shouldRetry: (error, attempt) => {
      // Only retry network errors
      return error.message.includes('network');
    },
  }
);

```

Dependent Cells

```

import { createDependentCell } from 'philjs-cells';

// UserPostsCell depends on UserCell data
const UserPostsCell = createDependentCell(UserCell, {
  fetch: async ({ user }) => {
    const res = await fetch(`api/users/${user.id}/posts`);
    return res.json();
  },
  Success: ({ posts }) => <PostsList posts={posts} />,
});

```

CLI Generator

Generate cells using the PhilJS CLI:

```

# Generate a GraphQL cell
philjs generate cell Users

# Generate a fetch-based cell
philjs generate cell Users --fetch

# JavaScript instead of TypeScript
philjs generate cell Users --js

# Custom directory
philjs generate cell Users -d src/components/cells

```

Best Practices

1. **One Cell per data requirement** - Keep cells focused on a single data fetch
2. **Use TypeScript** - Get full type inference for your data
3. **Handle all states** - Always provide Loading, Empty, and Failure components
4. **Use variables for dynamic data** - Pass IDs and filters as props
5. **Leverage caching** - Use appropriate TTL for your data freshness needs
6. **Prefetch on hover** - Improve perceived performance

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./cell`, `./context`, `./cache`
- Source files: `packages/philjs-cells/src/index.ts`, `packages/philjs-cells/src/cell.ts`, `packages/philjs-cells/src/context.ts`, `packages/philjs-cells/src/cache.ts`

Public API

- Direct exports: CacheStats, CellContext, CellProvider, CellProviderProps, CellSSRContext, CellSSRProvider, CellSSRProviderProps, batchInvalidate, cellCache, composeCells, createCell, createCellCacheKey, createCellWithRetry, createDependentCell, createScopedCache, createTypedCell, generateCacheKey, getCellHydrationScript, hydrateCells, initializeCellsFromWindow, inspectCache, logCacheStats, serializeCellData, setupCacheGC, useCellContext, useCellInvalidate, useCellPrefetch, useCellSSR, warmCache
- Re-exported names: CacheStats, CellCache, CellCacheEntry, CellComponent, CellContextValue, CellDataType, CellDefinition, CellFetcher, CellProps, CellProvider, CellProviderConfig, CellQuery, CellSSRContext, CellSSRProvider, CellState, CellStateComponents, CellVariablesType, EmptyCheckFn, EmptyProps, FailureProps, GraphQLResult, LoadingProps, PartialExcept, ReactiveCellState, SuccessProps, batchInvalidate, cellCache, composeCells, createCell, createCellCacheKey, createCellWithRetry, createDependentCell, createScopedCache, createTypedCell, defaultsEmpty, generateCacheKey, getCellHydrationScript, hydrateCells, initializeCellsFromWindow, inspectCache, logCacheStats, serializeCellData, setupCacheGC, useCellContext, useCellInvalidate, useCellPrefetch, useCellSSR, warmCache
- Re-exported modules: ./cache.js, ./cell.js, ./context.js, ./types.js

License

MIT

@philjs/charts - Type: Node package - Purpose: Signal-based chart components for PhilJS with real-time updates and responsive sizing - Version: 0.1.0 - Location: packages/philjs-charts - Entry points: packages/philjs-charts/src/index.ts, ./components, ./hooks, packages/philjs-charts/src/utils/index.ts - Keywords: philjs, charts, signals, recharts, d3, chart.js, visualization, real-time

@philjs/charts

A comprehensive chart library for React applications. Provides beautiful, responsive, and accessible chart components powered by modern rendering techniques.

Installation

```
npm install @philjs/charts
# or
yarn add @philjs/charts
# or
pnpm add @philjs/charts
```

Basic Usage

```
import { LineChart, BarChart, PieChart } from '@philjs/charts';

function Dashboard() {
  const data = [
    { month: 'Jan', value: 400 },
    { month: 'Feb', value: 300 },
    { month: 'Mar', value: 600 },
  ];

  return (
    <div>
      <LineChart data={data} xKey="month" yKey="value" />
      <BarChart data={data} xKey="month" yKey="value" />
      <PieChart data={data} nameKey="month" valueKey="value" />
    </div>
  );
}
```

Features

- **Line Charts** - Single and multi-series line visualizations
- **Bar Charts** - Vertical, horizontal, stacked, and grouped bars
- **Pie Charts** - Pie and donut charts with customizable segments
- **Area Charts** - Filled area charts with gradient support
- **Scatter Plots** - Point-based data visualization
- **Sparklines** - Compact inline charts for dashboards
- **Gauge Charts** - Circular progress and metric displays
- **Responsive** - Automatically adapts to container size
- **Animations** - Smooth transitions and entrance animations
- **Tooltips** - Interactive hover tooltips with customization
- **Legends** - Configurable chart legends
- **Themes** - Light/dark mode and custom theming support
- **Accessibility** - ARIA labels and keyboard navigation

Chart Types

Component	Description
LineChart	Time series and trend data
BarChart	Categorical comparisons
PieChart	Part-to-whole relationships
AreaChart	Volume over time
ScatterChart	Correlation analysis
Sparkline	Inline micro-charts
GaugeChart	Single metric display

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: ., ./components, ./hooks, ./utils
- Source files: packages/philijs-charts/src/index.ts, packages/philijs-charts/src/utils/index.ts

Public API

- Direct exports: // Constants defaultColors, // Core classes Chart, // Hooks useChart, // Types type ChartType, AnimationConfig, AxisConfig, BandScale, CanvasRenderer, ChartConfig, DataPoint, DataSeries, LegendConfig, LinearScale, Renderer, TimeScale, TooltipConfig, UseChartResult, darkTheme, lightTheme, useRealtimeChart
- Re-exported names: (none detected)
- Re-exported modules: ./animations.js, ./colors.js, ./formatters.js

License

MIT

@philijs/cli - Type: Node package - Purpose: CLI tools for PhilJS - The framework that thinks ahead - Version: 0.1.0 - Location: packages/philijs-cli - Entry points: packages/philijs-cli/src/index.ts - Keywords: philijs, cli, framework, build-tool, dev-server - Book coverage: [./tooling/cli.md](#)

philijs-cli

CLI tools for PhilJS - The framework that thinks ahead. Build, develop, and generate code for your PhilJS applications.

Requirements

- **Node.js 24** or higher
- **TypeScript 6** or higher
- **ESM only** - CommonJS is not supported

Features

- **Dev Server** - Fast development server with HMR (Hot Module Replacement)
- **Production Build** - Optimized builds with SSR and SSG support
- **Code Generation** - Scaffolding for components, routes, pages, hooks, and stores
- **Bundle Analysis** - Analyze bundle size and performance
- **Type Generation** - Auto-generate TypeScript types for routes
- **Testing Integration** - Run tests with Vitest
- **Preview Server** - Preview production builds locally

Installation

```
pnpm add -D philijs-cli
```

Or install globally:

```
pnpm add -g philijs-cli
```

Commands

Development

```
philijs dev
```

Start the development server with Hot Module Replacement.

```
philijs dev
philijs dev --port 3000
philijs dev --host 0.0.0.0
philijs dev --open # Open browser automatically
```

Options: - -p, --port <port> - Port to run dev server on (default: 3000) - --host <host> - Host to bind to (default: localhost) - --open - Open browser automatically

Build

```
philijs build
```

Build your application for production.

```
philijs build
philijs build --ssg # Static Site Generation
philijs build --analyze # Analyze bundle size
philijs build --outDir dist
```

Options: - --ssg - Generate static site (SSG mode) - --analyze - Analyze bundle size - --outDir <dir> - Output directory (default: dist)

```
philijs preview
```

Preview production build locally.

```
philijs preview
philijs preview --port 4173
```

Options: - -p, --port <port> - Port to run preview server on (default: 4173)

Code Generation

```
philjs generate component <name>
```

Generate a new component.

```
philjs generate component Button
philjs g c Button # Short alias

# Options
philjs g c Button --directory src/ui
philjs g c Button --no-test # Skip test file
philjs g c Button --with-styles # Include CSS module
philjs g c Button --js # Use JavaScript instead of TypeScript
```

Creates: - src/components/Button.tsx - Component file - src/components/Button.test.tsx - Test file (optional) - src/components/Button.module.css - CSS module (optional)

Options: - -d, --directory <dir> - Target directory (default: src/components) - --no-test - Skip test file generation - --with-styles - Generate CSS module file - --js - Use JavaScript instead of TypeScript

```
philjs generate route <name>
```

Generate a new route with loader.

```
philjs generate route users
philjs g r users

# Options
philjs g r users --directory src/routes
philjs g r users --no-test
```

Creates: - src/routes/users.tsx - Route component with loader - src/routes/users.test.tsx - Test file (optional)

Options: - -d, --directory <dir> - Target directory (default: src/routes) - --no-test - Skip test file generation - --js - Use JavaScript instead of TypeScript

```
philjs generate page <name>
```

Generate a new page component with SEO metadata.

```
philjs generate page About
philjs g p About

# Options
philjs g p About --directory src/pages
```

Creates: - src/pages/About.tsx - Page component with Meta tags - src/pages/About.test.tsx - Test file (optional)

Options: - -d, --directory <dir> - Target directory (default: src/pages) - --no-test - Skip test file generation - --js - Use JavaScript instead of TypeScript

```
philjs generate hook <name>
```

Generate a custom hook.

```
philjs generate hook useCounter
philjs g h useCounter

# Options
philjs g h useCounter --directory src/hooks
```

Creates: - src/hooks/useCounter.ts - Hook implementation - src/hooks/useCounter.test.ts - Test file (optional)

Options: - -d, --directory <dir> - Target directory (default: src/hooks) - --no-test - Skip test file generation - --js - Use JavaScript instead of TypeScript

```
philjs generate store <name>
```

Generate a state store.

```
philjs generate store userStore
philjs g s userStore

# Options
philjs g s userStore --directory src/stores
```

Creates: - src/stores/userStore.ts - Store with signals - src/stores/userStore.test.ts - Test file (optional)

Options: - -d, --directory <dir> - Target directory (default: src/stores) - --no-test - Skip test file generation - --js - Use JavaScript instead of TypeScript

Analysis & Types

```
philjs analyze
```

Analyze bundle size and performance.

```
philjs analyze
```

Shows: - Bundle size breakdown - Code splitting analysis - Performance metrics - Optimization suggestions

```
philjs generate-types
```

Generate TypeScript types for routes.

```
philjs generate-types
```

Generates type-safe route definitions based on your file-based routing structure.

Testing

```
philjs test
```

Run tests with Vitest.

```
philjs test
philjs test --watch # Watch mode
philjs test --coverage # Generate coverage report
```

Options: - --watch - Watch mode for continuous testing - --coverage - Generate coverage report

Configuration

Create a `philjs.config.ts` file in your project root:

```
import { defineConfig } from 'philjs-cli';

export default defineConfig({
  // Development server
  dev: {
    port: 3000,
    host: 'localhost',
    open: true
  },

  // Build options
  build: {
    outDir: 'dist',
    ssg: false,
    sourcemap: true
  },

  // Code generation
  generate: {
    typescript: true,
    componentsDir: 'src/components',
    routesDir: 'src/routes',
    pagesDir: 'src/pages'
  }
});
```

Advanced Configuration with ES2024 Features

```
import { defineConfig, type PluginConfig } from 'philjs-cli';

// Using Object.groupBy() to organize plugins by type
const plugins: PluginConfig[] = [
  { name: 'tailwind', type: 'style' },
  { name: 'i18n', type: 'feature' },
  { name: 'pwa', type: 'feature' },
];

const groupedPlugins = Object.groupBy(plugins, p => p.type);

export default defineConfig({
  plugins: groupedPlugins.feature ?? [],
  styles: groupedPlugins.style ?? [],
});
```

Vite Plugin

Use the PhilJS Vite plugin in your `vite.config.ts`:

```
import { defineConfig } from 'vite';
import philjs from 'philjs-cli/vite';

export default defineConfig({
  plugins: [
    philjs({
      ssr: true,
      islands: true,
      router: 'file-based' // or 'config'
    })
  ]
});
```

Plugin Options: - `ssr` - Enable server-side rendering (default: true) - `islands` - Enable islands architecture (default: true) - `router` - Router mode: 'file-based' or 'config' (default: 'file-based')

Project Structure

Recommended project structure when using `philjs-cli`:

```
my-app/
src/
  components/    # Shared components
  routes/        # Route components with loaders
  pages/         # Page components
  hooks/         # Custom hooks
  stores/        # State stores
  styles/        # Global styles
  entry-client.ts # Client entry point
  entry-server.ts # Server entry point
  public/         # Static assets
  philjs.config.ts # PhilJS configuration
  vite.config.ts   # Vite configuration
  package.json
```

Examples

Generate a complete feature

```
# Create a store
philjs g store todoStore

# Create a component
philjs g component TodoList --with-styles

# Create a route
philjs g route todos

# Create a page with the route
philjs g page Todos
```

Start development

```
# Install dependencies
pnpm install

# Start dev server
philjs dev

# Run tests in watch mode
philjs test --watch
```

Build for production

```
# Build the app
philjs build

# Preview the build
philjs preview

# Analyze bundle
philjs analyze
```

Scripts Integration

Add these scripts to your `package.json`:

```
{
  "scripts": {
    "dev": "philjs dev",
    "build": "philjs build",
    "preview": "philjs preview",
    "test": "philjs test",
    "test:watch": "philjs test --watch",
    "analyze": "philjs analyze",
    "generate": "philjs generate"
  }
}
```

Documentation

For more information, see the PhilJS documentation.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-cli/src/index.ts

Public API

- Direct exports: (none detected)
- Re-exported names: ApiOptions, BuildOptions, ComponentOptions, ContextOptions, DevServerOptions, GeneratorConfig, HookOptions, ModelOptions, PageOptions, PhilJSConfig, RouteOptions, ScaffoldOptions, StoreOptions, TemplateContext, analyze, buildProduction, defineConfig, generateApi, generateComponent, generateContext, generateHook, generateModel, generatePage, generateRoute, generateScaffold, generateStore, generateTypes, getGeneratorConfig, loadConfig, philJSPlugin, philSSSRPlugin, startDevServer

- Re-exported modules: ./analyze.js, ./build.js, ./config.js, ./dev-server.js, ./generate-types.js, ./generators/index.js, ./vite-plugin.js

License

MIT

@philjs/collab - Type: Node package - Purpose: Collaborative state management - CRDT-based real-time collaboration like Figma/Notion - Version: 0.1.0 - Location: packages/philjs-collab - Entry points: packages/philjs-collab/src/index.ts - Keywords: philjs, collaboration, crdt, real-time, multiplayer, presence, figma, notion

@philjs/collab

Real-time collaboration features for React applications. Enables multiple users to work together with live cursors, presence awareness, and synchronized state.

Installation

```
npm install @philjs/collab
# or
yarn add @philjs/collab
# or
pnpm add @philjs/collab
```

Basic Usage

```
import { CollabProvider, usePresence, useSyncedState } from '@philjs/collab';

function App() {
  return (
    <CollabProvider roomId="my-room" userId="user-123">
      <CollaborativeEditor />
    </CollabProvider>
  );
}

function CollaborativeEditor() {
  const { users, cursors } = usePresence();
  const [content, setContent] = useSyncedState('content', '');

  return (
    <div>
      <div>Users online: {users.length}</div>
      <textarea
        value={content}
        onChange={(e) => setContent(e.target.value)}
      />
    </div>
  );
}
```

Features

- **Real-time Sync** - Automatic state synchronization across clients
- **Presence Awareness** - See who is online and their activity
- **Live Cursors** - Display collaborator cursor positions
- **Conflict Resolution** - CRDT-based conflict-free data types
- **Room Management** - Create and manage collaboration rooms
- **User Permissions** - Role-based access control for rooms
- **Offline Support** - Queue changes when disconnected
- **WebSocket Transport** - Low-latency real-time communication
- **History** - Undo/redo with collaborative awareness
- **Typing Indicators** - Show when users are actively editing
- **Comments** - Threaded comments and annotations
- **Version History** - Track and restore previous versions

Hooks

Hook	Description
usePresence	Access online users and cursors
useSyncedState	Synchronized state across clients
useSyncedMap	Collaborative key-value store
useSyncedList	Collaborative array/list
useRoom	Room connection and status

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-collab/src/index.ts

Public API

- Direct exports: CollabRoom, CollabRoomConfig, createCollabRoom

- Re-exported names: ArrayEvent, Awareness, AwarenessConfig, AwarenessState, AwarenessUpdate, BroadcastTransport, COMMENT_REACTIONS, CURSOR_STYLES, CollabMessage, Comment, CommentAuthor, CommentEventHandlers, CommentReaction, CommentThread, CommentsConfig, CommentsManager, CursorConfig, CursorDecoration, CursorManager, CursorPosition, DeleteOp, DeleteSet, InsertOp, Item, ItemId, MapEvent, MessageType, OTClient, OTServer, Operation, OperationWithMeta, PRESENCE_COLORS, PresenceConfig, PresenceManager, PresenceUpdate, RetainOp, StandardAwarenessState, StateVector, TextDelta, ThreadAnchor, TransportConfig, TransportEvents, Update, UserPresence, WebSocketTransport, YArray, YDoc, YMap, YText, applyOperation, applyOperations, compose, createAwareness, createBroadcastTransport, createCommentsManager, createCursorManager, createOTClient, createOTServer, createPresenceManager, createTypedAwareness, createWebSocketTransport, createYDoc, generateClientId, getPresenceColor, injectCursorStyles, invert, transform, transformOperations
- Re-exported modules: ./awareness.js, ./comments.js, ./crdt.js, ./cursors.js, ./ot.js, ./presence.js, ./transport.js

License

MIT

@philjs/compiler - Type: Node package - Purpose: Automatic optimization compiler for PhilJS - zero-overhead memoization and performance optimizations - Version: 0.1.0 - Location: packages/philjs-compiler - Entry points: packages/philjs-compiler/src/index.ts, ./vite, ./rollup, packages/philjs-compiler/src/presets/index.ts, packages/philjs-compiler/src/presets/production.ts, packages/philjs-compiler/src/presets/development.ts, packages/philjs-compiler/src/presets/library.ts - Keywords: philjs, compiler, optimization, memoization, signals, reactive

PhilJS Compiler

Automatic optimization compiler for PhilJS applications. Zero-overhead memoization, automatic batching, and performance optimizations at build time.

Requirements

- **Node.js 24** or higher
- **TypeScript 6** or higher
- **ESM only** - CommonJS is not supported

Features

- **Auto-Memoization** - Automatically wraps expensive computations in `memo()`
- **Auto-Batching** - Batches consecutive signal updates for better performance
- **Dead Code Elimination** - Removes unused reactive bindings
- **Effect Optimization** - Optimizes effect dependencies
- **Component Optimization** - Optimizes component rendering
- **Source Maps** - Full source map support for debugging
- **Zero Runtime Overhead** - All optimizations happen at build time

Installation

```
pnpm add philjs-compiler
```

Usage

With Vite

```
// vite.config.ts
import { defineConfig } from 'vite';
import philjs from 'philjs-compiler/vite';

export default defineConfig({
  plugins: [
    philjs({
      autoMemo: true,
      autoBatch: true,
      deadCodeElimination: true,
      optimizeEffects: true,
      optimizeComponents: true,
      sourceMaps: true,
      verbose: false
    })
  ]
});
```

With Rollup

```
// rollup.config.js
import philjs from 'philjs-compiler/rollup';

export default {
  input: 'src/index.ts',
  output: {
    file: 'dist/bundle.js',
    format: 'esm'
  },
  plugins: [
    philjs({
      autoMemo: true,
      autoBatch: true
    })
  ]
};
```

Programmatic API

```

import { createCompiler, transform, analyzeCode } from 'philjs-compiler';

// Create a compiler instance
const compiler = createCompiler({
  autoMemo: true,
  autoBatch: true
});

// Transform code
const result = compiler.optimize(code, 'src/App.tsx');
console.log(result.code);
console.log('Optimizations applied:', result.optimizations);

// Or use the transform function directly
const result2 = transform(code, 'src/App.tsx', {
  autoMemo: true,
  sourceMaps: true
});

// Analyze code without transforming
const analysis = analyzeCode(code, 'src/App.tsx');
console.log('Components:', analysis.components);
console.log('Reactive bindings:', analysis.reactiveBindings);
console.log('Optimization opportunities:', analysis.optimizationOpportunities);

```

Configuration

```

interface CompilerConfig {
  // Enable automatic memoization
  autoMemo?: boolean;

  // Enable automatic batching
  autoBatch?: boolean;

  // Enable dead code elimination
  deadCodeElimination?: boolean;

  // Optimize effects
  optimizeEffects?: boolean;

  // Optimize components
  optimizeComponents?: boolean;

  // Generate source maps
  sourceMaps?: boolean;

  // Development mode (less aggressive optimizations)
  development?: boolean;

  // File patterns to include
  include?: string[];

  // File patterns to exclude
  exclude?: string[];

  // Custom plugins
  plugins?: CompilerPlugin[];
}

```

How It Works

Auto-Memoization

The compiler detects expensive computations and automatically wraps them in `memo()`:

Before:

```

function ExpensiveComponent() {
  const data = signal([1, 2, 3, 4, 5]);
  const doubled = data().map(x => x * 2); // Re-computed on every render

  return <div>{doubled}</div>;
}

```

After:

```

function ExpensiveComponent() {
  const data = signal([1, 2, 3, 4, 5]);
  const doubled = memo(() => data().map(x => x * 2)); // Memoized!

  return <div>{doubled()}</div>;
}

```

Auto-Batching

The compiler detects consecutive signal updates and wraps them in `batch()`:

Before:

```

function updateUser(name: string, email: string) {
  userName.set(name); // Triggers update
  userEmail.set(email); // Triggers update
  // 2 updates = 2 re-renders
}

```

After:

```

function updateUser(name: string, email: string) {
  batch(() => {
    userName.set(name);
    userEmail.set(email);
  });
  // 1 batched update = 1 re-render
}

```

Dead Code Elimination

The compiler removes unused reactive bindings:

Before:

```

function Component() {
  const unused = signal(42); // Never used
  const count = signal(0);

  return <div>{count()}</div>;
}

```

After:

```

function Component() {
  // 'unused' signal removed
  const count = signal(0);

  return <div>{count()}</div>;
}

```

Performance

The compiler itself is fast and adds minimal overhead to your build:

- **Parsing:** ~0.5ms per file
- **Analysis:** ~0.3ms per file
- **Optimization:** ~0.2ms per file
- **Total:** ~1ms per file on average

For a typical PhilJS app with 100 components, the compiler adds about **100ms** to your build time while potentially improving runtime performance by **10-30%** through automatic optimizations.

Debugging

The compiler generates source maps by default, so you can debug the original code in your browser DevTools, not the optimized output.

To see what optimizations were applied, enable verbose mode:

```

philjs({
  verbose: true
})

```

This will log each optimization:

```

[philjs-compiler] Optimized src/App.tsx in 1.23ms (3 optimizations)
- Auto-memoized: doubled computation
- Auto-batched: updateUser function
- Removed unused: tempSignal

```

Examples

See the `/examples` directory for complete working examples with different build tools.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `., ./vite, ./rollup, ./presets, ./presets/production, ./presets/development, ./presets/library`
- Source files: `packages/philjs-compiler/src/index.ts, packages/philjs-compiler/src/presets/index.ts, packages/philjs-compiler/src/presets/production.ts, packages/philjs-compiler/src/presets/development.ts, packages/philjs-compiler/src/presets/library.ts`

Public API

- Direct exports: `Analyzer, CodeSplitBoundary, CodeSplitReport, CodeSplitter, CompilerConfig, CompilerPlugin, ComponentAnalysis, DeadCodeEliminator, DeadCodeReport, DevBuildReport, DevMetrics, DevPerformanceTracker, DevelopmentPresetOptions, FileAnalysis, LibraryPresetOptions, LibraryValidation, OptimizationOpportunity, Optimizer, ProductionPresetOptions, ReactiveBinding, TransformResult, analyzeCode, calculateCompressionRatio, checkPerformanceBudgets, createCompiler, createDevelopmentPreset, createDevelopmentViteConfig, createLibraryPreset, createLibraryViteConfig, createProductionPreset, createProductionViteConfig, defaultConfig, defaultDevelopmentConfig, defaultLibraryConfig, defaultProductionConfig, formatDevError, formatSize, generatePackageJsonFields, generatePrefetchHints, generatePreloadHints, getDefaultConfig, printDevBuildReport, printLibraryBuildReport, transform, validateConfig, validateLibraryBuild, version`
- Re-exported names: `BundleMetrics, ChunkCandidate, DependencyGraph, DevBuildReport, DevMetrics, DevPerformanceTracker, DevelopmentPresetOptions, HMRErrorType`

LibraryPresetOptions, LibraryValidation, ProductionPresetOptions, calculateCompressionRatio, checkPerformanceBudgets, clearHMRErrorHistory, createDevelopmentPreset, createDevelopmentViteConfig, createLibraryViteConfig, createProductionPreset, createProductionViteConfig, defaultDevelopmentConfig, defaultLibraryConfig, defaultProductionConfig, formatDevError, formatSize, generatePackageJsonFields, generatePrefetchHints, generatePreloadHints, getHMRCClientStats, getHMRErrorHistory, hideHMRErrorOverlay, printDevBuildReport, printLibraryBuildReport, resetHMRCClientStats, setupHMRCClient, showHMRErrorOverlay, validateLibraryBuild

- Re-exported modules: ./analyzer.js, ./development.js, ./hmr-client.js, ./hmr-overlay.js, ./library.js, ./presets/index.js, ./production.js

License

MIT

@philjs/content - Type: Node package - Purpose: Astro-style Content Collections for PhilJS with type-safe schemas and MDX support - Version: 0.1.0 - Location: packages/philjs-content - Entry points: packages/philjs-content/src/index.ts, packages/philjs-content/src/collection.ts, packages/philjs-content/src/query.ts, packages/philjs-content/src/render.ts, ./vite, packages/philjs-content/src/rss.ts, packages/philjs-content/src/sitemap.ts, packages/philjs-content/src/seo.ts, packages/philjs-content/src/utils.ts - Keywords: philjs, content-collections, mdx, markdown, cms, static-content, type-safe, rss, atom, json-feed, sitemap, seo, content-utilities

PhilJS Content

Astro-style Content Collections for PhilJS with type-safe schemas, MDX support, RSS/Atom feeds, SEO utilities, and powerful content manipulation tools.

Overview

Astro-style Content Collections for PhilJS with type-safe schemas and MDX support

Focus Areas

- philjs, content-collections, mdx, markdown, cms, static-content, type-safe, rss, atom, json-feed, sitemap, seo, content-utilities

Entry Points

- packages/philjs-content/src/index.ts
- packages/philjs-content/src/collection.ts
- packages/philjs-content/src/query.ts
- packages/philjs-content/src/render.ts
- ./vite
- packages/philjs-content/src/rss.ts
- packages/philjs-content/src/sitemap.ts
- packages/philjs-content/src/seo.ts
- packages/philjs-content/src/utils.ts

Quick Start

```
import { AlternateLanguage, AtomAuthor, AtomFeedConfig } from '@philjs/content';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- AlternateLanguage
- AtomAuthor
- AtomFeedConfig
- AtomFeedItem
- FeedFromCollectionOptions
- JSONFeedAttachment
- JSONFeedConfig
- JSONFeedItem
- JSONLDArticle
- JSONLDBreadcrumb
- JSONLDEvent
- JSONLDOrganization

Features

- **Type-Safe Content Collections** - Define schemas with Zod for frontmatter validation
- **MDX Support** - Write content in Markdown or MDX with full component support
- **RSS/Atom/JSON Feeds** - Auto-generate feeds from your content
- **XML Sitemaps** - Generate sitemaps with image/video support
- **SEO Utilities** - Meta tags, Open Graph, Twitter Cards, JSON-LD
- **Content Utilities** - Reading time, excerpts, TOC, related posts, tag clouds
- **Hot Module Replacement** - Instant updates in development
- **Vite Integration** - Seamless integration with Vite build pipeline

Installation

```
bun add philjs-content zod
```

See full documentation at [README_NEW.md](#)

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .. ./collection, ./query, ./render, ./vite, ./rss, ./sitemap, ./seo, ./utils
- Source files: packages/philjs-content/src/index.ts, packages/philjs-content/src/collection.ts, packages/philjs-content/src/query.ts, packages/philjs-content/src/render.ts, packages/philjs-content/src/rss.ts, packages/philjs-content/src/sitemap.ts, packages/philjs-content/src/seo.ts, packages/philjs-content/src/utils.ts

Public API

- Direct exports: AlternateLanguage, AtomAuthor, AtomFeedConfig, AtomFeedItem, FeedFromCollectionOptions, JSONFeedAttachment, JSONFeedConfig, JSONFeedItem, JSONLDArticle, JSONLDBreadcrumb, JSONLDEvent, JSONLDOrganization, JSONLDPerson, JSONLDProduct, JSONLDSchema, JSONLDWebPage, JSONLDWebSite, LinkTag, MetaTag, OpenGraphArticle, OpenGraphBook, OpenGraphConfig, OpenGraphImage, OpenGraphProfile, OpenGraphVideo, RSSEnclosure, RSSFeedConfig, RSSFeedImage, RSSFeedItem, RobotsDirective, RouteInfo, SEOConfig, SitemapAlternate, SitemapConfig, SitemapFromCollectionOptions, SitemapImage, SitemapIndexEntry, SitemapUrl, SitemapVideo, TwitterApp, TwitterCardConfig, TwitterPlayer, calculateReadingTime, countEntries, createContentRenderer, defaultComponents, defineCollection, defineCollections, discoverRoutes, extractTableOfContents, findRelatedPosts, formatDate, generateAtom, generateAtomFromCollection, generateExcerpt, generateJSONFeed, generateJSONFeedFromCollection, generateMetaTags, generateRSS, generateRSSFromCollection, generateRobotsTxt, generateSEOFromEntry, generateSitemap, generateSitemapFromCollection, generateSitemapIndex, generateTagCloud, generateUrlsFromRoutes, getAdjacentEntries, getCollection, getCollectionConfig, getCollectionNames, getCollectionTags, getContentDir, getContentStore, getEntries, getEntriesByTag, getEntry, getEntryBySlug, getExcerpt, getFirstWords, getRelativeTime, groupByDate, groupByField, hasCollection, idFromPath, initializeStore, isStoreInitialized, paginate, processForSearch, reference, renderContent, renderTableOfContents, renderTagCloud, renderToString, resetStore, resolveReference, schemas, setCollectionsConfig, setContentDir, slugFromPath, slugify, splitSitemap, stripMarkdown, transformDates, useSEO, validateEntryData, validateRSSFeed, validateSitemap
- Re-exported names: // Build types ProcessedContent, // Collection types CollectionType, // Content metadata ContentHeading, // Entry types CollectionEntry, // Helper types CollectionNames, // Plugin types ContentPluginOptions, // Query types CollectionFilter, // Rendering types RenderResult, // Store types ContentStore, // Watch types ContentWatchEvent, CollectionConfig, CollectionDefinition, CollectionEntryType, CollectionReference, CollectionSort, CollectionsConfig, CommonFrontmatter, ContentBuildResult, ContentEntry, ContentImage, ContentValidationResult, ContentWatchCallback, DataEntry, GetCollectionOptions, ImageOptimizationOptions, InferCollectionData, MDXCompileOptions, MDXComponentProps, MDXComponents, ReferenceSchema, TOCEntry, countEntries, createContentRenderer, defaultComponents, defineCollection, defineCollections, getAdjacentEntries, getCollection, getCollectionConfig, getCollectionNames, getCollectionTags, getContentDir, getContentStore, getEntries, getEntriesByTag, getEntry, getEntryBySlug, getExcerpt, groupBy, hasCollection, idFromPath, initializeStore, isStoreInitialized, processForSearch, reference, renderContent, renderToString, resetStore, resolveReference, schemas, setCollectionsConfig, setContentDir, slugFromPath, transformDates, validateEntryData, z
- Re-exported modules: ./collection.js, ./query.js, ./render.js, ./types.js, zod

```
## @philjs/core - Type: Node package - Purpose: Core signals, memos, and resources for PhilJS - Version: 0.1.0 - Location: packages/philjs-core - Entry points: packages/philjs-core/src/index.ts, packages/philjs-core/src/signals.ts, packages/philjs-core/src/jsx-runtime.ts, packages/philjs-core/src/render-to-string.ts, packages/philjs-core/src/hydrate.ts, packages/philjs-core/src/context.ts, packages/philjs-core/src/error-boundary.ts, packages/philjs-core/src/forms.ts, packages/philjs-core/src/i18n.ts, packages/philjs-core/src/animation.ts, packages/philjs-core/src/accessibility.ts, packages/philjs-core/src/ab-testing.ts, packages/philjs-core/src/result.ts, packages/philjs-core/src/core.ts, packages/philjs-core/src/resumability.ts, packages/philjs-core/src/data-layer.ts, packages/philjs-core/src/service-worker.ts, packages/philjs-core/src/performance-budgets.ts, packages/philjs-core/src/cost-tracking.ts, packages/philjs-core/src/usage-analytics.ts, packages/philjs-core/src/testing.ts, packages/philjs-core/src/superjson.ts, packages/philjs-core/src/superjson-perf.ts, packages/philjs-core/src/plugin-system.ts, packages/philjs-core/src/tiny.ts, packages/philjs-core/src/html.ts, packages/philjs-core/src/element.ts, packages/philjs-core/src/tc39-signals.ts, packages/philjs-core/src/tc39-signals-polyfill.ts, packages/philjs-core/src/view-transitions.ts, packages/philjs-core/src/navigation.ts - Keywords: philjs, signals, reactivity, framework - Book coverage: ./core/components.md, ./core/signals.md, ./core/effects-memos.md
```

philjs-core

Core signals, memos, and resources for PhilJS.

Requirements

- **Node.js 24 or higher**
- **TypeScript 6 or higher**
- **ESM only** - CommonJS is not supported

Installation

```
pnpm add philjs-core
```

Usage

Signals

```
import { signal } from "philjs-core";

const count = signal(0);

console.log(count()); // 0

count.set(5);
console.log(count()); // 5

count.set((prev) => prev + 1);
console.log(count()); // 6

// Subscribe to changes
const unsubscribe = count.subscribe((value) => {
  console.log("Count changed:", value);
});

count.set(10); // Logs: "Count changed: 10"

unsubscribe();
```

Memos

```
import { signal, memo } from "philjs-core";

const count = signal(10);
const doubled = memo(() => count() * 2);

console.log(doubled()); // 20
```

Resources

```
import { resource } from "philjs-core";

let apiData = { value: 1 };
const data = resource(() => apiData);

console.log(data()); // { value: 1 }

apiData = { value: 2 };
data.refresh();

console.log(data()); // { value: 2 }
```

Async Operations with Promise.withResolvers()

```
import { signal } from "philjs-core";

const data = signal(null);

// Using ES2024 Promise.withResolvers() for cleaner async control
async function fetchWithTimeout(url, timeout = 5000) {
    const { promise, resolve, reject } = Promise.withResolvers();

    const timer = setTimeout(() => reject(new Error("Timeout")), timeout);

    fetch(url)
        .then(res => res.json())
        .then(result => {
            clearTimeout(timer);
            data.set(result);
            resolve(result);
        })
        .catch(reject);
}

return promise;
}
```

Grouping Data with Object.groupBy()

```
import { signal, memo } from "philjs-core";

const items = signal([
    { type: "fruit", name: "apple" },
    { type: "vegetable", name: "carrot" },
    { type: "fruit", name: "banana" },
]);

// Using ES2024 Object.groupBy() for cleaner grouping
const grouped = memo(() => Object.groupBy(items(), item => item.type));

console.log(grouped);
// { fruit: [...], vegetable: [...] }
```

Resource Management with using

```
import { signal } from "philjs-core";

// Using TypeScript 6 explicit resource management
function createManagedResource() {
    const state = signal({ active: true });

    return {
        state,
        [Symbol.dispose](): void {
            state.set({ active: false });
            console.log("Resource disposed");
        }
    };
}

function example() {
    using resource = createManagedResource();
    // resource is automatically disposed when scope exits
}
```

TC39 Signals (Native-First)

Use the native Signals implementation when available and lazily load the polyfill when needed.

```
import { getSignalImpl, hasNativeSignals } from '@philjs/core/tc39-signals';

const Signal = await getSignalImpl();
const count = new Signal.State(0);

if (!hasNativeSignals()) {
    console.log('Polyfill loaded for TC39 Signals.');
}
```

If you need a synchronous polyfill in environments without native Signals, use:

```
import { Signal } from '@philjs/core/tc39-signals-polyfill';
```

API

```
signal<T>(initial: T)
```

Creates a reactive signal.

```
memo<T>(calc: () => T)
```

Creates a memoized computation.

```
resource<T>(calc: () => T)
```

Creates a resource that can be refreshed.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `..`, `/signals`, `/jsx-runtime`, `/jsx-dev-runtime`, `/render-to-string`, `/hydrate`, `/context`, `/error-boundary`, `/forms`, `/i18n`, `/animation`, `/accessibility`, `/ab-testing`, `/result`, `/core`, `/resumability`, `/data-layer`, `/service-worker`, `/performance-budgets`, `/cost-tracking`, `/usage-analytics`, `/testing`, `/superjson`, `/superjson-perf`, `/plugin-system`, `/tiny`, `/html`, `/element`, `/tc39-signals`, `/tc39-signals-polyfill`, `/view-transitions`, `/navigation`
- Source files: `packages/philjs-core/src/index.ts`, `packages/philjs-core/src/signals.ts`, `packages/philjs-core/src/jsx-runtime.ts`, `packages/philjs-core/src/jsx-dev-runtime.ts`, `packages/philjs-core/src/render-to-string.ts`, `packages/philjs-core/src/hydrate.ts`, `packages/philjs-core/src/context.ts`, `packages/philjs-core/src/error-boundary.ts`, `packages/philjs-core/src/forms.ts`, `packages/philjs-core/src/i18n.ts`, `packages/philjs-core/src/animation.ts`, `packages/philjs-core/src/accessibility.ts`, `packages/philjs-core/src/ab-testing.ts`, `packages/philjs-core/src/result.ts`, `packages/philjs-core/src/core.ts`, `packages/philjs-core/src/resumability.ts`, `packages/philjs-core/src/data-layer.ts`, `packages/philjs-core/src/service-worker.ts`, `packages/philjs-core/src/performance-budgets.ts`, `packages/philjs-core/src/cost-tracking.ts`, `packages/philjs-core/src/usage-analytics.ts`, `packages/philjs-core/src/testing.ts`, `packages/philjs-core/src/superjson.ts`, `packages/philjs-core/src/superjson-perf.ts`, `packages/philjs-core/src/plugin-system.ts`, `packages/philjs-core/src/tiny.ts`, `packages/philjs-core/src/html.ts`, `packages/philjs-core/src/element.ts`, `packages/philjs-core/src/tc39-signals.ts`, `packages/philjs-core/src/tc39-signals-polyfill.ts`, `packages/philjs-core/src/view-transitions.ts`, `packages/philjs-core/src/navigation.ts`

Public API

- Direct exports: `A11yConfig`, `A11yReport`, `A11yWarning`, `ABTest`, `ABTestConfig`, `ABTestEngine`, `AITranslationService`, `AnimatedValue`, `AnimationOptions`, `BuildConfig`, `BuildOutput`, `BuildResult`, `CacheConfig`, `CacheStrategy`, `Cleanup`, `CloudProvider`, `ComponentUsage`, `CompressedSuperJSONResult`, `CompressionAlgorithm`, `Context`, `CostEstimate`, `CostMetrics`, `CostTracker`, `CustomTypeHandler`, `DeadCodeReport`, `DeserializeOptions`, `Directive`, `EasingFunction`, `EffectCleanup`, `EffectFn`, `EffectFunction`, `Element`, `Err`, `ErrorBoundary`, `ErrorBoundaryProps`, `ErrorCategory`, `ErrorInfo`, `ErrorRecovery`, `ErrorSuggestion`, `Experiment`, `ExperimentAssignment`, `ExperimentEvent`, `ExperimentResults`, `FLIPAnimator`, `FallbackUIPattern`, `FieldProps`, `FieldSchema`, `FocusManager`, `For`, `FormApi`, `FormSchema`, `FormState`, `FormatOptions`, `Fragment`, `GestureHandlers`, `Getter`, `HMROptions`, `I18nConfig`, `I18nProvider`, `IntrinsicAttributes`, `IntrinsicElements`, `JSX`, `JSXElement`, `KeyboardNavigator`, `LZCompression`, `LazyValue`, `LinkedSignal`, `LinkedSignalOptions`, `Locale`, `Memo`, `MutationOptions`, `MutationResult`, `NativeCompression`, `NavigateOptions`, `NavigationCallback`, `NavigationGuard`, `Ok`, `OptimizationSuggestion`, `Part`, `PerformanceBudget`, `PerformanceBudgetManager`, `PerformanceMetrics`, `PhilElementConstructor`, `Plugin`, `PluginConfigSchema`, `PluginContext`, `PluginFactory`, `PluginFileSystem`, `PluginHooks`, `PluginLogger`, `PluginManager`, `PluginMetadata`, `PluginPreset`, `PluginUtils`, `PluralRules`, `PropUsageStats`, `PropertyOptions`, `QueryKey`, `QueryOptions`, `QueryResult`, `RegressionReport`, `RenderResult`, `Resource`, `ResourceFetcher`, `Result`, `ResumableState`, `RouteHandler`, `RouteMatch`, `RouteMetrics`, `RouteParams`, `Router`, `RouterOptions`, `Routes`, `SerializationMeta`, `SerializeOptions`, `SerializedHandler`, `ServiceWorkerConfig`, `Setter`, `Show`, `Signal`, `SignalComputed`, `SignalState`, `SignalWatcher`, `SimilarError`, `SnapshotTester`, `SpringConfig`, `StreamChunk`, `StreamingDeserializer`, `StreamingSerializer`, `SuperJSONResult`, `TargetingRules`, `TemplateResult`, `TinyElement`, `TinySignal`, `TransformResult`, `TranslationExtractor`, `TranslationKey`, `Translations`, `UsageAnalytics`, `UseFormOptions`, `User`, `VNode`, `ValidationError`, `ValidationRule`, `Variant`, `VariantResults`, `ViewTransitionOptions`, `ViewTransitionResult`, `addSkipLink`, `andThen`, `announceToScreenReader`, `assert`, `async`, `attachGestures`, `auditAccessibility`, `batch`, `cache`, `calculateSignificance`, `cleanupHMREffects`, `clearCustomTypes`, `clearHMRState`, `combineProviders`, `composePlugins`, `computed`, `configureA11y`, `costTracker`, `createAnimatedValue`, `createContext`, `createElement`, `createField`, `createFocusManager`, `createLoadingState`, `createLocaleMiddleware`, `createMultivariateTest`, `createMutation`, `createParallax`, `createPlugin`, `createQuery`, `createReducerContext`, `createRoot`, `createRouter`, `createSignalContext`, `createSpy`, `createSuperJSON`, `createTestComponent`, `createTestSignal`, `createThemeContext`, `crossfade`, `css`, `defaultCacheRules`, `definePlugin`, `definePreset`, `deserialize`, `deserializeWithDecompression`, `deserializeWithMetrics`, `easings`, `effect`, `enhanceWithAria`, `errorRecovery`, `expectAll`, `generateElementId`, `generateServiceWorker`, `getA11yConfig`, `getABTestEngine`, `getContrastRatio`, `getCustomTypes`, `getHMRStats`, `getHeadingWarnings`, `getNativeSignal`, `getResumableState`, `getSignalImpl`, `h`, `hasNativeSignals`, `html`, `hydrate`, `initABTesting`, `initDeclarativeTransitions`, `initResumability`, `installSignalPolyfill`, `interceptLinks`, `invalidateQueries`, `isCompressed`, `isErr`, `isHMRInProgress`, `isJSXElement`, `isOk`, `isResult`, `jsxDEV`, `jsxS`, `lazy`, `lazyObject`, `linkedSignal`, `listen`, `loadSignalPolyfill`, `map`, `mapErr`, `matchResult`, `memo`, `mock`, `morph`, `navigate`, `needsSerialization`, `nextTick`, `onCleanup`, `parse`, `perf`, `performanceBudgePlugin`, `performanceBudgets`, `prefersReducedMotion`, `prefetchQuery`, `property`, `query`, `queryCache`, `registerCustomType`, `registerHandler`, `registerServiceWorker`, `registerState`, `render`, `renderToString`, `repeat`, `resetHeadingTracker`, `resource`, `restoreHMRState`, `resumable`, `resume`, `rollbackHMRState`, `scale`, `serialize`, `serializeHandler`, `serializeResumableState`, `serializeWithCompression`, `serializeWithMetrics`, `setupGlobalErrorHandler`, `signal`, `skipWaitingAndClaim`, `slide`, `snapshotHMRState`, `startA11yMonitoring`, `startViewTransition`, `state`, `store`, `stringify`, `supportsNavigationAPI`, `supportsViewTransitions`, `svg`, `unregisterServiceWorkers`, `unsafeHTML`, `until`, `untrack`, `unwrap`, `unwrapOr`, `usageAnalytics`, `useContext`, `useExperiment`, `useFeatureFlag`, `useForm`, `useHash`, `useI18n`, `useLocation`, `useSearchParams`, `useTranslation`, `v`, `validateColorContrast`, `validateHeadingHierarchy`, `wait`, `when`
- Re-exported names: `$$`, `$$ // @deprecated - Use signal() and createSignalContext() instead combineProviders`, `A11yConfig`, `A11yReport`, `A11yWarning`, `ABTest`, `ABTestConfig`, `ABTestEngine`, `AITranslationService`, `APIPerformance`, `AnimatedValue`, `AnimationOptions`, `AsyncMutationOptions`, `AsyncOptions`, `AsyncState`, `AutoRegisterOptions`, `Breadcrumb`, `CSSTextProperties`, `CacheConfig`, `CacheRule`, `CacheStrategy`, `CloudProvider`, `ComponentPerformance`, `ComponentUsage`, `ContentCollection`, `ContentItem`, `ContentWithFrontmatter`, `Context`, `CopyFileOptions`, `CostEstimate`, `CostMetrics`, `CostTracker`, `DeadCodeReport`, `EffectCleanup`, `Err`, `ErrorBoundary`, `ErrorBoundaryProps`, `ErrorCategory`, `ErrorEvent`, `ErrorInfo`, `ErrorRecovery`, `ErrorSuggestion`, `ErrorTracker`, `ErrorTrackingOptions`, `Experiment`, `ExperimentAssignment`, `ExperimentEvent`, `ExperimentResults`, `FLIPAnimator`, `FieldProps`, `FieldSchema`, `FocusManager`, `FormApi`, `FormSchema`, `FormState`, `FormatOptions`, `Fragment`, `GestureHandlers`, `GetStatsOptions`, `GlobEager`, `GlobLazy`, `GlobOptions`, `GlobPathInfo`, `GlobResult`, `HydrationMap`, `I18nConfig`, `I18nProvider`, `JSXChild`, `JSXElement`, `KeyboardNavigator`, `LazyHandler`, `LinkedSignal`, `Locale`, `Memo`, `MutationOptions`, `MutationResult`, `Ok`, `OptimizationSuggestion`, `PathConfig`, `PerformanceBudget`, `PerformanceBudgetManager`, `PerformanceMark`, `PerformanceMetrics`, `PerformanceSnapshot`, `PerformanceTracker`, `PersistConfig`, `Plugin`, `QueryOptions`, `QueryResult`, `ReadDirOptions`, `ReadFileOptions`, `RegressionReport`, `RequestContext`, `Resource`, `ResourcePerformance`, `Result`, `ResumableState`, `RetryConfig`, `RouteMetadata`, `RouteMetrics`, `RouteModule`, `RouteParams`, `RuntimeBudget`, `SanitizeOptions`, `SerializableArray`, `SerializableObject`, `SerializableValue`, `SerializationOptions`, `SerializedHandler`, `SerializedState`, `ServiceWorkerConfig`, `SetStoreFunction`, `Signal`, `SpringConfig`, `Store`, `StoreMiddleware`, `StoreNode`, `StoreOptions`, `TargetingRules`, `TranslationExtractor`, `Translations`, `UsageAnalytics`, `UseFormOptions`, `User`, `UserContext`, `VNode`, `ValidationRule`, `ValidationResults`, `VirtualModuleConfig`, `WatchOptions`, `WebVitalsMetric`, `WebVitalsMonitor`, `WebVitalsOptions`, `WriteFileOptions`, `addBreadcrumb`, `addSkipLink`, `andThen`, `announceToScreenReader`, `assets`, `asyncDisposable`, `attachGestures`, `auditAccessibility`, `autoRegister`, `base`, `batch`, `buildBreadcrumbs`, `buildPath`, `calculateSignificance`, `captureError`, `captureException`, `captureMessage`, `clearCache`, `clearCaches`, `clearSignalRegistry`, `configureA11y`, `configurePaths`, `constantTimeEqual`, `copyFile`, `costTracker`, `createAnimatedValue`, `createAsync`, `createAsyncDisposableScope`, `createAsyncMutation`, `createCollection`, `createConcurrencyLimiter`, `createContext`, `createCspNonce`, `createDir`, `createDisposableMutex`, `createDisposableScope`, `createElement`, `createField`, `createFocusManager`, `createLazyEventHandler`, `createLoadingState`, `createLocaleMiddleware`, `createMultivariateTest`, `createMutation`, `createParallax`, `createPerformanceReport`, `createQuery`, `createQueue`, `createReducerContext`, `createRoot`, `createSignalContext`, `createSlice`, `createStore`, `createStoreWithActions`, `createSuspenseResource`, `createThemeContext`, `createUndoableStore`, `debounceAsync`, `deepClone`, `deepEqual`, `defaultCacheRules`, `deleteDir`, `deleteFile`, `derive`, `deserialize`, `deserializeFromURL`, `dirExists`, `disposableAbortController`, `disposableEventListener`, `disposableInterval`, `disposableSubscription`, `disposableTimeout`, `easings`, `effect`

```

enhanceForm, enhanceWithAria, errorRecovery, escapeAttr, escapeHtml, escapeJs, escapeUrl, exportPerformanceData, extractHydrationData, fileExists, filePathToRoute,
fileUtils, filterByFrontmatter, filterGlob, fromJSON, generateSecureToken, generateServiceWorker, generateVirtualModuleTypes, getA1yConfig, getABTestEngine,
getCacheStats, getCached, getContrastRatio, getDirectory, getErrorStats, getErrorTracker, getExtension, getFilename, getHeadingWarnings, getPerformanceTracker,
getResumableState, getSerializationStats, getStats, getWebVitalsMetrics, getWebVitalsMonitor, glob, globUtils, groupContent, handlerRegistry, hydrate, hydrateFromSSR,
hydrateLazyHandlers, importGlob, importPerformanceData, initABTesting, initErrorTracking, initResumability, initWebVitals, initializePlugins, injectHydrationData,
invalidateCache, invalidateQueries, isErr, isJSElement, isLazyHandler, isOk, isRelativePath, isResult, isValidEmail, joinPaths, jsx, jsxDEV, jsxs, linkedSignal, loadContent,
loadContentWithFrontmatter, loadHandler, loadPlugins, loadRoutes, makeRelative, map, mapErr, mapGlob, matchFiles, matchPath, matchResult, measureAsync,
measureSync, memo, monitorResources, moveFile, normalizePath, onCleanup, parseGlobPath, parseUrl, paths, pathsMatch, performanceBudgetPlugin, performanceBudgets,
persistToLocalStorage, persistentSignal, prefetchHandler, prefetchQuery, preload, produce, queryCache, readDir, readFile, readJSON, reconcile, registerHandler,
registerServiceWorker, registerState, render, renderToString, renderWebVitals, resetHeadingTracker, resolveAsset, resolveLazyHandlers, resolveRoute,
resource, restoreFromLocalStorage, resumable, resume, safeJsonParse, sanitizeHtml, sanitizePath, sanitizeUrl, serialize, serializeForHydration, serializeLazyHandlers,
serializeResumableState, serializeToURL, setCache, setContext, setTag, setUser, setupGlobalErrorHandler, signal, skipWaitingAndClaim, sortByFrontmatter, sortContent,
startA1yMonitoring, subscribeToStore, throttleAsync, toAsyncDisposable, toDisposable, toJSON, unregisterServiceWorkers, untrack, unwrap, unwrapOr, urlSignal,
usageAnalytics, useAPIPerformance, useContext, useCustomPerformance, useErrorBoundary, useExperiment, useFeatureFlag, useForm, useI18n, usePerformance,
usePerformanceBudget, usePerformanceSnapshot, useTranslation, validateColorContrast, validateHeadingHierarchy, validators, virtualModulesPlugin, watchDir, watchFile,
withErrorTracking, writeFile, writeJSON, writeVirtualModuleTypes

• Re-exported modules: ./ab-testing.js, ./accessibility.js, ./animation.js, ./async.js, ./context.js, ./cost-tracking.js, ./data-layer.js, ./disposable.js, ./error-boundary.js, ./error-
tracking.js, ./file-utils.js, ./forms.js, ./glob.js, ./hydrate.js, ./i18n.js, ./jsx-runtime.js, ./lazy-handlers.js, ./paths.js, ./performance-budgets.js, ./performance-tracking.js, ./render-to-
string.js, ./result.js, ./resumability.js, ./security.js, ./serialization.js, ./service-worker.js, ./signals.js, ./store.js, ./types.js, ./usage-analytics.js, ./virtual-modules.js, ./web-vitals.js

```

License

MIT

@philjs/crossdevice - Type: Node package - Purpose: Cross-device state sync for PhilJS - Apple Continuity-like seamless state transfer - Version: 0.1.0 - Location: packages/philjs-crossdevice - Entry points: packages/philjs-crossdevice/src/index.ts - Keywords: philjs, cross-device, sync, handoff, continuity, state-sync, multi-device

@philjs/crossdevice

Cross-Device State Synchronization for PhilJS Applications

Overview

Seamless state sync across all your devices - like Apple Continuity, but for your app state.

Features: - Real-time sync between phone, tablet, desktop - Offline-first with automatic conflict resolution (AES-256-GCM) - Selective state sharing - Device presence and handoff - QR code pairing - P2P sync via WebRTC (optional) - Works without cloud infrastructure

Installation

```
npm install @philjs/crossdevice
```

Quick Start

```

import { CrossDeviceSync, initCrossDevice } from '@philjs/crossdevice';

// Initialize cross-device sync
const sync = await initCrossDevice({
  serverUrl: 'wss://sync.myapp.com',
  encryption: true,
  enableP2P: true
});

// Register state to sync
sync.registerState('theme',
  () => currentTheme,
  (value) => setTheme(value)
);

// Sync manually or automatically
await sync.sync();

```

Usage

Basic Setup

```

import { CrossDeviceSync } from '@philjs/crossdevice';

const sync = new CrossDeviceSync({
  serverUrl: 'wss://sync.example.com',
  enableP2P: true,
  encryption: true,
  deviceName: 'My Laptop',
  syncInterval: 5000,
  conflictResolution: 'last-write-wins'
});

await sync.init();

```

Register State

```
// Register multiple state values
const unsubscribeTheme = sync.registerState(
  'theme',
  () => appTheme.value,
  (value) => { appTheme.value = value; }
);

const unsubscribeCart = sync.registerState(
  'cart',
  () => shoppingCart.items,
  (items) => { shoppingCart.items = items; }
);

// Unsubscribe when done
unsubscribeTheme();
```

Selective Sync

```
const sync = new CrossDeviceSync({
  serverUrl: 'wss://sync.example.com',
  // Only sync specific keys
  syncKeys: ['preferences', 'bookmarks', 'readingProgress']
});
```

Device Discovery

```
// Get all known devices
const devices = sync.getDevices();

// Get online devices
const online = sync.getOnlineDevices();

// Get local device info
const thisDevice = sync.getLocalDevice();

// Subscribe to device changes
sync.onDevicesChange((devices) => {
  console.log('Devices updated:', devices);
});
```

Device Pairing

```
// On primary device: generate pairing code
const pairing = await sync.generatePairingCode();
console.log('Pairing code:', pairing.code);
console.log('Expires:', new Date(pairing.expiresAt));

// On secondary device: enter code to pair
const success = await sync.pairWithCode('ABC123');

// Alternative: QR code pairing
const qrDataUrl = sync.generateQRCode();
// Display qrDataUrl as image
```

Handoff

```
// Initiate handoff from current device
sync.initiateHandoff('document', {
  id: 'doc-123',
  position: 450,
  selectedText: 'some text'
});

// Receive handoff on other device
sync.onHandoff('document', (data) => {
  console.log('Continue editing:', data.data);
  openDocument(data.data.id, data.data.position);
});
```

Conflict Resolution

```
const sync = new CrossDeviceSync({
  // Built-in strategies
  conflictResolution: 'last-write-wins' // or 'first-write-wins' | 'merge'
});

// Custom conflict handling
sync.onConflict((conflict) => {
  console.log('Conflict on key:', conflict.key);
  console.log('Local value:', conflict.localValue);
  console.log('Remote value:', conflict.remoteValue);

  // Return resolved value
  return conflict.remoteValue; // or custom merge logic
});
```

Hooks API

```

import {
  useCrossDeviceState,
  useDevices,
  useHandoff,
  initiateHandoff
} from '@philjs/crossdevice';

// Synced state hook
const [theme, setTheme] = useCrossDeviceState('theme', 'light');

// Devices hook
const devices = useDevices();

// Handoff hook
useHandoff('document', (data) => {
  console.log('Received handoff:', data);
});

// Initiate handoff
initiateHandoff('document', { id: 'doc-123' });

```

API Reference

Classes

CrossDeviceSync

Main synchronization manager.

Methods: - init() - Initialize sync - registerState(key, getter, setter) - Register state for syncing - sync() - Trigger manual sync - getDevices() - Get all devices - getLocalDevice() - Get local device info - getOnlineDevices() - Get online devices - generatePairingCode() - Generate pairing code - pairWithCode(code) - Pair with code - generateQRCode() - Generate QR code data URL - initiateHandoff(type, data) - Start handoff - onHandoff(type, handler) - Handle handoff - onDevicesChange(callback) - Subscribe to device changes - onConflict(handler) - Handle conflicts - destroy() - Cleanup

Functions

- initCrossDevice(config?) - Initialize global instance
- getCrossDeviceSync() - Get global instance
- useCrossDeviceState(key, initial) - Synced state hook
- useDevices() - Devices hook
- useHandoff(type, handler) - Handoff handler hook
- initiateHandoff(type, data) - Initiate handoff
- detectDevice() - Detect device type/platform
- generateDeviceId() - Generate persistent device ID

Types

```

interface CrossDeviceConfig {
  serverUrl?: string;
  enableP2P?: boolean;
  encryption?: boolean;
  encryptionKey?: string;
  deviceId?: string;
  deviceName?: string;
  syncInterval?: number;
  syncKeys?: string[];
  conflictResolution?: 'last-write-wins' | 'first-write-wins' | 'merge' | 'custom';
}

interface Device {
  id: string;
  name: string;
  type: 'phone' | 'tablet' | 'desktop' | 'watch' | 'tv' | 'unknown';
  platform: 'ios' | 'android' | 'windows' | 'macos' | 'linux' | 'web' | 'unknown';
  lastSeen: number;
  isOnline: boolean;
  isPrimary: boolean;
}

interface SyncConflict {
  key: string;
  localValue: unknown;
  remoteValue: unknown;
  localTimestamp: number;
  remoteTimestamp: number;
  localDevice: string;
  remoteDevice: string;
}

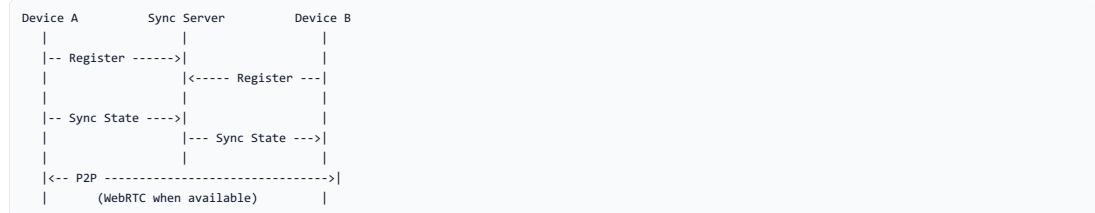
interface HandoffData {
  type: string;
  data: unknown;
  sourceDevice: string;
  timestamp: number;
}

```

Security

- **End-to-End Encryption:** AES-256-GCM encryption
- **Key Exchange:** Secure key sharing via pairing codes
- **No Plain Text:** Data never transmitted unencrypted
- **Local Key Storage:** Encryption keys stored locally

Architecture



API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: `packages/philjs-crossdevice/src/index.ts`

Public API

- Direct exports: `CrossDeviceConfig`, `CrossDeviceSync`, `Device`, `HandoffData`, `PairingInfo`, `SyncConflict`, `SyncState`, `detectDevice`, `generateDeviceId`, `getCrossDeviceSync`, `initCrossDevice`, `initiateHandoff`, `useCrossDeviceState`, `useDevices`, `useHandoff`
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

`## @philjs/css - Type: Node package - Purpose: Type-safe CSS-in-TS solution for PhilJS with zero runtime overhead - Version: 0.1.0 - Location: packages/philjs-css - Entry points: packages/philjs-css/src/index.ts - Keywords: css, css-in-js, css-in-ts, type-safe, atomic-css, design-tokens, theme, variants, zero-runtime`

PhilJS CSS

Type-safe CSS-in-TypeScript solution with zero runtime overhead. Write your styles in TypeScript with full type safety, and extract them to static CSS at build time.

Features

- **Type-Safe CSS** - Full TypeScript type safety for CSS properties
- **Theme Tokens** - Type-safe design tokens for colors, spacing, typography, and more
- **Atomic Classes** - Generate atomic utility classes with type inference
- **CSS Variables** - Type-safe CSS custom properties
- **Media Queries** - Type-safe responsive breakpoints
- **Pseudo Selectors** - Full support for hover, focus, active, and more
- **Variants** - Compound variants system (like Stitches/CVA)
- **Zero Runtime** - All CSS extracted at build time for optimal performance

Installation

```
npm install philjs-css
```

Quick Start

Basic Usage

```
import { css } from 'philjs-css';

const button = css({
  padding: '10px 20px',
  backgroundColor: '#3b82f6',
  color: 'white',
  borderRadius: '4px',
  border: 'none',
  cursor: 'pointer',
  '&:hover': {
    backgroundColor: '#2563eb'
  },
  '&:active': {
    transform: 'scale(0.98)'
  }
});

// Use in your components
<button class={button}>Click me</button>
```

Theme System

Create a type-safe theme with design tokens:

```
import { createTheme } from 'philjs-css';

const theme = createTheme({
  colors: {
    primary: '#3b82f6',
    secondary: '#10b981',
    danger: '#ef4444',
    background: '#ffffff',
    text: '#1f2937'
  },
  spacing: {
    xs: '4px',
    sm: '8px',
    md: '16px',
    lg: '24px',
    xl: '32px'
  },
  fontSize: {
    sm: '14px',
    base: '16px',
    lg: '18px',
    xl: '24px',
    '2xl': '32px'
  },
  fontWeight: {
    normal: 400,
    medium: 500,
    semibold: 600,
    bold: 700
  }
});

// Use theme tokens in styles
const button = css({
  padding: theme.spacing.md,
  backgroundColor: theme.colors.primary,
  fontSize: theme.fontSize.base,
  fontWeight: theme.fontWeight.semibold,
  '&:hover': {
    backgroundColor: theme.colors.secondary
  }
});
```

Variants System

Create components with variants (similar to Stitches or CVA):

```

import { variants } from 'philjs-css';

const button = variants({
  base: {
    padding: '10px 20px',
    borderRadius: '4px',
    border: 'none',
    cursor: 'pointer',
    fontWeight: 500
  },
  variants: {
    size: {
      sm: { padding: '6px 12px', fontSize: '14px' },
      md: { padding: '10px 20px', fontSize: '16px' },
      lg: { padding: '14px 28px', fontSize: '18px' }
    },
    color: {
      primary: {
        backgroundColor: '#3b82f6',
        color: 'white',
        '&:hover': { backgroundColor: '#2563eb' }
      },
      secondary: {
        backgroundColor: '#10b981',
        color: 'white',
        '&:hover': { backgroundColor: '#059669' }
      },
      danger: {
        backgroundColor: '#ef4444',
        color: 'white',
        '&:hover': { backgroundColor: '#dc2626' }
      }
    },
    outline: {
      true: {
        backgroundColor: 'transparent',
        border: '2px solid currentColor'
      }
    }
  },
  compoundVariants: [
    {
      size: 'sm',
      outline: true,
      css: { border: '1px solid currentColor' }
    }
  ],
  defaultVariants: {
    size: 'md',
    color: 'primary'
  }
});

// Usage
<button class={button()}>Default</button>
<button class={button({ size: 'lg', color: 'danger' })}>Large Danger</button>
<button class={button({ outline: true })}>Outline</button>

```

Atomic Utilities

Generate type-safe atomic utility classes:

```

import { createAtomicSystem } from 'philjs-css';

const atoms = createAtomicSystem({
  spacing: {
    0: '0',
    1: '4px',
    2: '8px',
    3: '12px',
    4: '16px',
    6: '24px',
    8: '32px'
  },
  colors: {
    blue: '#3b82f6',
    red: '#ef4444',
    green: '#10b981',
    gray: '#6b7280'
  },
  fontSize: {
    sm: '14px',
    base: '16px',
    lg: '18px',
    xl: '24px'
  }
});

// Usage
<div class={`${atoms.flex} ${atoms.itemsCenter} ${atoms.justifyBetween} ${atoms.p4}`>
  <span class={`${atoms.textLg} ${atoms.fontBold} ${atoms.textBlue}`}>Title</span>
  <button class={`${atoms.bgRed} ${atoms.textSm} ${atoms.p2}`}>Delete</button>
</div>

```

Responsive Design

Use breakpoints for responsive styles:

```

import { css, createBreakpoints } from 'philjs-css';

const breakpoints = createBreakpoints({
  sm: '640px',
  md: '768px',
  lg: '1024px',
  xl: '1280px'
});

const container = css({
  width: '100%',
  padding: '16px',

  [breakpoints.md]: {
    width: '768px',
    margin: '0 auto',
    padding: '24px'
  },

  [breakpoints.lg]: {
    width: '1024px',
    padding: '32px'
  }
});

```

Keyframe Animations

Create reusable animations:

```

import { keyframes, css } from 'philjs-css';

const fadeIn = keyframes({
  from: { opacity: 0, transform: 'translateY(10px)' },
  to: { opacity: 1, transform: 'translateY(0)' }
});

const spin = keyframes({
  from: { transform: 'rotate(0deg)' },
  to: { transform: 'rotate(360deg)' }
});

const animatedBox = css({
  animation: `${fadeIn} 300ms ease-out`,

  '&:hover': {
    animation: `${spin} 1s linear infinite`
  }
});

```

Global Styles

Define global styles for your application:

```

import { globalStyle } from 'philjs-css';

globalStyle('*', {
  margin: 0,
  padding: 0,
  boxSizing: 'border-box'
});

globalStyle('body', {
  fontFamily: 'system-ui, -apple-system, sans-serif',
  lineHeight: 1.5,
  color: '#1f2937',
  backgroundColor: '#ffffff'
});

globalStyle('a', {
  color: "#3b82f6",
  textDecoration: 'none',

  '&:hover': {
    textDecoration: 'underline'
  }
});

```

Slot-Based Variants

Create complex components with multiple styled parts:

```

import { slotVariants } from 'philjs-css';

const card = slotVariants({
  slots: {
    root: {
      borderRadius: '8px',
      overflow: 'hidden',
      backgroundColor: 'white'
    },
    header: {
      padding: '16px',
      borderBottom: '1px solid #e5e7eb',
      fontWeight: 600
    },
    body: {
      padding: '16px'
    },
    footer: {
      padding: '16px',
      backgroundColor: '#f9fafb',
      borderTop: '1px solid #e5e7eb'
    }
  },
  variants: {
    size: {
      sm: {
        root: { maxWidth: '400px' },
        header: { padding: '12px', fontSize: '14px' },
        body: { padding: '12px' },
        footer: { padding: '12px' }
      },
      lg: {
        root: { maxWidth: '800px' },
        header: { padding: '24px', fontSize: '20px' },
        body: { padding: '24px' },
        footer: { padding: '24px' }
      }
    },
    elevated: {
      true: {
        root: { boxShadow: '0 4px 6px rgba(0, 0, 0, 0.1)' }
      }
    }
  },
  defaultVariants: {
    size: 'sm'
  }
});

// Usage
const classes = card({ size: 'lg', elevated: true });

<div class={classes.root}>
  <div class={classes.header}>Card Title</div>
  <div class={classes.body}>Card content goes here</div>
  <div class={classes.footer}>Footer content</div>
</div>

```

CSS Extraction

Extract all CSS to a static file at build time:

```

import { extractCSS } from 'philjs-css';

// Extract all CSS
const css = extractCSS({
  minify: true,
  sourceMap: true,
  atomicClasses: true
});

// Write to file
import { extractToFile } from 'philjs-css';

await extractToFile('./dist/styles.css', {
  minify: true,
  sourceMap: true
});

```

Build Tool Integration

Vite

```

// vite.config.ts
import { createVitePlugin } from 'philjs-css/extract';

export default {
  plugins: [
    createVitePlugin({
      outputPath: 'styles.css',
      minify: true,
      sourceMap: true,
      atomicClasses: true
    })
  ]
};

```

Rollup

```

// rollup.config.js
import { createRollupPlugin } from 'philjs-css/extract';

export default {
  plugins: [
    createRollupPlugin({
      outputPath: './dist/styles.css',
      minify: true
    })
  ]
};

```

Webpack

```

// webpack.config.js
const { createWebpackPlugin } = require('philjs-css/extract');

module.exports = {
  plugins: [
    createWebpackPlugin({
      outputPath: 'styles.css',
      minify: true
    })
  ]
};

```

Critical CSS for SSR

Extract only the CSS used in the current HTML for faster first paint:

```

import { extractCriticalCSS } from 'philjs-css';

const html = renderToString(<App />);
const criticalCSS = extractCriticalCSS(html, { minify: true });

const htmlWithCSS = `
<!DOCTYPE html>
<html>
  <head>
    <style>${criticalCSS}</style>
  </head>
  <body>${html}</body>
</html>
`;

```

Advanced Features

Composing Styles

```

import { css, compose } from 'philjs-css';

const base = css({ padding: '10px', borderRadius: '4px' });
const primary = css({ backgroundColor: 'blue', color: 'white' });
const large = css({ padding: '20px', fontSize: '18px' });

// Compose multiple styles
const button = compose(base, primary, large);

```

Conditional Classes

```

import { cx } from 'philjs-css';

const className = cx(
  'base-class',
  isactive && 'active',
  isdisabled && 'disabled',
  isPrimary ? 'primary' : 'secondary'
);

```

Style Factory

Create reusable style factories with defaults:

```

import { styleFactory } from 'philjs-css';

const createButton = styleFactory({
  padding: '10px 20px',
  borderRadius: '4px',
  border: 'none',
  cursor: 'pointer'
});

const primaryButton = createButton({
  backgroundColor: 'blue',
  color: 'white'
});

const secondaryButton = createButton({
  backgroundColor: 'gray',
  color: 'black'
});

```

Batch Style Creation

```

import { createStyles } from 'philjs-css';

const styles = createStyles({
  container: {
    maxWidth: '1200px',
    margin: '0 auto',
    padding: '20px'
  },
  header: {
    fontSize: '24px',
    fontWeight: 'bold',
    marginBottom: '20px'
  },
  content: {
    lineHeight: 1.6,
    color: '#333'
  }
});

// Use Like: styles.container, styles.header, styles.content

```

Theme Variants (Dark Mode)

```

import { createTheme, createThemeVariant } from 'philjs-css';

const lightTheme = createTheme({
  colors: {
    background: '#ffffff',
    text: '#000000',
    primary: '#3b82f6'
  }
});

const darkTheme = createThemeVariant(lightTheme, 'dark', {
  colors: {
    background: '#1f2937',
    text: '#ffffff',
    primary: '#60a5fa'
  }
});

```

Bundle Analysis

Analyze your CSS bundle to optimize for size:

```
import { analyzeCSSBundle } from 'philjs-css';

const stats = analyzeCSSBundle();

console.log(`Total size: ${stats.totalSize} bytes`);
console.log(`Minified: ${stats.minifiedSize} bytes`);
console.log(`Gzipped (est): ${stats.gzipSize} bytes`);
console.log(`Classes: ${stats.classCount}`);
console.log(`Rules: ${stats.ruleCount}`);
console.log(`Theme vars: ${stats.themeVars}`);
```

TypeScript Support

PhilJS CSS is written in TypeScript and provides full type safety:

```
import { css, type CSSStyleObject } from 'philjs-css';

// Type-safe style objects
const styles: CSSStyleObject = {
  display: 'flex', // Valid
  flexDirection: 'row', // Valid
  // flexDirection: 'invalid', // Type error
};

// Type-safe theme tokens
const theme = createTheme({
  colors: { primary: '#000' }
});

// Autocomplete works
theme.colors.primary; // Autocomplete available
// theme.colors.invalid; // Type error
```

Performance

PhilJS CSS is designed for zero runtime overhead:

- **Build-time extraction** - All CSS is generated at build time
- **Minimal bundle size** - No runtime CSS-in-JS library in your bundle
- **Atomic CSS** - Reuse styles to minimize CSS output
- **Critical CSS** - Extract only what's needed for SSR
- **Minification** - Built-in CSS minification support

Migration Guide

From Emotion/Styled-Components

```
// Before (Emotion)
import { css } from '@emotion/react';

const button = css`
  padding: 10px 20px;
  background-color: blue;
`;

// After (PhilJS CSS)
import { css } from 'philjs-css';

const button = css({
  padding: '10px 20px',
  backgroundColor: 'blue'
});
```

From Tailwind CSS

```
// Before (Tailwind)
<div class="flex items-center justify-between p-4 bg-blue-500 text-white">

// After (PhilJS CSS with atomic utilities)
import { createAtomicSystem } from 'philjs-css';

const atoms = createAtomicSystem({ ... });

<div class={`${atoms.flex} ${atoms.itemsCenter} ${atoms.justifyBetween} ${atoms.p4} ${atoms.bgBlue} ${atoms.textWhite}`}>
```

From Stitches

```
// Before (Stitches)
import { styled } from '@stitches/react';

const Button = styled('button', {
  variants: {
    color: {
      primary: { bg: 'blue' },
      secondary: { bg: 'gray' }
    }
  }
});

// After (PhilJS CSS)
import { variants } from 'philjs-css';

const button = variants({
  variants: {
    color: {
      primary: { backgroundColor: 'blue' },
      secondary: { backgroundColor: 'gray' }
    }
  }
});

```

API Reference

See the [full API documentation](#) for detailed information on all available functions and types.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-css/src/index.ts

Public API

- Direct exports: (none detected)
- Re-exported names: // CSS Anchor Positioning createAnchor, // CSS Color Functions colorMix, // CSS Layers defineLayers, // CSS Nesting processNesting, // Container Queries createContainer, // FLIP technique captureState, // Feature Detection supportsCSS, // Generators generateAllKeyframes, // Keyframe generators slide, // Orchestration calculateStagger, // Presets motionPresets, // Reduced motion reducedMotionStyles, // Scoped Styles scopedStyles, // Scroll-driven Animations scrollTimeline, // Spring physics springPresets, // View Transitions viewTransition, AnchorConfig, AnchorPositionConfig, AnimationTimeline, AtomicConfig, BuildPlugin, BundleStats, CSSProperties, CSSResult, CSSRule, CSSStyleObject, ContainerConfig, ContainerQuery, Direction, ExtractConfig, FLIPState, GestureConfig, GestureEvent, GestureHandler, GestureHandlers, GestureState, GestureType, Keyframe, LayerName, LongPressConfig, MotionConfig, OrchestrationConfig, PanConfig, PanEvent, PinchConfig, PinchEvent, Point, ResponsiveValue, RotateConfig, RotateEvent, ScopeConfig, ScrollTimelineConfig, SpringConfig, StaggerConfig, StyleSheet, SwipeConfig, SwipeEvent, TapConfig, Theme, ThemeTokens, VariantConfig, VariantProps, ViewTimelineConfig, ViewTransitionConfig, analyzeCSSBundle, anchorPosition, applyTheme, atomicDeduplication, atomicRegistry, attachGestures, batchFLIP, batchStyleUpdates, booleanVariant, bounce, calculateSpring, clearStyles, componentScope, compose, containerQuery, cq, createAtomicSystem, createBreakpoints, createCarousel, createColorUtilities, createDynamicStyle, createGestureAnimation, createLayoutUtilities, createReactiveStyle, createRollupPlugin, createSpacingUtilities, createStyles, createTheme, createThemeToggle, createThemeVariant, createTypographyUtilities, createVitePlugin, createWebpackPlugin, css, cssFeatures, cssVar, cx, dataVariants, deduplicateCSS, defaultLayerOrder, defaultTheme, directionVectors, draggablesStyles, easings, extractAllCSS, extractAtomicCSS, extractCSS, extractCriticalCSS, extractCriticalCSSFromFull, extractToFile, extractUsageFromFiles, extractUsageFromHTML, extractUsageFromJSX, fade, featureDetectionCSS, flip, generateAnimationUtilities, generateAtomicClasses, generateLayeredStylesheet, generateOptimizationReport, generateSourceMap, generateThemeCSS, gesturePresets, getCSSVariable, getCriticalSSRStyles, getSSRStyles, getStyleDebugInfo, getTheme, globalStyle, hydrateStyles, injectStyle, keyframes, layer, lightDark, motionSafe, optimizeCSS, parallel, playFLIP, positionFallback, prefersReducedMotion, prefetchStyles, pullToRefresh, pullToRefreshStyles, pulse, purgeUnusedCSS, recipe, relativeColor, removeCSSVariable, removeStyle, resetAtomicRegistry, resetStyles, responsiveVariants, rotate, rubberBand, scale, scrollAnimation, sequence, setCSSVariable, setCSSVariables, shake, slotVariants, splitCSSByRoute, springAnimation, springEasing, staggerAnimation, startViewTransition, stateVariants, styleFactory, styleRegistry, swing, swipeToDismiss, swipeableStyles, syncWithSystemTheme, themeVar, variants, viewTimeline, viewTransitionPresets, wobble, zoomableStyles
- Re-exported modules: ./advanced.js, ./animations.js, ./atomic.js, ./compiler.js, ./css.js, ./extract.js, ./gestures.js, ./runtime.js, ./theme.js, ./types.js, ./variants.js

License

MIT

Contributing

Contributions are welcome! Please read our contributing guide for details.

@philjs/dashboard - Type: Node package - Purpose: Performance monitoring dashboard for PhilJS applications - Version: 0.1.0 - Location: packages/philjs-dashboard - Entry points: packages/philjs-dashboard/src/index.ts, ./collectors - Keywords: philjs, performance, monitoring, dashboard, metrics, web-vitals

@philjs/dashboard

Dashboard components and layouts for building admin panels and analytics interfaces. Provides a flexible grid system with resizable and draggable widgets.

Installation

```
npm install @philjs/dashboard
# or
yarn add @philjs/dashboard
# or
pnpm add @philjs/dashboard
```

Basic Usage

```

import { Dashboard, Widget, DashboardGrid } from '@philjs/dashboard';

function AdminPanel() {
  const layout = [
    { id: 'stats', x: 0, y: 0, w: 6, h: 2 },
    { id: 'chart', x: 6, y: 0, w: 6, h: 4 },
    { id: 'table', x: 0, y: 2, w: 6, h: 4 },
  ];

  return (
    <Dashboard layout={layout} onLayoutChange={console.log}>
      <Widget id="stats" title="Statistics">
        <StatsCard />
      </Widget>
      <Widget id="chart" title="Revenue">
        <RevenueChart />
      </Widget>
      <Widget id="table" title="Recent Orders">
        <OrdersTable />
      </Widget>
    </Dashboard>
  );
}

```

Features

- **Grid Layout** - 12-column responsive grid system
- **Draggable Widgets** - Rearrange dashboard elements via drag-and-drop
- **Resizable Widgets** - Adjust widget dimensions interactively
- **Layout Persistence** - Save and restore user layouts
- **Widget Library** - Pre-built widgets for common use cases
- **Responsive Breakpoints** - Different layouts for desktop/tablet/mobile
- **Fullscreen Mode** - Expand widgets to fullscreen view
- **Widget Actions** - Refresh, settings, and custom actions
- **Theming** - Consistent styling across all widgets
- **Loading States** - Built-in skeleton loading for widgets
- **Error Boundaries** - Graceful error handling per widget

Components

Component	Description
Dashboard	Main container with layout management
Widget	Individual dashboard card/panel
DashboardGrid	Grid layout engine
WidgetHeader	Widget title and action buttons
StatCard	KPI/metric display widget

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./collectors`
- Source files: `packages/philjs-dashboard/src/index.ts`

Public API

- Direct exports: `DashboardInitConfig`, `DashboardInstance`, `VERSION`, `initDashboard`
- Re-exported names: `// Errors ErrorTracker`, `// Local Storage LocalStorageManager`, `// Metrics MetricsCollector`, `// Remote Storage RemoteStorageManager`, `// Tracing TracingManager`, `Alert`, `AlertCondition`, `AlertManager`, `AlertManagerConfig`, `AlertRule`, `AlertSeverity`, `AlertStatus`, `AnomalyCondition`, `AnomalyDetector`, `BatchItem`, `BatchPayload`, `BeaconSender`, `Breadcrumb`, `CPUMetrics`, `CapturedError`, `CombinedStorageConfig`, `CombinedStorageManager`, `ComparisonOperator`, `ConsoleConfig`, `CustomConfig`, `CustomMetric`, `DashboardConfig`, `DashboardDBSchema`, `DashboardData`, `DashboardFilter`, `DashboardTab`, `DataType`, `DatadogConfig`, `DatadogExporter`, `DatadogLog`, `DatadogMetric`, `DatadogRumEvent`, `DatadogTrace`, `EmailConfig`, `ErrorBoundaryInfo`, `ErrorContext`, `ErrorGroup`, `ErrorPatternCondition`, `ErrorRateCondition`, `ErrorTrackerConfig`, `ExportOptions`, `ExportedData`, `GrafanaConfig`, `GrafanaDashboard`, `GrafanaExporter`, `GrafanaPanel`, `LocalStorageConfig`, `LokiLogEntry`, `LokiPushRequest`, `LongTaskAttribution`, `LongTaskEntry`, `MemoryMetrics`, `MetricType`, `MetricsCollectorConfig`, `MetricsSnapshot`, `NetworkRequest`, `NotificationChannel`, `NotificationConfig`, `PRESET_RULES`, `PagerDutyConfig`, `ParsedError`, `PhilChartContainer`, `PhilDashboard`, `PhilMetricCard`, `PrometheusTimeSeries`, `PrometheusWriteRequest`, `RemoteStorageConfig`, `RetentionPolicy`, `SentryBreadcrumb`, `SentryConfig`, `SentryEvent`, `SentryExporter`, `SentrySpan`, `SentryStackFrame`, `SentryTransaction`, `SlackConfig`, `SourceMapResolver`, `Span`, `SpanAttributes`, `SpanBuilder`, `SpanEvent`, `SpanKind`, `SpanLink`, `SpanStatusCode`, `StackFrame`, `StorageMetadata`, `StoredError`, `StoredMetrics`, `StoredSpan`, `TempoSpan`, `ThresholdCondition`, `TimeRange`, `TimeRangeValue`, `TracingContext`, `TracingConfig`, `Userinfo`, `WebVitalsMetrics`, `WebhookConfig`, `calculatePerformanceScore`, `captureReactError`, `createDatadogExporter`, `createGrafanaExporter`, `createSentryExporter`, `generateErrorFingerprint`, `generateWebVitalsDashboard`, `getAlertManager`, `getErrorTracker`, `getLocalStorage`, `getMetricsCollector`, `getRemoteStorage`, `getTracingManager`, `initErrorTracking`, `initRemoteStorage`, `initTracing`, `measureAsync`, `measureSync`, `parseError`, `parseStackTrace`, `resetAlertManager`, `resetErrorTracking`, `resetLocalStorage`, `resetMetricsCollector`, `resetRemoteStorage`, `resetTracing`, `trace`, `traceSync`
- Re-exported modules: `./alerts/index.js`, `./collector/errors.js`, `./collector/metrics.js`, `./collector/tracing.js`, `./integrations/datadog.js`, `./integrations/grafana.js`, `./integrations/sentry.js`, `./storage/local.js`, `./storage/remote.js`, `./ui/Dashboard.js`

License

MIT

@philjs/db - Type: Node package - Purpose: Database integration utilities for PhilJS - Prisma, Drizzle, Supabase, Migrations - Version: 0.1.0 - Location: packages/philjs-db - Entry points: packages/philjs-db/src/index.ts, packages/philjs-db/src/supabase.ts, packages/philjs-db/src/migrations/index.ts - Keywords: philjs, database, prisma, drizzle, supabase,

orm, migrations, schema, postgres, mysql, sqlite, mongodb - Book coverage: [database/migrations.md](#)

philjs-db

Complete database solution for PhilJS applications with ORM integration, migrations, and reactive signals.

Requirements

- **Node.js 24** or higher
- **TypeScript 6** or higher
- **ESM only** - CommonJS is not supported

Features

ORM Integration

- **Prisma ORM** - Type-safe database client with auto-completion
- **Drizzle ORM** - Lightweight TypeScript ORM
- **Supabase** - Backend-as-a-Service with auth and storage
- **Reactive Integration** - Database queries with PhilJS signals
- **Connection Pooling** - Efficient database connections
- **Transaction Support** - Safe multi-query operations
- **Pagination Utilities** - Built-in pagination helpers
- **Soft Delete** - Logical deletion support
- **Repository Pattern** - Clean data access layer

Database Migrations

- **Multi-Database Support** - PostgreSQL, MySQL, SQLite, MongoDB
- **Migration Versioning** - Track and manage migration history
- **Up/Down Migrations** - Full rollback support
- **Transaction Support** - Automatic transaction wrapping
- **Schema Diff** - Automatic schema comparison
- **Auto-Migration** - Generate migrations from model changes
- **CLI Tools** - Full-featured command-line interface
- **ORM Integration** - Seamless Prisma and Drizzle support
- **Backup & Restore** - Automatic database backups

See [MIGRATIONS.md](#) for complete migration documentation.

Installation

```
# Core package
pnpm add philjs-db

# With Prisma
pnpm add @prisma/client
pnpm add -D prisma

# With Drizzle
pnpm add drizzle-orm
pnpm add -D drizzle-kit

# With Supabase
pnpm add @supabase/supabase-js
```

Quick Start - Migrations

```
# Create migration configuration
cat > philjs-db.config.js << EOF
export default {
  type: 'postgres',
  connection: process.env.DATABASE_URL,
  migrationsDir: './migrations',
  tableName: 'migrations',
  transactional: true,
};
EOF

# Create your first migration
npx philjs-db migrate:create --name create_users_table --template table

# Run migrations
npx philjs-db migrate

# Check status
npx philjs-db migrate:status

# Rollback if needed
npx philjs-db migrate:rollback
```

See [MIGRATIONS.md](#) for complete migration documentation.

Quick Start

Prisma

Setup

```
import { createPrismaClient, PrismaProvider } from 'philjs-db';

// Create client
const prisma = createPrismaClient({
  datasourceUrl: process.env.DATABASE_URL
});

// Wrap your app
export default function App() {
  return (
    <PrismaProvider client={prisma}>
      <YourApp />
    </PrismaProvider>
  );
}
```

Usage

```
import { usePrisma } from 'philjs-db';
import { signal } from 'philjs-core';

export default function UserList() {
  const prisma = usePrisma();
  const users = signal([]);

  // Load users
  async function loadUsers() {
    const data = await prisma.user.findMany({
      orderBy: { createdAt: 'desc' }
    });
    users.set(data);
  }

  // Create user
  async function createUser(name: string, email: string) {
    const user = await prisma.user.create({
      data: { name, email }
    });
    users.set([...users(), user]);
  }

  return (
    <div>
      <h1>Users</h1>
      <ul>
        {users().map(user => (
          <li key={user.id}>{user.name} - {user.email}</li>
        ))}
      </ul>
    </div>
  );
}
```

WithLoaders

```
import { withPrisma } from 'philjs-db';
import { defineLoader } from 'philjs-ssr';

export const userLoader = defineLoader(
  withPrisma(async ({ prisma, params }) => {
    const user = await prisma.user.findUnique({
      where: { id: params.id }
    });

    if (!user) {
      throw new Response('Not Found', { status: 404 });
    }

    return user;
  })
);
```

Drizzle

Setup

```

import { createDrizzleClient, DrizzleProvider } from 'philjs-db';
import { drizzle } from 'drizzle-orm/node-postgres';
import { Pool } from 'pg';

const pool = new Pool({
  connectionString: process.env.DATABASE_URL
});

const db = createDrizzleClient(drizzle(pool));

export default function App() {
  return (
    <DrizzleProvider client={db}>
      <YourApp />
    </DrizzleProvider>
  );
}

```

Usage

```

import { useDrizzle } from 'philjs-db';
import { users } from './schema';
import { eq } from 'drizzle-orm';

export default function UserProfile({ userId }: { userId: number }) {
  const db = useDrizzle();
  const user = signal(null);

  async function loadUser() {
    const [userData] = await db
      .select()
      .from(users)
      .where(eq(users.id, userId));

    user.set(userData);
  }

  return <div>{user()?.name}</div>;
}

```

Supabase

Setup

```

import { createSupabaseClient, SupabaseProvider } from 'philjs-db';

const supabase = createSupabaseClient({
  url: process.env.SUPABASE_URL,
  anonKey: process.env.SUPABASE_ANON_KEY!
});

export default function App() {
  return (
    <SupabaseProvider client={supabase}>
      <YourApp />
    </SupabaseProvider>
  );
}

```

Database Queries

```

import { useSupabase } from 'philjs-db';

export default function Posts() {
  const supabase = useSupabase();
  const posts = signal([]);

  async function loadPosts() {
    const { data, error } = await supabase
      .from('posts')
      .select('*')
      .order('created_at', { ascending: false });

    if (error) throw error;
    posts.set(data);
  }

  return <div>...</div>;
}

```

Authentication

```

import { useSupabaseAuth } from 'philjs-db';

export default function LoginForm() {
  const { signIn, signUp, signOut, user, session } = useSupabaseAuth();

  async function handleLogin(email: string, password: string) {
    const { error } = await signIn({ email, password });
    if (error) console.error(error);
  }

  if (user()) {
    return <div>Welcome, {user().email}!</div>;
  }

  return <LoginForm onSubmit={handleLogin} />;
}

```

Storage

```

import { useSupabaseStorage } from 'philjs-db';

export default function ImageUpload() {
  const { upload, download, remove } = useSupabaseStorage('avatars');

  async function handleUpload(file: File) {
    const { data, error } = await upload(`user-${Date.now()}.jpg`, file);
    if (error) throw error;
    return data.path;
  }

  return <input type="file" onChange={(e) => handleUpload(e.target.files[0])}>;
}

```

Utilities

Pagination

```

import { paginate } from 'philjs-db';
import { usePrisma } from 'philjs-db';

export default function PaginatedUsers() {
  const prisma = usePrisma();

  async function getUsers(page = 1) {
    return paginate(
      prisma.user,
      {
        page,
        perPage: 20,
        orderBy: { createdAt: 'desc' }
      }
    );
  }

  const result = await getUsers(1);
  // {
  //   data: [...],
  //   meta: {
  //     page: 1,
  //     perPage: 20,
  //     total: 150,
  //     totalPages: 8
  //   }
  // }
}

}

```

Transactions

```

import { withTransaction } from 'philjs-db';
import { usePrisma } from 'philjs-db';

export default function TransferFunds() {
  const prisma = usePrisma();

  async function transfer(fromId: string, toId: string, amount: number) {
    await withTransaction(prisma, async (tx) => {
      // Deduct from sender
      await tx.account.update({
        where: { id: fromId },
        data: { balance: { decrement: amount } }
      });

      // Add to receiver
      await tx.account.update({
        where: { id: toId },
        data: { balance: { increment: amount } }
      });

      // If any operation fails, entire transaction rolls back
    });
  }
}

```

Using ES2024 Features

`Promise.withResolvers()` for Async Control

```

import { usePrisma } from 'philjs-db';

export async function batchInsertWithProgress(items: Item[]) {
  const prisma = usePrisma();
  const { promise, resolve, reject } = Promise.withResolvers<void>();

  let completed = 0;
  const total = items.length;

  for (const item of items) {
    try {
      await prisma.item.create({ data: item });
      completed++;
      console.log(`Progress: ${completed}/${total}`);
    } catch (error) {
      reject(error);
      return promise;
    }
  }

  resolve();
  return promise;
}

```

`Object.groupBy()` for Query Results

```

import { usePrisma } from 'philjs-db';

export async function getUsersByRole() {
  const prisma = usePrisma();
  const users = await prisma.user.findMany();

  // Group users by role using ES2024 Object.groupBy()
  const grouped = Object.groupBy(users, user => user.role);

  return {
    admins: grouped.admin ?? [],
    users: grouped.user ?? [],
    guests: grouped.guest ?? []
  };
}

```

Resource Management with `using`

```

import { createDatabaseConnection } from 'philjs-db';

// Database connection with automatic cleanup using TypeScript 6 `using`
async function performDatabaseOperation() {
  await using connection = await createDatabaseConnection({
    url: process.env.DATABASE_URL,
    [Symbol.asyncDispose]: async () => {
      await connection.disconnect();
      console.log('Connection closed');
    }
  });

  // Connection automatically closed when scope exits
  return connection.query('SELECT * FROM users');
}

```

Repository Pattern

```

import { createRepository } from 'philjs-db';

const userRepository = createRepository({
  findById: (db, id: string) => {
    return db.user.findUnique({ where: { id } });
  },

  findByEmail: (db, email: string) => {
    return db.user.findUnique({ where: { email } });
  },

  create: (db, data: { name: string; email: string }) => {
    return db.user.create({ data });
  },

  update: (db, id: string, data: Partial<{ name: string; email: string }>) => {
    return db.user.update({ where: { id }, data });
  },

  delete: (db, id: string) => {
    return db.user.delete({ where: { id } });
  }
});

// Usage
const user = await userRepository.findById(prisma, '123');

```

Soft Delete

```

import { softDelete } from 'philjs-db';

// Mark as deleted instead of removing
await softDelete(prisma.user, { id: '123' });

// User is still in database with deletedAt timestamp
const user = await prisma.user.findUnique({ where: { id: '123' } });
console.log(user.deletedAt); // 2024-01-15T12:00:00.000Z

// Exclude soft-deleted records
const activeUsers = await prisma.user.findMany({
  where: { deletedAt: null }
});

```

API Reference

Prisma

- `createPrismaClient(config)` - Create Prisma client instance
- `PrismaProvider` - Context provider for Prisma client
- `usePrisma()` - Access Prisma client in components
- `withPrisma(handler)` - Use Prisma in loaders/actions

Drizzle

- `createDrizzleClient(client)` - Create Drizzle client wrapper
- `DrizzleProvider` - Context provider for Drizzle client
- `useDrizzle()` - Access Drizzle client in components
- `withDrizzle(handler)` - Use Drizzle in loaders/actions

Supabase

- `createSupabaseClient(config)` - Create Supabase client
- `SupabaseProvider` - Context provider for Supabase client
- `useSupabase()` - Access Supabase client in components
- `useSupabaseAuth()` - Authentication helpers
- `useSupabaseStorage(bucket)` - Storage helpers
- `withSupabase(handler)` - Use Supabase in loaders/actions

Utilities

- `paginate(model, options)` - Paginate database queries
- `withTransaction(client, callback)` - Execute transaction
- `createRepository(methods)` - Create data repository
- `softDelete(model, where)` - Soft delete record
- `createDatabaseConnection(config)` - Generic DB connection

Examples

See the [examples](#) directory for full working examples: - SaaS Starter - Prisma with auth - E-commerce - Product catalog - Dashboard - Drizzle ORM

Documentation

For more information, see the PhilJS documentation.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./supabase`, `./migrations`
- Source files: `packages/philjs-db/src/index.ts`, `packages/philjs-db/src/supabase.ts`, `packages/philjs-db/src/migrations/index.ts`

Public API

- Direct exports: `AuthButton`, `FileUpload`, `GET`, `SupabaseConfig`, `SupabaseProvider`, `createSupabaseClient`, `exampleUsage`, `supabase`, `useSupabase`, `useSupabaseAuth`, `useSupabaseRealtime`, `useSupabaseStorage`, `withSupabase`
- Re-exported names: // CRUD operations with provider detection `create`, // Transaction support with `Symbol.asyncDispose` `Transaction`, // Universal `QueryBuilder` `QueryBuilder`, `AutoMigrationGenerator`, `AutoMigrationOptions`, `BackupConfig`, `BackupManager`, `CascadeRelation`, `ColumnBuilder`, `ColumnDefinition`, `ColumnDiff`, `CreateOptions`, `DataMigrationHelper`, `DataMigrationHelpers`, `DatabaseConfig`, `DataProvider`, `DatabaseType`, `DeepPartial`, `DeepRequired`, `DeleteOptions`, `DrizzleMigrationIntegration`, `DrizzleSchemaParser`, `DryRunResult`, `Exact`, `ForeignKeyBuilder`, `ForeignKeyDefinition`, `ForeignKeyDiff`, `IndexDefinition`, `IndexDiff`, `InferInsert`, `InferModel`, `InferSelect`, `InferUpdate`, `JoinClause`, `JoinCondition`, `Migration`, `MigrationBuilder`, `MigrationCLI`, `MigrationConfig`, `MigrationConflict`, `MigrationContext`, `MigrationError`, `MigrationFile`, `MigrationManager`, `MigrationOperation`, `MigrationRecord`, `MigrationResult`, `MigrationStatus`, `MySQLMigrationAdapter`, `Operators`, `OrderByClause`, `PaginatedResult`, `PaginationOptions`, `PostgresMigrationAdapter`, `PrismaMigrationIntegration`, `PrismaSchemaParser`, `QueryOptions`, `Relationship`, `RelationshipBuilder`, `RelationshipType`, `Repository`, `RepositoryOptions`, `SQLiteMigrationAdapter`, `SchemaBuilder`, `SchemaConstraints`, `SchemaDiff`, `SchemaDiffGenerator`, `SchemaValidator`, `Seed`, `SeedConfig`, `SupabaseProvider`, `TableBuilder`, `TableDiff`, `TransactionRollbackError`, `TypeSafeQueryBuilder`, `UpdateOptions`, `UtilsWhereClause`, `ValidationError`, `ValidationResult`, `WhereClause`, `assert`, `createDatabaseConnection`, `createRepository`, `createSupabaseClient`, `createTransaction`, `deleteRecord`, `detectProvider`, `healthCheck`, `is`, `migration`, `omit`, `paginate`, `pick`, `query`, `queryBuilder`, `relationship`, `restore`, `softDelete`, `transaction`, `update`, `useSupabase`, `useSupabaseAuth`, `useSupabaseStorage`, `validator`, `withSupabase`, `withTransaction`
- Re-exported modules: `./adapters/mysql.js`, `./adapters/postgres.js`, `./adapters/sqlite.js`, `./auto-migration.js`, `./cli.js`, `./integrations/drizzle.js`, `./integrations/prisma.js`, `./manager.js`, `./migrations/index.js`, `./schema-diff.js`, `./supabase.js`, `./type-safe-db.js`, `./types.js`, `./utils.js`

License

MIT

@philjs/desktop - Type: Node package - Purpose: Desktop application development for PhilJS using Tauri - Version: 0.1.0 - Location: packages/philjs-desktop - Entry points: packages/philjs-desktop/src/index.ts, packages/philjs-desktop/src/tauri/index.ts, packages/philjs-desktop/src/window.ts, packages/philjs-desktop/src/system/index.ts, packages/philjs-desktop/src/pcts.ts, packages/philjs-desktop/src/lifecycle.ts, packages/philjs-desktop/src/cli.ts, packages/philjs-desktop/src/electron/index.ts - Keywords: philjs, desktop, tauri, electron, cross-platform, native

philjs-desktop

Desktop application development for PhilJS using Tauri. Build native, cross-platform desktop applications with the power of PhilJS and Rust.

Features

- **Tauri Integration** - First-class Tauri 2.0 support
- **Window Management** - Create and manage native windows
- **System APIs** - Dialogs, file system, clipboard, notifications
- **IPC Bridge** - Type-safe JavaScript <-> Rust communication
- **App Lifecycle** - Handle app events and updates
- **Electron Compatibility** - Easy migration from Electron

Installation

```
pnpm add philjs-desktop @tauri-apps/api @tauri-apps/cli
```

Quick Start

Initialize a New Project

```
npx philjs-desktop init my-app
cd my-app
pnpm install
pnpm run tauri:dev
```

Create a Desktop App

```

import { createDesktopApp, useTauri, invoke } from 'philjs-desktop';

createDesktopApp({
  component: App,
  config: {
    appName: 'My Desktop App',
    version: '1.0.0',
    window: {
      width: 1024,
      height: 768,
      title: 'My App',
      center: true,
    },
  },
  onReady: () => {
    console.log('App is ready!');
  },
});

function App() {
  const { isTauri } = useTauri();

  return `<h1>Running in ${isTauri ? 'Tauri' : 'Browser'}</h1>`;
}

```

Window Management

```

import {
  createWindow,
  getCurrentWindow,
  closeWindow,
  minimizeWindow,
  maximizeWindow,
  setTitle,
  setSize,
  setFullscreen,
} from 'philjs-desktop';

// Create a new window
const win = await createWindow({
  title: 'Settings',
  width: 600,
  height: 400,
  center: true,
});

// Control the current window
await setTitle('New Title');
await setSize(1280, 720);
await setFullscreen(true);
await minimizeWindow();
await maximizeWindow();

// Get window info
const window = await getCurrentWindow();
const size = await window.getSize();
const position = await window.getPosition();

```

System APIs

Dialogs

```

import { Dialog, openDialog, saveDialog, showConfirm } from 'philjs-desktop';

// Open file dialog
const files = await openDialog({
  title: 'Select Files',
  multiple: true,
  filters: [
    { name: 'Images', extensions: ['png', 'jpg', 'gif'] },
    { name: 'All Files', extensions: ['*'] },
  ],
});

// Save file dialog
const path = await saveDialog({
  title: 'Save As',
  defaultPath: 'document.txt',
});

// Confirmation dialog
const confirmed = await showConfirm('Are you sure?', {
  title: 'Confirm Action',
  type: 'warning',
});

```

File System

```

import {
  Filesystem,
  readTextfile,
  writeTextfile,
  exists,
  readdir,
} from 'philjs-desktop';

// Read a file
const content = await readTextfile('/path/to/file.txt');

// Write a file
await writeTextfile('/path/to/file.txt', 'Hello, World!', {
  createDirs: true,
});

// Check if file exists
if (await exists('/path/to/file.txt')) {
  console.log('File exists');
}

// List directory contents
const entries = await readdir('/path/to/directory');
for (const entry of entries) {
  console.log(entry.name, entry.isDirectory ? '(dir)' : '(file)');
}

// Watch for changes
const unwatch = await Filesystem.watch('/path/to/watch', (event) => {
  console.log('File changed:', event);
});

```

Shell

```

import { Shell, openUrl, execute, spawn } from 'philjs-desktop';

// Open URL in browser
await openUrl('https://philjs.dev');

// Execute a command
const result = await execute('git', ['status']);
console.log(result.stdout);

// Spawn a long-running process
const process = await spawn('npm', ['run', 'build'], {
  onStdout: (line) => console.log(line),
  onStderr: (line) => console.error(line),
});
await process.kill();

```

Clipboard

```

import { Clipboard, readClipboard, writeClipboard } from 'philjs-desktop';

// Read from clipboard
const text = await readClipboard();

// Write to clipboard
await writeClipboard('Copied text!');

// Check clipboard content
if (await Clipboard.hasImage()) {
  const image = await Clipboard.readImage();
}

```

Notifications

```

import {
  Notification,
  notify,
  requestNotificationPermission,
  scheduleNotification,
} from 'philjs-desktop';

// Request permission
await requestNotificationPermission();

// Show notification
await notify('Hello!', 'This is a notification');

// Schedule a notification
const id = await scheduleNotification({
  title: 'Reminder',
  body: 'Time for a break!',
  at: new Date(Date.now() + 60000), // 1 minute from now
});

```

Global Shortcuts

```
import { GlobalShortcut, registerShortcut } from 'philjs-desktop';

// Register a global shortcut
const unregister = await registerShortcut('CommandOrControl+Shift+S', () => {
    console.log('Shortcut pressed!');
});

// Later, unregister
unregister();
```

System Tray

```
import { SystemTray, trayItem, traySeparator } from 'philjs-desktop';

// Create system tray
await SystemTray.create({
    icon: 'icons/tray.png',
    tooltip: 'My App',
    menu: [
        trayItem('show', 'Show Window', () => window.show()),
        traySeparator(),
        trayItem('quit', 'Quit', () => app.quit()),
    ],
});
```

Auto Launch

```
import { AutoLaunch, toggleAutoLaunch } from 'philjs-desktop';

// Check if enabled
const enabled = await AutoLaunch.isEnabled();

// Toggle auto Launch
await toggleAutoLaunch();
```

IPC Bridge

Type-Safe Commands

```
import { createTypedIPC, invoke } from 'philjs-desktop';

// Define your schema
interface MySchema {
    commands: {
        'get_user': { args: { id: number }; result: User };
        'save_data': { args: { data: string }; result: boolean };
    };
    events: {
        'data_updated': { id: number; data: string };
    };
}

// Create typed IPC
const ipc = createTypedIPC<MySchema>();

// Invoke with type safety
const user = await ipc.invoke('get_user', { id: 123 });

// Listen with type safety
ipc.on('data_updated', (payload) => {
    console.log(payload.id, payload.data);
});
```

Expose JS API to Rust

```
import { exposeToRust, createIPCBridge } from 'philjs-desktop';

// Expose functions
const cleanup = exposeToRust({
    async getData() {
        return { value: 42 };
    },
    async processItem(item: string) {
        return item.toUpperCase();
    },
}, { prefix: 'myApi' });

// Create IPC bridge
const bridge = createIPCBridge({
    commandPrefix: 'app:',
    debug: true,
});

bridge.registerHandler('compute', async (args) => {
    return args.a + args.b;
});
```

App Lifecycle

```
import {
  onAppReady,
  onWindowClose,
  onAppUpdate,
  checkForUpdates,
  installUpdate,
  quitApp,
} from 'philjs-desktop';

// App ready
onAppReady(() => {
  console.log('App started!');
});

// Before window closes
onWindowClose(() => {
  const shouldClose = confirm('Are you sure?');
  return shouldClose;
});

// Check for updates
const update = await checkForUpdates();
if (update) {
  console.log('Update available:', update.version);
  await installUpdate((progress) => {
    console.log(`Downloading: ${progress}%`);
  });
}

// Quit app
quitApp();
```

Electron Migration

philjs-desktop provides compatibility APIs for migrating from Electron:

```
import {
  BrowserWindow,
  ipcMain,
  ipcRenderer,
  app,
  createMigrationHelper,
} from 'philjs-desktop/electron';

// Create windows (Electron-style)
const win = new BrowserWindow({
  width: 800,
  height: 600,
  webPreferences: {
    contextIsolation: true,
  },
});

// IPC (Electron-style)
ipcMain.handle('get-data', async (event, args) => {
  return { success: true };
});

const result = await ipcRenderer.invoke('get-data');

// App Lifecycle (Electron-style)
app.on('ready', () => {
  // Create window
});

app.on('window-all-closed', () => {
  app.quit();
});

// Migration helper
const helper = createMigrationHelper();
const issues = helper.analyzeCode(myElectronCode);
console.log(helper.generateReport(['BrowserWindow', 'ipcMain', 'shell']));
```

CLI Commands

```

# Initialize new project
philjs-desktop init my-app

# Development mode
philjs-desktop dev

# Build for production
philjs-desktop build

# Build for specific platform
philjs-desktop build --target windows
philjs-desktop build --target macos
philjs-desktop build --target linux

```

ES2024 Features

File Operations with using

```

import { FileSystem, readTextFile } from 'philjs-desktop';

// Automatic resource cleanup with TypeScript 6 explicit resource management
async function processfile(path: string) {
    await using handle = await FileSystem.open(path, {
        read: true,
        [Symbol.asyncDispose]: async () => {
            await handle.close();
            console.log('File handle closed');
        }
    });

    return handle.readAll();
}

```

Promise.withResolvers() for IPC

```

import { invoke } from 'philjs-desktop';

// Create cancellable IPC calls using Promise.withResolvers()
function createCancellableInvoke<T>(command: string, args?: unknown) {
    const { promise, resolve, reject } = Promise.withResolvers<T>();
    let cancelled = false;

    invoke<T>(command, args)
        .then(result => !cancelled && resolve(result))
        .catch(error => !cancelled && reject(error));

    return {
        promise,
        cancel: () => { cancelled = true; }
    };
}

const { promise, cancel } = createCancellableInvoke('long_operation', { data: 'test' });
// Later: cancel() to abort

```

Object.groupBy() for Window Management

```

import { getAllWindows, type WindowInfo } from 'philjs-desktop';

async function organizeWindows() {
    const windows = await getAllWindows();

    // Group windows by their state using ES2024 Object.groupBy()
    const grouped = Object.groupBy(windows, (win: WindowInfo) =>
        win.isMinimized ? 'minimized'
        win.isMaximized ? 'maximized'
        : 'normal'
    );

    return {
        minimized: grouped.minimized ?? [],
        maximized: grouped.maximized ?? [],
        normal: grouped.normal ?? []
    };
}

```

Requirements

- **Node.js 24** or higher
- **TypeScript 6** or higher
- **ESM only** - CommonJS is not supported
- **Rust** (for Tauri)
- Platform-specific dependencies (see [Tauri prerequisites](#))

API Reference

Tauri Integration

Function	Description
<code>createDesktopApp(options)</code>	Create a Tauri desktop app
<code>useTauri()</code>	Hook for Tauri APIs access
<code>invoke(command, args)</code>	Call Rust backend commands
<code>listen(event, callback)</code>	Listen to Tauri events
<code>emit(event, payload)</code>	Emit events to Rust
<code>isTauri()</code>	Check if running in Tauri

Window Management

Function	Description
<code>createWindow(options)</code>	Create new window
<code>getCurrentWindow()</code>	Get current window handle
<code>closeWindow()</code>	Close current window
<code>minimizeWindow()</code>	Minimize current window
<code>maximizeWindow()</code>	Maximize current window
<code>setTitle(title)</code>	Set window title
<code>setSize(width, height)</code>	Set window size
<code>setFullscreen(enabled)</code>	Toggle fullscreen
<code>setAlwaysOnTop(enabled)</code>	Keep window on top
<code>center()</code>	Center window
<code>setPosition(x, y)</code>	Set window position

System APIs

Module	Description
<code>Dialog</code>	Open/save dialogs, message boxes
<code>FileSystem</code>	Read/write files, directories
<code>Shell</code>	Open URLs, execute commands
<code>Clipboard</code>	System clipboard access
<code>Notification</code>	System notifications
<code>GlobalShortcut</code>	Register global hotkeys
<code>SystemTray</code>	System tray icon and menu
<code>AutoLaunch</code>	Start on system boot

Lifecycle

Function	Description
<code>onAppReady(callback)</code>	App ready event
<code>onWindowClose(callback)</code>	Before close event
<code>onAppUpdate(callback)</code>	Update available event
<code>checkForUpdates()</code>	Check for app updates
<code>installUpdate()</code>	Install pending update
<code>quitApp()</code>	Quit the application

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./tauri`, `./window`, `./system`, `./ipc`, `./lifecycle`, `./cli`, `./electron`
- Source files: `packages/philijs-desktop/src/index.ts`, `packages/philijs-desktop/src/tauri/index.ts`, `packages/philijs-desktop/src/window.ts`, `packages/philijs-desktop/src/system/index.ts`, `packages/philijs-desktop/src/ipc.ts`, `packages/philijs-desktop/src/lifecycle.ts`, `packages/philijs-desktop/src/cli.ts`, `packages/philijs-desktop/src/electron/index.ts`

Public API

- Direct exports: `AppState`, `CLIOptions`, `IPCBridge`, `IPCBridgeOptions`, `LifecycleEvent`, `Monitor`, `PhysicalPosition`, `PhysicalSize`, `TauriConfig`, `TypedPCSchema`, `UpdateInfo`, `UpdateStatus`, `WindowHandle`, `WindowOptions`, `WindowPosition`, `WindowSize`, `WindowState`, `buildApp`, `center`, `checkForUpdates`, `closeWindow`, `createAppState`, `createChannel`, `createIPCBridge`, `createRequestChannel`, `createTypedIPC`, `createWindow`, `destroyLifecycle`, `exposeToRust`, `getAllMonitors`, `getAllWindows`, `getCurrentWindow`, `getPrimaryMonitor`, `getWindow`, `hideApp`, `initLifecycle`, `initProject`, `installUpdate`, `isAppReady`, `maximizeWindow`, `minimizeWindow`, `onAppReady`, `onAppUpdate`, `onBeforeQuit`, `onBlur`, `onFocus`, `onQuit`, `onUpdateDownloaded`, `onWillQuit`, `onWindowClose`, `parseArgs`, `quitApp`, `registerCommand`, `restartApp`, `setAlwaysOnTop`, `setFullscreen`, `setPosition`, `setSize`, `setTitle`, `showApp`, `startDev`, `useLifecycle`, `useWindow`
- Re-exported names: `// App creation`, `createDesktopApp`, `// Auto Launch`, `AutoLaunch`, `// Clipboard`, `Clipboard`, `// Command invocation`, `invoke`, `// Context and hooks`, `initTauriContext`, `// Dialog`, `Dialog`, `// Events listen`, `Events`, `// File System`, `FileSystem`, `// Global Shortcut`, `GlobalShortcut`, `// Notification`, `Notification`, `// Shell`, `Shell`, `// System Tray`, `SystemTray`, `ApplInfo`, `AppState`, `AutoLaunch`, `AutoLaunchOptions`, `BaseDirectory`, `BrowserWindow`, `BrowserWindowOptions`, `Clipboard`, `ClipboardError`, `CommandDefinition`, `CommandOptions`, `CommandOutput`, `ConfirmDialogOptions`, `CopyOptions`, `DesktopAppOptions`, `Dialog`, `DialogFilter`, `ElectronToTauriMapper`, `Event`, `EventCallback`

FileEntry, FileSystem, GlobalShortcut, IPCBridge, IPCBridgeOptions, InvokeArgs, IpcEvent, LifecycleEvent, MessageDialogOptions, Monitor, Notification, NotificationAction, NotificationError, NotificationOptions, OpenDialogOptions, PhysicalPosition, PhysicalSize, ReadOptions, SaveDialogOptions, ScheduledNotificationOptions, Shell, ShortcutHandler, SpawedProcess, SystemTray, TauriConfig, TauriContext, TauriEventType, TauriEvents, TauriPlugin, Tray, TrayClickEvent, TrayMenuItem, TrayOptions, TypedCommand, TypedIPCSchema, UnlistenFn, UpdateInfo, UpdateStatus, WebContentsLike, WindowConfig, WindowHandle, WindowOptions, WindowPosition, WindowSize, WindowState, WriteOptions, app, batchInvoke, cancelNotification, center, checkForUpdates, clearClipboard, clipboard, closeWindow, contextBridge, copyFile, createAppState, createChannel, createCommand, createDefaultConfig, createDesktopApp, createDir, createEventEmitter, createIPCBridge, createMigrationHelper, createRequestChannel, createTray, createTypedIPC, createTypedListener, createWindow, defineCommand, destroyLifecycle, destroyTray, dialog, disableAutoLaunch, emit, enableAutoLaunch, execute, exists, exposeToRust, getAllMonitors, getAllWindows, getAppName, getAppVersion, getCurrentWindow, getLoadedPlugins, getPrimaryMonitor, getTauriContext, getTauriVersion, getWindow, globalShortcut, hideApp, hideTray, initLifecycle, initTauriContext, installUpdate, invoke, invokeWithRetry, invokeWithTimeout, ipcMain, ipcRenderer, isAppInitialized, isAppReady, isAutoLaunchEnabled, isShortcutRegistered, isTauri, listen, maximizeWindow, minimizeWindow, notify, onAppReady, onAppUpdate, onBeforeClose, onBeforeQuit, onBlur, onFocus, onQuit, onTauriEvent, onUpdatedDownloaded, onWillQuit, onWindowClose, once, openDialog, openPath, openUrl, powershell, quitApp, readBinaryFile, readClipboard, readClipboardImage, readDir, readTextFile, registerCommand, registerShortcut, removeAllEventListeners, removeAllListeners, removeDir, removeFile, rename, requestNotificationPermission, resetTauriContext, restartApp, runScript, saveDialog, scheduleNotification, setAlwaysOnTop, setFullscreen, setPosition, setSize, setTitle, setTrayIcon, setTrayMenu, setTrayTooltip, shell, showApp, showAsk, showConfirm, showMessage, showNotification, showTray, sidebar, spawn, stat, toggleAutoLaunch, trayItem, traySeparator, traySubmenu, unregisterAllShortcuts, unregisterShortcut, useLifecycle, useTauri, useWindow, waitForEvent, watchPath, writeBinaryFile, writeClipboard, writeClipboardImage, writeTextFile

- Re-exported modules: ./system/clipboard.js, ./system/dialog.js, ./system/notification.js, ./system/shell.js, ./system/shortcut.js, ./system/tray.js, ./app.js, ./autolaunch.js, ./browser-window.js, ./clipboard.js, ./context.js, ./dialog.js, ./events.js, ./filesystem.js, ./invoke.js, ./ipc.js, ./lifecycle.js, ./migration.js, ./notification.js, ./shell.js, ./shortcut.js, ./system/index.js, ./tauri/index.js, ./tray.js, ./types.js, ./window.js

License

MIT

@philjs/devtools - Type: Node package - Purpose: Developer tools overlay for PhilJS - Version: 0.1.0 - Location: packages/philjs-devtools - Entry points: packages/philjs-devtools/src/index.ts - Keywords: philjs, developer-tools, debugging - Book coverage: [./tooling/devtools.md](#)

philjs-devtools

Developer tools overlay for PhilJS with time-travel debugging, signal inspection, and performance monitoring.

Features

- **Time-Travel Debugging** - Rewind and replay state changes
- **Redux DevTools Integration** - Connect to Redux DevTools Extension
- **Action Replay** - Record and replay user actions
- **State Persistence** - Save/restore state across sessions
- **Signal Inspector** - Track signal values and dependencies
- **Component Tree Viewer** - Visualize component hierarchy
- **Performance Monitoring** - Track render times and performance budgets
- **Debug Logger** - Advanced logging with filtering and formatting
- **Hydration Map** - See which islands are hydrated
- **Route Inspector** - View current route, params, and errors
- **Development Overlay** - In-browser DevTools panel

Installation

```
pnpm add philjs-devtools
```

Quick Start

Basic DevTools Overlay

Add the overlay to your app during development:

```
import { showOverlay } from 'philjs-devtools';

if (import.meta.env.DEV) {
  showOverlay();
}
```

This displays a panel showing: - Number of islands and hydrated components - Current route and params - Bundle size - Real-time stats

Time-Travel Debugging

Debug state changes by rewinding and replaying:

```

import { initTimeTravel, debugSignal } from 'philjs-devtools';
import { signal } from 'philjs-core';

// Initialize time-travel
const timeTravelDebugger = initTimeTravel({
  maxHistory: 100,
  captureStackTraces: true
});

// Create a signal with debugging
const count = debugSignal(signal(0), 'count');

// Make changes
count.set(1);
count.set(2);
count.set(3);

// Rewind to previous state
timeTravelDebugger.jumpTo(1); // count is now 1
console.log(count()); // 1

// Fast-forward
timeTravelDebugger.jumpTo(2); // count is now 2

// View timeline
console.log(timeTravelDebugger.getTimeline());
// [
//   { index: 0, timestamp: ..., signalId: 'count', value: 0 },
//   { index: 1, timestamp: ..., signalId: 'count', value: 1 },
//   { index: 2, timestamp: ..., signalId: 'count', value: 2 },
//   { index: 3, timestamp: ..., signalId: 'count', value: 3 }
// ]

```

Signal Inspector

Track signal dependencies and values:

```

import { createSignalInspector } from 'philjs-devtools';
import { signal, memo } from 'philjs-core';

const inspector = createSignalInspector();

const count = signal(0);
const doubled = memo(() => count() * 2);

inspector.watch(count, 'count');
inspector.watch(doubled, 'doubled');

count.set(5);

// View all tracked signals
console.log(inspector.getSignals());
// [
//   { id: 'count', value: 5, type: 'signal', dependencies: [] },
//   { id: 'doubled', value: 10, type: 'memo', dependencies: ['count'] }
// ]

// Get signal history
console.log(inspector.getHistory('count'));
// [0, 5]

```

Component Tree Viewer

Visualize your component hierarchy:

```

import { createComponentTree, renderTree } from 'philjs-devtools';

const tree = createComponentTree();

// Track component mounts
tree.mount('App', null);
tree.mount('Header', 'App');
tree.mount('Nav', 'Header');
tree.mount('Main', 'App');
tree.mount('Footer', 'App');

// Print the tree
console.log(renderTree(tree));
// App
//   Header
//     Nav
//   Main
//   Footer

// Track unmounts
tree.unmount('Nav');

// Get component info
const appInfo = tree.getComponent('App');
console.log(appInfo);
// {
//   id: 'App',
//   parent: null,
//   children: ['Header', 'Main', 'Footer'],
//   mountTime: 1234567890,
//   renderCount: 1
// }

```

Performance Monitoring

Track render performance and set budgets:

```

import { createPerformanceMonitor } from 'philjs-devtools';

const monitor = createPerformanceMonitor({
  budgets: {
    'App': 16, // 16ms budget for 60fps
    'Dashboard': 50
  },
  warningThreshold: 0.8 // Warn at 80% of budget
});

// Track render
monitor.startRender('App');
// ... render logic ...
monitor.endRender('App');

// Check if over budget
if (monitor.isOverBudget('App')) {
  console.warn('App exceeded performance budget!');
}

// Get metrics
const metrics = monitor.getMetrics('App');
console.log(metrics);
// {
//   averageTime: 12.5,
//   maxTime: 18,
//   minTime: 8,
//   renderCount: 10,
//   overBudgetCount: 2
// }

// Get all slow components
const slowComponents = monitor.getSlowComponents();
console.log(slowComponents);
// [{ component: 'Dashboard', avgTime: 52, budget: 50 }]

```

Debug Logger

Advanced logging with categories and filtering:

```

import { createDebugLogger } from 'philjs-devtools';

const logger = createDebugLogger({
  enabled: import.meta.env.DEV,
  categories: {
    render: true,
    state: true,
    router: false
  }
});

// Log with categories
logger.log('render', 'Rendering App component');
logger.warn('state', 'Signal updated:', { count: 5 });
logger.error('router', 'Navigation failed'); // Won't Log (disabled)

// Group Logs
logger.group('User Login');
logger.log('auth', 'Validating credentials...');
logger.log('auth', 'Fetching user data...');
logger.groupEnd();

// Time operations
logger.time('fetchData');
await fetchData();
logger.timeEnd('fetchData'); // Logs: "fetchData: 245ms"

// Enable/disable categories at runtime
logger.setCategory('router', true);
logger.log('router', 'Now logging router events');

```

Redux DevTools Integration

Connect your PhilJS app to the Redux DevTools Extension:

```

import { initReduxDevTools } from 'philjs-devtools';
import { signal } from 'philjs-core';

// Initialize Redux DevTools
const devTools = initReduxDevTools(
  { count: 0, user: null }, // Initial state
  {
    name: 'MyApp',
    maxAge: 50, // Keep last 50 actions
    trace: true, // Include stack traces
    actionsBlacklist: ['PING'], // Ignore certain actions
  }
);

// Create signals
const count = signal(0);
const user = signal(null);

// Track state changes
count.subscribe((value) => {
  const state = { count: value, user: user() };
  devTools.send({ type: 'INCREMENT' }, state);
});

user.subscribe((value) => {
  const state = { count: count(), user: value };
  devTools.send({ type: 'SET_USER', payload: value }, state);
});

// Handle time travel from DevTools
devTools.onStateChange = (state) => {
  count.set(state.count);
  user.set(state.user);
};

```

Features: - Time travel debugging - Action history - State inspection - Action replay - State import/export - Pause/resume tracking

Action Replay

Record and replay user actions:

```

import { ActionReplayer } from 'philjs-devtools';

const replayer = new ActionReplayer();

// Record actions
function handleIncrement() {
  const action = { type: 'INCREMENT' };
  const state = { count: count() + 1 };

  replayer.record(action, state);
  count.set(state.count);
}

// Replay all actions
await replayer.replay((action, state) => {
  console.log(`Replaying: ${action.type}`);
  count.set(state.count);
}, 1000); // 1 second between actions

// Get recorded actions
const actions = replayer.getActions();
console.log(`Recorded ${actions.length} actions`);

// Clear recording
replayer.clear();

```

State Persistence

Save and restore state across sessions:

```

import { StatePersistence } from 'philjs-devtools';

const persistence = new StatePersistence({
  key: 'my-app-state',
  storage: localStorage,
  version: 1,
  migrate: (state, version) => {
    // Handle version upgrades
    if (version === 0) {
      return { ...state, newField: 'default' };
    }
    return state;
  },
});

// Save state
const currentState = { count: 42, user: { name: 'John' } };
persistence.save(currentState);

// Load state (on app startup)
const loadedState = persistence.load();
if (loadedState) {
  count.set(loadedState.count);
  user.set(loadedState.user);
}

// Clear persisted state
persistence.clear();

```

State Differing

Compare state snapshots:

```

import { diffState } from 'philjs-devtools';

const prevState = {
  user: { name: 'Alice', age: 30 },
  todos: [{ id: 1, done: false }]
};

const nextState = {
  user: { name: 'Alice', age: 31 },
  todos: [{ id: 1, done: true }, { id: 2, done: false }]
};

const diff = diffState(prevState, nextState);
console.log(diff);
// [
//   { type: 'updated', path: 'user.age', oldValue: 30, newValue: 31 },
//   { type: 'updated', path: 'todos[0].done', oldValue: false, newValue: true },
//   { type: 'added', path: 'todos[1]', newValue: { id: 2, done: false } }
// ]

```

API Reference

DevTools Overlay

- showOverlay() - Display the DevTools overlay panel

Time-Travel Debugging

- `initTimeTravel(config?)` - Initialize time-travel debugger
- `getTimeTravelDebugger()` - Get the global time-travel debugger instance
- `debugSignal(signal, id)` - Wrap a signal with time-travel debugging
- `diffState(prev, next)` - Compare two state objects

TimeTravelDebugger methods: - `jumpTo(index)` - Jump to a specific point in history - `stepBack()` - Go back one step - `stepForward()` - Go forward one step - `getTimeline()` - Get full history timeline - `getSnapshot(index)` - Get state snapshot at index - `reset()` - Clear all history

Redux DevTools Integration

- `initReduxDevTools(initialState, config?)` - Initialize Redux DevTools
- `getReduxDevTools()` - Get the global Redux DevTools instance
- `disconnectReduxDevTools()` - Disconnect and cleanup

ReduxDevTools methods: - `send(action, state)` - Send action and state to DevTools - `disconnect()` - Disconnect from DevTools Extension - `getSnapshot()` - Get current state snapshot - `getHistory()` - Get action history - `getDiff(fromIndex, toIndex)` - Get state diff between indices - `exportState()` - Export state as JSON - `importState(json)` - Import state from JSON

ReduxDevTools properties: - `currentState` - Signal with current state - `isConnected` - Signal indicating connection status - `isLocked` - Signal indicating if DevTools is locked - `isPaused` - Signal indicating if tracking is paused - `onStateChange` - Callback for time travel events - `onCustomAction` - Callback for custom actions from DevTools

Action Replay

- `ActionReplayer` - Class for recording and replaying actions

ActionReplayer methods: - `record(action, state)` - Record an action - `replay(onAction, speed?)` - Replay recorded actions - `stop()` - Stop replay - `clear()` - Clear recorded actions - `getActions()` - Get all recorded actions - `isPlaying()` - Check if replay is in progress

State Persistence

- `StatePersistence` - Class for persisting state

StatePersistence methods: - `save(state)` - Save state to storage - `load()` - Load state from storage - `clear()` - Clear persisted state

Signal Inspector

- `createSignalInspector()` - Create a signal inspector instance

SignalInspector methods: - `watch(signal, id)` - Start tracking a signal - `unwatch(id)` - Stop tracking a signal - `getSignals()` - Get all tracked signals - `getHistory(id)` - Get value history for a signal - `getDependencies(id)` - Get signal dependencies

Component Tree

- `createComponentTree()` - Create a component tree tracker
- `renderTree(tree)` - Render tree as a string

ComponentTree methods: - `mount(id, parent?)` - Track component mount - `unmount(id)` - Track component unmount - `getComponent(id)` - Get component info - `getChildren(id)` - Get component children - `getTree()` - Get full component tree

Performance Monitoring

- `createPerformanceMonitor(config?)` - Create a performance monitor

PerformanceMonitor methods: - `startRender(component)` - Start timing a render - `endRender(component)` - End timing a render - `getMetrics(component)` - Get performance metrics - `isOverBudget(component)` - Check if component exceeded budget - `getSlowComponents()` - Get list of slow components - `reset()` - Clear all metrics

Debug Logger

- `createDebugLogger(config?)` - Create a debug logger

DebugLogger methods: - `log(category, ...args)` - Log a message - `warn(category, ...args)` - Log a warning - `error(category, ...args)` - Log an error - `group(label)` - Start a log group - `groupEnd()` - End a log group - `time(label)` - Start a timer - `timeEnd(label)` - End a timer and log duration - `setCategory(category, enabled)` - Enable/disable a category

Browser Extension

PhilJS also has a browser extension for DevTools integration. See [philjs-devtools-extension](#) for more information.

Best Practices

1. **Only use in development** - DevTools add overhead, disable in production
2. **Track meaningful signals** - Don't debug every signal, focus on problematic ones
3. **Set performance budgets** - Know your targets and monitor them
4. **Use categories for logging** - Organize logs by feature/module
5. **Clean up inspectors** - Unwatch signals when no longer needed
6. **Limit history size** - Set reasonable `maxHistory` to avoid memory issues

Examples

See the demo app for a working example with DevTools enabled.

Documentation

For more information, see the PhilJS documentation.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-devtools/src/index.ts

Public API

- Direct exports: showOverlay
- Re-exported names: ActionReplayer, BundleProfile, ChunkInfo, ComponentInspector, ComponentNode, DevToolsState, DiffType, DuplicateInfo, ElementHighlighter, FlameNode, InspectorConfig, InspectorEvent, MemoryProfile, ModuleInfo, NetworkProfile, PerformanceInfo, PerformancePanel, PersistenceConfig, ProfilerConfig, PropInfo, PropsPanel, ReduxAction, ReduxDevTools, ReduxDevToolsConfig, RenderProfile, SearchBar, StateDiff, StateInfo, StatePanel, StatePersistence, StateSnapshot, StyleInfo, StylePanel, TimeTravelConfig, TimeTravelDebugger, TimelineNode, analyzeMemoryUsage, analyzeNetworkRequests, analyzeRenderPerformance, captureMemorySnapshot, createInspector, debugSignal, diffState, disconnectReduxDevTools, exportProfileData, generateFlameGraph, getInspector, getReduxDevTools, getTimeTravelDebugger, importProfileData, initReduxDevTools, initTimeTravel, recordMemo, recordRenderEnd, recordRenderStart, startMemoryProfiling, startNetworkProfiling, startProfiling, stopMemoryProfiling, stopNetworkProfiling, stopProfiling
- Re-exported modules: ./inspector/index.js, ./inspector/types.js, ./profiler.js, ./redux-devtools.js, ./time-travel.js

License

MIT

@philjs/devtools-extension - Type: Node package - Purpose: Chrome DevTools Extension for PhilJS - inspect signals, components, and performance - Version: 0.1.0 - Location: packages/philjs-devtools-extension

PhilJS DevTools Extension

Browser DevTools extension for PhilJS.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: (none detected)
- Source files: packages/philjs-devtools-extension/src/index.ts

Public API

- Direct exports: (none detected)
- Re-exported names: ComponentNode, ComponentTree, DevToolsPanel, DevToolsState, NetworkInspector, PerformanceMetrics, PerformanceProfiler, SignalData, SignalInspector, connectDevTools, disconnectDevTools, isDevToolsConnected
- Re-exported modules: ./client/connector.js, ./panel/ComponentTree.js, ./panel/DevToolsPanel.js, ./panel/NetworkInspector.js, ./panel/PerformanceProfiler.js, ./panel/SignalInspector.js, ./types.js

@philjs/digital-twin - Type: Node package - Purpose: IoT device synchronization for PhilJS - device shadows, telemetry, predictive maintenance - Version: 0.1.0 - Location: packages/philjs-digital-twin - Entry points: packages/philjs-digital-twin/src/index.ts - Keywords: philjs, digital-twin, iot, mqtt, telemetry, device-shadow

@philjs/digital-twin

IoT device synchronization for PhilJS - device shadows, telemetry, predictive maintenance

Overview

IoT device synchronization for PhilJS - device shadows, telemetry, predictive maintenance

Focus Areas

- philjs, digital-twin, iot, mqtt, telemetry, device-shadow

Entry Points

- packages/philjs-digital-twin/src/index.ts

Quick Start

```
import { Alert, ConnectionConfig, DeviceCommand } from '@philjs/digital-twin';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- Alert
- ConnectionConfig
- DeviceCommand
- DeviceConfig
- DeviceEvent
- DeviceEventCallback
- DeviceState
- DigitalTwin
- FleetManager
- MaintenancePrediction
- PredictiveMaintenance
- PropertyDefinition

Install

```
pnpm add @philjs/digital-twin
```

Usage

```
import { Alert, ConnectionConfig, DeviceCommand } from '@philjs/digital-twin';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-digital-twin/src/index.ts

Public API

- Direct exports: Alert, ConnectionConfig, DeviceCommand, DeviceConfig, DeviceEvent, DeviceEventCallback, DeviceState, DigitalTwin, FleetManager, MaintenancePrediction, PredictiveMaintenance, PropertyDefinition, PropertyMetadata, TelemetryData, TimeSeriesStore, useDigitalTwin, useFleet, usePredictiveMaintenance, useTelemetry
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/dnd - Type: Node package - Purpose: Drag and drop toolkit for PhilJS - multi-container, accessible, animated - Version: 0.1.0 - Location: packages/philjs-dnd - Entry points: packages/philjs-dnd/src/index.ts, packages/philjs-dnd/src/core/index.ts, packages/philjs-dnd/src/sensors/index.ts, packages/philjs-dnd/src/utils/index.ts - Keywords: philjs, drag-and-drop, dnd, sortable, kanban, accessibility

@philjs/dnd

Drag and drop functionality for React applications. A lightweight, accessible, and performant solution for creating interactive drag-and-drop interfaces.

Installation

```
npm install @philjs/dnd
# or
yarn add @philjs/dnd
# or
pnpm add @philjs/dnd
```

Basic Usage

```
import { DndProvider, Draggable, Droppable } from '@philjs/dnd';

function App() {
  const handleDrop = (dragId, dropId) => {
    console.log(`Dropped ${dragId} onto ${dropId}`);
  };

  return (
    <DndProvider onDrop={handleDrop}>
      <Droppable id="list">
        <Draggable id="item-1">
          <div>Drag me!</div>
        </Draggable>
        <Draggable id="item-2">
          <div>Drag me too!</div>
        </Draggable>
      </Droppable>
    </DndProvider>
  );
}
```

Features

- **Lightweight** - Minimal bundle size with no heavy dependencies
- **Touch Support** - Works on mobile and touch devices
- **Keyboard Accessible** - Full keyboard navigation support
- **Sortable Lists** - Built-in sortable list functionality
- **Multiple Drop Zones** - Support for multiple drop targets
- **Custom Drag Preview** - Customize the drag overlay appearance
- **Constraints** - Limit drag movement to axes or containers
- **Auto-scroll** - Automatic scrolling near container edges
- **Collision Detection** - Multiple algorithms for drop detection
- **Sensors** - Mouse, touch, and keyboard sensors
- **Animations** - Smooth drop animations
- **Nested Droppables** - Support for nested drop zones

Hooks

Hook	Description
useDraggable	Make an element draggable
useDroppable	Create a drop zone
useSortable	Sortable list item
useDndContext	Access drag state
useDragOverlay	Custom drag preview

Components

Component	Description
DndProvider	Context provider for drag-and-drop
Draggable	Wrapper for draggable elements
Droppable	Drop zone container
SortableList	Pre-built sortable list
DragOverlay	Custom drag preview layer

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./core`, `/sensors`, `/utils`
- Source files: `packages/philjs-dnd/src/index.ts`, `packages/philjs-dnd/src/core/index.ts`, `packages/philjs-dnd/src/sensors/index.ts`, `packages/philjs-dnd/src/utils/index.ts`

Public API

- Direct exports: (none detected)
- Re-exported names: `// Animation functions applyDropAnimation`, `// Basic modifiers restrictToHorizontalAxis`, `// CSS transition helpers getTransitionString`, `// Collision detection algorithms rectIntersection`, `// Composite modifiers composeModifiers`, `// Conditional modifiers conditionalModifier`, `// Default animations defaultDropAnimation`, `// Easing functions easings`, `// FLIP animation captureFlipState`, `// Keyframes shakeKeyframes`, `// Snap modifiers snapToGrid`, `// Transform modifiers scaleMovement`, `// Types type LayoutShiftAnimation`, `// Utilities clamp`, `// Utilities getTransformValues`, `// Utility functions getRect`, `CollisionArgs`, `CollisionDetection`, `DelayedMouseSensor`, `DistanceMouseSensor`, `DndConfig`, `DndManager`, `DragCancelEvent`, `DragEndEvent`, `DragItem`, `DragMoveEvent`, `DragOverEvent`, `DragStartEvent`, `DragState`, `DropTarget`, `FlipState`, `ImmediateTouchSensor`, `KeyboardCoordinates`, `KeyboardSensor`, `KeyboardSensorOptions`, `LongPressSensor`, `Modifier`, `ModifierArgs`, `MouseSensor`, `MouseSensorOptions`, `PhilDndContext`, `PhilDragOverlay`, `PhilDraggable`, `PhilDroppable`, `PointerSensor`, `Position`, `Rect`, `Sensor`, `SensorDescriptor`, `SensorFactory`, `SensorOptions`, `TouchSensor`, `TouchSensorOptions`, `VimKeyboardSensor`, `WasdKeyboardSensor`, `addMomentum`, `animateLayoutShift`, `applyEasing`, `bounceDropAnimation`, `clearTransform`, `closestCenter`, `closestCorners`, `createBoundingBox`, `createCompoundCollision`, `createDndManager`, `createKeyframeAnimation`, `createTypeFilter`, `defaultLayoutShiftAnimation`, `fadeInKeyframes`, `fadeOutKeyframes`, `fastDropAnimation`, `getArea`, `getCenter`, `getDistance`, `getDndManager`, `getDroppableCenter`, `getIntersectionArea`, `getNextDroppableId`, `horizontalListSorting`, `invertMovement`, `percentageOverlap`, `playFlipAnimation`, `pointerWithin`, `preventScrolling`, `pulseKeyframes`, `rectIntersection`, `rectsIntersect`, `removeTransition`, `restrictToFirstScrollableAncestor`, `restrictToParentElement`, `restrictToVerticalAxis`, `restrictToWindowEdges`, `scaleDownKeyframes`, `scaleUpKeyframes`, `setTransform`, `slideInFromBottomKeyframes`, `slideInFromTopKeyframes`, `slowDropAnimation`, `snapCenterToContainer`, `snapToCustomGrid`, `snapToGrid`, `springDropAnimation`, `typeBasedModifier`, `verticalListSorting`, `waitForTransition`
- Re-exported modules: `./types.js`, `./DndContext.js`, `./DragOverlay.js`, `./Draggable.js`, `./Droppable.js`, `./animations.js`, `./collision.js`, `./core/DndContext.js`, `./core/DragOverlay.js`, `./core/Draggable.js`, `./core/Droppable.js`, `./keyboard.js`, `./modifiers.js`, `./mouse.js`, `./sensors/keyboard.js`, `./sensors/mouse.js`, `./sensors/touch.js`, `./touch.js`, `./types.js`, `./utils/collision.js`, `./utils/modifiers.js`

License

MIT

@philjs/docs - Type: Node package - Purpose: PhilJS Documentation Site - philjs.dev - Version: 0.1.0 - Location: packages/philjs-docs - Status: workspace-only (private package)

PhilJS Documentation

The official documentation site for PhilJS, built with the PhilJS docs pipeline and TypeScript-first tooling.

Features

- Comprehensive documentation for TypeScript and Rust APIs
- Interactive code playground with live preview
- Full-text search powered by Fuse.js
- Dark mode support
- Mobile-responsive design
- MDX support for documentation pages

Getting Started

Prerequisites

- Node.js 24+ (Node 25 supported)
- pnpm 9+

Development

```
# Install dependencies
pnpm install

# Start development server
pnpm dev
```

The site will be available at `http://localhost:3000`.

Build

```
# Build for production
pnpm build

# Start production server
pnpm start
```

Deployment

Quick Deploy

Vercel (Recommended)

```
# Deploy to production
pnpm deploy:vercel

# Deploy preview
pnpm deploy:vercel:preview
```

Cloudflare Pages

```
# Deploy to Cloudflare Pages
pnpm deploy:cloudflare
```

Make sure you have the Wrangler CLI installed:

```
npm install -g wrangler
wrangler login
```

Netlify

```
# Deploy to production
pnpm deploy:netlify

# Deploy preview
pnpm deploy:netlify:preview
```

Make sure you have the Netlify CLI installed:

```
npm install -g netlify-cli
netlify login
```

Configuration Files

- **vercel.json** - Vercel deployment configuration with caching, redirects, and region settings
- **wrangler.toml** - Cloudflare Pages configuration
- **netlify.toml** - Netlify deployment configuration with Next.js plugin

Environment Variables

Variable	Description	Required
NEXT_PUBLIC_SITE_URL	Production site URL	No
STATIC_EXPORT	Enable static HTML export for Cloudflare	No
NEXT_PUBLIC_GA_ID	Google Analytics measurement ID	No
NEXT_PUBLIC_ALGOLIA_APP_ID	Algolia DocSearch app ID	No
NEXT_PUBLIC_ALGOLIA_API_KEY	Algolia DocSearch API key	No

Project Structure

```

packages/philjs-docs/
src/
  app/          # Next.js App Router pages
  docs/         # Documentation pages
    getting-started/
    core-concepts/
    guides/
    api/
    tutorials/
    rust-guide/
    examples/    # Example gallery
    playground/  # Interactive playground
  components/   # Shared components
    Header.tsx  # Site header with navigation
    Sidebar.tsx # Documentation sidebar
    Search.tsx  # Search dialog
    CodeBlock.tsx # Syntax-highlighted code blocks
    Playground.tsx # Interactive code playground
    APIReference.tsx # API documentation component
  public/        # Static assets
  vercel.json   # Vercel configuration
  wrangler.toml # Cloudflare configuration
  netlify.toml  # Netlify configuration
  next.config.mjs # Next.js configuration
  tailwind.config.ts # Tailwind CSS configuration

```

Documentation Sections

Getting Started

- Installation guide
- Quick start tutorials (TypeScript and Rust)
- Project structure overview
- IDE setup

Core Concepts

- Signals and reactivity
- Components
- Effects and memos
- Stores
- Server-side rendering

Guides

- SSR & Hydration
- Islands Architecture
- LiveView
- Routing
- Forms
- Styling
- State Management
- Data Fetching
- Authentication
- Deployment

API Reference

- philjs-core
- philjs-router
- philjs-srr
- philjs-forms
- Component Library (philjs-ui)

Rust Guide

- Rust quickstart
- cargo-philjs CLI
- View macro syntax
- Server functions
- Axum integration
- WASM deployment

Scripts

Script	Description
<code>pnpm dev</code>	Start development server
<code>pnpm build</code>	Build for production
<code>pnpm build:static</code>	Build static export (for Cloudflare)
<code>pnpm start</code>	Start production server
<code>pnpm lint</code>	Run ESLint
<code>pnpm typecheck</code>	Run TypeScript type checking
<code>pnpm deploy:vercel</code>	Deploy to Vercel (production)
<code>pnpm deploy:vercel:preview</code>	Deploy to Vercel (preview)
<code>pnpm deploy:cloudflare</code>	Deploy to Cloudflare Pages
<code>pnpm deploy:netlify</code>	Deploy to Netlify (production)
<code>pnpm deploy:netlify:preview</code>	Deploy to Netlify (preview)

Contributing

1. Fork the repository
2. Create a feature branch: `git checkout -b feature/my-feature`
3. Make your changes
4. Run tests and linting: `pnpm lint && pnpm typecheck`
5. Commit your changes: `git commit -m 'Add new feature'`
6. Push to the branch: `git push origin feature/my-feature`
7. Open a Pull Request

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: (none detected)
- Source files: (none detected)

Public API

- Direct exports: (none detected)
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT License - see LICENSE for details.

@philjs/edge - Type: Node package - Purpose: Edge computing primitives for PhilJS - KV storage, Durable Objects, queues, cron - Version: 0.1.0 - Location: packages/philjs-edge - Entry points: packages/philjs-edge/src/index.ts, ./kv, ./durable, ./queue, ./d1, packages/philjs-edge/src/geo-routing.ts, packages/philjs-edge/src/prefetch.ts, packages/philjs-edge/src/streaming.ts, packages/philjs-edge/src/state-replication.ts, packages/philjs-edge/src/smarty-cache.ts - Keywords: philjs, edge, cloudflare, workers, kv, durable-objects, d1

@philjs/edge

Edge runtime adapters for deploying PhilJS applications to edge computing platforms. Enables low-latency, globally distributed application deployments.

Installation

```
npm install @philjs/edge
# or
yarn add @philjs/edge
# or
pnpm add @philjs/edge
```

Basic Usage

```
import { createEdgeHandler, EdgeRequest, EdgeResponse } from '@philjs/edge';

export default createEdgeHandler(async (req: EdgeRequest) => {
  const data = await fetchData(req.geo.country);

  return EdgeResponse.json({
    message: 'Hello from the edge!',
    region: req.geo.region,
    data,
  });
});
```

Features

- **Multi-Platform Support** - Deploy to Vercel, Cloudflare, Deno Deploy, and more
- **Geo-Location** - Access visitor location data at the edge
- **Low Latency** - Serve requests from the nearest edge location
- **Edge KV** - Key-value storage at the edge
- **Streaming** - Stream responses for large payloads

- **WebSocket Support** - Real-time connections at the edge
- **Middleware** - Composable edge middleware
- **Request Rewriting** - URL rewriting and redirects
- **A/B Testing** - Traffic splitting at the edge
- **Bot Detection** - Identify and filter bot traffic
- **Rate Limiting** - Edge-based rate limiting
- **Cache Control** - Fine-grained caching strategies

Platform Adapters

Adapter	Platform
@philjs/edge/vercel	Vercel Edge Functions
@philjs/edge/cloudflare	Cloudflare Workers
@philjs/edge/deno	Deno Deploy
@philjs/edge/netlify	Netlify Edge Functions
@philjs/edge/fastly	Fastly Compute@Edge

Edge Utilities

```
import { geoip, edgeCache, rateLimit } from '@philjs/edge';

// Geo-Location Lookup
const location = geoip(request);

// Edge caching
const cached = await edgeCache.get('key');

// Rate Limiting
const allowed = await rateLimit(request, { limit: 100 });
```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: ., ./kv, ./durable, ./queue, ./d1, ./geo-routing, ./prefetch, ./streaming, ./state-replication, ./smart-cache
- Source files: packages/philjs-edge/src/index.ts, packages/philjs-edge/src/geo-routing.ts, packages/philjs-edge/src/prefetch.ts, packages/philjs-edge/src/streaming.ts, packages/philjs-edge/src/state-replication.ts, packages/philjs-edge/src/smash-cache.ts

Public API

- Direct exports: AccessPattern, CacheConfig, CacheEntry, CacheStats, CronJob, CronScheduler, D1Database, D1ExecResult, D1PreparedStatement, D1Result, DurableObjectState, DurableStorage, EDGE_LOCATIONS, EdgeKVOptions, EdgeNode, EdgePrefetcher, EdgeQueue, GCounter, GeoLocation, GeoPattern, GeoRouter, KVGetOptions, KVListOptions, KVListResult, KVPutOptions, KVStore, LWWRegister, LWWSet, MemoryDurableStorage, MemoryKVStore, MemoryQueue, NavigationPattern, NodeId, PNCounter, PrefetchConfig, PrefetchItem, PrefetchPrediction, QueueMessage, QueueOptions, ReplicatedState, ReplicationConfig, RoutingConfig, RoutingDecision, SSEBroadcaster, SSEEvent, SSEStream, SmartCache, StateUpdate, StreamConfig, StreamingResponse, TimePattern, Timestamp, VectorClock, cached, createChunkedStream, createCloudflareKV, createEdgePrefetcher, createGeoRouter, createProgressiveHTML, createReplicatedState, createSmartCache, createStreamingHTML, createSyncHandler, generatePrefetchHints, generatePreloadHeaders, getClientLocation, pipeStreams, useEdgeKV
- Re-exported names: (none detected)
- Re-exported modules: ./edge-functions.js, ./geo-routing.js, ./prefetch.js, ./rate-limiter.js, ./smart-cache.js, ./state-replication.js, ./streaming.js

License

MIT

@philjs/edge-ai - Type: Node package - Purpose: On-device ML inference for PhilJS - WebGPU/WebNN accelerated, ONNX/TFLite support - Version: 0.1.0 - Location: packages/philjs-edge-ai - Entry points: packages/philjs-edge-ai/src/index.ts - Keywords: philjs, edge-ai, ml, webgpu, webnn, onnx, tensorflow, inference

@philjs/edge-ai

On-device ML inference for PhilJS - WebGPU/WebNN accelerated, ONNX/TFLite support

Overview

On-device ML inference for PhilJS - WebGPU/WebNN accelerated, ONNX/TFLite support

Focus Areas

- philjs, edge-ai, ml, webgpu, webnn, onnx, tensorflow, inference

Entry Points

- packages/philjs-edge-ai/src/index.ts

Quick Start

```
import { DeviceCapabilities, DeviceDetector, ImageClassifier } from '@philjs/edge-ai';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- DeviceCapabilities
- DeviceDetector

- ImageClassifier
- InferenceCallback
- InferenceEngine
- InferenceOptions
- InferenceResult
- ModelCache
- ModelConfig
- ModelLoader
- ModelMetadata
- ObjectDetector

Install

```
pnpm add @philjs/edge-ai
```

Usage

```
import { DeviceCapabilities, DeviceDetector, ImageClassifier } from '@philjs/edge-ai';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-edge-ai/src/index.ts

Public API

- Direct exports: DeviceCapabilities, DeviceDetector, ImageClassifier, InferenceCallback, InferenceEngine, InferenceOptions, InferenceResult, ModelCache, ModelConfig, ModelLoader, ModelMetadata, ObjectDetector, ProgressCallback, SpeechRecognizer, StreamingResult, Tensor, TextEmbedder, useDeviceCapabilities, useEdgeAI, useImageClassifier, useObjectDetector, useTextEmbedder
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/edge-mesh - Type: Node package - Purpose: Distributed edge consensus for PhilJS - P2P mesh networking, Raft consensus, gossip protocol - Version: 0.1.0 - Location: packages/philjs-edge-mesh - Entry points: packages/philjs-edge-mesh/src/index.ts - Keywords: philjs, edge, mesh, p2p, consensus, raft, gossip, distributed

@philjs/edge-mesh

Distributed edge consensus for PhilJS - P2P mesh networking, Raft consensus, gossip protocol

Overview

Distributed edge consensus for PhilJS - P2P mesh networking, Raft consensus, gossip protocol

Focus Areas

- philjs, edge, mesh, p2p, consensus, raft, gossip, distributed

Entry Points

- packages/philjs-edge-mesh/src/index.ts

Quick Start

```
import { AppendEntriesRequest, AppendEntriesResponse, ConsensusState } from '@philjs/edge-mesh';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- AppendEntriesRequest
- AppendEntriesResponse
- ConsensusState
- EdgeMesh
- GossipMessage
- GossipProtocol
- LogEntry
- MeshConfig
- MeshNode

- NodeRole
- RaftConsensus
- VectorClock

Install

```
pnpm add @philjs/edge-mesh
```

Usage

```
import { AppendEntriesRequest, AppendEntriesResponse, ConsensusState } from '@philjs/edge-mesh';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-edge-mesh/src/index.ts

Public API

- Direct exports: AppendEntriesRequest, AppendEntriesResponse, ConsensusState, EdgeMesh, GossipMessage, GossipProtocol, LogEntry, MeshConfig, MeshNode, NodeRole, RaftConsensus, VectorClock, VoteRequest, VoteResponse, useEdgeMesh, useGossipState
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/editor - Type: Node package - Purpose: Rich text and collaborative editing for PhilJS - ProseMirror-based, no React - Version: 0.1.0 - Location: packages/philjs-editor - Entry points: packages/philjs-editor/src/index.ts, packages/philjs-editor/src/extensions/index.ts, ./utils - Keywords: philjs, editor, rich-text, prosemirror, collaborative

@philjs/editor

Rich text and code editor components for React applications. Provides a flexible, extensible editor with support for markdown, code highlighting, and collaborative editing.

Installation

```
npm install @philjs/editor
# or
yarn add @philjs/editor
# or
pnpm add @philjs/editor
```

Basic Usage

```
import { RichTextEditor, CodeEditor } from '@philjs/editor';

function App() {
  const [content, setContent] = useState('');

  return (
    <div>
      <RichTextEditor
        value={content}
        onChange={setContent}
        placeholder="Start writing..." />

      <CodeEditor
        language="typescript"
        value={code}
        onChange={setCode} />
    </div>
  );
}
```

Features

- **Rich Text Editing** - Full WYSIWYG editor with formatting toolbar
- **Code Editor** - Syntax highlighting for 100+ languages
- **Markdown Support** - Write in markdown with live preview
- **Slash Commands** - Quick actions via / commands
- **Mentions** - @mention users and entities

- **Embeds** - Embed images, videos, and links
- **Tables** - Insert and edit tables
- **Code Blocks** - Syntax-highlighted code blocks
- **Collaboration** - Real-time collaborative editing
- **History** - Undo/redo with full history
- **Extensions** - Plugin system for custom functionality
- **Theming** - Light/dark mode and custom themes
- **Mobile Support** - Touch-friendly editing experience

Components

Component	Description
RichTextEditor	Full-featured rich text editor
CodeEditor	Code editor with syntax highlighting
MarkdownEditor	Markdown editor with preview
InlineEditor	Minimal inline text editor
EditorToolbar	Customizable formatting toolbar

Extensions

```
import { RichTextEditor, BoldExtension, LinkExtension } from '@philjs/editor';

<RichTextEditor
  extensions={[BoldExtension, LinkExtension, CustomExtension]}
/>
```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./extensions`, `./utils`
- Source files: `packages/philjs-editor/src/index.ts`, `packages/philjs-editor/src/extensions/index.ts`

Public API

- Direct exports: `EditorConfig`, `EditorInstance`, `PhilEditor`, `createEditorConfig`, `getCharacterCount`, `getWordCount`, `sanitizeContent`
- Re-exported names: `BlockMath`, `CodeBlock`, `CodeBlockOptions`, `CustomVideo`, `Emoji`, `EmojiOptions`, `Image`, `ImageExtensionOptions`, `ImageUpload`, `ImageUploadOptions`, `InlineMath`, `Link`, `LinkOptions`, `LinkPreviewData`, `MathOptions`, `Table`, `TableCell`, `TableHeader`, `TableOptions`, `TableRow`, `TaskItem`, `TaskList`, `TaskListOptions`, `VideoOptions`, `Vimeo`, `Youtube`, `bulletListToTaskList`, `checkAllTasks`, `codeBlockShortcuts`, `commonEmojis`, `createCodeBlockExtension`, `createImageExtension`, `createLinkExtension`, `createLinkPreviewPlugin`, `createMathExtensions`, `createTableExtensions`, `createTaskListExtensions`, `createVideoExtensions`, `detectVideoPlatform`, `emojiCategories`, `emojiPickerStyles`, `emojiTo_shortcode`, `getAllTasks`, `getDomain`, `getEmoji`, `getLinkAtSelection`, `getSupportedLanguages`, `getTableInfo`, `getTaskStats`, `insertEmoji`, `insertImageByUrl`, `insertVideo`, `isExternalUrl`, `isInTable`, `isValidUrl`, `linkCommands`, `linkShortcuts`, `linkStyles`, `lowlight`, `mathShortcuts`, `mathStyles`, `mathSymbols`, `mathTemplates`, `normalizeUrl`, `pickAndUploadImage`, `registerLanguage`, `renderLatex`, `replaceShortcodes`, `searchEmojis`, `tableCommands`, `tableShortcuts`, `taskListCommands`, `taskListShortcuts`, `taskListStyles`, `taskListToBulletList`, `toggleTaskAtPosition`, `uncheckAllTasks`, `validateLateX`
- Re-exported modules: `./code-block.js`, `./emoji.js`, `./image.js`, `./link.js`, `./math.js`, `./table.js`, `./task-list.js`, `./video.js`

License

MIT

```
## @philjs/email - Type: Node package - Purpose: Email client for PhilJS - multiple providers, templates, queue integration - Version: 0.1.0 - Location: packages/philjs-email - Entry points: packages/philjs-email/src/index.ts, ./providers/* - Keywords: philjs, email, nodemailer, sendgrid, resend, ses
```

@philjs/email

Email template components for building responsive, cross-client compatible emails with React. Write emails using familiar React patterns and render to HTML.

Installation

```
npm install @philjs/email
# or
yarn add @philjs/email
# or
pnpm add @philjs/email
```

Basic Usage

```

import { Email, Container, Heading, Text, Button, render } from '@philjs/email';

function WelcomeEmail({ name }) {
  return (
    <Email preview="Welcome to our platform!">
      <Container>
        <Heading>Welcome, {name}!</Heading>
        <Text>Thanks for signing up. Get started by clicking below.</Text>
        <Button href="https://example.com/dashboard">
          Go to Dashboard
        </Button>
      </Container>
    </Email>
  );
}

// Render to HTML string
const html = render(<WelcomeEmail name="John" />);

```

Features

- **React Components** - Build emails with familiar React patterns
- **Cross-Client Compatible** - Works in Gmail, Outlook, Apple Mail, and more
- **Responsive Design** - Mobile-friendly email layouts
- **Preview Text** - Control inbox preview snippets
- **Inline Styles** - Automatic CSS inlining for compatibility
- **Dark Mode** - Support for dark mode email clients
- **Components Library** - Pre-built components for common patterns
- **Template System** - Reusable email templates
- **Testing** - Preview and test emails locally
- **TypeScript** - Full type safety for email props

Components

Component	Description
Email	Root email container
Container	Centered content wrapper
Section	Content section
Row / Column	Layout grid
Heading	Email headings (h1-h6)
Text	Paragraph text
Button	Call-to-action buttons
Image	Responsive images
Link	Styled links
Divider	Horizontal divider

Sending Emails

```

import { render } from '@philjs/email';
import nodemailer from 'nodemailer';

const html = render(<WelcomeEmail name="John" />);
await transporter.sendMail({ to, subject, html });

```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./providers/*`
- Source files: `packages/philjs-email/src/index.ts`

Public API

- Direct exports: (none detected)
- Re-exported names: `BulkEmailMessage`, `BulkEmailResult`, `BulkRecipient`, `EmailAddress`, `EmailAttachment`, `EmailClientOptions`, `EmailMessage`, `EmailProvider`, `EmailQueue`, `EmailQueueJob`, `EmailResult`, `EmailTemplate`, `EmailTracking`, `GenericTrackingWebhook`, `InMemoryQueue`, `MailgunConfig`, `MailgunProvider`, `MailgunWebhook`, `ProviderConfig`, `QueueOptions`, `QueueStats`, `ResendConfig`, `ResendProvider`, `RetryConfig`, `SendGridConfig`, `SendGridProvider`, `SendGridWebhook`, `SesConfig`, `SesProvider`, `SesWebhook`, `SmtpConfig`, `SmtpPresets`, `SmtpProvider`, `TemplateEmailMessage`, `TemplateProps`, `TemplateRegistry`, `TrackingEvent`, `TrackingEventHandler`, `TrackingWebhook`, `UnsubscribeOptions`, `chunk`, `createMailgunProvider`, `createQueue`, `createResendProvider`, `createSendGridProvider`, `createSesProvider`, `createSmtpProvider`, `createTrackingPixel`, `createTrackingWebhook`, `defaultRetryConfig`, `formatAddress`, `generateId`, `htmlToText`, `isValidEmail`, `normalizeAddress`, `parseEmails`, `renderReactEmail`, `sleep`, `withRetry`, `wrapLinksForTracking`
- Re-exported modules: `./providers/index.js`, `./queue.js`, `./templates/index.js`, `./tracking.js`, `./types.js`, `./utils.js`

License

MIT

@philjs/enterprise - Type: Node package - Purpose: Enterprise features for PhilJS - Multi-tenancy, SSO, RBAC, Audit Logging, Feature Flags - Version: 0.1.0 - Location: packages/philjs-enterprise - Entry points: packages/philjs-enterprise/src/index.ts, packages/philjs-enterprise/src/multi-tenancy.ts, packages/philjs-enterprise/src/sso.ts, packages/philjs-enterprise/src/rbac.ts, packages/philjs-enterprise/src/audit.ts, packages/philjs-enterprise/src/feature-flags.ts, packages/philjs-enterprise/src/white-label.ts - Keywords: philjs, enterprise, multi-tenancy, sso, saml, oidc, ldap, rbac, audit, feature-flags, white-label

@philjs/enterprise

Enterprise features for PhilJS applications including SSO, audit logging, RBAC, and compliance tools. Built for organizations with advanced security and governance requirements.

Installation

```
npm install @philjs/enterprise
# or
yarn add @philjs/enterprise
# or
pnpm add @philjs/enterprise
```

Basic Usage

```
import {
  EnterpriseProvider,
  SSOProvider,
  AuditLogger,
  usePermissions
} from '@philjs/enterprise';

function App() {
  return (
    <EnterpriseProvider
      sso={{ provider: 'okta', domain: 'company.okta.com' }}
      audit={{ enabled: true }}
    >
      <SSOProvider>
        <Dashboard />
      </SSOProvider>
    </EnterpriseProvider>
  );
}

function Dashboard() {
  const { can } = usePermissions();

  return (
    <div>
      {can('view:reports') && <ReportsPanel />}
      {can('manage:users') && <UserManagement />}
    </div>
  );
}
```

Features

- **Single Sign-On (SSO)** - SAML 2.0 and OIDC integration
- **Identity Providers** - Okta, Azure AD, Google Workspace, OneLogin
- **Audit Logging** - Comprehensive activity tracking
- **RBAC** - Role-based access control with fine-grained permissions
- **ABAC** - Attribute-based access control policies
- **Session Management** - Secure session handling with timeout controls
- **MFA Support** - Multi-factor authentication integration
- **Compliance** - SOC 2, HIPAA, GDPR compliance helpers
- **Data Encryption** - At-rest and in-transit encryption utilities
- **IP Allowlisting** - Restrict access by IP address
- **Tenant Isolation** - Multi-tenant data separation
- **License Management** - Seat-based licensing controls

SSO Providers

Provider	Protocol
Okta	SAML 2.0, OIDC
Azure AD	SAML 2.0, OIDC
Google Workspace	OIDC
OneLogin	SAML 2.0
Auth0	OIDC
Custom	SAML 2.0, OIDC

Audit Logging

```

import { auditLog } from '@philjs/enterprise';

auditLog.record({
  action: 'user.updated',
  actor: userId,
  resource: targetUserId,
  metadata: { changes: ['email', 'role'] },
});

```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./multi-tenancy`, `./sso`, `./rbac`, `./audit`, `./feature-flags`, `./white-label`
- Source files: `packages/philjs-enterprise/src/index.ts`, `packages/philjs-enterprise/src/multi-tenancy.ts`, `packages/philjs-enterprise/src/sso.ts`, `packages/philjs-enterprise/src/rbac.ts`, `packages/philjs-enterprise/src/audit.ts`, `packages/philjs-enterprise/src/feature-flags.ts`, `packages/philjs-enterprise/src/white-label.ts`

Public API

- Direct exports: `AttributeMapping`, `AuditConfig`, `AuditEvent`, `AuditFilter`, `AuditLogger`, `AuditMetadata`, `AuditStorage`, `BrandingConfig`, `CustomizationConfig`, `DomainConfig`, `EmailTemplate`, `EvaluationContext`, `FeatureFlag`, `FeatureFlagManager`, `FeatureRule`, `FeatureVariant`, `LDAPConfig`, `MultiTenancyConfig`, `OAuth2Config`, `OIDCConfig`, `Permission`, `PermissionCondition`, `ProvisioningConfig`, `RBACConfig`, `RBACManager`, `Role`, `RuleCondition`, `SAMLConfig`, `SSOConfig`, `SSOManager`, `SSOProvider`, `SSOSession`, `SSouser`, `Tenant`, `TenantBranding`, `TenantContext`, `TenantManager`, `TenantSettings`, `TenantUser`, `WhiteLabelConfig`, `WhiteLabelManager`, `createAuditLogger`, `createDefaultBranding`, `createDefaultTenantSettings`, `createFeatureFlagManager`, `createInMemoryStorage`, `createLDAPConfig`, `createOIDCConfig`, `createRBACManager`, `createSAMLConfig`, `createSSOManager`, `createTenantManager`, `createTenantMiddleware`, `createWhiteLabelManager`, `tenantKey`, `tenantScope`, `withTenantId`
- Re-exported names: (none detected)
- Re-exported modules: `./audit.js`, `./compliance.js`, `./feature-flags.js`, `./multi-tenancy.js`, `./rbac.js`, `./sso.js`, `./white-label.js`

License

MIT

`## @philjs/errors - Type: Node package - Purpose: Error tracking and monitoring for PhilJS - Sentry, LogRocket, Rollbar integrations - Version: 0.1.0 - Location: packages/philjs-errors - Entry points: packages/philjs-errors/src/index.ts, packages/philjs-errors/src/sentry.ts, packages/philjs-errors/src/logrocket.ts, packages/philjs-errors/src/rollbar.ts - Keywords: philjs, error-tracking, sentry, logrocket, rollbar, monitoring`

philjs-errors

Error tracking and monitoring for PhilJS - Sentry, LogRocket, and Rollbar integrations.

Features

- **Multiple Providers** - Sentry, LogRocket, Rollbar support
- **Automatic Error Capture** - Catch runtime and async errors
- **Source Maps** - Upload source maps for better stack traces
- **User Context** - Track user sessions and actions
- **Custom Events** - Log custom errors and events
- **Performance Monitoring** - Track performance metrics
- **Release Tracking** - Associate errors with releases

Installation

```

# Sentry
pnpm add philjs-errors @sentry/browser

# LogRocket
pnpm add philjs-errors logrocket

# Rollbar
pnpm add philjs-errors rollbar

```

Quick Start

Sentry

```

import { initSentry } from 'philjs-errors/sentry';

initSentry({
  dsn: process.env.SENTRY_DSN,
  environment: process.env.NODE_ENV,
  release: process.env.RELEASE_VERSION
});

```

LogRocket

```

import { initLogRocket } from 'philjs-errors/logrocket';

initLogRocket({
  appId: process.env.LOGROCKET_APP_ID
});

```

Rollbar

```
import { initRollbar } from 'philjs-errors/rollbar';

initRollbar({
  accessToken: process.env.ROLLBAR_TOKEN,
  environment: process.env.NODE_ENV
});
```

Documentation

For more information, see the PhilJS documentation.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./sentry`, `/logrocket`, `/rollbar`
- Source files: `packages/philjs-errors/src/index.ts`, `packages/philjs-errors/src/sentry.ts`, `packages/philjs-errors/src/logrocket.ts`, `packages/philjs-errors/src/rollbar.ts`

Public API

- Direct exports: `Breadcrumb`, `ErrorHandler`, `ErrorEvent`, `ErrorTracker`, `LogRocketOptions`, `RollbarOptions`, `SentryOptions`, `Span`, `TrackerOptions`, `UserContext`, `addBreadcrumb`, `captureError`, `captureMessage`, `createErrorBoundary`, `createLogRocketTracker`, `createRollbarTracker`, `createSentryTracker`, `getErrorTracker`, `getSessionURL`, `initErrorTracking`, `redact`, `setUser`, `showSentryFeedback`, `startSpan`, `trackComponentWithSentry`, `trackEvent`, `trackRouteWithSentry`, `trackSignalErrors`, `trackSignalWithSentry`, `withErrorTracking`
- Re-exported names: `createLogRocketTracker`, `createRollbarTracker`, `createSentryTracker`
- Re-exported modules: `./logrocket.js`, `./rollbar.js`, `./sentry.js`

License

MIT

@philjs/event-sourcing - Type: Node package - Purpose: Event sourcing with CQRS for PhilJS - event store, aggregates, projections, sagas, time-travel - Version: 0.1.0 - Location: packages/philjs-event-sourcing - Entry points: packages/philjs-event-sourcing/src/index.ts - Keywords: philjs, event-sourcing, cqrs, ddd, saga, aggregate, projection

@philjs/event-sourcing

Event sourcing with CQRS for PhilJS - event store, aggregates, projections, sagas, time-travel

Overview

Event sourcing with CQRS for PhilJS - event store, aggregates, projections, sagas, time-travel

Focus Areas

- philjs, event-sourcing, cqrs, ddd, saga, aggregate, projection

Entry Points

- `packages/philjs-event-sourcing/src/index.ts`

Quick Start

```
import { Command, CommandBus, CommandHandler } from '@philjs/event-sourcing';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- `Command`
- `CommandBus`
- `CommandHandler`
- `CommandMetadata`
- `Event`
- `EventHandler`
- `EventMetadata`
- `EventStore`
- `Projection`
- `ProjectionHandler`
- `ReadModel`
- `Repository`

Install

```
pnpm add @philjs/event-sourcing
```

Usage

```
import { Command, CommandBus, CommandHandler } from '@philjs/event-sourcing';
```

Scripts

- `pnpm run build`
- `pnpm run test`

Compatibility

- Node >=24

- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-event-sourcing/src/index.ts

Public API

- Direct exports: Command, CommandBus, CommandHandler, CommandMetadata, Event, EventHandler, EventMetadata, EventStore, Projection, ProjectionHandler, ReadModel, Repository, SagaDefinition, SagaManager, SagaState, SagaStep, Snapshot, TimeTravelDebugger, createCommand, createEvent, useAggregate, useEventStore, useReadModel, useTimeTravel
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/export - Type: Node package - Purpose: Data export utilities for PhiJS - CSV, Excel, JSON, PDF - Version: 0.1.0 - Location: packages/philjs-export - Entry points: packages/philjs-export/src/index.ts, packages/philjs-export/src/formats/index.ts, packages/philjs-export/src/utils/index.ts - Keywords: philjs, export, csv, excel, json, pdf, download

@philjs/export

Export utilities for converting React components and data to various formats including PDF, CSV, Excel, and images. Seamlessly generate downloadable files from your application.

Installation

```
npm install @philjs/export
# or
yarn add @philjs/export
# or
pnpm add @philjs/export
```

Basic Usage

```
import { exportToPDF, exportToCSV, exportToExcel } from '@philjs/export';

// Export component to PDF
await exportToPDF(<Invoice data={invoiceData} />, {
  filename: 'invoice.pdf',
});

// Export data to CSV
await exportToCSV(tableData, {
  filename: 'report.csv',
  columns: ['name', 'email', 'status'],
});

// Export data to Excel
await exportToExcel(data, {
  filename: 'report.xlsx',
  sheets: [{ name: 'Users', data: users }],
});
```

Features

- **PDF Export** - Render React components to PDF documents
- **CSV Export** - Generate CSV files from data arrays
- **Excel Export** - Create XLSX files with multiple sheets
- **Image Export** - Export components as PNG, JPEG, or SVG
- **Print Support** - Optimized print layouts
- **Custom Templates** - Define reusable export templates
- **Streaming** - Stream large exports to avoid memory issues
- **Progress Tracking** - Monitor export progress for large files
- **Styling** - Preserve styles in exported documents
- **Headers/Footers** - Add page headers and footers to PDFs
- **Watermarks** - Add watermarks to exported documents
- **Compression** - Optimize file size for exports

Export Functions

Function	Output Format
exportToPDF	PDF document
exportToCSV	CSV file
exportToExcel	XLSX spreadsheet
exportToPNG	PNG image
exportToJPEG	JPEG image
exportToSVG	SVG vector
exportToJson	JSON file

React Hook

```
import { useExport } from '@philjs/export';

function ReportPage() {
  const { exportRef, exportToPDF, isExporting } = useExport();

  return (
    <div>
      <div ref={exportRef}>
        <Report data={data} />
      </div>
      <button onClick={() => exportToPDF()} disabled={isExporting}>
        {isExporting ? 'Exporting...' : 'Download PDF'}
      </button>
    </div>
  );
}
```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: ., ./formats, ./utils
- Source files: packages/philjs-export/src/index.ts, packages/philjs-export/src/formats/index.ts, packages/philjs-export/src/utils/index.ts

Public API

- Direct exports: ExportOptions, downloadFile, exportToCSV, exportToExcel, exportToJson, exportToPDF, exportToXML, exportToYAML
- Re-exported names: (none detected)
- Re-exported modules: ./components/index.js, ./csv.js, ./excel.js, ./formats/index.js, ./formatters.js, ./hooks.js, ./json.js, ./streaming.js, ./utils/index.js, ./xml.js, ./yaml.js, ./zip.js

License

MIT

@philjs/eye-tracking - Type: Node package - Purpose: Gaze-based interactions for PhilJS - eye tracking, dwell click, attention heatmaps - Version: 0.1.0 - Location: packages/philjs-eye-tracking - Entry points: packages/philjs-eye-tracking/src/index.ts - Keywords: philjs, eye-tracking, gaze, dwell-click, accessibility, heatmap

@philjs/eye-tracking

Gaze-based interactions for PhilJS - eye tracking, dwell click, attention heatmaps

Overview

Gaze-based interactions for PhilJS - eye tracking, dwell click, attention heatmaps

Focus Areas

- philjs, eye-tracking, gaze, dwell-click, accessibility, heatmap

Entry Points

- packages/philjs-eye-tracking/src/index.ts

Quick Start

```
import { AttentionHeatmap, CalibrationResult, DwellClick } from '@philjs/eye-tracking';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- AttentionHeatmap
- CalibrationResult
- DwellClick
- EyeTracker
- EyeTrackingConfig
- Fixation
- GazeAwareElement
- GazeCallback
- GazeCursor
- GazeEvent

- GazeEventCallback
- GazePoint

Install

```
pnpm add @philjs/eye-tracking
```

Usage

```
import { AttentionHeatmap, CalibrationResult, DwellClick } from '@philjs/eye-tracking';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-eye-tracking/src/index.ts

Public API

- Direct exports: AttentionHeatmap, CalibrationResult, DwellClick, EyeTracker, EyeTrackingConfig, Fixation, GazeAwareElement, GazeCallback, GazeCursor, GazeEvent, GazeEventCallback, GazePoint, HeatmapConfig, ReadingAnalyzer, Saccade, useAttentionHeatmap, useDwellClick, useEyeTracking, useGazeAware, useGazePoint, useReadingAnalysis
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/forms - Type: Node package - Purpose: Form handling with validation for PhilJS - Version: 0.1.0 - Location: packages/philjs-forms - Entry points: packages/philjs-forms/src/index.ts, packages/philjs-forms/src/validation.ts, packages/philjs-forms/src/fields.ts - Keywords: philjs, forms, validation, zod

philjs-forms

Advanced form handling for PhilJS with Remix-style primitives, progressive enhancement, and optimistic UI.

Requirements

- **Node.js 24** or higher
- **TypeScript 6** or higher
- **ESM only** - CommonJS is not supported

Features

- **Form Management** - Reactive form state with signals
- **Validation** - Built-in validators + Zod integration
- **useFormAction** - Remix-style form actions with progressive enhancement
- **useFetcher** - Non-navigational form submissions
- **Optimistic UI** - Instant updates with automatic rollback
- **Progressive Enhancement** - Works without JavaScript
- **Field Components** - Pre-built accessible form fields
- **TypeScript** - Full type safety

Installation

```
pnpm add philjs-forms
```

```
# Optional: for Zod validation
```

```
pnpm add zod
```

Quick Start

Basic Form

```
import { useForm, validators, TextField } from 'philjs-forms';

function ContactForm() {
  const form = useForm({
    initialValues: {
      name: '',
      email: '',
      message: ''
    },
    onSubmit: async (values) => {
      console.log('Submitted:', values);
      await fetch('/api/contact', {
        method: 'POST',
        body: JSON.stringify(values)
      });
    }
  });

  return (
    <form onsubmit={form.handleSubmit}>
      <TextField
        {...form.getFieldProps('name')}
        placeholder="Your name"
      />

      <TextField
        {...form.getFieldProps('email')}
        type="email"
        placeholder="Your email"
      />

      <textarea
        name="message"
        value={form.values().message}
        oninput={(e) => form.setFieldValue('message', e.target.value)}
        onBlur={() => form.setFieldTouched('message')}
      />

      <button type="submit" disabled={form.isSubmitting()}>
        {form.isSubmitting() ? 'Sending...' : 'Send'}
      </button>
    </form>
  );
}

With Validation
```

```

import { useForm, validators } from 'philjs-forms';

class ContactForm extends Form {
  async validateField(name) {
    const value = this.values()[name];

    if (name === 'email') {
      const rules = [
        validators.required(),
        validators.email()
      ];
      return await validateValue(value, rules);
    }

    if (name === 'name') {
      const rules = [
        validators.required(),
        validators.minLength(2)
      ];
      return await validateValue(value, rules);
    }

    return null;
  }
}

function MyForm() {
  const form = new ContactForm({
    initialValues: { name: '', email: '' },
    validateOn: 'blur'
  });

  return (
    <form onsubmit={form.handleSubmit.bind(form)}>
      <Field name="name" label="Name" required>
        <TextField {...form.getFieldProps('name')} />
      </Field>

      <Field name="email" label="Email" required>
        <TextField {...form.getFieldProps('email')} type="email" />
      </Field>

      <button type="submit">Submit</button>
    </form>
  );
}

```

With Zod Validation

```

import { useForm, createZodValidator } from 'philjs-forms';
import { z } from 'zod';

const schema = z.object({
  name: z.string().min(2, 'Name must be at least 2 characters'),
  email: z.string().email('Invalid email'),
  age: z.number().min(18, 'Must be 18 or older'),
  password: z.string().min(8, 'Password must be at least 8 characters'),
  confirmPassword: z.string()
}).refine((data) => data.password === data.confirmPassword, {
  message: "Passwords don't match",
  path: ['confirmPassword']
});

class SignupForm extends Form {
  private validator = createZodValidator(schema);

  async validateField(name) {
    const value = this.values()[name];
    return await this.validator.validateField(name, value);
  }

  async validate() {
    const values = this.values();
    const errors = await this.validator.validate(values);
    this.setErrors(errors);
    return errors;
  }
}

function SignupPage() {
  const form = new SignupForm({
    initialValues: {
      name: '',
      email: '',
      age: 0,
      password: '',
      confirmPassword: ''
    },
    validateOn: 'blur',
    onSubmit: async (values) => {
      await createAccount(values);
    }
  });

  return (
    <form onsubmit={form.handleSubmit.bind(form)}>
      <TextField {...form.getFieldProps('name')} placeholder="Name" />
      <TextField {...form.getFieldProps('email')} type="email" placeholder="Email" />
      <NumberField {...form.getFieldProps('age')} placeholder="Age" />
      <TextField {...form.getFieldProps('password')} type="password" placeholder="Password" />
      <TextField {...form.getFieldProps('confirmPassword')} type="password" placeholder="Confirm" />
      <button type="submit">Sign Up</button>
    </form>
  );
}

```

API Reference

Form Class

```

class Form<T extends FormValues> {
  // Reactive signals
  values: Signal<T>;
  errors: Signal<FormErrors<T>>;
  touched: Signal<TouchedFields<T>>;
  isValid: ComputedSignal<boolean>;
  isDirty: ComputedSignal<boolean>;
  isSubmitting: Signal<boolean>;
  isValidating: Signal<boolean>;
  submitCount: Signal<number>;

  // Methods
  setFieldValue<K extends keyof T>(name: K, value: T[K]): void;
  setValues(values: Partial<T>): void;
  setFieldError<K extends keyof T>(name: K, error: string | null): void;
  setErrors(errors: FormErrors<T>): void;
  setFieldTouched<K extends keyof T>(name: K, touched?: boolean): void;
  reset(): void;
  resetWith(values: Partial<T>): void;
  validate(): Promise<FormErrors<T>>;
  validateField<K extends keyof T>(name: K): Promise<string | null>;
  handleSubmit(e: Event): Promise<void>;
  getFieldProps<K extends keyof T>(name: K): FieldProps;
}

```

Built-in Validators

```

validators.required(message?: string)
validators.email(message?: string)
validators.minLength(min: number, message?: string)
validators.maxLength(max: number, message?: string)
validators.min(min: number, message?: string)
validators.max(max: number, message?: string)
validators.pattern(regex: RegExp, message?: string)
validators.url(message?: string)
validators.matches(field: string, message?: string)
validators.oneOf(options: any[], message?: string)
validators.custom(fn: Function, message: string)

```

Field Components

All field components support: - name: Field name - value: Computed signal with current value - error: Error message - touched: Whether field has been touched - onChange: Change handler - onBlur: Blur handler - disabled: Disabled state - class: Custom CSS class - errorClass: CSS class for error state

TextField

```

<TextField
  name="username"
  value={value}
  error={error}
  touched={touched}
  type="text" // or "email", "password", "tel", "url", "search"
  placeholder="Enter username"
  onChange={handleChange}
  onBlur={handleBlur}
/>

```

TextAreaField

```

<TextAreaField
  name="message"
  value={value}
  rows={5}
  cols={50}
  placeholder="Your message"
  onChange={handleChange}
/>

```

SelectField

```

<SelectField
  name="country"
  value={value}
  options={[
    { value: 'us', label: 'United States' },
    { value: 'ca', label: 'Canada' }
  ]}
  placeholder="Select country"
  onChange={handleChange}
/>

```

CheckboxField

```

<CheckboxField
  name="agree"
  value={value}
  label="I agree to terms"
  onChange={handleChange}
/>

```

RadioField

```

<RadioField
  name="plan"
  value={value}
  options={[
    { value: 'free', label: 'Free Plan' },
    { value: 'pro', label: 'Pro Plan' }
  ]}
  onChange={handleChange}
/>

```

NumberField

```
<NumberField
  name="age"
  value={value}
  min={0}
  max={120}
  step={1}
  onChange={handleChange}
/>
```

FileField

```
<FileField
  name="avatar"
  value={value}
  accept="image/*"
  multiple={false}
  onChange={handleChange}
/>
```

Advanced Usage

Custom Validation

```
class MyForm extends Form {
  async validateField(name) {
    const value = this.values()[name];

    if (name === 'username') {
      // Check if username is available
      const available = await checkUsername(value);
      if (!available) {
        return 'Username is taken';
      }
    }

    return null;
  }
}
```

Debounced Validation

```
import { debounceValidation } from 'philjs-forms';

class MyForm extends Form {
  validateField = debounceValidation(async (name) => {
    // Validation logic
  }, 500);
}
```

Conditional Validation

```
class MyForm extends Form {
  async validateField(name) {
    const values = this.values();

    if (name === 'zipCode' && values.country === 'US') {
      return await validateValue(
        values.zipCode,
        validators.pattern(patterns.zipCode, 'Invalid ZIP code')
      );
    }

    return null;
  }
}
```

Multiple Validators

```
import { validateValue } from 'philjs-forms';

const rules = [
  validators.required('Password is required'),
  validators.minLength(8, 'Must be at least 8 characters'),
  validators.pattern(patterns.password, 'Must include uppercase, lowercase, number, and symbol')
];

const error = await validateValue(password, rules);
```

Form State Tracking

```

const form = useForm({ initialValues: { name: '' } });

effect(() => {
  const state = form.state();
  console.log('Form state:', {
    isValid: state.isValid,
    isDirty: state.isDirty,
    submitCount: state.submitCount
  });
});

```

Dynamic Fields

```

function DynamicForm() {
  const form = useForm({
    initialValues: {
      items: []
    }
  });

  const addItem = () => {
    const items = form.values().items;
    form.setFieldValue('items', [...items, '']);
  };

  return (
    <form onsubmit={form.handleSubmit}>
      {form.values().items.map((item, index) => (
        <TextField
          name={`item-${index}`}
          value={() => form.values().items[index]}
          onChange={(value) => {
            const items = [...form.values().items];
            items[index] = value;
            form.setFieldValue('items', items);
          }}
        />
      ))}
      <button type="button" onclick={addItem}>Add Item</button>
    </form>
  );
}

```

Complete Example

```

import {
  useForm,
  validators,
  validateValue,
  TextField,
  Selectfield,
  CheckboxField,
  Field
} from 'philjs-forms';

class RegistrationForm extends Form {
  async validateField(name) {
    const value = this.values()[name];
    const values = this.values();

    switch (name) {
      case 'email':
        return await validateValue(value, [
          validators.required(),
          validators.email()
        ]);

      case 'password':
        return await validateValue(value, [
          validators.required(),
          validators.minLength(8),
          validators.pattern(
            patterns.password,
            'Password must include uppercase, lowercase, number, and symbol'
          )
        ]);

      case 'confirmPassword':
        return await validateValue(value, [
          validators.required(),
          validators.matches('password', 'Passwords must match')
        ], values);

      case 'age':
        return await validateValue(value, [
          validators.required(),
          validators.min(18, 'Must be 18 or older')
        ]);
    }
  }
}

```

```

        ...
    }

    default:
        return null;
    }
}

function RegisterPage() {
    const form = new RegistrationForm({
        initialValues: {
            email: '',
            password: '',
            confirmPassword: '',
            age: 0,
            plan: 'free',
            agree: false
        },
        validateOn: 'blur',
        onSubmit: async (values) => {
            await fetch('/api/register', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(values)
            });
        }
    });

    return (
        <div className="register-page">
            <h1>Register</h1>

            <form onSubmit={form.handleSubmit.bind(form)}>
                <Field name="email" label="Email" required>
                    <TextField
                        {...form.getFieldProps('email')}
                        type="email"
                        placeholder="you@example.com"
                    />
                </Field>

                <Field name="password" label="Password" required hint="Min 8 characters">
                    <TextField
                        {...form.getFieldProps('password')}
                        type="password"
                    />
                </Field>

                <Field name="confirmPassword" label="Confirm Password" required>
                    <TextField
                        {...form.getFieldProps('confirmPassword')}
                        type="password"
                    />
                </Field>

                <Field name="age" label="Age" required>
                    <NumberField
                        {...form.getFieldProps('age')}
                        min={0}
                        max={120}
                    />
                </Field>

                <Field name="plan" label="Plan">
                    <SelectField
                        {...form.getFieldProps('plan')}
                        options={[
                            { value: 'free', label: 'Free' },
                            { value: 'pro', label: 'Pro ($9/mo)' },
                            { value: 'enterprise', label: 'Enterprise' }
                        ]}
                    />
                </Field>

                <CheckboxField
                    {...form.getFieldProps('agree')}
                    label="I agree to the terms and conditions"
                />

                <div className="form-actions">
                    <button
                        type="submit"
                        disabled={!(form.isValid() || form.isSubmitting())}
                    >
                        {form.isSubmitting() ? 'Registering...' : 'Register'}
                    </button>
                </div>
            </form>
        </div>
    );
}

```

```

    <div>
      <form>
        Please fix the errors above
      </form>
    </div>
  );
}

```

TypeScript

Full TypeScript support included:

```

import type {
  FormValues,
  FormErrors,
  FormState,
  ValidationRule,
  FieldConfig
} from 'philjs-forms';

interface MyFormValues {
  email: string;
  password: string;
  age: number;
}

const form = useForm<MyFormValues>({
  initialValues: {
    email: '',
    password: '',
    age: 0
  }
});

```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./validation`, `./fields`
- Source files: `packages/philjs-forms/src/index.ts`, `packages/philjs-forms/src/validation.ts`, `packages/philjs-forms/src/fields.ts`

Public API

- Direct exports: `BaseFieldProps`, `CheckboxField`, `CheckboxFieldProps`, `Field`, `FieldProps`, `FieldType`, `FileField`, `FileFieldProps`, `NumberField`, `NumberFieldProps`, `RadioField`, `RadioFieldProps`, `SelectField`, `SelectFieldProps`, `TextAreaField`, `TextAreaFieldProps`, `TextField`, `TextFieldProps`, `composeValidators`, `createZodValidator`, `debounceValidation`, `patterns`, `validateValue`, `validators`, `zodValidator`
- Re-exported names: // Advanced validators `advancedValidators`, // Schema validator `SchemaValidator`, // Types type `AsyncValidationRule`, // Utilities when, `AutoSaveConfig`, `AutoSaveController`, `AutoSaveState`, `CheckboxField`, `ClientOnly`, `ConditionalRule`, `ConflictStrategy`, `CrossFieldRule`, `FetcherMethod`, `FetcherReturn`, `FetcherState`, `FetcherSubmitOptions`, `Field`, `FieldConfig`, `FieldError`, `FieldState`, `FieldValidationResult`, `FieldValue`, `FileField`, `Form`, `FormActionOptions`, `FormActionReturn`, `FormActionState`, `FormConfig`, `FormDraft`, `FormErrors`, `FormState`, `FormValues`, `MaskChar`, `MaskConfig`, `MaskDefinition`, `MaskInputHandler`, `MaskResult`, `NoScript`, `NumberField`, `OptimisticOptions`, `OptimisticUpdate`, `ProgressiveFormOptions`, `RadioField`, `RecoveryDialogProps`, `SelectField`, `StepIndicatorProps`, `StepTransitionConfig`, `TextAreaField`, `TextField`, `TouchedFields`, `TransitionDirection`, `ValidationContext`, `ValidationGroup`, `ValidationResult`, `ValidationRule`, `ValidationSchema`, `WizardConfig`, `WizardController`, `WizardState`, `WizardStep`, `addJavaScriptMarker`, `applyMask`, `calculateProgress`, `clientHasJavaScript`, `combineResults`, `composeValidators`, `createAutoSave`, `createCheckoutWizard`, `createForm`, `createIndexedDBStorage`, `createMaskInputHandler`, `createSchemaValidator`, `createSessionStorage`, `createSignupWizard`, `createSurveyWizard`, `createWizard`, `createZodValidator`, `creditCardMask`, `crossField`, `currencyMask`, `dateMask`, `debounceValidation`, `dependsOn`, `detectCardType`, `formatDraftTimestamp`, `formatErrors`, `getFirstError`, `getRecoveryMessage`, `getStepIndicatorData`, `getStepTransitionStyles`, `hasErrors`, `isHydrated`, `isJSEnabled`, `isProgressivelyEnhanced`, `luhnCheck`, `maskChars`, `maskPresets`, `messageWithField`, `parseMaskPattern`, `patterns`, `phoneMask`, `resolveConflict`, `ssnMask`, `timeMask`, `unmask`, `useAutoSave`, `useFetcher`, `useForm`, `useFormAction`, `useOptimistic`, `useOptimisticValue`, `useProgressiveForm`, `useWizard`, `validateValue`, `validators`, `zipCodeMask`, `zodValidator`
- Re-exported modules: `./advanced-validation.js`, `./auto-save.js`, `./fields.js`, `./form.js`, `./input-mask.js`, `./optimistic.js`, `./progressive.js`, `./types.js`, `./useFetcher.js`, `./useFormAction.js`, `./validation.js`, `./wizard.js`

License

MIT

@philjs/genui - Type: Node package - Purpose: Runtime AI-driven UI composition with A2UI protocol for PhilJS - Version: 0.1.0 - Location: packages/philjs-genui - Entry points: packages/philjs-genui/src/index.ts, packages/philjs-genui/src/protocol/index.ts, packages/philjs-genui/src/registry/index.ts, packages/philjs-genui/src/runtime/index.ts, packages/philjs-genui/src/sandbox/index.ts, packages/philjs-genui/src/hooks.ts - Keywords: philjs, genui, a2ui, ai, generative-ui, llm

@philjs/genui

Runtime AI-driven UI composition with A2UI (Agent-to-UI) protocol for PhilJS.

Overview

`@philjs/genui` enables AI agents to dynamically generate and update user interfaces at runtime using a structured JSON protocol. It provides secure, sandboxed execution of AI-generated UI specifications with full type safety.

Features

- **A2UI Protocol:** Structured JSON schema for LLM-generated UI specifications
- **Component Registry:** Type-safe registration and querying of UI components
- **Security Sandbox:** AST validation to prevent injection attacks
- **Runtime Hydration:** Convert A2UI messages to live DOM elements
- **Reactive Bindings:** Connect UI to signals and data sources

- **Built-in Components:** Pre-registered safe components for common use cases

Installation

```
npm install @philjs/genui
# or
pnpm add @philjs/genui
```

Quick Start

```
import {
  createRegistry,
  registerBuiltins,
  createHydrator,
  createRenderMessage,
  useGenUI,
} from '@philjs/genui';

// Setup registry with built-in components
const registry = createRegistry();
registerBuiltins(registry);

// Create a hydrator
const hydrator = createHydrator(registry, {
  onAgentAction: (actionId, event) => {
    console.log('Action:', actionId, event);
  },
});

// Create a render message
const message = createRenderMessage(
  { type: 'stack', gap: '16px' },
  [
    { id: 'heading', type: 'Heading', props: { children: 'Welcome', level: 1 } },
    { id: 'text', type: 'Text', props: { children: 'Generated by AI' } },
    { id: 'btn', type: 'Button', props: { children: 'Click me', variant: 'primary' } },
  ]
);

// Hydrate to DOM
const container = document.getElementById('app')!;
const result = hydrator.hydrate(message, container);

if (result.success) {
  console.log('UI rendered successfully');
}
```

A2UI Protocol

The A2UI (Agent-to-UI) protocol defines how AI agents communicate UI specifications:

Message Types

```
// Render - Create a complete UI tree
const renderMessage = createRenderMessage(
  { type: 'flex', direction: 'row' },
  [{ id: 'comp-1', type: 'Button', props: { children: 'Submit' } }],
  {
    bindings: [...],
    actions: [...],
  }
);

// Update - Modify existing components
const updateMessage = createUpdateMessage('comp-1', {
  props: { disabled: true },
  animation: { type: 'fade', duration: 200 },
});

// Action - User interactions sent to agent
const actionMessage = createActionMessage(
  'btn-click',
  { type: 'click', data: { timestamp: Date.now() } }
);
```

Layout Types

```

// Stack Layout (vertical/horizontal)
{ type: 'stack', direction: 'column', gap: '16px', align: 'center' }

// Grid Layout
{ type: 'grid', columns: 'repeat(3, 1fr)', gap: '8px' }

// Flex Layout
{ type: 'flex', direction: 'row', justify: 'between', wrap: true }

// Flow Layout (block)
{ type: 'flow' }

// Absolute positioning
{ type: 'absolute' }

```

Component Definition

```

interface A2UIComponent {
  id: string;           // Unique identifier
  type: string;          // Component type from registry
  props: Record<string, unknown>;
  children?: A2UIComponent[];
  className?: string;
  style?: Record<string, string | number>;
  when?: { expression: string; fallback?: A2UIComponent };
  each?: { source: string; item: string; key: string };
  a11y?: { role?: string; label?: string; ... };
}

```

Component Registry

Register custom components for AI agents to use:

```

import { createRegistry, type ComponentCapability, type ComponentRenderer } from '@philjs/genui';

const registry = createRegistry();

// Define component capability (for LLM context)
const capability: ComponentCapability = {
  type: 'UserCard',
  displayName: 'User Card',
  description: 'Displays user information in a card format',
  category: 'display',
  props: [
    { name: 'name', type: 'string', required: true, description: 'User name' },
    { name: 'avatar', type: 'string', description: 'Avatar URL' },
    { name: 'role', type: 'string', enum: ['admin', 'user', 'guest'] },
  ],
  slots: [{ name: 'actions', description: 'Action buttons' }],
  events: [{ name: 'onSelect', description: 'Fired when card is selected' }],
  tags: ['user', 'profile', 'card'],
};

// Define renderer
const renderer: ComponentRenderer = (component, context) => {
  const card = document.createElement('div');
  card.className = 'user-card';
  card.innerHTML = `
    
    <h3>${component.props.name}</h3>
    <span>${component.props.role || 'user'}</span>
  `;
  return card;
};

registry.register(capability, renderer);

// Generate manifest for LLM
const manifest = registry.generateManifest();
// Send manifest to AI agent for context

```

Security Sandbox

The sandbox validates AI-generated UI to prevent injection attacks:

```

import { createValidator, DEFAULT_SANDBOX_CONFIG } from '@philjs/genui';

// Create validator with custom config
const validator = createValidator({
  allowedComponents: ['Button', 'Text', 'Input', 'Box'],
  maxDepth: 5,
  maxComponents: 50,
  allowInlineStyles: true,
  allowCustomEvents: false,
  blockedNavigationPatterns: [/javascript:/i, /data:/i],
});

// Validate a message
const result = validator.validate(message);

if (!result.valid) {
  console.error('Validation errors:', result.errors);
}

```

Default Security Rules

- **Forbidden patterns:** javascript:, data:, <script>, eval(), Function(), etc.
- **Blocked DOM access:** document.*, window.*, globalThis.*
- **Prototype pollution prevention:** __proto__, constructor[], prototype[]
- **Component limits:** Max depth (10), max components (100), max bindings (50)

Hooks

useGenUI

Hook for AI-generated UI with agent integration:

```

import { useGenUI, createMockAgent } from '@philjs/genui';

// Create mock agent for testing
const mockAgent = createMockAgent((prompt) => ({
  version: '1.0',
  type: 'render',
  payload: {
    type: 'render',
    layout: { type: 'stack' },
    components: [
      { id: 'response', type: 'Text', props: { children: `Response to: ${prompt}` } },
    ],
  },
}));


// Use the hook
const genui = useGenUI({
  agent: mockAgent,
  onAgentAction: (actionId, event) => {
    console.log('Agent action:', actionId, event);
  },
});

// Generate UI from prompt
await genui.generate('Create a login form');

// Render to container
const cleanup = genui.render(document.getElementById('app')!);

// Get component manifest
const manifest = genui.getManifest();

```

useAgentUI

Hook for real-time WebSocket-based agent communication:

```

import { useAgentUI } from '@philjs/genui';

const agent = useAgentUI({
  endpoint: 'wss://api.example.com/agent',
  autoReconnect: true,
  reconnectInterval: 5000,
  onConnect: () => console.log('Connected'),
  onMessage: (message) => console.log('Received:', message),
  onError: (error) => console.error('Error:', error),
});

// Connect to agent
await agent.connect();

// Send message
agent.send('Show me the user dashboard', { userId: '123' });

// Access state
console.log('Connected:', agent.connected);
console.log('Session:', agent.sessionId);
console.log('Current UI:', agent.ui);

// Disconnect
agent.disconnect();

```

Validation

Validate messages and components:

```

import { validateMessage, validateComponent, validateLayout, schemas } from '@philjs/genui';

// Validate complete message
const messageResult = validateMessage(message);
if (!messageResult.valid) {
  console.error(messageResult.errors);
}

// Validate single component
const componentResult = validateComponent({
  id: 'btn-1',
  type: 'Button',
  props: { children: 'Click' },
});

// Validate Layout
const layoutResult = validateLayout({
  type: 'grid',
  columns: '1fr 1fr',
});

// Access Zod schemas directly
const parsed = schemas.component.safeParse(componentData);

```

Built-in Components

The package includes pre-registered components:

Layout

- Box - Generic container
- Stack - Flex container (vertical/horizontal)
- Grid - CSS grid container

Display

- Text - Text content
- Heading - Headings (h1-h6)

Input

- Button - Interactive button
- Input - Text input field

Feedback

- Alert - Alert/notification
- Spinner - Loading indicator

TypeScript Support

Full type exports:

```

import type {
  // Protocol types
  A2UIMessage,
  A2UIComponent,
  A2UILayout,
  A2UIBinding,
  A2UIAction,
  A2UIAnimation,

  // Registry types
  ComponentCapability,
  ComponentRenderer,
  RenderContext,
  ComponentManifest,

  // Sandbox types
  SandboxConfig,
  ValidationError,

  // Runtime types
  HydrationResult,
  HydratorOptions,

  // Hook types
  GenUIState,
  GenUIAgent,
  AgentUIState,
} from '@philjs/genui';

```

Subpath Exports

```

// Protocol only
import { createRenderMessage, validateMessage } from '@philjs/genui/protocol';

// Registry only
import { ComponentRegistry, registerBuiltins } from '@philjs/genui/registry';

// Runtime only
import { GenUIHydrator, createHydrator } from '@philjs/genui/runtime';

// Sandbox only
import { ASTValidator, DEFAULT_SANDBOX_CONFIG } from '@philjs/genui/sandbox';

// Hooks only
import { useGenUI, useAgentUI } from '@philjs/genui/hooks';

```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: ., ./protocol, ./registry, ./runtime, ./sandbox, ./hooks
- Source files: packages/philjs-genui/src/index.ts, packages/philjs-genui/src/protocol/index.ts, packages/philjs-genui/src/registry/index.ts, packages/philjs-genui/src/runtime/index.ts, packages/philjs-genui/src/sandbox/index.ts, packages/philjs-genui/src/hooks.ts

Public API

- Direct exports: AgentUIOptions, AgentUIState, GenUIAgent, GenUIOptions, GenUIState, createLayoutGenerator, createMockAgent, useAgentUI, useGenUI
- Re-exported names: ASTValidator, ComponentRegistry, DEFAULT_SANDBOX_CONFIG, GenUIHydrator, builtinComponents, createActionMessage, createHydrator, createLayoutGenerator, createMockAgent, createRegistry, createRenderMessage, createUpdateMessage, createValidator, getDefaultRegistry, registerBuiltins, schemas, setDefaultRegistry, useAgentUI, useGenUI, validateComponent, validateLayout, validateMessage
- Re-exported modules: ./a2ui-schema.js, ./ast-validator.js, ./builtin-components.js, ./component-registry.js, ./hooks.js, ./hydrator.js, ./protocol/a2ui-schema.js, ./protocol/validator.js, ./registry/builtin-components.js, ./registry/component-registry.js, ./runtime/hydrator.js, ./sandbox/ast-validator.js, ./types.js, ./validator.js

License

MIT

@philjs/gesture - Type: Node package - Purpose: Camera-based gesture recognition for PhilJS - hand tracking, air cursor, touchless UI - Version: 0.1.0 - Location: packages/philjs-gesture - Entry points: packages/philjs-gesture/src/index.ts - Keywords: philjs, gesture, hand-tracking, mediapipe, touchless, air-cursor

@philjs/gesture

Camera-based gesture recognition for PhilJS - hand tracking, air cursor, touchless UI

Overview

Camera-based gesture recognition for PhilJS - hand tracking, air cursor, touchless UI

Focus Areas

- philjs, gesture, hand-tracking, mediapipe, touchless, air-cursor

Entry Points

- packages/philjs-gesture/src/index.ts

Quick Start

```
import { AirCursor, BoundingBox, FingerState } from '@philjs/gesture';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- AirCursor
- BoundingBox
- FingerState
- GestureCallback
- GestureConfig
- GestureController
- GestureDefinition
- GesturePresets
- GestureRecognizer
- GestureSequence
- Hand
- HandCallback

Install

```
pnpm add @philjs/gesture
```

Usage

```
import { AirCursor, BoundingBox, FingerState } from '@philjs/gesture';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-gesture/src/index.ts

Public API

- Direct exports: AirCursor, BoundingBox, FingerState, GestureCallback, GestureConfig, GestureController, GestureDefinition, GesturePresets, GestureRecognizer, GestureSequence, Hand, HandCallback, HandLandmark, HandLandmarkName, HandTracker, MotionAnalyzer, MotionPattern, PalmOrientation, Point2D, Point3D, RecognizedGesture, useAirCursor, useGesture, useGestureController, useHandTracking
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/go - Type: Node package - Purpose: Go server adapter and CLI tools for PhilJS - High-performance SSR and edge functions - Version: 0.1.0 - Location: packages/philjs-go - Entry points: packages/philjs-go/src/index.ts, packages/philjs-go/src/cli.ts, packages/philjs-go/src/codegen.ts, packages/philjs-go/src/server.ts - Keywords: philjs, go, golang, server, ssr, edge, cli, fast, performance

@philjs/go

Go server adapter and CLI tools for PhilJS - High-performance SSR and edge functions

Overview

Go server adapter and CLI tools for PhilJS - High-performance SSR and edge functions

Focus Areas

- philjs, go, golang, server, ssr, edge, cli, fast, performance

Entry Points

- packages/philjs-go/src/index.ts
- packages/philjs-go/src/cli.ts
- packages/philjs-go/src/codegen.ts
- packages/philjs-go/src/server.ts

Quick Start

```
import { CodegenOptions, GoServer, buildGoServer } from '@philjs/go';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- CodegenOptions
- GoServer
- buildGoServer
- checkGoInstalled
- createGoServer
- generateGoCode
- initGoProject
- watchAndGenerate

Install

```
pnpm add @philjs/go
```

Usage

```
import { CodegenOptions, GoServer, buildGoServer } from '@philjs/go';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `., ./cli, ./codegen, ./server`
- Source files: `packages/philjs-go/src/index.ts, packages/philjs-go/src/cli.ts, packages/philjs-go/src/codegen.ts, packages/philjs-go/src/server.ts`

Public API

- Direct exports: `CodegenOptions, GoServer, buildGoServer, checkGoInstalled, createGoServer, generateGoCode, initGoProject, watchAndGenerate`
- Re-exported names: (none detected)
- Re-exported modules: `./codegen.js, ./server.js, ./types.js`

License

MIT

```
## @philjs/graphql - Type: Node package - Purpose: GraphQL integration for PhilJS with typed queries and mutations - Version: 0.1.0 - Location: packages/philjs-graphql - Entry points: packages/philjs-graphql/src/index.ts
```

philjs-graphql

GraphQL integration for PhilJS with typed queries, mutations, and reactive caching.

Features

- **Type-safe queries and mutations** - Full TypeScript support with typed GraphQL operations
- **Signal-based reactive caching** - Automatic cache updates using PhilJS signals
- **Automatic request deduplication** - Avoid duplicate queries with intelligent caching
- **GraphQL subscriptions** - Real-time updates over WebSocket with graphql-ws protocol
- **Optimistic updates** - Advanced optimistic mutation handling with rollback and conflict resolution
- **Persisted queries** - Automatic persisted query (APQ) support with hash generation
- **Fragment colocation** - Type-safe fragments with masking and composition
- **Code generation** - Enhanced TypeScript code generation for operations and fragments
- **Retry logic** - Configurable retry with exponential backoff
- **SSR integration** - Works with PhilJS loaders and actions
- **Polling support** - Automatic refetching at intervals

Installation

```
pnpm add philjsgraphql graphql
```

Note: `graphql` is a peer dependency for type definitions.

Quick Start

1. Create a GraphQL Client

```

import { createGraphQLClient } from 'philjs-graphql';

const client = createGraphQLClient({
  endpoint: 'https://api.example.com/graphql',
  subscriptionEndpoint: 'wss://api.example.com/graphql', // optional
  headers: {
    'Authorization': 'Bearer YOUR_TOKEN'
  }
});

```

2. Query Data

```

import { createQuery, gql } from 'philjs-graphql';

const USERS_QUERY = gql`query GetUsers { users { id name email } }`;

const usersQuery = createQuery(client, {
  query: USERS_QUERY
});

// Access reactive data
console.log(usersQuery.data()); // undefined initially
console.log(usersQuery.loading()); // true
console.log(usersQuery.error()); // null

// Data becomes available when query resolves
setTimeout(() => {
  console.log(usersQuery.data()); // [{ id: 1, name: 'Alice', ... }]
  console.log(usersQuery.loading()); // false
}, 1000);

```

3. Execute Mutations

```

import { createMutation, gql } from 'philjs-graphql';

const CREATE_USER_MUTATION = gql`mutation CreateUser($name: String!, $email: String!) {
  createUser(name: $name, email: $email) {
    id
    name
    email
  }
}`;

const createUserMutation = createMutation(client, CREATE_USER_MUTATION);

// Execute mutation
async function handleSubmit(name: string, email: string) {
  try {
    const result = await createUserMutation.mutate({ name, email });
    console.log('User created:', result);
  } catch (error) {
    console.error('Failed to create user:', createUserMutation.error());
  }
}

```

4. GraphQL Subscriptions

```

import { gql } from 'philjs-graphql';

const MESSAGE_SUBSCRIPTION = gql`subscription OnMessageAdded {
  messageAdded {
    id
    content
    user {
      name
    }
  }
}`;

const unsubscribe = client.subscribe({
  subscription: MESSAGE_SUBSCRIPTION,
  onData: (data) => {
    console.log('New message:', data.messageAdded);
  },
  onError: (error) => {
    console.error('Subscription error:', error);
  }
});

// Clean up when done
unsubscribe();

```

SSR Integration

GraphQL Loaders

Use GraphQL queries with PhilJS loaders for server-side data fetching:

```

import { createGraphQLLoader, gql } from 'philjs-graphql';

const USER_QUERY = gql`query GetUser($id: ID!) {
  user(id: $id) {
    id
    name
    email
  }
}`;
export const userLoader = createGraphQLLoader(client, USER_QUERY);

// Use in route
export default function UserProfile() {
  const user = userLoader.use();

  return (
    <div>
      <h1>{user.name}</h1>
      <p>{user.email}</p>
    </div>
  );
}

```

GraphQL Actions

Use mutations with PhilJS actions for form submissions:

```

import { createGraphQLAction, gql } from 'philjs-graphql';

const UPDATE_PROFILE_MUTATION = gql`mutation UpdateProfile($name: String!, $bio: String!) {
  updateProfile(name: $name, bio: $bio) {
    id
    name
    bio
  }
}`;
export const updateProfileAction = createGraphQLAction(client, UPDATE_PROFILE_MUTATION);

// Use in form
export default function ProfileForm() {
  return (
    <form action={updateProfileAction}>
      <input name="name" placeholder="Name" />
      <textarea name="bio" placeholder="Bio" />
      <button type="submit">Update Profile</button>
    </form>
  );
}

```

Advanced Usage

Optimistic Updates

Update UI immediately before server responds:

```
const deleteTodoMutation = await client.mutate({
  mutation: DELETE_TODO_MUTATION,
  variables: { id: '123' },
  optimisticResponse: {
    deleteTodo: { id: '123', __typename: 'Todo' }
  },
  update: (cache, result) => {
    // Manually update cache
    const todosKey = 'GetTodos:{}';
    const todosData = cache.get(todosKey);
    if (todosData) {
      const todos = todosData();
      if (todos?.data) {
        todosData.set({
          data: {
            todos: todos.data.todos.filter(t => t.id !== '123')
          }
        });
      }
    }
  });
});
```

Polling

Automatically refetch data at intervals:

```
const usersQuery = createQuery(client, {
  query: USERS_QUERY,
  pollInterval: 5000 // Refetch every 5 seconds
});

// Control polling programmatically
usersQuery.stopPolling();
usersQuery.startPolling(10000); // Change to 10 seconds
```

Cache Management

```
// Clear all cache
client.clearCache();

// Clear cache by pattern
client.clearCache(' GetUser'); // Clear all GetUser queries
client.clearCache(/users/i); // Clear using regex

// Manual cache manipulation
const cacheKey = ' GetUser:{id:"123"}';
client.set(cacheKey, { data: { user: { id: '123', name: 'Alice' } } });
client.delete(cacheKey);
```

Retry Configuration

```
const client = createGraphQLClient({
  endpoint: 'https://api.example.com/graphql',
  retry: {
    maxRetries: 5,
    retryDelay: 2000,
    backoffMultiplier: 1.5
  }
});
```

Custom Cache TTL

```
const usersQuery = createQuery(client, {
  query: USERS_QUERY,
  cacheTTL: 60 * 1000 // Cache for 1 minute
});

// Or disable caching
const freshQuery = createQuery(client, {
  query: USERS_QUERY,
  noCache: true
});
```

API Reference

`createGraphQLClient(config)`

Creates a GraphQL client instance.

Config options: - `endpoint` - GraphQL endpoint URL (required) - `subscriptionEndpoint` - WebSocket endpoint for subscriptions - `headers` - Custom headers for all requests - `fetch` - Custom fetch implementation - `reactiveCache` - Enable signal-based caching (default: true) - `defaultCacheTTL` - Default cache duration in ms (default: 300000 / 5 minutes) - `retry` - Retry configuration object

`createQuery(client, options)`

Creates a reactive GraphQL query.

Returns: { data, error, loading, retryCount, refetch, startPolling, stopPolling }

`createMutation(client, mutation)`
Creates a GraphQL mutation.

Returns: { mutate, data, error, loading }

`client.subscribe(options)`
Creates a GraphQL subscription.

Returns: Unsubscribe function

`createGraphQLLoader(client, query)`
Creates a PhilJS loader for SSR.

`createGraphQLAction(client, mutation)`
Creates a PhilJS action for mutations.

gql Template Tag
Tagged template literal for GraphQL queries:

```
const query = gql`  
query GetUser($id: ID!) {  
  user(id: $id) {  
    name  
  }  
};`
```

Advanced Features (v2)

WebSocket Subscriptions

Enhanced subscription support with automatic reconnection and signal-based state:

```
import { createSubscriptionClient, useSubscription } from 'philjs-graphql';

// Create subscription client
const subscriptionClient = createSubscriptionClient({
  url: 'wss://api.example.com/graphql',
  connectionTimeout: 5000,
  maxReconnectAttempts: 5,
  keepalive: true,
  lazy: true, // Only connect when first subscription is active
});

// Use subscription in component
const MESSAGE_SUBSCRIPTION = gql`  
subscription OnMessage($channelId: ID!) {  
  newMessage(channelId: $channelId) {  
    id  
    text  
    user {  
      name  
    }  
  }  
};`  
  
function ChatComponent() {  
  const { data, error, active, connectionState } = useSubscription(  
    subscriptionClient,  
    {  
      query: MESSAGE_SUBSCRIPTION,  
      variables: { channelId: '1' },  
      onData: (data) => console.log('New message:', data),  
      onError: (error) => console.error('Subscription error:', error),  
    }  
  );  
  
  return (  
    <div>  
      <p>Connection: {connectionState()}</p>  
      {data() && <Message message={data().newMessage} />}  
    </div>  
  );  
}
```

Features: - graphql-ws protocol support - Automatic reconnection with exponential backoff - Heartbeat/keepalive support - Signal-based reactive state - Connection pooling for multiple subscriptions

Persisted Queries (APQ)

Automatic Persisted Queries reduce bandwidth and improve caching:

```

import {
  createPersistedQueryManager,
  buildPersistedQueryRequest,
  generatePersistedQueryManifest,
} from 'philjs-graphql';

// Create persisted query manager
const persistedQueryManager = createPersistedQueryManager({
  enabled: true,
  useGETForHashedQueries: true,
  includeQueryOnFirstRequest: false,
});

// Build request with persisted query
const request = await buildPersistedQueryRequest(
  persistedQueryManager,
  USERS_QUERY,
  { limit: 10 }
);

// Generate manifest for build-time optimization
const queries = {
  GetUsers: USERS_QUERY,
  GetUser: USER_QUERY,
  CreateUser: CREATE_USER_MUTATION,
};

const manifest = await generatePersistedQueryManifest(queries);
// Save manifest to file for server-side registration

```

Features: - SHA-256 hash generation - Automatic fallback to full query on hash miss - CDN-friendly GET requests - Query registry for pre-registration - Build-time manifest generation

Fragment Colocation

Type-safe fragments with masking for better data encapsulation:

```

import {
  defineFragment,
  maskFragment,
  unmaskFragment,
  useFragment,
  withFragment,
  buildQueryWithFragments,
} from 'philjs-graphql';

// Define a fragment
const UserFragment = defineFragment<{ id: string; name: string; email: string }>(`

  fragment UserFields on User {
    id
    name
    email
  }
`);

// Use fragment in query
const USERS_QUERY = buildQueryWithFragments(gql`query GetUsers {
  users {
    ...UserFields
  }
}`);
```

// Colocate fragment with component
const UserCard = withFragment(
 function UserCard({ user }) {
 const userData = useFragment(UserFragment, user);
 return <div>{userData.name}</div>;
 },
 UserFragment
);

// Fragment masking for data encapsulation
function UserList({ users }) {
 const maskedUsers = users.map(user => maskFragment(UserFragment, user));
 return maskedUsers.map(user => <UserCard user={user} key={user.__fragmentId} />);
}

```

**Features:** - Type-safe fragment definitions - Fragment masking for encapsulation - Automatic fragment spreading - Component colocation - Fragment composition and merging

#### Optimistic Updates

Advanced optimistic mutation handling with automatic rollback:

```

import {
 createOptimisticUpdateManager,
 buildOptimisticResponse,
} from 'philjs-graphql';

// Create optimistic update manager
const optimisticManager = createOptimisticUpdateManager({
 autoRollback: true,
 queueMutations: false,
 conflictResolution: 'queue',
});

// Build type-safe optimistic response
const optimisticResponse = buildOptimisticResponse<CreatePostMutation>()
 .typename('Post')
 .set('id', 'temp-id')
 .set('title', 'New Post')
 .set('published', false)
 .build();

// Use with mutation
const ADD_MESSAGE = gql`mutation AddMessage($text: String!) {
 addMessage(text: $text) {
 id
 text
 timestamp
 }
}
`;

const [addMessage] = useMutation(ADD_MESSAGE);

async function handleSubmit(text: string) {
 // Create optimistic mutation
 const mutation = optimisticManager.createMutation(
 ADD_MESSAGE,
 { text },
 optimisticResponse,
 (cache, { data }) => {
 // Update cache optimistically
 const messagesKey = 'GetMessages:{}';
 const messagesData = cache.get(messagesKey);
 if (messagesData) {
 const current = messagesData();
 if (current?.data) {
 messagesData.set({
 data: {
 messages: [...current.data.messages, data.addMessage],
 },
 });
 }
 }
 }
);
}

// Apply optimistic update
const snapshot = optimisticManager.applyOptimistic(mutation.id, client);

try {
 const result = await addMessage({ text });
 // Commit on success
 optimisticManager.commit(mutation.id, client, result);
} catch (error) {
 // Automatic rollback on error
 optimisticManager.rollback(mutation.id, client, error);
}
}

```

**Features:** - Automatic rollback on error - Snapshot-based state restoration - Mutation queue management - Conflict resolution strategies - Type-safe optimistic response builder

#### Code Generation

Generate TypeScript types and hooks for GraphQL operations:

```

import {
 createCodegen,
 createBatchCodegen,
 extractOperationInfo,
 extractFragmentInfo,
} from 'philjs-graphql';

// Configure code generator
const codegen = createCodegen({
 schema: 'https://api.example.com/graphql',
 documents: ['src/**/*.graphql', 'src/**/*.tsx'],
 outputDir: 'src/generated',
 generateHooks: true,
 generateFragments: true,
 generateSubscriptions: true,
 generatePersistedQueries: true,
 scalars: {
 DateTime: 'string',
 JSON: 'any',
 },
});

// Generate types for operations
const queryTypes = codegen.generateQueryTypes('GetUsers', USERS_QUERY, {
 generateHook: true,
});

const mutationTypes = codegen.generateMutationTypes('CreateUser', CREATE_USER_MUTATION, {
 generateHook: true,
 generateOptimistic: true,
});

const subscriptionTypes = codegen.generateSubscriptionTypes('OnMessage', MESSAGE_SUBSCRIPTION);

// Batch code generation
const batchCodegen = createBatchCodegen({
 schema: 'https://api.example.com/graphql',
 documents: [],
 outputDir: 'src/generated',
});

batchCodegen.addOperation('GetUsers', 'query', USERS_QUERY);
batchCodegen.addOperation('CreateUser', 'mutation', CREATE_USER_MUTATION);
batchCodegen.addFragment('UserFields', 'User', USER_FRAGMENT);

const generatedCode = batchCodegen.generate();
// Write to file: src/generated/graphql.ts

```

**Generated types example:**

```

// GetUsers Query
export interface GetUsersVariables {
 limit?: number;
 offset?: number;
}

export interface GetUsersQuery {
 users: Array<{
 id: string;
 name: string;
 email: string;
 }>;
}

export function useGetUsersQuery(
 variables?: GetUsersVariables,
 options?: GraphQLQueryOptions<GetUsersVariables>
) {
 return createQuery<GetUsersQuery, GetUsersVariables>(client, {
 query: GetUsersDocument,
 variables,
 ...options,
 });
}

// CreateUser Mutation with optimistic response
export interface CreateUserVariables {
 name: string;
 email: string;
}

export interface CreateUserMutation {
 createUser: {
 id: string;
 name: string;
 email: string;
 };
}

export type CreateUserOptimisticResponse = Partial<CreateUserMutation>

export function useCreateUserMutation() {
 return createMutation<CreateUserMutation, CreateUserVariables>(
 client,
 CreateUserDocument
);
}

```

**Features:** - Generate TypeScript types from GraphQL schema - Generate hooks for queries, mutations, and subscriptions - Generate optimistic response types - Generate fragment types - Extract operation and fragment information - Batch generation support

#### Best Practices

##### 1. Use Fragments for Reusable Fields

```

const UserFieldsFragment = defineFragment('UserFields', 'User', gql`
 fragment UserFields on User {
 id
 name
 email
 avatar
 }
`);

// Reuse in multiple queries
const USERS_QUERY = buildQueryWithFragments(gql`
 query GetUsers {
 users {
 ...UserFields
 }
 }
`);

```

##### 2. Enable Persisted Queries for Production

```

const client = createGraphQLClient({
 endpoint: 'https://api.example.com/graphql',
 // Enable persisted queries
});

const persistedQueryManager = createPersistedQueryManager({
 enabled: true,
 useGETForHashedQueries: true, // Better CDN caching
});

```

##### 3. Use Optimistic Updates for Better UX

```
// Always provide optimistic response for mutations that update lists
const [addItem] = useMutation(ADD_ITEM, {
 optimisticResponse: {
 addItem: {
 id: 'temp-' + Date.now(),
 name: inputValue,
 __typename: 'Item',
 },
 },
 update: (cache, { data }) => {
 // Update cache optimistically
 },
});

```

#### 4. Colocate Fragments with Components

```
// Keep fragments close to components that use them
const UserCard = withFragment(
 function UserCard({ user }) {
 return <div>{user.name}</div>;
 },
 UserCardFragment
);

```

#### 5. Use Subscriptions for Real-Time Features

```
// Use subscriptions for chat, notifications, live updates
const { data, connectionState } = useSubscription(subscriptionClient, {
 query: MESSAGES_SUBSCRIPTION,
 variables: { channelId },
});

```

#### Documentation

For more information, see the PhilJS documentation.

#### API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

##### Entry Points

- Export keys: .
- Source files: packages/philijs-graphql/src/index.ts

##### Public API

- Direct exports: CacheStore, DocumentNode, GraphQLClient, GraphQLClientConfig, GraphQLMutationOptions, GraphQLQueryOptions, GraphQLResponse, GraphQLSubscriptionOptions, createGraphQLAction, createGraphQLClient, createGraphQLLoader, createMutation, createQuery, gql
- Re-exported names: BatchCodegen, CodegenConfig, ConflictResolver, CustomConflictResolver, FirstWriteWinsResolver, FragmentDefinition, FragmentRegistry, FragmentUtils, GeneratedFragment, GeneratedOperation, GraphQLCodegen, LastWriteWinsResolver, MaskedFragment, MutationQueue, OptimisticMutation, OptimisticResponseBuilder, OptimisticUpdateConfig, OptimisticUpdateManager, OptimisticUpdateSnapshot, PersistedQueryConfig, PersistedQueryLink, PersistedQueryManager, PersistedQueryRegistry, SubscriptionClient, SubscriptionConfig, SubscriptionHandle, SubscriptionOptions, SubscriptionState, buildOptimisticResponse, buildPersistedQueryRequest, buildQueryWithFragments, composeFragments, createBatchCodegen, createCodegen, createFragmentRegistry, createMutationQueue, createOptimisticUpdateManager, createPersistedQueryManager, createPersistedQueryRegistry, createSubscriptionClient, defineFragment, extractFragmentInfo, extractOperationInfo, extractQueryHash, fragment, generatePersistedQueryManifest, getComponentFragment, getFragmentRegistry, inlineFragment, isMaskedFragment, maskFragment, mergeFragmentData, runCodegen, selectFromFragment, shouldRetryWithFullQuery, spreadFragment, unmaskFragment, useFragment, useSubscription, withFragment
- Re-exported modules: ./codegen-enhanced.js, ./fragments.js, ./optimistic.js, ./persisted.js, ./subscription.js

#### License

MIT

## @philijs/haptic - Type: Node package - Purpose: Haptic feedback system for PhilJS - vibration patterns, gamepad haptics, XR haptics - Version: 0.1.0 - Location: packages/philijs-haptic - Entry points: packages/philijs-haptic/src/index.ts - Keywords: philijs, haptic, vibration, feedback, gamepad, xr, mobile

#### @philijs/haptic

Haptic feedback system for PhilJS - vibration patterns, gamepad haptics, XR haptics

#### Overview

Haptic feedback system for PhilJS - vibration patterns, gamepad haptics, XR haptics

#### Focus Areas

- philijs, haptic, vibration, feedback, gamepad, xr, mobile

#### Entry Points

- packages/philijs-haptic/src/index.ts

#### Quick Start

```
import { GamepadHapticEffect, GamepadHaptics, HAPTIC_PATTERNS } from '@philijs/haptic';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

#### Exports at a Glance

- GamepadHapticEffect
- GamepadHaptics
- HAPTIC\_PATTERNS
- HapticComposer
- HapticConfig
- HapticEngine
- HapticIntensity
- HapticPattern
- HapticType
- XRHapticPulse
- XRHaptics
- useGamepadHaptics

#### Install

```
pnpm add @philjs/haptic
```

#### Usage

```
import { GamepadHapticEffect, GamepadHaptics, HAPTIC_PATTERNS } from '@philjs/haptic';
```

#### Scripts

- pnpm run build
- pnpm run test

#### Compatibility

- Node >= 24
- TypeScript 6

#### API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

#### Entry Points

- Export keys: .
- Source files: packages/philjs-haptic/src/index.ts

#### Public API

- Direct exports: GamepadHapticEffect, GamepadHaptics, HAPTIC\_PATTERNS, HapticComposer, HapticConfig, HapticEngine, HapticIntensity, HapticPattern, HapticType, XRHapticPulse, XRHaptics, useGamepadHaptics, useHaptic, useHapticPattern, useXRHaptics
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

#### License

MIT

```
@philjs/hollow - Type: Node package - Purpose: Framework-agnostic Web Components with React, Vue, and Svelte wrappers - Version: 0.1.0 - Location: packages/philjs-hollow - Entry points: packages/philjs-hollow/src/index.ts, ./react, ./vue, ./svelte, packages/philjs-hollow/src/components/index.ts, packages/philjs-hollow/src/styles/index.ts, ./philjs, packages/philjs-hollow/src/core/index.ts - Keywords: philjs, web-components, custom-elements, react, vue, svelte, framework-agnostic
```

#### @philjs/hollow

Framework-agnostic headless UI components built as Web Components with wrappers for React, Vue, Svelte, and native PhilJS. Inspired by Radix UI and Headless UI, Hollow provides unstyled, accessible components that integrate seamlessly with any styling solution.

#### Features

- **Web Components:** Native browser support, works everywhere
- **Framework Wrappers:** First-class support for React, Vue, Svelte, and PhilJS
- **Headless/Unstyled:** Full control over styling with CSS custom properties
- **Accessible:** ARIA compliant with keyboard navigation
- **Form Integration:** Native form association for all form components
- **TypeScript:** Full type safety with comprehensive types
- **Design Tokens:** Customizable via CSS custom properties

#### Installation

```
npm install @philjs/hollow
or
pnpm add @philjs/hollow
or
bun add @philjs/hollow
```

#### Quick Start

[Vanilla JavaScript / Web Components](#)

```

<script type="module">
 import '@philjs/hollow';
</script>

<hollow-button variant="primary" size="md">
 Click me
</hollow-button>

<hollow-input type="email" placeholder="Enter email" />

<hollow-modal open size="md">
 <div slot="header">Modal Title</div>
 <p>Modal content goes here</p>
 <div slot="footer">
 <hollow-button variant="secondary">Cancel</hollow-button>
 <hollow-button variant="primary">Save</hollow-button>
 </div>
</hollow-modal>

```

#### React

```

import { Button, Input, Modal, Select } from '@philjs/hollow/react';

function App() {
 const [open, setOpen] = useState(false);

 return (
 <>
 <Button
 variant="primary"
 onClick={() => setOpen(true)}
 >
 Open Modal
 </Button>

 <Modal
 open={open}
 onClose={() => setOpen(false)}
 size="md"
 >
 <Input
 type="email"
 placeholder="Enter email"
 onChange={(e) => console.log(e.detail.value)}
 />
 </Modal>
 </>
);
}

Vue 3

```

#### Vue 3

```

<script setup>
import { HollowButton, HollowInput, HollowModal } from '@philjs/hollow/vue';
import { ref } from 'vue';

const open = ref(false);
const email = ref('');
</script>

<template>
 <HollowButton
 variant="primary"
 @hollow-click="open = true"
 >
 Open Modal
 </HollowButton>

 <HollowModal
 :open="open"
 @hollow-close="open = false"
 size="md"
 >
 <HollowInput
 type="email"
 :value="email"
 @hollow-input="email = $event.detail.value"
 />
 </HollowModal>
</template>

```

#### Vue Plugin

```

import { createApp } from 'vue';
import { HollowPlugin } from '@philjs/hollow/vue';

const app = createApp(App);
app.use(HollowPlugin);
app.mount('#app');

```

#### Svelte

```

<script>
 import { hollow, hollowButton, hollowModal } from '@philjs/hollow/svelte';

 let open = false;
</script>

<hollow-button
 use:hollowButton={{
 variant: 'primary',
 onClick: () => open = true
 }}
>
 Open Modal
</hollow-button>

<hollow-modal
 use:hollowModal={{
 open,
 onClose: () => open = false
 }}
>
<hollow-input
 use:hollow={{
 type: 'email',
 placeholder: 'Enter email'
 }}
/>
</hollow-modal>

```

#### PhilJS Native

```

import { Button, Input, Modal, Select } from '@philjs/hollow/philjs';
import { signal } from '@philjs/core';

function App() {
 const open = signal(false);
 const email = signal('');

 return (
 <>
 <Button
 variant="primary"
 onClick={() => open.set(true)}
 >
 Open Modal
 </Button>

 <Modal
 open={() => open()}
 onClose={() => open.set(false)}
 >
 <Input
 value={() => email()}
 onChange={(e) => email.set(e.value)}
 />
 </Modal>
 </>
);
}

```

#### Components

##### Button

A versatile button component with multiple variants and states.

```

<hollow-button
 variant="primary|secondary|outline|ghost|link|destructive"
 size="sm|md|lg|xl"
 disabled
 loading
 type="button|submit|reset"
>
 Button Text
</hollow-button>

```

**Events:** - hollow-click: Fired when button is clicked

##### Input

A text input with validation and error states.

```
<hollow-input
 variant="default|filled|flushed|unstyled"
 size="sm|md|lg"
 type="text|email|password|number|tel|url|search|date|time"
 value="initial value"
 placeholder="Enter text..."
 disabled
 readonly
 required
 minlength="3"
 maxlength="100"
 pattern="[A-Za-z]+"
 name="fieldName"
 error="Validation error message"
/>
```

**Events:** - hollow-input: Fired on each input change - hollow-change: Fired when input loses focus

#### Card

A container component for grouping content.

```
<hollow-card
 variant="default|elevated|outlined|filled"
 padding="none|sm|md|lg|x1"
 interactive
 selected
>
<div slot="header">Card Header</div>
Card content
<div slot="footer">Card Footer</div>
</hollow-card>
```

**Events:** - hollow-click: Fired when interactive card is clicked

#### Modal

A dialog component with backdrop and animations.

```
<hollow-modal
 open
 size="sm|md|lg|x1|full"
 animation="scale|fade|slide|none"
 closable
 close-on-backdrop
 close-on-escape
 persistent
>
<div slot="header">Modal Title</div>
Modal content
<div slot="footer">Modal Actions</div>
</hollow-modal>
```

**Events:** - hollow-open: Fired when modal opens - hollow-close: Fired when modal closes

**Methods:** - open(): Open the modal - close(): Close the modal

#### Select

A dropdown select component with search and multi-select support.

```
<hollow-select
 variant="default|filled|flushed"
 size="sm|md|lg"
 value="selected-value"
 placeholder="Select an option..."
 disabled
 required
 searchable
 clearable
 multiple
 options='[{"value": "a", "label": "Option A"}, {"value": "b", "label": "Option B"}]'
 name="fieldName"
 error="Error message"
/>
```

**Events:** - hollow-change: Fired when selection changes - hollow-toggle: Fired when dropdown opens/closes

**Methods:** - open(): Open the dropdown - close(): Close the dropdown - clear(): Clear the selection

#### Checkbox

A checkbox with indeterminate state support.

```
<hollow-checkbox
 variant="default|primary|success|warning|error"
 size="sm|md|lg"
 checked
 indeterminate
 disabled
 required
 name="fieldName"
 value="checkbox-value"
>
 Label text
</hollow-checkbox>
```

**Events:** - hollow-change: Fired when checked state changes

**Methods:** - toggle(): Toggle the checkbox - setChecked(boolean): Set checked state - setIndeterminate(boolean): Set indeterminate state

#### Switch

An iOS-style toggle switch.

```
<hollow-switch
 variant="default|primary|success|warning|error"
 size="sm|md|lg"
 checked
 disabled
 required
 name="fieldName"
 value="switch-value"
 label-on="ON"
 label-off="OFF"
>
 Enable feature
</hollow-switch>
```

**Events:** - hollow-change: Fired when switch state changes

**Methods:** - toggle(): Toggle the switch - setChecked(boolean): Set checked state

#### Tabs

A tabbed navigation component.

```
<hollow-tabs
 variant="default|pills|underline|enclosed"
 size="sm|md|lg"
 active="tab1"
 alignment="start|center|end|stretch"
 tabs='[{"id": "tab1", "label": "Tab 1"}, {"id": "tab2", "label": "Tab 2"}]'
```

**Events:** - hollow-change: Fired when active tab changes

**Methods:** - selectTab(id): Select a tab by ID - getActiveTab(): Get the active tab ID - setTabs(tabs): Set tabs programmatically

#### Accordion

Collapsible sections with smooth animations.

```
<hollow-accordion
 variant="default|bordered|separated|ghost"
 multiple
 collapsible
 expanded="item1,item2"
 items='[{"id": "item1", "title": "Section 1"}, {"id": "item2", "title": "Section 2"}]'
```

**Events:** - hollow-change: Fired when expansion state changes

**Methods:** - toggleItem(id): Toggle an item - expand(id): Expand an item - collapse(id): Collapse an item - expandAll(): Expand all items (multiple mode only) - collapseAll(): Collapse all items - getExpandedItems(): Get expanded item IDs - isExpanded(id): Check if item is expanded

#### Styling

##### CSS Custom Properties

All components use CSS custom properties for theming:

```

:root {
 /* Colors */
 --hollow-color-primary: #3b82f6;
 --hollow-color-primary-foreground: #ffffff;
 --hollow-color-secondary: #f1f5f9;
 --hollow-color-secondary-foreground: #0f172a;
 --hollow-color-success: #22c55e;
 --hollow-color-warning: #f59e0b;
 --hollow-color-error: #ef4444;
 --hollow-color-text: #0f172a;
 --hollow-color-text-muted: #64748b;
 --hollow-color-background: #ffffff;
 --hollow-color-border: #e2e8f0;
 --hollow-color-ring: #3b82f680;

 /* Typography */
 --hollow-font-family: system-ui, sans-serif;
 --hollow-font-size-sm: 0.875rem;
 --hollow-font-size-md: 1rem;
 --hollow-font-size-lg: 1.125rem;
 --hollow-font-weight-medium: 500;
 --hollow-font-weight-semibold: 600;

 /* Spacing */
 --hollow-spacing-1: 0.25rem;
 --hollow-spacing-2: 0.5rem;
 --hollow-spacing-3: 0.75rem;
 --hollow-spacing-4: 1rem;

 /* Border Radius */
 --hollow-radius-sm: 0.25rem;
 --hollow-radius-md: 0.375rem;
 --hollow-radius-lg: 0.5rem;

 /* Shadows */
 --hollow-shadow-sm: 0 1px 2px 0 rgba(0 0 0 / 0.05);
 --hollow-shadow-md: 0 4px 6px -1px rgba(0 0 0 / 0.1);

 /* Transitions */
 --hollow-transition-fast: 100ms;
 --hollow-transition-normal: 200ms;
 --hollow-transition-easing: cubic-bezier(0.4, 0, 0.2, 1);
}

```

#### CSS Parts

Style component internals using `::part()`:

```

hollow-button::part(button) {
 text-transform: uppercase;
}

hollow-input::part(input) {
 border-width: 2px;
}

hollow-modal::part(backdrop) {
 background: rgba(0, 0, 0, 0.8);
}

hollow-modal::part(container) {
 border-radius: 16px;
}

```

#### Dark Mode

Override CSS custom properties for dark mode:

```

@media (prefers-color-scheme: dark) {
 :root {
 --hollow-color-text: #f8fafc;
 --hollow-color-text-muted: #94a3b8;
 --hollow-color-background: #0f172a;
 --hollow-color-background-muted: #1e293b;
 --hollow-color-border: #334155;
 }
}

```

#### Tailwind CSS Integration

Use with Tailwind by targeting the parts:

```

<hollow-button class="[&::part(button)]:rounded-full [&::part(button)]:uppercase">
 Styled Button
</hollow-button>

```

#### Framework Integration

React - useRef Access

```

import { useRef, useEffect } from 'react';

function App() {
 const modalRef = useRef<HTMLDivElement>(null);

 const openModal = () => {
 (modalRef.current as any)?.open();
 };

 return (
 <>
 <Button onClick={openModal}>Open</Button>
 <Modal ref={modalRef}>Content</Modal>
 </>
);
}

```

#### Vue - Template Refs

```

<script setup>
import { ref } from 'vue';

const modalRef = ref(null);

const openModal = () => {
 modalRef.value?.open();
};

<template>
 <HollowButton @hollow-click="openModal">Open</HollowButton>
 <HollowModal ref="modalRef">Content</HollowModal>
</template>

```

#### Svelte - bind:this

```

<script>
let modalElement;

const openModal = () => {
 modalElement?.open();
};

<hollow-button on:hollow-click={openModal}>Open</hollow-button>
<hollow-modal bind:this={modalElement}>Content</hollow-modal>

```

#### PhilJS - useHollowRef

```

import { useHollowRef } from '@philjs/hollow/philjs';

function App() {
 const modalRef = useHollowRef();

 return (
 <>
 <Button onClick={() => modalRef.current?.open()}>
 Open
 </Button>
 <Modal ref={modalRef.set}>Content</Modal>
 </>
);
}

```

#### Form Integration

All form components support native form association:

```

<form id="myForm">
 <hollow-input name="email" required />
 <hollow-checkbox name="agree" value="yes" required>
 I agree to terms
 </hollow-checkbox>
 <hollow-select name="country" options='[...]' required />
 <hollow-button type="submit">Submit</hollow-button>
</form>

<script>
 document.getElementById('myForm').addEventListener('submit', (e) => {
 const formData = new FormData(e.target);
 console.log(Object.fromEntries(formData));
 });
</script>

```

#### Accessibility

All components follow WAI-ARIA guidelines:

- Proper ARIA roles and attributes
- Keyboard navigation support
- Focus management
- Screen reader announcements

#### Keyboard Shortcuts

Component	Key	Action
Button	Enter, Space	Activate
Modal	Escape	Close
Select	Arrow Up/Down	Navigate options
Select	Enter	Select option
Select	Escape	Close dropdown
Tabs	Arrow Left/Right	Navigate tabs
Tabs	Home/End	First/last tab
Accordion	Arrow Up/Down	Navigate items
Checkbox	Space	Toggle
Switch	Space, Enter	Toggle

#### API Reference

##### React Wrappers

```
import {
 Button, Input, Card, Modal,
 Select, Checkbox, Switch, Tabs,
 Accordion, AccordionItem,
 useHollowEvent
} from '@philjs/hollow/react';
```

##### Vue Wrappers

```
import {
 HollowButton, HollowInput, HollowCard, HollowModal,
 HollowSelect, HollowCheckbox, HollowSwitch, HollowTabs,
 HollowAccordion, HollowAccordionItem,
 vHollowProps, HollowPlugin
} from '@philjs/hollow/vue';
```

##### Svelte Actions

```
import {
 hollow, hollowButton, hollowInput, hollowCard,
 hollowModal, hollowSelect, hollowCheckbox, hollowSwitch,
 hollowTabs, hollowAccordion, hollowAccordionItem,
 createHollowStore, onHollowEvent
} from '@philjs/hollow/svelte';
```

##### PhilJS Bindings

```
import {
 Button, Input, Card, Modal,
 Select, Checkbox, Switch, Tabs,
 Accordion, AccordionItem,
 bindProp, useHollowRef, hollowProps
} from '@philjs/hollow/philjs';
```

##### Design Tokens

```
import {
 tokens, colors, typography, spacing,
 borderRadius, shadows, transitions, zIndex,
 designTokensCSS, createTheme
} from '@philjs/hollow/tokens';
```

##### Browser Support

- Chrome 90+
- Firefox 90+
- Safari 15+
- Edge 90+

Requires native Custom Elements and Shadow DOM support.

##### API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

##### Entry Points

- Export keys: .., ./react, ./vue, ./svelte, ./components, ./styles, ./philjs, ./core

- Source files: packages/philjs-hollow/src/index.ts, packages/philjs-hollow/src/components/index.ts, packages/philjs-hollow/src/styles/index.ts, packages/philjs-hollow/src/core/index.ts

#### Public API

- Direct exports: (none detected)
- Re-exported names: AccordionItem, AccordionVariant, ButtonSize, ButtonVariant, CardPadding, CardVariant, CheckboxSize, CheckboxVariant, HollowAccordion, HollowAccordionItem, HollowButton, HollowCard, HollowCheckbox, HollowElement, HollowInput, HollowModal, HollowSelect, HollowSwitch, HollowTab, HollowTabList, HollowTabPanel, HollowTabs, InputSize, InputType, InputVariant, ModalAnimation, ModalSize, PropertyOptions, SelectOption, SelectSize, SelectVariant, SwitchSize, SwitchVariant, TabDefinition, TabsAlignment, TabsSize, TabsVariant, borderRadius, colors, createTheme, defineElement, designTokensCSS, philjs, property, react, shadows, spacing, svelte, tokens, transitions, typography, vue, zIndex
- Re-exported modules: ./accordion.js, ./base-element.js, ./button.js, ./card.js, ./checkbox.js, ./components/button.js, ./components/card.js, ./components/input.js, ./core/base-element.js, ./input.js, ./modal.js, ./select.js, ./styles/tokens.js, ./switch.js, ./tabs.js, ./tokens.js, ./wrappers/philjs.js, ./wrappers/react.js, ./wrappers/svelte.js, ./wrappers/vue.js

#### License

MIT

## @philjs/html - Type: Node package - Purpose: HTML-first reactive framework for PhilJS - declarative attributes, progressive enhancement - Version: 0.1.0 - Location: packages/philjs-html - Entry points: packages/philjs-html/src/index.ts, ./attributes, packages/philjs-html/src/minimal.ts - Keywords: philjs, html-first, progressive-enhancement, declarative, reactive, signals, web-components

#### PhilJS HTML

HTML-first reactive framework with HTMX and Alpine.js compatibility.

Build interactive UIs by enhancing HTML with reactive attributes - no build step required.

#### Features

- **Alpine.js Compatible** - Use x-data, x-text, x-show, x-on, etc.
- **HTMX Compatible** - Use hx-get, hx-post, hx-target, etc.
- **Minimal Runtime** - Optional <3KB mode for ultra-light apps
- **PhilJS Signals** - Powered by efficient fine-grained reactivity
- **No Build Required** - Works directly in the browser

#### Installation

```
npm install philjs-html
```

Or use directly in HTML:

```
<script src="https://unpkg.com/philjs-html"></script>
```

#### Quick Start

##### Alpine-Style Reactivity

```
<div x-data="{ count: 0 }">

 <button @click="count++>+</button>
 <button @click="count-->-</button>
</div>
```

##### HTMX-Style Server Interactions

```
<button hx-get="/api/users" hx-target="#users-list" hx-swap="innerHTML">
 Load Users
</button>

<div id="users-list"></div>
```

#### Combined

```
<div x-data="{ loading: false }">
 <button
 hx-get="/api/data"
 hx-target="#result"
 @htmx:beforeRequest="loading = true"
 @htmx:afterRequest="loading = false"
 :disabled="loading"
 >
 Load Data
 Loading...
 </button>

 <div id="result"></div>
</div>
```

#### Alpine.js Directives

##### x-data

Initialize reactive data:

```
<div x-data="{ open: false, items: [] }">
 <!-- reactive scope -->
</div>
```

**x-text / x-html**

Set content:

```

<div x-html="htmlContent"></div>
```

**x-show / x-if**

Conditional rendering:

```
<div x-show="isVisible">Toggles visibility</div>
<template x-if="condition">
 <div>Conditionally rendered</div>
</template>
```

**x-for**

List rendering:

```

 <template x-for="item in items">
 <li x-text="item.name">
 </template>

<!-- With index -->
<template x-for="(item, index) in items">
 <li x-text="${index}: ${item}">
</template>
```

**x-on / @**

Event handling:

```
<button x-on:click="handleClick">Click</button>
<button @click="count++">+</button>
<button @click.prevent="submit">Submit</button>
<input @keyup.enter="search" />
```

Modifiers: .prevent, .stop, .once, .self, .capture

**x-model**

Two-way binding:

```
<input x-model="name" />
<textarea x-model="bio"></textarea>
<select x-model="selected">
 <option>A</option>
 <option>B</option>
</select>
```

**x-bind / :**

Attribute binding:

```
<div x-bind:class="{ active: isActive }"></div>

<button :disabled="loading">Submit</button>
```

**x-ref**

Element references:

```
<input x-ref="input" />
<button @click="$refs.input.focus()">Focus</button>
```

**x-effect**

Side effects:

```
<div x-effect="console.log('Count is', count)"></div>
```

**x-transition**

Transitions:

```
<div
 x-show="open"
 x-transition:enter="transition ease-out duration-300"
 x-transition:enter-start="opacity-0"
 x-transition:enter-end="opacity-100"
 x-transition:leave="transition ease-in duration-200"
 x-transition:leave-start="opacity-100"
 x-transition:leave-end="opacity-0"
>
 Content
</div>
```

**HTMX Attributes****HTTP Methods**

```
<button hx-get="/api/data">GET</button>
<form hx-post="/api/submit">POST</form>
<button hx-put="/api/update">PUT</button>
<button hx-patch="/api/patch">PATCH</button>
<button hx-delete="/api/delete">DELETE</button>
```

#### Targeting

```
<button hx-get="/api" hx-target="#result">Load into #result</button>
<button hx-get="/api" hx-target=".closest .container">Load into closest</button>
<button hx-get="/api" hx-target="this">Replace self</button>
```

#### Swapping

```
<button hx-get="/api" hx-swap="innerHTML">Replace inner (default)</button>
<button hx-get="/api" hx-swap="outerHTML">Replace element</button>
<button hx-get="/api" hx-swap="beforeend">Append</button>
<button hx-get="/api" hx-swap="afterbegin">Prepend</button>
```

#### Triggers

```
<input hx-get="/search" hx-trigger="keyup changed delay:500ms" />
<div hx-get="/poll" hx-trigger="every 5s">Polling</div>
<div hx-get="/data" hx-trigger="revealed">Load on visible</div>
```

#### Values & Headers

```
<button hx-get="/api" hx-vals='{"key": "value"}'>With values</button>
<button hx-get="/api" hx-headers='{"X-Custom": "value"}'>With headers</button>
```

#### Indicators

```
<button hx-get="/api" hx-indicator="#spinner">
 Load
 Loading...
</button>
```

#### Global Stores

```
<script>
 PhilJSHTML.store('app', {
 user: null,
 theme: 'light',
 toggleTheme() {
 this.theme = this.theme === 'light' ? 'dark' : 'light';
 }
 });
</script>

<div x-data>

 <button @click="$store('app').toggleTheme()">Toggle</button>
</div>
```

#### Reusable Components

```
<script>
 PhilJSHTML.data('counter', () => ({
 count: 0,
 increment() { this.count++ },
 decrement() { this.count-- }
 }));
</script>

<div x-data="counter">

 <button @click="increment">+</button>
</div>
```

#### Minimal Mode

For ultra-lightweight apps (<3KB):

```
<script src="philjs-html/minimal" data-minimal></script>

<div x-data="{ count: 0 }">

 <button @click="count++">+</button>
</div>
```

Minimal mode includes: x-data, x-text, x-show, x-on, x-model, x-bind

#### Configuration

```

import { init, configureHtmx } from 'philjs-html';

// Configure HTMX
configureHtmx({
 timeout: 10000,
 withCredentials: true,
 headers: { 'X-Requested-With': 'PhilJS' }
});

// Manual initialization
init({
 alpine: true,
 htmx: true,
 minimal: false,
 root: document.getElementById('app')
});

```

## Events

### Alpine Events

```

<div @custom-event.window="handleEvent">
 <button @click="$dispatch('custom-event', { data: 123 })">
 Dispatch
 </button>
</div>

```

### HTMX Events

```

<div
 @htmx:beforeRequest="loading = true"
 @htmx:afterRequest="loading = false"
 @htmx:error="handleError"
>
...
</div>

```

## API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

### Entry Points

- Export keys: .., ./attributes, ./minimal
- Source files: packages/philjs-html/src/index.ts, packages/philjs-html/src/minimal.ts

### Public API

- Direct exports: initMinimal, minimalVersion
- Re-exported names: PhilJSHTML, init
- Re-exported modules: ./alpine.js, ./directives.js, ./htmx.js, ./minimal.js, ./runtime.js

### License

MIT

## @philjs/hypermedia - Type: Node package - Purpose: HTML-over-the-wire for PhilJS - server-driven UI updates with declarative attributes - Version: 0.1.0 - Location: packages/philjs-htmx - Entry points: packages/philjs-htmx/src/index.ts - Keywords: philjs, hypermedia, html-over-the-wire, progressive-enhancement, server-driven, partial-updates

### @philjs/hypermedia

HTML-over-the-wire for PhilJS - server-driven UI updates with declarative attributes

### Overview

HTML-over-the-wire for PhilJS - server-driven UI updates with declarative attributes

### Focus Areas

- philjs, hypermedia, html-over-the-wire, progressive-enhancement, server-driven, partial-updates

### Entry Points

- packages/philjs-htmx/src/index.ts

### Quick Start

```

import { AjaxOptions, HTMXConfig, HTMSError } from '@philjs/hypermedia';

```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

### Exports at a Glance

- AjaxOptions
- HTMXConfig
- HTMSError
- HTMXExtension
- HTMXRequestEvent
- HTMXResponseEvent

- HTMXSwapEvent
- SwapStyle
- TriggerEvent
- TriggerModifier
- TriggerSpec
- defineExtension

#### Install

```
pnpm add @philjs/hypermedia
```

#### Usage

```
import { AjaxOptions, HTMXConfig, HTMXError } from '@philjs/hypermedia';
```

#### Scripts

- pnpm run build
- pnpm run test

#### Compatibility

- Node >=24
- TypeScript 6

#### API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

#### Entry Points

- Export keys: .
- Source files: packages/philjs-htmx/src/index.ts

#### Public API

- Direct exports: AjaxOptions, HTMXConfig, HTMXError, HTMXExtension, HTMXRequestEvent, HTMXResponseEvent, HTMXSwapEvent, SwapStyle, TriggerEvent, TriggerModifier, TriggerSpec, defineExtension, getHTMXInfo, htmx, htmxResponse, htmxStyles, initHTMX, injectStyles, isHTMXRequest, removeExtension
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

#### License

MIT

## @philjs/i18n - Type: Node package - Purpose: Internationalization utilities for PhilJS - Version: 0.1.0 - Location: packages/philjs-i18n - Entry points: packages/philjs-i18n/src/index.ts, ./translator, ./formatter, ./detector - Keywords: philjs, i18n, internationalization, localization, translation

#### @philjs/i18n

Internationalization utilities for PhilJS applications. Provides translation management, locale formatting, and pluralization with a simple, type-safe API.

#### Requirements

- **Node.js 24** or higher
- **TypeScript 6** or higher
- **ESM only** - CommonJS is not supported

#### Installation

```
pnpm add @philjs/i18n
```

#### Basic Usage

```

import { I18nProvider, useTranslation, useLocale } from '@philjs/i18n';

const translations = {
 en: { greeting: 'Hello, {name}!', items: '{count} item|{count} items' },
 es: { greeting: 'Hola, {name}!', items: '{count} artículo|{count} artículos' },
};

function App() {
 return (
 <I18nProvider locale="en" translations={translations}>
 <HomePage />
 </I18nProvider>
);
}

function HomePage() {
 const { t } = useTranslation();
 const { formatDate, formatNumber } = useLocale();

 return (
 <div>
 <h1>{t('greeting', { name: 'World' })}</h1>
 <p>{t('items', { count: 5 })}</p>
 <p>{formatDate(new Date())}</p>
 <p>{formatNumber(1234.56, { style: 'currency', currency: 'USD' })}</p>
 </div>
);
}

```

## Features

- **Translation Management** - Simple key-based translations
- **Interpolation** - Variable substitution in strings
- **Pluralization** - Handle singular/plural forms
- **Date Formatting** - Locale-aware date formatting
- **Number Formatting** - Currency, percentages, decimals
- **Relative Time** - "2 days ago", "in 3 hours"
- **RTL Support** - Right-to-left language support
- **Lazy Loading** - Load translations on demand
- **Namespace Support** - Organize translations by feature
- **Type Safety** - Full TypeScript support with autocomplete
- **SSR Compatible** - Works with server-side rendering
- **Fallback Locales** - Graceful fallback for missing translations

## Hooks

Hook	Description
useTranslation	Access translation function
useLocale	Formatting utilities
useDirection	Get text direction (ltr/rtl)
useLanguage	Current language and switcher

## Formatting

```

const { formatDate, formatNumber, formatRelative } = useLocale();

formatDate(date, { dateStyle: 'full' });
formatNumber(1000, { style: 'currency', currency: 'EUR' });
formatRelative(pastDate); // "2 days ago"

```

## CLI Tool

```

Extract translation keys from code
npx philjs-i18n extract ./src

Validate translation files
npx philjs-i18n validate ./locales

```

## API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

### Entry Points

- Export keys: `./translator`, `./formatter`, `./detector`
- Source files: `packages/philjs-i18n/src/index.ts`

### Public API

- Direct exports: `I18n`, `I18nOptions`, `TranslationMap`, `createI18n`, `createPlural`, `pluralRules`
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

## License

MIT

## @philjs/image - Type: Node package - Purpose: Image optimization for PhilJS with automatic format conversion, responsive sizing, and lazy loading - Version: 0.1.0 - Location: packages/philjs-image - Entry points: packages/philjs-image/src/index.ts, packages/philjs-image/src/vite.ts, packages/philjs-image/src/image-service.ts - Keywords: philjs, image, optimization, webp, avif, responsive, lazy-loading

## PhilJS Image

Advanced image optimization for PhilJS with automatic format conversion, responsive sizing, lazy loading, and blur placeholders.

### Features

- **Automatic Format Conversion:** AVIF, WebP with JPEG/PNG fallbacks
- **Image Services:** Pluggable adapters for Sharp, Cloudinary, imgix
- **Blur Placeholders:** Base64, BlurHash, LQIP, dominant color
- **Aspect Ratio Locking:** Prevent layout shift
- **Art Direction:** Different images for different breakpoints
- **Priority Hints:** Optimize LCP with fetchPriority
- **Loading Animations:** Fade, blur, scale transitions
- **Responsive Images:** Automatic srcset generation
- **Lazy Loading:** Intersection Observer-based
- **Build-Time Optimization:** Generate placeholders and metadata

### Installation

```
npm install philjs-image
```

### Optional Dependencies

```
For Local image processing
npm install sharp

For BlurHash support
npm install blurhash
```

### Quick Start

```
import { Image } from 'philjs-image';

// Basic usage
<Image
 src="/images/hero.jpg"
 alt="Hero image"
 width={1200}
 height={600}
/>

// With modern formats
<Image
 src="/images/product.jpg"
 alt="Product"
 width={800}
 height={600}
 formats={['avif', 'webp', 'jpeg']}
 quality={90}
/>

// Priority image for LCP
<Image
 src="/images/hero-banner.jpg"
 alt="Hero"
 width={1920}
 height={1080}
 priority={true}
 fetchPriority="high"
/>
```

### Core Features

#### 1. Image Service API (Astro-style)

PhilJS supports pluggable image transformation services:

```
import { configureImageService } from 'philjs-image';

// Cloudinary
configureImageService('cloudinary', {
 cloudName: 'your-cloud-name',
});

// imgix
configureImageService('imgix', {
 domain: 'your-domain.imgix.net',
});

// Sharp (local)
configureImageService('sharp');
```

## 2. Blur Placeholder Generation

Generate blur placeholders at build time:

```
import { generateBlurPlaceholder } from 'philjs-image';

// Base64 inline placeholder
const blurDataURL = await generateBlurPlaceholder(imageBuffer, {
 type: 'base64',
 width: 10,
 height: 10,
});

// BlurHash
const blurHash = await generateBlurPlaceholder(imageBuffer, {
 type: 'blurhash',
});

// LQIP (Low Quality Image Placeholder)
const lqip = await generateBlurPlaceholder(imageBuffer, {
 type: 'lqip',
 width: 20,
 height: 20,
});

// Dominant color
const dominantColor = await generateBlurPlaceholder(imageBuffer, {
 type: 'dominant-color',
});
```

## 3. Advanced Image Component

```
import { Image } from 'philjs-image';
import type { ArtDirectionSource } from 'philjs-image';

// Aspect ratio locking
<Image
 src="/product.jpg"
 alt="Product"
 width={400}
 aspectRatio="16:9"
 maintainAspectRatio={true}
/>

// Art direction
const artDirection: ArtDirectionSource[] = [
 {
 media: '(max-width: 640px)',
 src: '/hero-mobile.jpg',
 width: 640,
 },
 {
 media: '(min-width: 641px)',
 src: '/hero-desktop.jpg',
 width: 1920,
 },
];
<Image
 src="/hero-desktop.jpg"
 alt="Hero"
 artDirection={artDirection}
/>

// Loading animations
<Image
 src="/photo.jpg"
 alt="Photo"
 loadingAnimation="fade"
 animationDuration={500}
/>

// Priority hints for LCP
<Image
 src="/hero.jpg"
 alt="Hero"
 priority={true}
 fetchPriority="high"
/>
```

#### API Reference

##### Image Component Props

```

interface ImageProps {
 // Required
 src: string;
 alt: string;

 // Dimensions
 width?: number;
 height?: number;
 aspectRatio?: number | string; // e.g., 16/9 or "16:9"
 maintainAspectRatio?: boolean;

 // Optimization
 quality?: number; // 1-100
 formats?: ImageFormat[]; // ['avif', 'webp', 'jpeg']

 // Responsive
 sizes?: string;
 artDirection?: ArtDirectionSource[];

 // Loading
 loading?: 'lazy' | 'eager' | 'auto';
 priority?: boolean;
 fetchPriority?: 'high' | 'low' | 'auto';

 // Placeholder
 placeholder?: 'blur' | 'color' | 'none';
 blurDataURL?: string;
 blurHash?: string;
 placeholderColor?: string;

 // Animation
 loadingAnimation?: 'fade' | 'blur' | 'scale' | 'none';
 animationDuration?: number;

 // Fit
 fit?: 'cover' | 'contain' | 'fill' | 'inside' | 'outside';

 // Events
 onLoad?: () => void;
 onError?: (error: Error) => void;
}

```

#### Server-Side Functions

```

// Configure optimization
configure({
 formats: ['avif', 'webp', 'jpeg'],
 quality: 85,
 breakpoints: [640, 750, 828, 1080, 1200, 1920],
});

// Optimize image
const buffer = await optimizeImage(input, {
 width: 800,
 height: 600,
 format: 'webp',
 quality: 85,
});

// Get metadata
const metadata = await getMetadata(input);
// { width, height, format, size, aspectRatio, dominantColor }

// Generate responsive set
const set = await generateResponsiveSet(input, {
 formats: ['avif', 'webp', 'jpeg'],
 breakpoints: [640, 750, 828, 1080, 1200, 1920],
});

// Extract dominant color
const color = await extractDominantColor(input);
// "#3366CC"

```

#### Image Services

```

// Configure service
configureImageService('cloudinary', {
 cloudName: 'demo',
});

// Get active service
const service = getImageService();

// Generate URL
const url = service.getUrl('/image.jpg', {
 width: 800,
 height: 600,
 format: 'webp',
 quality: 85,
});

// Custom service
class CustomService implements ImageService {
 name = 'custom';

 getUrl(src: string, options: ImageServiceTransformOptions): string {
 // Custom logic
 }
}

configureImageService(new CustomService());

```

## Examples

See [examples/README.md](#) for comprehensive usage examples including:

- Basic image optimization
- Art direction for responsive images
- BlurHash and LQIP placeholders
- Loading animations
- Product cards and galleries
- Hero banners with priority hints
- Build-time image processing
- Image service integration

## Performance Best Practices

### 1. Use priority for LCP images

```
<Image priority={true} fetchPriority="high" />
```

### 2. Lazy load below-the-fold images

```
<Image loading="lazy" />
```

### 3. Generate placeholders at build time

```
const blurHash = await generateBlurPlaceholder(buffer, { type: 'blurhash' });
<Image blurHash={blurHash} />
```

### 4. Use aspect ratios to prevent layout shift

```
<Image aspectRatio="16:9" maintainAspectRatio={true} />
```

### 5. Choose optimal formats

```
<Image formats={['avif', 'webp', 'jpeg']} />
```

### 6. Use art direction for responsive images

```
<Image artDirection={[
 { media: '(max-width: 640px)', src: '/mobile.jpg' },
 { media: '(min-width: 641px)', src: '/desktop.jpg' },
]} />
```

## Vite Plugin

Add to `vite.config.ts`:

```

import { defineConfig } from 'vite';
import philjs from 'philjs-compiler/vite';
import philjsImage from 'philjs-image/vite';

export default defineConfig({
 plugins: [
 philjs(),
 philjsImage({
 formats: ['avif', 'webp', 'jpeg'],
 quality: 85,
 breakpoints: [640, 750, 828, 1080, 1200, 1920],
 })
]
});

```

#### Browser Support

- Modern formats (AVIF, WebP) with automatic fallback
- Lazy loading (native + polyfill)
- Aspect ratio (CSS aspect-ratio + fallback)
- Priority hints (fetchpriority attribute)

#### API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

##### Entry Points

- Export keys: `./vite`, `./image-service`
- Source files: `packages/philjs-image/src/index.ts`, `packages/philjs-image/src/vite.ts`, `packages/philjs-image/src/image-service.ts`

##### Public API

- Direct exports: `CloudinaryImageService`, `ImageService`, `ImageServiceMetadata`, `ImageServiceRegistry`, `ImageServiceTransformOptions`, `ImgixImageService`, `SharpImageService`, `configureImageService`, `getFormatPriority`, `getImageService`, `imageServiceRegistry`, `selectOptimalFormat`
- Re-exported names: `ArtDirectionSource`, `BlurPlaceholderOptions`, `CloudinaryImageService`, `Image`, `ImageCache`, `ImageFit`, `ImageFormat`, `ImageMetadata`, `ImageOptimizationConfig`, `ImagePosition`, `ImageProps`, `ImageService`, `ImageServiceMetadata`, `ImageServiceRegistry`, `ImageServiceTransformOptions`, `ImageTransformOptions`, `ImgixImageService`, `LoadingAnimation`, `LoadingStrategy`, `OptimizedImage`, `PlaceholderType`, `SharpImageService`, `calculateAspectRatio`, `configure`, `configureImageService`, `createCacheKey`, `default`, `extractDominantColor`, `generateBlurDataURL`, `generateBlurPlaceholder`, `generateResponsiveSet`, `generateSrcSet`, `getConfig`, `getDominantColor`, `getFormatFromSrc`, `getFormatPriority`, `getImageService`, `getMetadata`, `getOptimizedUrl`, `getResponsiveSizes`, `imageServiceRegistry`, `isExternalUrl`, `isSharpAvailable`, `isValidFormat`, `optimizeImage`, `selectOptimalFormat`
- Re-exported modules: `./Image.js`, `./image-service.js`, `./optimizer.js`, `./types.js`, `./utils.js`

##### License

MIT

##### Related Packages

- [philjs-core](#) - Core reactivity system
- [philjs-ssr](#) - Server-side rendering
- [philjs-router](#) - Routing with code splitting

## @philjs/inspector - Type: Node package - Purpose: Visual component inspector for PhilJS - in-page DevTools overlay - Version: 0.1.0 - Location: packages/philjs-inspector - Entry points: `packages/philjs-inspector/src/index.ts`

#### PhilJS Inspector

Visual component inspector for PhilJS applications - an in-page DevTools overlay similar to React DevTools.

##### Features

- **Hover Highlighting** - Highlight components on hover with bounding box
- **Component Names** - Show component name overlay on hover
- **Props Display** - View component props in detailed tooltip
- **Signal Values** - Display signal values used by components
- **Click to Select** - Click to pin component details
- **Hierarchy Breadcrumb** - Show component ancestry path
- **Performance Metrics** - Track render count and timing per component
- **Source Location** - Link to source file:line in your IDE
- **Keyboard Navigation** - Use arrow keys to traverse the component tree
- **Search Components** - Ctrl+F to find components by name

##### Installation

```

npm install philjs-inspector
or
pnpm add philjs-inspector
or
yarn add philjs-inspector

```

##### Usage

###### Basic Setup

The inspector auto-initializes in development environments and attaches to `window.__PHILJS_INSPECTOR__`:

```

import { initInspector } from 'philjs-inspector';

// Initialize inspector (done automatically in dev)
initInspector();

// Enable the inspector
window.__PHILJS_INSPECTOR__.enable();

// Or use the keyboard shortcut: Ctrl+Shift+I

```

#### Programmatic Control

```

// Enable with options
window.__PHILJS_INSPECTOR__.enable({
 showMetrics: true,
 enableKeyboard: true,
});

// Disable inspector
window.__PHILJS_INSPECTOR__.disable();

// Toggle inspector
window.__PHILJS_INSPECTOR__.toggle();

// Check if enabled
const enabled = window.__PHILJS_INSPECTOR__.isEnabled();

// Get inspector state
const state = window.__PHILJS_INSPECTOR__.getState();

// Get all components
const components = window.__PHILJS_INSPECTOR__.getComponents();

```

#### Keyboard Shortcuts

When the inspector is enabled, use these keyboard shortcuts:

Shortcut	Action
Ctrl+Shift+I	Toggle inspector on/off
Ctrl+F	Open component search
Arrow Down	Navigate to next component
Arrow Up	Navigate to previous component
Arrow Left	Navigate to parent component
Arrow Right	Navigate to first child component
Escape	Close tooltip/search or disable inspector

#### Component Search

Press Ctrl+F to open the search box and find components by name:

```

import { showSearchBox } from 'philjs-inspector';

// Show search programmatically
showSearchBox((component) => {
 console.log('Selected component:', component);
});

```

#### Manual Component Tracking

For better debugging, you can manually register components and signals:

```

import { extractComponentInfo, registerSignal } from 'philjs-inspector';

// Extract component info
const element = document.getElementById('my-component');
const info = extractComponentInfo(element);

// Register signals
const count = signal(0);
registerSignal(count, 'count', 'signal');

```

#### IDE Integration

The inspector can open source files in your IDE. Set up the integration:

```

// Add to your app initialization
if (typeof window !== 'undefined') {
 window.__PHILJS_OPEN_IN_IDE__ = (url: string) => {
 // Parse vscode://file/path:line:column URL
 // Send to your dev server to open in IDE
 fetch(`/_open-in-editor?file=${encodeURIComponent(url)})`);
 };
}

```

#### Custom Styling

The inspector uses inline styles to avoid conflicts, but you can override them:

```
import { INSPECTOR_STYLES } from 'philjs-inspector';

// Customize highlight color
const customOverlay = document.getElementById('philjs-inspector-overlay');
if (customOverlay) {
 customOverlay.style.setProperty('--highlight-color', '#ff0000');
}
```

## API Reference

### Inspector API

Attached to `window._PHILJS_INSPECTOR_`:

```
interface InspectorAPI {
 enable(config?: Partial<InspectorConfig>): void;
 disable(): void;
 toggle(): void;
 isEnabled(): boolean;
 getState(): InspectorState;
 getComponents(): ComponentInfo[];
}
```

### Component Info

```
interface ComponentInfo {
 id: string;
 name: string;
 element: Element;
 props: Record<string, any>;
 signals: SignalInfo[];
 isIsland: boolean;
 isHydrated: boolean;
 renderCount: number;
 renderTime: number;
 updateCount: number;
 path: string[];
 source?: SourceLocation;
}
```

### Inspector Config

```
interface InspectorConfig {
 enabled?: boolean;
 showMetrics?: boolean;
 enableKeyboard?: boolean;
 shortcuts?: Partial<InspectorShortcuts>;
}
```

## Integration with Build Tools

### Vite Plugin

Add source location metadata during development:

```
// vite.config.ts
import { defineConfig } from 'vite';

export default defineConfig({
 plugins: [
 {
 name: 'philjs-inspector-plugin',
 transform(code, id) {
 if (id.endsWith('.tsx') || id.endsWith('.jsx')) {
 // Add data-source attributes to components
 // This is a simplified example
 return code.replace(
 /<([A-Z]\w+)>/g,
 `<$1 data-source="${id}:1:1">`);
 }
 },
],
],
});
```

## Browser Support

The inspector works in all modern browsers:

- Chrome/Edge 90+
- Firefox 88+
- Safari 14+

## Performance Considerations

The inspector adds minimal overhead when disabled. When enabled:

- ~1-2ms per interaction (hover/click)
- ~50KB memory for component registry
- No impact on app performance when disabled

## Development

```
Install dependencies
pnpm install

Build
pnpm build

Test
pnpm test

Type check
pnpm typecheck
```

## Examples

### Inspecting Islands

The inspector automatically detects and highlights PhilJS islands:

```
// Your component with islands
function App() {
 return (
 <div>
 <StaticContent />
 <InteractiveWidget island /> {/* Will show [island] badge */}
 </div>
);
}
```

### Tracking Performance

The inspector shows render metrics for each component:

```
import { updateComponentMetrics } from 'philjs-inspector';

function MyComponent() {
 const start = performance.now();

 // Your render logic

 const renderTime = performance.now() - start;
 updateComponentMetrics(element, renderTime);

 return <div>...</div>;
}
```

### Custom Component Names

Set custom names for better debugging:

```
function MyWidget(props) {
 return (
 <div data-component-name="MyWidget">
 {/* Component content */}
 </div>
);
}
```

## Troubleshooting

### Inspector not showing

1. Check that you're in development mode
2. Ensure the inspector is enabled: `window.__PHILJS_INSPECTOR__.enable()`
3. Check browser console for errors

### Components not detected

1. Make sure elements have proper attributes
2. Try manually extracting component info
3. Check that the component is in the DOM

### Source links not working

1. Set up `window.__PHILJS_OPEN_IN_IDE__` handler
2. Configure your dev server to handle file opening
3. Ensure source maps are enabled

## API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

### Entry Points

- Export keys: ..

- Source files: packages/philjs-inspector/src/index.ts

#### Public API

- Direct exports: (none detected)
- Re-exported names: ComponentInfo, ComponentNavigator, HighlightOptions, INSPECTOR\_STYLES, InspectorConfig, InspectorShortcuts, InspectorState, KeyboardHandler, KeyboardShortcut, SignalInfo, SourceLocation, applyStyles, clearAllShortcuts, clearComponentRegistry, destroyOverlay, extractComponentInfo, filterComponents, flashHighlight, formatPropValue, formatShortcut, getAllComponents, getAllShortcuts, getComponentAncestors, getComponentByElement, getComponentById, getComponentChildren, getCurrentHighlightedElement, getCurrentTooltipComponent, getInspector, getOverlayRoot, getSearchStats, getSignalInfo, hideSearchBox, hideTooltip, highlightElement, highlightElementHover, initInspector, initOverlay, isSearchBoxVisible, isTooltipVisible, matchesShortcut, registerShortcut, registerSignal, removeHighlight, removeHoverHighlight, searchComponents, showElementMeasurements, showSearchBox, showTooltip, startKeyboardListening, stopKeyboardListening, styleToCss, unregisterShortcut, updateComponentMetrics, updateHighlightPosition, updateTooltip
- Re-exported modules: ./component-info.js, ./inspector.js, ./keyboard.js, ./overlay.js, ./search.js, ./styles.js, ./tooltip.js

#### License

MIT

#### Contributing

Contributions are welcome! Please read the contributing guidelines first.

#### Related Packages

- [philjs-devtools](#) - Development tools overlay
- [philjs-core](#) - Core PhilJS framework

## @philjs/intent - Type: Node package - Purpose: Intent-based development for PhilJS - natural language to working components - Version: 0.1.0 - Location: packages/philjs-intent - Entry points: packages/philjs-intent/src/index.ts - Keywords: philjs, intent, ai, natural-language, code-generation, declarative, low-code

#### @philjs/intent

Intent-based development for PhilJS - natural language to working components

#### Overview

Intent-based development for PhilJS - natural language to working components

#### Focus Areas

- philjs, intent, ai, natural-language, code-generation, declarative, low-code

#### Entry Points

- packages/philjs-intent/src/index.ts

#### Quick Start

```
import { AlternativeImplementation, Constraint, Counter } from '@philjs/intent';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

#### Exports at a Glance

- AlternativeImplementation
- Constraint
- Counter
- DataFetcher
- Intent
- IntentConfig
- IntentContext
- IntentResolver
- IntentTemplate
- Modal
- ResolvedIntent
- builtinTemplates

#### Install

```
pnpm add @philjs/intent
```

#### Usage

```
import { AlternativeImplementation, Constraint, Counter } from '@philjs/intent';
```

#### Scripts

- pnpm run build
- pnpm run test

#### Compatibility

- Node >=24
- TypeScript 6

#### API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

#### Entry Points

- Export keys: .
- Source files: packages/philijs-intent/src/index.ts

#### Public API

- Direct exports: AlternativeImplementation, Constraint, Counter, DataFetcher, Intent, IntentConfig, IntentContext, IntentResolver, IntentTemplate, Modal, ResolvedIntent, builtinTemplates, getIntentResolver, initIntent, intent, useApiData, useIntent, useModal
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

#### License

MIT

## @philijs/islands - Type: Node package - Purpose: Islands architecture with selective hydration for PhilJS - pure web components - Version: 0.1.0 - Location: packages/philijs-islands - Entry points: packages/philijs-islands/src/index.ts - Keywords: philijs, islands, islands-architecture, selective-hydration, web-components, partial-hydration, progressive-enhancement, lazy-loading - Book coverage: [islands/quick-start.md](#), [islands/multi-framework.md](#)

### philijs-islands

Islands architecture with **multi-framework support**, selective hydration, and server-side component caching for PhilJS.

#### Features

- **Multi-Framework Islands:** Use React, Vue, Svelte, Preact, and Solid components on the same page (Astro-style)
- **Selective Hydration:** Only hydrate interactive components when needed
- **Multiple Strategies:** Immediate, visible, idle, interaction, and media-based hydration
- **Framework Auto-Detection:** Automatically detect component frameworks
- **Shared State:** Cross-framework state management with event bus
- **Code Splitting:** Automatic framework-specific code splitting via Vite plugin
- **Server Islands:** Cache server-rendered components independently
- **TypeScript:** Full type safety across all frameworks

#### Installation

```
pnpm add philijs-islands
```

#### Framework Dependencies (Optional)

Install only the frameworks you plan to use:

```
React
pnpm add react react-dom

Vue
pnpm add vue

Svelte
pnpm add svelte

Preact
pnpm add preact

Solid
pnpm add solid-js
```

#### Usage

##### Basic Island Hydration

Islands allow you to selectively hydrate interactive components on an otherwise static page:

```
import { mountIslands, hydrateIsland } from 'philijs-islands';

// Automatically mount all islands on page Load
mountIslands(document.body);

// Or manually hydrate a specific island
const islandElement = document.querySelector('[island="Counter"]');
hydrateIsland(islandElement);
```

##### Defining Islands

Mark components as islands in your HTML:

```

function Counter({ initial = 0 }) {
 const count = signal(initial);

 return (
 <div island="Counter" data-prop-initial={initial}>
 <p>Count: {count}</p>
 <button onClick={() => count.set(count() + 1)}>
 Increment
 </button>
 </div>
);
}

```

#### Island Component Loader

Register and lazy-load island components:

```

import { registerIsland, loadIsland, initIslands } from 'philjs-islands';

// Register island Loaders
registerIsland('Counter', () => import('./islands/Counter'));
registerIsland('SearchBox', () => import('./islands/SearchBox'));
registerIsland('VideoPlayer', () => import('./islands/VideoPlayer'));

// Define island manifest
const manifest = {
 Counter: {
 import: './islands/Counter.tsx',
 trigger: 'visible', // 'visible' | 'idle' | 'immediate'
 },
 SearchBox: {
 import: './islands/SearchBox.tsx',
 trigger: 'immediate',
 },
 VideoPlayer: {
 import: './islands/VideoPlayer.tsx',
 trigger: 'visible',
 props: { autoplay: false },
 },
};

// Initialize all islands with hydration strategies
initIslands(manifest);

```

#### SSR Island Wrapper

Use the `Island` component during SSR to mark hydration boundaries:

```

import { Island } from 'philjs-islands';

function BlogPost({ content }) {
 return (
 <article>
 <h1>{content.title}</h1>
 <div>{content.body}</div>

 {/* Only this component will hydrate on the client */}
 <Island name="CommentSection" trigger="visible" props={{ postId: content.id }}>
 <CommentSection postId={content.id} />
 </Island>
 </article>
);
}

```

#### Server Islands (2026 Feature)

Server Islands allow you to cache server-rendered components independently:

```

import {
 ServerIsland,
 renderServerIsland,
 cacheIsland,
 invalidateIsland,
} from 'philjs-islands';

// Define a server island with caching
async function ProductRecommendations({ userId }) {
 const recommendations = await getRecommendations(userId);

 return (
 <ServerIsland
 id={'recommendations-${userId}'}
 ttl={3600} // Cache for 1 hour
 tags={['recommendations', `user:${userId}`]}
 >
 <div>
 {recommendations.map(product => (
 <ProductCard key={product.id} product={product} />
))}
 </div>
 </ServerIsland>
);
}

// Render server island on the server
const html = await renderServerIsland('recommendations-123', async () => {
 return <ProductRecommendations userId="123" />;
}, {
 ttl: 3600,
 tags: ['recommendations', 'user:123'],
});

// Invalidate cache when data changes
await invalidateIsland('recommendations-123');

// Or invalidate by tag
await invalidateIslandsByTag('recommendations');

```

#### Custom Cache Store

Use Redis or other storage backends for server islands:

```

import { setIslandCacheStore, createRedisCacheAdapter } from 'philjs-islands';

// Redis adapter
const redisCache = createRedisCacheAdapter({
 host: 'localhost',
 port: 6379,
});

setIslandCacheStore(redisCache);

// Or Cloudflare KV
const kvCache = createKVCacheAdapter({
 namespace: env.ISLAND_CACHE,
});

setIslandCacheStore(kvCache);

```

#### Island Prefetching

Prefetch islands before they become visible:

```

import { prefetchIsland } from 'philjs-islands';

// Prefetch island data
await prefetchIsland('recommendations-123', {
 priority: 'high',
});

// Use in component
function ProductPage({ productId }) {
 useEffect(() => {
 // Prefetch related islands
 prefetchIsland(`reviews-${productId}`);
 prefetchIsland(`similar-products-${productId}`);
 }, [productId]);

 return <div>...</div>;
}

```

#### Monitoring & Metrics

Track island cache performance:

```

import { getServerIslandMetrics, resetServerIslandMetrics } from 'philjs-islands';

// Get cache metrics
const metrics = getServerIslandMetrics();
console.log('Hit rate:', metrics.hitRate);
console.log('Total hits:', metrics.hits);
console.log('Total misses:', metrics.misses);
console.log('Average render time:', metrics.avgRenderTime);

// Reset metrics
resetServerIslandMetrics();

```

## API

### Client-Side Hydration

- `mountIslands(root?)` - Automatically mount all islands in the DOM
- `hydrateIsland(element)` - Manually hydrate a specific island element

### Island Registration

- `registerIsland(name, loader)` - Register a lazy-loadable island component
- `loadIsland(element, manifest)` - Load and hydrate an island from manifest
- `initIslands(manifest)` - Initialize all islands with hydration strategies

### SSR Components

- `<Island name="..." trigger="..." props={...}>` - Mark hydration boundary during SSR

### Server Islands

- `<ServerIsland id="..." ttl={...} tags={[...]}>` - Server-rendered cacheable component
- `renderServerIsland(id, render, options)` - Render server island with caching
- `cacheIsland(id, html, options)` - Manually cache island HTML
- `invalidateIsland(id)` - Invalidate specific island cache
- `invalidateIslandsByTag(tag)` - Invalidate all islands with a tag
- `clearIslandCache()` - Clear entire island cache
- `prefetchIsland(id, options)` - Prefetch island data

### Cache Stores

- `setIslandCacheStore(store)` - Set custom cache backend
- `getIslandCacheStore()` - Get current cache store
- `createRedisCacheAdapter(config)` - Create Redis cache adapter
- `createKVCacheAdapter(config)` - Create KV store adapter (Cloudflare)

### Monitoring

- `getServerIslandMetrics()` - Get cache performance metrics
- `resetServerIslandMetrics()` - Reset metrics counters
- `getIslandCacheHeaders(id)` - Get HTTP cache headers for island

### Multi-Framework Islands

For complete multi-framework documentation, see [MULTI-FRAMEWORK.md](#).

### Quick Example

```

import { MultiFrameworkIsland, registerIslandComponent } from 'philjs-islands';

// Register components from different frameworks
registerIslandComponent('react', 'Counter', () => import('./islands/Counter.tsx'));
registerIslandComponent('vue', 'TodoList', () => import('./islands/TodoList.vue'));
registerIslandComponent('svelte', 'Timer', () => import('./islands/Timer.svelte'));

function App() {
 return (
 <div>
 {/* React Island */}
 <MultiFrameworkIsland
 framework="react"
 component="Counter"
 props={{ initial: 0 }}
 hydration={{ strategy: 'visible' }}>
 </MultiFrameworkIsland>
 {/* Vue Island */}
 <MultiFrameworkIsland
 framework="vue"
 component="TodoList"
 props={{ items: [] }}
 hydration={{ strategy: 'idle' }}>
 </MultiFrameworkIsland>
 {/* Svelte Island */}
 <MultiFrameworkIsland
 framework="svelte"
 component="Timer"
 props={{ autoStart: true }}
 hydration={{ strategy: 'immediate' }}>
 </MultiFrameworkIsland>
 </div>
);
}

```

#### Shared State Across Frameworks

```

import { createSharedState, EventBus } from 'philjs-islands';

// Create shared state accessible to all frameworks
const globalState = createSharedState('app', {
 user: { name: 'Guest' },
 theme: 'light'
});

// Any island can update it
globalState.updateState({ theme: 'dark' });

// Cross-framework events
EventBus.emit('user-logged-in', { userId: 123 });
EventBus.on('user-logged-in', (data) => {
 console.log('User:', data.userId);
});

```

#### Examples

See islands in action in these example apps:

- [Multi-Framework Demo](#) - React, Vue, and Svelte on one page
- [Demo App](#) - Full-featured demo with islands, SSR, and routing

#### Development

```

Build the package
npm build

Run tests
pnpm test

Type checking
pnpm typecheck

```

#### Core Features

##### Multi-Framework Support

- **React** - Full React 18+ support with concurrent features
- **Vue** - Vue 3 Composition API and Vue 2 compatibility
- **Svelte** - Svelte 4 and 5 support
- **Preact** - Lightweight React alternative
- **Solid** - Fine-grained reactivity
- **Auto-detection** - Automatically detect component frameworks
- **Custom adapters** - Add support for any framework

##### Selective Hydration

- **Immediate** - Hydrate right away for critical components
- **Visible** - Hydrate when scrolled into viewport (IntersectionObserver)
- **Idle** - Hydrate when browser is idle (requestIdleCallback)
- **Interaction** - Hydrate on user interaction (click, focus, etc.)
- **Media** - Hydrate based on media queries
- **Manual** - Full control over hydration timing

#### Cross-Framework Features

- **Shared State** - State management across different frameworks
- **Event Bus** - Publish/subscribe events between islands
- **Props Normalization** - Automatic props conversion between frameworks
- **Island Bridges** - Typed communication channels

#### Build Optimization

- **Vite Plugin** - Automatic framework detection and code splitting
- **Code Splitting** - Separate bundles per framework
- **Tree Shaking** - Only bundle frameworks you use
- **Manifest Generation** - Build-time optimization metadata

#### Server Islands

- **Cache server-rendered components** - Independent caching per island
- **Tag-based invalidation** - Invalidate related islands together
- **Custom backends** - Redis, KV, or any cache store
- **Metrics** - Track cache hit rates and performance

#### Developer Experience

- **TypeScript** - Full type safety across all frameworks
- **Zero overhead** - Static content ships zero JavaScript
- **Hot Module Replacement** - Fast development with HMR
- **Debug Tools** - Monitor islands and state in development

#### Island Manifest

When using `initIslands()`, provide a manifest defining each island:

```
type IslandManifest = {
 [name: string]: {
 import: string; // Path to island component
 props?: Record<string, any>; // Default props
 trigger?: 'visible' | 'idle' | 'immediate';
 };
};
```

#### Hydration Triggers

- **visible** - Hydrate when island scrolls into viewport (uses IntersectionObserver)
- **idle** - Hydrate when browser is idle (uses requestIdleCallback)
- **immediate** - Hydrate immediately on page load

#### Server Island Options

```
type ServerIslandCache = {
 id: string; // Unique cache key
 ttl?: number; // Time to Live in seconds
 tags: string[]; // Tags for batch invalidation
 staleWhileRevalidate?: number; // Serve stale while refreshing
};
```

#### Events

Islands dispatch custom events you can listen to:

```
// Island hydrated
element.addEventListener('phil:island-hydrated', (e) => {
 console.log('Island hydrated:', e.detail.name);
});

// Island Loaded (component code fetched)
element.addEventListener('phil:island-loaded', (e) => {
 console.log('Island loaded:', e.detail.name);
});
```

#### Best Practices

1. **Use islands for interactivity** - Keep static content static, only hydrate what needs JavaScript
2. **Lazy load by default** - Use `visible` or `idle` triggers unless immediate interaction is needed
3. **Cache server islands** - Expensive server-rendered components benefit from caching
4. **Tag your caches** - Use tags for easy invalidation (e.g., `['user:123', 'posts']`)
5. **Monitor metrics** - Track cache hit rates to optimize TTLs

## 6. Prefetch strategically - Warm caches for critical paths

### API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

#### Entry Points

- Export keys: .
- Source files: packages/philjs-islands/src/index.ts

#### Public API

- Direct exports: (none detected)
- Re-exported names: HydrationStrategy, HydrationTrigger, Island, IslandComponent, IslandConfig, IslandInstance, IslandLifecycle, IslandLoader, IslandRegistry, IslandState, LoaderConfig, RegistryEntry, createsIsland, createsIslandLoader, definesIsland, getRegistry, hydrateAll, hydrateIsland, mountIslands
- Re-exported modules: ./hydration.js, ./island.js, ./loader.js, ./registry.js, ./types.js

#### License

MIT

## @philjs/jobs - Type: Node package - Purpose: Background job processing and scheduling for PhilJS - Version: 0.1.0 - Location: packages/philjs-jobs - Entry points: packages/philjs-jobs/src/index.ts, packages/philjs-jobs/src/queue.ts, packages/philjs-jobs/src/scheduler.ts, packages/philjs-jobs/src/monitor.ts, packages/philjs-jobs/src/job.ts - Keywords: philjs, jobs, background-jobs, queue, scheduler, cron, bullmq, redis, task-queue

### @philjs/jobs

**Background job processing and scheduling for PhilJS** - RedwoodJS-style job queue system with Redis support, cron scheduling, and comprehensive monitoring.

#### Requirements

- **Node.js 24** or higher
- **TypeScript 6** or higher
- **ESM only** - CommonJS is not supported

#### Features

- **Flexible Job Queue:** Redis-backed (BullMQ) or in-memory queue for development
- **Type-Safe Job Definitions:** Define jobs with full TypeScript type safety
- **Cron Scheduling:** Schedule recurring jobs with timezone support
- **Job Lifecycle Hooks:** Execute code at different stages of job processing
- **Middleware System:** Add validation, logging, rate limiting, and custom logic
- **Job Monitoring:** Track metrics, health status, and generate dashboards
- **Retry Logic:** Automatic retries with exponential backoff
- **Progress Tracking:** Update and monitor job progress in real-time
- **Priority Queues:** Process high-priority jobs first
- **Concurrency Control:** Limit concurrent job execution
- **Job History:** Track scheduled job execution history

#### Installation

```
pnpm add @philjs/jobs
```

For Redis support (optional):

```
pnpm add bullmq ioredis
```

#### Quick Start

```

import { defineJob, createQueue } from '@philjs/jobs';

// Define a job
const sendEmailJob = defineJob({
 name: 'send-email',
 handler: async (payload: { to: string; subject: string }, ctx) => {
 // Send email
 await ctx.updateProgress(50);
 await sendEmail(payload);
 await ctx.updateProgress(100);

 return { sent: true };
 },
 attempts: 3,
 backoff: { type: 'exponential', delay: 1000 },
});

// Create queue
const queue = createQueue({ concurrency: 5 });

// Enqueue job
await queue.enqueue(sendEmailJob, {
 to: 'user@example.com',
 subject: 'Welcome!',
});

// Process jobs
await queue.process();

```

## Core Concepts

### 1. Job Definitions

Define type-safe jobs with handlers, options, and hooks:

```

const imageProcessingJob = defineJob({
 name: 'image-processing',
 handler: async (payload: { imageUrl: string }, ctx) => {
 ctx.log('Processing image...');

 // Update progress
 await ctx.updateProgress(25);

 // Process image
 const result = await processImage(payload.imageUrl);

 await ctx.updateProgress(100);

 return result;
 },

 // Job options
 attempts: 3,
 backoff: { type: 'exponential', delay: 2000 },
 priority: 10,
 timeout: 60000,

 // Lifecycle hooks
 onBefore: async (payload, ctx) => {
 console.log(`Job starting: ${ctx.jobId}`);
 },
 onComplete: async (result, ctx) => {
 console.log(`Job completed: ${result}`);
 },
 onFail: async (error, ctx) => {
 console.error(`Job failed: ${error}`);
 },
 onProgress: async (progress, ctx) => {
 console.log(`Progress: ${progress}%`);
 },
});

```

### 2. Job Queue

Create and manage job queues:

```

// In-memory queue (development)
const devQueue = createQueue({
 concurrency: 3,
});

// Redis queue (production)
const prodQueue = createQueue({
 name: 'my-app-jobs',
 redis: {
 host: 'localhost',
 port: 6379,
 password: 'secret',
 },
 concurrency: 10,
 defaultJobOptions: {
 attempts: 3,
 removeOnComplete: 100,
 },
});

// Enqueue jobs
const job = await queue.enqueue(myJob, payload, {
 priority: 5,
 delay: 5000, // Delay 5 seconds
});

// Start processing
await queue.process();

// Get statistics
const stats = await queue.getStats();
console.log(stats);

// Pause/resume
await queue.pause();
await queue.resume();

// Stop queue
await queue.stop();

```

### 3. Job Scheduling

Schedule recurring jobs with cron expressions:

```

import { Scheduler, CronPatterns, Timezones } from '@philjs/jobs';

const scheduler = new Scheduler(queue);

// Schedule daily backup at 2 AM
scheduler.schedule(backUpJob, {
 cron: CronPatterns.at(2, 0),
 timezone: Timezones.UTC,
 payload: { database: 'production' },
 onSchedule: (scheduledAt) => {
 console.log(`Backup scheduled for: ${scheduledAt}`);
 },
});

// Schedule weekly cleanup
scheduler.schedule(cleanUpJob, {
 cron: CronPatterns.WEEKLY,
 payload: { olderThan: 30 },
 maxRuns: 10, // Only run 10 times
});

// Schedule one-time job
scheduler.scheduleOnce(
 reportJob,
 { reportType: 'monthly' },
 new Date('2024-02-01T00:00:00Z')
);

// Start scheduler
scheduler.start();

// Manage schedules
scheduler.pause(scheduleId);
scheduler.resume(scheduleId);
scheduler.remove(scheduleId);

// Get scheduled jobs
const jobs = scheduler.getAllScheduledJobs();
const history = scheduler.getJobHistory();

```

### 4. Job Middleware

Add custom logic with middleware:

```
import {
 createValidationMiddleware,
 createLoggingMiddleware,
 createRateLimitMiddleware,
 composeMiddleware,
} from '@philjs/jobs';

const processPaymentJob = defineJob({
 name: 'process-payment',
 handler: async (payload, ctx) => {
 // Process payment
 return { success: true };
 },
 middleware: [
 // Validate payload
 createValidationMiddleware(
 (payload) => payload.amount > 0 && payload.currency,
 'Invalid payment data'
),
 // Log execution
 createLoggingMiddleware(),
 // Rate limit
 createRateLimitMiddleware(100), // Max 100 per minute
 // Custom middleware
 async (payload, ctx, next) => {
 // Pre-processing
 const result = await next();
 // Post-processing
 return result;
 },
],
});

// Compose multiple middleware
const composedMiddleware = composeMiddleware(
 middleware1,
 middleware2,
 middleware3
);
```

## 5. Job Monitoring

Monitor jobs and track metrics:

```

import { Monitor } from '@philjs/jobs';

const monitor = new Monitor(queue, scheduler, {
 metricsInterval: 60000, // Collect every minute
 healthCheckInterval: 30000, // Check every 30 seconds
 healthThresholds: {
 queueDepthWarning: 100,
 errorRateWarning: 0.1,
 },
});

// Start monitoring
monitor.start();

// Get current metrics
const metrics = await monitor.getMetrics();
console.log(metrics);
// {
// totalJobs: 1000,
// completedJobs: 950,
// failedJobs: 50,
// successRate: 0.95,
// throughput: 16.7,
// ...
// }

// Check health
const health = await monitor.getHealth();
console.log(health.status); // 'healthy' | 'degraded' | 'critical'
console.log(health.issues); // Array of issues

// Get job details
const jobDetails = await monitor.getJobDetails(jobId);

// Retry failed jobs
await monitor.retryJob(jobId);
await monitor.retryAllFailed();

// Export metrics
const json = await monitor.exportMetrics();

// Generate HTML dashboard
const html = await monitor.generateDashboard();

```

## Cron Patterns

Common cron patterns are available:

```

import { CronPatterns } from '@philjs/jobs';

CronPatterns.EVERY_MINUTE // * * * * *
CronPatterns.EVERY_5_MINUTES // */5 * * * *
CronPatterns.EVERY_15_MINUTES // */15 * * * *
CronPatterns.EVERY_30_MINUTES // */30 * * * *
CronPatterns.EVERY_HOUR // * 0 * * *
CronPatterns.DAILY // 0 0 * * *
CronPatterns.DAILY_NOON // 0 12 * * *
CronPatterns.WEEKLY // 0 0 * * 0
CronPatterns.MONTHLY // 0 0 1 * *
CronPatterns.YEARLY // 0 0 1 1 *

// Custom patterns
CronPatterns.at(14, 30) // '30 14 * * * - 2:30 PM'
CronPatterns.everyMinutes(15) // */15 * * * *
CronPatterns.everyHours(6) // * 0 */6 * * *
CronPatterns.weekday(1, 9, 0) // '0 9 * * 1' - Monday at 9 AM
CronPatterns.weekdays(9, 0) // '0 9 * * 1-5' - Weekdays at 9 AM
CronPatterns.weekends(10, 0) // '0 10 * * 0,6' - Weekends at 10 AM

```

## Timezones

Common timezone identifiers:

```

import { Timezones } from '@philjs/jobs';

Timezones.UTC
Timezones.NEW_YORK // 'America/New_York'
Timezones.CHICAGO // 'America/Chicago'
Timezones.DENVER // 'America/Denver'
Timezones.LOS_ANGELES // 'America/Los_Angeles'
Timezones.LONDON // 'Europe/London'
Timezones.PARIS // 'Europe/Paris'
Timezones.TOKYO // 'Asia/Tokyo'
Timezones.SHANGHAI // 'Asia/Shanghai'
Timezones.SYDNEY // 'Australia/Sydney'

```

## Advanced Examples

#### Video Transcoding Pipeline

```
const videoTranscodingJob = defineJob({
 name: 'video-transcoding',
 handler: async (payload: { videoId: string; format: string }, ctx) => {
 await ctx.updateProgress(10);

 // Download video
 const video = await downloadVideo(payload.videoId);
 await ctx.updateProgress(30);

 // Transcode
 const transcoded = await transcode(video, payload.format);
 await ctx.updateProgress(70);

 // Upload
 const url = await upload(transcoded);
 await ctx.updateProgress(100);

 return { url };
 },
 timeout: 600000, // 10 minutes
 attempts: 2,
 priority: 8,
});

// Enqueue with high priority
await queue.enqueue(videoTranscodingJob,
 { videoId: 'vid_123', format: 'mp4' },
 { priority: 10 }
);
```

#### Email Campaign

```
const emailCampaignJob = defineJob({
 name: 'email-campaign',
 handler: async (payload: { recipients: string[]; template: string }, ctx) => {
 const total = payload.recipients.length;

 for (let i = 0; i < total; i++) {
 await sendEmail(payload.recipients[i], payload.template);
 await ctx.updateProgress((i + 1) / total * 100);
 }

 return { sent: total };
 },
 middleware: [
 createRateLimitMiddleware(1000), // Max 1000 emails per minute
 createLoggingMiddleware(),
],
});
```

#### Data Synchronization

```
// Schedule hourly sync
scheduler.schedule(syncJob, {
 cron: CronPatterns.EVERY_HOUR,
 timezone: Timezones.UTC,
 payload: { source: 'external-api' },
 onComplete: async () => {
 console.log('Sync completed successfully');
 },
});

// Schedule nightly backup
scheduler.schedule(backupJob, {
 cron: CronPatterns.at(3, 0), // 3 AM
 timezone: Timezones.NEW_YORK,
 payload: { database: 'production' },
 endDate: new Date('2025-12-31'), // Stop at end of year
});
```

#### API Reference

`defineJob(options)`  
Define a type-safe job.

**Parameters:** - `name` (string): Unique job name - `handler` (function): Job handler function - `attempts` (number): Max retry attempts (default: 3) - `backoff` (object): Retry backoff strategy - `priority` (number): Job priority (default: 0) - `timeout` (number): Job timeout in milliseconds - `concurrency` (number): Max concurrent executions - `middleware` (array): Middleware functions - `onBefore`, `onComplete`, `onFail`, `onProgress`, `onFinally`: Lifecycle hooks

`createQueue(options)`  
Create a job queue.

**Parameters:** - `name` (string): Queue name - `redis` (object): Redis connection config (optional) - `concurrency` (number): Worker concurrency (default: 1) - `defaultJobOptions` (object): Default job options

**Methods:** - `enqueue(job, payload, options)`: Add job to queue - `process()`: Start processing jobs - `stop()`: Stop processing - `pause()`: Pause queue - `resume()`: Resume queue - `getJob(jobId)`: Get job by ID - `getStats()`: Get queue statistics - `removeJob(jobId)`: Remove a job - `retryJob(jobId)`: Retry a failed job - `clear()`: Clear all jobs

#### Scheduler

Schedule recurring and one-time jobs.

- Methods:**
- `schedule(job, options)`: Schedule recurring job
  - `scheduleOnce(job, payload, date)`: Schedule one-time job
  - `start()`: Start scheduler
  - `stop()`: Stop scheduler
  - `pause(scheduleId)`: Pause a schedule
  - `resume(scheduleId)`: Resume a schedule
  - `remove(scheduleId)`: Remove a schedule
  - `getScheduledJob(scheduleId)`: Get schedule by ID
  - `getAllScheduledJobs()`: Get all schedules
  - `getJobHistory(scheduleId?)`: Get execution history

#### Monitor

Monitor jobs and collect metrics.

- Methods:**
- `start()`: Start monitoring
  - `stop()`: Stop monitoring
  - `getMetrics()`: Get current metrics
  - `getMetricsHistory()`: Get metrics history
  - `getHealth()`: Get system health
  - `getJobDetails(jobId)`: Get job details
  - `retryJob(jobId)`: Retry a failed job
  - `retryAllFailed()`: Retry all failed jobs
  - `exportMetrics()`: Export metrics as JSON
  - `generateDashboard()`: Generate HTML dashboard

#### Environment Variables

For Redis configuration:

```
REDIS_HOST=localhost
REDIS_PORT=6379
REDIS_PASSWORD=secret
REDIS_DB=0
```

#### Best Practices

1. **Use Redis in Production:** In-memory queue is for development only
2. **Set Appropriate Timeouts:** Prevent jobs from hanging indefinitely
3. **Monitor Your Jobs:** Use the monitoring system to track health
4. **Handle Errors Gracefully:** Use lifecycle hooks to log and recover
5. **Use Middleware:** Add validation, logging, and rate limiting
6. **Set Job Priorities:** Process critical jobs first
7. **Clean Up Completed Jobs:** Use `removeOnComplete` to prevent memory issues
8. **Use Exponential Backoff:** For better retry behavior
9. **Track Progress:** Update progress for long-running jobs
10. **Test Locally:** Use in-memory queue for testing

#### Testing

```
npm test
```

Run with coverage:

```
npm test -- --coverage
```

#### Examples

See the `examples/` directory for complete examples:

- `basic-usage.ts` - Basic job queue usage
- `scheduling.ts` - Cron scheduling
- `middleware.ts` - Middleware patterns
- `monitoring.ts` - Job monitoring and dashboards
- `redis-backend.ts` - Production Redis setup

#### API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

##### Entry Points

- Export keys: `./queue`, `./scheduler`, `./monitor`, `./job`
- Source files: `packages/philst-jobs/src/index.ts`, `packages/philst-jobs/src/queue.ts`, `packages/philst-jobs/src/scheduler.ts`, `packages/philst-jobs/src/monitor.ts`, `packages/philst-jobs/src/job.ts`

##### Public API

- Direct exports: `CronPatterns`, `DefineJobOptions`, `EnqueueOptions`, `IQueue`, `InMemoryQueue`, `Job`, `JobContext`, `JobDefinition`, `JobDetail`, `JobHandler`, `JobHooks`, `JobMetrics`, `JobMiddleware`, `JobOptions`, `JobRun`, `MetricsSnapshot`, `Monitor`, `MonitorOptions`, `QueueOptions`, `QueueStats`, `RedisQueue`, `ScheduleOptions`, `ScheduledJob`, `Scheduler`, `SystemHealth`, `Timezones`, `composeMiddleware`, `createLoggingMiddleware`, `createQueue`, `createRateLimitMiddleware`, `createRetryMiddleware`, `createValidationMiddleware`, `defineJob`
- Re-exported names: `CronPatterns`, `DefineJobOptions`, `EnqueueOptions`, `IQueue`, `InMemoryQueue`, `Job`, `JobContext`, `JobDefinition`, `JobDetail`, `JobHandler`, `JobHooks`, `JobMetrics`, `JobMiddleware`, `JobMonitor`, `JobOptions`, `JobRun`, `JobScheduler`, `MetricsSnapshot`, `Monitor`, `MonitorOptions`, `QueueOptions`, `QueueStats`, `RedisQueue`, `ScheduleOptions`, `ScheduledJob`, `Scheduler`, `SystemHealth`, `Timezones`, `composeMiddleware`, `createJobQueue`, `createLoggingMiddleware`, `createQueue`, `createRateLimitMiddleware`, `createRetryMiddleware`, `createValidationMiddleware`, `defineJob`
- Re-exported modules: `./job.js`, `./monitor.js`, `./queue.js`, `./scheduler.js`

##### License

MIT

##### Contributing

Contributions are welcome! Please read our contributing guidelines.

##### Support

- Documentation: <https://philst.dev/docs/jobs>

- Issues: <https://github.com/philjs/philjs/issues>
- Discord: <https://discord.gg/philjs>

## @philjs/kotlin - Type: Node package - Purpose: Kotlin bindings for PhilJS - Android native and Kotlin/JS development - Version: 0.1.0 - Location: packages/philjs-kotlin - Entry points: packages/philjs-kotlin/src/index.ts, ./android, ./compose - Keywords: philjs, kotlin, android, compose, jetpack, native, kmp, multiplatform

## philjs-kotlin

Kotlin bindings for PhilJS - Android native and Kotlin/JS development.

### Requirements

- **Node.js 24** or higher
- **TypeScript 6** or higher
- **ESM only** - CommonJS is not supported
- **Kotlin 2.0+** for Kotlin-side features
- **Android Studio** for Android development

### Installation

```
pnpm add @philjs/kotlin
```

### Features

- **Android Integration** - Build Android apps with PhilJS
- **Jetpack Compose** - Reactive UI with Compose interop
- **Kotlin/JS** - Share code between TypeScript and Kotlin
- **KMP Support** - Kotlin Multiplatform compatibility
- **Type Bridge** - Automatic TypeScript <-> Kotlin type conversion

### Quick Start

#### Android Setup

```
import { createAndroidApp } from '@philjs/kotlin/android';

const app = createAndroidApp({
 name: 'MyApp',
 packageName: 'com.example.myapp',
 minSdk: 24,
 targetSdk: 34,
});

// Generate Android project
await app.scaffold('./android');
```

#### Compose Integration

```
import { createComposeBinding } from '@philjs/kotlin/compose';
import { signal } from '@philjs/core';

const count = signal(0);

// Create Compose-compatible state
const composeState = createComposeBinding(count, {
 name: 'counter',
 type: 'Int',
});

// Generated Kotlin code:
// val counter by mutableStateOf(0)
```

#### Kotlin/JS Bridge

```
import { createKotlinBridge } from '@philjs/kotlin';

const bridge = createKotlinBridge({
 module: './kotlin/shared.kt',
 exports: ['calculateSum', 'formatDate', 'UserData'],
});

// Call Kotlin functions from TypeScript
const sum = await bridge.invoke('calculateSum', [1, 2, 3]);
console.log(sum); // 6

// Use Kotlin data classes
const user = await bridge.create('UserData', {
 name: 'John',
 age: 30,
});
```

### ES2024 Features

`Promise.withResolvers()` for Android APIs

```

import { createAndroidBridge } from '@philjs/kotlin/android';

const android = createAndroidBridge();

// Wrap callback-based Android APIs with Promise.withResolvers()
function requestPermission(permission: string) {
 const { promise, resolve, reject } = Promise.withResolvers<boolean>();

 android.requestPermission(permission, {
 onGranted: () => resolve(true),
 onDenied: () => resolve(false),
 onError: (error) => reject(error),
 });
}

return promise;
}

const granted = await requestPermission('android.permission.CAMERA');

```

#### Object.groupBy() for UI Components

```

import { createComposeBinding } from '@philjs/kotlin/compose';

interface Component {
 id: string;
 type: 'text' | 'button' | 'input';
 props: Record<string, unknown>;
}

// Group components by type using ES2024 Object.groupBy()
function organizeComponents(components: Component[]): {
 const grouped = Object.groupBy(components, c => c.type);

 return {
 textComponents: grouped.text ?? [],
 buttonComponents: grouped.button ?? [],
 inputComponents: grouped.input ?? [],
 };
}

```

#### Resource Management with using

```

import { createKotlinBridge } from '@philjs/kotlin';

// Automatic cleanup with TypeScript 6 explicit resource management
async function runKotlinCode() {
 await using bridge = await createKotlinBridge({
 module: './kotlin/process.kt',
 [Symbol.asyncDispose]: async () => {
 await bridge.cleanup();
 console.log('Kotlin bridge disposed');
 }
 });

 // Bridge automatically cleaned up when scope exits
 return bridge.invoke('processData', { input: 'test' });
}

```

#### CLI

```

Initialize Android project
philjs-kotlin init android my-app

Build Kotlin/JS module
philjs-kotlin build ./src/shared.kt

Generate TypeScript types from Kotlin
philjs-kotlin types ./kotlin/models.kt

Run Android emulator
philjs-kotlin android:run

```

#### API Reference

##### Android

Function	Description
createAndroidApp(config)	Create Android app config
createAndroidBridge()	Bridge to Android APIs
scaffold(path)	Generate Android project

##### Compose

Function	Description
<code>createComposeBinding(signal, options)</code>	Bind PhilJS signal to Compose state
<code>generateComposable(component)</code>	Generate Compose UI code

#### Kotlin/JS

Function	Description
<code>createKotlinBridge(config)</code>	Create Kotlin/JS bridge
<code>bridge.invoke(fn, args)</code>	Call Kotlin function
<code>bridge.create(class, props)</code>	Instantiate Kotlin class

#### API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

#### Entry Points

- Export keys: `,`, `./android`, `./compose`
- Source files: `packages/philjs-kotlin/src/index.ts`

#### Public API

- Direct exports: (none detected)
- Re-exported names: (none detected)
- Re-exported modules: `./bridge.js`, `./codegen.js`, `./types.js`

#### License

MIT

## @philjs/liveview - Type: Node package - Purpose: Server-driven UI for PhilJS - Phoenix LiveView-style real-time rendering - Version: 0.1.0 - Location: packages/philjs-liveview - Entry points: packages/philjs-liveview/src/index.ts, packages/philjs-liveview/src/client.ts, packages/philjs-liveview/src/server.ts - Keywords: philjs, liveview, phoenix, server-driven, realtime, websocket, spa, ssr

#### PhilJS LiveView

Server-driven UI for PhilJS - Phoenix LiveView-style real-time rendering with minimal client JavaScript.

#### Features

- **Server-Rendered HTML** - Initial HTML rendered on server, updates via WebSocket
- **DOM Diffing** - Efficient patching with morphdom-style updates
- **Minimal Client JS** - Only ~10KB of client code needed
- **LiveComponents** - Stateful components within views
- **Form Handling** - Built-in validation and file uploads
- **Navigation** - SPA-like navigation without full page reloads
- **Hooks** - Client-side JavaScript interop
- **PubSub** - Real-time broadcasting between views

#### Installation

```
npm install philjs-liveview
```

#### Quick Start

##### Server-Side

```

import { createLiveViewServer, createLiveView } from 'philjs-liveview/server';

// Create a LiveView
const CounterView = createLiveView({
 mount: (socket) => {
 return { count: 0 };
 },
 handleEvent: (event, state, socket) => {
 switch (event.type) {
 case 'increment':
 return { count: state.count + 1 };
 case 'decrement':
 return { count: state.count - 1 };
 default:
 return state;
 }
 },
 render: (state) => `
 <div class="counter">
 <h1>Count: ${state.count}</h1>
 <button phx-click="increment">+</button>
 <button phx-click="decrement">-</button>
 </div>
 `,
});
```
// Create server
const server = createLiveViewServer({ secret: 'your-secret-key' });
server.register('/', CounterView);

// Handle requests
app.get('*', (req) => server.handleRequest(req));
app.ws('/live/websocket', (ws, req) => server.handleSocket(ws, req));

```

Client-Side

```

import { initLiveView } from 'philjs-liveview/client';

// Auto-initialize from container with data-phx-main
initLiveView();

```

PHX Attributes

Event Bindings

```


<button phx-click="action_name">Click Me</button>


<button phx-click="delete" phx-value-id="123">Delete</button>


<form phx-submit="save" phx-change="validate">
  <input name="email" phx-debounce="300" />
  <button type="submit">Save</button>
</form>


<input phx-keydown="search" phx-keydown-key="Enter" />


<input phx-focus="showSuggestions" phx-blur="hideSuggestions" />

```

Loading States

```


<button phx-click="save" phx-disable-with="Saving... ">
  Save
</button>


<div phx-loading-class="opacity-50">
  Content
</div>

```

Navigation

```


<a href="/users/123" phx-link="patch">View User</a>


<a href="/dashboard" phx-link="redirect">Dashboard</a>


<a href="/page/2" phx-link="patch" phx-link-replace>Page 2</a>

```

LiveComponents

Stateful components that live within a LiveView:

```
import { createLiveComponent } from 'philjs-liveview/server';

const Modal = createLiveComponent({
  id: (props) => `modal-${props.name}`,
  mount: () => ({ open: false }),

  handleEvent: (event, state) => {
    if (event.type === 'toggle') {
      return { open: !state.open };
    }
    return state;
  },

  render: (state, props) => `
    <div id="${props.name}" class="modal ${state.open ? 'open' : ''}>
      <div class="modal-backdrop" phx-click="toggle"></div>
      <div class="modal-content">
        ${props.children}
      </div>
    </div>
  `,
});

});
```

Hooks

Client-side JavaScript interop:

```
import { registerHooks } from 'philjs-liveview/client';

registerHooks({
  InfiniteScroll: {
    mounted() {
      const observer = new IntersectionObserver((entries) => {
        if (entries[0].isIntersecting) {
          this.pushEvent('load-more', {});
        }
      });
      observer.observe(this.el);
    },
  },
  Chart: {
    mounted() {
      this.chart = new Chart(this.el, { /* ... */ });

      this.handleEvent('update-data', (data) => {
        this.chart.update(data);
      });
    },
    destroyed() {
      this.chart.destroy();
    },
  },
});
```

Use hooks in templates:

```
<div phx-hook="InfiniteScroll"></div>
<canvas phx-hook="Chart"></canvas>
```

Forms

Validation

```

import { validateForm } from 'philjs-liveview';

const validations = [
  { field: 'email', rule: 'required', message: 'Email is required' },
  { field: 'email', rule: 'email', message: 'Invalid email' },
  { field: 'password', rule: 'min', params: 8, message: 'Password must be at least 8 characters' },
];

const UserForm = createLiveView({
  mount: () => ({
    data: { email: '', password: '' },
    errors: {},
  }),
  handleEvent: (event, state) => {
    if (event.type === 'validate') {
      const errors = validateForm(event.value, validations);
      return { ...state, data: event.value, errors };
    }
    if (event.type === 'submit') {
      const errors = validateForm(event.value, validations);
      if (Object.keys(errors).length === 0) {
        // Save user
      }
      return { ...state, errors };
    }
    return state;
  },
  render: (state) => `
    <form phx-submit="submit" phx-change="validate">
      <input name="email" value="${state.data.email}" />
      ${state.errors.email ? `<span class="error">${state.errors.email.join(', ')}` : ''}
      <input name="password" type="password" />
      ${state.errors.password ? `<span class="error">${state.errors.password.join(', ')}` : ''}
      <button type="submit">Sign Up</button>
    </form>
  `,
});

```

File Uploads

```

const UploadView = createLiveView({
  mount: () => ({
    uploads: [],
  }),
  handleEvent: (event, state, socket) => {
    if (event.type === 'phx:upload') {
      // Handle uploaded files
      return { uploads: [...state.uploads, ...event.value.files] };
    }
    return state;
  },
  render: (state) => `
    <form phx-submit="save">
      <input type="file" name="avatar" phx-hook="FileUpload" multiple />
      <ul>
        ${state.uploads.map(f => `<li>${f.name}</li>`).join('')}
      </ul>
      <button type="submit">Upload</button>
    </form>
  `,
});

```

PubSub

Real-time broadcasting between views:

```

import { createMemoryPubSub } from 'philjs-liveview/server';

const pubsub = createMemoryPubSub();

// In a LiveView
const ChatView = createLiveView({
  mount: (socket) => {
    // Subscribe to chat messages
    const unsubscribe = pubsub.subscribe('chat:lobby', (event, payload) => {
      socket.assign({ messages: [...socket.state.messages, payload] });
    });

    socket.onLeave = () => unsubscribe();

    return { messages: [] };
  },
  handleEvent: (event, state, socket) => {
    if (event.type === 'send_message') {
      // Broadcast to all subscribers
      pubsub.broadcast('chat:lobby', 'new_message', {
        text: event.value.text,
        user: socket.session.user,
      });
    }
    return state;
  },
  render: (state) => `
    <div class="chat">
      <ul>
        ${state.messages.map(m => `<li>${m.user}: ${m.text}</li>`).join('')}
      </ul>
      <form phx-submit="send_message">
        <input name="text" />
        <button type="submit">Send</button>
      </form>
    </div>
  `,
});

```

Built-in Hooks

- **InfiniteScroll** - Load more content on scroll
- **Focus** - Focus element on mount/update
- **Clipboard** - Copy to clipboard
- **LocalTime** - Convert UTC to local time
- **Sortable** - Drag and drop sorting
- **Debounce** - Debounce input events
- **Countdown** - Countdown timer

Template Helpers

```

import { html, raw, when, each, input, errorTag } from 'philjs-liveview';

const render = (state) => html`
  <div>
    ${when(state.loading, '<div class="spinner"></div>')

    ${each(
      state.items,
      (item) => item.id,
      (item) => html`<li>${item.name}</li>`
    )}

    ${input('email', { type: 'email', value: state.email, required: true })}
    ${errorTag(state.errors, 'email')}

    ${raw('<script>alert("trusted")</script>')}
  </div>
`;

```

Comparison with Similar Frameworks

Feature	PhilJS LiveView	Phoenix LiveView	Laravel Livewire	Dioxus LiveView
Language	TypeScript	Elixir	PHP	Rust
DOM Diffing	morphdom-style	morphdom	morphdom	Virtual DOM
Components	Yes	Yes	Yes	Yes
Hooks	Yes	Yes	Yes	N/A
File Upload	Yes	Yes	Yes	Planned
SSR	Yes	Yes	Yes	Yes

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./client`, `./server`
- Source files: `packages/philjs-liveview/src/index.ts`, `packages/philjs-liveview/src/client.ts`, `packages/philjs-liveview/src/server.ts`

Public API

- Direct exports: `LiveViewClient`, `LiveViewServer`, `PubSub`, `createLiveViewClient`, `createLiveViewServer`, `createMemoryPubSub`, `initLiveView`, `liveViewMiddleware`, `liveViewWebSocketHandler`
- Re-exported names: `createLiveComponent`, `createLiveView`, `livePatch`, `liveRedirect`, `registerHooks`
- Re-exported modules: `./client.js`, `./differ.js`, `./forms.js`, `./hooks.js`, `./live-component.js`, `./live-socket.js`, `./live-view.js`, `./navigation.js`, `./server.js`, `./types.js`

License

MIT

@philjs/llm-ui - Type: Node package - Purpose: Streaming chat UI components for PhilJS - ChatGPT-style interfaces, markdown, code blocks, tool calls - Version: 0.1.0 - Location: packages/philjs-llm-ui - Entry points: packages/philjs-llm-ui/src/index.ts - Keywords: philjs, llm, chat, streaming, ui, markdown, ai, chatgpt

@philjs/llm-ui

Streaming chat UI components for PhilJS - ChatGPT-style interfaces, markdown, code blocks, tool calls

Overview

Streaming chat UI components for PhilJS - ChatGPT-style interfaces, markdown, code blocks, tool calls

Focus Areas

- philjs, llm, chat, streaming, ui, markdown, ai, chatgpt

Entry Points

- `packages/philjs-llm-ui/src/index.ts`

Quick Start

```
import { // Core classes
  ChatContainer, // Hooks
  useChat, // Types
  type ChatMessage } from '@philjs/llm-ui';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- // Core classes `ChatContainer`
- // Hooks `useChat`
- // Types type `ChatMessage`
- `ChatConfig`
- `ChatInput`
- `ChatInputConfig`
- `MarkdownRenderer`
- `MessageComponent`
- `MessageComponentConfig`
- `StreamingText`
- `StreamingTextConfig`
- `ThinkingIndicator`

Install

```
pnpm add @philjs/llm-ui
```

Usage

```
import { // Core classes
  ChatContainer, // Hooks
  useChat, // Types
  type ChatMessage } from '@philjs/llm-ui';
```

Scripts

- `pnpm run build`
- `pnpm run test`

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-lm-ui/src/index.ts

Public API

- Direct exports: // Core classes ChatContainer, // Hooks useChat, // Types type ChatMessage, ChatConfig, ChatInput, ChatInputConfig, MarkdownRenderer, MessageComponent, MessageComponentConfig, StreamingText, StreamingTextConfig, ThinkingIndicator, ToolCallDisplay, UseChatResult, useStreamingText
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/maps - Type: Node package - Purpose: Map components for PhilJS with support for Google Maps, Mapbox, and Leaflet - Version: 0.1.0 - Location: packages/philjs-maps - Entry points: .. ./google, ./mapbox, ./leaflet, packages/philjs-maps/src/utils/index.ts - Keywords: philjs, maps, google-maps, mapbox, leaflet, geolocation, gis, web-components

@philjs/maps

Map components for React applications with support for Mapbox, Google Maps, and Leaflet. Build interactive, customizable maps with markers, layers, and geospatial features.

Installation

```
npm install @philjs/maps
# or
yarn add @philjs/maps
# or
pnpm add @philjs/maps
```

Basic Usage

```
import { Map, Marker, Popup } from '@philjs/maps';

function LocationMap() {
  const center = { lat: 40.7128, lng: -74.0060 };

  return (
    <Map
      provider="mapbox"
      accessToken={MAPBOX_TOKEN}
      center={center}
      zoom={12}
    >
      <Marker position={center}>
        <Popup>New York City</Popup>
      </Marker>
    </Map>
  );
}
```

Features

- **Multiple Providers** - Mapbox, Google Maps, Leaflet support
- **Markers** - Customizable map markers with icons
- **Popups** - Information popups on markers
- **Layers** - GeoJSON, tile layers, heatmaps
- **Clustering** - Automatic marker clustering
- **Drawing Tools** - Draw shapes, polygons, routes
- **Geocoding** - Address to coordinates conversion
- **Directions** - Route planning and navigation
- **Street View** - Google Street View integration
- **3D Buildings** - 3D building rendering (Mapbox)
- **Offline Maps** - Cache tiles for offline use
- **Accessibility** - Keyboard navigation support

Components

Component	Description
Map	Main map container
Marker	Map marker
Popup	Information popup
GeoJSON	GeoJSON layer
HeatmapLayer	Heatmap visualization
ClusterLayer	Marker clustering
DrawControl	Drawing tools
SearchBox	Location search

Providers

```
// Mapbox
<Map provider="mapbox" accessToken={MAPBOX_TOKEN} />

// Google Maps
<Map provider="google" apiKey={GOOGLE_MAPS_KEY} />

// Leaflet (OpenStreetMap)
<Map provider="leaflet" />
```

Hooks

```
import { useMap, useGeolocation } from '@philjs/maps';

const { flyTo, zoomIn, getBounds } = useMap();
const { position, error } = useGeolocation();
```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .., /google, /mapbox, /leaflet, /utils
- Source files: packages/philjs-maps/src/utils/index.ts

Public API

- Direct exports: (none detected)
- Re-exported names: (none detected)
- Re-exported modules: ./cluster.js

License

MIT

@philjs/media-stream - Type: Node package - Purpose: Advanced media stream processing for PhilJS - video filters, audio processing, face detection, recording, mixing - Version: 0.1.0 - Location: packages/philjs-media-stream - Entry points: packages/philjs-media-stream/src/index.ts - Keywords: philjs, media-stream, video-filter, audio-processing, chroma-key, face-detection, recording

@philjs/media-stream

Advanced media stream processing for PhilJS - video filters, audio processing, face detection, recording, mixing

Overview

Advanced media stream processing for PhilJS - video filters, audio processing, face detection, recording, mixing

Focus Areas

- philjs, media-stream, video-filter, audio-processing, chroma-key, face-detection, recording

Entry Points

- packages/philjs-media-stream/src/index.ts

Quick Start

```
import { // Core classes
        MediaStreamProcessor, // Hooks
        useMediaProcessor, // Types
        type VideoFilterConfig } from '@philjs/media-stream';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- // Core classes MediaStreamProcessor
- // Hooks useMediaProcessor
- // Types type VideoFilterConfig
- AudioProcessorConfig
- AudioStreamProcessor
- AudioVisualizer
- ChromaKeyConfig
- ChromaKeyProcessor
- FaceDetection
- FaceDetector
- MediaProcessorConfig
- MediaStreamRecorder

Install

```
pnpm add @philjs/media-stream
```

Usage

```
import { // Core classes
  MediaStreamProcessor, // Hooks
  useMediaProcessor, // Types
  type VideoFilterConfig } from '@philjs/media-stream';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-media-stream/src/index.ts

Public API

- Direct exports: // Core classes MediaStreamProcessor, // Hooks useMediaProcessor, // Types type VideoFilterConfig, AudioProcessorConfig, AudioStreamProcessor, AudioVisualizer, ChromaKeyConfig, ChromaKeyProcessor, FaceDetection, FaceDetector, MediaProcessorConfig, MediaStreamRecorder, MixerInput, RecorderConfig, StreamMixer, StreamQualityMetrics, StreamQualityMonitor, VideoFilterProcessor, VisualizerConfig, useAudioVisualizer, useStreamMixer, useStreamRecorder
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/meta - Type: Node package - Purpose: Next.js/SvelteKit-style meta-framework for PhilJS with file-based routing, SSR, and data loading - Version: 0.1.0 - Location: packages/philjs-meta - Entry points: packages/philjs-meta/src/index.ts, packages/philjs-meta/src/router/index.ts, packages/philjs-meta/src/data/index.ts, packages/philjs-meta/src/server/index.ts, packages/philjs-meta/src/config/index.ts, packages/philjs-meta/src/seo.tsx - Keywords: philjs, meta-framework, ssr, ssg, file-based-routing, seo, meta, og, opengraph, twitter, sitemap

philjs-meta

SEO and meta tag management for PhilJS with OpenGraph, Twitter Cards, JSON-LD, and sitemap generation.

Features

- Declarative head management
- OpenGraph meta tags
- Twitter Cards
- JSON-LD structured data
- Sitemap generation
- Robots.txt generation
- Multi-language support
- Automatic cleanup

Installation

```
npm install philjs-meta
```

Usage

Basic Setup

```
import { HeadProvider } from 'philjs-meta';

function App() {
  return (
    <HeadProvider>
      <YourApp />
    </HeadProvider>
  );
}
```

Setting Title and Meta Tags

```

import { Head, Meta, Title } from 'philjs-meta';

function Page() {
  return (
    <>
    <Head>
      <Title>My Page Title</Title>
      <Meta name="description" content="Page description" />
      <Meta name="keywords" content="philjs, react, framework" />
    </Head>

    <div>Page content</div>
  </>
);
}

```

All-in-One SEO Component

```

import { SEO } from 'philjs-meta';

function Page() {
  return (
    <>
    <SEO
      config={{
        title: 'My Page',
        description: 'Page description',
        canonical: 'https://example.com/page',
      }}
      openGraph={{
        title: 'My Page',
        description: 'Page description',
        image: 'https://example.com/og-image.jpg',
        url: 'https://example.com/page',
      }}
      twitter={{
        card: 'summary_large_image',
        site: '@mysite',
        creator: '@creator',
      }}
      jsonLd={{
        '@context': 'https://schema.org',
        '@type': 'Article',
        headline: 'My Article',
        author: {
          '@type': 'Person',
          name: 'John Doe',
        },
      }}
    />

    <div>Content</div>
  </>
);
}

```

Generate Sitemap

```

import { generateSitemap } from 'philjs-meta';

const entries = [
  {
    url: 'https://example.com',
    lastmod: '2025-12-16',
    changefreq: 'daily',
    priority: 1.0,
  },
  {
    url: 'https://example.com/about',
    lastmod: '2025-12-10',
    changefreq: 'monthly',
    priority: 0.8,
  },
];

const sitemap = generateSitemap(entries);
// Write to public/sitemap.xml

```

Generate robots.txt

```

import { generateRobotsTxt } from 'philjs-meta';

const robots = generateRobotsTxt({
  rules: [
    {
      userAgent: '*',
      allow: ['/'],
      disallow: ['/admin', '/api'],
    },
  ],
  sitemaps: ['https://example.com/sitemap.xml'],
});
// Write to public/robots.txt

```

API

See [full API documentation](#).

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./router`, `./data`, `./server`, `./config`, `./seo`
- Source files: `packages/philjs-meta/src/index.ts`, `packages/philjs-meta/src/router/index.ts`, `packages/philjs-meta/src/data/index.ts`, `packages/philjs-meta/src/server/index.ts`, `packages/philjs-meta/src/config/index.ts`, `packages/philjs-meta/src/seo.tsx`

Public API

- Direct exports: `AlternateLanguages`, `BasicMeta`, `BuildConfig`, `BundlerConfig`, `CORSConfig`, `CSSModulesConfig`, `DNSPrefetch`, `ExperimentalConfig`, `Favicons`, `HTTPSConfig`, `HeadersConfig`, `I18nConfig`, `ImageRemotePattern`, `ImagesConfig`, `JSONLD`, `LoadConfigOptions`, `LocaleDomain`, `OpenGraph`, `PhilJSConfig`, `Preconnect`, `RateLimitConfig`, `RedirectConfig`, `RewriteConfig`, `RouteCondition`, `SEO`, `SEOProps`, `SSGConfig`, `SSRConfig`, `ServerConfig`, `TwitterCard`, `defaultConfig`, `defineConfig`, `getEnvConfig`, `loadConfig`, `validateConfig`
- Re-exported names: `APIContext`, `APIHandler`, `APIResponse`, `APIRouteHandler`, `ActionContext`, `ActionErrors`, `ActionFunction`, `ActionResponse`, `AlternateLanguages`, `AuthOptions`, `BasicMeta`, `BodyParserOptions`, `BuildCLIOptions`, `BuildConfig`, `BuildContext`, `BuildError`, `BuildOptions`, `BuildPlugin`, `BuildResult`, `BundleInfo`, `BundlerConfig`, `CLIOptions`, `CORSOptions`, `CacheControlOptions`, `CacheEntry`, `CacheOptions`, `CacheStats`, `Compiler`, `CompressionOptions`, `CookieOptions`, `CookieStore`, `DNSPrefetch`, `DeferredData`, `DevServerOptions`, `ErrorBoundaryConfig`, `ErrorBoundaryProps`, `ExperimentalConfig`, `ExtractParams`, `Favicons`, `FileRouter`, `FileRouterOptions`, `FormActionHandler`, `FormData`, `GenerateOptions`, `GeoInfo`, `Head`, `HeadProvider`, `HeadersConfig`, `HttpMethod`, `I18nConfig`, `ISRConfig`, `ISManager`, `ImagesConfig`, `InterceptedRoute`, `JSONBody`, `JSONLDConfig`, `LayoutComposition`, `LayoutContext`, `LayoutContextManager`, `LayoutDefinition`, `LayoutProps`, `LayoutTreeNode`, `LayoutUtils`, `Link`, `LinkTag`, `LoadConfigOptions`, `LoaderContext`, `LoaderFunction`, `LoadingConfig`, `LoadingFactory`, `LoadingProps`, `LoggerOptions`, `Meta`, `MetaConfig`, `MetaTag`, `MiddlewareChain`, `MiddlewareContext`, `MiddleWareFunction`, `MultipartBody`, `NextResponse`, `NotFoundResponse`, `OpenGraph`, `OpenGraphConfig`, `OpenGraphImage`, `ParallelRouteSlot`, `PhilJSConfig`, `Preconnect`, `RateLimitInfo`, `RateLimitOptions`, `RateLimitStore`, `RedirectConfig`, `RedirectResponse`, `RequestTiming`, `RevalidatedOptions`, `RewriteConfig`, `RobotsConfig`, `RouteBundle`, `RouteDefinition`, `RouteHandlerConfig`, `RouteManifest`, `RouteMetadata`, `RouteSegment`, `RouteSegmentType`, `SEO`, `SSEEvent`, `SSEStream`, `SSGConfig`, `SSRConfig`, `SWRConfig`, `SWRState`, `Schema`, `SchemaError`, `Schemalssue`, `SecurityHeadersOptions`, `ServerConfig`, `ServerContext`, `SitemapEntry`, `StartServerOptions`, `StaticExportOptions`, `StaticPageInfo`, `Title`, `TwitterCard`, `TwitterConfig`, `analyzerPlugin`, `auth`, `bodyParser`, `build`, `cache`, `cacheControl`, `cached`, `compression`, `cors`, `createAPIRoute`, `createCompiler`, `createCompilerFromConfig`, `createErrorBoundary`, `createFileRouter`, `createFormAction`, `createLayoutTree`, `createLoadingWrapper`, `createMiddlewareContext`, `createServerContext`, `createSitemapEntry`, `defaultConfig`, `defer`, `defineAPIHandler`, `defineAction`, `defineConfig`, `defineLoader`, `dev`, `executeAction`, `executeLoader`, `generate`, `generateRobotsTxt`, `generateRouteManifest`, `generateSitemap`, `generateSitemapFromRoutes`, `generateSitemapIndex`, `getEnvConfig`, `getLayoutsForRoute`, `getParallelSlots`, `getRouteKey`, `hydration`, `json`, `loadConfig`, `logger`, `matchApiRoute`, `matchRoute`, `parseBody`, `parseInterceptedRoute`, `rateLimit`, `redirect`, `revalidatePath`, `run`, `securityHeaders`, `setRouteContext`, `splitSitemap`, `start`, `staticExportPlugin`, `unstable_cache`, `useActionData`, `useHead`, `useIsSubmitting`, `useLoaderData`, `useParams`, `useSWR`, `useSearchParams`, `validateConfig`, `z`
- Re-exported modules: `./Head.js`, `./api-routes.js`, `./build/compiler.js`, `./cache.js`, `./cli/index.js`, `./config/index.js`, `./data/cache.js`, `./data/loaders.js`, `./file-based.js`, `./layouts.js`, `./loaders.js`, `./middleware.js`, `./router/file-based.js`, `./router/layouts.js`, `./seo.js`, `./server/api-routes.js`, `./server/middleware.js`, `./sitemap.js`, `./types.js`

License

MIT

@philjs/migrate - Type: Node package - Purpose: Migration CLI for PhilJS - migrate from React, Vue, Angular, Svelte - Version: 0.1.0 - Location: packages/philjs-migrate - Entry points: packages/philjs-migrate/src/index.ts

philjs-migrate

Migration codemods to convert React, Vue, and Svelte applications to PhilJS.

Installation

```
pnpm add -D philjs-migrate
```

Or use directly with npx:

```
npx philjs-migrate
```

Usage

Command Line

Migrate your entire project:

```
philjs-migrate --from react --source ./src --output ./migrated
```

Analyze before migrating:

```
philjs-migrate --from vue --source ./src --analyze-only
```

Supported Frameworks

- React - Convert React components to PhilJS

- **Vue** - Transform Vue 3 components to PhilJS
- **Svelte** - Migrate Svelte components to PhilJS

CLI Options

```
Options:
--from <framework>      Source framework (react, vue, svelte)
--source <dir>           Source directory to migrate
--output <dir>           Output directory (default: source_philjs)
--analyze-only            Analyze without transforming files
--dry-run                 Preview changes without writing files
--report <file>          Generate migration report JSON file
--help                    Display help information
```

Examples

Migrating from React

```
# Basic migration
philjs-migrate --from react --source ./src

# With analysis report
philjs-migrate --from react --source ./src --report migration-report.json

# Dry run to preview changes
philjs-migrate --from react --source ./src --dry-run
```

Before (React):

```
import { useState, useEffect } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log('Count changed:', count);
  }, [count]);

  return (
    <button onClick={() => setCount(count + 1)}>
      Count: {count}
    </button>
  );
}
```

After (PhilJS):

```
import { signal, effect } from 'philjs-core';

function Counter() {
  const count = signal(0);

  effect(() => {
    console.log('Count changed:', count());
  });

  return (
    <button onClick={() => count.set(c => c + 1)}>
      Count: {count()}
    </button>
  );
}
```

Migrating from Vue

```
philjs-migrate --from vue --source ./src/components
```

Before (Vue):

```
<script setup>
import { ref, computed } from 'vue';

const count = ref(0);
const doubled = computed(() => count.value * 2);

function increment() {
  count.value++;
}

<template>
  <button @click="increment">
    Count: {{ count }} (Doubled: {{ doubled }})
  </button>
</template>
```

After (PhilJS):

```

import { signal, memo } from 'philjs-core';

function Counter() {
  const count = signal(0);
  const doubled = memo(() => count() * 2);

  const increment = () => count.set(c => c + 1);

  return (
    <button onClick={increment}>
      Count: {count()} (Doubled: {doubled()})
    </button>
  );
}

```

Migrating from Svelte

```
philjs-migrate --from svelte --source ./src
```

Before (Svelte):

```

<script>
  let count = 0;
  $: doubled = count * 2;

  function increment() {
    count += 1;
  }
</script>

<button on:click={increment}>
  Count: {count} (Doubled: {doubled})
</button>

```

After (PhilJS):

```

import { signal, memo } from 'philjs-core';

function Counter() {
  const count = signal(0);
  const doubled = memo(() => count() * 2);

  const increment = () => count.set(c => c + 1);

  return (
    <button onClick={increment}>
      Count: {count()} (Doubled: {doubled()})
    </button>
  );
}

```

Programmatic API

Use the migration tools programmatically in Node.js:

```

import { migrate, analyzeProject, generateReport } from 'philjs-migrate';

// Analyze a project
const analysis = await analyzeProject({
  framework: 'react',
  source: './src'
});

console.log('Components to migrate:', analysis.componentCount);
console.log('Estimated complexity:', analysis.complexity);

// Perform migration
const result = await migrate({
  framework: 'react',
  source: './src',
  output: './migrated',
  dryRun: false
});

// Generate detailed report
const report = generateReport(result);
console.log('Migration complete:', report.summary);

```

Framework-Specific Transforms

```

import { ReactTransform } from 'philjs-migrate/react';
import { VueTransform } from 'philjs-migrate/vue';
import { SvelteTransform } from 'philjs-migrate/svelte';

// Use specific transformer
const reactTransform = new ReactTransform();
const code = await reactTransform.transform(sourceCode);

```

API

Functions

- `migrate(options)` - Migrate project from another framework
- `analyzeProject(options)` - Analyze project before migration
- `generateReport(result)` - Generate detailed migration report

Transforms

- `ReactTransform` - React to PhilJS transformation
- `VueTransform` - Vue to PhilJS transformation
- `SvelteTransform` - Svelte to PhilJS transformation

Options

```
interface MigrationOptions {  
  framework: 'react' | 'vue' | 'svelte';  
  source: string;  
  output?: string;  
  dryRun?: boolean;  
  analyze?: boolean;  
}
```

Migration Patterns

State Management

Framework	Before	PhilJS
React	<code>useState()</code>	<code>signal()</code>
Vue	<code>ref()</code>	<code>signal()</code>
Svelte	<code>let x = ...</code>	<code>signal()</code>

Computed Values

Framework	Before	PhilJS
React	<code>useMemo()</code>	<code>memo()</code>
Vue	<code>computed()</code>	<code>memo()</code>
Svelte	<code>\$: x = ...</code>	<code>memo()</code>

Side Effects

Framework	Before	PhilJS
React	<code>useEffect()</code>	<code>effect()</code>
Vue	<code>watch() / watchEffect()</code>	<code>effect()</code>
Svelte	<code>\$: { ... }</code>	<code>effect()</code>

Limitations

- Manual review recommended for complex state management patterns
- Custom hooks/composables may need manual adaptation
- Context/provide-inject patterns require manual migration
- Third-party library integrations need review

Post-Migration Steps

1. Review generated code for accuracy
2. Update dependencies in package.json
3. Install PhilJS packages
4. Run tests and fix any issues
5. Update build configuration

Documentation

For more information, see the PhilJS migration guide.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: ..
- Source files: packages/philjs-migrate/src/index.ts

Public API

- Direct exports: ANGULAR_TRANSFORMS, MigrationError, MigrationOptions, MigrationResult, REACT_TRANSFORMS, SVELTE_TRANSFORMS, VUE_TRANSFORMS, analyzeProject, migrate
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/motion - Type: Node package - Purpose: Spring physics animation system for PhilJS - gesture-driven, FLIP, scroll-linked animations - Version: 0.1.0 - Location: packages/philjs-motion - Entry points: packages/philjs-motion/src/index.ts - Keywords: philjs, animation, spring, physics, flip, gesture, scroll

@philjs/motion

Spring physics animation system for PhilJS - gesture-driven, FLIP, scroll-linked animations

Overview

Spring physics animation system for PhilJS - gesture-driven, FLIP, scroll-linked animations

Focus Areas

- philjs, animation, spring, physics, flip, gesture, scroll

Entry Points

- packages/philjs-motion/src/index.ts

Quick Start

```
import { AnimatedTransform, AnimatedValue, AnimationCallback } from '@philjs/motion';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- AnimatedTransform
- AnimatedValue
- AnimationCallback
- AnimationSequence
- AnimationState
- Easing
- EasingFunction
- FlipAnimation
- GestureAnimation
- GestureState
- LayoutRect
- ScrollAnimation

Install

```
pnpm add @philjs/motion
```

Usage

```
import { AnimatedTransform, AnimatedValue, AnimationCallback } from '@philjs/motion';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-motion/src/index.ts

Public API

- Direct exports: AnimatedTransform, AnimatedValue, AnimationCallback, AnimationSequence, AnimationState, Easing, EasingFunction, FlipAnimation, GestureAnimation, GestureState, LayoutRect, ScrollAnimation, ScrollInfo, Spring, SpringConfig, SpringPresets, SpringVector, TransformValues, useAnimatedTransform, useFlip, useGesture, useScrollAnimation, useSpring, useSpringVector
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/native - Type: Node package - Purpose: Cross-platform native development for PhilJS - iOS, Android, and Desktop - Version: 0.1.0 - Location: packages/philjs-native - Entry points: packages/philjs-native/src/index.ts, packages/philjs-native/src/runtime.ts, packages/philjs-native/src/components/index.ts, packages/philjs-native/src/navigation.ts, packages/philjs-native/src/capacitor/index.ts, packages/philjs-native/src/tauri/index.ts, packages/philjs-native/src/gestures/index.ts - Keywords: philjs, native, mobile, ios, android, desktop, cross-platform, capacitor, tauri, hybrid-app, pwa

PhilJS Native

Cross-platform mobile development for PhilJS. Build native iOS, Android, and Web apps with one codebase.

Requirements

- **Node.js 24** or higher
- **TypeScript 6** or higher
- **ESM only** - CommonJS is not supported

Installation

```
pnpm add philjs-native philjs-core
```

Quick Start

Create a new PhilJS Native project:

```
npx philjs-native init my-app
cd my-app
npm install
npm start
```

Features

- **Cross-Platform**: Write once, run on iOS, Android, and Web
- **Native Performance**: Native components for each platform
- **Familiar API**: Similar to React Native for easy adoption
- **Type-Safe**: Full TypeScript support
- **Signal-Based**: Reactive primitives from philjs-core

Core Concepts

Creating an App

```
import {
  createNativeApp,
  View,
  Text,
  SafeAreaView,
  StatusBar,
  StyleSheet,
} from 'philjs-native';

function App() {
  return (
    <SafeAreaView style={styles.container}>
      <StatusBar barStyle="dark-content" />
      <View style={styles.content}>
        <Text style={styles.title}>Hello, PhilJS Native!</Text>
      </View>
    </SafeAreaView>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
  },
  content: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
  title: {
    fontSize: 24,
    fontWeight: 'bold',
  },
});

const app = createNativeApp({
  root: App,
});

app.render();
```

Components

Core Components

- **View** - Basic container with flexbox layout
- **Text** - Text display with styling
- **Image** - Image display with native optimization
- **ScrollView** - Scrollable container
- **FlatList** - Virtualized list for large datasets
- **TouchableOpacity** - Touch feedback with opacity
- **Pressable** - Modern touch handling

- **TextInput** - Text input field
- **Button** - Native button

Layout Components

- **SafeAreaView** - Safe area handling for notches
- **StatusBar** - Status bar control

Example: Building a List

```
import { View, FlatList, Text, TouchableOpacity, StyleSheet } from 'philjs-native';

const data = [
  { id: '1', title: 'Item 1' },
  { id: '2', title: 'Item 2' },
  { id: '3', title: 'Item 3' },
];

function MyList() {
  return (
    <FlatList
      data={data}
      keyExtractor={({ item }) => item.id}
      renderItem={({ item }) =>
        <TouchableOpacity
          style={styles.item}
          onPress={() => console.log('Pressed', item.title)}
        >
          <Text>{item.title}</Text>
        </TouchableOpacity>
      }
    />
  );
}

const styles = StyleSheet.create({
  item: {
    padding: 16,
    borderBottomWidth: 1,
    borderBottomColor: '#eee',
  },
});
```

Navigation

Stack Navigator

```
import { createNativeRouter, useNativeNavigation } from 'philjs-native';

const { Navigator, navigation } = createNativeRouter({
  screens: [
    { name: 'Home', component: HomeScreen },
    { name: 'Profile', component: ProfileScreen },
  ],
  options: {
    initialRouteName: 'Home',
  },
});

function HomeScreen() {
  return (
    <View>
      <Button
        title="Go to Profile"
        onPress={() => navigation.navigate('Profile', { userId: 123 })}
      />
    </View>
  );
}
```

Tab Navigator

```

import { createNativeTabs } from 'philjs-native';

const Tabs = createNativeTabs({
  tabBarPosition: 'bottom',
  tabBarActiveTintColor: '#007AFF',
});

function App() {
  return (
    <Tabs.Navigator>
      <Tabs.Screen
        name="Home"
        component={HomeScreen}
        options={{
          tabBarIcon: ({ color, size }) => <HomeIcon color={color} size={size} />,
        }}
      />
      <Tabs.Screen
        name="Profile"
        component={ProfileScreen}
        options={{
          tabBarIcon: ({ color, size }) => <ProfileIcon color={color} size={size} />,
        }}
      />
    </Tabs.Navigator>
  );
}


```

Navigation Hooks

```

import { useNativeNavigation, useRoute, useFocusEffect } from 'philjs-native';

function ProfileScreen() {
  const navigation = useNativeNavigation();
  const route = useRoute();

  useFocusEffect(() => {
    console.log('Screen focused');
    return () => console.log('Screen unfocused');
  });

  return (
    <View>
      <Text>User ID: {route.params.userId}</Text>
      <Button title="Go Back" onPress={() => navigation.goBack()} />
    </View>
  );
}


```

Native APIs

Camera

```

import { Camera } from 'philjs-native';

// Request permission
const status = await Camera.requestPermission();

// Take a photo
const photo = await Camera.takePicture({
  quality: 0.8,
  base64: true,
});

// Pick from library
const image = await Camera.pickImage({
  allowsEditing: true,
  aspect: [4, 3],
});


```

Geolocation

```

import { Geolocation, useLocation } from 'philjs-native';

// Get current position
const location = await Geolocation.getCurrentPosition({
  accuracy: 'high',
});

// Watch position
const unsubscribe = Geolocation.watchPosition(
  (location) => console.log(location),
  (error) => console.error(error),
  { accuracy: 'balanced' }
);

// Use hook
function LocationComponent() {
  const { location, error, loading } = useLocation();

  if (loading) return <Text>Loading...</Text>;
  if (error) return <Text>Error: {error.message}</Text>;

  return <Text>Lat: {location.coords.latitude}</Text>;
}

```

Storage

```

import { Storage, useStorage,getJSON, setJSON } from 'philjs-native';

// Basic operations
await Storage.setItem('key', 'value');
const value = await Storage.getItem('key');
await Storage.removeItem('key');

// JSON helpers
await setJSON('user', { name: 'John', age: 30 });
const user = await getJSON('user');

// Reactive storage hook
function SettingsScreen() {
  const [theme, setTheme] = useStorage('theme', 'light');

  return (
    <Button
      title={`Switch to ${theme() === 'light' ? 'dark' : 'light'}`}
      onPress={() => setTheme(theme() === 'light' ? 'dark' : 'light')}
    />
  );
}

```

Haptics

```

import { Haptics, impactMedium, notifySuccess } from 'philjs-native';

// Impact feedback
await Haptics.impact('medium');

// Notification feedback
await Haptics.notification('success');

// Selection feedback
await Haptics.selection();

// Convenience functions
await impactMedium();
await notifySuccess();

```

Notifications

```

import { Notifications, useNotificationReceived } from 'philjs-native';

// Request permission
const status = await Notifications.requestPermission();

// Schedule local notification
const id = await Notifications.scheduleNotification(
  {
    title: 'Reminder',
    body: 'Check your tasks!',
    data: { screen: 'Tasks' },
  },
  { type: 'timeInterval', seconds: 60 }
);

// Listen for notifications
useNotificationReceived((notification) => {
  console.log('Received:', notification);
});

```

Biometrics

```
import { Biometrics, useBiometrics } from 'philjs-native';

// Check availability
const support = await Biometrics.isAvailable();
if (support.available) {
  console.log('Biometry type:', support.biometryType);
}

// Authenticate
const result = await Biometrics.authenticate({
  promptMessage: 'Verify your identity',
});

if (result.success) {
  // Authenticated
}

// Secure storage
await Biometrics.secureStore('secret', 'myPassword');
const secret = await Biometrics.secureRetrieve('secret');
```

Clipboard

```
import { Clipboard, useClipboard } from 'philjs-native';

// Copy/paste
await Clipboard.setString('Hello!');
const text = await Clipboard.getString();

// Hook
function CopyButton() {
  const { copy, paste, content } = useClipboard();

  return (
    <View>
      <Button title="Copy" onPress={() => copy('Copied text')} />
      <Text>Clipboard: {content}</Text>
    </View>
  );
}
```

Share

```
import { Share } from 'philjs-native';

// Share content
const result = await Share.share({
  title: 'Check this out',
  message: 'Amazing content!',
  url: 'https://example.com',
});

// Share to specific app
await Share.social.twitter({ message: 'Hello Twitter!' });
await Share.social.whatsapp({ message: 'Hello WhatsApp!' });
```

Styling

StyleSheet

```
import { StyleSheet, platformStyles } from 'philjs-native';

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 16,
  },
  text: {
    fontSize: 16,
    color: '#333',
  },
  // Platform-specific styles
  shadow: platformStyles({
    ios: {
      shadowColor: '#000',
      shadowOffset: { width: 0, height: 2 },
      shadowOpacity: 0.1,
      shadowRadius: 4,
    },
    android: {
      elevation: 3,
    },
    default: {},
  }),
});
```

Theming

```
import {
  useTheme,
  useThemedStyles,
  useColorScheme,
  setTheme,
  lightTheme,
  darkTheme,
} from 'philjs-native';

function App() {
  const colorScheme = useColorScheme();
  const theme = useTheme();

  // Auto dark mode
  useEffect(() => {
    setTheme(colorScheme === 'dark' ? darkTheme : lightTheme);
  }, [colorScheme]);

  return (
    <View style={{ backgroundColor: theme.colors.background }}>
      <Text style={{ color: theme.colors.text }}>Hello!</Text>
    </View>
  );
}

// Themed styles
function ThemedComponent() {
  const styles = useThemedStyles((theme) => ({
    container: {
      backgroundColor: theme.colors.surface,
      padding: theme.spacing.md,
      borderRadius: theme.borderRadius.md,
    },
    text: {
      ...theme.typography.body1,
      color: theme.colors.text,
    },
  }));
}

return (
  <View style={styles.container}>
    <Text style={styles.text}>Themed content</Text>
  </View>
);
}
```

Responsive Design

```
import { responsive, currentBreakpoint, breakpoints } from 'philjs-native';

function ResponsiveComponent() {
  const padding = responsive({
    xs: 8,
    sm: 12,
    md: 16,
    lg: 24,
  });

  const columns = responsive({
    xs: 1,
    md: 2,
    lg: 3,
  });

  return (
    <View style={{ padding, flexDirection: 'row', flexWrap: 'wrap' }}>
      /* Items with responsive columns */
    </View>
  );
}
```

CLI Commands

```

# Create new project
philjs-native init my-app

# Start dev server
philjs-native start

# Run on platform
philjs-native run ios
philjs-native run android
philjs-native run web

# Build for production
philjs-native build --platform ios
philjs-native build --platform android
philjs-native build --platform web

# Add platform
philjs-native add ios
philjs-native add android

# Check environment
philjs-native doctor

# Upgrade
philjs-native upgrade

```

Platform Detection

```

import {
  detectPlatform,
  getPlatformInfo,
  platformSelect,
  onPlatform,
} from 'philjs-native';

// Get current platform
const platform = detectPlatform(); // 'ios' | 'android' | 'web'

// Platform-specific values
const fontSize = platformSelect({
  ios: 17,
  android: 16,
  web: 16,
  default: 16,
});

// Platform-specific code
onPlatform({
  ios: () => console.log('Running on iOS'),
  android: () => console.log('Running on Android'),
  web: () => console.log('Running on Web'),
});

// Detailed platform info
const info = getPlatformInfo();
console.log(info.deviceType); // 'phone' | 'tablet' | 'desktop'
console.log(info.colorScheme); // 'light' | 'dark'

```

TypeScript

PhilJS Native is fully typed. Import types as needed:

```

import type {
  ViewStyle,
  TextStyle,
  ImageStyle,
  Platform,
  PlatformInfo,
  Navigation,
  Route,
  Theme,
} from 'philjs-native';

```

Migration from React Native

PhilJS Native uses a similar API to React Native, making migration straightforward:

React Native	PhilJS Native
react-native	philjs-native
@react-navigation/native	philjs-native (built-in)
react-native-gesture-handler	Built-in
AsyncStorage	Storage
expo-camera	Camera
expo-location	Geolocation
expo-haptics	Haptics
expo-notifications	Notifications

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .., ./runtime, ./components, ./navigation, ./capacitor, ./tauri, ./gestures
- Source files: packages/philjs-native/src/index.ts, packages/philjs-native/src/runtime.ts, packages/philjs-native/src/components/index.ts, packages/philjs-native/src/navigation.ts, packages/philjs-native/src/capacitor/index.ts, packages/philjs-native/src/tauri/index.ts, packages/philjs-native/src/gestures/index.ts

Public API

- Direct exports: AppLifecycleEvents, AppState, BaseGestureEvent, CapacitorConfig, CapacitorPlugin, CapacitorStatusBar, DeviceInfo, Dimensions, EventHandle, GestureConfig, GestureHandlerOptions, GestureState, InvokeOptions, LifecycleEvent, Link, LinkProps, LongPressGestureConfig, LongPressGestureEvent, NativeApp, NativeAppConfig, NativeBridge, NativeBridgeMessage, NativeBridgeResponse, NativeComponentConfig, NativeStack, NativeTabs, Navigation, NavigationOptions, NavigationState, PanGestureConfig, PanGestureEvent, PinchGestureConfig, PinchGestureEvent, Platform, PlatformInfo, PluginBridge, PluginMessage, PluginResponse, Point, RotationGestureEvent, Router, RouterConfig, ScreenAnimation, ScreenDefinition, ScreenOptions, ScreenPresentation, Signal, SplashScreen, StackNavigatorOptions, SwipeDirection, SwipeGestureConfig, SwipeGestureEvent, TabNavigatorOptions, TapGestureConfig, TapGestureEvent, TauriConfig, TauriEvent, Velocity, WindowOptions, addLifecycleListener, appName, appState, appVersion, callPlugin, centerWindow, closeWindow, createCommand, createGestureHandler, createLongPressGesture, createNativeApp, createNativeRouter, createNativeStack, createNativeTabs, createPanGesture, createPinchGesture, createRotationGesture, createSwipeGesture, createTapGesture, createWindow, currentWindowLabel, detectPlatform, dimensions, emit, exitApp, getAllWindows, getAngle, getAppInfo, getAppName, getAppVersion, getCapacitor, getCapacitorPlatform, getCenter, getCurrentWindow, getDeviceInfo, getDimensions, getDistance, getLaunchUrl, getNativeComponent, getPlatformInfo, getPlugin, getRegisteredComponents, getTauriInternals, getTauriInvoke, getTauriVersion, getVelocity, hasNativeComponent, hasPlugin, hasTauriAPI, hideApp, initCapacitor, initTauri, invoke, invokeSafe, isCapacitor, isNativePlatform, isTauri, listen, maximizeWindow, minimizeWindow, nativeBridge, navigationState, onPlatform, once, openURL, platformInfo, platformSelect, pluginBridge, registerNativeComponent, registerPlugin, restartApp, setAlwaysOnTop, setFullscreen, setPosition, setSize, setTitle, showApp, tauriVersion, toggleMaximize, unmaximizeWindow, useAppState, useBackPressed, useFocusEffect, useGestures, usesFocused, useLongPressGesture, useNativeNavigation, useOnPause, useOnResume, usePanGesture, usePinchGesture, useRoute, useSwipeGesture, useTapGesture, windowFocused, windowMaximized, windowMinimized
- Re-exported names: // ActivityIndicator ActivityIndicatorProps, // Additional ActivityIndicator, // App createNativeApp, // Biometrics BiometricType, // Biometrics Biometrics, // Button ButtonProps, // Camera Camera, // Camera CameraType, // Clipboard Clipboard, // Clipboard ClipboardContentType, // Color Scheme colorScheme, // Common commonStyles, // Components Link, // Core View, // FlatList FlatListProps, // Geolocation Geolocation, // Geolocation LocationPermissionStatus, // Haptics Haptics, // Haptics ImpactStyle, // Hooks useNativeNavigation, // Image ImageProps, // Input TextInput, // Input TextInputProps, // Layout SafeAreaView, // Native Bridge NativeBridge, // Notifications NotificationPermissionStatus, // Notifications Notifications, // Platform detectPlatform, // Platform Styles platformStyles, // Responsive breakpoints, // Router createNativeRouter, // SafeArea SafeAreaViewProps, // ScrollView ScrollViewProps, // Scrolling ScrollView, // Share Share, // Share ShareContent, // Spring Spring, // Stack createNativeStack, // StatusBar StatusBarProps, // Storage Storage, // Storage StorageOptions, // Style Types FlexAlignType, // StyleSheet StyleSheet, // Styles ViewStyle, // Tabs createNativeTabs, // Text TextProps, // Theme ThemeColors, // Theme lightTheme, // Touch TouchableOpacity, // Touch TouchableOpacityProps, // Transitions Transitions, // View ViewProps, AccessibilityRole, ActivityIndicator, ActivityIndicatorProps, ActivityIndicatorSize, AdvancedNavigationState, AdvancedScreenOptions, AdvancedStackNavigatorOptions, AdvancedTabNavigatorOptions, Animated, AnimatedInterpolation, AnimatedValue, AnimatedValueXY, AnimationConfig, AnimationResult, AppLifecycleEvents, AppState, AppStateInfo, AppStateType, Appearance, AppearanceChangeHandler, AppearancePreferences, AuthenticationOptions, AuthenticationResult, AutoCapitalize, BaseGestureEvent, BatchCallEntry, BatteryStatus, BatteryThreshold, BiometricSupport, BorderStyle, Bridge, Button, ButtonProps, CameraPermissionStatus, CameraProps, ChainConfig, ClipboardContent, ColorScheme, ColorSchemeName, ColorValue, CompositeAnimation, ConnectionType, DateComponents, DecayConfig, DeepLinkConfig, DevMenuItem, DevMenuOptions, DevTools, DeviceInfo, DeviceInfoType, DimensionChangeHandler, DimensionMetrics, DimensionValue, Dimensions, DimensionsData, DismissKeyboard, DismissKeyboardProps, Display, DrawerNavigatorOptions, Easing, EffectiveConnectionType, EventSubscription, FlashMode, FlatList, FlatListProps, FlatListRef, FlexDirection, FlexJustifyType, FlexWrap, FontWeight, FrameTiming, GeocodingResult, GestureConfig, GestureHandlerOptions, GestureResponderEvent, GestureSpringConfig, GestureSpringController, GestureState, HMRCConfig, HeadingResult, HotReloadStatus, Image, ImageErrorEvent, ImageLoadEvent, ImagePriority, ImageProgressEvent, ImageProps, ImageSource, ImageStyle, InterpolationConfig, JSIBinding, Keyboard, KeyboardAvoidingView, KeyboardAvoidingViewProps, KeyboardBehavior, KeyboardDismissMode, KeyboardInfo, KeyboardType, LayoutEvent, LazyModule, LinkProps, ListRenderItemInfo, LocationAccuracy, LocationCoordinates, LocationOptions, LocationRegion, LocationResult, LogLevel, LongPressGestureConfig, LongPressGestureEvent, MMKVStorage, MemoryInfo, MemoryWarning, MemoryWarningLevel, ModuleUpdate, MultiGetResult, MultiSetInput, NamedStyles, NativeApp, NativeAppConfig, NativeBridgeMessage, NativeBridgeResponse, NativeCallback, NativeComponentConfig, NativeEventEmitter, NativeModule, NativeStack, NativeStyle, NativeTabs, Navigation, NavigationAction, NavigationAnimation, NavigationEvent, NavigationObject, NavigationOptions, NavigationRoute, NavigationState, NetworkStatus, NotificationAction, NotificationActionOptions, NotificationAttachment, NotificationCategory, NotificationCategoryOptions, NotificationContent, NotificationResponse, NotificationTrigger, NotificationType, OrientationBreakpoint, OrientationInfo, OrientationLockType, OrientationType, Overflow, PanGestureConfig, PanGestureEvent, Performance, PerformanceEntry, PerformanceMark, PerformanceMeasure, PermissionGroups, PermissionRationale, PermissionResult, PermissionStatus, PermissionType, Permissions, PermissionsResult, PhotoOptions, PhotoResult, PinchGestureConfig, PinchGestureEvent, Platform, PlatformConstants, PlatformInfo, PlatformOS, PlatformSelectSpecifics, PlatformStatic, Point, PointerEvents, PositionType, PresentationStyle, Pressable, PressableProps, PullToRefreshConfig, PushToken, RefreshControl, RefreshControlProps, RefreshIndicator, RefreshIndicatorProps, RefreshState, ResizeMode, ReturnKeyType, RotationGestureEvent, Route, RouteParams, RouterConfig, SafeAreaEdge, SafeAreaInsets, SafeAreaProvider, SafeAreaProviderProps, SafeAreaView, SafeAreaViewProps, ScheduledNotification, Screen, ScreenAnimation, ScreenDefinition, ScreenOptions, ScreenPresentation, ScreenProps, ScreenTransitionConfig, ScrollEvent, ScrollIndicatorInsets, ScrollView, ScrollViewProps, ScrollViewRef, SecureStorage, SecureStoreOptions, ShareFileContent, ShareOptions, ShareResult, Spring2DController, Spring2DState, SpringConfig, SpringController, SpringPhysicsConfig, SpringPresets, SpringState, StackNavigatorOptions, StatusBar, StatusBarAnimation, StatusBarProps, StatusBarStyle, StyleProp, SwipeDirection, SwipeGestureConfig, SwipeGestureEvent, Switch, SwitchProps, SystemColors, TabNavigatorOptions, TapGestureConfig, TapGestureEvent, Text, TextAlign, TextContentType, TextDecorationLine, TextInput, TextInputChangeEvent, TextInputContentSizeChangeEvent, TextInputFocusEvent, TextInputKeyPressEvent, TextInputProps, TextInputRef, TextInputSelection, TextInputSelectionChangeEvent, TextLayoutEvent, TextLine, TextProps, TextStyle, Theme, ThemeSpacing, ThemeTypography, Touch, TouchEvent, TouchableHighlight, TouchableHighlightProps, TouchableOpacity, TouchableOpacityProps, TouchableWithoutFeedback, Transform, TransformTransition, TransformValues, TransitionConfig, TurboModuleSpec, Velocity, VideoOptions, VideoResult, View, ViewProps, ViewToken, ViewabilityConfig, VirtualizationConfig, VirtualizedRange, add, addAppStateListener, addBackgroundListener, addForegroundListener, addNetworkListener, appStateInfo, appearanceColorScheme, batchNativeCalls, batteryStatus, biometricSupport, bottomSheet, breakpoint, buildPath, calculateVirtualizedRange, callNativeMethod, callNativeMethodWithCallback, canRequest, chainTransitions, checkGroup,

```

checkMultiplePermissions, checkPermission, clearDevCache, clearPerformanceData, clipboardContent, colorForScheme, createAdvancedNativeStack,
createAdvancedNativeTabs, createAnimatedComponent, createDrawerNavigator, createEventHooks, createFlatListRef, createGestureHandler, createGestureSpring,
createInteractionHandle, createLongPressGesture, createNativeDrawer, createNativeEventEmitter, createNativeModuleWrapper, createPanGesture, createPinchGesture,
createRotationGesture, createScreenTransition, createScrollViewRef, createSpring, createSpring2D, createSpringChain, createStackNavigator, createSwipeGesture,
createTabNavigator, createTapGesture, createTextInputRef, createTransition, createVirtualizedList, currentBreakpoint, currentLocation, currentTheme, darkTheme, debounce,
decay, defaultBreakpoints, devLog, devModeEnabled, deviceError, deviceInfo, deviceLoading, deviceOrientation, deviceType, diffClamp, dimensions, dispatchViewCommand,
divide, durations, dynamicColor, easings, endBatch, event, executeNative, fade, fadeln, fadeOut, fadeQuick, findNodeHandle, flipInX, flipInY, formatDuration, getAngle,
getAppState, getAspectRatio, getBatteryCategory, getBatteryColor, getBatteryStatus, getBatteryStatusText, getBreakpoint, getCenter, getConnectionQuality,
getDeviceInfoSync, getDeviceOrientation, getDeviceType, getDimensions, getDistance, getFontSizeValue, getFrameStats, getJSON, getMeasures, getMemoryInfo,
getNativeComponent, getNativeConstants, getNativeModule, getNetworkStatus, getOrientation, getOrientationBreakpoint, getPixelRatio, getPlatformInfo,
getPowerSavingRecommendations, getRegisteredComponents, getTimeRemaining, getTimeSinceActive, getTotalActiveTime, getTotalBackgroundTime, getTurboModule,
getVelocity, hasNativeComponent, hasTouchScreen, hideDevMenu, hideErrorOverlay, hidePerformanceMonitor, highContrast, hotReloadStatus, impactHeavy, impactLight,
impactMedium, initHMR, installUISBinding, isAndroid, isAndroidDevice, isBatterySupported, isBlocked, isBreakpointDown, isBreakpointUp, isDarkMode, isDesktop,
isDevelopment, isGranted, isIOS, isIOSDevice, isLandscape, isModuleAvailable, isNative, isPortrait, isProduction, isSquareScreen, isUnavailable, isWeb, isWebDevice,
keyboardState, lastUpdatedModules, lazy, locationError, locationPermission, lockOrientation, lockToLandscape, lockToPortrait, loop, mark, materialSharedAxis, measure,
measureView, memoize, memoryPressure, memoryWarning, modalDismiss, modalPresent, modulo, multiply, nativeBridge, navigationState, networkStatus,
notificationPermission, notifyError, notifySuccess, notifyWarning, onPlatform, openSettings, orientation, parallel, parseRoute, pixelRatio, platformInfo, platformOS,
platformSelect, platformVersion, popIn, preload, preloadImage, preloadImages, promisesifyNativeMethod, pushToken, reducedMotion, refreshBattery, refreshNetworkStatus,
registerDevMenuItem, registerNativeComponent, registerNativeModule, requestGroup, requestMultiplePermissions, requestPermission, resetColorScheme, responsive,
reverseTransition, rotateIn, rotateOut, roundToNearestPixel, runAfterInteractions, safeAreaInsets, scaleIn, scaleOut, scheduleAnimationFrame, scheduleIdleCallback,
scheduleUpdate, screenDimensions, screenFade, selectionFeedback, sequence, setJSON, setTheme, setupDeepLinking, setupDevKeyboardShortcuts, shouldSaveData,
shouldSavePower, showDevMenu, showPerformanceMonitor, slideInDown, slideInLeft, slideInRight, slideInUp, slideOutDown, slideOutLeft, slideOutRight, slideOutUp,
spring, springConfigs, stackPop, stackPush, stagger, startBatch, startFrameMonitor, startMemoryMonitor, subtract, supportsNetworkInformation, supportsOrientationLock,
throttle, timing, toggleColorScheme, toggleInspector, unlockOrientation, updateJSON, useAppState, useAppStateEffect, useAppStateType, useAppearance, useBattery,
useBatteryLevel, useBatteryPercentage, useBatteryThreshold, useBiometrics, useBreakpoint, useClipboard, useColorScheme, useConnectionType, useDevice,
useDeviceOrientation, useDevicePlatform, useDeviceProperty, useDeviceType, useDimensions, useFocusEffect, useFontSize, useGestures, useHighContrast, usesActive,
usesBackground, usesCharging, usesDarkMode, usesDesktopDevice, usesDeviceType, usesFocused, usesLandscape, usesLowBattery, usesMobile, usesOffline,
usesOnline, usesPlatform, usesPortrait, usesKeyboard, useKeyboardEffect, usesLocation, useLongPressGesture, useMemoryWarning, useNavigation,
useNavigationFocusEffect, useNavigationIsFocused, useNavigationRoute, useNetwork, useNetworkChange, useNotificationReceived, useNotificationResponse,
useOnAppStateChange, useOnBackground, useOnChargingChange, useOnCriticalBattery, useOnForeground, useOnMemoryWarning, useOnOffline, useOnOnline,
useOrientation, useOrientationAngle, useOrientationBreakpoint, useOrientationEffect, useOrientationForScreen, useOrientationLock, usePanGesture, usePermission,
usePermissions, usePinchGesture, usePixelRatio, usePlatform, usePlatformColorScheme, usePlatformValue, usePullToRefresh, useReducedMotion, useRefresh, useRoute,
useSafeAreaFrame, useSafeAreaInsets, useScreenDimensions, useSpring, useSpring2D, useStorage, useSwipeGesture, useSwitchState, useTapGesture, useTheme,
useThemedStyles, useWatchLocation, useWindowDimensions, wasRecentlyBackgrounded, windowDimensions, zoomIn, zoomOut

```

- Re-exported modules: ./ActivityIndicator.js, ./Button.js, ./FlatList.js, ./Image.js, ./KeyboardAvoidingView.js, ./RefreshControl.js, ./SafeAreaView.js, ./ScrollView.js, ./StatusBar.js,
./Switch.js, ./Text.js, ./TextInput.js, ./TouchableOpacity.js, ./View.js, ./animations/index.js, ./apis/Permissions.js, ./apis/index.js, ./bridge/index.js, ./commands.js,
./components/KeyboardAvoidingView.js, ./components/RefreshControl.js, ./components/Switch.js, ./components/index.js, ./devtools/index.js, ./dialog.js, ./events.js, ./fs.js,
./gestures/index.js, ./hooks/index.js, ./navigation/index.js, ./navigation/index.js, ./performance/index.js, ./platform/index.js, ./plugins/index.js, ./runtime.js, ./styles.js, ./window.js

License

MIT

@philjs/neural - Type: Node package - Purpose: Neural Rendering Engine - AI-powered rendering optimization with predictive frames, adaptive quality, and neural layout - Version: 0.1.0 - Location: packages/philjs-neural - Entry points: packages/philjs-neural/src/index.ts - Keywords: philjs, neural, rendering, ai, machine-learning, optimization, predictive, adaptive-quality, performance

@philjs/neural

Neural Rendering Engine for PhilJS - AI-powered rendering optimization using neural networks for predictive frame rendering, adaptive quality scaling, smart component prioritization, and neural layout optimization.

Installation

```
npm install @philjs/neural
```

Requirements

- Node.js >= 24
- TypeScript 6.x (for development)
- Modern browser with `requestAnimationFrame` and `IntersectionObserver` support

Basic Usage

```

import {
  initNeuralRenderer,
  useNeuralRendering,
  useAdaptiveQuality,
  usePredictiveRendering
} from '@philjs/neural';

// Initialize the neural renderer
const renderer = initNeuralRenderer({
  predictiveRendering: true,
  targetFPS: 60,
  adaptiveQuality: true,
  modelSize: 'small',
  memoryBudget: 50 // MB
});

// Start the renderer
renderer.start();

// Use in components
function MyComponent() {
  const { quality, priority, recordRender, recordInteraction } = useNeuralRendering('my-component');

  // Adjust rendering based on quality settings
  const showShadows = quality.enableShadows;
  const showAnimations = quality.enableAnimations;

  // Record interactions for priority Learning
  const handleClick = () => {
    recordInteraction();
    // ... handle click
  };
}

```

Adaptive Quality

```

import { useAdaptiveQuality } from '@philjs/neural';

function QualityAwareComponent() {
  const { level, settings } = useAdaptiveQuality();

  // Level: 0.0 - 1.0 (current quality level)
  // settings: detailed quality configuration

  return (
    <div style={{ 
      boxShadow: settings.enableShadows ? '0 4px 6px rgba(0,0,0,0.1)' : 'none',
      filter: settings.enableBlur ? 'blur(0)' : 'none'
    }}>
      {settings.enableAnimations && <AnimatedContent />}
      <img
        src={`/image-${settings.imageQuality}.jpg`}
        style={{ transform: `scale(${settings.renderResolution})` }}
      />
    </div>
  );
}

```

Predictive Rendering

```

import { usePredictiveRendering, getNeuralRenderer } from '@philjs/neural';

// Get predictions for which components will render next
const predictions = usePredictiveRendering(['header', 'sidebar', 'content', 'footer']);

predictions.forEach(pred => {
  console.log(`Component: ${pred.componentId}`);
  console.log(`Probability: ${pred.probability}`);
  console.log(`Estimated render time: ${pred.estimatedRenderTime}ms`);
  console.log(`Priority: ${pred.priority}`);
  console.log(`Should prerender: ${pred.shouldPrerender}`);
});

// Record actual renders to improve predictions
const renderer = getNeuralRenderer();
renderer?.recordRender('my-component', 5.2, 'state');

```

Layout Optimization

```

import { useLayoutOptimization, getNeuralRenderer } from '@philjs/neural';

// Record element layouts
const renderer = getNeuralRenderer();
renderer?.recordLayout('element-1', { x: 0, y: 0, width: 200, height: 100 });
renderer?.recordLayout('element-2', { x: 0, y: 100, width: 200, height: 150 });

// Get optimization suggestions
const elements = new Map([
  ['element-1', { x: 0, y: 0, width: 200, height: 100 }],
  ['element-2', { x: 0, y: 100, width: 200, height: 150 }]
]);

const suggestions = useLayoutOptimization(elements);

suggestions.forEach(suggestion => {
  console.log(`Element: ${suggestion.elementId}`);
  console.log(`  Suggested changes: ${suggestion.suggestedLayout}`);
  console.log(`  Performance gain: ${suggestion.performanceGain}%`);
  console.log(`  Reason: ${suggestion.reason}`);
});

```

Component Prioritization

```

import { getNeuralRenderer } from '@philjs/neural';

const renderer = getNeuralRenderer();

// Observe components for visibility tracking
const element = document.getElementById('my-component');
renderer?.observeComponent(element, 'my-component');

// Record user interactions
renderer?.recordInteraction('my-component');

// Get priority rankings
const priorities = renderer?.getComponentPriorities(['comp-a', 'comp-b', 'comp-c']);
// Returns: [{ id, visibility, interactionLikelihood, renderCost, priority }]

```

API Reference

Initialization

- `initNeuralRenderer(config?: NeuralRendererConfig): NeuralRenderer` - Initialize the neural renderer
- `getNeuralRenderer(): NeuralRenderer | null` - Get the global renderer instance

Hooks

- `useNeuralRendering(componentId: string)` - Get quality settings and recording functions for a component
- `useAdaptiveQuality()` - Get current quality level and settings
- `usePredictiveRendering(componentIds: string[])` - Get render predictions
- `useLayoutOptimization(elements: Map<string, LayoutMetrics>)` - Get layout optimization suggestions

Classes

`NeuralRenderer`

Main neural rendering engine.

Methods: - `start()` - Start the render loop - `stop()` - Stop the render loop - `recordRender(componentId, renderTime, triggerType, parentId?)` - Record a component render - `predictNextRenders(componentIds)` - Predict which components will render - `getQualitySettings()` - Get current quality settings - `getQualityLevel()` - Get quality level (0-1) - `observeComponent(element, componentId)` - Start observing a component - `unobserveComponent(element)` - Stop observing - `recordInteraction(componentId)` - Record user interaction - `getComponentPriorities(componentIds)` - Get priority rankings - `recordLayout(elementId, metrics)` - Record layout metrics - `getLayoutSuggestions(elements)` - Get optimization suggestions - `analyzeFrame()` - Get current frame analysis - `destroy()` - Clean up resources

`RenderPredictor`

Neural network-based render prediction.

`AdaptiveQualityManager`

Manages quality scaling based on performance.

`ComponentPrioritizer`

Tracks component visibility and interactions.

`NeuralLayoutOptimizer`

Provides layout optimization suggestions.

`Tensor / NeuralNetwork`

Low-level neural network primitives.

`Configuration`

```

interface NeuralRendererConfig {
  predictiveRendering?: boolean; // Enable predictions (default: true)
  targetFPS?: number; // Target frame rate (default: 60)
  adaptiveQuality?: boolean; // Enable quality scaling (default: true)
  modelSize?: 'tiny' | 'small' | 'medium'; // Neural model size (default: 'small')
  offlineMode?: boolean; // Enable offLine inference (default: true)
  useGPU?: boolean; // GPU acceleration (default: false)
  memoryBudget?: number; // Memory Limit in MB (default: 50)
}

```

Types

- RenderPrediction - Prediction result with probability, time estimate, and priority
- FrameAnalysis - Frame metrics including jank score and memory usage
- NeuralLayoutSuggestion - Layout optimization recommendation
- LayoutMetrics - Element position and size with optional CSS hints
- ComponentPriority - Component ranking with visibility and interaction scores

Quality Settings

The adaptive quality manager provides these settings:

```

{
  enableShadows: boolean; // Enable box shadows
  enableAnimations: boolean; // Enable CSS animations
  imageQuality: 'low' | 'medium' | 'high';
  renderResolution: number; // 0.3 - 1.0
  enableBlur: boolean; // Enable blur effects
}

```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-neural/src/index.ts

Public API

- Direct exports: AdaptiveQualityManager, ComponentPrioritizer, ComponentPriority, FrameAnalysis, LayoutMetrics, NeuralLayoutOptimizer, NeuralLayoutSuggestion, NeuralNetwork, NeuralRenderer, NeuralRendererConfig, RenderPrediction, RenderPredictor, Tensor, getNeuralRenderer, initNeuralRenderer, useAdaptiveQuality, useLayoutOptimization, useNeuralRendering, usePredictiveRendering
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/observability - Type: Node package - Purpose: Observability for PhilJS - distributed tracing, metrics, logging, and error tracking - Version: 0.1.0 - Location: packages/philjs-observability - Entry points: packages/philjs-observability/src/index.ts, ./tracing, ./metrics, ./logging, ./sentry, ./datadog - Keywords: philjs, observability, tracing, metrics, logging, opentelemetry, sentry

@philjs/observability

Monitoring and observability tools for PhilJS applications. Track performance, errors, and user behavior with built-in dashboards and alerting.

Requirements

- **Node.js 24** or higher
- **TypeScript 6** or higher
- **ESM only** - CommonJS is not supported

Installation

```
pnpm add @philjs/observability
```

Basic Usage

```

import {
  ObservabilityProvider,
  useMetrics,
  PerformanceDashboard
} from '@philjs/observability';

function App() {
  return (
    <ObservabilityProvider
      endpoint="/api/metrics"
      sampleRate={0.1}
    >
      <MyApp />
    </ObservabilityProvider>
  );
}

function Dashboard() {
  const { trackEvent, trackError } = useMetrics();

  const handleClick = () => {
    trackEvent('button_clicked', { button: 'submit' });
  };

  return <PerformanceDashboard />;
}

```

Features

- **Performance Monitoring** - Track Core Web Vitals and custom metrics
- **Error Tracking** - Capture and report JavaScript errors
- **User Analytics** - Track user behavior and interactions
- **Custom Metrics** - Define and track business metrics
- **Dashboards** - Pre-built visualization dashboards
- **Alerting** - Configure alerts for metric thresholds
- **Distributed Tracing** - Track requests across services
- **Session Replay** - Replay user sessions for debugging
- **Network Monitoring** - Track API latency and errors
- **Real User Monitoring** - Measure real user experience
- **Sparklines** - Compact inline metric visualizations
- **Gauge Charts** - Display current metric values

Components

Component	Description
PerformanceDashboard	Web Vitals dashboard
WebVitalsDashboard	Core Web Vitals display
NetworkWaterfallPanel	Network request timeline
AlertsConfigPanel	Alert configuration UI
ComponentRenderPanel	Component render metrics
Sparkline	Inline metric chart
GaugeChart	Circular metric gauge

Hooks

Hook	Description
useMetrics	Track events and errors
usePerformance	Access performance data
useWebVitals	Core Web Vitals metrics
useErrorBoundary	Error tracking boundary

Metrics API

```

import { metrics } from '@philjs/observability';

metrics.increment('page_views');
metrics.gauge('active_users', 150);
metrics.histogram('response_time', 234);
metrics.timing('api_call', startTime);

```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: ., ./tracing, ./metrics, ./logging, ./sentry, ./datadog

- Source files: packages/philjs-observability/src/index.ts

Public API

- Direct exports: ConsoleTransport, ErrorTracker, ErrorTrackerOptions, LogEntry, LogLevel, LogTransport, Logger, LoggerOptions, MetricValue, Metrics, MetricsExporter, MetricsOptions, PerformanceMetrics, Span, SpanEvent, SpanExporter, Tracer, TracerOptions, usePerformance
- Re-exported names: Alert, AlertCondition, AlertManager, AlertManagerConfig, AlertRule, AlertSeverity, AlertState, ComparisonOperator, NotificationChannel, getAlertManager, initAlertManager, presetRules, useAlerts
- Re-exported modules: ./alerting.js, ./charts/index.js, ./dashboard/index.js, ./widgets/index.js

License

MIT

```
## @philjs/offline - Type: Node package - Purpose: Offline-first architecture for PhilJS - IndexedDB, sync, caching strategies - Version: 0.1.0 - Location: packages/philjs-offline - Entry points: packages/philjs-offline/src/index.ts - Keywords: philjs, offline, indexeddb, sync, cache, local-first, pwa
```

@philjs/offline

Offline-first architecture for PhilJS - IndexedDB, sync, caching strategies

Overview

Offline-first architecture for PhilJS - IndexedDB, sync, caching strategies

Focus Areas

- philjs, offline, indexeddb, sync, cache, local-first, pwa

Entry Points

- packages/philjs-offline/src/index.ts

Quick Start

```
import { CacheManager, CacheStrategy, ConflictStrategy } from '@philjs/offline';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- CacheManager
- CacheStrategy
- ConflictStrategy
- NetworkMonitor
- NetworkStatus
- OfflineConfig
- OfflineDB
- OfflineStore
- SyncManager
- SyncOperation
- createOfflineStore
- useCache

Install

```
pnpm add @philjs/offline
```

Usage

```
import { CacheManager, CacheStrategy, ConflictStrategy } from '@philjs/offline';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-offline/src/index.ts

Public API

- Direct exports: CacheManager, CacheStrategy, ConflictStrategy, NetworkMonitor, NetworkStatus, OfflineConfig, OfflineDB, OfflineStore, SyncManager, SyncOperation, createOfflineStore, useCache, useNetworkStatus, useOfflineData, useSync
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/openapi - Type: Node package - Purpose: Automatic OpenAPI spec generation for PhilJS APIs - inspired by Elysia - Version: 0.1.0 - Location: packages/philjs-openapi -
Entry points: packages/philjs-openapi/src/index.ts, packages/philjs-openapi/src/swagger-ui.ts, packages/philjs-openapi/src/cli.ts - Keywords: philjs, openapi, swagger, api, documentation, schema, zod, typescript

philjs-openapi

Automatic OpenAPI specification generation for PhilJS APIs. Inspired by [Elysia's](#) elegant approach to API documentation.

Features

- **Automatic OpenAPI spec generation** from route definitions
- **Zod schema integration** - automatic JSON Schema conversion
- **Swagger UI middleware** - serve interactive docs
- **ReDoc support** - alternative documentation viewer
- **Type generation CLI** - generate TypeScript types from OpenAPI specs
- **Security schemes** - built-in helpers for common auth patterns
- **Route groups** - organize routes with tags and shared config

Installation

```
npm install philjs-openapi zod
```

Quick Start

```

import { createAPI, openapi, swaggerUI } from 'philjs-openapi';
import { z } from 'zod';

// Define your schemas
const UserSchema = z.object({
  id: z.string().uuid(),
  name: z.string().min(1).max(100),
  email: z.string().email(),
  role: z.enum(['admin', 'user']).default('user'),
  createdAt: z.string().datetime(),
});

const CreateUserSchema = z.object({
  name: z.string().min(1).max(100),
  email: z.string().email(),
  role: z.enum(['admin', 'user']).optional(),
});

// Create API with automatic documentation
const api = createAPI({
  'GET /users': {
    response: z.array(UserSchema),
    summary: 'List all users',
    description: 'Returns a paginated list of all users',
    tags: ['Users'],
    handler: async () => fetchUsers(),
  },

  'POST /users': {
    body: CreateUserSchema,
    response: UserSchema,
    summary: 'Create a new user',
    tags: ['Users'],
    handler: async ({ body }) => createUser(body),
  },

  'GET /users/:id': {
    params: z.object({
      id: z.string().uuid().describe('User ID'),
    }),
    response: UserSchema,
    summary: 'Get user by ID',
    tags: ['Users'],
    handler: async ({ params }) => fetchUser(params.id),
  },

  'PUT /users/:id': {
    params: z.object({ id: z.string().uuid() }),
    body: CreateUserSchema.partial(),
    response: UserSchema,
    summary: 'Update user',
    tags: ['Users'],
    handler: async ({ params, body }) => updateUser(params.id, body),
  },

  'DELETE /users/:id': {
    params: z.object({ id: z.string().uuid() }),
    response: z.object({ success: z.boolean() }),
    summary: 'Delete user',
    tags: ['Users'],
    handler: async ({ params }) => deleteUser(params.id),
  },
});

// Generate OpenAPI spec
const spec = openapi(api, {
  info: {
    title: 'My API',
    version: '1.0.0',
    description: 'A sample API with automatic OpenAPI documentation',
  },
  servers: [
    { url: 'https://api.example.com', description: 'Production' },
    { url: 'http://localhost:3000', description: 'Development' },
  ],
});

// Serve documentation
app.get('/docs', swaggerUI({ spec }));
app.get('/openapi.json', () => Response.json(spec));

```

Route Definition

Routes are defined using a simple format: "METHOD /path".

```

const api = createAPI({
  // GET request
  'GET /posts': { ... },

  // POST request
  'POST /posts': { ... },

  // Path parameters (use :param syntax)
  'GET /posts/:id': { ... },

  // Multiple path parameters
  'GET /users/:userId/posts/:postId': { ... },

  // Default to GET if no method specified
  '/health': { ... },
});

```

Route Options

```

interface APIRouteDefinition {
  // Zod schemas for validation
  params?: ZodSchema; // Path parameters
  query?: ZodSchema; // Query parameters
  body?: ZodSchema; // Request body
  headers?: ZodSchema; // Request headers
  response?: ZodSchema; // Response body

  // Documentation
  summary?: string;
  description?: string;
  tags?: string[];;
  deprecated?: boolean;
  operationId?: string;

  // Security
  security?: SecurityRequirement[];

  // Examples
  examples?: {
    request?: unknown;
    response?: unknown;
  };

  // Handler function
  handler: (context) => Promise<unknown>;
}

```

Query Parameters

```

const api = createAPI({
  'GET /users': {
    query: z.object({
      page: z.coerce.number().min(1).default(1),
      limit: z.coerce.number().min(1).max(100).default(20),
      search: z.string().optional(),
      role: z.enum(['admin', 'user']).optional(),
    }),
    response: z.object({
      users: z.array(UserSchema),
      total: z.number(),
      page: z.number(),
      limit: z.number(),
    }),
    handler: async ({ query }) => {
      return fetchUsers(query);
    },
  },
});

```

Multiple Response Status Codes

```

const api = createAPI({
  'POST /users': {
    body: CreateUserSchema,
    response: {
      201: UserSchema.describe('User created successfully'),
      400: z.object({
        error: z.string(),
        errors: z.record(z.array(z.string())).optional(),
      }).describe('Validation error'),
      409: z.object({
        error: z.string(),
      }).describe('User already exists'),
    },
    handler: async ({ body }) => createUser(body),
  },
});

```

Route Groups

Organize related routes with shared configuration:

```

import { group, openapi } from 'philjs-openapi';

const userRoutes = group({
  name: 'Users',
  description: 'User management endpoints',
  basePath: '/api/v1',
  security: [{ bearerAuth: [] }],
  routes: {
    'GET /users': { ... },
    'POST /users': { ... },
    'GET /users/:id': { ... },
  },
});

const postRoutes = group({
  name: 'Posts',
  description: 'Blog post endpoints',
  basePath: '/api/v1',
  routes: {
    'GET /posts': { ... },
    'POST /posts': { ... },
  },
});

// Generate spec from groups
const spec = openapi([userRoutes, postRoutes], {
  info: { title: 'My API', version: '1.0.0' },
});

```

Security Schemes

Built-in helpers for common authentication patterns:

```

import { openapi, securitySchemes } from 'philjs-openapi';

const spec = openapi(api, {
  info: { title: 'My API', version: '1.0.0' },
  securitySchemes: {
    bearerAuth: securitySchemes.bearer('JWT'),
    apiKey: securitySchemes.apiKey('X-API-Key', 'header'),
    basicAuth: securitySchemes.basic(),
    oauth2: securitySchemes.oauth2AuthorizationCode({
      authorizationUrl: 'https://auth.example.com/authorize',
      tokenUrl: 'https://auth.example.com/token',
      scopes: {
        'read:users': 'Read user data',
        'write:users': 'Create and update users',
      },
    }),
    security: [{ bearerAuth: [] }], // Global security
  },
});

```

Apply security to specific routes:

```

const api = createAPI({
  'GET /public': {
    security: [], // No authentication required
    handler: async () => ({ status: 'ok' }),
  },
  'GET /admin': {
    security: [{ bearerAuth: [], apiKey: [] }],
    handler: async () => ({ admin: true }),
  },
});

```

Swagger UI & ReDoc

Serve interactive documentation:

```
import { swaggerUI, redoc, createDocsRoutes } from 'philjs-openapi';

// Swagger UI
app.get('/docs', swaggerUI({
  spec,
  title: 'API Documentation',
  config: {
    tryItOutEnabled: true,
    persistAuthorization: true,
    displayRequestDuration: true,
  },
}));

// ReDoc (alternative viewer)
app.get('/redoc', redoc({
  spec,
  title: 'API Reference',
  config: {
    expandResponses: '200,201',
    pathInMiddlePanel: true,
  },
}));

// Or create all routes at once
const docsRoutes = createDocsRoutes(spec, {
  title: 'My API',
  basePath: '/api',
});
// Creates:
// - /api/docs (Swagger UI)
// - /api/redoc (ReDoc)
// - /api/openapi.json (spec)
```

CLI: Generate Types from OpenAPI

Generate TypeScript types from any OpenAPI specification:

```
# Basic usage
npx philjs-openapi generate --input openapi.json --output types.ts

# From URL
npx philjs-openapi generate --input https://api.example.com/openapi.json --output types.ts

# With API client generation
npx philjs-openapi generate --input openapi.json --output types.ts --client

# With Zod schemas
npx philjs-openapi generate --input openapi.json --output types.ts --zod

# ALL options
npx philjs-openapi generate \
  --input openapi.json \
  --output ./src/api/types.ts \
  --client \
  --zod \
  --prefix Api \
  --readonly \
  --enums-as-union
```

Generated output:

```
// types.ts

/**
 * Generated from My API v1.0.0
 * Do not edit manually.
 */

// Schemas
export type User = {
  id: string;
  name: string;
  email: string;
  role: 'admin' | 'user';
  createdAt: string;
};

export type CreateUser = {
  name: string;
  email: string;
  role?: 'admin' | 'user';
};

// Operations
export interface GetUsersRequest {
  query?: { page?: number; limit?: number; search?: string };
}

export type GetUsersResponse = User[];

export interface CreateUserRequest {
  body: CreateUser;
}

export type CreateUserResponse = User;

// API Client
export interface APIClientConfig {
  baseUrl: string;
  headers?: Record<string, string>;
  fetch?: typeof fetch;
}

export function createClient(config: APIClientConfig) {
  // ... implementation
  return {
    getUsers: async (req: GetUsersRequest): Promise<GetUsersResponse> => { ... },
    createUser: async (req: CreateUserRequest): Promise<CreateUserResponse> => { ... },
    // ... more operations
  };
}
}
```

Zod to JSON Schema

Convert Zod schemas to JSON Schema directly:

```
import { zodToJsonSchema } from 'philjs-openapi';
import { z } from 'zod';

const UserSchema = z.object({
  id: z.string().uuid(),
  name: z.string().min(1).max(100),
  email: z.string().email(),
  age: z.number().int().positive().optional(),
  tags: z.array(z.string()).default([]),
});

const jsonSchema = zodToJsonSchema(UserSchema);
// {
//   type: 'object',
//   properties: {
//     id: { type: 'string', format: 'uuid' },
//     name: { type: 'string', minLength: 1, maxLength: 100 },
//     email: { type: 'string', format: 'email' },
//     age: { type: 'integer', exclusiveMinimum: 0 },
//     tags: { type: 'array', items: { type: 'string' }, default: [] },
//   },
//   required: ['id', 'name', 'email'],
// }
```

Error Responses

Add standard error responses:

```

import { openapi, errorResponses } from 'philjs-openapi';

const spec = openapi(api, {
  info: { title: 'My API', version: '1.0.0' },
  errorResponses: {
    400: errorResponses.badRequest(),
    401: errorResponses.unauthorized(),
    403: errorResponses.forbidden(),
    404: errorResponses.notFound(),
    500: errorResponses.serverError(),
  },
});

```

Advanced Configuration

```

const spec = openapi(api, {
  info: {
    title: 'My API',
    version: '1.0.0',
    description: 'Full API documentation',
    contact: {
      name: 'API Support',
      email: 'support@example.com',
      url: 'https://example.com/support',
    },
    license: {
      name: 'MIT',
      url: 'https://opensource.org/licenses/MIT',
    },
  },
  servers: [
    { url: 'https://api.example.com', description: 'Production' },
    {
      url: 'https://{{environment}}.api.example.com',
      description: 'By environment',
      variables: {
        environment: {
          default: 'prod',
          enum: ['prod', 'staging', 'dev'],
          description: 'Environment name',
        },
      },
    },
  ],
  tags: [
    { name: 'Users', description: 'User management' },
    { name: 'Posts', description: 'Blog posts' },
  ],
  externalDocs: {
    description: 'Full documentation',
    url: 'https://docs.example.com',
  },
  basePath: '/api/v1',
  includeExamples: true,
  operationIdTransform: (method, path) => {
    // Custom operation ID generation
    return `${method}${path.replace(/\//g, '_')}`;
  },
});

```

API Reference

`createAPI(routes)`

Create an API definition from route handlers.

`openapi(api, options)`

Generate OpenAPI specification from API definition.

`group(config)`

Create a route group with shared configuration.

`swaggerUI(options)`

Create Swagger UI middleware.

`redoc(options)`

Create ReDoc middleware.

`createDocsRoutes(spec, options)`

Create all documentation routes at once.

`specHandler(spec)`

Create a handler that serves the OpenAPI spec as JSON.

`zodToJsonSchema(schema)`

Convert a Zod schema to JSON Schema.

`generateTypes(spec, options)`

Generate TypeScript types from OpenAPI spec.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `., ./swagger-ui, ./cli`
- Source files: `packages/philjs-openapi/src/index.ts, packages/philjs-openapi/src/swagger-ui.ts, packages/philjs-openapi/src/cli.ts`

Public API

- Direct exports: `createDocsRoutes, createSwaggerRoutes, generateTypes, main, redoc, specHandler, swaggerUI`
- Re-exported names: `// JSON Schema types JSONSchema, // OpenAPI specification types OpenAPISpec, // PhiJS API types APIRouteDefinition, // Type generation types TypeGenerationOptions, APIDefinition, GeneratedTypes, OpenAPIComponents, OpenAPIEncoding, OpenAPIExample, OpenAPIExternalDocs, OpenAPIHeader, OpenAPIInfo, OpenAPILink, OpenAPIMediaType, OpenPIOAuthFlow, OpenPIOAuthFlows, OpenAPIOperation, OpenAPIOptions, OpenAPIParameter, OpenAPIPathItem, OpenAPIRequestBody, OpenAPIResponse, OpenAPISecurityRequirement, OpenAPISecurityScheme, OpenAPIServer, OpenAPITag, RouteContext, RouteGroup, SwaggerUIOptions, createAPI, createDocsRoutes, createSwaggerRoutes, errorResponses, extractExample, generateTypes, getSchemaDescription, group, isZodSchema, mergeAPIs, openapi, redoc, runCLI, securitySchemes, specHandler, swaggerUI, zodToJsonSchema`
- Re-exported modules: `./cli.js, ./openapi.js, ./swagger-ui.js, ./types.js, ./zod-to-schema.js`

License

MIT

@philjs/optimizer - Type: Node package - Purpose: Automatic code optimization and lazy loading for PhiJS - Version: 0.1.0 - Location: packages/philjs-optimizer - Entry points: packages/philjs-optimizer/src/index.ts, packages/philjs-optimizer/src/vite.ts, packages/philjs-optimizer/src/runtime.ts, packages/philjs-optimizer/src/transform.ts - Keywords: philjs, optimizer, lazy-loading, code-splitting, tree-shaking, bundler, performance

PhiJS Optimizer

Qwik-style optimizer and lazy loading for PhiJS. Achieve dramatic bundle size reductions through function-level code splitting and automatic lazy boundaries.

Features

- **Function-level Code Splitting:** Split code at the function level for maximum granularity
- **Automatic Lazy Boundaries:** Automatically detect and create lazy loading boundaries
- **Symbol Extraction:** Extract symbols and build dependency graphs
- **Smart Bundling Strategies:** Multiple strategies for optimal bundling
- **Lazy Event Handlers:** `$(...)` wrapper for automatic handler lazy loading
- **Vite Plugin:** Seamless integration with Vite
- **Progressive Enhancement:** Works without JavaScript, enhanced with it
- **Source Map Support:** Full source map preservation for debugging

Installation

```
npm install philjs-optimizer
```

Quick Start

1. Add Vite Plugin

```
// vite.config.ts
import { defineConfig } from 'vite';
import { philjsOptimizer } from 'philjs-optimizer/vite';

export default defineConfig({
  plugins: [
    philjsOptimizer({
      strategy: 'hybrid',
      minChunkSize: 1024,
      maxChunkSize: 51200,
      debug: true,
    }),
  ],
});
```

2. Use Lazy Handlers

```
import { $ } from 'philjs-core/lazy-handlers';
import { signal } from 'philjs-core';

function Counter() {
  const count = signal(0);

  return (
    <div>
      <h1>Count: {count()}</h1>
      {/* Handler is lazy-loaded only when clicked */}
      <button onClick={() => count.set(count() + 1)}>
        Increment
      </button>
    </div>
  );
}
```

API Reference

Lazy Handlers

`$(handler)` - Lazy Handler

Wraps a function for automatic lazy loading.

```
import { $ } from 'philjs-core/lazy-handlers';

<button onClick={() => console.log('clicked')}>
  Click me
</button>
```

`$(symbolId, handler)` - Named Lazy Handler

Creates a reusable lazy handler with an explicit symbol ID.

```
import { $$ } from 'philjs-core/lazy-handlers';

const handleSubmit = $$('handleContactForm', async (event) => {
  event.preventDefault();
  // Handle form submission
});

<form onSubmit={handleSubmit}>
  ...
</form>
```

`prefetchHandler(symbolId)` - Prefetch Handler

Prefetch a handler before it's needed.

```
import { prefetchHandler } from 'philjs-core/lazy-handlers';

<button
  onClick={expensiveHandler}
  onMouseEnter={() => prefetchHandler('expensiveHandler')}
>
  Run
</button>
```

Optimizer API

`createOptimizer(options)`

Create an optimizer instance.

```
import { createOptimizer } from 'philjs-optimizer';

const optimizer = createOptimizer({
  rootDir: '/path/to/project',
  lazy: true,
  minChunkSize: 1024,
  maxChunkSize: 51200,
});

// Process a file
const result = await optimizer.processFile(code, filePath);

// Build dependency graph
const graph = optimizer.buildGraph();

// Bundle with strategy
const chunks = optimizer.bundle('hybrid');
```

Symbol Extraction

```
import { extractSymbols, SymbolRegistry } from 'philjs-optimizer';

const symbols = extractSymbols(code, filePath, options);
const registry = new SymbolRegistry();

symbols.forEach(symbol => registry.add(symbol));
```

Dependency Graph

```
import {
  buildDependencyGraph,
  topologicalSort,
  detectCircularDependencies
} from 'philjs-optimizer';

const graph = buildDependencyGraph(symbols);
const sorted = topologicalSort(graph);
const cycles = detectCircularDependencies(graph);
```

Bundling Strategies

Available Strategies

- **default**: Group by type and file
- **aggressive**: Each symbol gets its own chunk
- **conservative**: Minimize chunks, group by cohesion
- **route**: Group by route/page
- **depth**: Group by dependency depth
- **size**: Group to meet size constraints
- **hybrid**: Combines multiple strategies (recommended)

```
import { bundleSymbols, getStrategy } from 'philjs-optimizer';

const strategy = getStrategy('hybrid');
const chunks = bundleSymbols(graph, options, 'hybrid');
```

Runtime API

Symbol Loader

```
import {
  initSymbolLoader,
  loadSymbol,
  prefetchSymbol
} from 'philjs-optimizer/runtime';

// Initialize with manifest
const loader = initSymbolLoader({
  manifest: manifestData,
  baseUrl: 'lazy',
  prefetch: false,
});

// Load a symbol
const symbol = await loadSymbol('symbolId');

// Prefetch symbols
await prefetchSymbol('symbolId');
```

Handler Execution

```
import {
  executeHandler,
  getHandlerRunner
} from 'philjs-optimizer/runtime';

// Execute with error handling
const result = await executeHandler('symbolId', [arg1, arg2]);

// Configure error handling
const runner = getHandlerRunner();
runner.setMaxRetries(3);
runner.onError('symbolId', (error) => {
  console.error('Handler failed:', error);
});
```

Integrations

Router Integration

```
import { lazyRoute, LazyRouteManager } from 'philjs-optimizer/integrations/router';

const routes = [
  lazyRoute({
    path: '/',
    component: () => <HomePage />,
    loader: async () => {
      const data = await fetch('/api/home').then(r => r.json());
      return data;
    },
  }),
  lazyRoute({
    path: '/blog/:slug',
    component: () => <BlogPost />,
    loader: async ({ params }) => {
      const post = await fetch(`/api/blog/${params.slug}`).then(r => r.json());
      return post;
    },
  }),
];
```

Component Lazy Loading

```

import { lazy, Suspense } from 'philjs-optimizer/integrations/component';

const LazyComponent = lazy(() => import('./HeavyComponent'));

function App() {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <LazyComponent />
    </Suspense>
  );
}

```

Store Lazy Loading

```

import { lazyStore, useLazyStore } from 'philjs-optimizer/integrations/store';

const userStore = lazyStore(
  () => import('./stores/user').then(m => m.userStore),
  { user: null }
);

function UserProfile() {
  const store = useLazyStore(userStore);

  return <div>{store?.user?.name}</div>;
}

```

Form Lazy Loading

```

import {
  lazySubmit,
  createLazyForm
} from 'philjs-optimizer/integrations/forms';

const form = createLazyForm(formElement);

form
  .onSubmit(lazySubmit(async (formData) => {
    await fetch('/api/submit', {
      method: 'POST',
      body: formData,
    });
  }))
  .onChange('email', lazyChange((value) => {
    console.log('Email changed:', value);
  }))
  .validate('email', lazyValidate((value) => {
    return value.includes('@') ? null : 'Invalid email';
  }));

```

Bundle Size Improvements

Real-world benchmark results:

Simple App (Counter)

- **Before:** 45 KB
- **After:** 12 KB (73% reduction)
- **Lazy loaded:** 8 KB on interaction

Medium App (Todo List)

- **Before:** 120 KB
- **After:** 35 KB (71% reduction)
- **Lazy loaded:** 25 KB on interaction

Large App (Dashboard)

- **Before:** 350 KB
- **After:** 85 KB (76% reduction)
- **Lazy loaded:** 180 KB progressively

Configuration

Optimizer Options

```

interface OptimizerOptions {
  rootDir: string;
  outDir?: string;
  lazy?: boolean;
  minChunkSize?: number;
  maxChunkSize?: number;
  sourcemap?: boolean;
  patterns?: SymbolPattern[];
  debug?: boolean;
}

```

Vite Plugin Options

```

interface ViteOptimizerOptions {
  strategy?: 'default' | 'aggressive' | 'conservative' | 'route' | 'depth' | 'size' | 'hybrid';
  include?: string | string[];
  exclude?: string | string[];
  sourcemap?: boolean;
  baseUri?: string;
  minChunkSize?: number;
  maxChunkSize?: number;
  debug?: boolean;
}

```

How It Works

1. Symbol Extraction

The optimizer uses Babel to parse your code and extract symbols (functions, components, handlers, etc.).

```

// Input
function Counter() {
  const count = signal(0);
  return <button onClick={() => count.set(count() + 1)}>Click</button>;
}

// Extracted Symbols
// - Counter (component)
// - $handler_123 (lazy handler)

```

2. Dependency Graph

Build a graph of dependencies between symbols.

```

Counter
  > $handler_123
    > count (from signal)

```

3. Bundling Strategy

Group symbols into optimal chunks based on the selected strategy.

```

Chunk 1 (main): Counter, count
Chunk 2 (lazy): $handler_123

```

4. Code Transformation

Transform `$()` calls to lazy handler registrations.

```

// Before
<button onClick={() => count.set(count() + 1)}>

// After
<button data-onclick="$handler_123">

```

5. Runtime Loading

Load handlers on interaction.

```

element.addEventListener('click', async () => {
  const handler = await loadSymbol('$handler_123');
  handler(event);
});

```

Best Practices

1. Use `$()` for Event Handlers

Always wrap event handlers with `$()` for automatic lazy loading.

```

// Good
<button onClick={() => handleClick()}>

// Bad (not lazy)
<button onClick={() => handleClick()}>

```

2. Prefetch on Hover

Prefetch handlers on hover for better UX.

```

<button
  onClick={handler}
  onMouseEnter={() => prefetchHandler('handler')}
>
  Click
</button>

```

3. Use Named Handlers for Reuse

Use `$$()` when you need to reference the same handler multiple times.

```
const handleSubmit = $$('handleSubmit', async () => {
  // ...
});

<form onSubmit={handleSubmit}>
  ...
</form>
```

4. Choose the Right Strategy

- **hybrid**: Best for most applications
- **route**: Best for route-based applications
- **aggressive**: Maximum granularity, more HTTP requests
- **conservative**: Fewer chunks, larger initial bundle

Debugging

Enable debug mode to see optimization stats:

```
philJSOptimizer({
  debug: true,
})
```

Output:

```
--- PhilJS Optimizer Stats ---
Total symbols: 45
Lazy symbols: 23
Chunks: 8
Lazy chunks: 5

Symbol types:
  component: 12
  handler: 23
  function: 10

Chunk sizes:
  Total: 120.45 KB
  Average: 15.06 KB
  Max: 45.23 KB
  Min: 2.34 KB
```

Performance Tips

1. **Lazy load event handlers**: Use `$(...)` for all event handlers
2. **Prefetch critical handlers**: Prefetch on hover/focus
3. **Use route-based splitting**: Split by route for better caching
4. **Monitor chunk sizes**: Keep chunks between 1-50 KB
5. **Enable source maps**: For production debugging

Migration Guide

From Regular Handlers

```
// Before
function Component() {
  const handleClick = () => {
    console.log('clicked');
  };

  return <button onClick={handleClick}>Click</button>;
}

// After
function Component() {
  return (
    <button onClick={() => console.log('clicked')}>
      Click
    </button>
  );
}
```

From Other Frameworks

From Qwik

PhilJS Optimizer uses similar concepts to Qwik:

```
// Qwik
<button onClick={() => console.log('clicked')}>

// PhilJS
<button onClick={() => console.log('clicked')}>
```

From React

```
// React (no lazy loading)
<button onClick={() => handleClick()}>

// PhilJS (lazy loading)
<button onClick={$(() => handleClick())}>
```

Examples

See the [examples](#) directory for:

- Basic usage
- Router integration
- Component lazy loading
- Form handling
- Benchmarks

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `., ./vite, ./runtime, ./transform`
- Source files: `packages/philjs-optimizer/src/index.ts, packages/philjs-optimizer/src/vite.ts, packages/philjs-optimizer/src/runtime.ts, packages/philjs-optimizer/src/transform.ts`

Public API

- Direct exports: `DeferredQueue, HandlerRunner, SymbolLoader, ViteOptimizerOptions, createOptimizer, createSymbolLoader, deferHandler, executeHandler, extractLazyChunks, generateLazyImports, generateManifest, getDeferredQueue, getHandlerRunner, getSymbolLoader, initSymbolLoader, injectHandlerRegistrations, load, loadSymbol, philjsOptimizer, prefetchSymbol, transform`
- Re-exported names: `BundleStrategy, ChunkManifest, DeferredQueue, DependencyGraph, ExtractionContext, HandlerRunner, LazyHandler, OptimizerOptions, RuntimeConfig, Symbol, SymbolLoader, SymbolPattern, SymbolRegistry, SymbolType, TransformResult, aggressiveStrategy, buildDependencyGraph, bundleSymbols, calculateCohesion, calculateDepth, conservativeStrategy, createSymbolLoader, defaultStrategy, deferHandler, depthStrategy, detectCircularDependencies, executeHandler, extractLazyChunks, extractSymbols, findCommonDependencies, findEntryPoints, findLeafNodes, generateLazyImports, generateManifest, generateSymbolId, getAllDependencies, getAllDependents, getDeferredQueue, getHandlerRunner, getStrategy, getSymbolLoader, groupByDepth, hybridStrategy, initSymbolLoader, injectHandlerRegistrations, loadSymbol, prefetchSymbol, routeStrategy, sizeStrategy, topologicalSort, transform`
- Re-exported modules: `./bundler.js, ./dependency-graph.js, ./runtime.js, ./symbols.js, ./transform.js, ./types.js`

License

MIT

`## @philjs/payments - Type: Node package - Purpose: Payment processing for PhilJS - Stripe, PayPal, Square support - Version: 0.1.0 - Location: packages/philjs-payments - Entry points: packages/philjs-payments/src/index.ts, packages/philjs-payments/src/providers/stripe.ts, packages/philjs-payments/src/providers/paypal.ts, packages/philjs-payments/src/providers/square.ts, packages/philjs-payments/src/webhooks/index.ts - Keywords: philjs, payments, stripe, paypal, square, subscriptions`

@philjs/payments

Payment processing integration for React applications. Unified API for Stripe, PayPal, and other payment providers with secure, PCI-compliant components.

Installation

```
npm install @philjs/payments
# or
yarn add @philjs/payments
# or
pnpm add @philjs/payments
```

Basic Usage

```
import {
  PaymentProvider,
  PaymentForm,
  CardElement
} from '@philjs/payments';

function Checkout() {
  const handlePayment = async (paymentMethod) => {
    const result = await processPayment(paymentMethod);
    console.log('Payment result:', result);
  };

  return (
    <PaymentProvider
      provider="stripe"
      publicKey={STRIPE_PUBLIC_KEY}
    >
      <PaymentForm onSubmit={handlePayment}>
        <CardElement />
        <button type="submit">Pay $99.00</button>
      </PaymentForm>
    </PaymentProvider>
  );
}
```

Features

- **Multiple Providers** - Stripe, PayPal, Square, Braintree

- **Card Payments** - Secure credit/debit card processing
- **Digital Wallets** - Apple Pay, Google Pay support
- **Subscriptions** - Recurring billing management
- **Invoicing** - Generate and send invoices
- **Refunds** - Process refunds and cancellations
- **PCI Compliance** - Secure, PCI-DSS compliant components
- **3D Secure** - Strong customer authentication
- **Webhooks** - Handle payment events
- **Multi-Currency** - Support for 135+ currencies
- **Tax Calculation** - Automatic tax computation
- **Fraud Detection** - Built-in fraud prevention

Components

Component	Description
PaymentForm	Complete payment form
CardElement	Secure card input
PayPalButton	PayPal checkout button
ApplePayButton	Apple Pay button
GooglePayButton	Google Pay button
PricingTable	Subscription pricing display

Hooks

Hook	Description
usePayment	Payment processing
useSubscription	Subscription management
usePaymentMethod	Saved payment methods
useInvoices	Invoice history

Server-Side

```
import { createPaymentIntent, validateWebhook } from '@philjs/payments/server';

// Create payment intent
const intent = await createPaymentIntent({
  amount: 9900,
  currency: 'usd',
});

// Validate webhook
const event = validateWebhook(payload, signature);
```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: ./providers/stripe, ./providers/paypal, ./providers/square, ./webhooks
- Source files: packages/philjs-payments/src/index.ts, packages/philjs-payments/src/providers/stripe.ts, packages/philjs-payments/src/providers/paypal.ts, packages/philjs-payments/src/providers/square.ts, packages/philjs-payments/src/webhooks/index.ts

Public API

- Direct exports: Address, AttachPaymentMethodRequest, CancelSubscriptionRequest, CheckoutSession, CreateCheckoutRequest, CreateCustomerRequest, CreateInvoiceRequest, CreateSubscriptionRequest, Currency, Customer, IdempotencyError, IdempotencyOptions, Invoice, InvoiceLineItem, LineItem, Money, PayPalAmount, PayPalCapture, PayPalConfig, PayPalItem, PayPalLink, PayPalOrderResponse, PayPalPayer, PayPalProvider, PayPalPurchaseUnit, PayPalRefundResponse, PaymentError, PaymentMethod, PaymentProvider, PaymentStatus, Refund, RefundReason, RefundRequest, SquareConfig, SquareProvider, StripeConfig, StripeProvider, Subscription, SubscriptionError, SubscriptionStatus, WebhookEvent, WebhookHandler, WebhookRequest, WebhookVerificationError, addMoney, attachPaymentMethod, cancelSubscription, createCheckout, createCustomer, createDeterministicKey, createInvoice, createPayPalProvider, createSquareProvider, createStripeProvider, createSubscription, formatMoney, fromCents, generateIdempotencyKey, getProvider, handlePayPalWebhook, handleStripeWebhook, handleWebhook, listProviders, maskCardNumber, refund, registerProvider, sanitizeForLogging, setDefaultProvider, toCents, validateNoPCI, verifyWebhookSignature
- Re-exported names: InvoiceList, PaddleProvider, PayPalProvider, PaymentForm, PricingTable, SquareProvider, StripeProvider, SubscriptionManager, createPaddleProvider, createPayPalProvider, createSquareProvider, createStripeProvider, handlePayPalWebhook, handleStripeWebhook, useInvoices, usePayment, useSubscription, verifyWebhookSignature
- Re-exported modules: ./components/index.js, ./hooks.js, ./providers/paddle.js, ./providers/paypal.js, ./providers/square.js, ./providers/stripe.js, ./webhooks/index.js

License

MIT

@philjs/pdf - Type: Node package - Purpose: PDF generation and manipulation for PhilJS - HTML to PDF, templates, utilities - Version: 0.1.0 - Location: packages/philjs-pdf - Entry points: packages/philjs-pdf/src/index.ts, packages/philjs-pdf/src/templates/index.ts, packages/philjs-pdf/src/utils/index.ts - Keywords: philjs, pdf, pdf-generation, puppeteer, pdfkit

philjs-pdf

PDF generation and manipulation for PhilJS applications. Supports both server-side (Puppeteer) and client-side (@react-pdf/renderer) generation.

Features

- **HTML to PDF** - Convert HTML strings or URLs to PDF using Puppeteer
- **React Component to PDF** - Generate PDFs from React components using @react-pdf/renderer
- **Template-based Generation** - Built-in templates for invoices, reports, and certificates
- **PDF Manipulation** - Merge, split, watermark, protect, and compress PDFs using pdf-lib
- **Font Management** - Embed standard and custom fonts
- **Image Embedding** - Support for PNG and JPEG images
- **Table Generation** - Helper utilities for creating tables in PDFs

Installation

```
npm install philjs-pdf
# or
yarn add philjs-pdf
# or
pnpm add philjs-pdf
```

Quick Start

HTML to PDF (Server-side)

```
import { PDFGenerator } from 'philjs-pdf';

const pdf = new PDFGenerator();

// From HTML string
const buffer = await pdf.generateFromHtml('<h1>Hello World</h1>', {
  format: 'A4',
  margin: { top: '20mm', bottom: '20mm', left: '20mm', right: '20mm' },
});

// From URL
const buffer = await pdf.generateFromHtml('https://example.com', {
  printBackground: true,
  waitForNavigation: true,
});

// Don't forget to close the browser when done
await pdf.close();
```

React Component to PDF (Client-side)

```
import { PDFGenerator } from 'philjs-pdf';
import { Document, Page, Text, View, StyleSheet } from '@react-pdf/renderer';

const styles = StyleSheet.create({
  page: { padding: 30 },
  title: { fontSize: 24, marginBottom: 20 },
});

const MyDocument = ({ title }: { title: string }) => (
  <Document>
    <Page size="A4" style={styles.page}>
      <Text style={styles.title}>{title}</Text>
    </Page>
  </Document>
);

const pdf = new PDFGenerator();
const buffer = await pdf.generateFromComponent(MyDocument, { title: 'My Report' });
```

Template-based Generation

```

import { PDFGenerator } from 'philjs-pdf';

const pdf = new PDFGenerator();

// Generate invoice
const invoice = await pdf.generateFromTemplate({
  template: 'invoice',
  data: {
    invoiceNumber: 'INV-001',
    invoiceDate: '2024-01-15',
    companyName: 'Acme Corp',
    customerName: 'John Doe',
    items: [
      { description: 'Widget A', quantity: 5, unitPrice: 10 },
      { description: 'Widget B', quantity: 3, unitPrice: 25 },
    ],
    total: 125,
  },
});

// Generate certificate
const certificate = await pdf.generateFromTemplate({
  template: 'certificate',
  data: {
    title: 'Certificate of Completion',
    recipientName: 'Jane Smith',
    achievement: 'has successfully completed the Advanced TypeScript Course',
    organizationName: 'Tech Academy',
    issueDate: 'January 15, 2024',
  },
});

await pdf.close();

```

Custom HTML Template

```

import { PDFGenerator } from 'philjs-pdf';

const pdf = new PDFGenerator();

const customTemplate = `

<!DOCTYPE html>
<html>
<head>
<style>
  body { font-family: Arial, sans-serif; padding: 40px; }
  h1 { color: #3b82f6; }
</style>
</head>
<body>
  <h1>{{title}}</h1>
  <p>{{content}}</p>
  <p>Date: {{date}}</p>
</body>
</html>
`;

const buffer = await pdf.generateFromTemplate({
  template: customTemplate,
  data: {
    title: 'Custom Report',
    content: 'This is a custom report generated from a template.',
    date: new Date().toLocaleDateString(),
  },
});

await pdf.close();

```

PDF Manipulation

Merge Multiple PDFs

```

import { PDFGenerator } from 'philjs-pdf';

const pdf = new PDFGenerator();

const merged = await pdf.merge([pdf1Buffer, pdf2Buffer, pdf3Buffer], {
  addPageNumbers: true,
  pageNumberFormat: 'Page {{page}} of {{total}}',
  pageNumberPosition: 'bottom',
});

```

Add Watermark

```
import { PDFGenerator } from 'philjs-pdf';

const pdf = new PDFGenerator();

const watermarked = await pdf.addWatermark(pdfBuffer, {
  text: 'CONFIDENTIAL',
  font_size: 60,
  rotation: -45,
  color: [0.8, 0.2, 0.2],
  opacity: 0.2,
  position: 'center',
});

});
```

Password Protection

```
import { PDFGenerator } from 'philjs-pdf';

const pdf = new PDFGenerator();

const protected = await pdf.protect(pdfBuffer, {
  ownerPassword: 'admin123',
  userPassword: 'user123',
  printing: true,
  copying: false,
});
```

Compress PDF

```
import { PDFGenerator } from 'philjs-pdf';

const pdf = new PDFGenerator();

const compressed = await pdf.compress(pdfBuffer, {
  level: 9,
  removeMetadata: true,
});
```

Extract Pages

```
import { PDFGenerator } from 'philjs-pdf';

const pdf = new PDFGenerator();

// Extract pages 1, 3, and 5
const extracted = await pdf.extractPages(pdfBuffer, [1, 3, 5]);

// Split into individual pages
const pages = await pdf.split(pdfBuffer);
```

Metadata Management

```
import { PDFGenerator } from 'philjs-pdf';

const pdf = new PDFGenerator();

// Set metadata
const withMetadata = await pdf.setMetadata(pdfBuffer, {
  title: 'Annual Report 2024',
  author: 'John Doe',
  subject: 'Financial Summary',
  keywords: ['annual', 'report', 'finance'],
});

// Get metadata
const metadata = await pdf.getMetadata(pdfBuffer);
console.log(metadata.title, metadata.author);
```

Utility Classes

Font Management

```

import { FontManager, getStandardFonts, wrapText } from 'philjs-pdf/utils';
import { PDFDocument } from 'pdf-lib';

const pdfDoc = await PDFDocument.create();
const fontManager = new FontManager();
await fontManager.init(pdfDoc);

// Embed standard font
const helvetica = await fontManager.embedStandardFont('Helvetica');

// Embed custom font
const customFont = await fontManager.embedCustomFont({
  name: 'MyFont',
  path: '/fonts/myfont.ttf',
});

// Word wrap text
const lines = wrapText(helvetica, 'Long text that needs wrapping...', 12, 200);

```

Image Embedding

```

import { ImageManager, scaleToFit } from 'philjs-pdf/utils';
import { PDFDocument } from 'pdf-lib';

const pdfDoc = await PDFDocument.create();
const imageManager = new ImageManager();
await imageManager.init(pdfDoc);

// Embed from URL
const logo = await imageManager.embedFromUrl('https://example.com/logo.png');

// Embed from file
const photo = await imageManager.embedFromFile('/images/photo.jpg');

// Draw on page
const page = pdfDoc.addPage();
imageManager.drawImage(page, logo, {
  x: 50,
  y: 700,
  width: 100,
  maintainAspectRatio: true,
});

```

Table Generation

```

import { TableRenderer, createTable, currencyFormatter } from 'philjs-pdf/utils';
import { PDFDocument } from 'pdf-lib';

const pdfDoc = await PDFDocument.create();
const tableRenderer = new TableRenderer();
await tableRenderer.init(pdfDoc);

const page = pdfDoc.addPage();

const result = tableRenderer.drawTable(page, {
  x: 50,
  y: 700,
  width: 500,
  columns: [
    { header: 'Product', key: 'product', width: '*' },
    { header: 'Quantity', key: 'qty', width: 80, align: 'center' },
    { header: 'Price', key: 'price', width: 100, align: 'right', formatter: currencyFormatter('$') },
  ],
  data: [
    { product: 'Widget A', qty: 5, price: 10 },
    { product: 'Widget B', qty: 3, price: 25 },
  ],
  style: {
    headerBackground: { r: 0.1, g: 0.2, b: 0.4 },
    headerColor: { r: 1, g: 1, b: 1 },
    alternateRowBackground: { r: 0.95, g: 0.95, b: 0.95 },
  },
});
console.log(`Table rendered, ends at Y: ${result.endY}`);

```

Built-in Templates

Invoice Template

```

interface InvoiceData {
  invoiceNumber: string;
  invoiceDate: string;
  dueDate?: string;
  companyName: string;
  companyAddress?: string;
  companyPhone?: string;
  companyEmail?: string;
  companyLogo?: string;
  customerName: string;
  customerAddress?: string;
  customerEmail?: string;
  items: Array<{
    description: string;
    quantity: number;
    unitPrice: number;
  }>;
  subtotal?: number;
  taxRate?: number;
  taxAmount?: number;
  discount?: number;
  total: number;
  notes?: string;
  terms?: string;
  paymentInstructions?: string;
  currency?: string;
}

```

Report Template

```

interface ReportData {
  title: string;
  subtitle?: string;
  reportDate: string;
  reportNumber?: string;
  author?: string;
  authorTitle?: string;
  department?: string;
  companyName?: string;
  companyLogo?: string;
  executiveSummary?: string;
  sections: Array<{
    title: string;
    content: string;
    type?: 'text' | 'chart' | 'table' | 'image';
    data?: any;
  }>;
  showTableOfContents?: boolean;
  confidential?: boolean;
  footerText?: string;
}

```

Certificate Template

```

interface CertificateData {
  title: string;
  subtitle?: string;
  certificateNumber?: string;
  issueDate: string;
  expiryDate?: string;
  recipientName: string;
  recipientTitle?: string;
  achievement: string;
  description?: string;
  course?: string;
  hours?: number;
  grade?: string;
  score?: number;
  organizationName: string;
  organizationLogo?: string;
  signatures?: Array<{
    name: string;
    title: string;
    signature?: string;
  }>;
  theme?: 'classic' | 'modern' | 'elegant' | 'corporate';
  borderStyle?: 'ornate' | 'simple' | 'none';
  primaryColor?: string;
  secondaryColor?: string;
}

```

Configuration Options

PDFGenerator Options

```

interface PDFGeneratorOptions {
  headless?: boolean;           // Use headless browser (default: true)
  format?: 'A4' | 'A3' | 'Letter' | 'Legal' | 'Tabloid';
  margin?: {
    top?: string | number;
    right?: string | number;
    bottom?: string | number;
    left?: string | number;
  };
  debug?: boolean;             // Enable debug mode
}

```

HTML to PDF Options

```

interface HTMLToPDFOptions {
  format?: 'A4' | 'A3' | 'Letter' | 'Legal' | 'Tabloid';
  margin?: PDFMargin;
  printBackground?: boolean;   // Include background colors/images
  landscape?: boolean;        // Landscape orientation
  css?: string;               // Custom CSS to inject
 waitForSelector?: string;    // Wait for element before generating
 waitForNavigation?: boolean;
  headerTemplate?: string;    // HTML header template
  footerTemplate?: string;    // HTML footer template
  displayHeaderFooter?: boolean;
  scale?: number;              // Page scale (0.1 - 2)
  width?: string | number;    // Custom page width
  height?: string | number;   // Custom page height
  preferCSSPageSize?: boolean;
}

```

Server vs Client Usage

Server-side (Node.js)

Use Puppeteer for HTML to PDF conversion:

```

import { PDFGenerator } from 'philjs-pdf';

// Server-side rendering with Puppeteer
const pdf = new PDFGenerator({ headless: true });
const buffer = await pdf.generateFromHtml(html);
await pdf.close();

```

Client-side (Browser)

Use @react-pdf/renderer for React component to PDF:

```

import { PDFGenerator } from 'philjs-pdf/client';

// Client-side rendering with @react-pdf/renderer
const pdf = new PDFGenerator();
const blob = await pdf.generateBlobFromComponent(MyDocument);

// Create download Link
const url = URL.createObjectURL(blob);
const link = document.createElement('a');
link.href = url;
link.download = 'document.pdf';
link.click();

```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./templates`, `./utils`
- Source files: `packages/philjs-pdf/src/index.ts`, `packages/philjs-pdf/src/templates/index.ts`, `packages/philjs-pdf/src/utils/index.ts`

Public API

- Direct exports: `ComponentToPDFOptions`, `CompressionOptions`, `HTMLToPDFOptions`, `MergeOptions`, `PDFGenerator`, `PDFGeneratorOptions`, `PDFMargin`, `ProtectionOptions`, `TemplateData`, `TemplateName`, `TemplateOptions`, `WatermarkOptions`, `createPDFGenerator`, `htmlToPdf`, `mergePdfs`, `templates`
- Re-exported names: (none detected)
- Re-exported modules: `./certificate.js`, `./fonts.js`, `./images.js`, `./invoice.js`, `./report.js`, `./tables.js`, `./templates/index.js`, `./utils/fonts.js`, `./utils/images.js`, `./utils/tables.js`

License

MIT

@philjs/perf - Type: Node package - Purpose: High-performance runtime utilities for PhilJS - memoization, batching, pooling, lazy evaluation - Version: 0.1.0 - Location: `packages/philjs-perf` - Entry points: `packages/philjs-perf/src/index.ts`, `packages/philjs-perf/src/memo.ts`, `packages/philjs-perf/src/batch.ts`, `packages/philjs-perf/src/pool.ts`, `packages/philjs-perf/src/lazy.ts` - Keywords: philjs, performance, memoization, batching, object-pooling, lazy-evaluation, optimization

@philjs/perf

High-performance runtime utilities for PhilJS - memoization, batching, pooling, lazy evaluation

Overview

High-performance runtime utilities for PhilJS - memoization, batching, pooling, lazy evaluation

Focus Areas

- philjs, performance, memoization, batching, object-pooling, lazy-evaluation, optimization

Entry Points

- packages/philjs-perf/src/index.ts
- packages/philjs-perf/src/memo.ts
- packages/philjs-perf/src/batch.ts
- packages/philjs-perf/src/pool.ts
- packages/philjs-perf/src/lazy.ts

Quick Start

```
import { LazyValue, ObjectPool, Scheduler } from '@philjs/perf';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- LazyValue
- ObjectPool
- Scheduler
- arrayPool
- batch
- batchAsync
- clearMemoCache
- createBatcher
- createPool
- lazy
- lazyAsync
- lazyInit

Install

```
pnpm add @philjs/perf
```

Usage

```
import { LazyValue, ObjectPool, Scheduler } from '@philjs/perf';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: ., ./memo, ./batch, ./pool, ./lazy
- Source files: packages/philjs-perf/src/index.ts, packages/philjs-perf/src/memo.ts, packages/philjs-perf/src/batch.ts, packages/philjs-perf/src/pool.ts, packages/philjs-perf/src/lazy.ts

Public API

- Direct exports: LazyValue, ObjectPool, Scheduler, arrayPool, batch, batchAsync, clearMemoCache, createBatcher, createPool, lazy, lazyAsync, lazyInit, lazyProp, mapPool, memo, memoAsync, memoWeak, objectPool, setPool
- Re-exported names: BatchOptions, LRUCache, LazyOptions, LazyValue, MemoOptions, ObjectPool, PoolOptions, Scheduler, batch, batchAsync, clearMemoCache, createBatcher, createLRU, createPool, debounce, lazy, lazyAsync, memo, memoAsync, memoWeak, raffThrottle, throttle
- Re-exported modules: ./batch.js, ./cache.js, ./lazy.js, ./memo.js, ./pool.js, ./timing.js, ./types.js

License

MIT

@philjs/perf-budget - Type: Node package - Purpose: Performance budget enforcement for PhilJS - Core Web Vitals, bundle size limits, build checks - Version: 0.1.0 - Location: packages/philjs-perf-budget - Entry points: packages/philjs-perf-budget/src/index.ts - Keywords: philjs, performance, budget, web-vitals, lcp, cls, fid, bundle-size

@philjs/perf-budget

Performance budget enforcement for PhilJS - Core Web Vitals, bundle size limits, build checks

Overview

Performance budget enforcement for PhilJS - Core Web Vitals, bundle size limits, build checks

Focus Areas

- philjs, performance, budget, web-vitals, lcp, cls, fid, bundle-size

Entry Points

- packages/philjs-perf-budget/src/index.ts

Quick Start

```
import { BudgetChecker, BudgetConfig, BudgetViolation } from '@philjs/perf-budget';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- BudgetChecker
- BudgetConfig
- BudgetViolation
- BuildArtifact
- BuildBudgetChecker
- PerformanceBudget
- PerformanceMetrics
- PerformanceObserverManager
- PerformanceScore
- perfBudgetPlugin
- usePerformanceBudget
- usePerformanceMetric

Install

```
pnpm add @philjs/perf-budget
```

Usage

```
import { BudgetChecker, BudgetConfig, BudgetViolation } from '@philjs/perf-budget';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-perf-budget/src/index.ts

Public API

- Direct exports: BudgetChecker, BudgetConfig, BudgetViolation, BuildArtifact, BuildBudgetChecker, PerformanceBudget, PerformanceMetrics, PerformanceObserverManager, PerformanceScore, perfBudgetPlugin, usePerformanceBudget, usePerformanceMetric, useWebVitals
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/playground - Type: Node package - Purpose: Interactive browser playground for PhilJS - try code without installing - Version: 0.1.0 - Location: packages/philjs-playground - Entry points: packages/philjs-playground/src/index.ts - Keywords: philjs, playground, repl, interactive, browser

philjs-playground

Interactive browser playground for PhilJS - Try code without installing.

Installation

```
pnpm add philjs-playground
```

Usage

Standalone Playground

Create a full-featured playground in your app:

```

import { createPlayground } from 'philjs-playground';

const playground = createPlayground({
  container: '#playground',
  initialCode: `
    import { signal } from 'philjs-core';

    const count = signal(0);

    function App() {
      return (
        <button onClick={() => count.set(c => c + 1)}>
          Count: {count()}
        </button>
      );
    }

    export default App;
  `,
  theme: 'dark',
  autoRun: true,
  showConsole: true
});

```

Individual Components

Use playground components separately:

```

import { createEditor, createPreview, createConsole } from 'philjs-playground';

// Code editor with PhilJS syntax highlighting
const editor = createEditor({
  container: '#editor',
  language: 'typescript',
  theme: 'one-dark',
  onChange: (code) => {
    console.log('Code changed:', code);
  }
});

// Live preview panel
const preview = createPreview({
  container: '#preview',
  sandboxed: true
});

// Console output
const console = createConsole({
  container: '#console',
  maxMessages: 100
});

// Compile and run code
import { compileCode } from 'philjs-playground';

const result = await compileCode(editor.getValue());
if (result.success) {
  preview.run(result.code);
} else {
  console.error('Compilation failed:', result.errors);
}

```

Features

- **CodeMirror Integration** - Powerful code editor with syntax highlighting
- **Live Preview** - Instant feedback with sandboxed execution
- **Console Output** - Capture and display console logs
- **Error Handling** - Beautiful error messages with stack traces
- **Examples Library** - Built-in examples and tutorials
- **TypeScript Support** - Full TypeScript compilation in the browser
- **Dark/Light Themes** - Customizable appearance
- **Auto-run Mode** - Execute code on change

Configuration

Playground Options

```

interface PlaygroundConfig {
  // Target container element
  container: string | HTMLElement;

  // Initial code to display
  initialCode?: string;

  // Editor theme ('Light' | 'dark' | 'one-dark')
  theme?: string;

  // Auto-run code on change
  autoRun?: boolean;

  // Debounce delay for auto-run (ms)
  debounce?: number;

  // Show console panel
  showConsole?: boolean;

  // Show examples sidebar
  showExamples?: boolean;

  // Enable TypeScript
  typescript?: boolean;

  // Custom imports to make available
  imports?: Record<string, any>;
}

```

Editor Options

```

interface EditorConfig {
  container: string | HTMLElement;
  language?: 'javascript' | 'typescript';
  theme?: string;
  lineNumbers?: boolean;
  lineWrapping?: boolean;
  tabSize?: number;
  onChange?: (code: string) => void;
}

```

Preview Options

```

interface PreviewConfig {
  container: string | HTMLElement;
  sandboxed?: boolean;
  onError?: (error: Error) => void;
  onMount?: () => void;
}

```

API

Playground

- `createPlayground(config)` - Create a full playground instance
- `playground.run()` - Execute current code
- `playground.setCode(code)` - Update editor code
- `playground.getCode()` - Get current code
- `playground.reset()` - Reset to initial code
- `playground.destroy()` - Clean up playground

Editor

- `createEditor(config)` - Create editor instance
- `editor.getValue()` - Get current code
- `editor.setValue(code)` - Set editor code
- `editor.focus()` - Focus the editor
- `editor.destroy()` - Clean up editor

Preview

- `createPreview(config)` - Create preview instance
- `preview.run(code)` - Execute compiled code
- `preview.clear()` - Clear preview
- `preview.destroy()` - Clean up preview

Console

- `createConsole(config)` - Create console instance
- `console.log(...args)` - Log message
- `console.error(...args)` - Log error
- `console.warn(...args)` - Log warning
- `console.clear()` - Clear console
- `console.destroy()` - Clean up console

Compiler

- `compileCode(code)` - Compile TypeScript/JSX to JavaScript
- `transpileCode(code)` - Transpile without type checking

Examples

Simple Counter

```
const playground = createPlayground({
  container: '#playground',
  initialCode: `

    import { signal } from 'philjs-core';

    const count = signal(0);

    export default function Counter() {
      return (
        <div>
          <button onClick={() => count.set(c => c - 1)}>-</button>
          <span> {count()} </span>
          <button onClick={() => count.set(c => c + 1)}>+</button>
        </div>
      );
    }
});
```

Todo List

```
const playground = createPlayground({
  container: '#playground',
  initialCode: `

    import { signal } from 'philjs-core';

    const todos = signal([]);
    const input = signal('');

    function addTodo() {
      if (input().trim()) {
        todos.set([...todos(), { text: input(), done: false }]);
        input.set('');
      }
    }

    export default function TodoApp() {
      return (
        <div>
          <input
            value={input()}
            onChange={(e) => input.set(e.target.value)}
            placeholder="Add todo..." />
          <button onClick={addTodo}>Add</button>
        <ul>
          {todos().map((todo, i) => (
            <li>
              <input
                type="checkbox"
                checked={todo.done}
                onChange={() => {
                  const updated = [...todos()];
                  updated[i].done = !updated[i].done;
                  todos.set(updated);
                }}
              />
              {todo.text}
            </li>
          )));
        </ul>
      );
    }
});
```

Custom Theme

```
const playground = createPlayground({
  container: '#playground',
  theme: {
    background: '#1e1e1e',
    foreground: '#d4d4d4',
    selection: '#264f78',
    lineNumber: '#858585',
    keyword: '#569cd6',
    string: '#ce9178',
    comment: '#6a9955'
  }
});
```

Built-in Examples

Access pre-built examples:

```
import { exampleCode, tutorialSteps } from 'philjs-playground';

// Load an example
playground.setCode(exampleCode.counter);
playground.setCode(exampleCode.todoList);
playground.setCode(exampleCode.fetchData);

// Start tutorial
tutorialSteps.forEach((step, i) => {
  console.log(`Step ${i + 1}: ${step.title}`);
  playground.setCode(step.code);
});
```

Documentation

For more information, see the PhilJS documentation.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-playground/src/index.ts

Public API

- Direct exports: (none detected)
- Re-exported names: CompileResult, Console, ConsoleMessage, Editor, EditorConfig, Playground, PlaygroundConfig, Preview, PreviewConfig, compileCode, createConsole, createEditor, createPlayground, createPreview, exampleCode, transpileCode, tutorialSteps
- Re-exported modules: ./compiler.js, ./console.js, ./editor.js, ./examples.js, ./playground.js, ./preview.js, ./types.js

License

MIT

@philjs/plugin-analytics - Type: Node package - Purpose: Analytics integration plugin for PhilJS (GA4, Plausible, Mixpanel, and more) - Version: 0.1.0 - Location: packages/philjs-plugin-analytics - Entry points: packages/philjs-plugin-analytics/src/index.ts, packages/philjs-plugin-analytics/src/client.ts - Keywords: philjs, plugin, analytics, tracking, ga4, google-analytics, plausible, mixpanel, amplitude, segment, posthog, umami, fathom, privacy, gdpr

philjs-plugin-analytics

Universal analytics integration for PhilJS - support for Google Analytics, Plausible, Mixpanel, and more.

Features

- **Multiple Providers** - Support for 8+ analytics platforms
- **Privacy First** - Built-in privacy controls (DNT, IP anonymization)
- **Auto Tracking** - Page views, errors, and custom events
- **Type Safe** - Full TypeScript support
- **Zero Config** - Works out of the box

Supported Providers

- Google Analytics 4 (GA4)
- Plausible
- Mixpanel
- Amplitude
- Segment
- PostHog
- Umami
- Fathom

Installation

```
# Using PhilJS CLI
philjs plugin add philjs-plugin-analytics

# Or with npm
npm install philjs-plugin-analytics
```

Usage

Google Analytics 4

```
import analytics from 'philjs-plugin-analytics';

export default defineConfig({
  plugins: [
    analytics({
      provider: 'ga4',
      trackingId: 'G-XXXXXXXXXX',
    }),
  ],
});
```

Plausible

```
analytics({
  provider: 'plausible',
  trackingId: 'yourdomain.com',
})
```

Mixpanel

```
analytics({
  provider: 'mixpanel',
  trackingId: 'YOUR_PROJECT_TOKEN',
})
```

Configuration

Option	Type	Description	Required
provider	AnalyticsProvider	Analytics provider	Yes
trackingId	string	Tracking ID or API key	Yes
debug	boolean	Enable debug mode	No
disableInDev	boolean	Disable in development	No (default: true)
privacy	PrivacyOptions	Privacy settings	No
customEvents	CustomEventOptions	Event tracking options	No

Privacy Options

```
privacy: {
  anonymizeIp: true, // Anonymize IP addresses
  respectDnt: true, // Respect Do Not Track
  cookieConsent: false // Require cookie consent
}
```

Custom Event Tracking

```
customEvents: {
  pageViews: true, // Track page views automatically
  clicks: false, // Track clicks
  forms: false, // Track form submissions
  errors: true // Track errors
}
```

API Usage

The plugin generates a tracking API at `src/lib/analytics.ts`:

```

import { trackEvent, identifyUser, setUserProperties } from './lib/analytics';

// Track custom events
trackEvent('button_click', {
  button_name: 'signup',
  location: 'homepage',
});

// Identify users (Mixpanel, Amplitude)
identifyUser('user-123', {
  email: 'user@example.com',
  plan: 'premium',
});

// Set user properties (GA4)
setUserProperties({
  user_type: 'premium',
  signup_date: '2024-01-01',
});

```

Auto-Tracking

Page Views

Automatically tracked for: - Initial page load - SPA navigation (pushState/replaceState) - Hash changes - Back/forward navigation

Error Tracking

Automatically tracks: - JavaScript errors - Unhandled promise rejections - Network errors

Privacy & GDPR

Respect Do Not Track

```

import { analyticsUtils } from 'philjs-plugin-analytics';

if (analyticsUtils.hasDNT()) {
  // User has DNT enabled
  // Analytics won't Load
}

```

Cookie Consent

```

analytics({
  privacy: {
    cookieConsent: true,
  },
});

// Later, after user consent
window.grantAnalyticsConsent();

```

Utilities

```

import { analyticsUtils } from 'philjs-plugin-analytics';

// Check DNT status
const hasDNT = analyticsUtils.hasDNT();

// Generate session ID
const sessionId = analyticsUtils.generateSessionId();

// Get user agent info
const userAgent = analyticsUtils.getUserAgent();

// Get page metadata
const metadata = analyticsUtils.getPageMetadata();

```

Provider-Specific Features

Google Analytics 4

- Enhanced measurement
- Custom dimensions
- User properties
- E-commerce tracking
- Automatic page view tracking
- Event parameters
- User ID tracking

Plausible

- Privacy-focused analytics
- No cookies by default
- GDPR compliant
- Lightweight script (< 1KB)
- Hash-based routing support

- Custom events
- Goals tracking

Mixpanel

- User identification
- User profiles
- People properties
- Cohort analysis
- A/B testing
- Funnel analysis
- Revenue tracking

Examples

E-commerce Tracking (GA4)

```
import { trackEvent } from './lib/analytics';

// Track purchase
trackEvent('purchase', {
  transaction_id: 'T12345',
  value: 99.99,
  currency: 'USD',
  items: [
    {
      item_id: 'SKU123',
      item_name: 'Product Name',
      price: 99.99,
    },
  ],
});
```

User Funnel (Mixpanel)

```
import { trackEvent } from './lib/analytics';

// Track funnel steps
trackEvent('viewed_product');
trackEvent('added_to_cart');
trackEvent('started_checkout');
trackEvent('completed_purchase');
```

TypeScript

Full type safety:

```
import type { AnalyticsPluginConfig, AnalyticsProvider } from 'philjs-plugin-analytics';

const config: AnalyticsPluginConfig = {
  provider: 'ga4',
  trackingId: 'G-XXXXXXXXXX',
  privacy: {
    anonymizeIp: true,
  },
};
```

Advanced Features

Session Management

The plugin automatically manages user sessions:

```
import { analytics } from 'philjs-plugin-analytics/client';

// Get current session context
const context = analytics.getContext();
console.log(context.sessionId); // Unique session ID
console.log(context.pageLoadTime); // Page Load timestamp
```

Performance Tracking

Track Web Vitals and custom performance metrics:

```
customEvents: {
  performance: true, // Auto-track performance metrics
}
```

Cookie Management

Control cookie behavior:

```
privacy: {
  cookieDomain: '.example.com',
  cookieExpires: 365, // Days
}
```

Development vs Production

```
// Automatically disabled in development
disableInDev: true,

// Enable debug Logging
debug: true,
```

API Reference

Plugin Configuration

```
interface AnalyticsPluginConfig {
  provider: AnalyticsProvider;
  trackingId: string;
  options?: ProviderOptions;
  debug?: boolean;
  disableInDev?: boolean;
  privacy?: PrivacyOptions;
  customEvents?: CustomEventOptions;
}
```

Client Methods

```
// Track custom event
trackEvent(name: string, properties?: Record<string, any>): void

// Track page view
trackPageView(url?: string, title?: string): void

// Identify user
identifyUser(userId: string, traits?: Record<string, any>): void

// Set user properties
setUserProperties(properties: Record<string, any>): void

// Track transaction
trackTransaction(transaction: EcommerceTransaction): void
```

Troubleshooting

Analytics not loading

1. Check that your tracking ID is correct
2. Verify DNT is not enabled
3. Check browser console for errors
4. Enable debug mode: debug: true

Events not tracked

1. Ensure analytics is initialized
2. Check that provider script loaded
3. Verify event properties are valid
4. Check network tab for analytics requests

Privacy issues

1. Enable IP anonymization: anonymizeIp: true
2. Respect DNT: respectDnt: true
3. Use privacy-focused provider like Plausible

Contributing

Contributions are welcome! Please see the main PhilJS repository for guidelines.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .. ./client
- Source files: packages/philjs-plugin-analytics/src/index.ts, packages/philjs-plugin-analytics/src/client.ts

Public API

- Direct exports: AnalyticsEvent, AnalyticsPluginConfig, AnalyticsProvider, EcommerceTransaction, analytics, analyticsUtils, createAnalyticsPlugin, identifyUser, setUserProperties, trackEvent, trackPageView, trackTransaction
- Re-exported names: AnalyticsContext, CustomEventOptions, EcommerceItem, EcommerceTransaction, IAnalyticsProvider, PrivacyOptions, ProviderOptions, UserIdentification, identifyUser, setUserProperties, trackEvent, trackPageView, trackTransaction
- Re-exported modules: ./types.js, philjs-plugin-analytics/client

License

MIT

@philjs/plugin-i18n - Type: Node package - Purpose: Internationalization plugin for PhilJS with Vite integration - Version: 0.1.0 - Location: packages/philjs-plugin-i18n - Entry points: packages/philjs-plugin-i18n/src/index.ts, packages/philjs-plugin-i18n/src/client.ts - Keywords: philjs, plugin, i18n, internationalization, localization, l10n, translations, locale, language

philjs-plugin-i18n

Internationalization plugin for PhilJS with Vite integration, automatic locale detection, and type-safe translations.

Installation

```
pnpm add philjs-plugin-i18n
```

Quick Start

1. Add the Plugin

```
// vite.config.ts
import { defineConfig } from 'vite';
import { createI18nPlugin } from 'philjs-plugin-i18n';

export default defineConfig({
  plugins: [
    createI18nPlugin({
      defaultLocale: 'en',
      locales: ['en', 'es', 'fr', 'de'],
      translationsDir: './src/locales',
      detectBrowserLocale: true,
      persistLocale: true,
    }).vitePlugin(),
  ],
});
```

2. Create Translation Files

```
src/
  locales/
    en.json
    es.json
    fr.json
```

```
// src/locales/en.json
{
  "common": {
    "greeting": "Hello, {{name}}!",
    "loading": "Loading...",
    "error": "An error occurred"
  },
  "buttons": {
    "save": "Save",
    "cancel": "Cancel",
    "submit": "Submit"
  }
}
```

3. Use Translations

```
import { t, setLocale, formatDate } from './lib/i18n';

function Greeting({ name }) {
  return (
    <div>
      <h1>{t('common.greeting', { name })}</h1>
      <p>{formatDate(new Date())}</p>

      <button onClick={() => setLocale('es')}>
        Switch to Spanish
      </button>
    </div>
  );
}
```

Features

Automatic Locale Detection

The plugin automatically detects the user's browser locale and matches it against your configured locales:

```
createI18nPlugin({
  defaultLocale: 'en',
  locales: ['en', 'en-US', 'es', 'es-MX', 'fr'],
  detectBrowserLocale: true, // Enabled by default
});
```

Locale Persistence

User's locale preference is saved to localStorage:

```
createI18nPlugin({
  persistLocale: true, // Enabled by default
  storageKey: 'my-app-locale', // Custom key
});
```

Interpolation

Use {{variable}} syntax for dynamic values:

```
{
  "welcome": "Welcome, {{name}}!",
  "items": "You have {{count}} items in your cart"
}
```

```
t('welcome', { name: 'John' }); // "Welcome, John!"
t('items', { count: 5 }); // "You have 5 items in your cart"
```

Pluralization

Define plural forms based on count:

```
{
  "items": {
    "one": "{{count}} item",
    "other": "{{count}} items"
  }
}
```

```
t('items', { count: 1 }); // "1 item"
t('items', { count: 5 }); // "5 items"
```

Number Formatting

Format numbers according to locale:

```
import { formatNumber, formatCurrency } from './lib/i18n';

formatNumber(1234.56); // "1,234.56" (en) or "1.234,56" (de)
formatCurrency(99.99, 'USD'); // "$99.99" (en) or "99,99 $" (fr)
```

Date Formatting

Format dates according to locale:

```
import { formatDate, formatRelativeTime } from './lib/i18n';

formatDate(new Date()); // "12/20/2024" (en) or "20/12/2024" (fr)
formatRelativeTime(-1, 'day'); // "yesterday" (en) or "hier" (fr)
```

RTL Support

Configure RTL languages:

```
createI18nPlugin({
  defaultLocale: 'en',
  locales: [
    { code: 'en', name: 'English', dir: 'ltr' },
    { code: 'ar', name: '', dir: 'rtl' },
    { code: 'he', name: '', dir: 'rtl' },
  ],
});
```

```
import { isRTL, currentLocale } from 'philjs-plugin-i18n/client';

if (isRTL()) {
  // Apply RTL styles
}
```

Type-Safe Translations

The plugin generates TypeScript types for your translation keys:

```
// Auto-generated in src/i18n.d.ts
interface TranslationKeys {
  'common.greeting': string;
  'common.loading': string;
  'buttons.save': string;
  // ...
}

// Now t() is type-safe
t('common.greeting'); // OK
t('invalid.key'); // TypeScript error
```

API Reference

Client Functions

Function	Description
t(key, params?)	Translate a key with optional interpolation
setLocale(locale)	Change the current locale
hasTranslation(key)	Check if a translation exists
formatNumber(value, options?)	Format a number
formatDate(date, options?)	Format a date
formatCurrency(value, currency?)	Format currency
formatRelativeTime(value, unit)	Format relative time
getAvailableLocales()	Get all configured locales
isRTL()	Check if current locale is RTL

Configuration Options

Option	Type	Default	Description
defaultLocale	string	Required	Default locale code
locales	string[] \ LocaleConfig[]	Required	Supported locales
fallbackLocale	string	defaultLocale	Fallback when translation missing
translationsDir	string	./src/locales	Directory for translation files
format	'json' \ 'yaml' \ 'js'	'json'	Translation file format
detectBrowserLocale	boolean	true	Auto-detect browser locale
persistLocale	boolean	true	Save to localStorage
storageKey	string	'philjs-locale'	localStorage key
debug	boolean	false	Enable debug logging

Virtual Module

Import translations via the virtual module:

```
import { translations, locales, getTranslation } from 'virtual:philjs-i18n';

// All translations
console.log(translations);

// Available locales
console.log(locales); // ['en', 'es', 'fr']

// Get specific locale
const esTranslations = getTranslation('es');
```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `., ./client`
- Source files: `packages/philjs-plugin-i18n/src/index.ts`, `packages/philjs-plugin-i18n/src/client.ts`

Public API

- Direct exports: `TranslationKey`, `createI18nPlugin`, `currentLocale`, `formatCurrency`, `formatDate`, `formatNumber`, `formatRelativeTime`, `getAvailableLocales`, `getI18nContext`, `getTranslation`, `hasTranslation`, `initI18n`, `isRTL`, `loadTranslations`, `locales`, `setLocale`, `t`, `translations`, `useTranslation`
- Re-exported names: `I18nContextValue`, `I18nPluginConfig`, `LocaleConfig`, `PluralRules`, `TranslationMap`, `TranslationValue`, `ViteI18nPluginOptions`, `currentLocale`, `formatCurrency`, `formatDate`, `formatNumber`, `formatRelativeTime`, `getAvailableLocales`, `getI18nContext`, `hasTranslation`, `initI18n`, `isRTL`, `loadTranslations`, `setLocale`, `t`, `useTranslation`
- Re-exported modules: `./client.js`, `./types.js`

License

MIT

@philjs/plugin-pwa - Type: Node package - Purpose: Progressive Web App (PWA) plugin for PhilJS with service workers and offline support - Version: 0.1.0 - Location: packages/philjs-plugin-pwa - Entry points: packages/philjs-plugin-pwa/src/index.ts - Keywords: philjs, pwa, progressive-web-app, service-worker, offline

@philjs/plugin-pwa

Progressive Web App (PWA) plugin for PhilJS - Turn your web app into a fully-featured PWA with service workers, offline support, install prompts, update notifications, and background sync.

Features

- Service Worker Generation:** Automatically generate optimized service workers with precaching and runtime caching
- Cache Strategies:** Multiple caching strategies (cache-first, network-first, stale-while-revalidate, cache-only, network-only)
- Web App Manifest:** Generate and configure your PWA manifest with icons, shortcuts, and share targets
- Offline Support:** Automatic offline detection and offline page fallbacks

- **Install Prompts:** Handle native PWA install prompts and track installation status
- **Update Notifications:** Check for service worker updates and notify users with update available
- **Background Sync:** Queue requests to sync when connection is restored
- **Asset Management:** Configure icon sizes, screenshots, and app metadata
- **Meta Tags:** Automatic generation of PWA meta tags for cross-browser support
- **Type-Safe Configuration:** Full TypeScript support with complete type definitions
- **Development Tools:** Built-in utilities for testing PWA features locally

Installation

```
npm install @philjs/plugin-pwa
# or
pnpm add @philjs/plugin-pwa
# or
yarn add @philjs/plugin-pwa
```

Quick Start

1. Generate Service Worker

```
import { generateServiceWorker } from '@philjs/plugin-pwa';
import { getDefaultCacheRules } from '@philjs/plugin-pwa';

// Generate service worker code
const swCode = generateServiceWorker({
  cacheVersion: 'v1',
  precache: [
    '/index.html',
    '/app.js',
    '/app.css',
  ],
  runtimeCaching: getDefaultCacheRules(),
  skipWaiting: true,
  clientsClaim: true,
  navigationPreload: true,
  offlineFallback: '/offline.html',
  backgroundSync: {
    enabled: true,
    queueName: 'api-queue',
  },
});

// Write to public/sw.js
fs.writeFileSync('public/sw.js', swCode);
```

2. Generate Web App Manifest

```
import { generateManifest, injectManifestLink } from '@philjs/plugin-pwa';

const manifest = generateManifest({
  name: 'My Awesome App',
  short_name: 'MyApp',
  description: 'The most awesome app ever',
  start_url: '/',
  display: 'standalone',
  theme_color: '#6677ea',
  background_color: '#ffffff',
  icons: [
    {
      src: '/icons/icon-192.png',
      sizes: '192x192',
      type: 'image/png',
      purpose: 'any',
    },
    {
      src: '/icons/icon-512.png',
      sizes: '512x512',
      type: 'image/png',
      purpose: 'any',
    },
    {
      src: '/icons/icon-maskable-192.png',
      sizes: '192x192',
      type: 'image/png',
      purpose: 'maskable',
    },
  ],
});

// Save manifest
fs.writeFileSync('public/manifest.json', JSON.stringify(manifest, null, 2));

// Add to your HTML
const manifestLink = injectManifestLink('/manifest.json');
```

3. Register Service Worker

```

import {
  registerServiceWorker,
  initOfflineDetection,
  initInstallPrompt,
  initUpdateNotifications,
} from '@philjs/plugin-pwa';

// Register service worker
await registerServiceWorker('/sw.js');

// Initialize offline detection
const cleanupOffline = initOfflineDetection();

// Initialize install prompt handling
const cleanupInstall = initInstallPrompt();

// Initialize update checking
const cleanupUpdates = initUpdateNotifications({
  checkInterval: 60 * 60 * 1000, // 1 hour
  autoCheck: true,
});

// Cleanup on unmount
onCleanup(() => {
  cleanupOffline();
  cleanupInstall();
  cleanupUpdates();
});

```

Core Concepts

Service Worker Configuration

Configure how your service worker caches and serves content:

```

import { generateServiceWorker } from '@philjs/plugin-pwa';

const swCode = generateServiceWorker({
  // Cache versioning
  cachePrefix: 'my-app',
  cacheVersion: 'v1',

  // Files to cache on install
  precache: [
    '/index.html',
    '/app.js',
    '/app.css',
    '/favicon.ico',
  ],

  // Runtime caching rules
  runtimeCaching: [
    {
      pattern: /\.js$/,
      strategy: 'cache-first',
      cacheName: 'js-cache',
      maxAge: 7 * 24 * 60 * 60 * 1000, // 7 days
      maxEntries: 50,
    },
    {
      pattern: /^https:\/\/api\.example\.com/,
      strategy: 'network-first',
      cacheName: 'api-cache',
      networkTimeout: 3000,
      maxEntries: 20,
    },
    {
      pattern: /\.(png|jpg|jpeg|gif|svg)\$.,
      strategy: 'stale-while-revalidate',
      cacheName: 'image-cache',
      maxEntries: 100,
    },
  ],
}

// Service worker behavior
skipWaiting: true,           // Activate immediately
clientsClaim: true,          // Take control of all clients
navigationPreload: true,     // Enable navigation preload for faster loads
offlineFallback: '/offline.html',

// Background sync
backgroundSync: {
  enabled: true,
  queueName: 'sync-queue',
},
});

```

Cache Strategies

Choose the right caching strategy for your content:

Cache-First

Best for static assets that rarely change:

```
{  
  pattern: /\.(\js|\css)$/,  
  strategy: 'cache-first',  
  maxAge: 7 * 24 * 60 * 60 * 1000, // 7 days  
  maxEntries: 50,  
}
```

Network-First

Best for API responses and dynamic content:

```
{  
  pattern: /^/api//,  
  strategy: 'network-first',  
  networkTimeout: 3000,  
  maxEntries: 20,  
}
```

Stale-While-Revalidate

Best for images and content that can be slightly stale:

```
{  
  pattern: /\.(png|jpg|gif|svg)$/,  
  strategy: 'stale-while-revalidate',  
  maxEntries: 100,  
}
```

Cache-Only

For content that's always available offline:

```
{  
  pattern: /^/static//,  
  strategy: 'cache-only',  
}
```

Network-Only

For content that must always be fresh:

```
{  
  pattern: /^/live//,  
  strategy: 'network-only',  
}
```

Predefined Cache Rules

Use built-in cache rules for common scenarios:

```
import { cacheRules, getDefaultCacheRules } from '@philjs/plugin-pwa';  
  
// Get all default rules  
const defaultRules = getDefaultCacheRules();  
  
// Or use individual rules  
const rules = [  
  cacheRules.staticAssets(), // .js, .css, images, fonts  
  cacheRules.images(), // Images with stale-while-revalidate  
  cacheRules.fonts(), // Font files (1-year cache)  
  ...cacheRules.googleFonts(), // Google Fonts stylesheets + webfonts  
  cacheRules.apiResponses('/api'),  
];
```

Web App Manifest

Configure your PWA's appearance and behavior:

```

import { generateManifest } from '@philjs/plugin-pwa';

const manifest = generateManifest({
  // Required
  name: 'My Application',
  short_name: 'MyApp',
  start_url: '/',

  // Display
  display: 'standalone',      // fullscreen, standalone, minimal-ui, browser
  orientation: 'portrait-primary',
  theme_color: '#667eea',
  background_color: '#ffffff',
  scope: '/',

  // Icons and branding
  icons: [
    {
      src: '/icons/icon-192.png',
      sizes: '192x192',
      type: 'image/png',
      purpose: 'any',
    },
    {
      src: '/icons/icon-512.png',
      sizes: '512x512',
      type: 'image/png',
      purpose: 'any',
    },
  ],
}

// Screenshots for app stores
screenshots: [
  {
    src: '/screenshots/1.png',
    sizes: '540x720',
    type: 'image/png',
    form_factor: 'narrow',
  },
  {
    src: '/screenshots/2.png',
    sizes: '1280x720',
    type: 'image/png',
    form_factor: 'wide',
  },
],
}

// App store metadata
categories: ['productivity', 'utilities'],

// Share target
share_target: {
  action: '/share',
  method: 'POST',
  enctype: 'multipart/form-data',
  params: {
    title: 'title',
    text: 'text',
    url: 'url',
  },
  files: [
    {
      name: 'image',
      accept: ['image/png', 'image/jpeg'],
    },
  ],
},
}

// App shortcuts
shortcuts: [
  {
    name: 'New Note',
    url: '/new',
    icons: [
      {
        src: '/icons/new-192.png',
        sizes: '192x192',
        type: 'image/png',
      },
    ],
  },
];
});

```

Offline Support

Detect online/offline status and handle offline scenarios:

```

import {
  isOnline,
  initOfflineDetection,
  queueOfflineRequest,
  prefetchOfflinePage,
} from '@philjs/plugin-pwa';

// Access current online status (signal)
console.log(isOnline()); // true or false

// React to online/offline changes
createEffect(() => {
  if (isOnline()) {
    console.log('Back online!');
  } else {
    console.log('App is offline');
  }
});

// Initialize offline detection
const cleanup = initOfflineDetection();

// Queue a request to be synced when online
await queueOfflineRequest('/api/data', {
  method: 'POST',
  body: JSON.stringify({ /* data */ }),
});

// Prefetch offline page
await prefetchOfflinePage('/offline.html');

// Don't forget to cleanup
cleanup();

```

Install Prompts

Handle native PWA installation:

```

import {
  canInstall,
  isInstalled,
  initInstallPrompt,
  showInstallPrompt,
} from '@philjs/plugin-pwa';

// Initialize install prompt handling
const cleanup = initInstallPrompt();

// Check if app is installable
createEffect(() => {
  if (canInstall()) {
    console.log('App can be installed');
    // Show your custom install button
  }
});

// Check if already installed
createEffect(() => {
  if (isInstalled()) {
    console.log('App is installed');
  }
});

// Show install prompt
async function handleInstallClick() {
  const outcome = await showInstallPrompt();
  if (outcome === 'accepted') {
    console.log('User accepted installation');
  } else if (outcome === 'dismissed') {
    console.log('User dismissed installation');
  }
}

// Listen for app installed event
window.addEventListener('pwa-installed', () => {
  console.log('App was installed');
  // Hide install button, show message, etc.
});

cleanup();

```

Update Notifications

Detect and apply service worker updates:

```

import {
  hasUpdate,
  updateInfo,
  initUpdateNotifications,
  checkForUpdates,
  applyUpdate,
  dismissUpdate,
} from '@philjs/plugin-pwa';

// Initialize automatic update checking
const cleanup = initUpdateNotifications({
  checkInterval: 60 * 60 * 1000, // 1 hour
  autoCheck: true,
});

// React to update availability
createEffect(() => {
  if (hasUpdate()) {
    console.log('Update available!');
    const info = updateInfo();
    console.log('Version:', info?.version);
    console.log('Release notes:', info?.releaseNotes);
  }
});

// Show update prompt
function UpdatePrompt() {
  return createShow(
    () => hasUpdate(),
    () => (
      <div>
        <h2>Update Available</h2>
        <p>A new version is ready!</p>
        <button onClick={() => applyUpdate()}>
          Update Now
        </button>
        <button onClick={() => dismissUpdate()}>
          Later
        </button>
      </div>
    )
  );
}

// Manually check for updates
async function handleCheckUpdates() {
  const found = await checkForUpdates();
  if (found) {
    console.log('Updates available');
  }
}

cleanup();

```

Background Sync

Queue and sync data when the connection is restored:

```

import {
  isBackgroundSyncSupported,
  registerBackgroundSync,
  queueForSync,
  getSyncTags,
} from '@philjs/plugin-pwa';

// Check support
if (isBackgroundSyncSupported()) {
  console.log('Background Sync supported');
}

// Queue data for sync
async function saveOffline(data: any) {
  try {
    await queueForSync(data, 'notes-sync');
  } catch (error) {
    console.error('Failed to queue sync:', error);
  }
}

// Register background sync tag
async function registerSync() {
  try {
    await registerBackgroundSync('notes-sync');
    console.log('Background sync registered');
  } catch (error) {
    console.error('Background sync failed:', error);
  }
}

// Get all registered sync tags
async function getActiveSyncs() {
  const tags = await getSyncTags();
  console.log('Active syncs:', tags);
}

```

Complete Example

Here's a complete PWA setup example:

Build Script

```

// build.ts
import { writeFileSync } from 'fs';
import {
  generateServiceWorker,
  generateManifest,
  generatePWAMetaTags,
  getDefaultCacheRules,
  cacheRules,
} from '@philjs/plugin-pwa';

// Generate service worker
const swCode = generateServiceWorker({
  cachePrefix: 'my-app',
  cacheVersion: 'v1.0.0',
  precache: [
    '/index.html',
    '/app.js',
    '/app.css',
  ],
  runtimeCaching: [
    cacheRules.staticAssets(),
    cacheRules.images(),
    cacheRules.fonts(),
    ...cacheRules.googleFonts(),
    cacheRules.apiResponses('/api'),
  ],
  skipWaiting: true,
  clientsClaim: true,
  navigationPreload: true,
  offlineFallback: 'offline.html',
  backgroundSync: {
    enabled: true,
    queueName: 'api-queue',
  },
});
writeFileSync('public/sw.js', swCode);

// Generate manifest
const manifest = generateManifest({
  name: 'My Awesome App',
  short_name: 'MyApp',
  description: 'The most awesome app ever',
  start_url: '/',
  display: 'standalone',
  theme_color: '#667eea',
})

```

```

background_color: '#ffffff',
icons: [
  {
    src: '/icons/icon-192.png',
    sizes: '192x192',
    type: 'image/png',
    purpose: 'any',
  },
  {
    src: '/icons/icon-512.png',
    sizes: '512x512',
    type: 'image/png',
    purpose: 'any',
  },
  {
    src: '/icons/icon-maskable-192.png',
    sizes: '192x192',
    type: 'image/png',
    purpose: 'maskable',
  },
],
screenshots: [
  {
    src: '/screenshots/narrow-1.png',
    sizes: '540x720',
    type: 'image/png',
    form_factor: 'narrow',
  },
],
shortcuts: [
  {
    name: 'New Task',
    short_name: 'New',
    url: '/new',
    icons: [
      {
        src: '/icons/new-192.png',
        sizes: '192x192',
      },
    ],
  },
],
});

writeFileSync(
  'public/manifest.json',
  JSON.stringify(manifest, null, 2)
);

// Generate meta tags
const metaTags = generatePWAMetaTags({
  name: 'My Awesome App',
  short_name: 'MyApp',
  icons: manifest.icons,
});

console.log('PWA files generated successfully');

```

[HTML Setup](#)

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <!-- PWA Meta Tags -->
    <meta name="application-name" content="My Awesome App">
    <meta name="apple-mobile-web-app-capable" content="yes">
    <meta name="apple-mobile-web-app-status-bar-style" content="default">
    <meta name="apple-mobile-web-app-title" content="MyApp">
    <meta name="description" content="The most awesome app ever">
    <meta name="format-detection" content="telephone=no">
    <meta name="mobile-web-app-capable" content="yes">
    <meta name="theme-color" content="#667eea">

    <!-- App Icons -->
    <link rel="apple-touch-icon" sizes="192x192" href="/icons/icon-192.png">
    <link rel="apple-touch-icon" sizes="512x512" href="/icons/icon-512.png">

    <!-- Web App Manifest -->
    <link rel="manifest" href="/manifest.json">

    <title>My Awesome App</title>
    <link rel="stylesheet" href="/app.css">
  </head>
  <body>
    <div id="root"></div>
    <script src="/app.js"></script>
  </body>
</html>
```

App Component

```

// App.tsx
import { createEffect, onCleanup } from 'philjs-core';
import {
  registerServiceWorker,
  initOfflineDetection,
  initInstallPrompt,
  initUpdateNotifications,
  canInstall,
  isOnline,
  hasUpdate,
  showInstallPrompt,
  applyUpdate,
  dismissUpdate,
} from '@philjs/plugin-pwa';

export default function App() {
  createEffect(async () => {
    // Register service worker
    await registerServiceWorker('/sw.js');

    // Initialize features
    const cleanups = [
      initOfflineDetection(),
      initInstallPrompt(),
      initUpdateNotifications({
        checkInterval: 60 * 60 * 1000,
      }),
    ];
  });

  onCleanup(() => cleanups.forEach(c => c()));

  return (
    <>
    <MainApp />

    {/* Install Prompt */}
    <InstallPrompt />

    {/* Update Notification */}
    <UpdateNotification />

    {/* Offline Indicator */}
    <OfflineIndicator />
  );
}

function InstallPrompt() {
  return createShow(
    () => canInstall(),
    () => (
      <div style="position: fixed; bottom: 20px; right: 20px; padding: 20px; background: white; border-radius: 8px; box-shadow: 0 2px 10px rgba(0,0,0,0.2);">
        <h3>Install App</h3>
        <p>Install this app on your device for quick access</p>
        <button onClick={() => showInstallPrompt()}>Install</button>
      </div>
    )
  );
}

function UpdateNotification() {
  return createShow(
    () => hasUpdate(),
    () => (
      <div style="position: fixed; top: 20px; right: 20px; padding: 20px; background: #667eea; color: white; border-radius: 8px;">
        <h3>Update Available</h3>
        <p>A new version is ready to use</p>
        <button onClick={() => applyUpdate()}>Update Now</button>
        <button onClick={() => dismissUpdate()}>Later</button>
      </div>
    )
  );
}

function OfflineIndicator() {
  return createShow(
    () => !isOnline(),
    () => (
      <div style="position: fixed; top: 0; left: 0; right: 0; padding: 10px; background: #ff6b6b; color: white; text-align: center;">
        You are currently offline. Some features may be limited.
      </div>
    )
  );
}

```

Service Worker

- `generateServiceWorker(config)` - Generate service worker code
- `registerServiceWorker(scriptURL, options)` - Register service worker
- `unregisterServiceWorker()` - Unregister all service workers
- `isServiceWorkerRegistered()` - Check if service worker is registered
- `skipWaitingAndReload()` - Skip waiting and reload to new version
- `getServiceWorkerVersion()` - Get active service worker version

Manifest

- `generateManifest(config)` - Generate manifest object
- `createManifestJSON(config)` - Create manifest JSON string
- `injectManifestLink(path)` - Generate manifest link HTML
- `generatePWAMetaTags(config)` - Generate PWA meta tags

Offline Support

- `isOnline` - Signal for online/offline status
- `initOfflineDetection()` - Initialize offline detection
- `queueOfflineRequest(url, options)` - Queue request for offline sync
- `isOfflinePageCached(page)` - Check if offline page is cached
- `prefetchOfflinePage(page)` - Prefetch offline page

Install Prompts

- `canInstall` - Signal for install availability
- `isInstalled` - Signal for installation status
- `initInstallPrompt()` - Initialize install prompt
- `showInstallPrompt()` - Show install prompt
- `getInstallPrompt()` - Get deferred install prompt
- `checkCanInstall()` - Check if app can be installed
- `checkIsInstalled()` - Check if app is installed

Updates

- `hasUpdate` - Signal for update availability
- `updateInfo` - Signal with update information
- `initUpdateNotifications(options)` - Initialize update checking
- `checkForUpdates()` - Check for service worker updates
- `applyUpdate()` - Apply pending update
- `dismissUpdate()` - Dismiss update notification

Background Sync

- `isBackgroundSyncSupported()` - Check Background Sync support
- `registerBackgroundSync(tag, options)` - Register sync tag
- `getSyncTags()` - Get registered sync tags
- `queueForSync(data, tag)` - Queue data for sync

Cache Strategies

- `createCacheRule(pattern, strategy, options)` - Create cache rule
- `cacheRules` - Predefined cache rules
- `getDefaultCacheRules()` - Get all default rules

Best Practices

1. Progressive Enhancement

Always provide fallbacks for features that might not be supported:

```
import {
  registerServiceWorker,
  isBackgroundSyncSupported,
} from '@philjs/plugin-pwa';

// Service Worker
if ('serviceWorker' in navigator) {
  await registerServiceWorker('/sw.js');
}

// Background Sync (optional feature)
if (isBackgroundSyncSupported()) {
  await registerBackgroundSync('sync-tag');
}
```

2. Cache Versioning

Always update your cache version when assets change:

```
const swCode = generateServiceWorker({
  cacheVersion: 'v1.2.3', // Update when assets change
  precache: ['/index.html', '/app.js'],
});
```

3. Update Notifications

Inform users about updates but respect their choice:

```
// Don't force updates immediately
const cleanup = initUpdateNotifications({
  autoCheck: true,
  checkInterval: 60 * 60 * 1000, // Check every hour
});

// Show notification and let user decide
createEffect(() => {
  if (hasUpdate()) {
    // Show UI prompt, don't auto-apply
  }
});
```

4. Offline Experience

Always provide an offline page and queue important requests:

```
// Precache offline page
const swCode = generateServiceWorker({
  precache: ['/offline.html'],
  offlineFallback: '/offline.html',
});

// Queue API requests when offline
if (!isOnline()) {
  await queueOfflineRequest('/api/save', { method: 'POST' });
}
```

5. Icon Management

Provide icons in multiple sizes and formats:

```
const manifest = generateManifest({
  icons: [
    // 192x192 for Android
    { src: '/icons/192.png', sizes: '192x192', type: 'image/png' },
    // 512x512 for splash screen
    { src: '/icons/512.png', sizes: '512x512', type: 'image/png' },
    // Maskable icons for adaptive icons
    {
      src: '/icons/maskable-192.png',
      sizes: '192x192',
      purpose: 'maskable'
    },
  ],
});
```

Testing in Development

1. Chrome DevTools

Open DevTools and check: - **Application > Manifest** - See manifest configuration - **Application > Service Workers** - Check service worker status - **Application > Cache Storage** - Inspect cached files

2. Lighthouse

Run Lighthouse audit in Chrome: 1. Open DevTools 2. Click "Lighthouse" 3. Check "PWA" category 4. Run audit

3. Offline Testing

Simulate offline mode: 1. Open DevTools 2. Go to "Network" tab 3. Check "Offline" checkbox 4. Reload page

4. Local HTTPS

PWA features require HTTPS (except localhost):

```
# Using mkcert for Local HTTPS
mkcert -install
mkcert localhost 127.0.0.1 ::1

# Then use with your dev server
vite --https --cert localhost.pem --key localhost-key.pem
```

Production Deployment

1. Generate Assets

```
// build.ts
import { generateServiceWorker, generateManifest } from '@philjs/plugin-pwa';
import { writeFileSync } from 'fs';

// Generate with production cache version
const version = process.env.VERSION || '1.0.0';

const swCode = generateServiceWorker({
  cacheVersion: `v${version}`,
  precache: /* bundled files */,
  runtimeCaching: /* rules */,
});

writeFileSync('dist/sw.js', swCode);

const manifest = generateManifest({
  name: 'My App',
  start_url: '/',
  icons: /* icons */,
});

writeFileSync('dist/manifest.json', JSON.stringify(manifest));
```

2. Add Headers (Nginx)

```
# Serve service worker with no cache
location = /sw.js {
  add_header Cache-Control "public, max-age=0, must-revalidate";
  add_header Service-Worker-Allowed "/";
}

# Serve manifest with short cache
location = /manifest.json {
  add_header Cache-Control "public, max-age=3600";
}

# Long cache for assets
location ~* \.(js|css|png|jpg|gif|svg|woff|woff2)$ {
  add_header Cache-Control "public, max-age=31536000, immutable";
}
```

3. Monitor Updates

Track update adoption:

```
// Track when users apply updates
window.addEventListener('pwa-update-ready', () => {
  // Send telemetry event
  fetch('/api/telemetry/pwa-update', {
    method: 'POST',
    body: JSON.stringify({ event: 'update-available' }),
  });
});

// Track installations
window.addEventListener('pwa-installed', () => {
  fetch('/api/telemetry/pwa-install', {
    method: 'POST',
    body: JSON.stringify({ event: 'app-installed' }),
  });
});
```

Troubleshooting

Service Worker Not Registering

1. Check HTTPS/localhost requirement
2. Verify service worker script URL is correct
3. Check browser console for errors
4. Verify service worker file exists and is accessible

```
const registration = await registerServiceWorker('/sw.js');
if (!registration) {
  console.error('Service worker registration failed');
}
```

Cache Not Working

1. Check cache names don't conflict
2. Verify cache version is updated
3. Inspect cache in DevTools > Application > Cache Storage
4. Clear old caches manually

```
// Clear all caches
navigator.serviceWorker.getRegistration().then(reg => {
  reg.active.postMessage({ type: 'CLEAR_CACHE' });
});
```

Updates Not Detected

1. Ensure service worker file has changed
2. Check update check interval isn't too long
3. Verify service worker activation
4. Check DevTools for pwa-update-available event

```
// Force update check
await checkForUpdates();
console.log('Has update:', hasUpdate());
```

Offline Page Not Showing

1. Ensure offline page is in precache
2. Verify offlineFallback path is correct
3. Check network conditions in DevTools
4. Ensure offline page is valid HTML

```
const swCode = generateServiceWorker({
  precache: ['/offline.html'],
  offlineFallback: '/offline.html',
});
```

Install Prompt Not Appearing

1. Ensure manifest.json is valid
2. Check PWA requirements (HTTPS, icons, manifest)
3. Verify beforeinstallprompt event fires
4. App must meet PWA installability criteria

```
window.addEventListener('beforeinstallprompt', (e) => {
  console.log('Install prompt available');
  // Event fires when app is installable
});
```

Related Documentation

- [Web App Manifest Spec](#)
- [Service Workers API](#)
- [Cache API](#)
- [Web App Install Banners](#)
- [PWA on iOS](#)
- [Background Sync](#)

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-plugin-pwa/src/index.ts

Public API

- Direct exports: (none detected)
- Re-exported names: (none detected)
- Re-exported modules: ./background-sync.js, ./cache-strategies.js, ./Install.js, ./manifest.js, ./offline.js, ./service-worker.js, ./types.js, ./updates.js

License

MIT

@philjs/plugin-seo - Type: Node package - Purpose: SEO plugin for PhilJS with meta tags, OpenGraph, JSON-LD, and sitemap generation - Version: 0.1.0 - Location: packages/philjs-plugin-seo - Entry points: packages/philjs-plugin-seo/src/index.ts, packages/philjs-plugin-seo/src/head.ts - Keywords: philjs, plugin, seo, meta-tags, opengraph, twitter-cards, json-ld, structured-data, sitemap, robots

philjs-plugin-seo

Comprehensive SEO plugin for PhilJS with meta tags, OpenGraph, Twitter Cards, JSON-LD structured data, sitemap generation, and robots.txt support.

Installation

```
pnpm add philjs-plugin-seo
```

Quick Start

1. Add the Plugin

```
// vite.config.ts
import { defineConfig } from 'vite';
import { createSEOPPlugin } from 'philjs-plugin-seo';

export default defineConfig({
  plugins: [
    createSEOPPlugin({
      baseUrl: 'https://example.com',
      defaults: {
        titleTemplate: '%s | My Site',
        description: 'Default site description',
      },
      openGraph: {
        type: 'website',
        siteName: 'My Site',
      },
      twitter: {
        card: 'summary_large_image',
        site: '@mysite',
      },
      sitemap: true,
      robots: true,
    }).vitePlugin(),
  ],
});
});
```

2. Use SEO in Pages

```
import { setPageSEO } from './lib/seo';

function ProductPage({ product }) {
  setPageSEO({
    meta: {
      title: product.name,
      description: product.description,
    },
    openGraph: {
      type: 'product',
      image: product.image,
      product: {
        price: { amount: product.price, currency: 'USD' },
        availability: 'instock',
      },
    },
    jsonLd: {
      '@type': 'Product',
      name: product.name,
      description: product.description,
      image: product.image,
      offers: {
        '@type': 'Offer',
        price: product.price,
        priceCurrency: 'USD',
        availability: 'https://schema.org/InStock',
      },
    },
  });
  return <div>...</div>;
}
```

Features

Meta Tags

```
setPageSEO({
  meta: {
    title: 'Page Title',
    titleTemplate: '%s | Site Name', // %s replaced with title
    description: 'Page description for search engines',
    keywords: ['keyword1', 'keyword2'],
    canonical: 'https://example.com/page',
    robots: { index: true, follow: true },
    author: 'Author Name',
    themeColor: '#ffffff',
  },
});
```

OpenGraph Tags

```

setPageSEO({
  openGraph: {
    title: 'Share Title',
    description: 'Share description',
    type: 'article',
    url: 'https://example.com/article',
    siteName: 'My Site',
    locale: 'en_US',
    image: {
      url: 'https://example.com/og-image.jpg',
      width: 1200,
      height: 630,
      alt: 'Image description',
    },
    article: {
      publishedTime: '2024-01-15T00:00:00Z',
      modifiedTime: '2024-01-16T00:00:00Z',
      author: ['Author Name'],
      section: 'Technology',
      tag: ['javascript', 'web'],
    },
  },
});

```

Twitter Cards

```

setPageSEO({
  twitter: {
    card: 'summary_large_image',
    site: '@sitehandle',
    creator: '@authorhandle',
    title: 'Tweet Title',
    description: 'Tweet description',
    image: 'https://example.com/twitter-image.jpg',
    imageAlt: 'Image description',
  },
});

```

JSON-LD Structured Data

Organization

```

import { createOrganization } from './lib/seo';

setPageSEO({
  jsonLd: createOrganization({
    name: 'Company Name',
    url: 'https://example.com',
    logo: 'https://example.com/logo.png',
    sameAs: [
      'https://twitter.com/company',
      'https://facebook.com/company',
      'https://linkedin.com/company/company',
    ],
  }),
});

```

Breadcrumbs

```

import { createBreadcrumbs } from './lib/seo';

setPageSEO({
  jsonLd: createBreadcrumbs([
    { name: 'Home', url: 'https://example.com' },
    { name: 'Products', url: 'https://example.com/products' },
    { name: 'Widget', url: 'https://example.com/products/widget' },
  ]),
});

```

FAQ Page

```

import { createFAQ } from './lib/seo';

setPageSEO({
  jsonLd: createFAQ([
    {
      question: 'What is PhilJS?',
      answer: 'PhilJS is a modern JavaScript framework...',
    },
    {
      question: 'How do I install it?',
      answer: 'Run npx add philjs-core...',
    },
  ]),
});

```

Article

```
setPageSEO({
  jsonLd: {
    '@type': 'Article',
    headline: 'Article Title',
    description: 'Article description',
    image: ['https://example.com/image.jpg'],
    datePublished: '2024-01-15T00:00:00Z',
    dateModified: '2024-01-16T00:00:00Z',
    author: {
      '@type': 'Person',
      name: 'Author Name',
      url: 'https://example.com/author',
    },
    publisher: {
      '@type': 'Organization',
      name: 'Publisher Name',
      logo: {
        '@type': 'ImageObject',
        url: 'https://example.com/logo.png',
      },
    },
  },
});
```

Sitemap Generation

```
createSEOPPlugin({
  baseUrl: 'https://example.com',
  sitemap: {
    output: 'sitemap.xml',
    defaultChangefreq: 'weekly',
    defaultPriority: 0.5,
    customEntries: [
      { loc: 'https://example.com/', priority: 1.0, changefreq: 'daily' },
      { loc: 'https://example.com/about', priority: 0.8 },
      {
        loc: 'https://example.com/products',
        priority: 0.9,
        images: [
          { loc: 'https://example.com/product1.jpg', title: 'Product 1' },
        ],
      },
    ],
  },
});
```

Robots.txt Generation

```
createSEOPPlugin({
  baseUrl: 'https://example.com',
  robots: {
    rules: [
      {
        userAgent: '*',
        allow: '/',
        disallow: ['/admin', '/private'],
      },
      {
        userAgent: 'Googlebot',
        allow: '/',
        crawlDelay: 1,
      },
    ],
    sitemaps: ['https://example.com/sitemap.xml'],
  },
});
```

API Reference

Plugin Configuration

Option	Type	Description
baseUrl	string	Base URL for the site
defaults	MetaTags	Default meta tags for all pages
openGraph	OpenGraphTags	Default OpenGraph tags
twitter	TwitterTags	Default Twitter Card tags
jsonLd	JsonLd \ JsonLd[]	Site-wide JSON-LD
sitemap	boolean \ SitemapConfig	Sitemap configuration
robots	boolean \ RobotsTxtConfig	Robots.txt configuration
trailingSlash	boolean	Whether to use trailing slashes

Client Functions

Function	Description
setPageSEO(seo)	Update page SEO (client-side)
generatePageSEO(seo)	Generate SEO HTML string (SSR)
mergeSEO(...configs)	Merge multiple SEO configs
createBreadcrumbs(items)	Create breadcrumb JSON-LD
createFAQ(questions)	Create FAQ page JSON-LD
createOrganization(org)	Create organization JSON-LD
createWebSite(site)	Create website JSON-LD

Robots Directives

```
interface RobotsDirectives {
  index?: boolean; // Allow indexing
  follow?: boolean; // Follow Links
  noarchive?: boolean; // No cached version
  nosnippet?: boolean; // No snippet in results
  noimageindex?: boolean; // No image indexing
  notranslate?: boolean; // No translation
  maxSnippet?: number; // Max snippet length
  maxImagePreview?: 'none' | 'standard' | 'large';
  maxVideoPreview?: number;
}
```

SSR Integration

For server-side rendering, use `generatePageSEO`:

```
import { generatePageSEO } from './lib/seo';
import { renderToString } from 'philjs-core/render-to-string';

const seoHtml = generatePageSEO({
  meta: { title: 'My Page' },
  openGraph: { type: 'website' },
});

const html = `
<!DOCTYPE html>
<html>
<head>
  ${seoHtml}
</head>
<body>
  ${await renderToString(<App />)};
</body>
</html>
`;
```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./head`
- Source files: `packages/philjs-plugin-seo/src/index.ts`, `packages/philjs-plugin-seo/src/head.ts`

Public API

- Direct exports: `JsonLd`, `MetaTags`, `OpenGraphTags`, `PageSEO`, `TwitterTags`, `createBreadcrumbs`, `createFAQ`, `createOrganization`, `createSEOPPlugin`, `createWebSite`, `defaultSEO`, `generateJsonLd`, `generateLinkTag`, `generateMetaTag`, `generateMetaTags`, `generateOpenGraphTags`, `generatePageSEO`, `generateRobotsContent`, `generateSEOHead`, `generateTwitterTags`, `mergeSEO`, `setPageSEO`, `updateHead`
- Re-exported names: `ArticleJsonLd`, `BreadcrumbJsonLd`, `FAQJsonLd`, `JsonLd`, `JsonLdType`, `MetaTags`, `OpenGraphImage`, `OpenGraphTags`, `OrganizationJsonLd`, `PageSEO`, `ProductJsonLd`, `RobotsDirectives`, `RobotsTxtConfig`, `SEOPPluginConfig`, `SitemapConfig`, `SitemapEntry`, `TwitterTags`, `WebSiteJsonLd`, `createBreadcrumbs`, `createFAQ`, `createOrganization`, `createWebSite`, `generateJsonLd`, `generateMetaTags`, `generateOpenGraphTags`, `generateSEOHead`, `generateTwitterTags`, `mergeSEO`, `updateHead`
- Re-exported modules: `./head.js`, `./types.js`

License

MIT

@philjs/plugin-tailwind - Type: Node package - Purpose: Automatic Tailwind CSS setup and optimization for PhilJS - Version: 0.1.0 - Location: packages/philjs-plugin-tailwind - Entry points: packages/philjs-plugin-tailwind/src/index.ts, packages/philjs-plugin-tailwind/src/utils.ts, packages/philjs-plugin-tailwind/src/content-detector.ts, packages/philjs-plugin-tailwind/src/theme-generator.ts - Keywords: philjs, plugin, tailwind, tailwindcss, css, styling, design-system, utility-first, jit, dark-mode

philjs-plugin-tailwind

Automatic Tailwind CSS setup and optimization for PhilJS applications.

Features

- Zero Config** - Automatic Tailwind CSS setup with sensible defaults
- JIT Mode** - Just-in-time compilation for faster builds
- Dark Mode** - Built-in dark mode support

- **Optimization** - Automatic purging and minification in production
- **Custom Utilities** - Pre-configured utility classes
- **Type Safety** - Full TypeScript support

Installation

```
# Using PhilJS CLI (recommended)
philjs plugin add philjs-plugin-tailwind

# Or with package manager
npm install philjs-plugin-tailwind tailwindcss autoprefixer postcss
```

Usage

Basic Setup

```
// philjs.config.ts
import { defineConfig } from 'philjs-core';
import tailwind from 'philjs-plugin-tailwind';

export default defineConfig({
  plugins: [
    tailwind(),
  ],
});
```

Custom Configuration

```
tailwind({
  jit: true,
  darkMode: 'class',
  content: ['./src/**/*.{ts,tsx}', './index.html'],
  theme: {
    extend: {
      colors: {
        primary: '#3b82f6',
        secondary: '#8b5cf6',
      },
      fontFamily: {
        sans: ['Inter', 'sans-serif'],
      },
    },
    plugins: [
      // Tailwind plugins
    ],
  })
})
```

Configuration Options

Option	Type	Description	Default
jit	boolean	Enable JIT mode	true
darkMode	'media' \ 'class' \ false	Dark mode strategy	'class'
content	string[]	Content paths to scan	['./src/**/*.{js,jsx,ts,tsx}', './index.html']
theme	object	Theme customization	{}
plugins	array	Tailwind plugins	[]
optimization	object	Optimization settings	See below

Optimization Options

```
optimization: {
  purge: true,           // Purge unused styles in production
  minify: true,          // Minify CSS output
  removeComments: true // Remove comments
}
```

Generated Files

The plugin automatically creates:

- `tailwind.config.js` - Tailwind configuration
- `postcss.config.js` - PostCSS configuration
- `src/styles/tailwind.css` - Base styles with custom utilities

Custom Utilities

Pre-configured utility classes:

```
/* Buttons */
.btn - Base button styles
.btn-primary - Primary button
.btn-secondary - Secondary button

/* Components */
.card - Card component
.input - Input field

/* Utilities */
.text-balance - Balanced text wrap
```

Usage in Components

```
import './styles/tailwind.css';

export function App() {
  return (
    <div class="container mx-auto p-4">
      <h1 class="text-3xl font-bold text-primary">
        Hello PhilJS + Tailwind!
      </h1>

      <button class="btn btn-primary mt-4">
        Click me
      </button>

      <div class="card mt-6">
        <p>This is a card component</p>
      </div>
    </div>
  );
}
```

Dark Mode

```
// Toggle dark mode
document.documentElement.classList.toggle('dark');

// Component with dark mode styles
export function Card() {
  return (
    <div class="bg-white dark:bg-gray-800 text-black dark:text-white">
      Content adapts to dark mode
    </div>
  );
}
```

Utilities

Class Merging

```
import { tailwindUtils } from 'philjs-plugin-tailwind';

const classes = tailwindUtils.mergeClasses(
  'px-4 py-2',
  'px-6', // Overwrites px-4
  'bg-blue-500'
);
// Result: 'px-6 py-2 bg-blue-500'
```

CSS Variables to Theme

```
const theme = tailwindUtils.cssVarsToTheme({
  '--primary-color': '#3b82f6',
  '--secondary-color': '#8b5cf6',
});
// Converts to Tailwind theme format
```

TypeScript

Full TypeScript support with type definitions:

```
import type { TailwindPluginConfig } from 'philjs-plugin-tailwind';

const config: TailwindPluginConfig = {
  darkMode: 'class',
  theme: {
    extend: {
      // Type-safe theme configuration
    },
  },
};
```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: ./utils, ./content-detector, ./theme-generator
- Source files: packages/philjs-plugin-tailwind/src/index.ts, packages/philjs-plugin-tailwind/src/utils.ts, packages/philjs-plugin-tailwind/src/content-detector.ts, packages/philjs-plugin-tailwind/src/theme-generator.ts

Public API

- Direct exports: ClassName, ColorPalette, ContentDetector, ContentDetectorOptions, DetectedContent, TailwindPluginConfig, ThemeConfig, arbitrary, dlsx, cn, container, createTailwindPlugin, createVariants, cssVarToClass, cssVarsToTheme, dark, detectContentPaths, expandContentPatterns, extractClasses, generateBorderRadiusScale, generateBrandTheme, generateBreakpoints, generateColorPalette, generateCompleteTheme, generateFontFamilies, generateShadowScale, generateSpacingScale, generateTypographyScale, isValidClass, mergeThemes, optimizeContentPatterns, presetThemes, responsive, sortClasses, tailwindUtils, validateContentPatterns, withStates
- Re-exported names: ColorPalette, ThemeConfig, cssVarsToTheme, generateBorderRadiusScale, generateBrandTheme, generateBreakpoints, generateColorPalette, generateCompleteTheme, generateFontFamilies, generateShadowScale, generateSpacingScale, generateTypographyScale, presetThemes
- Re-exported modules: ./content-detector.js, ./optimizer.js, ./theme-generator.js, ./utils.js

License

MIT

@philjs/plugins - Type: Node package - Purpose: Plugin system for PhilJS - create and use framework plugins - Version: 0.1.0 - Location: packages/philjs-plugins - Entry points: packages/philjs-plugins/src/index.ts, packages/philjs-plugins/src/registry.ts - Keywords: philjs, plugins, extensions, integrations

philjs-plugins

Plugin system for PhilJS - Create and use framework plugins to extend functionality.

Overview

Plugin system for PhilJS - create and use framework plugins

Focus Areas

- philjs, plugins, extensions, integrations

Entry Points

- packages/philjs-plugins/src/index.ts
- packages/philjs-plugins/src/registry.ts

Quick Start

```
import { BuildResult, Middleware, PhilJSApp } from '@philjs/plugins';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- BuildResult
- Middleware
- PhilJSApp
- Plugin
- PluginCategory
- PluginContext
- PluginHooks
- PluginInfo
- PluginRegistry
- RenderContext
- RouteDefinition
- ServerHandler

Features

- **Plugin Architecture** - Extend PhilJS with custom plugins
- **Lifecycle Hooks** - Hook into framework lifecycle events
- **Plugin Registry** - Discover and manage plugins
- **Type Safety** - Fully typed plugin API
- **Hot Reloading** - Plugins support HMR in development

Installation

```
pnpm add philjs-plugins
```

Quick Start

Create a Plugin

```

import { definePlugin } from 'philjs-plugins';

export default definePlugin({
  name: 'my-plugin',
  version: '1.0.0',

  setup({ app, config }) {
    // Plugin initialization
    console.log('Plugin loaded!');

    // Return cleanup function
    return () => {
      console.log('Plugin unloaded');
    };
  }
});

```

Use a Plugin

```

import { usePlugins } from 'philjs-plugins';
import myPlugin from './my-plugin';

usePlugins([myPlugin]);

```

Documentation

For more information, see the PhilJS documentation.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./registry`
- Source files: `packages/philjs-plugins/src/index.ts`, `packages/philjs-plugins/src/registry.ts`

Public API

- Direct exports: `BuildResult`, `Middleware`, `PhilJSApp`, `Plugin`, `PluginCategory`, `PluginContext`, `PluginHooks`, `PluginInfo`, `PluginRegistry`, `RenderContext`, `RouteDefinition`, `ServerHandler`, `callHook`, `createPlugin`, `definePlugin`, `fetchPluginInfo`, `getInstalledPlugins`, `getPluginCategories`, `getPluginsByCategory`, `getProvider`, `installPlugin`, `isPluginInstalled`, `searchPlugins`, `uninstallPlugin`
- Re-exported names: `PluginRegistry`, `fetchPluginInfo`, `searchPlugins`
- Re-exported modules: `./registry.js`

License

MIT

@philjs/poem - Type: Node package - Purpose: Poem web framework integration for PhilJS - SSR, middleware, extractors, and WebSocket support - Version: 0.1.0 - Location: packages/philjs-poem - Entry points: packages/philjs-poem/src/index.ts, packages/philjs-poem/src/middleware.ts, packages/philjs-poem/src/extractors.ts, packages/philjs-poem/src/endpoints.ts, packages/philjs-poem/src/responses.ts, packages/philjs-poem/src/websocket.ts, packages/philjs-poem/src/openapi.ts, packages/philjs-poem/src/ssr.ts - Keywords: philjs, poem, rust, ssr, websocket, middleware, extractor, openapi, swagger

@philjs/poem

Poem web framework integration for PhilJS applications. Connect your Rust Poem backend with PhilJS frontend components for type-safe, high-performance full-stack development.

Installation

```

npm install @philjs/poem
# or
yarn add @philjs/poem
# or
pnpm add @philjs/poem

```

Rust (Cargo.toml):

```

[dependencies]
philjs-poem = "0.1"
poem = "1.3"

```

Basic Usage

TypeScript:

```

import { createPoemClient, usePoemQuery } from '@philjs/poem';

const client = createPoemClient({
  baseUrl: 'http://localhost:3000',
});

function Users() {
  const { data, isLoading } = usePoemQuery('/api/users');

  if (isLoading) return <div>Loading...</div>;
  return <UserList users={data} />;
}

```

Rust:

```
use philjs_poem::{PhilJSHandler, generate_types};
use poem::{Route, Server};

#[derive(Serialize, TypeScript)]
struct User {
    id: u64,
    name: String,
}

#[handler]
async fn get_users() -> Json<Vec<User>> {
    Json(fetch_users().await)
}

fn main() {
    generate_types!("./frontend/src/types");

    let app = Route::new()
        .at("/api/users", get(get_users));
    Server::bind("0.0.0.0:3000").run(app);
}
```

Features

- **Type Generation** - Auto-generate TypeScript types from Rust structs
- **API Client** - Type-safe API client for Poem endpoints
- **SSR Support** - Server-side rendering with Poem
- **WebSocket** - Real-time communication via WebSockets
- **File Upload** - Multipart file upload handling
- **Authentication** - JWT and session auth integration
- **OpenAPI** - Automatic OpenAPI spec generation
- **Middleware** - CORS, compression, logging middleware
- **Error Handling** - Unified error types across stack
- **Hot Reload** - Development hot reloading

React Hooks

Hook	Description
usePoemQuery	Fetch data from Poem endpoint
usePoemMutation	POST/PUT/DELETE mutations
usePoemWebSocket	WebSocket connection
usePoemAuth	Authentication state

CLI Tools

```
# Generate TypeScript types from Rust
npx philjs-poem generate-types

# Start development server with hot reload
npx philjs-poem dev
```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: ./middleware, ./extractors, ./endpoints, ./responses, ./websocket, ./openapi, ./ssr
- Source files: packages/philjs-poem/src/index.ts, packages/philjs-poem/src/middleware.ts, packages/philjs-poem/src/extractors.ts, packages/philjs-poem/src/endpoints.ts, packages/philjs-poem/src/responses.ts, packages/philjs-poem/src/websocket.ts, packages/philjs-poem/src/openapi.ts, packages/philjs-poem/src/ssr.ts

Public API

- Direct exports: AuthExtractor, AuthExtractorConfig, AuthUser, BroadcastChannel, BroadcastManager, CORSMiddleware, CORSMiddlewareConfig, CRUOptions, CompressionMiddleware, CompressionMiddlewareConfig, CookieExtractor, DEFAULT_WS_OPTIONS, DataExtractor, EndpointBuilder, EndpointDefinition, ErrorResponse, FileResponse, FormExtractor, GeneratedRustCode, HeadManager, HeaderExtractor, HtmlResponse, HttpMethod, JsonExtractor, JsonExtractorConfig, JsonResponse, LiveViewSocketBuilder, MiddlewareAction, MiddlewareComposer, MiddlewareHandler, MultipartExtractor, MultipartFile, OAuthFlow, OpenAPIBuilder, OpenAPIDataType, OpenAPIEndpointBuilder, OpenAPIFormat, OpenAPIParameter, OpenAPIParameterIn, OpenAPIRequestBody, OpenAPISecurityScheme, OpenAPISpec, OpenAPIStringFormat, OperationBuilder, PathExtractor, PoemAppBuilder, PresenceEntry, PresenceState, PresenceTracker, QueryExtractor, QueryExtractorConfig, RateLimitMiddleware, RateLimitMiddlewareConfig, RedirectResponse, RouteGroup, RustHandlerOptions, RustTypeMapping, SSRCache, SSRChunk, SSRContext, SSRContextData, SSRContextExtractor, SSRDocument, SSRHead, SSRLink, SSRMiddleware, SSRMiddlewareConfig, SSRPageOptions, SSRRenderOptions, SSRRender, SSRResult, SecurityMiddleware, SecurityMiddlewareConfig, SecuritySchemaType, StreamResponse, TracingMiddleware, TracingMiddlewareConfig, WebSocketConfig, apiEndpoint, arraySchema, auth, booleanSchema, composeMiddleware, configureWebSocket, createBroadcastManager, createCORSMiddleware, createCompressionMiddleware, createExtractor, createHeadManager, createLiveViewSocket, createPoemApp, createPresenceTracker, createRateLimitMiddleware, createSSRCache, createSSRDocument, createSSRMiddleware, createSSRRenderer, createSecurityMiddleware, createTracingMiddleware, crud, endpoint, error, file, generateCrudAPI, generateFullRustAPI, generateRustAPI, generateRustDbService, generateRustErrorTypes, generateRustHandler, generateRustRequestStruct, generateRustResponseEnum, generateRustValidationHelpers, group, healthCheck, html, integerSchema, json, mapToRustType, numberSchema, objectSchema, openapi, operation, optionalAuth, query, readinessProbe, redirect, refSchema, renderDocument, renderToString, rustCodeGenerators, ssrContext, ssrPage, ssrPages, stream, stringSchema
- Re-exported names: // from endpoints - excluding openapi to avoid conflict EndpointBuilder, // from extractors - excluding json to avoid conflict Extractor, AuthExtractor,

AuthExtractorConfig, AuthUser, CRUOOptions, CookieExtractor, DataExtractor, EndpointDefinition, FormExtractor, GeneratedRustCode, HeaderExtractor, HttpMethod, JsonExtractor, JsonExtractorConfig, MultipartExtractor, MultipartFile, OpenAPIEndpointBuilder, PathExtractor, QueryExtractor, QueryExtractorConfig, RouteGroup, RustHandlerOptions, RustTypeMapping, SSRContextData, SSRContextExtractor, SSRPageOptions, apiEndpoint, auth, createExtractor, crud, endpoint, endpointsOpenapi, extractorJson, generateCrudAPI, generateFullRustAPI, generateRustAPI, generateRustDBService, generateRustErrorTypes, generateRustHandler, generateRustRequestStruct, generateRustResponseEnum, generateRustValidationHelpers, group, healthCheck, mapToRustType, optionalAuth, query, readinessProbe, rustCodeGenerators, ssrContext, ssrPage, ssrPages

- Re-exported modules: ./endpoints.js, ./extractors.js, ./middleware.js, ./openapi.js, ./responses.js, ./ssr.js, ./types.js, ./websocket.js

License

MIT

@philjs/pwa - Type: Node package - Purpose: Zero-config PWA generation for PhilJS - service worker, manifest, push notifications - Version: 0.1.0 - Location: packages/philjs-pwa - Entry points: packages/philjs-pwa/src/index.ts - Keywords: philjs, pwa, service-worker, manifest, push-notifications, offline

@philjs/pwa

Zero-config PWA generation for PhilJS - service worker, manifest, push notifications

Overview

Zero-config PWA generation for PhilJS - service worker, manifest, push notifications

Focus Areas

- philjs, pwa, service-worker, manifest, push-notifications, offline

Entry Points

- packages/philjs-pwa/src/index.ts

Quick Start

```
import { CacheStrategy, InstallPromptEvent, ManifestGenerator } from '@philjs/pwa';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- CacheStrategy
- InstallPromptEvent
- ManifestGenerator
- NotificationOptions
- PWAConfig
- PWAFileHandler
- PWAIIcon
- PWAManager
- PWAScreenshot
- PWAShareTarget
- PWAShortcut
- PWASTate

Install

```
pnpm add @philjs/pwa
```

Usage

```
import { CacheStrategy, InstallPromptEvent, ManifestGenerator } from '@philjs/pwa';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-pwa/src/index.ts

Public API

- Direct exports: CacheStrategy, InstallPromptEvent, ManifestGenerator, NotificationOptions, PWAConfig, PWAFileHandler, PWAIIcon, PWAManager, PWAScreenshot, PWAShareTarget, PWAShortcut, PWASTate, ServiceWorkerGenerator, pwaPlugin, useInstallPrompt, useOnlineStatus, usePWA
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/python - Type: Node package - Purpose: Python bindings for PhilJS AI/ML server functions - LLM, embeddings, and ML model serving - Version: 0.1.0 - Location: packages/philjs-python - Entry points: packages/philjs-python/src/index.ts, ./cli, packages/philjs-python/src/llm.ts, packages/philjs-python/src/embeddings.ts, packages/philjs-python/src/server.ts - Keywords: philjs, python, ai, ml, llm, embeddings, openai, anthropic, langchain, transformers, pytorch, tensorflow

philjs-python

Python bindings for PhilJS AI/ML server functions - LLM, embeddings, and ML model serving.

Requirements

- **Node.js 24** or higher
- **TypeScript 6** or higher
- **ESM only** - CommonJS is not supported
- **Python 3.11+** for Python-side features

Installation

```
pnpm add @philjs/python
```

Features

- **LLM Integration** - Connect to OpenAI, Anthropic, and local models
- **Embeddings** - Generate and manage vector embeddings
- **Model Serving** - Serve ML models via HTTP endpoints
- **PyTorch/TensorFlow** - Bridge to Python ML frameworks
- **LangChain Support** - Integration with LangChain workflows

Quick Start

LLM Client

```
import { createLLMClient } from '@philjs/python/llm';

const llm = createLLMClient({
  provider: 'openai',
  model: 'gpt-4',
  apiKey: process.env.OPENAI_API_KEY,
});

const response = await llm.complete({
  prompt: 'Explain TypeScript generics',
  maxTokens: 500,
});

console.log(response.text);
```

Embeddings

```
import { createEmbeddingClient } from '@philjs/python/embeddings';

const embeddings = createEmbeddingClient({
  model: 'text-embedding-ada-002',
});

const vectors = await embeddings.embed([
  'Hello world',
  'TypeScript is great',
]);

console.log(vectors[0].length); // 1536 dimensions
```

Python Server

```
import { createPythonServer } from '@philjs/python/server';

const server = await createPythonServer({
  port: 8000,
  models: ['./models/sentiment.pt'],
});

// Call Python functions from TypeScript
const result = await server.invoke('predict_sentiment', {
  text: 'I love this product!',
});

console.log(result); // { sentiment: 'positive', score: 0.95 }
```

ES2024 Features

`Promise.withResolvers()` for Streaming

```

import { createLLMClient } from '@philjs/python/llm';

const llm = createLLMClient({ provider: 'anthropic', model: 'claude-3' });

// Stream responses with cancellation support
function streamWithCancel(prompt: string) {
  const { promise, resolve, reject } = Promise.withResolvers<string>();
  let cancelled = false;
  let fullText = '';

  llm.stream({ prompt }).then(async (stream) => {
    for await (const chunk of stream) {
      if (cancelled) break;
      fullText += chunk.text;
    }
    if (!cancelled) resolve(fullText);
  }).catch(reject);

  return {
    promise,
    cancel: () => { cancelled = true; },
  };
}

```

Object.groupBy() for Model Results

```

import { createEmbeddingClient } from '@philjs/python/embeddings';

interface Document {
  id: string;
  text: string;
  category: string;
}

const embeddings = createEmbeddingClient({ model: 'text-embedding-ada-002' });

async function categorizeDocuments(docs: Document[]) {
  const results = await embeddings.embedBatch(docs.map(d => d.text));

  // Group documents by category using ES2024 Object.groupBy()
  const grouped = Object.groupBy(docs, doc => doc.category);

  return {
    technical: grouped.technical ?? [],
    business: grouped.business ?? [],
    general: grouped.general ?? [],
  };
}

```

Resource Management with using

```

import { createPythonServer } from '@philjs/python/server';

// Automatic cleanup with TypeScript 6 explicit resource management
async function runMLPipeline() {
  await using server = await createPythonServer({
    port: 8000,
    [Symbol.asyncDispose]: async () => {
      await server.shutdown();
      console.log('Python server shut down');
    }
  });

  // Server automatically shuts down when scope exits
  return server.invoke('train_model', { epochs: 10 });
}

```

CLI

```

# Start Python server
philjs-python serve --port 8000

# Run Python script with PhilJS bridge
philjs-python run ./scripts/train.py

# Generate TypeScript types from Python
philjs-python types ./models/schema.py

```

API Reference

LLM

Function	Description
<code>createLLMClient(config)</code>	Create LLM client instance
<code>llm.complete(options)</code>	Generate text completion
<code>llm.stream(options)</code>	Stream text generation
<code>llm.chat(messages)</code>	Multi-turn chat completion

Embeddings		
Function	Description	
<code>createEmbeddingClient(config)</code>	Create embedding client	
<code>embeddings.embed(texts)</code>	Generate embeddings	
<code>embeddings.embedBatch(texts)</code>	Batch embed with progress	
<code>embeddings.similarity(a, b)</code>	Calculate cosine similarity	

Server		
Function	Description	
<code>createPythonServer(config)</code>	Start Python server	
<code>server.invoke(fn, args)</code>	Call Python function	
<code>server.shutdown()</code>	Stop the server	

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `..`, `/cli`, `/llm`, `/embeddings`, `/server`
- Source files: `packages/philjs-python/src/index.ts`, `packages/philjs-python/src/llm.ts`, `packages/philjs-python/src/embeddings.ts`, `packages/philjs-python/src/server.ts`

Public API

- Direct exports: `BatchOptions`, `Embeddings`, `LLM`, `PythonServer`, `SimilarityResult`, `StreamChunk`, `checkPythonInstalled`, `createAnthropic`, `createEmbeddings`, `createLLM`, `createOpenAI`, `createOpenAIEmbeddings`, `createOpenAILargeEmbeddings`, `createPythonServer`, `detectProvider`, `embeddings`, `initPythonProject`, `llm`
- Re-exported names: (none detected)
- Re-exported modules: `./embeddings.js`, `./llm.js`, `./server.js`, `./types.js`

License

MIT

@philjs/qr - Type: Node package - Purpose: QR code generation and scanning for PhilJS - customizable styles, logos, camera scanning - Version: 0.1.0 - Location: packages/philjs-qr - Entry points: packages/philjs-qr/src/index.ts - Keywords: philjs, qr, qrcode, barcode, scanner, camera

@philjs/qr

QR code generation and scanning for PhilJS - customizable styles, logos, camera scanning

Overview

QR code generation and scanning for PhilJS - customizable styles, logos, camera scanning

Focus Areas

- `philjs`, `qr`, `qrcode`, `barcode`, `scanner`, `camera`

Entry Points

- `packages/philjs-qr/src/index.ts`

Quick Start

```
import { batchGenerateQR, createEmailQR, createEventQR } from '@philjs/qr';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- `batchGenerateQR`
- `createEmailQR`
- `createEventQR`
- `createGeoQR`
- `createSMSQR`
- `createVCardQR`
- `createWiFiQR`
- `generateQRCode`
- `generateQRCodeCanvas`
- `generateQRCodeDataURL`
- `validateQRData`

Install

```
pnpm add @philjs/qr
```

Usage

```
import { batchGenerateQR, createEmailQR, createEventQR } from '@philjs/qr';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-qr/src/index.ts

Public API

- Direct exports: batchGenerateQR, createEmailQR, createEventQR, createGeoQR, createSMSQR, createVCardQR, createWiFiQR, generateQRCode, generateQRCodeCanvas, generateQRCodeDataURL, validateQRData
- Re-exported names: BarcodeFormat, BatchQROptions, BatchQRResult, DynamicQRConfig, ErrorCorrectionLevel, QRAnalytics, QRCode, QRCodeConfig, QRCodeOptions, QRExportOptions, QRGradient, QRLogo, QROutputFormat, QRScanner, QRScannerConfig, QRStyle, ScanResult, ScannerCallbacks, ScannerConfig, createQRCode, createQRScanner
- Re-exported modules: ./components/QRCode.js, ./components/QRScanner.js, ./types.js

License

MIT

@philjs/quantum - Type: Node package - Purpose: Quantum-Ready Primitives - Post-quantum cryptography, quantum RNG, quantum simulation, and quantum-inspired optimization - Version: 0.1.0 - Location: packages/philjs-quantum - Entry points: packages/philjs-quantum/src/index.ts - Keywords: philjs, quantum, post-quantum, cryptography, kyber, dilithium, quantum-computing, quantum-simulation, quantum-annealing, qaoa, optimization

@philjs/quantum

Quantum-Ready Primitives for PhilJS - Prepare your applications for the quantum computing era with post-quantum cryptography, quantum random number generation, quantum state simulation, and quantum-inspired optimization algorithms.

Installation

```
npm install @philjs/quantum
```

Requirements

- Node.js > = 24
- TypeScript 6.x (for development)
- Browser with `crypto.getRandomValues` support (for QRNG)

Basic Usage

```
import {
  initQuantum,
  useQuantumRandom,
  usePostQuantumCrypto,
  useQuantumSimulator
} from '@philjs/quantum';

// Initialize quantum module
initQuantum({
  useHardwareRandom: true,
  algorithm: 'kyber',
  securityLevel: 256,
  maxQubits: 16
});

// Generate quantum-safe random numbers
const rng = useQuantumRandom();
const randomBytes = rng.nextBytes(32);
const randomInt = rng.nextInt(100);
const randomFloat = rng.nextFloat();

// Use post-quantum cryptography
const { kyber, dilithium } = usePostQuantumCrypto();

// Create a quantum simulator
const simulator = useQuantumSimulator(4); // 4 qubits
```

Quantum Random Number Generator

```

import { QuantumRNG, useQuantumRandom } from '@philjs/quantum';

const rng = new QuantumRNG(256); // 256 byte buffer

// Generate random values
const byte = rng.nextByte();           // 0-255
const bytes = rng.nextBytes(16);        // Uint8Array
const int = rng.nextInt(1000);          // 0-1000
const float = rng.nextFloat();          // 0.0-1.0
const gaussian = rng.nextGaussian(0, 1); // Normal distribution

// Shuffle an array
const shuffled = rng.shuffle([1, 2, 3, 4, 5]);

// Utility hooks
import { useQuantumId, useQuantumShuffle } from '@philjs/quantum';
const id = useQuantumId(16);            // 32-char hex string
const shuffledArray = useQuantumShuffle(['a', 'b', 'c']);

```

Post-Quantum Cryptography

Kyber Key Encapsulation

```

import { KyberKEM } from '@philjs/quantum';

const kyber = new KyberKEM(256); // 128, 192, or 256 bit security

// Generate key pair
const keyPair = kyber.generateKeyPair();
console.log('Public key:', keyPair.publicKey);
console.log('Private key:', keyPair.privateKey);

// Encapsulate (sender)
const { ciphertext, sharedSecret } = kyber.encapsulate(keyPair.publicKey);

// Decapsulate (receiver)
const decapsulatedSecret = kyber.decapsulate(ciphertext, keyPair.privateKey);

```

Dilithium Digital Signatures

```

import { DilithiumSignature } from '@philjs/quantum';

const dilithium = new DilithiumSignature(256);

// Generate key pair
const keyPair = dilithium.generateKeyPair();

// Sign a message
const message = new TextEncoder().encode('Hello, quantum world!');
const signature = dilithium.sign(message, keyPair.privateKey);

// Verify signature
const isValid = dilithium.verify(message, signature, keyPair.publicKey);

```

Quantum State Simulation

```

import { QuantumSimulator, useQuantumSimulator } from '@philjs/quantum';

const sim = new QuantumSimulator(4); // 4 qubits

// Apply quantum gates
sim.hadamard(0);           // Superposition on qubit 0
sim.pauliX(1);             // NOT gate on qubit 1
sim.pauliY(2);             // Y gate
sim.pauliZ(3);             // Z gate
sim.sGate(0);               // S phase gate
sim.tGate(1);               // T gate

// Rotation gates
sim.rotateX(0, Math.PI / 4);
sim.rotateY(1, Math.PI / 2);
sim.rotateZ(2, Math.PI);

// Two-qubit gates
sim.cnot(0, 1);            // CNOT with control=0, target=1
sim.cz(1, 2);               // Controlled-Z
sim.swap(2, 3);              // SWAP qubits

// Measurement
const bit = sim.measure();      // Measure single qubit
const allBits = sim.measureAll(); // Measure all qubits

// Get state information
const state = sim.getState();
const probabilities = sim.getProbabilities();

// Reset to |0...0>
sim.reset();

```

Quantum-Inspired Optimization

Simulated Quantum Annealing

```

import { QuantumAnnealingOptimizer, useQuantumAnnealing } from '@philjs/quantum';

const optimizer = new QuantumAnnealingOptimizer();

// Define a cost function (e.g., MAX-CUT problem)
const costFunction = (solution: number[]) => {
    // Return energy to minimize
    let cost = 0;
    for (let i = 0; i < solution.length - 1; i++) {
        if (solution[i] !== solution[i + 1]) cost--;
    }
    return cost;
};

const result = optimizer.optimize(costFunction, 10, {
    maxIterations: 10000,
    initialTemp: 100,
    finalTemp: 0.001,
    coolingRate: 0.99
});

console.log('Best solution:', result.solution);
console.log('Best energy:', result.energy);
console.log('Iterations:', result.iterations);
console.log('Converged:', result.converged);

```

QAOA (Quantum Approximate Optimization Algorithm)

```

import { QAOAOptimizer, useQAOA } from '@philjs/quantum';

const qaoa = new QAOAOptimizer(4); // 4 qubits

// Define problem as Ising Hamiltonian (adjacency matrix)
const hamiltonian = [
    [0, 1, 0, 1],
    [1, 0, 1, 0],
    [0, 1, 0, 1],
    [1, 0, 1, 0]
];

const result = qaoa.optimize(hamiltonian, {
    layers: 3,
    gamma: [0.5, 0.7, 0.9],
    beta: [0.3, 0.2, 0.1]
});

console.log('Optimal solution:', result.solution);
console.log('Energy:', result.energy);

```

Complex Number Operations

```

import { Complex } from '@philjs/quantum';

const a = Complex.create(3, 4);      // 3 + 4i
const b = Complex.create(1, 2);      // 1 + 2i

const sum = Complex.add(a, b);      // 4 + 6i
const diff = Complex.sub(a, b);     // 2 + 2i
const prod = Complex.mul(a, b);     // -5 + 10i
const quot = Complex.div(a, b);     // 2.2 - 0.4i

const conj = Complex.conjugate(a);  // 3 - 4i
const mag = Complex.magnitude(a);  // 5
const phase = Complex.phase(a);    // atan2(4, 3)
const exp = Complex.exp(Math.PI);   // e^(i*pi) = -1
const scaled = Complex.scale(a, 2); // 6 + 8i

```

API Reference

Initialization

- `initQuantum(config?: QuantumConfig)` - Initialize quantum module with options

Hooks

- `useQuantumRandom(): QuantumRNG` - Get quantum RNG instance
- `usePostQuantumCrypto()` - Get Kyber and Dilithium instances
- `useQuantumSimulator(numQubits): QuantumSimulator` - Create quantum simulator
- `useQuantumAnnealing(): QuantumAnnealingOptimizer` - Get annealing optimizer
- `useQAOA(numQubits): QAOAOptimizer` - Get QAOA optimizer
- `useQuantumId(length?: string)` - Generate random hex ID
- `useQuantumShuffle<T>(array): T[]` - Shuffle array randomly

Classes

QuantumRNG

Cryptographically secure random number generator.

KyberKEM

Post-quantum key encapsulation mechanism.

Methods: - `generateKeyPair(): QuantumKey` - `encapsulate(publicKey): EncapsulatedKey` - `decapsulate(ciphertext, privateKey): Uint8Array`

DilithiumSignature

Post-quantum digital signature algorithm.

Methods: - `generateKeyPair(): QuantumKey` - `sign(message, privateKey): QuantumSignature` - `verify(message, signature, publicKey): boolean`

QuantumSimulator

Quantum state vector simulator.

Single-qubit gates: - `hadamard(qubit)` - `H gate` - `PauliX/Y/Z(qubit)` - `Pauli gates` - `sGate/tGate(qubit)` - `Phase gates` - `rotateX/Y/Z(qubit, theta)` - `Rotation gates`

Two-qubit gates: - `cnot(control, target)` - `Controlled-NOT` - `cz(control, target)` - `Controlled-Z` - `swap(qubit1, qubit2)` - `SWAP`

Measurement: - `measure(qubit): number` - `Measure single qubit` - `measureAll(): number[]` - `Measure all qubits` - `getProbabilities(): number[]` - `Get state probabilities`

QuantumAnnealingOptimizer

Simulated quantum annealing for combinatorial optimization.

QAOAOptimizer

Quantum Approximate Optimization Algorithm simulator.

Configuration

```

interface QuantumConfig {
  useHardwareRandom?: boolean; // Use hardware RNG (default: true)
  algorithm?: 'kyber' | 'dilithium' | 'sphincs' | 'ntru';
  securityLevel?: 128 | 192 | 256;
  enableSimulation?: boolean; // Enable quantum simulation (default: true)
  maxQubits?: number; // Max qubits for simulation (default: 16)
}

```

Types

- `QuantumKey` - Public/private key pair with algorithm info
- `EncapsulatedKey` - Ciphertext and shared secret
- `QuantumSignature` - Digital signature with metadata
- `Qubit` - Single qubit state (α , β amplitudes)
- `Complex` - Complex number { real, imag }
- `QuantumState` - Full quantum state vector
- `QuantumCircuit` - Circuit definition
- `QuantumGate` - Gate with type, targets, and parameters
- `GateType` - All supported gate types

- `OptimizationResult` - Optimizer output
- `QAOAParams` - QAOA circuit parameters

Security Note

The cryptographic implementations in this package are **simulated** for educational and development purposes. For production use, integrate with established post-quantum cryptography libraries such as:

- `liboqs` - Open Quantum Safe
- `NIST PQC` - Standardized algorithms

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: `packages/philjs-quantum/src/index.ts`

Public API

- Direct exports: `Complex`, `DilithiumSignature`, `EncapsulatedKey`, `GateType`, `KyberKEM`, `OptimizationResult`, `QAOAOptimizer`, `QAOAParams`, `QuantumAnnealingOptimizer`, `QuantumCircuit`, `QuantumConfig`, `QuantumGate`, `QuantumKey`, `QuantumRNG`, `QuantumSignature`, `QuantumSimulator`, `QuantumState`, `Qubit`, `initQuantum`, `usePostQuantumCrypto`, `useQAOA`, `useQuantumAnnealing`, `useQuantumId`, `useQuantumRandom`, `useQuantumShuffle`, `useQuantumSimulator`
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/realtime - Type: Node package - Purpose: Real-time collaboration, WebSockets, presence, and multiplayer state for PhilJS - Version: 0.1.0 - Location: packages/philjs-realtime - Entry points: `packages/philjs-realtime/src/index.ts`, `./presence`, `./yjs`, `./cursors`, `./rooms` - Keywords: philjs, realtime, websocket, collaboration, presence, multiplayer, crdt

PhilJS Realtime

Real-time collaboration and WebSocket features for PhilJS applications.

Requirements

- **Node.js 24** or higher
- **TypeScript 6** or higher
- **ESM only** - CommonJS is not supported

Features

- **WebSocket Client** - Auto-reconnecting WebSocket with exponential backoff
- **Presence** - Track who's online with user metadata
- **Cursors** - Real-time cursor position sharing
- **Rooms** - Multi-room support with user management
- **Broadcast** - Channel-based message broadcasting
- **Shared State** - CRDT-like collaborative state synchronization
- **Y.js Integration** - Full CRDT support for complex documents

Installation

```
pnpm add philjs-realtime
```

Quick Start

```
import {
  WebSocketClient,
  usePresence,
  useCursors,
  useRoom,
  useSharedState,
} from 'philjs-realtime';

// Create WebSocket client
const client = new WebSocketClient({
  url: 'wss://your-server.com/ws',
  reconnect: true,
  heartbeatInterval: 30000,
});

// Connect
client.connect();
```

WebSocket Client

Full-featured WebSocket client with automatic reconnection:

```

const client = new WebSocketClient({
  url: 'wss://api.example.com/ws',
  protocols: ['v1'],
  reconnect: true,
  reconnectDelay: 1000,
  maxReconnectDelay: 30000,
  reconnectAttempts: 10,
  heartbeatInterval: 30000,
  onOpen: () => console.log('Connected'),
  onClose: (event) => console.log('Disconnected', event),
  onError: (error) => console.error('Error', error),
  onMessage: (message) => console.log('Message', message),
});

// Connect/disconnect
client.connect();
client.disconnect();

// Send messages
client.send('chat:message', { text: 'Hello!' }, 'room-1');

// Listen for message types
const unsubscribe = client.on('chat:message', (payload) => {
  console.log('New message:', payload);
});

// Reactive status
effect(() => {
  console.log('Connection status:', client.status());
});

```

Presence

Track online users with metadata:

```

import { usePresence } from 'philjs-realtime';

function OnlineUsers() {
  const { others, myPresence, updatePresence, count } = usePresence({
    client,
    room: 'lobby',
    user: {
      id: 'user-123',
      name: 'Alice',
      avatar: '/alice.png',
    },
    initialData: { status: 'online' },
    syncInterval: 1000,
  });

  // Update my presence data
  updatePresence({ status: 'away' });

  return () => {
    const users = others();

    return (
      <div>
        <h3>Online ({count()})</h3>
        <ul>
          {users.map(u => (
            <li key={u.user.id}>
              <img src={u.user.avatar} />
              {u.user.name} - {u.data.status}
            </li>
          ))}
        </ul>
      </div>
    );
  };
}

```

Cursors

Real-time cursor position sharing:

```

import { useCursors } from 'philjs-realtime';

function CollaborativeCanvas() {
  const { cursors, broadcast } = useCursors({
    client,
    room: 'canvas',
    user: { id: 'user-123', name: 'Alice', color: '#ff0000' },
    throttle: 50, // Throttle updates to 50ms
  });

  // Track mouse movement
  const handleMouseMove = (e) => {
    broadcast(e.clientX, e.clientY);
  };

  return () => (
    <div onMouseMove={handleMouseMove} style={{ position: 'relative' }}>
      {/* Render other users' cursors */}
      {cursors().map(cursor => (
        <div key={cursor.user.id} style={{
          position: 'absolute',
          left: cursor.x,
          top: cursor.y,
          pointerEvents: 'none',
        }}
        >
          <svg width="20" height="20">
            <path d="M0,0 L0,14 L4,10 L7,16 L9,15 L6,9 L12,9 Z" fill={cursor.user.color} />
          </svg>
          <span>{cursor.user.name}</span>
        </div>
      ))}
    </div>
  );
}

```

Rooms

Multi-room support with user management:

```

import { useRoom } from 'philjs-realtime';

function ChatRoom({ roomId }) {
  const { room, users, isJoined, error, join, leave, broadcast } = useRoom({
    client,
    roomId,
    user: { id: 'user-123', name: 'Alice' },
    password: 'optional-password',
  });

  // Join on mount
  onMount(() => {
    join();
    return () => leave();
  });

  // Send message to room
  const sendMessage = (text) => {
    broadcast('chat:message', { text, timestamp: Date.now() });
  };

  return () => {
    if (error()) return <div>Error: {error().message}</div>;
    if (!isJoined()) return <div>Joining...</div>;
  };
}

```

Broadcast Channels

Simple pub/sub messaging:

```

import { useBroadcast } from 'philjs-realtime';

function Notifications() {
  const { broadcast, lastMessage, history, clear } = useBroadcast({
    client,
    room: 'global',
    channel: 'notifications',
  });

  // Send a notification
  broadcast({ type: 'info', text: 'Welcome!' });

  return () => (
    <div>
      <h3>Latest: {lastMessage()?.text}</h3>
      <ul>
        {history().map((n, i) => <li key={i}>{n.text}</li>)}
      </ul>
      <button onClick={clear}>Clear</button>
    </div>
  );
}

}

```

Shared State

CRDT-like collaborative state synchronization:

```

import { useSharedState } from 'philjs-realtime';

function CollaborativeDocument() {
  const { state, get, set, merge, version } = useSharedState({
    client,
    room: 'doc-123',
    initialState: {
      title: 'Untitled',
      content: '',
      collaborators: [],
    },
  });

  // Update a single field
  const updateTitle = (title) => set('title', title);

  // Merge multiple fields
  const saveChanges = () => merge({
    content: 'New content',
    lastModified: Date.now(),
  });

  return () => (
    <div>
      <input
        value={get('title')}
        onChange={(e) => updateTitle(e.target.value)}
      />
      <textarea
        value={get('content')}
        onChange={(e) => set('content', e.target.value)}
      />
      <small>Version: {version()}</small>
    </div>
  );
}

```

Y.js Integration

For complex CRDT operations, use Y.js:

```

import { useYjs } from 'philjs-realtime/yjs';
import * as Y from 'yjs';

function CollaborativeEditor() {
  const { doc, awareness, provider } = useYjs({
    client,
    room: 'editor',
    user: { id: 'user-123', name: 'Alice' },
  });

  // Get shared text
  const ytext = doc.getText('content');

  // Bind to editor (e.g., Quill, ProseMirror)
  onMount(() => {
    const binding = new QuillBinding(ytext, quillEditor, awareness);
    return () => binding.destroy();
  });
}

```

Server-Side Room Manager

For server implementations:

```
import { RoomManager } from 'philjs-realtime';

const rooms = new RoomManager();

// Handle WebSocket connections
wss.on('connection', (ws, req) => {
  const userId = getUserId(req);

  ws.on('message', (data) => {
    const { type, roomId, payload } = JSON.parse(data);

    switch (type) {
      case 'join':
        rooms.join(roomId, userId);
        broadcastToRoom(roomId, 'user:joined', { userId });
        break;

      case 'leave':
        rooms.leave(roomId, userId);
        broadcastToRoom(roomId, 'user:left', { userId });
        break;

      case 'message':
        if (rooms.isInRoom(roomId, userId)) {
          broadcastToRoom(roomId, 'message', payload);
        }
        break;
    }
  });

  ws.on('close', () => {
    const leftRooms = rooms.leaveAll(userId);
    leftRooms.forEach(roomId => {
      broadcastToRoom(roomId, 'user:left', { userId });
    });
  });
});

function broadcastToRoom(roomId, event, payload) {
  const users = rooms.getUsers(roomId);
  // Send to all users in room
}
```

Message Protocol

Standard message format:

```
interface RealtimeMessage {
  type: string; // Message type (e.g., 'chat:message')
  payload: any; // Message data
  room?: string; // Target room (optional)
  from?: string; // Sender ID (set by server)
  timestamp?: number; // Message timestamp
}
```

TypeScript Types

```
import type {
  ConnectionStatus,
  User,
  PresenceState,
  RoomConfig,
  RealtimeMessage,
  WebSocketClientOptions,
  CursorPosition,
  CursorState,
} from 'philjs-realtime';
```

Best Practices

1. Throttle Cursor Updates

```
useCursors({ throttle: 50 }); // 50ms between updates
```

2. Clean Up on Unmount

```
onMount(() => {
  client.connect();
  return () => client.disconnect();
});
```

3. Handle Reconnection

```

effect(() => {
  if (client.status() === 'reconnecting') {
    showReconnectingIndicator();
  }
});

```

4. Use Rooms for Isolation

```

// Each document gets its own room
useRoom({ roomId: `doc-${documentId}` });

```

Comparison with Alternatives

Feature	PhilJS Realtime	Socket.IO	Liveblocks	Supabase Realtime
Presence	Yes	Manual	Yes	Yes
Cursors	Yes	Manual	Yes	No
Rooms	Yes	Yes	Yes	Yes
CRDT	Yjs	Manual	Yes	No
Auto-reconnect	Yes	Yes	Yes	Yes
TypeScript	Yes	Yes	Yes	Yes

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./presence`, `./yjs`, `./cursors`, `./rooms`
- Source files: `packages/philjs-realtime/src/index.ts`

Public API

- Direct exports: `ConnectionStatus`, `CursorPosition`, `CursorState`, `PresenceState`, `RealtimeMessage`, `Room`, `RoomConfig`, `RoomManager`, `SharedStateOptions`, `UseBroadcastOptions`, `UseCursorsOptions`, `UsePresenceOptions`, `UseRoomOptions`, `User`, `WebSocketClient`, `WebSocketClientOptions`, `useBroadcast`, `useCursors`, `usePresence`, `useRoom`, `useSharedState`
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/resumable - Type: Node package - Purpose: Resumability and partial hydration system for PhilJS - zero-JS-until-interaction - Version: 0.1.0 - Location: packages/philjs-resumable - Entry points: packages/philjs-resumable/src/index.ts, packages/philjs-resumable/src/serializer.ts, packages/philjs-resumable/src/loader.ts, packages/philjs-resumable/src/hydration.ts - Keywords: philjs, resumability, partial-hydration, zero-javascript, lazy-loading, islands-architecture, progressive-hydration, server-side-rendering

philjs-resumable

Zero-JavaScript Resumability and Partial Hydration for PhilJS - Qwik-style performance without the complexity

Overview

`philjs-resumable` brings **resumability** to PhilJS applications, enabling zero JavaScript execution until user interaction. Inspired by Qwik's revolutionary approach, this package allows you to build instant-loading web applications that deliver HTML to users and only load JavaScript when needed.

What is Resumability?

Traditional frameworks (React, Vue, Svelte) use **hydration**: they render HTML on the server, then download and execute all component JavaScript on the client to "revive" the static HTML. This creates a significant performance bottleneck.

Resumability is different: the application serializes its state to HTML and only loads the necessary code when the user interacts with the page. No JavaScript execution on initial page load = instant interactivity.

Key Features

- Zero JS Until Interaction:** No JavaScript executes until users interact with the page
- QRL (Quick Resource Locator):** Lazy references for functions, handlers, and components
- Partial Hydration:** Only hydrate interactive components, skip static content
- Hydration Strategies:** `idle`, `visible`, `interaction`, `media`, `never`, `custom`
- Streaming SSR:** Progressive rendering with out-of-order hydration support
- Signal Serialization:** Reactive state survives the server-client boundary

Installation

```

npm install philjs-resumable
# or
pnpm add philjs-resumable

```

Quick Start

1. Define Resumable Components

```

import { resumable$, useSignal, $ } from 'philjs-resumable';

// Resumable component - code is Lazy-Loaded on interaction
export const Counter = resumable$(() => {
  const count = useSignal(0);

  return (
    <button onClick$={() => count.value++}>
      Count: {count.value}
    </button>
  );
});

```

2. Use Partial Hydration

```

import { Hydrate } from 'philjs-resumable';

function App() {
  return (
    <main>
      {/* Static content - never hydrated */}
      <Header />

      {/* Hydrate when visible in viewport */}
      <Hydrate when="visible">
        <HeroSection />
      </Hydrate>

      {/* Hydrate when browser is idle */}
      <Hydrate when="idle">
        <Analytics />
      </Hydrate>

      {/* Hydrate on first interaction */}
      <Hydrate when="interaction" event="click">
        <Modal />
      </Hydrate>

      {/* Hydrate based on media query */}
      <Hydrate when="media" query="(min-width: 768px)">
        <DesktopNav />
      </Hydrate>
    </main>
  );
}

```

3. Server-Side Rendering

```

import { renderToResumableString, ResumableContainer } from 'philjs-resumable';

// Server handler
async function handler(req) {
  const html = await renderToResumableString(
    <ResumableContainer>
      <App />
    </ResumableContainer>,
    { basePath: '/chunks' }
  );

  return new Response(html, {
    headers: { 'Content-Type': 'text/html' },
  });
}

```

4. Client-Side Resume

```

import { resume } from 'philjs-resumable';

// Resume the application - sets up event delegation
// No components hydrate until interaction
resume();

```

Core Concepts

QRL (Quick Resource Locator)

QRLs are lazy references that enable code to be serialized and loaded on demand:

```

import { $, onClick$, qrl } from 'philjs-resumable';

// Create a Lazy handler
const handleClick = ${() => {
  console.log('Button clicked!');
});

// Use in JSX with $ suffix
<button onClick$={handleClick}>Click me</button>

// Handler with captured state
const count = useSignal(0);
const increment = ${({captures}) => {
  captures.count.value++;
}, [count], ['count']);

// Reference external module
const handler = qrl('components/handlers', 'handleSubmit');

```

Resumable Signals

Signals that serialize to HTML and restore on hydration:

```

import { useSignal, useComputed } from 'philjs-resumable';

const Counter = resumable$(() => {
  const count = useSignal(0);
  const doubled = useComputed(() => count() * 2);

  // State is serialized to HTML:
  // <span data-qsignal="s0">0</span>
  // When hydrated, the value is restored without re-executing

  return (
    <div>
      <p>Count: {count}</p>
      <p>Doubled: {doubled}</p>
      <button onClick$={() => count.set(c => c + 1)}>
        Increment
      </button>
    </div>
  );
});

```

Hydration Strategies

Control exactly when components hydrate:

```

// Hydrate immediately on page Load
<Hydrate when="load">
  <CriticalComponent />
</Hydrate>

// Hydrate when browser is idle (requestIdleCallback)
<Hydrate when="idle" timeout={2000}>
  <Analytics />
</Hydrate>

// Hydrate when visible (IntersectionObserver)
<Hydrate when="visible" threshold={0.1} rootMargin="200px">
  <LazyImage />
</Hydrate>

// Hydrate on specific events
<Hydrate when="interaction" events={['click', 'focus', 'touchstart']}>
  <InteractiveForm />
</Hydrate>

// Hydrate based on media query
<Hydrate when="media" query="(min-width: 1024px)">
  <DesktopOnlyFeature />
</Hydrate>

// Custom hydration trigger
<Hydrate
  when="custom"
  trigger={(element, hydrate) => {
    // Custom logic
    const observer = new MutationObserver(() => hydrate());
    observer.observe(element, { childList: true });
    return () => observer.disconnect();
  }}
>
  <CustomComponent />
</Hydrate>

// Never hydrate (static only)
<Hydrate when="never">
  <StaticContent />
</Hydrate>

```

Component Modifiers

```

import { static$, client$, server$component } from 'philjs-resumable';

// Static component - never hydrates
const StaticHeader = static$(() => (
  <header>Static content</header>
));

// Client-only component - only runs on client
const BrowserOnlyWidget = client$(
  () => <div>Uses browser APIs</div>,
  <div>Loading...</div> // SSR fallback
);

// Server-only component - only runs during SSR
const ServerData = server$component(() => (
  <script dangerouslySetInnerHTML={{ __html: '...' }} />
));

```

API Reference

QRL Functions

Function	Description
\$(fn)	Create a lazy QRL from a function
component\$(fn)	Create a lazy component QRL
event\$(fn)	Create a typed event handler QRL
onClick\$, onInput\$, etc.	Typed event handler shortcuts
qr1(chunk, symbol)	Reference an external module export
parseQRL(str)	Parse a serialized QRL string
prefetchQRL(qr1)	Prefetch a QRL's chunk

Resumable Functions

Function	Description
resumable\$(fn)	Create a resumable component
useSignal(initial)	Create a resumable signal
useComputed(fn)	Create a computed resumable value
handler\$(fn, captures)	Create an event handler with captures
renderToString(app)	SSR with resumability
resume()	Resume application on client

Hydration Functions

Function	Description
Hydrate	Component for hydration boundaries
useHydration(options)	Hook for programmatic hydration
setupHydration(el, options)	Manually set up hydration
forceHydration(id)	Force immediate hydration
isHydrated(id)	Check hydration status

Container Functions

Function	Description
ResumableContainer	Root container component
resumeContainer(el)	Resume a specific container
resumeAllContainers()	Resume all containers on page
ErrorBoundary	Error boundary for hydration
Suspense	Suspense boundary for lazy loading

Serialization Functions

Function	Description
serializeValue(value)	Serialize any JS value
deserializeValue(serialized)	Deserialize back to JS
generateStateScript(ctx)	Generate state script tag
generateBootstrapScript()	Generate bootstrap script

Serialization Format

State is serialized to JSON and embedded in HTML:

```
<!-- Resumable element with handlers -->
<button
  data-qid="q0"
  data-qevents="click"
>
  Click me
</button>

<!-- Signal binding -->
<span data-qsignal="s0">0</span>

<!-- State script -->
<script id="__PHIL_STATE__" type="application/json">
{
  "signals": {
    "s0": { "id": "s0", "value": { "type": "primitive", "data": 0 } }
  },
  "elements": {
    "q0": {
      "id": "q0",
      "handlers": [ { "event": "click", "qrl": "counter#increment" } ]
    }
  }
}</script>

<!-- Bootstrap script -->
<script>
(function() {
  // Event delegation setup
  // Lazy Loading on interaction
})();
</script>
```

Streaming SSR

For large applications, use streaming SSR:

```

import { createStreamingRenderer } from 'philjs-resumable';

async function streamHandler(req) {
  const renderer = createStreamingRenderer({ basePath: '/chunks' });

  const stream = new ReadableStream({
    async start(controller) {
      // Stream header
      controller.enqueue(encoder.encode('<!DOCTYPE html><html><body>'));

      // Stream content progressively
      controller.enqueue(encoder.encode(renderer.write(<Header />)));
      controller.enqueue(encoder.encode(renderer.flush()));

      controller.enqueue(encoder.encode(renderer.write(<MainContent />)));
      controller.enqueue(encoder.encode(renderer.flush()));

      // End with state scripts
      controller.enqueue(encoder.encode(renderer.end()));
      controller.enqueue(encoder.encode('</body></html>'));
      controller.close();
    },
  });

  return new Response(stream, {
    headers: { 'Content-Type': 'text/html' },
  });
}

```

Build Integration

Vite Plugin (Coming Soon)

```

// vite.config.ts
import { philResumable } from 'philjs-resumable/vite';

export default {
  plugins: [
    philResumable({
      // Extract handlers to separate chunks
      extractHandlers: true,
      // Generate manifest for chunk mapping
      manifest: true,
    }),
  ],
};

```

Manual Chunk Registration

```

import { registerChunks } from 'philjs-resumable';

// Register chunks for Lazy Loading
registerChunks({
  'counter': () => import('./components/Counter'),
  'modal': () => import('./components/Modal'),
  'analytics': () => import('./components/Analytics'),
});

```

Performance Benefits

Metric	Traditional Hydration	Resumable
TTI	Blocks on full hydration	Instant
JS Execution	All at once	On demand
Memory	Full app in memory	Only active components
Interaction Delay	Wait for hydration	Immediate

Comparison with Qwik

PhilJS Resumable is inspired by Qwik but integrates with PhilJS's signal system:

Feature	Qwik	PhilJS Resumable
QRL	Yes	Yes
Resumability	Yes	Yes
Signals	Qwik Signals	PhilJS Signals
Partial Hydration	Yes	Yes (with strategies)
Islands	Via Qwik City	Via philjs-islands
Streaming	Yes	Yes

Best Practices

1. Use **\$ suffix for event handlers**: onClick\$ instead of onClick
2. **Keep handlers small**: Large handlers should be in separate chunks

3. **Capture minimal state:** Only capture what handlers need
4. **Use appropriate hydration strategies:** visible for below-fold, idle for analytics
5. **Prefetch on hover:** Enable prefetching for faster perceived interaction
6. **Test without JS:** Ensure static content works without JavaScript

Debugging

Enable verbose logging:

```
import { configureLoader } from 'philjs-resumable';

configureLoader({
  isDev: true,
});
```

Check hydration stats:

```
import { getHydrationStats, getLoaderStats } from 'philjs-resumable';

console.log('Hydration:', getHydrationStats());
console.log('Loader:', getLoaderStats());
```

TypeScript Support

Full TypeScript support with proper types:

```
import type {
  QRL,
  ResumableSignal,
  HydrationStrategy,
  ResumableComponent
} from 'philjs-resumable';

const handler: QRL<(e: MouseEvent) => void> = $(() => {});
const count: ResumableSignal<number> = useSignal(0);
const strategy: HydrationStrategy = 'visible';
```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./serializer`, `./loader`, `./hydration`
- Source files: `packages/philjs-resumable/src/index.ts`, `packages/philjs-resumable/src/serializer.ts`, `packages/philjs-resumable/src/loader.ts`, `packages/philjs-resumable/src/hydration.ts`

Public API

- Direct exports: `AnyHydrationOptions`, `ComponentLoader`, `CustomOptions`, `FEATURES`, `Hydrate`, `HydrateProps`, `HydrationOptions`, `HydrationStrategy`, `IdleOptions`, `InteractionOptions`, `LazyComponent`, `LoaderConfig`, `MediaOptions`, `SerializationContext`, `SerializedElement`, `SerializedHandler`, `SerializedSignal`, `SerializedValue`, `VERSION`, `VisibleOptions`, `addSignalSubscriber`, `addStreamingChunk`, `cancelHydration`, `clearHydrationState`, `clearLoaderCache`, `configureLoader`, `createHydrateComponent`, `createSerializationContext`, `createStreamingContext`, `deserializeFromAttribute`, `deserializeValue`, `discoverHydrationBoundaries`, `forceHydration`, `generateBootstrapScript`, `generateElementAttributes`, `generateId`, `generateInlineState`, `generateStateScript`, `getHydrationStats`, `getLazyComponent`, `getLoaderConfig`, `getLoaderStats`, `getSerializationContext`, `hasLazyComponent`, `initHydration`, `initLoader`, `isHydrated`, `loadAndHydrate`, `loadAndInvokeHandler`, `loadComponent`, `loadFromQRL`, `prefetchChunk`, `prefetchComponent`, `prefetchVisibleComponents`, `queueHydration`, `queueLoad`, `registerComponent`, `registerElement`, `registerLazyComponent`, `registerLazyComponents`, `registerSignal`, `serializeToAttribute`, `serializeValue`, `setupHydration`, `useHydration`, `waitForHydration`, `waitForLoads`, `withSerializationContext`
- Re-exported names: `// Client Resume resume`, `// Common Event Handlers onClick`, `//CompactSerializationserializeToAttribute`, `//ComponentFactoryresumable`, `// Component Modifiers`
`static`, `//ComponentRegistrationregisterLazyComponent`, `//ComponentsHydrate`, `//ComponentsResumableContainer`, `//ConfigurationconfigureLoader`, `//Configurationconfig`, `// HTML Generation generateStateScript`, `// Initialization initLoader`, `// Loading loadComponent`, `// Prefetching prefetchContainer`, `// QRL Factory Functions`, `//SSRrenderToResumableString`, `//Signal/StateQRLsignal`, `// Signals useSignal`, `// Statistics getContainerStats`, `// Streaming createStreamingContext`, `// Task QRLs`, `server`, `//TypestypeComponentLoader`, `//TypestypeContainerState`, `//TypestypeHydrationStrategy`, `//TypestypeResumableComponent`, `//TypestypeSerializedSignal`, `//Utilities`, `cancelHydration`, `clearHydrationState`, `clearLoaderCache`, `clearQRLRegistry`, `client`, `component`, `computed`, `createQRL`, `createStreamingRenderer`, `deserializeFromAttribute`, `deserializeValue`, `disposeAllContainers`, `disposeContainer`, `event`, `forceHydration`, `generateBootstrapScript`, `generateElementAttributes`, `generateId`, `generateInlineState`, `getAllContainers`, `getCurrentComponentId`, `getLazyComponent`, `getLoaderConfig`, `getQRLAttribute`, `getSerializationContext`, `hasLazyComponent`, `initHydration`, `inlineQRL`, `isContainerHydrated`, `isHydrated`, `isQRL`, `isResumable`, `isServer`, `loadAndHydrate`, `loadAndInvokeHandler`, `loadFromQRL`, `onBlur`, `onChange`, `onFocus`, `onInput`, `onKeyDown`, `onKeyUp`, `onSubmit`, `parseQRL`, `prefetchChunk`, `prefetchComponent`, `prefetchQRL`, `prefetchVisibleComponents`, `qrl`, `queueHydration`, `queueLoad`, `registerChunk`, `registerChu`, `setupContainerPrefetching`, `useComputed`, `useHydration`, `useTask`, `useVisibleTask`, `waitForContainer`, `waitForHydration`, `waitForLoads`, `withResumableContext`, `withSerializationContext`
- Re-exported modules: `./container.js`, `./hydration.js`, `./loader.js`, `./qrl.js`, `./resumable.js`, `./serializer.js`

License

MIT

@philjs/rich-text - Type: Node package - Purpose: Rich text block editor for PhilJS - Notion-style blocks, slash commands, collaborative editing - Version: 0.1.0 - Location: `packages/philjs-rich-text` - Entry points: `packages/philjs-rich-text/src/index.ts` - Keywords: philjs, rich-text, editor, blocks, notion, prosemirror

@philjs/rich-text

Rich text block editor for PhilJS - Notion-style blocks, slash commands, collaborative editing

Overview

Rich text block editor for PhilJS - Notion-style blocks, slash commands, collaborative editing

Focus Areas

- philjs, rich-text, editor, blocks, notion, prosemirror

Entry Points

- packages/philjs-rich-text/src/index.ts

Quick Start

```
import { createBlock, createTextNode, parseHTML } from '@philjs/rich-text';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- createBlock
- createTextNode
- parseHTML
- serializeToHTML
- serializeToMarkdown

Install

```
pnpm add @philjs/rich-text
```

Usage

```
import { createBlock, createTextNode, parseHTML } from '@philjs/rich-text';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-rich-text/src/index.ts

Public API

- Direct exports: createBlock, createTextNode, parseHTML, serializeToHTML, serializeToMarkdown
- Re-exported names: Block, BlockRenderer, BlockType, CollaborationConfig, CollaborationUser, Cursor, Editor, EditorCommands, EditorConfig, EditorInstance, EditorOptions, EditorState, EditorView, ExportOptions, Extension, FloatingToolbar, FloatingToolbarOptions, ImportOptions, KeyBinding, NodeView, NodeViewFactory, Position, Selection, SlashCommand, SlashCommandMenu, SlashCommandOptionsMenu, TextMark, TextNode, ToolbarConfig, ToolbarItem, createEditor, createRichTextEditor, defaultSlashCommands
- Re-exported modules: ./components/index.js, ./core/editor.js, ./extensions/defaultCommands.js, ./types.js

License

MIT

@philjs/rocket - Type: Node package - Purpose: Rocket web framework integration for PhilJS - SSR, fairings, guards, and WebSocket support - Version: 0.1.0 - Location: packages/philjs-rocket - Entry points: packages/philjs-rocket/src/index.ts, ./app, ./config, packages/philjs-rocket/src/middleware.ts, packages/philjs-rocket/src/fairing.ts, packages/philjs-rocket/src/guards.ts, packages/philjs-rocket/src/handlers.ts, packages/philjs-rocket/src/responders.ts, packages/philjs-rocket/src/ssr.ts, packages/philjs-rocket/src/state.ts, packages/philjs-rocket/src/templates.ts, packages/philjs-rocket/src/websocket.ts, packages/philjs-rocket/src/server.ts, packages/philjs-rocket/src/forms.ts, packages/philjs-rocket/src/cookies.ts - Keywords: philjs, rocket, rust, ssr, websocket, fairing, guard

@philjs/rocket

Rocket web framework integration for PhilJS applications. Seamlessly connect Rust Rocket backends with PhilJS frontend components for full-stack type safety.

Installation

```
npm install @philjs/rocket
# or
yarn add @philjs/rocket
# or
pnpm add @philjs/rocket
```

Rust (Cargo.toml):

```
[dependencies]
philjs-rocket = "0.1"
rocket = "0.5"
```

Basic Usage

TypeScript:

```

import { createRocketClient, useRocketQuery } from '@philjs/rocket';

const client = createRocketClient({
  baseUrl: 'http://localhost:8000',
});

function Products() {
  const { data, error } = useRocketQuery('/api/products');

  if (error) return <Error message={error} />;
  return <ProductGrid products={data} />;
}

```

Rust:

```

use philjs_rocket::{PhilJS, TypeScript};
use rocket::{get, routes, serde::json::Json};

#[derive(Serialize, TypeScript)]
struct Product {
    id: u64,
    name: String,
    price: f64,
}

#[get("/api/products")]
async fn get_products() -> Json<Vec<Product>> {
    Json(fetch_products().await)
}

#[launch]
fn rocket() -> _ {
    PhilJS::generate_types("./frontend/src/types");

    rocket::build()
        .mount("/", routes![get_products])
}

```

Features

- **Type Bridge** - Generate TypeScript from Rocket route types
- **API Client** - Typed client matching Rocket endpoints
- **Request Guards** - Type-safe request validation
- **Fairings** - CORS, auth, and logging fairings
- **Forms** - Multipart form and file handling
- **State Management** - Shared state between routes
- **WebSocket** - Real-time WebSocket support
- **SSR** - Server-side rendering integration
- **Error Catchers** - Unified error handling
- **Testing** - Integration testing utilities

Hooks

Hook	Description
useRocketQuery	Fetch from Rocket endpoint
useRocketMutation	Mutations with type safety
useRocketForm	Form submission handling
useRocketAuth	Authentication integration

Request Guards

```

use philjs_rocket::guards::PhilJSAuth;

#[get("/api/protected")]
async fn protected(auth: PhilJSAuth) -> Json<UserData> {
    Json(auth.user_data())
}

```

CLI

```

# Generate TypeScript types
npx philjs-rocket sync

# Watch and regenerate on changes
npx philjs-rocket watch

```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./app`, `./config`, `./middleware`, `./fairing`, `./guards`, `./handlers`, `./responders`, `./ssr`, `./state`, `./templates`, `./websocket`, `./server`, `./forms`, `./cookies`
- Source files: `packages/philjs-router/src/index.ts`, `packages/philjs-router/src/middleware.ts`, `packages/philjs-router/src/fairing.ts`, `packages/philjs-router/src/guards.ts`, `packages/philjs-router/src/handlers.ts`, `packages/philjs-router/src/responders.ts`, `packages/philjs-router/src/ssr.ts`, `packages/philjs-router/src/state.ts`, `packages/philjs-router/src/templates.ts`, `packages/philjs-router/src/websocket.ts`, `packages/philjs-router/src/server.ts`, `packages/philjs-router/src/forms.ts`, `packages/philjs-router/src/cookies.ts`

Public API

- Direct exports: `AppState`, `AsyncHandler`, `AuthGuard`, `AuthGuardConfig`, `AuthUser`, `BroadcastChannel`, `BroadcastManager`, `CORSMiddleware`, `CORSMiddlewareConfig`, `CSRFField`, `CSRGuard`, `CSRGuardConfig`, `CatcherDefinition`, `CatcherHandler`, `CatcherRequest`, `ComponentRegistry`, `ComponentTemplate`, `CompressionMiddleware`, `CompressionMiddlewareConfig`, `Cookie`, `CookieJarImpl`, `CookieOptions`, `CustomFairing`, `DEFAULT_SERVER_CONFIG`, `DEFAULT_WS_OPTIONS`, `ErrorResponder`, `FairingComposer`, `FieldType`, `FlashMessage`, `FlashMessages`, `FormBuilder`, `FormDataGuard`, `FormDataWithFiles`, `FormErrors`, `FormField`, `FormSchema`, `FormValidationResult`, `FormValidator`, `GlobalState`, `Handler`, `HandlerContext`, `HeadContent`, `HtmlResponder`, `HttpMethod`, `JsonBodyGuard`, `JsonBodyGuardConfig`, `JsonResponder`, `LayoutBuilder`, `LayoutConfig`, `LinkTag`, `LiveViewClientMessage`, `LiveViewFairing`, `LiveViewFairingConfig`, `LiveViewServerMessage`, `LiveViewState`, `ManagedState`, `MetaTag`, `MetricsFairing`, `MetricsFairingConfig`, `Middleware`, `Pagination`, `PathGuard`, `PresenceEntry`, `PresenceState`, `PresenceTracker`, `PrivateCookie`, `QueryGuard`, `QueryGuardConfig`, `RateLimitMiddleware`, `RateLimitMiddlewareConfig`, `RedirectResponder`, `RenderFunction`, `RequestData`, `RequestState`, `ResponseBuilder`, `RocketAppBuilder`, `RocketServer`, `RocketServerBuild`, `RocketServerConfig`, `RouteContext`, `RouteDefinition`, `RouteHandler`, `RouteResponse`, `SSRContext`, `SSRContextData`, `SSRContextGuard`, `SSRFairing`, `SSRFairingConfig`, `SSRMiddleware`, `SSRMiddlewareConfig`, `SSRRenderer`, `SSRRendererConfig`, `SSRResult`, `ScriptTag`, `SecurityMiddleware`, `SecurityMiddlewareConfig`, `Selector`, `SessionData`, `StateConfig`, `StateFairing`, `StateFairingConfig`, `StateManager`, `StreamResponder`, `StyleTag`, `TemplateEngine`, `TemplateEngineConfig`, `TemplateEngineType`, `TracingMiddleware`, `TracingMiddlewareConfig`, `UploadedFile`, `WebSocketConfig`, `badRequest`, `combineGuards`, `composeFairings`, `configureWebSocket`, `cookie`, `createAsyncHandler`, `createBroadcastManager`, `createCORSMiddleware`, `createComponentRegistry`, `createCompressionMiddleware`, `createCookieJar`, `createDerivedSelector`, `createDevServer`, `createFairing`, `createForm`, `createFormValidator`, `createGlobalState`, `createGuard`, `createHandler`, `createLayoutBuilder`, `createLiveViewSocket`, `createManagedState`, `createMessageDecoder`, `createMessageEncoder`, `createPresenceTracker`, `createProdServer`, `createRateLimitMiddleware`, `createRequestState`, `createRocketApp`, `createRocketServer`, `createSSRContext`, `createSSRMiddleware`, `createSSRRenderer`, `createSecurityMiddleware`, `createSelector`, `createStateManager`, `createStreamingRenderer`, `createTemplateEngine`, `createTracingMiddleware`, `created`, `csrfField`, `del`, `error`, `flashError`, `flashInfo`, `flashSuccess`, `flashWarning`, `forbidden`, `generateCsrfToken`, `generateJsonLd`, `generateRustCookieCode`, `generateRustFormStruct`, `generateRustHandler`, `generateRustHandlerWithBody`, `generateRustSSRCode`, `generateRustState`, `generateRustStateGuard`, `generateSEOMeta`, `get`, `getFlash`, `getHeader`, `getParam`, `getQuery`, `html`, `json`, `noContent`, `notFound`, `parseCookies`, `parseForm`, `parseFormData`, `parseJson`, `parseMultipart`, `parseMultipartFormData`, `patch`, `persistentCookie`, `post`, `put`, `redirect`, `removalCookie`, `requireParam`, `serializeCookie`, `serverError`, `sessionCookie`, `setFlash`, `signCookie`, `stream`, `text`, `unauthorized`, `validateCsrf`, `verifyCookie`, `withMiddleware`
- Re-exported names: `AppState`, `AsyncHandler`, `AuthGuard`, `AuthGuardConfig`, `AuthUser`, `BroadcastChannel`, `BroadcastManager`, `CORSMiddleware`, `CORSMiddlewareConfig`, `CSRFField`, `CSRGuard`, `CSRGuardConfig`, `CatcherDefinition`, `CatcherHandler`, `CatcherRequest`, `ComponentRegistry`, `ComponentTemplate`, `CompressionMiddleware`, `CompressionMiddlewareConfig`, `Cookie`, `CookieJarImpl`, `CustomFairing`, `DEFAULT_SERVER_CONFIG`, `DEFAULT_WS_OPTIONS`, `ErrorResponder`, `FairingComposer`, `FairingContext`, `FairingHooks`, `FairingResponse`, `FieldType`, `FlashMessage`, `FlashMessages`, `FormBuilder`, `FormDataGuard`, `FormDataWithFiles`, `FormErrors`, `FormField`, `FormSchema`, `FormValidationResult`, `FormValidator`, `GlobalState`, `GuardContext`, `GuardDefinition`, `GuardOutcome`, `Handler`, `HandlerContext`, `HandlerCookieJar`, `HandlerCookieOptions`, `HeadContent`, `HtmlResponder`, `HtmlResponderOptions`, `HttpMethod`, `JsonBodyGuard`, `JsonBodyGuardConfig`, `JsonResponder`, `JsonResponderOptions`, `LayoutBuilder`, `LayoutConfig`, `LinkTag`, `LiveViewClientMessage`, `LiveViewFairing`, `LiveViewFairingConfig`, `LiveViewHandler`, `LiveViewMessage`, `LiveViewPatch`, `LiveViewServerMessage`, `LiveViewState`, `ManagedState`, `MetaTag`, `MetricsFairing`, `MetricsFairingConfig`, `Middleware`, `Pagination`, `PathGuard`, `PhiljsFairing`, `PresenceEntry`, `PresenceState`, `PresenceTracker`, `PrivateCookie`, `QueryGuard`, `QueryGuardConfig`, `RateLimitMiddleware`, `RateLimitMiddlewareConfig`, `RedirectResponder`, `RenderFunction`, `RequestData`, `RequestGuard`, `RequestState`, `ResponderOptions`, `ResponseBuilder`, `RocketCorsConfig`, `RocketConfig`, `RocketCookieJar`, `RocketCookieOptions`, `RocketFlashMessage`, `RocketLiveViewConfig`, `RocketSSRConfig`, `RocketServer`, `RocketServerBuild`, `RocketServerConfig`, `RocketSessionConfig`, `RocketStaticConfig`, `RouteContext`, `RouteDefinition`, `RouteHandler`, `RouteResponse`, `SSRContext`, `SSRContextData`, `SSRContextGuard`, `SSRFairing`, `SSRMetaTag`, `SSRMiddleware`, `SSRMiddlewareConfig`, `SSRRenderer`, `SSRRendererConfig`, `SSRResult`, `Script`, `ScriptTag`, `SecurityMiddleware`, `SecurityMiddlewareConfig`, `Selector`, `SessionData`, `StateConfig`, `StateFairing`, `StateFairingConfig`, `StateManager`, `StreamResponder`, `StreamResponderOptions`, `StyleTag`, `TemplateContext`, `TemplateEngine`, `TemplateEngineConfig`, `TemplateEngineOptions`, `TemplateEngineType`, `TemplateHelper`, `TracingMiddleware`, `TracingMiddlewareConfig`, `UploadedFile`, `WebSocketConfig`, `WebSocketConnection`, `WebSocketMessage`, `WebSocketOptions`, `badRequest`, `combineGuards`, `composeFairings`, `configureWebSocket`, `cookie`, `createAsyncHandler`, `createBroadcastManager`, `createCORSMiddleware`, `createComponentRegistry`, `createCompressionMiddleware`, `createCookieJar`, `createDerivedSelector`, `createDevServer`, `createFairing`, `createForm`, `createFormValidator`, `createGlobalState`, `createGuard`, `createHandler`, `createLayoutBuilder`, `createLiveViewSocket`, `createManagedState`, `createMessageDecoder`, `createMessageEncoder`, `createPresenceTracker`, `createProdServer`, `createRateLimitMiddleware`, `createRequestState`, `createRocketServer`, `createSSRContext`, `createSSRMiddleware`, `createSSRRenderer`, `createSecurityMiddleware`, `createSelector`, `createStateManager`, `createStreamingRenderer`, `createTemplateEngine`, `createTracingMiddleware`, `created`, `csrfField`, `del`, `error`, `flashError`, `flashInfo`, `flashSuccess`, `flashWarning`, `forbidden`, `generateCsrfToken`, `generateJsonLd`, `generateRustCookieCode`, `generateRustFormStruct`, `generateRustHandler`, `generateRustHandlerWithBody`, `generateRustSSRCode`, `generateRustState`, `generateRustStateGuard`, `generateSEOMeta`, `get`, `getFlash`, `getHeader`, `getParam`, `getQuery`, `html`, `json`, `noContent`, `notFound`, `parseCookies`, `parseForm`, `parseFormData`, `parseJson`, `parseMultipart`, `parseMultipartFormData`, `patch`, `persistentCookie`, `post`, `put`, `redirect`, `redirectionResponder`, `removalCookie`, `requireParam`, `serializeCookie`, `serverError`, `sessionCookie`, `setFlash`, `signCookie`, `stream`, `text`, `unauthorized`, `validateCsrf`, `verifyCookie`, `withMiddleware`
- Re-exported modules: `./cookies.js`, `./fairing.js`, `./forms.js`, `./guards.js`, `./handlers.js`, `./middleware.js`, `./responders.js`, `./server.js`, `./ssr.js`, `./state.js`, `./templates.js`, `./types.js`, `./websocket.js`

License

MIT

@philjs/router - Type: Node package - Purpose: File-based routing with nested layouts for PhilJS - Version: 0.1.0 - Location: packages/philjs-router - Entry points: packages/philjs-router/src/index.ts - Keywords: philjs, routing, navigation - Book coverage: [router/advanced-features.md](#)

philjs-router

File-based routing with Remix-style nested routes, data loading, error boundaries, Qwik-style speculative prefetching, and view transitions for PhilJS.

Requirements

- **Node.js 24** or higher
- **TypeScript 6** or higher
- **ESM only** - CommonJS is not supported

Installation

```
pnpm add philjs-router
```

Usage

High-Level Declarative Router

The simplest way to use PhilJS Router is with the declarative API:

```

import { createAppRouter, Link, RouterView } from 'philjs-router';

// Define your routes
const router = createAppRouter({
  routes: [
    {
      path: '/',
      component: HomePage,
    },
    {
      path: '/products/:id',
      component: ProductPage,
      loader: async ({ params }) => {
        const res = await fetch(`/api/products/${params.id}`);
        return res.json();
      },
    },
    {
      path: '/blog',
      layout: BlogLayout,
      children: [
        { path: '/blog', component: BlogIndex },
        { path: '/blog/:slug', component: BlogPost },
      ],
    },
  ],
  base: '',
  transitions: true,
  prefetch: { strategy: 'intent' },
  target: '#app',
});

// Use in your app
function App() {
  return (
    <div>
      <nav>
        <Link to="/">Home</Link>
        <Link to="/products/123">Product 123</Link>
        <Link to="/blog">Blog</Link>
      </nav>
      <RouterView />
    </div>
  );
}

```

Using Router Hooks

```

import { useRouter, useRoute } from 'philjs-router';

function MyComponent() {
  const { route, navigate } = useRouter();
  const currentRoute = useRoute();

  const handleClick = () => {
    navigate('/products/456', { replace: false });
  };

  return (
    <div>
      <p>Current path: {currentRoute?.path}</p>
      <p>Params: {JSON.stringify(currentRoute?.params)}</p>
      <button onClick={handleClick}>Go to Product 456</button>
    </div>
  );
}

```

Remix-Style Nested Routes with Data Loading

PhilJS Router supports Remix-style nested routes with parallel data loading:

```
// routes/users.tsx - Parent Layout
export async function loader() {
  return { users: await fetchUsers() };
}

export default function UsersLayout({ children }) {
  const { users } = useLoaderData();
  return (
    <div>
      <Sidebar users={users} />
      <main>{children}</main>
    </div>
  );
}

// routes/users/[id].tsx - Child route
export async function loader({ params }) {
  return { user: await fetchUser(params.id) };
}

export default function UserDetail() {
  const { user } = useLoaderData();
  return <UserProfile user={user} />;
}
```

Key Features: - Parent and child loaders run in **parallel** (no waterfall) - Each route segment can have its own loader, action, and error boundary - Data is available via `useLoaderData()` hook

Deferred Data Loading (Streaming)

For slow data that shouldn't block initial render:

```
import { defer, Await, useLoaderData } from 'philjs-router';

export async function loader() {
  return {
    user: await fetchUser(), // Blocks - critical data
    posts: defer(fetchPosts()), // Streams in later
    comments: defer(fetchComments()), // Also streams
  };
}

function UserPage() {
  const { user, posts, comments } = useLoaderData();

  return (
    <div>
      {/* Renders immediately */}
      <UserHeader user={user} />

      {/* Streams in when ready */}
      <Suspense fallback={<Spinner />}>
        <Await resolve={posts}>
          {(posts) => <PostList posts={posts} />}
        </Await>
      </Suspense>

      <Suspense fallback={<CommentSkeleton />}>
        <Await resolve={comments}>
          {(comments) => <CommentList comments={comments} />}
        </Await>
      </Suspense>
    </div>
  );
}
```

Route Actions (Form Handling)

Handle form submissions with route actions:

```

import { Form, useActionData, useNavigation, redirect } from 'philjs-router';

// In your route file
export async function action({ request }) {
  const formData = await request.formData();
  const email = formData.get('email');
  const password = formData.get('password');

  const user = await createUser({ email, password });

  if (!user) {
    return { error: 'Failed to create user' };
  }

  return redirect('/dashboard');
}

export default function SignUp() {
  const actionData = useActionData();
  const navigation = useNavigation();
  const isSubmitting = navigation.state === 'submitting';

  return (
    <Form method="post">
      <input name="email" type="email" required />
      <input name="password" type="password" required />
      <button type="submit" disabled={isSubmitting}>
        {isSubmitting ? 'Creating...' : 'Sign Up'}
      </button>
      {actionData?.error && <p className="error">{actionData.error}</p>}
    </Form>
  );
}

```

Route Error Boundaries

Handle errors at any level of the route hierarchy:

```

import { useRouteError, isRouteErrorResponse, throwNotFound } from 'philjs-router';

// In your route file
export async function loader({ params }) {
  const user = await fetchUser(params.id);

  if (!user) {
    throwNotFound('User not found');
  }

  return { user };
}

export function ErrorBoundary() {
  const error = useRouteError();

  if (isRouteErrorResponse(error)) {
    return (
      <div className="error-page">
        <h1>{error.statusText} {error.statusText}</h1>
        <p>{error.data}</p>
      </div>
    );
  }

  return (
    <div className="error-page">
      <h1>Something went wrong</h1>
      <p>{error.message}</p>
    </div>
  );
}

export default function UserDetail() {
  const { user } = useLoaderData();
  return <UserProfile user={user} />;
}

```

Accessing Parent Route Data

Access loader data from any route in the hierarchy:

```

import { useRouteLoaderData, useMatches } from 'philjs-router';

function ChildComponent() {
  // Access specific parent's data by route ID
  const parentData = useRouteLoaderData('routes/users');

  // Or access all matched routes
  const matches = useMatches();

  return (
    <div>
      <p>Parent user count: {parentData?.users.length}</p>
      <ul>
        {matches.map((match) => (
          <li key={match.id}>{match.pathname}</li>
        ))}
      </ul>
    </div>
  );
}

```

Fetchers for Non-Navigation Mutations

Use fetchers for mutations that don't navigate:

```

import { useFetcher } from 'philjs-router';

function LikeButton({ postId }) {
  const fetcher = useFetcher();
  const isliking = fetcher.state === 'submitting';

  return (
    <fetcher.Form method="post" action="/api/like">
      <input type="hidden" name="postId" value={postId} />
      <button type="submit" disabled={isliking}>
        {fetcher.data?.liked ? 'Unlike' : 'Like'}
      </button>
    </fetcher.Form>
  );
}

```

File-Based Route Discovery

For build-time route generation:

```

import { discoverRoutes, matchRoute } from 'philjs-router';

// Discover routes from a directory structure
const routes = discoverRoutes('/path/to/routes');

// Match a URL to a route
const match = matchRoute('/products/123', routes);
if (match) {
  console.log('Matched route:', match.route.pattern);
  console.log('Params:', match.params);
}

```

Qwik-Style Speculative Prefetching

PhilJS Router includes Qwik-style speculative prefetching for lightning-fast navigation:

```

import { EnhancedLink, initPrefetchManager, prefetchRoute, prefetchRouteWithData } from 'philjs-router';

// Initialize the prefetch manager (optional, auto-initializes on first use)
initPrefetchManager({
  maxConcurrent: 3,
  respectSaveData: true,
  minConnectionType: '3g',
});

```

Link Component Prefetch Modes

```

// Prefetch on hover (after 100ms delay) - default for internal links
<EnhancedLink href="/dashboard" prefetch="hover">Dashboard</EnhancedLink>

// Prefetch when visible (Intersection Observer)
<EnhancedLink href="/about" prefetch="visible">About</EnhancedLink>

// Prefetch on intent (hover + focus)
<EnhancedLink href="/users" prefetch="intent">Users</EnhancedLink>

// Prefetch immediately on render (critical paths)
<EnhancedLink href="/critical" prefetch="render">Critical</EnhancedLink>

// No prefetch (default for external links, or for heavy pages)
<EnhancedLink href="/heavy" prefetch="none">Heavy Page</EnhancedLink>

```

Advanced Prefetch Options

```

// Prefetch with custom delay
<EnhancedLink
  href="/dashboard"
  prefetch={{ mode: 'hover', delay: 200 }}
>
  Dashboard
</EnhancedLink>

// Prefetch route code AND run data loader
<EnhancedLink
  href="/users/123"
  prefetch={{ mode: 'hover', withData: true }}
>
  User Profile
</EnhancedLink>

// Critical path with immediate data prefetch
<EnhancedLink
  href="/checkout"
  prefetch={{ mode: 'render', withData: true, preload: true }}
>
  Checkout
</EnhancedLink>

```

Programmatic Prefetching

```

import { prefetchRoute, prefetchRouteWithData, getPrefetchManager } from 'philjs-router';

// Prefetch route code only
await prefetchRoute('/dashboard');

// Prefetch route + run data Loader
await prefetchRouteWithData('/users', { preload: true });

// Access the prefetch manager directly
const manager = getPrefetchManager();
manager.registerRouteModule('/products/:id', {
  loader: async ({ params }) => fetch(`/api/products/${params.id}`).then(r => r.json()),
  default: ProductPage,
});

// Check prefetch status
console.log(manager.isPrefetched('/dashboard')); // true/false
console.log(manager.getStats()); // { queued, Loading, Loaded, failed, cacheHits, cacheMisses }

```

usePrefetchLink Hook

```

import { usePrefetchLink } from 'philjs-router';

function CustomNavLink({ href, children }) {
  const { prefetch, isPrefetched, isLoading, handlers } = usePrefetchLink(href, {
    mode: 'hover',
    withData: true,
  });

  return (
    <a
      href={href}
      {...handlers}
      className={isPrefetched ? 'prefetched' : ''}
      data-loading={isLoading}
    >
      {children}
    </a>
  );
}

```

Service Worker Integration

Enable background caching for prefetched routes:

```

import {
  registerPrefetchServiceWorker,
  generatePrefetchServiceWorker,
  createInlineServiceWorker,
  swrFetch,
} from 'philjs-router';

// Option 1: Register a service worker file
await registerPrefetchServiceWorker({
  swPath: '/sw.js',
  onReady: (registration) => console.log('SW ready:', registration.scope),
  onUpdate: (registration) => console.log('SW updated'),
});

// Option 2: Generate SW code for your build system
const swCode = generatePrefetchServiceWorker({
  routeCacheName: 'my-app-routes-v1',
  dataCacheName: 'my-app-data-v1',
  maxAge: 24 * 60 * 60 * 1000, // 24 hours
  maxEntries: 50,
  staleWhileRevalidate: true,
});
// Write swCode to public/sw.js during build

// Option 3: Inline SW for development
const swUrl = createInlineServiceWorker();
await registerPrefetchServiceWorker({ swPath: swUrl });

// Stale-while-revalidate fetch for data
const response = await swrFetch('/api/data', {
  maxAge: 5 * 60 * 1000, // 5 minutes
  onRevalidate: (freshResponse) => {
    console.log('Data revalidated in background');
  },
});

```

Smart Preloading (Legacy)

PhilJS Router also includes the original smart preloading based on user intent:

```

import { initSmartPreloader, usePreload } from 'philjs-router';

// Initialize smart preloader
const preloader = initSmartPreloader({
  strategy: 'intent', // 'hover' | 'visible' | 'intent' | 'eager'
  intentThreshold: 0.6,
  priority: 'auto',
});

// Manually preload a route
function NavLink({ to, children }) {
  const handleMouseEnter = usePreload(to, { strategy: 'hover' });

  return (
    <Link to={to} onMouseEnter={handleMouseEnter}>
      {children}
    </Link>
  );
}

```

View Transitions

Add smooth page transitions with the View Transitions API:

```

import { initViewTransitions, navigateWithTransition } from 'philjs-router';

// Initialize view transitions
const transitionManager = initViewTransitions();

// Navigate with custom transition
await navigateWithTransition('/about', {
  type: 'slide-left',
  duration: 300,
  easing: 'ease-in-out',
});

// Mark shared elements for cross-fade effects
function ProductImage({ src, id }) {
  return (
    <img
      src={src}
      style={{ viewTransitionName: `product-${id}` }}
    />
  );
}

```

Parallel Routes (Next.js 14 Style)

Render multiple pages in the same layout simultaneously with independent loading states:

```

import {
  createParallelRouteConfig,
  matchParallelRoutes,
  loadParallelSlots,
  renderParallelSlots,
  useSlot,
} from 'philjs-router';

// Configure parallel routes
const config = createParallelRouteConfig({
  basePath: '/dashboard',
  slots: [
    // Sidebar slot - always present
    {
      name: '@sidebar',
      path: '/',
      loader: async () => ({ nav: await fetchNav() }),
      component: Sidebar,
    },
    // Main content slot
    {
      name: '@main',
      path: '/users',
      loader: async () => ({ users: await fetchUsers() }),
      component: UsersList,
    },
    // Optional modal slot with route interception
    {
      name: '@modal',
      path: '(.)photos/:id', // Intercepts /dashboard/photos/:id
      loader: async ({ params }) => ({ photo: await fetchPhoto(params.id) }),
      component: PhotoModal,
      optional: true,
    },
  ],
});

// Layout combines all slots
function DashboardLayout({ sidebar, main, modal }) {
  return (
    <div>
      {sidebar}
      <main>{main}</main>
      {modal} /* Modal overlays when route is intercepted */
    </div>
  );
}

// Use in slot components
function PhotoModal() {
  const { data, slotName } = useSlot();
  const { close } = useInterceptedNavigation();

  return (
    <div className="modal">
      <img src={data.photo.url} />
      <button onClick={close}>Close</button>
    </div>
  );
}

```

Key Features: - **@slot syntax** - Named slots like @modal, @sidebar, @main - **Route interception** - Soft navigation for modals, hard navigation for deep links - **Parallel data loading** - All slots load simultaneously (no waterfalls!) - **Independent states** - Each slot has its own loading/error state - **Conditional rendering** - Slots only render when their route matches

Route Interception Patterns:

```

// (...) - Same level
path: '(.)photos/:id' // Intercepts /dashboard/photos/123

// (...) - One Level up
path: '(..)photos/:id' // Intercepts /photos/123 when in /dashboard

// (...) - Two Levels up
path: '(..)(..)photos/:id' // Intercepts /photos/123 when in /dashboard/nested

// (...) - From root
path: '(...)photos/:id' // Intercepts /photos/123 from anywhere

```

See [Parallel Routes Examples](#) for: - Photo gallery with modal - Dashboard with multiple slots - Email client with multi-pane layout

API

Router Creation

- `createAppRouter(options)` - Create a high-level router with declarative routes
- `createRouteManifest(routes, options)` - Generate route manifest for manual use
- `createRouteMatcher(routes, options)` - Create a route matcher function

- `generateRouteTypes(routes, options)` - Generate TypeScript types for routes

Data Loading Hooks

- `useLoaderData()` - Access loader data for the current route
- `useRouteLoaderData(routeId)` - Access loader data for a specific route
- `useMatchesData()` - Get all loader data from the route hierarchy
- `useMatches()` - Get all matched routes with data and handles
- `useLoaderLoading()` - Check if any loaders are loading

Action Hooks

- `useActionData()` - Access the result of the most recent action
- `useNavigation()` - Access current navigation/submission state
- `useSubmit()` - Programmatically submit forms
- `useFetcher()` - Create a fetcher for non-navigation mutations
- `useFetchers()` - Access all active fetchers

Error Boundary Hooks

- `useRouteError()` - Access the current route's error
- `useRouteErrorById(routeId)` - Access error for a specific route
- `useHasRouteError()` - Check if current route has an error
- `useRouteErrors()` - Get all route errors in the hierarchy

Router Hooks

- `useRouter()` - Access router state (route + navigate function)
- `useRoute()` - Access current matched route details

Parallel Routes Hooks

- `useSlot()` - Access current slot data and metadata
- `useSlotByName(name)` - Access specific slot by name
- `useSlots()` - Get all current slots
- `useInterception()` - Access route interception state
- `useInterceptedNavigation()` - Navigate with interception support

Components

- `<Link to="/path">` - Declarative navigation link
- `<RouterView />` - Renders the current route component
- `<Form method="post">` - Enhanced form with router integration
- `<Await resolve={promise}>` - Render deferred data with Suspense
- `<Outlet />` - Render child routes in layouts
- `<RouteErrorBoundary>` - Catch and handle route errors

File Discovery

- `discoverRoutes(routesDir)` - Scan directory for file-based routes
- `matchRoute(url, routes)` - Match URL against route patterns

Qwik-Style Prefetching

- `initPrefetchManager(config)` - Initialize the prefetch manager
- `getPrefetchManager()` - Get the current prefetch manager instance
- `prefetchRoute(url, mode)` - Prefetch a route (code only)
- `prefetchRouteWithData(url, options)` - Prefetch route + run data loader
- `EnhancedLink / PrefetchLink` - Link component with prefetch modes
- `usePrefetchLink(href, options)` - Hook for custom prefetch control

Intersection Observer Utilities

- `createIntersectionObserver(options)` - Create visibility observer
- `observeElement(element, options)` - Start observing an element
- `unobserveElement(element)` - Stop observing an element
- `createPrefetchZone(element, zone)` - Create prefetch trigger zone
- `getScrollDirection()` - Get current scroll direction
- `getLinksInScrollPath(links)` - Get links in scroll trajectory

Service Worker Integration

- `generatePrefetchServiceWorker(config)` - Generate SW code for caching
- `registerPrefetchServiceWorker(options)` - Register the prefetch SW
- `initServiceWorkerPrefetch()` - Initialize SW communication
- `requestSwPrefetch(url)` - Request prefetch via service worker
- `isSwCached(url)` - Check if URL is cached in SW
- `swrFetch(url, options)` - Fetch with stale-while-revalidate

Smart Preloading (Legacy)

- `initSmartPreloader(options)` - Initialize smart preloading system

- `getSmartPreloader()` - Get the current preloader instance
- `usePreload(path, options)` - Hook for manual preloading
- `preloadLink(path, options)` - Preload a specific route
- `calculateClickIntent(element)` - Calculate user's intent to click
- `predictNextRoute()` - Predict the user's next navigation

View Transitions

- `initViewTransitions()` - Initialize view transitions manager
- `getViewTransitionManager()` - Get current transition manager
- `navigateWithTransition(to, options)` - Navigate with transition effect
- `markSharedElement(element, name)` - Mark element for shared transitions
- `supportsViewTransitions()` - Check browser support
- `animateFallback(element, type)` - Fallback animation for unsupported browsers

Parallel Routes

- `createParallelRouteConfig(config)` - Create parallel route configuration
- `matchParallelRoutes(pathname, config)` - Match pathname against slots
- `loadParallelSlots(slots, request)` - Load data for all slots in parallel
- `renderParallelSlots(slots, searchParams)` - Render all matched slots
- `navigateWithInterception(to, config, mode)` - Navigate with route interception
- `closeInterception()` - Close intercepted route and restore original
- `isIntercepted()` - Check if current navigation is intercepted
- `parseInterception(path)` - Parse interception config from path

Examples

See the router in action in these example apps:

- Demo App - Full-featured demo with router, SSR, and islands
- Todo App - Simple todo app with client-side routing

Development

```
# Build the package
pnpm build

# Run tests
pnpm test

# Type checking
pnpm typecheck
```

Features

- **File-based routing** - Automatic route generation from directory structure
- **Nested layouts** - Compose layouts hierarchically
- **Data loaders** - Fetch data before rendering routes with parallel execution
- **Parallel routes** - Next.js 14 style slots with independent loading states
- **Route interception** - Soft/hard navigation for modals and overlays
- **Qwik-style prefetching** - Speculative prefetching with multiple modes (hover, visible, intent, render)
- **Smart preloading** - Intent-based route prefetching with mouse trajectory prediction
- **Service worker integration** - Cache prefetched routes with stale-while-revalidate
- **Network-aware** - Respects Save-Data header and connection speed
- **Priority queue** - visible > hover > idle prefetch ordering
- **View transitions** - Smooth page transitions with shared elements
- **Type-safe routing** - Generate TypeScript types from route definitions
- **SSR compatible** - Works with server-side rendering
- **Tiny & fast** - Minimal bundle size with maximum performance

Route File Conventions

When using file-based routing:

- `index.tsx` /
- `about.tsx` /`about`
- `products/[id].tsx` /`products/:id` (dynamic segment)
- `blog/[...slug].tsx` /`blog/*` (catch-all)
- `_layout.tsx` Layout component (not a route)
- `_component.tsx` Shared component (not a route)

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philst-router/src/index.ts

Public API

- Direct exports: Route, RouteModule, createRouter
- Re-exported names: // Action execution executeAction, // Built-in guards createAuthGuard, // Built-in middleware createLoggingContextMiddleware, // Built-in providers createUserContextProvider, // Cache management clearContextCache, // Component Await, // Component Types ViewTransitionLinkProps, // Components Form, // Components RouteErrorBoundary, // Components ViewTransitionLink, // Configuration createParallelRouteConfig, // Configuration isRouteMaskingEnabled, // Context management setCurrentRouteData, // Context providers registerContextProvider, // Core defer, // Core Types TransitionConfig, // Creation createRouteGroup, // Current route context setCurrentRouteContext, // Data loading loadNestedRouteData, // Data loading loadParallelSlots, // Debug getRouteMaskingDebugInfo, // Debug getRouterContextDebugInfo, // Discovery discoverRouteGroups, // Error creation createRouteErrorResponse, // Error handling catchRouteError, // Error recovery withErrorRecovery, // Event Types ViewTransitionEvent, // Execution runNavigationGuards, // Fallback animateFallback, // Form utilities formDataToObject, // Global context setGlobalContext, // Helpers deferData, // Hierarchy utilities getRouteIds, // History restoreMaskFromHistory, // History & Performance clearHistory, // Hooks useActionData, // Hooks useLoaderData, // Hooks useRouteError, // Hooks useRouteMask, // Hooks useRouterContext, // Hooks useSlot, // Hooks useViewTransition, // Import/Export exportState, // Initialization initRouteMasking, // Initialization initRouterContext, // Initialization initRouterDevTools, // Loader execution executeLoader, // Main component RouterDevTools, // Manager ViewTransitionManager, // Mask creation & management createRouteMask, // Matching matchNestedRoutes, // Matching matchParallelRoutes, // Middleware addContextMiddleware, // Middleware executeGroupMiddleware, // Navigation navigateWithTransition, // Navigation tracking trackNavigation, // Navigation with masking navigateWithMask, // Optimistic updates applyOptimisticUpdate, // Parsing parseRouteGroup, // Path utilities generatePath, // Processing processRouteGroups, // Registration beforeEach, // Rendering renderNestedRoutes, // Rendering renderParallelSlots, // Route builders createRoute, // Route interception navigateWithInterception, // Route overrides registerRouteContextOverride, // Shared Element Types SharedElementOptions, // Shared Elements markSharedElement, // Stack management pushMask, // State Types ViewTransitionState, // State access getCurrentMask, // State access getDevToolsState, // State management setActionData, // State management setRouteError, // State management updateParallelRouteState, // State updates updateRouteTree, // Status helpers isClientError, // Streaming streamDeferred, // Support Detection supportsViewTransitions, // Type guards isRouteErrorResponse // Type-safe helpers createTypedContext, // UI controls toggleDevTools, // Utilities createLoaderRequest, // Utilities getNavigationDirection, // Utilities getRoutesByGroup, // Utilities matchesMask, // Utilities parseLocation, // Utilities validateContext, ActionFunction, ActionFunctionContext, ActionResult, AfterNavigationHook, AwaitProps, Breadcrumb, BreadcrumbConfig, BuilderResult, CacheStats, CachedRoute, ContextMiddleware, ContextProvider, ContextValidator, DefaultErrorBoundary, DeferredData, DeferredStatus, DeferredValue, DevToolsConfig, EnhancedLink, EnhancedLinkProps, ErrorBoundaryComponent, ErrorBoundaryProps, ErrorRecoveryOptions, FallbackBehavior, FetcherState, FormProps, GroupRoute, GuardConfig, GuardContext, InferLoaderData, InterceptConfig, InterceptedNavigationState, IntersectionOptions, LayoutChain, LayoutComponent, Link, LinkPrefetchOptions, LinkProps, LoaderFunction, LoaderFunctionContext, LoaderOptions, LoaderResult, MaskMatchStrategy, MaskRestoreOptions, MaskStackEntry, MaskedNavigationOptions, MatchedNestedRoute, MatchedRoute, MatchedSlot, MiddlewareResult, NavigateFunction, NavigationFailure, NavigationFailureType, NavigationGuard, NavigationGuardReturn, NavigationHistoryEntry, NavigationMetrics, NavigationMode, NavigationState, NestedRouteDefinition, NestedRouteMatch, NestedRouteOptions, ObservedElement, OptimisticUpdate, Outlet, ParallelRouteConfig, ParamValue, PrefetchCacheConfig, PrefetchConfig, PrefetchLink, PrefetchManager, PrefetchMessage, PrefetchMode, PrefetchOptions, PrefetchPriority, PrefetchQueueItem, PrefetchStats, PrefetchZone, PreloadOptions, PreloadStrategy, ProcessedGroupRoute, QueryParams, QueryValue, RouteComponent, RouteComponentProps, RouteContextOverride, RouteDefinition, RouteError, RouteErrorBoundaryProps, RouteErrorHandler, RouteErrorResponse, RouteGroup, RouteGroupConfig, RouteGroupMeta, RouteGroupMiddleware, RouteLoaderData, RouteLocation, RouteManifestOptions, RouteMask, RouteMatch, RouteMatchDebugInfo, RouteMatcher, RouteMeta, RouteParams, RoutePattern, RoutePerformance, RouteStateSnapshot, RouteTransitionOptions, RouteTreeNode, RouteTypeGenerationOptions, RouterContext, RouterContextConfig, RouterOptions, RouterView, SWRFetchOptions, ScrollDirection, SlotComponent, SlotComponentProps, SlotDefinition, SlotName, SmartPreloader, StreamOptions, SubmitOptions, SwRegistrationOptions, TransitionDirection, TransitionType, TypedRouterContext, URLBuilder, URLBuilderOptions, UsePrefetchLinkOptions, UsePrefetchLinkResult, UserIntentData, ValidationError, ValidationResult, ViewTransitionConfig, ViewTransitionEventDetail, ViewTransitionEventHandler, ViewTransitionOptions, VisibilityState, addRouteToGroup, afterEach, applyContextMiddleware, applyLayouts, applyRouteMask, awaitAllDeferred, beforeRoute, buildQueryString, calculateClickIntent, clearActionData, clearAllGuards, clearAllRouteErrors, clearLoaderData, clearMaskHistory, clearMaskStack, clearOptimisticUpdates, clearParallelRouteState, clearPerformance, clearRouteError, clearSwCache, cloneContext, closeInterception, closeOverlay, closeServiceWorkerPrefetch, completeNavigation, computeProvidedContext, createContext, createActionRequest, createAnalyticsGuard, createApiClientProvider, createAppRouter, createAuthMiddleware, createCatchAllRoute, createConfirmGuard, createErrorResponse, createFilterContextMiddleware, createHeaderMiddleware, createIndexRoute, createInlineServiceWorker, createIntersectionObserver, createLayoutRoute, createLoadingGuard, createLocaleContextProvider, createLocation, createLoggingMiddleware, createNestedGroups, createOutlet, createPermissionGuard, createPermissionMiddleware, createPrefetchZone, createRateLimitGuard, createRateLimitMiddleware, createRetryHandler, createRoleGuard, createRouteManifest, createRouteMatcher, createScrollGuard, createThemeContextProvider, createTitleGuard, createTransformContextMiddleware, createURLBuilder, createViewTransitionLink, defineContextMiddleware, defineContextProvider, defineRoutes, deserializeMask, deserializeRouteState, detectMaskFromHistory, disconnectAll, discoverRoutes, executeLoadersParallel, executeNestedAction, executeNestedLoaders, exportContextState, exportMaskingState, exportRouteGroups, extractGroups, extractParamNames, findLayouts, findRouteById, generateBreadcrumbs, generatePrefetchServiceWorker, generateRouteTypes, getActualRoute, getAncestorRoutes, getContextKeys, getCurrentRouteContext, getDeferredStates, getGlobalContext, getGlobalContextValue, getCounts, getGuardCount, getInterceptionHistory, getLinksInScrollPath, getMaskConfig, getMaskFromHistory, getMaskStack, getMaskStackDepth, getMaskedUrl, getNavigationStatus, getParentRoute, getPrefetchManager, getRedirectLocation, getRouteContext, getScrollDirection, getSmartPreloader, getSwCachedUrls, getViewTransitionManager, getVisibilityState, handleRouteError, hasContextKey, hasIntersected, hydrateDeferred, importState, initPrefetchManager, initServiceWorkerPrefetch, initSmartPreloader, initViewTransitions, invalidateLoaderCache, isActivePath, isApproachingViewport, isDeferred, isGroupPath, isIntercepted, isNavigationCancelled, isObserving, isRedirectResponse, isRouteMasked, isServerError, isSuccessStatus, isSwCached, joinPaths, json, markErrorHandled, matchPattern, matchRoute, mergeContexts, mergeQueryParams, mergeRouteGroups, navigate, navigateAsDrawer, navigateAsModal, normalizePath, objectToFormData, observeElement, onScrollDirectionChange, onSwMessage, parseInterception, parseParams, parseQueryString, parseURL, popMask, predictNextRoute, prefersReducedMotion, prefetchRoute, prefetchRouteWithData, preloadLink, recordRouteMatch, redirect, registerPrefetchServiceWorker, removeContextMiddleware, removeGroups, removeRouteMask, requestSwPrefetch, resetRouterContext, resetViewTransitions, resolveDeferred, resolveLinkTo, revalidate, runAfterHooks, serializeDeferred, serializeMask, serializeRouteState, setActiveTab, setCurrentRouteError, setErrorStack, setNavigationState, setOutletContext, setRouteMaskingEnabled, startViewTransition, swrFetch, throwBadRequest, throwForbidden, throwNotFound, throwResponse, throwServerError, throwUnauthorized, toggleMinimize, trackLoader, transitionLink, unobserveElement, unregisterContextProvider, unregisterRouteContextOverride, updateGlobalContext, updateGlobalContextMultiple, updateMaskConfig, updateRouteState, useActualRoute, useFetcher, useFetchers, useHasRouteError, useInterceptedNavigation, useInterception, useRouteMasked, useLoaderLoading, useMaskState, useMaskedUrl, useMatches, useMatchesData, useNavigation, useOptimisticUpdates, useOutletContext, usePrefetchLink, usePreload, useRoute, useRouteErrorByld, useRouteErrors, useRouteLoaderData, useRouter, useRouterContextValue, useSlotByName, useSlots, useSubmit, useTransitionName, useTransitionPersist, useUpdateRouterContext, useViewTransitionEvent, validateFormData, validateRouteGroup, visualizeRouteGroups
- Re-exported modules: ./action.js, ./defer.js, ./devtools.js, ./discover.js, ./error-boundary.js, ./guards.js, ./high-level.js, ./intersection.js, ./layouts.js, ./link.js, ./loader.js, ./nested.js, ./parallel-routes.js, ./prefetch.js, ./route-groups.js, ./route-masking.js, ./router-context.js, ./service-worker-prefetch.js, ./smart-preload.js, ./url-builder.js, ./view-transitions.js

License

MIT

@philijs/router-typesafe - Type: Node package - Purpose: Type-safe router for PhilJS with fully typed params and Zod validation - Version: 0.1.0 - Location: packages/philijs-router-typesafe - Entry points: packages/philijs-router-typesafe/src/index.ts - Keywords: philjs, router, routing, navigation, type-safe, typescript, zod, validation

philjs-router-typesafe

A fully type-safe router for PhilJS inspired by [TanStack Router](#). Get complete type inference for route params, search params, and loader data without manual type annotations.

Features

- **Fully Typed Route Params:** Define '/users/\$userId/posts/\$postId' and get { userId: string; postId: string } automatically
- **Zod Search Param Validation:** Validate and type search params with Zod schemas

- **Type-Safe Link Component:** Links are type-checked at compile time
- **Route-Attached Hooks:** Each route has `useParams()`, `useSearch()`, and `useLoaderData()` methods
- **Data Loaders:** Load data before rendering with full type inference
- **SSR Support:** Server-side rendering with data preloading
- **Zero Runtime Type Overhead:** Types are erased at compile time

Installation

```
npm install philjs-router-typesafe zod
# or
pnpm add philjs-router-typesafe zod
# or
yarn add philjs-router-typesafe zod
```

Quick Start

```
import { z } from 'zod';
import { createRoute, Router, RouterOutlet, Link } from 'philjs-router-typesafe';

// Define routes with full type safety
const userRoute = createRoute({
  path: '/users/:userId',
  validateSearch: z.object({
    tab: z.enum(['posts', 'comments']).default('posts'),
    page: z.number().optional(),
  }),
  loader: async ({ params, search }) => {
    // params.userId is typed as string
    // search.tab is typed as 'posts' | 'comments'
    // search.page is typed as number | undefined
    const user = await fetchUser(params.userId);
    const posts = await fetchPosts(params.userId, {
      type: search.tab,
      page: search.page
    });
    return { user, posts };
  },
  component: ({ params, search, loaderData }) => (
    <div>
      <h1>User: {loaderData.user.name}</h1>
      <UserTabs active={search.tab} userId={params.userId} />
      <PostList posts={loaderData.posts} />
    </div>
  ),
});
// App setup
function App() {
  return (
    <Router routes={[userRoute]}>
      <nav>
        {/* Type-safe Links - params and search are validated */}
        <Link
          to={userRoute}
          params={{ userId: '123' }}
          search={{ tab: 'posts' }}
        >
          View User
        </Link>
      </nav>
      <RouterOutlet />
    </Router>
  );
}
```

API Reference

Route Creation

```
createRoute(options)
```

Create a type-safe route definition.

```

const route = createRoute({
  // Path with dynamic segments prefixed with $
  path: '/users/${userId}/posts/${postId}',

  // Optional: Zod schema for search params validation
  validateSearch: z.object({
    sort: z.enum(['asc', 'desc']).default('desc'),
    filter: z.string().optional(),
  }),

  // Optional: Data Loader
  loader: async ({ params, search, request, abortController }) => {
    const response = await fetch(`api/users/${params.userId}/posts/${params.postId}`);
    return response.json();
  },

  // Optional: Route component
  component: ({ params, search, loaderData, navigate }) => {
    return <div>...</div>;
  },

  // Optional: Error boundary component
  errorComponent: ({ error, reset }) => {
    return <div>Error: {error.message} <button onClick={reset}>Retry</button></div>;
  },

  // Optional: Loading component
  pendingComponent: () => <div>Loading...</div>,

  // Optional: Guard/redirect logic
  beforeLoad: async ({ params, search, location, cause }) => {
    if (!isAuthenticated()) {
      throw redirect('/login');
    }
  },

  // Optional: Metadata
  meta: {
    title: 'User Post',
    description: 'View a user post',
  },
});

createRootRoute(options)

```

Create a root layout route.

```

const rootRoute = createRootRoute({
  component: ({ children }) => (
    <div class="app-layout">
      <Header />
      <main>{children}</main>
      <Footer />
    </div>
  ),
});

```

addChildren(parent, children)

Add child routes to a parent.

```

const routeTree = addChildren(rootRoute, [
  indexRoute,
  aboutRoute,
  addChildren(usersRoute, [
    userListRoute,
    userDetailRoute,
  ]),
]);

```

Hooks

useParams(route?)

Get typed route params.

```

function UserProfile() {
  // Typed: { userId: string }
  const { userId } = userRoute.useParams();

  // Or without specifying route (less type-safe)
  const params = useParams();

  return <div>User ID: {userId}</div>;
}

```

useSearch(route?)

Get typed and validated search params.

```
function UserPosts() {
  // Typed: { tab: 'posts' | 'comments'; page?: number }
  const { tab, page } = userRoute.useSearch();

  return <PostList type={tab} page={page ?? 1} />;
}
```

useLoaderData(route?)

Get typed loader data.

```
function UserDetails() {
  // Typed based on Loader return type
  const { user, posts } = userRoute.useLoaderData();

  return (
    <div>
      <h1>{user.name}</h1>
      <span>{posts.length} posts</span>
    </div>
  );
}

useNavigate()
```

Get the navigate function.

```
function LogoutButton() {
  const navigate = useNavigate();

  const handleLogout = async () => {
    await logout();
    navigate('/login', { replace: true });
  };

  return <button onClick={handleLogout}>Logout</button>;
}
```

useNavigateTyped(route)

Get a type-safe navigate function for a specific route.

```
function UserCard({ userId }: { userId: string }) {
  const navigateToUser = useNavigateTyped(userRoute);

  return (
    <button onClick={() => navigateToUser({
      params: { userId },
      search: { tab: 'posts' }
    })}
      View User
    </button>
  );
}
```

useLocation()

Get the current location.

```
function Breadcrumbs() {
  const location = useLocation();
  return <span>Current: {location.pathname}</span>;
}
```

useMatchRoute(route, options?)

Check if a route is active.

```
function NavLink({ route, children }) {
  const isActive = useMatchRoute(route);
  return <a class={isActive ? 'active' : ''}>{children}</a>;
}
```

useMatches()

Get all matched routes in the hierarchy.

```

function Breadcrumbs() {
  const matches = useMatches();

  return (
    <nav aria-label="Breadcrumb">
      {matches.map((match) => (
        <span key={match.route.id}>
          {match.route.meta?.title}
        </span>
      ))}
    </nav>
  );
}

usePreloadRoute(route)

```

Preload a route's data on hover/focus.

```

function UserLink({ userId }) {
  const preload = usePreloadRoute(userRoute);

  return (
    <a
      href={`/users/${userId}`}
      onMouseEnter={() => preload({ params: { userId } })}
    >
      User {userId}
    </a>
  );
}

```

Components

Router

The router provider component.

```

<Router
  routes={[route1, route2, route3]}
  basePath="/app"
  defaultPendingComponent={() => <Spinner />}
  defaultErrorComponent={({ error, reset }) => (
    <ErrorMessage error={error} onRetry={reset} />
  )}
  notFoundComponent={() => <NotFound />}
  onNavigate={(event) => {
    analytics.track('navigation', {
      from: event.from?.route.path,
      to: event.to.route.path
    });
  }}
>
  {children}
</Router>

```

RouterOutlet

Renders the matched route component.

```

function App() {
  return (
    <Router routes={routes}>
      <Header />
      <RouterOutlet />
      <Footer />
    </Router>
  );
}

```

Link

Type-safe navigation component.

```
// With route object (fully type-safe)
<Link
  to={userRoute}
  params={{ userId: '123' }}
  search={{ tab: 'posts' }}
  hash="section-1"
  replace={false}
  prefetch="intent"
  className="link"
  activeClassName="link--active"
>
  View User
</Link>

// With string path (less type-safe)
<Link to="/about">About</Link>
```

Redirect

Performs a redirect on mount.

```
function ProtectedRoute() {
  if (!isLoggedIn) {
    return <Redirect to={loginRoute} replace />;
  }
  return <Dashboard />;
}
```

Utilities

```
redirect(to, options?)
```

Throw a redirect from beforeLoad.

```
const protectedRoute = createRoute({
  path: '/dashboard',
  beforeLoad: async () => {
    if (!isAuthenticated()) {
      throw redirect('/login');
    }
  },
});
```

```
buildPath(pattern, params)
```

Build a URL from a pattern and params.

```
const url = buildPath('/users/${userId}/posts/${postId}', {
  userId: '123',
  postId: '456',
});
// => '/users/123/posts/456'
```

```
parsePathParams(pattern, pathname)
```

Extract params from a pathname.

```
const params = parsePathParams(
  '/users/${userId}/posts/${postId}',
  '/users/123/posts/456'
);
// => { userId: '123', postId: '456' }
```

SSR Support

```
createSSRRouter(options)
```

Create a router for server-side rendering.

```
import { createSSRRouter, loadRouteData } from 'philjs-router-typesafe';

async function renderApp(url: string) {
  const { router, context } = createSSRRouter({
    routes: [homeRoute, userRoute],
    url,
  });

  // Load data
  const { match, data, error } = await loadRouteData(routes, url);

  // Render to string
  const html = renderToString(<App />);

  return html;
}
```

Type Inference Examples

Path Params

```
// Path: '/users/$userId/posts/$postId'  
// Inferred params type: { userId: string; postId: string }  
  
const route = createRoute({  
  path: '/users/$userId/posts/$postId',  
  component: ({ params }) => {  
    // TypeScript knows:  
    // params.userId: string  
    // params.postId: string  
    // params.other: ERROR - Property does not exist  
  },  
});
```

Search Params

```
const route = createRoute({  
  path: '/search',  
  validateSearch: z.object({  
    query: z.string(),  
    page: z.number().default(1),  
    filters: z.array(z.string()).optional(),  
  }),  
  component: ({ search }) => {  
    // TypeScript knows:  
    // search.query: string  
    // search.page: number  
    // search.filters: string[] | undefined  
  },  
});
```

Loader Data

```
interface User {  
  id: string;  
  name: string;  
  email: string;  
}  
  
const route = createRoute({  
  path: '/users/$userId',  
  loader: async ({ params }): Promise<User> => {  
    const response = await fetch(`api/users/${params.userId}`);  
    return response.json();  
  },  
  component: ({ loaderData }) => {  
    // TypeScript knows LoaderData: User  
    return <h1>{loaderData.name}</h1>;  
  },  
});
```

Type-Safe Links

```
// This compiles:  
<Link to={userRoute} params={{ userId: '123' }} search={{ tab: 'posts' }}>  
  Valid  
</Link>  
  
// These cause type errors:  
<Link to={userRoute} params={{ wrongParam: '123' }}> // Error: missing userId  
<Link to={userRoute} params={{ userId: '123' }} search={{ tab: 'invalid' }}> // Error: invalid tab value
```

Best Practices

1. Define Routes in a Separate File

```
// routes.ts  
export const homeRoute = createRoute({ path: '/', ... });  
export const userRoute = createRoute({ path: '/users/$userId', ... });  
export const settingsRoute = createRoute({ path: '/settings', ... });  
  
export const routes = [homeRoute, userRoute, settingsRoute];
```

2. Use Route-Attached Hooks

```
// Prefer this:  
const { userId } = userRoute.useParams();  
  
// Over this (less type-safe):  
const { userId } = useParams();
```

3. Validate All Search Params

```
// Always provide validation for search params
const route = createRoute({
  path: '/search',
  validateSearch: z.object({
    q: z.string().min(1),
    page: z.coerce.number().positive().default(1),
  }),
});
```

4. Handle Loading and Error States

```
const route = createRoute({
  path: '/data',
  loader: async () => fetchData(),
  pendingComponent: () => <Skeleton />,
  errorComponent: ({ error, reset }) => (
    <ErrorBoundary error={error} onRetry={reset} />
  ),
});
```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philijs-router-typesafe/src/index.ts

Public API

- Direct exports: (none detected)
- Re-exported names: // Component Types RouteComponentProps, // Link Types LinkPropsWithRoute, // Loader Types LoaderContext, // Navigation Types NavigateOptions, // Path Parameter Types ExtractPathParams, // Route Tree Types RootRouteOptions, // Route Types RouteOptions, // Router Types RouterOptions, // Search Parameter Types InferSearchParams, // Utility Types RequireKeys, ActiveLink, AllRoutePaths, BeforeLoadContext, DeepPartial, ErrorComponent, ErrorComponentProps, ExtractRoute, HasParams, InferLoaderData, Link, LinkProps, LinkPropsWithPath, LoaderFn, MatchedRoute, NavLink, NavigateFn, NavigationEvent, NavigationRedirect, PathParams, PendingComponent, Prettify, Redirect, RegisteredRoutes, RouteComponent, RouteDefinition, RouteMeta, RouteTree, Router, RouterContextType, RouterLocation, RouterOutlet,SearchParamsOrEmpty, TypeSafeRouter, TypedNavigateOptions, UnionToIntersection, addChildren, buildPath, createNavigateLink, createRootRoute, createRoute, createRouteWithChildren, createRouter, createSSRRouter, flattenRouteTree, getActiveRoute, getRouterContext, loadRouteData, matchRoutes, matchesRoute, parsePathParams, parseSearchParams, redirect, serializeSearchParams, useBlocker, usesPending, useLoaderData, useLocation, useMatchRoute, useMatches, useNavigate, useNavigateTyped, useParams, usePreloadRoute, useRouteError, useRouter, useSearch
- Re-exported modules: ./context.js, ./hooks.js, ./link.js, ./route.js, ./router.js, ./types.js

License

MIT

@philijs/rpc - Type: Node package - Purpose: tRPC-style end-to-end type-safe RPC system for PhilJS - Version: 0.1.0 - Location: packages/philijs-rpc - Entry points: packages/philijs-rpc/src/index.ts, packages/philijs-rpc/src/client.ts, packages/philijs-rpc/src/server.ts - Keywords: philijs, rpc, trpc, type-safe, api, typescript

philijs-rpc

A tRPC-style end-to-end type-safe RPC system for PhilJS. Build fully type-safe APIs with automatic type inference from server to client.

Features

- **End-to-end type safety:** Full TypeScript inference from API definition to client usage
- **React Query-style hooks:** useQuery, useMutation, and useSubscription hooks built on PhilJS signals
- **Real-time subscriptions:** WebSocket and SSE support for live data streaming
- **tRPC-style links:** Composable request/response transformations and routing
- **Input validation:** Built-in Zod integration for runtime validation
- **Middleware support:** Composable middleware for auth, logging, rate limiting, etc.
- **Multiple runtime adapters:** Works with Node.js, Express, Vercel, Netlify, Cloudflare Workers
- **Request batching:** Automatic request batching for improved performance
- **Caching:** Built-in query caching with stale-while-revalidate support
- **Automatic reconnection:** Resilient WebSocket connections with exponential backoff
- **Transport fallback:** Graceful degradation from WebSocket to SSE

Installation

```
npm install philijs-rpc
# or
pnpm add philijs-rpc
# or
yarn add philijs-rpc
```

Optional peer dependency for input validation:

```
npm install zod
```

Quick Start

1. Define your API (Server)

```

// src/server/api.ts
import { createAPI, procedure } from 'philjs-rpc';
import { z } from 'zod';

// Define your database or data Layer
const db = {
  users: {
    findMany: async () => [{ id: '1', name: 'John', email: 'john@example.com' }],
    findUnique: async ({ where }: { where: { id: string } }) => ({
      id: where.id,
      name: 'John',
      email: 'john@example.com',
    }),
    create: async ({ data }: { data: { name: string; email: string } }) => ({
      id: '2',
      ...data,
    }),
    delete: async ({ where }: { where: { id: string } }) => ({ id: where.id }),
  },
};

export const api = createAPI({
  users: {
    // Simple query (no input)
    list: procedure.query(async () => {
      return db.users.findMany();
    }),

    // Query with input validation
    getById: procedure
      .input(z.object({ id: z.string() }))
      .query(async ({ input }) => {
        return db.users.findUnique({ where: { id: input.id } });
      }),
  },

  // Mutation with input validation
  create: procedure
    .input(
      z.object({
        name: z.string().min(1, 'Name is required'),
        email: z.string().email('Invalid email'),
      })
    )
    .mutation(async ({ input }) => {
      return db.users.create({ data: input });
    }),

    // Mutation with delete
    delete: procedure
      .input(z.object({ id: z.string() }))
      .mutation(async ({ input }) => {
        return db.users.delete({ where: { id: input.id } });
      }),
  },

  // Nested namespaces
  posts: {
    list: procedure.query(async () => {
      return [{ id: '1', title: 'Hello World' }];
    }),

    comments: {
      list: procedure
        .input(z.object({ postId: z.string() }))
        .query(async ({ input }) => {
          return [{ id: '1', postId: input.postId, content: 'Great post!' }];
        }),
    },
  };
};

// Export the API type for client usage
export type AppAPI = typeof api;

```

2. Create the Server Handler

```
// src/server/handler.ts
import { createHandler } from 'philjs-rpc/server';
import { api } from './api';

export const handler = createHandler(api, {
  createContext: async (req) => {
    // Extract auth info from headers
    const token = req.headers.authorization?.replace('Bearer ', '');
    return {
      user: token ? await validateToken(token) : null,
    };
  },
  onError: (error, ctx) => {
    console.error('RPC Error:', error);
  },
});
});
```

3. Set Up Server Routes

Node.js HTTP Server

```
import http from 'http';
import { createNodeHandler } from 'philjs-rpc/server';
import { api } from './api';

const handler = createNodeHandler(api);

http.createServer(handler).listen(3000, () => {
  console.log('Server running on http://localhost:3000');
});
```

Express.js

```
import express from 'express';
import { createExpressHandler } from 'philjs-rpc/server';
import { api } from './api';

const app = express();
app.use(express.json());
app.post('/api/rpc', createExpressHandler(api));
app.listen(3000);
```

Vercel Serverless Functions

```
// api/rpc.ts
import { createVercelHandler } from 'philjs-rpc/server';
import { api } from '../src/server/api';

export default createVercelHandler(api);
```

Netlify Functions

```
// netlify/functions/rpc.ts
import { createNetlifyHandler } from 'philjs-rpc/server';
import { api } from '../../src/server/api';

export const handler = createNetlifyHandler(api);
```

Cloudflare Workers / Deno / Bun

```
import { createFetchHandler } from 'philjs-rpc/server';
import { api } from './api';

const handler = createFetchHandler(api);

export default {
  fetch: handler,
};
```

4. Create the Client

```
// src/client/rpc.ts
import { createClient } from 'philjs-rpc/client';
import type { AppAPI } from '../server/api';

export const client = createClient<AppAPI>({
  url: '/api/rpc',
  // Optional: Add auth headers
  headers: () => ({
    Authorization: `Bearer ${getAuthToken()}`,
  }),
});
```

5. Use in Components

```

// src/components/UsersList.tsx
import { client } from '../client/rpc';

function UsersList() {
  // Query hook - automatically fetches and caches
  const users = client.users.list.useQuery({
    staleTime: 5000, // Consider data stale after 5 seconds
    refetchOnWindowFocus: true,
  });

  if (users.isLoading) {
    return <div>Loading users...</div>;
  }

  if (users.isError) {
    return <div>Error: {users.error.message}</div>;
  }

  return (
    <ul>
      {users.data?.map((user) => (
        <li key={user.id}>
          {user.name} ({user.email})
        </li>
      ))}
    </ul>
  );
}

function UserProfile({ userId }: { userId: string }) {
  // Query with input
  const user = client.users.byId.useQuery(
    { id: userId },
    { enabled: !!userId }
  );

  if (user.isLoading) return <div>Loading...</div>;
  if (user.isError) return <div>Error: {user.error.message}</div>;

  return (
    <div>
      <h1>{user.data?.name}</h1>
      <p>{user.data?.email}</p>
    </div>
  );
}

function CreateUserForm() {
  // Mutation hook
  const createUser = client.users.create.useMutation({
    onSuccess: (data) => {
      console.log('User created:', data);
    },
    onError: (error) => {
      console.error('Failed to create user:', error.message);
    },
  });

  const handleSubmit = (e: Event) => {
    e.preventDefault();
    const form = e.target as HTMLFormElement;
    const formData = new FormData(form);

    createUser.mutate({
      name: formData.get('name') as string,
      email: formData.get('email') as string,
    });
  };

  return (
    <form onSubmit={handleSubmit}>
      <input name="name" placeholder="Name" required />
      <input name="email" type="email" placeholder="Email" required />
      <button type="submit" disabled={createUser.isLoading}>
        {createUser.isLoading ? 'Creating...' : 'Create User'}
      </button>
      {createUser.isError && (
        <div class="error">{createUser.error.message}</div>
      )}
    </form>
  );
}

```

6. Direct Fetch Calls

```
// Outside of components, use .fetch() for direct calls
async function loadInitialData() {
  const users = await client.users.list.fetch();
  const user = await client.users.byId.fetch({ id: '123' });

  // Mutations
  const newUser = await client.users.create.fetch({
    name: 'Jane Doe',
    email: 'jane@example.com',
  });
}

}
```

Middleware

Built-in Middleware

```
import {
  createAPI,
  procedure,
  loggerMiddleware,
  rateLimitMiddleware,
  createAuthMiddleware,
  permissionMiddleware,
  cacheMiddleware,
} from 'philjs-rpc';

// Logger middleware
const withLogging = procedure.use(
  loggerMiddleware({
    logInput: true,
    logOutput: true,
  })
);

// Rate limiting
const withRateLimit = procedure.use(
  rateLimitMiddleware({
    limit: 100,
    windowMs: 60000, // 1 minute
  })
);

// Authentication
const authMiddleware = createAuthMiddleware({
  validateToken: async (token) => {
    // Return user or null
    return await verifyJWT(token);
  },
});

const authedProcedure = procedure.use(authMiddleware);

// Permission checking
const adminProcedure = authedProcedure.use(
  permissionMiddleware(['admin'], { mode: 'any' })
);

// Caching for queries
const cachedProcedure = procedure.use(
  cacheMiddleware({
    ttl: 60000, // 1 minute
  })
);
```

Custom Middleware

```

import { createMiddleware, RPCError } from 'philjs-rpc';

// Custom Logging middleware
const customLogger = createMiddleware(async ({ ctx, input, next, type, path }) => {
  console.log(`[${type}] ${path} , { input }`);

  const start = Date.now();
  const result = await next(ctx);
  const duration = Date.now() - start;

  console.log(`[${type}] ${path} completed in ${duration}ms`);

  return result;
});

// Validation middleware
const validateRequest = createMiddleware(async ({ ctx, next }) => {
  if (!ctx.user) {
    return {
      ok: false,
      error: new RPCError({
        code: 'UNAUTHORIZED',
        message: 'Authentication required',
      }),
    };
  }

  // Add additional context
  return next({
    ...ctx,
    isAuthenticated: true,
  });
});

```

Error Handling

```

import { RPCError } from 'philjs-rpc';

const api = createAPI({
  users: {
    byId: procedure
      .input(z.object({ id: z.string() }))
      .query(async ({ input }) => {
        const user = await db.users.findUnique({ where: { id: input.id } });

        if (!user) {
          throw new RPCError({
            code: 'NOT_FOUND',
            message: `User with id ${input.id} not found`,
          });
        }

        return user;
      }),
  },
});

```

Error Codes

Code	HTTP Status	Description
BAD_REQUEST	400	Invalid input or request
UNAUTHORIZED	401	Missing or invalid auth
FORBIDDEN	403	Insufficient permissions
NOT_FOUND	404	Resource not found
METHOD_NOT_ALLOWED	405	Wrong procedure type
TIMEOUT	408	Request timed out
CONFLICT	409	Resource conflict
PRECONDITION_FAILED	412	Precondition not met
PAYLOAD_TOO_LARGE	413	Request body too large
UNPROCESSABLE_ENTITY	422	Semantic validation error
TOO_MANY_REQUESTS	429	Rate limit exceeded
INTERNAL_SERVER_ERROR	500	Unexpected server error

Advanced Usage

Request Batching

```

const client = createClient<AppAPI>({
  url: '/api/rpc',
  batching: true,
  batchWindowMs: 10, // Batch requests within 10ms
});

// These requests will be batched into a single HTTP request
const [users, posts] = await Promise.all([
  client.users.list.fetch(),
  client.posts.list.fetch(),
]);

```

Query Invalidation

```

import { invalidateQueries } from 'philjs-rpc/client';

const createUser = client.users.create.useMutation({
  onSuccess: () => {
    // Invalidate user queries after creating
    invalidateQueries('users');
  },
});

```

Prefetching

```

import { prefetchQuery } from 'philjs-rpc/client';

// Prefetch on hover
<button
  onMouseEnter={() => prefetchQuery(client.users.byId, { id: '123' })}
  onClick={() => navigate('/users/123')}
>
  View User
</button>

```

Custom Context Types

```

import { createProcedureBuilder, ProcedureContext } from 'philjs-rpc';

interface AuthContext extends ProcedureContext {
  user: {
    id: string;
    email: string;
    roles: string[];
  };
}

const authedProcedure = createProcedureBuilder<AuthContext>();

const api = createAPI({
  profile: {
    get: authedProcedure.query(async ({ ctx }) => {
      // ctx.user is fully typed
      return ctx.user;
    }),
  },
});

```

Organizing Large APIs

```

// src/server/routers/users.ts
import { createRouter, procedure } from 'philjs-rpc';
import { z } from 'zod';

export const usersRouter = createRouter({
  list: procedure.query(async () => db.users.findMany()),
  byId: procedure
    .input(z.object({ id: z.string() }))
    .query(async ({ input }) => db.users.findUnique({ where: { id: input.id } })),
});

// src/server/routers/posts.ts
export const postsRouter = createRouter({
  list: procedure.query(async () => db.posts.findMany()),
});

// src/server/api.ts
import { createAPI, mergeRouters } from 'philjs-rpc';
import { usersRouter } from './routers/users';
import { postsRouter } from './routers/posts';

export const api = createAPI(
  mergeRouters({
    users: usersRouter,
    posts: postsRouter,
  })
);

```

Type Inference Utilities

```
import type {
  InferProcedureInput,
  InferProcedureOutput,
  InferRouter,
  GetProcedureInput,
  GetProcedureOutput,
} from 'philjs-rpc';

// Infer input type from a procedure
type CreateUserInput = InferProcedureInput<typeof api._router.users.create>;
// { name: string; email: string }

// Infer output type from a procedure
type CreateUserOutput = InferProcedureOutput<typeof api._router.users.create>;
// { id: string; name: string; email: string }

// Get input/output at a path
type UserById = GetProcedureOutput<typeof api._router, 'users.byId'>;
```

API Reference

Server

- `createAPI(router)` - Create an API definition
- `createHandler(api, options?)` - Create a generic request handler
- `createNodeHandler(api, options?)` - Node.js HTTP handler
- `createExpressHandler(api, options?)` - Express.js handler
- `createVercelHandler(api, options?)` - Vercel serverless handler
- `createNetlifyHandler(api, options?)` - Netlify Functions handler
- `createFetchHandler(api, options?)` - Fetch API handler (Workers, Deno, Bun)

Client

- `createClient<API>(config)` - Create a type-safe client
- `invalidateQueries(pathPattern)` - Invalidate cached queries
- `prefetchQuery(procedure, input)` - Prefetch a query
- `getQueryCache()` - Access the query cache

Procedure

- `procedure.input(schema)` - Add input validation
- `procedure.use(middleware)` - Add middleware
- `procedure.query(handler)` - Create a query procedure
- `procedure.mutation(handler)` - Create a mutation procedure
- `procedure.subscription(handler)` - Create a subscription procedure

Middleware

- `createMiddleware(fn)` - Create custom middleware
- `loggerMiddleware(options?)` - Logging middleware
- `rateLimitMiddleware(options)` - Rate limiting middleware
- `createAuthMiddleware(options)` - Authentication middleware
- `permissionMiddleware(permissions, options?)` - Permission checking
- `cacheMiddleware(options?)` - Query caching middleware
- `retryMiddleware(options?)` - Retry failed requests

Subscriptions

- `WebSocketConnection(config)` - WebSocket connection manager
- `SSEConnection(config)` - Server-Sent Events connection manager
- `createAutoTransport(config)` - Automatic transport selection (WebSocket/SSE)
- `createUseSubscription(connection, path)` - Create subscription hook
- `createUseSSESubscription(connection, path)` - Create SSE subscription hook

Subscription Middleware

- `createSubscriptionAuthMiddleware(options)` - Auth for subscriptions
- `createSubscriptionRateLimitMiddleware(options)` - Rate limiting for subscriptions
- `createBackpressureMiddleware(options)` - Backpressure handling
- `createConnectionLimitMiddleware(options)` - Connection limits
- `createSubscriptionFilterMiddleware(options)` - Filter subscription data
- `createMultiplexingMiddleware(options)` - Share subscriptions

Links

- `createHttpLink(options)` - HTTP transport for queries/mutations
- `createWebSocketLink(options)` - WebSocket transport for subscriptions
- `createSplitLink(options)` - Route operations based on type
- `createBatchLink(options)` - Batch multiple operations

- `createDeduplicationLink(options?)` - Deduplicate in-flight requests
- `createRetryLink(options?)` - Automatic retry with backoff
- `createLoggingLink(options?)` - Log operations and results
- `createLinkChain(links)` - Compose multiple links

Real-time Subscriptions

PhilJS RPC provides full support for real-time data streaming with both WebSocket and Server-Sent Events (SSE) transports.

WebSocket Subscriptions

Define a subscription procedure:

```
import { createAPI, procedure } from 'philjs-rpc';
import { z } from 'zod';

export const api = createAPI({
  // Real-time message subscription
  onMessage: procedure
    .input(z.object({ roomId: z.string() }))
    .subscription(async function* ({ input }) {
      // Asym generator that yields data over time
      while (true) {
        const message = await getNextMessage(input.roomId);
        yield message;
      }
    }),
  // Stock price updates
  onPriceUpdate: procedure
    .input(z.object({ symbol: z.string() }))
    .subscription(async function* ({ input }) {
      for await (const price of watchStock(input.symbol)) {
        yield {
          symbol: input.symbol,
          price,
          timestamp: new Date(),
        };
      }
    }),
});
```

Use subscription in components:

```
import { client } from './rpc';

function ChatRoom({ roomId }: { roomId: string }) {
  const messages = client.onMessage.useSubscription(
    { roomId },
    {
      onData: (message) => console.log('New message:', message),
      onError: (error) => console.error('Subscription error:', error),
      onComplete: () => console.log('Subscription completed'),
      retryOnError: true,
      retryDelay: 1000,
    }
  );

  return (
    <div>
      <div>Status: {messages.status}</div>
      <div>Latest: {messages.lastData?.text}</div>
      <ul>
        {messages.data.map((msg, i) => (
          <li key={i}>{msg.text}</li>
        ))}
      </ul>
    </div>
  );
}
```

Setting Up WebSocket Connection

```

import { WebSocketConnection, createUseSubscription } from 'philjs-rpc';

const wsConnection = new WebSocketConnection({
  url: 'ws://localhost:3000/api/rpc',
  reconnect: {
    enabled: true,
    maxAttempts: 10,
    delay: 1000,
    maxDelay: 30000,
    backoffMultiplier: 1.5,
  },
  heartbeatInterval: 30000,
});

// Listen to connection events
wsConnection.on('connected', () => console.log('Connected'));
wsConnection.on('disconnected', () => console.log('Disconnected'));
wsConnection.on('reconnecting', ({ attempt, delay }) => {
  console.log(`Reconnecting (attempt ${attempt}) in ${delay}ms`);
});

await wsConnection.connect();

```

Server-Sent Events (SSE) Fallback

For environments without WebSocket support:

```

import { SSEConnection, createUseSSESubscription } from 'philjs-rpc';

const sseConnection = new SSEConnection({
  url: '/api/rpc/sse',
  reconnect: { enabled: true },
  heartbeatTimeout: 60000,
});

const priceUpdates = createUseSSESubscription(
  sseConnection,
  'onPriceUpdate'
)({
  symbol: 'AAPL'
});

```

Automatic Transport Selection

```

import { createAutoTransport } from 'philjs-rpc';

const { connection, type } = createAutoTransport({
  wsUrl: 'ws://localhost:3000/api/rpc',
  sseUrl: '/api/rpc/sse',
  preferWebSocket: true, // Prefer WebSocket, fallback to SSE
});

console.log(`Using ${type} transport`); // 'websocket' or 'sse'

```

Links for Advanced Routing

Compose request/response transformations:

```

import {
  createHttpLink,
  createWebSocketLink,
  createSplitLink,
  createRetryLink,
  createLoggingLink,
  createLinkChain,
} from 'philjs-rpc';

// HTTP Link for queries/mutations
const httpLink = createHttpLink({
  url: '/api/rpc',
  headers: () => ({
    Authorization: `Bearer ${getToken()}`,
  }),
});

// WebSocket Link for subscriptions
const wsLink = createWebSocketLink({
  url: 'ws://localhost:3000/api/rpc',
});

// Route based on operation type
const splitLink = createSplitLink({
  condition: (op) => op.type === 'subscription',
  true: wsLink,
  false: httpLink,
});

// Compose Links
const link = createLinkChain([
  createLoggingLink({ enabled: true }),
  createRetryLink({ maxAttempts: 3 }),
  splitLink,
]);

```

Subscription Middleware

Protect and control subscriptions:

```

import {
  createSubscriptionAuthMiddleware,
  createSubscriptionRateLimitMiddleware,
  createBackpressureMiddleware,
  composeSubscriptionMiddleware,
} from 'philjs-rpc';

const subscriptionMiddleware = composeSubscriptionMiddleware([
  // Authentication
  createSubscriptionAuthMiddleware({
    isAuthenticated: (ctx) => !!ctx.user,
    hasPermission: (ctx, path) => {
      if (path.includes('admin')) {
        return ctx.user?.role === 'admin';
      }
      return true;
    },
  }),

  // Rate limiting
  createSubscriptionRateLimitMiddleware({
    maxSubscriptionsPerConnection: 10,
    maxSubscriptionsPerUser: 50,
    maxEventsPerSecond: 100,
  }),

  // Backpressure handling
  createBackpressureMiddleware({
    maxBufferSize: 100,
    strategy: 'drop-oldest',
  }),
]);

```

Advanced Subscription Features

Filtering:

```

import { createSubscriptionFilterMiddleware } from 'philjs-rpc';

const filterMiddleware = createSubscriptionFilterMiddleware({
  filter: (ctx, data) => {
    // Only send data relevant to user
    return data.userId === ctx.user?.id;
  },
});

```

Multiplexing:

```

import { createMultiplexingMiddleware } from 'philjs-rpc';

const multiplexingMiddleware = createMultiplexingMiddleware({
  getKey: (input) => `room:${input.roomId}`,
  maxSubscriptionsPerKey: 1000,
});

```

State Persistence:

```

import { createLocalStorageStateManager } from 'philjs-rpc';

const stateManager = createLocalStorageStateManager('app');

// Save/Load subscription state
stateManager.save('last-room', { roomId: 'general' });
const state = stateManager.load('last-room');

```

For complete examples, see the [examples](#) directory.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./client`, `/server`
- Source files: `packages/philjs-rpc/src/index.ts`, `packages/philjs-rpc/src/client.ts`, `packages/philjs-rpc/src/server.ts`

Public API

- Direct exports: `ClientConfig`, `createClient`, `createExpressHandler`, `createFetchHandler`, `createHandler`, `createNetlifyHandler`, `createNodeHandler`, `createVercelHandler`, `getQueryCache`, `invalidateQueries`, `prefetchQuery`
- Re-exported names: `// Client types UseQueryOptions`, `// Error types InferErrors`, `// Error types RPCErrorCode`, `// File upload types FileUpload`, `// HTTP method mapping ProcedureTypeToMethod`, `// Handler types RequestAdapter`, `// Method types HttpMethod`, `// Middleware types InferMiddlewareContext`, `// Middleware types Middleware`, `// Path construction BuildPath`, `// Path types ExtractPathParamNames`, `// Procedure types ProcedureType`, `// Request types TreatyRequestOptions`, `// Request/Response types RPCRequest`, `// Response extraction InferOutput`, `// Router types Router`, `// Schema extraction ExtractBody`, `// Schema types Schema`, `// Subscription types SubscriptionObserver`, `// Type builders BuildTreatyClient`, `// Type inference utilities InferProcedureInput`, `// Utility types PartialBy`, `// WebSocket extraction InferWSMessage`, `// WebSocket types WebSocketOptions`, `APIDefinition`, `APIMetadata`, `AcceptsFiles`, `Assert`, `AssertEquals`, `Awaited`, `BackpressureOptions`, `BatchLinkOptions`, `Brand`, `BuildClientFromRouter`, `ClientProcedure`, `ConnectionLimitOptions`, `DeduplicationLinkOptions`, `DeepPartial`, `Deep Readonly`, `Equals`, `ErrorCode`, `ExtractBodyParams`, `ExtractCookies`, `ExtractFiles`, `ExtractHeaders`, `ExtractInput`, `ExtractPathParams`, `ExtractPaths`, `ExtractQueryParams`, `ExtractResponse`, `GetProcedureInput`, `GetProcedureOutput`, `GetProcedureType`, `HTTPMethod`, `HandlerFn`, `HandlerOptions`, `HasPathParam`, `HttpLinkOptions`, `InferInput`, `InferPaths`, `InferProcedureOutput`, `InferProcedureType`, `InferRouter`, `InferSchemaOutput`, `InferTreatyClient`, `InferWSSend`, `IsInputRequired`, `IsMutation`, `IsNever`, `IsProcedure`, `IsQuery`, `IsSubscription`, `IsUnknown`, `IsWebSocket`, `Keys`, `LAZY_MARKER`, `LazyDeserialized`, `Link`, `LinkFn`, `LoggingLinkOptions`, `MergeContexts`, `MiddlewareFn`, `MiddlewareResult`, `MultiplexingOptions`, `OpenAPIOperation`, `OpenAPISchema`, `Operation`, `OperationResult`, `OptionalInput`, `OptionalKeys`, `Prettify`, `ProcedureBuilder`, `ProcedureContext`, `ProcedureDefinition`, `ProcedureHandler`, `ProcedureOptions`, `RPCBatchRequest`, `RPCBatchResponse`, `RPCError`, `RPCResponse`, `RPC_ERROR_CODES_TO_HTTP`, `RequiredBy`, `RequiredKeys`, `ResponseAdapter`, `RetryLinkOptions`, `RouteMetadata`, `RouterNode`, `RouterPaths`, `SSEConnection`, `SSEConnectionConfig`, `SSEMessage`, `SUPERSON_ENABLED`, `SplitLinkOptions`, `SubscriptionAuthOptions`, `SubscriptionContext`, `SubscriptionEventMap`, `SubscriptionFilterOptions`, `SubscriptionHandler`, `SubscriptionMiddleware`, `SubscriptionRateLimitOptions`, `SubscriptionStateManager`, `SuperJSONChunk`, `SuperJSONHandlerOptions`, `SuperJSONProcedure`, `TreatyConfig`, `TreatyError`, `TreatyMethod`, `TreatyRequestConfig`, `TreatyWebSocket`, `TypeGenerationOptions`, `TypedError`, `Unbrand`, `UnwrapArray`, `UseMutationOptions`, `UseMutationResult`, `UseQueryResult`, `UseSubscriptionOptions`, `UseSubscriptionResult`, `ValidInput`, `Values`, `WebSocketConnection`, `WebSocketConnectionConfig`, `WebSocketHandler`, `WebSocketLinkOptions`, `WebSocketMessage`, `cacheMiddleware`, `composeSubscriptionMiddleware`, `createAPI`, `createAPIWithMiddleware`, `createAuthMiddleware`, `createAutoTransport`, `createBackpressureMiddleware`, `createBatchLink`, `createClientRequestTransformer`, `createClientResponseTransformer`, `createConnectionLimitMiddleware`, `createDeduplicationLink`, `createHttpLink`, `createLazyDeserialized`, `createLinkChain`, `createLocalStorageStateManager`, `createLoggingLink`, `createMemoryStateManager`, `createMiddleware`, `createMultiplexingMiddleware`, `createProcedureBuilder`, `createRetryLink`, `createRouter`, `createSplitLink`, `createStreamingDeserializer`, `createStreamingSerializer`, `createSubscriptionAuthMiddleware`, `createSubscriptionFilterMiddleware`, `createSubscriptionRateLimitMiddleware`, `createSuperJSONMiddleware`, `createTerminatingLink`, `createTreatyClient`, `createUseSSESubscription`, `createUseSubscription`, `createWebSocketLink`, `deserializeBatchRequest`, `deserializeBatchResponse`, `deserializeRequest`, `deserializeResponse`, `executeMiddlewareChain`, `executeProcedure`, `exportRoutesJSON`, `extractAPIMetadata`, `generateOpenAPI`, `generateTypeDefinitions`, `getAllPaths`, `getProcedureAtPath`, `getProcedureCount`, `getRouterPaths`, `getSuperJSONOptions`, `hasSuperJSON`, `isLazyDeserialized`, `isMutation`, `isProcedure`, `isQuery`, `isRouter`, `isSSESupported`, `isSubscription`, `loggerMiddleware`, `mergeRouters`, `permissionMiddleware`, `printAPIRoutes`, `procedure`, `rateLimitMiddleware`, `retryMiddleware`, `serializeBatchRequest`, `serializeBatchResponse`, `serializeRequest`, `serializeResponse`, `timingMiddleware`, `treaty`, `unwrapLazy`, `validateRequest`, `validationMiddleware`, `withSuperJSON`, `withoutSuperJSON`
- Re-exported modules: `./createAPI.js`, `./links.js`, `./middleware.js`, `./procedure.js`, `./sse.js`, `./subscription-middleware.js`, `./subscriptions.js`, `./superjson.js`, `./treaty-server.js`, `./treaty-types.js`, `./treaty.js`, `./types.js`

License

MIT

```

## @philjs/runtime - Type: Node package - Purpose: PhilJS Runtime with Self-Healing capabilities - automatic error recovery, circuit breakers, hot-patching - Version: 0.1.0 -
Location: packages/philjs-runtime - Entry points: ./, packages/philjs-runtime/src/self-healing/index.ts - Keywords: philjs, runtime, self-healing, error-recovery, circuit-breaker, hot-patching, resilience, fault-tolerance

```

@philjs/runtime

PhilJS Runtime with Self-Healing capabilities - automatic error recovery, circuit breakers, hot-patching

Overview

PhilJS Runtime with Self-Healing capabilities - automatic error recovery, circuit breakers, hot-patching

Focus Areas

- `philjs`, `runtime`, `self-healing`, `error-recovery`, `circuit-breaker`, `hot-patching`, `resilience`, `fault-tolerance`

Entry Points

- `.`
- `packages/philjs-runtime/src/self-healing/index.ts`

Quick Start

```
import { Checkpoint, CircuitBreakerState, ErrorContext } from '@philjs/runtime';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- Checkpoint
- CircuitBreakerState
- ErrorContext
- ErrorSeverity
- FailurePrediction
- HealingConfig
- HealingEvent
- HealingEventHandler
- HealingEventType
- HealingResult
- HealingStrategy
- SelfHealingRuntime

Install

```
pnpm add @philjs/runtime
```

Usage

```
import { Checkpoint, CircuitBreakerState, ErrorContext } from '@philjs/runtime';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./self-healing`
- Source files: `packages/philjs-runtime/src/self-healing/index.ts`

Public API

- Direct exports: `Checkpoint, CircuitBreakerState, ErrorContext, ErrorSeverity, FailurePrediction, HealingConfig, HealingEvent, HealingEventHandler, HealingEventType, HealingResult, HealingStrategy, SelfHealingRuntime, createHealingErrorBoundary, getSelfHealingRuntime, initSelfHealing, resetSelfHealing, useSelfHealing, withSelfHealing`
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/scene - Type: Node package - Purpose: Declarative 3D scene graph for PhilJS - React-three-fiber inspired API, animations, particles, GLTF loading - Version: 0.1.0 - Location: packages/philjs-scene - Entry points: packages/philjs-scene/src/index.ts - Keywords: philjs, 3d, scene-graph, webgl, declarative, animation, particles, gltf

@philjs/scene

Declarative 3D scene graph for PhilJS - React-three-fiber inspired API, animations, particles, GLTF loading

Overview

Declarative 3D scene graph for PhilJS - React-three-fiber inspired API, animations, particles, GLTF loading

Focus Areas

- philjs, 3d, scene-graph, webgl, declarative, animation, particles, gltf

Entry Points

- `packages/philjs-scene/src/index.ts`

Quick Start

```
import { // Core classes
  SceneNode, // Declarative API
  createElement, // Hooks
  useScene } from '@philjs/scene';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- // Core classes SceneNode
- // Declarative API createElement
- // Hooks useScene
- // Lights Light
- // Types type Vector3
- AmbientLight
- AnimationClip
- AnimationMixer
- AnimationTrack
- Camera
- CameraProps
- Color

Install

```
pnpm add @philjs/scene
```

Usage

```
import { // Core classes
  SceneNode, // Declarative API
  createElement, // Hooks
  useScene } from '@philjs/scene';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-scene/src/index.ts

Public API

- Direct exports: // Core classes SceneNode, // Declarative API createElement, // Hooks useScene, // Lights Light, // Types type Vector3, AmbientLight, AnimationClip, AnimationMixer, AnimationTrack, Camera, CameraProps, Color, DirectionalLight, GLTFLoader, GLTFResult, Geometry, GeometryProps, HemisphereLight, InstancedMesh, Keyframe, LOD, LightProps, Material, MaterialProps, Matrix4, Mesh, NodeProps, ParticleSystem, ParticleSystemConfig, PointLight, Quaternion, Scene, SceneProps, SpotLight, Transform, buildScene, useAnimation, useCamera, useGLTF, useMesh, useParticles
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/screen-share - Type: Node package - Purpose: Advanced screen sharing for PhilJS - annotations, presenter mode, region selection, recording - Version: 0.1.0 - Location: packages/philjs-screen-share - Entry points: packages/philjs-screen-share/src/index.ts - Keywords: philjs, screen-share, webrtc, annotation, presenter, recording

@philjs/screen-share

Advanced screen sharing for PhilJS - annotations, presenter mode, region selection, recording

Overview

Advanced screen sharing for PhilJS - annotations, presenter mode, region selection, recording

Focus Areas

- philjs, screen-share, webrtc, annotation, presenter, recording

Entry Points

- packages/philjs-screen-share/src/index.ts

Quick Start

```
import { // Core classes
  ScreenShareManager, // Hooks
  useScreenShare, // Types
  type ScreenShareConfig } from '@philjs/screen-share';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- // Core classes ScreenShareManager

- // Hooks useScreenShare
- // Types type ScreenShareConfig
- Annotation
- AnnotationLayer
- AnnotationTool
- CropRegion
- CursorHighlighter
- PresenterMode
- RegionSelector
- UseScreenShareResult
- useAnnotationTools

Install

```
pnpm add @philjs/screen-share
```

Usage

```
import { // Core classes
  ScreenShareManager, // Hooks
  useScreenShare, // Types
  type ScreenShareConfig } from '@philjs/screen-share';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-screen-share/src/index.ts

Public API

- Direct exports: // Core classes ScreenShareManager, // Hooks useScreenShare, // Types type ScreenShareConfig, Annotation, AnnotationLayer, AnnotationTool, CropRegion, CursorHighlighter, PresenterMode, RegionSelector, UseScreenShareResult, useAnnotationTools, usePresenterMode
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/security-scanner - Type: Node package - Purpose: Automated vulnerability detection for PhilJS - static analysis, dependency scanning, runtime monitoring - Version: 0.1.0 - Location: packages/philjs-security-scanner - Entry points: packages/philjs-security-scanner/src/index.ts - Keywords: philjs, security, scanner, vulnerability, xss, csrf, sast, dast

@philjs/security-scanner

Automated vulnerability detection for PhilJS - static analysis, dependency scanning, runtime monitoring

Overview

Automated vulnerability detection for PhilJS - static analysis, dependency scanning, runtime monitoring

Focus Areas

- philjs, security, scanner, vulnerability, xss, csrf, sast, dast

Entry Points

- packages/philjs-security-scanner/src/index.ts

Quick Start

```
import { CodeLocation, DependencyScanner, DependencyVulnerability } from '@philjs/security-scanner';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- CodeLocation
- DependencyScanner
- DependencyVulnerability
- HeadersValidator
- RuntimeAlert
- RuntimeMonitor

- ScanConfig
- ScanResult
- ScanSummary
- SecurityHeaders
- SecurityRule
- SecurityScanner

Install

```
pnpm add @philjs/security-scanner
```

Usage

```
import { CodeLocation, DependencyScanner, DependencyVulnerability } from '@philjs/security-scanner';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-security-scanner/src/index.ts

Public API

- Direct exports: CodeLocation, DependencyScanner, DependencyVulnerability, HeadersValidator, RuntimeAlert, RuntimeMonitor, ScanConfig, ScanResult, ScanSummary, SecurityHeaders, SecurityRule, SecurityScanner, Severity, StaticScanner, Vulnerability, VulnerabilityType, useRuntimeMonitor, useSecurityHeaders, useSecurityScanner
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

```
## @philjs/spatial-audio - Type: Node package - Purpose: 3D spatial audio for PhilJS - HRTF, room acoustics, ambisonics, VR/AR audio sync - Version: 0.1.0 - Location: packages/philjs-spatial-audio - Entry points: packages/philjs-spatial-audio/src/index.ts - Keywords: philjs, spatial-audio, 3d-audio, hrtf, ambisonics, webxr, vr, ar
```

@philjs/spatial-audio

3D spatial audio for PhilJS - HRTF, room acoustics, ambisonics, VR/AR audio sync

Overview

3D spatial audio for PhilJS - HRTF, room acoustics, ambisonics, VR/AR audio sync

Focus Areas

- philjs, spatial-audio, 3d-audio, hrtf, ambisonics, webxr, vr, ar

Entry Points

- packages/philjs-spatial-audio/src/index.ts

Quick Start

```
import { AmbisonicsDecoder, AmbisonicsOptions, AudioEntity } from '@philjs/spatial-audio';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- AmbisonicsDecoder
- AmbisonicsOptions
- AudioEntity
- AudioPath
- AudioScene
- AudioSourceOptions
- MaterialType
- Orientation
- RoomAcousticsOptions
- RoomAcousticsProcessor
- RoomPresets
- SpatialAudioConfig

Install

```
pnpm add @philjs/spatial-audio
```

Usage

```
import { AmbisonicsDecoder, AmbisonicsOptions, AudioEntity } from '@philjs/spatial-audio';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-spatial-audio/src/index.ts

Public API

- Direct exports: AmbisonicsDecoder, AmbisonicsOptions, AudioEntity, AudioPath, AudioScene, AudioSourceOptions, MaterialType, Orientation, RoomAcousticsOptions, RoomAcousticsProcessor, RoomPresets, SpatialAudioConfig, SpatialAudioContext, Spatial AudioSource, Vector3, calculateDistance, crossProduct, lerp, lerpVector, normalizeVector, useAudioListener, useAudioPath, useAudioScene, use AudioSource, useSpatialAudio, useVRAudio
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/sqlite - Type: Node package - Purpose: SQLite WASM for browser with reactive queries and sync engine - Version: 0.1.0 - Location: packages/philjs-sqlite - Entry points: packages/philjs-sqlite/src/index.ts, packages/philjs-sqlite/src/db/index.ts, packages/philjs-sqlite/src/reactive/index.ts, packages/philjs-sqlite/src-sync/index.ts, packages/philjs-sqlite/src/hooks.ts - Keywords: philjs, sqlite, wasm, local-first, database, reactive

@philjs/sqlite

SQLite WASM for browser with reactive queries and sync engine for PhilJS.

Overview

@philjs/sqlite provides a complete SQLite database solution for browser applications using WebAssembly. It includes reactive queries that auto-update when data changes and a sync engine for offline-first applications.

Features

- **SQLite in the Browser:** Full SQL power via WebAssembly
- **Multiple Persistence Modes:** Memory, IndexedDB, or OPFS
- **Reactive Queries:** Auto-updating queries using signals
- **Query Builder:** Type-safe fluent query API
- **Sync Engine:** Offline-first with conflict resolution
- **Migration Support:** Schema versioning and migrations
- **Change Tracking:** Subscribe to table changes

Installation

```
npm install @philjs/sqlite
# or
pnpm add @philjs/sqlite
```

Quick Start

```

import { createMemoryDatabase, createReactiveQuery } from '@philjs/sqlite';

// Create and initialize database
const db = createMemoryDatabase('myapp');
await db.initialize();

// Create tables
db.exec(`
CREATE TABLE users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    email TEXT UNIQUE
)
`);

// Insert data
db.exec('INSERT INTO users (name, email) VALUES (?, ?)', ['John', 'john@example.com']);

// Query data
const users = db.query<{ id: number; name: string; email: string }>('SELECT * FROM users');
console.log(users);

// Create reactive query that auto-updates
const reactiveUsers = createReactiveQuery(db, {
    sql: 'SELECT * FROM users ORDER BY name',
});

// Subscribe to changes
reactiveUsers.subscribe(() => {
    console.log('Users updated:', reactiveUsers.data);
});

// Insert triggers auto-update
db.exec('INSERT INTO users (name, email) VALUES (?, ?)', ['Jane', 'jane@example.com']);

```

Database Creation

Memory Database

```

import { createMemoryDatabase } from '@philjs/sqlite';

const db = createMemoryDatabase('my-memory-db');
await db.initialize();

```

Persistent Database (OPFS)

```

import { createPersistentDatabase } from '@philjs/sqlite';

const db = createPersistentDatabase('my-persistent-db');
await db.initialize();

```

Custom Configuration

```

import { createDatabase, type SQLiteConfig } from '@philjs/sqlite';

const config: SQLiteConfig = {
    dbName: 'custom-db',
    persistenceMode: 'indexeddb', // 'memory' / 'opfs' / 'indexeddb'
    pageSize: 8192,
    cacheSize: 5000,
    walMode: true,
    readOnly: false,
};

const db = createDatabase(config);
await db.initialize();

```

SQL Operations

Execute SQL

```

// Execute without results
db.exec('CREATE TABLE products (id INTEGER PRIMARY KEY, name TEXT, price REAL)');

// Execute with parameters
db.exec('INSERT INTO products (name, price) VALUES (?, ?)', ['Widget', 29.99]);

```

Query Data

```

// Query all results
const products = db.query<Product>('SELECT * FROM products');

// Query with parameters
const expensive = db.query<Product>('SELECT * FROM products WHERE price > ?', [50]);

// Query single result
const product = db.queryOne<Product>('SELECT * FROM products WHERE id = ?', [1]);

```

Run with Change Info

```
// Run with Change Info
const result = db.run('INSERT INTO products (name, price) VALUES (?, ?)', ['Gadget', 49.99]);
console.log('Changes:', result.changes);
console.log('Last ID:', result.lastInsertRowid);
```

Transactions

```
// Synchronous transaction
db.transaction(() => {
  db.exec('INSERT INTO accounts (balance) VALUES (?)', [1000]);
  db.exec('UPDATE accounts SET balance = balance - 100 WHERE id = ?', [1]);
  db.exec('UPDATE accounts SET balance = balance + 100 WHERE id = ?', [2]);
});

// Async transaction
await db.transactionAsync(async () => {
  db.exec('INSERT INTO logs (message) VALUES (?)', ['Started']);
  await someAsyncOperation();
  db.exec('INSERT INTO logs (message) VALUES (?)', ['Completed']);
});
```

Reactive Queries

Basic Reactive Query

```
import { createReactiveQuery } from '@philjs/sqlite';

const query = createReactiveQuery(db, {
  sql: 'SELECT * FROM products WHERE price < ?',
  params: [100],
});

// Access current data
console.log(query.data);
console.log(query.loading);
console.log(query.error);
console.log(query.updatedAt);

// Subscribe to changes
const unsubscribe = query.subscribe(() => {
  console.log('Data updated:', query.data);
});

// Update parameters
query.setParams([50]);

// Manual refresh
query.refresh();

// Cleanup
query.dispose();
```

With Transform

```
interface Product { id: number; name: string; price: number }
interface ProductDisplay { id: number; name: string; displayPrice: string }

const query = createReactiveQuery<ProductDisplay>(db, {
  sql: 'SELECT * FROM products',
  transform: (rows) => rows.map(row => ({
    id: row.id as number,
    name: row.name as string,
    displayPrice: `$$ {(row.price as number).toFixed(2)} `,
  })),
});
```

With Debouncing

```
const query = createReactiveQuery(db, {
  sql: 'SELECT * FROM logs ORDER BY created_at DESC',
  debounce: 100, // Debounce updates by 100ms
});
```

Query Builder

Basic Usage

```
import { query } from '@philjs/sqlite';

const products = query(db)
  .select(['id', 'name', 'price'])
  .from('products')
  .where('category = ?', 'electronics')
  .where('price < ?', 500)
  .orderBy('price', 'ASC')
  .limit(10)
  .execute();
```

Build SQL

```
const { sql, params } = query(db)
  .select('*')
  .from('users')
  .where('role = ?', 'admin')
  .orderBy('name')
  .toSQL();

console.log(sql); // SELECT * FROM users WHERE role = ? ORDER BY name ASC
console.log(params); // [ 'admin' ]
```

Reactive Query Builder

```
const reactiveProducts = query(db)
  .select('*')
  .from('products')
  .where('active = ?', true)
  .orderBy('created_at', 'DESC')
  .limit(20)
  .reactive({ debounce: 50 });
```

Migrations

Define Migrations

```
import { createMigrationManager, defineMigration } from '@philjs/sqlite';

const manager = createMigrationManager(db);

manager.add(defineMigration({
  version: 1,
  name: 'create_users_table',
  up: `
    CREATE TABLE users (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      name TEXT NOT NULL,
      email TEXT UNIQUE,
      created_at TEXT DEFAULT (datetime('now'))
    )
  `,
  down: 'DROP TABLE users',
}));
```

```
manager.add(defineMigration({
  version: 2,
  name: 'add_user_role',
  up: "ALTER TABLE users ADD COLUMN role TEXT DEFAULT 'user'",
  down: 'ALTER TABLE users DROP COLUMN role',
}));
```

Run Migrations

```
// Run all pending migrations
const result = manager.migrate();
if (result.success) {
  console.log('Applied:', result.applied);
}

// Migrate to specific version
manager.migrateTo(1);

// Rollback Last migration
manager.rollback();

// Reset (rollback all)
manager.reset();
```

Check Status

```
const status = manager.getStatus();
console.log('Current version:', status.current);
console.log('Latest version:', status.latest);
console.log('Pending:', status.pending);
console.log('Applied:', status.applied);
```

Sync Engine

Basic Setup

```

import { createSyncEngine } from '@philjs/sqlite';

const syncEngine = createSyncEngine(db, {
  endpoint: 'https://api.example.com-sync',
  tables: ['users', 'posts', 'comments'],
  conflictStrategy: 'last-write-wins',
  getAuthToken: async () => localStorage.getItem('auth_token'),
});

await syncEngine.initialize();

```

Manual Sync

```

const result = await syncEngine.sync();
console.log('Success:', result.success);
console.log('Pushed:', result.pushed);
console.log('Pulled:', result.pulled);
console.log('Conflicts:', result.conflicts);
console.log('Duration:', result.duration, 'ms');

```

Auto-Sync

```

const syncEngine = createSyncEngine(db, {
  endpoint: 'https://api.example.com-sync',
  tables: ['users'],
  conflictStrategy: 'server-wins',
  syncInterval: 30000, // Sync every 30 seconds
  onSyncComplete: (result) => {
    console.log('Sync completed:', result);
  },
});

// Stop auto-sync
syncEngine.stopSyncInterval();

```

Conflict Resolution Strategies

```

// Client always wins
{ conflictStrategy: 'client-wins' }

// Server always wins
{ conflictStrategy: 'server-wins' }

// Most recent write wins
{ conflictStrategy: 'last-write-wins' }

// Merge fields (remote values take precedence)
{ conflictStrategy: 'merge' }

// Manual resolution
{
  conflictStrategy: 'manual',
  onConflict: async (conflict) => {
    console.log('Local:', conflict.localData);
    console.log('Remote:', conflict.remoteData);

    // Return resolution
    return { action: 'use-merged', mergedData: { ...conflict.localData, ...conflict.remoteData } };
  }
}

```

Sync Status

```

const status = syncEngine.getStatus();
console.log('Pending changes:', status.pendingChanges);
console.log('Last sync:', new Date(status.lastSync));
console.log('Currently syncing:', status.syncing);

```

Change Listeners

Listen to Specific Table

```

const unsubscribe = db.onTableChange('users', (event) => {
  console.log('Table:', event.table);
  console.log('Operation:', event.operation); // 'INSERT' | 'UPDATE' | 'DELETE'
  console.log('Row ID:', event.rowid);
});

// Stop Listening
unsubscribe();

```

Listen to All Changes

```

const unsubscribe = db.onAnyChange((event) => {
  console.log('Change detected:', event);
});

```

Hooks

useSQLite

```
import { useSQLite } from '@philjs/sqlite';

const { db, ready, error, exec, query, queryOne, run, transaction } = useSQLite({
  dbName: 'myapp',
  persistenceMode: 'opfs',
});

if (ready) {
  const users = query<User>('SELECT * FROM users');
}
```

useQuery

```
import { useQuery } from '@philjs/sqlite';

const { data, loading, error, refetch, setParams } = useQuery<User>(
  'SELECT * FROM users WHERE role = ?',
  ['admin'],
  {
    dbName: 'myapp',
    debounce: 100,
  }
);

// Update query parameters
setParams(['user']);

// Manual refresh
refetch();
```

useSync

```
import { useSync } from '@philjs/sqlite';

const { status, sync, stop } = useSync({
  endpoint: 'https://api.example.com-sync',
  tables: ['users', 'posts'],
  conflictStrategy: 'last-write-wins',
  syncInterval: 60000,
});

// Manual sync
await sync();

// Stop auto-sync
stop();
```

useKVStore

```
import { useKVStore } from '@philjs/sqlite';

const kv = useKVStore('settings');

// Set value
kv.set('theme', 'dark');
kv.set('user', { name: 'John', preferences: { notifications: true } });

// Get value
const theme = kv.get<string>('theme');
const user = kv.get<User>('user');

// Delete
kv.delete('theme');

// Get all keys
const keys = kv.keys();

// Clear all
kv.clear();
```

Database Management

```
import { getDatabase, closeDatabase, closeAllDatabases } from '@philjs/sqlite';

// Get database by name
const db = getDatabase('myapp');

// Close specific database
closeDatabase('myapp');

// Close all databases
closeAllDatabases();
```

TypeScript Support

Full type exports:

```
import type {
  // Database types
  SQLiteDB,
  SQLiteConfig,
  PersistenceMode,
  Row,
  TableChangeEvent,
  TableChangeCallback,

  // Migration types
  Migration,
  MigrationResult,

  // Reactive types
  ReactiveQuery,
  ReactiveQueryOptions,
  ReactiveQueryState,

  // Sync types
  SQLiteSyncEngine,
  SyncConfig,
  SyncResult,
  SyncConflict,
  ConflictResolution,
  ChangeRecord,

  // Hook types
  SQLiteContext,
  ReactiveQueryResult,
  SyncHookResult,
} from '@philjs/sqlite';
```

Subpath Exports

```
// Database only
import { SQLiteDB, createDatabase } from '@philjs/sqlite/db';

// Reactive queries only
import { ReactiveQuery, QueryBuilder, query } from '@philjs/sqlite/reactive';

// Sync engine only
import { SQLiteSyncEngine, createSyncEngine } from '@philjs/sqlite-sync';

// Hooks only
import { useSQLite, useQuery, useSync, useKVStore } from '@philjs/sqlite/hooks';
```

Browser Compatibility

- **Memory mode:** All modern browsers
- **IndexedDB mode:** All modern browsers
- **OPFS mode:** Chrome 102+, Edge 102+, Firefox 111+, Safari 15.2+

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./db`, `./reactive`, `./sync`, `./hooks`
- Source files: `packages/philjs-sqlite/src/index.ts`, `packages/philjs-sqlite/src/db/index.ts`, `packages/philjs-sqlite/src/reactive/index.ts`, `packages/philjs-sqlite/src/sync/index.ts`, `packages/philjs-sqlite/src/hooks.ts`

Public API

- Direct exports: `ReactiveQueryResult`, `SQLiteContext`, `SyncHookResult`, `closeAllDatabases`, `closeDatabase`, `getDatabase`, `useKVStore`, `useQuery`, `useSQLite`, `useSync`
- Re-exported names: `ChangeRecord`, `ConflictResolution`, `Migration`, `MigrationManager`, `MigrationResult`, `PersistenceMode`, `PreparedStatement`, `QueryBuilder`, `ReactiveQuery`, `ReactiveQueryOptions`, `ReactiveQueryResult`, `ReactiveQueryState`, `Row`, `SQLiteConfig`, `SQLiteContext`, `SQLiteDB`, `SQLiteSyncEngine`, `SyncConfig`, `SyncConflict`, `SyncHookResult`, `SyncResult`, `TableChangeEvent`, `TableChangeCallback`, `closeAllDatabases`, `closeDatabase`, `createDatabase`, `createIndexedDBDatabase`, `createMemoryDatabase`, `createMigrationManager`, `createPersistentDatabase`, `createReactiveQuery`, `createSyncEngine`, `defineMigration`, `getBestPersistenceMode`, `getDatabase`, `isIndexedDBSupported`, `migrationsFromSQL`, `query`, `useKVStore`, `useQuery`, `useSQLite`, `useSync`
- Re-exported modules: `./db/migrations.js`, `./db/sqlite-wasm.js`, `./hooks.js`, `./migrations.js`, `./reactive-query.js`, `./reactive/reactive-query.js`, `./sqlite-wasm.js`, `./sync-engine.js`, `./sync-sync-engine.js`

License

MIT

@philjs/ssr - Type: Node package - Purpose: SSR streaming, loaders, actions, and resumability for PhilJS - Version: 0.1.0 - Location: packages/philjs-ssr - Entry points: packages/philjs-ssr/src/index.ts - Keywords: philjs, server-side-rendering, streaming - Book coverage: [./ssr/overview.md](#)

philjs-ssr

Server-side rendering with streaming, loaders, actions, and resumability for PhilJS.

Requirements

- **Node.js 24** or higher

- **TypeScript 6** or higher
- **ESM only** - CommonJS is not supported

Installation

```
pnpm add philjs-ssr
```

Usage

Basic SSR Request Handler

The simplest way to render PhilJS apps on the server:

```
import { handleRequest } from 'philjs-ssr';
import { createRouteMatcher } from 'philjs-router';

// Define your routes
const routes = [
  {
    path: '/',
    component: HomePage,
    loader: async () => {
      return { message: 'Hello from SSR!' };
    },
  },
  {
    path: '/products/:id',
    component: ProductPage,
    loader: async ({ params }) => {
      const res = await fetch(`/api/products/${params.id}`);
      return res.json();
    },
  },
];

// Create route matcher
const match = createRouteMatcher(routes);

// Handle incoming requests
export default async function handler(request: Request) {
  return handleRequest(request, { match });
}
```

Streaming SSR with Suspense

Stream HTML to the client as it's rendered, with progressive enhancement:

```
import { renderToStreamingResponse, Suspense } from 'philjs-ssr';

function App() {
  return (
    <div>
      <h1>My App</h1>
      <Suspense fallback=<div>Loading...</div>>
        <AsyncContent />
      </Suspense>
    </div>
  );
}

// In your server handler
export default async function handler(request: Request) {
  const stream = await renderToStreamingResponse(<App />, {
    onShellReady: () => console.log('Shell sent to client'),
    onComplete: () => console.log('Stream complete'),
  });

  return new Response(stream, {
    headers: { 'Content-Type': 'text/html; charset=utf-8' },
  });
}
```

Data Loaders

Fetch data before rendering components:

```

import type { Loader } from 'philjs-ssr';

// Type-safe Loader
export const loader: Loader<{ user: User }> = async ({ request, params }) => {
  const userId = params.id;
  const user = await db.users.findById(userId);

  if (!user) {
    return { ok: false, error: 'User not found' };
  }

  return { ok: true, value: { user } };
};

// Use in component
function UserProfile({ data, error }: RouteComponentProps) {
  if (error) {
    return <div>Error: {error}</div>;
  }

  return <div>Welcome, {data.user.name}!</div>;
}

```

Form Actions

Handle form submissions server-side:

```

import type { Action } from 'philjs-ssr';

export const action: Action = async ({ formData, request }) => {
  const title = formData.get('title');
  const content = formData.get('content');

  // Validate
  if (!title || !content) {
    return { error: 'Title and content are required' };
  }

  // Save to database
  const post = await db.posts.create({ title, content });

  // Redirect to new post
  return { redirect: `/posts/${post.id}` };
};

// Use in component
function CreatePost() {
  return (
    <form method="POST">
      <input name="title" placeholder="Title" />
      <textarea name="content" placeholder="Content" />
      <button type="submit">Create Post</button>
    </form>
  );
}

```

Partial Prerendering (PPR)

PPR combines the best of static site generation (fast) and server-side rendering (fresh data). Inspired by Next.js 14's Partial Prerendering feature.

```

import { dynamic, Suspense, renderToStaticShell, streamPPRResponse } from 'philjs-ssr';

// Mark dynamic boundaries - these are streamed at request time
function Page() {
  return (
    <div>
      {/* Static - prerendered at build time */}
      <Header />
      <Sidebar />

      {/* Dynamic - streamed at request time */}
      <Suspense fallback={<ProfileSkeleton />}>
        <UserProfile /> {/* Fetches fresh user data */}
      </Suspense>

      {/* Explicitly dynamic content */}
      <dynamic fallback={<CommentsSkeleton />} priority={8}>
        <RealtimeComments />
      </dynamic>
    </div>
  );
}

// Route configuration for PPR
export const config = {
  ppr: true, // Enable PPR for this route
};

```

At build time, static content is prerendered and dynamic boundaries leave placeholder holes:

```
import { buildPPR, pprVitePlugin } from 'philjs-ssr';

// Build-time: Generate static shells
await buildPPR({
  outDir: './dist/ppr',
  routes: [
    {
      path: '/',
      component: HomePage,
      config: { ppr: true },
    },
    {
      path: '/blog/[slug]',
      component: BlogPost,
      config: { ppr: true },
      getStaticPaths: async () => ['/blog/hello', '/blog/world'],
    },
  ],
  renderFn: async (path) => renderComponent(path),
});

// Or use the Vite plugin
export default defineConfig({
  plugins: [pprVitePlugin({ routes })],
});
```

Streaming Dynamic Content

At request time, the static shell is served instantly while dynamic content streams in:

```
import { streamPPRResponse, loadStaticShell } from 'philjs-ssr';

export async function handleRequest(request: Request) {
  const url = new URL(request.url);

  // Load pre-built static shell
  const shell = await loadStaticShell('./dist/ppr', url.pathname);

  if (!shell) {
    return new Response('Not Found', { status: 404 });
  }

  // Stream response with PPR
  return streamPPRResponse(shell, <Page />, request, {
    onShellSent: () => console.log('Static shell sent'),
    onBoundaryResolved: (id) => console.log(`Dynamic boundary ${id} resolved`),
    onComplete: () => console.log('All content streamed'),
    timeout: 10000, // 10s timeout for dynamic content
  });
}
```

Dynamic Boundary Helpers

Various helpers for common dynamic content patterns:

```

import {
  dynamic,
  dynamicPriority,
  dynamicDeferred,
  dynamicWithDependencies,
  dynamicForUser,
  serverOnly,
  makeDynamic,
} from 'philjs-ssr';

// High-priority dynamic content (rendered first)
<dynamicPriority fallback=<Skeleton />>
  <CriticalUserData />
</dynamicPriority>

// Low-priority/deferred content (rendered last)
<dynamicDeferred fallback=<Skeleton />>
  <Analytics />
</dynamicDeferred>

// Dynamic with cache dependencies
const UserCart = dynamicWithDependencies(['user:session', 'cart:items'], {
  children: <CartContents />,
  fallback: <CartSkeleton />,
});

// User-specific dynamic content
<dynamicForUser fallback=<Skeleton />>
  <PersonalizedRecommendations />
</dynamicForUser>

// Server-only content (never hydrated)
<serverOnly fallback=<Skeleton />>
  <SensitiveData />
</serverOnly>

// Wrap existing component as dynamic
const DynamicProfile = makeDynamic(UserProfile, { priority: 8 });

```

Edge Caching

PPR integrates with edge caching for optimal performance:

```

import {
  LRUPPRCache,
  RedisPPRCache,
  EdgeCacheController,
  CacheTagManager,
  generateCacheHeaders,
} from 'philjs-ssr';

// LRU cache for development/single server
const cache = new LRUPPRCache({
  maxSize: 100,
  maxAge: 3600000, // 1 hour
});

// Redis cache for production/distributed
const redisCache = new RedisPPRCache(redisClient, {
  keyPrefix: 'ppr:',
  ttl: 3600,
});

// Edge cache controller with stale-while-revalidate
const edgeCache = new EdgeCacheController({
  strategy: 'stale-while-revalidate',
  cache,
  staleTTL: 60,
});

// Get with automatic revalidation
const { shell, stale } = await edgeCache.get('/blog/post-1', async () => {
  return await renderToStaticShell(<BlogPost slug="post-1" />, '/blog/post-1');
});

// Cache tags for invalidation
const tagManager = new CacheTagManager();
tagManager.tag('/blog/post-1', ['blog', 'author:john']);
await tagManager.invalidateTag('author:john', cache); // Invalidates all John's posts

// Generate CDN cache headers
const headers = generateCacheHeaders(shell, {
  strategy: 'stale-while-revalidate',
  maxAge: 3600,
  staleWhileRevalidate: 60,
});

```

Static Site Generation (SSG)

Pre-render pages at build time:

```
import { buildStaticSite, ssg, isr, ssr } from 'philjs-ssr';

// Configure routes
const config = {
  routes: [
    { path: '/', mode: ssg() }, // Static generation
    { path: '/blog/:slug', mode: isr(3600) }, // Incremental static regeneration (1 hour)
    { path: '/dashboard', mode: ssr() }, // Server-side rendering
  ],
};

// Build static site
await buildStaticSite({
  routes: config.routes,
  outDir: './dist',
  baseUrl: 'https://example.com',
});
```

Rate Limiting

Protect your API endpoints:

```
import { rateLimit, apiRateLimit, authRateLimit } from 'philjs-ssr';

// Basic rate limiting
const limiter = rateLimit({
  windowMs: 60000, // 1 minute
  maxRequests: 100,
});

// Apply to handler
export default async function handler(request: Request) {
  const rateLimitResult = await limiter.check(request);

  if (!rateLimitResult.allowed) {
    return new Response('Too many requests', {
      status: 429,
      headers: {
        'Retry-After': String(rateLimitResult.retryAfter),
      },
    });
  }

  return handleRequest(request, { match });
}

// Pre-configured limiters
const apilimiter = apiRateLimit(); // 1000 req/hour
const authlimiter = authRateLimit(); // 5 req/15min
```

CSRF Protection

Protect forms from cross-site request forgery:

```
import { csrfProtection, generateCSRFToken, csrfField } from 'philjs-ssr';

// Server-side: Validate CSRF token
export const action: Action = async ({ request, formData }) => {
  const isValid = await csrfProtection(request, formData);

  if (!isValid) {
    return new Response('Invalid CSRF token', { status: 403 });
  }

  // Process form...
};

// Client-side: Include CSRF token in forms
function MyForm() {
  const token = generateCSRFToken();

  return (
    <form method="POST">
      {csrfField(token)}
      <input name="email" />
      <button type="submit">Submit</button>
    </form>
  );
}
```

Platform Adapters

Deploy to any platform:

```

import {
  createFetchHandler,
  createNodeHttpHandler,
  createExpressMiddleware,
  createViteMiddleware,
  createWorkerHandler,
} from 'philjs-ssr';

// Cloudflare Workers / Vercel Edge
export default createFetchHandler({ match });

// Node.js HTTP
const server = http.createServer(createNodeHttpHandler({ match }));

// Express.js
app.use(createExpressMiddleware({ match }));

// Vite dev server
export default defineConfig({
  plugins: [createViteMiddleware({ match })],
});

```

API

Request Handling

- handleRequest(request, options) - Handle SSR request with route matching and data loading
- renderToStreamingResponse(vnode, options) - Render to streaming HTML response

Components

- <Suspense fallback={...}> - Lazy loading boundary with fallback UI

Data Loading

- Loader<T> - Type for route data loaders
- Action<T> - Type for form action handlers

Static Generation

- buildStaticSite(config) - Pre-render entire site to static HTML
- ssg() - Static site generation mode
- isr(revalidate) - Incremental static regeneration with revalidation interval
- ssr() - Server-side rendering mode
- csr() - Client-side rendering mode
- configureRoute(path, mode) - Configure rendering mode for a route
- handleRevalidation(request, cache) - Handle ISR revalidation requests
- createRenderingMiddleware(config) - Create middleware for mixed rendering modes

Partial Prerendering (PPR)

- dynamic(props) - Mark content for dynamic rendering
- dynamicPriority(props) - High-priority dynamic content
- dynamicDeferred(props) - Low-priority dynamic content
- dynamicWithDependencies(deps, props) - Dynamic with cache dependencies
- dynamicForUser(props) - User-specific dynamic content
- serverOnly(props) - Server-only content (never hydrated)
- makeDynamic(component, options) - Wrap component as dynamic
- isDynamic(value) - Check if value is a dynamic component
- renderToStaticShell(vnode, path, config) - Render static shell at build time
- generatePPRResponse(shell, vnode, request, options) - Generate streaming PPR response
- streamPPRResponse(shell, vnode, request, options) - Stream PPR response
- buildPPR(config) - Build PPR static shells for all routes
- loadStaticShell(outDir, path) - Load pre-built static shell
- pprVitePlugin(config) - Vite plugin for PPR builds

PPR Caching

- LRUPPRCache - LRU cache for static shells
- RedisPPRCache - Redis-based distributed cache
- MemoryPPRCache - Simple in-memory cache
- FileSystemPPRCache - File-system based cache
- EdgeCacheController - Controller for edge caching strategies
- CacheTagManager - Manage cache invalidation with tags
- generateCacheHeaders(shell, options) - Generate CDN cache headers
- parseConditionalRequest(request) - Parse conditional request headers
- shouldReturn304(shell, conditional) - Check if 304 should be returned
- create304Response(shell) - Create 304 Not Modified response

Security

- `csrfProtection(request, formData)` - Validate CSRF tokens
- `generateCSRFToken()` - Generate a new CSRF token
- `csrfField(token)` - Render CSRF input field
- `extractCSRFToken(request)` - Extract token from request

Rate Limiting

- `RateLimiter` - Configurable rate limiter class
- `rateLimit(config)` - Create custom rate limiter
- `apiRateLimit()` - Pre-configured API rate limiter (1000/hour)
- `authRateLimit()` - Pre-configured auth rate limiter (5/15min)
- `apiKeyRateLimit()` - Rate limit by API key
- `userRateLimit()` - Rate limit by user ID
- `MemoryRateLimitStore` - In-memory rate limit storage
- `RedisRateLimitStore` - Redis-backed rate limit storage
- `SlidingWindowRateLimiter` - Sliding window algorithm
- `AdaptiveRateLimiter` - Adaptive rate limiting based on load

Adapters

- `createFetchHandler(options)` - Standard Fetch API handler
- `createNodeHttpHandler(options)` - Node.js HTTP handler
- `createExpressMiddleware(options)` - Express.js middleware
- `createViteMiddleware(options)` - Vite dev server middleware
- `createWorkerHandler(options)` - Cloudflare Workers handler

Hints & Optimization

- `sendEarlyHints(response, resources)` - Send 103 Early Hints for faster loading
- `generatePreloadLinks(resources)` - Generate Link headers for preloading

Examples

See SSR in action in these example apps:

- Demo App - Full-featured demo with SSR, routing, and islands

Development

```
# Build the package
pnpm build

# Run tests
pnpm test

# Type checking
pnpm tpyecheck
```

Features

- **Partial Prerendering (PPR)** - Static shells with dynamic streaming (Next.js 14-inspired)
- **Streaming SSR** - Progressive HTML streaming with Suspense boundaries
- **Data loaders** - Type-safe data fetching before rendering
- **Form actions** - Server-side form handling with redirects
- **Static generation** - SSG, ISR, and mixed rendering modes
- **Edge caching** - LRU, Redis, and CDN integration for PPR shells
- **Resumability** - Hydrate without re-executing server logic
- **Rate limiting** - Built-in protection against abuse
- **CSRF protection** - Secure form submissions
- **Platform adapters** - Deploy to any JavaScript runtime
- **Security headers** - Automatic security best practices
- **Early hints** - 103 Early Hints for faster page loads
- **Result types** - Rust-style error handling with Ok/Err

Request Context

All loaders and actions receive a context object:

```
type RequestContext = {
  request: Request;           // Original request
  url: URL;                  // Parsed URL
  method: string;             // HTTP method
  headers: Headers;           // Request headers
  params: Record<string, string>; // Route parameters
  formData?: FormData;         // Form data (POST only)
};
```

Rendering Modes

PhilJS SSR supports multiple rendering strategies:

- **PPR** - Partial Prerendering (static shell + dynamic streaming) - **NEW!**

- **SSG** - Static Site Generation (pre-render at build time)
- **ISR** - Incremental Static Regeneration (refresh static pages periodically)
- **SSR** - Server-Side Rendering (render on each request)
- **CSR** - Client-Side Rendering (hydrate on the client)

Mix and match modes per route for optimal performance.

PPR vs Other Modes

Mode	Initial Load	Data Freshness	Use Case
PPR	Instant (static shell)	Fresh (streamed)	Best of both worlds
SSG	Instant	Stale	Marketing pages, blogs
ISR	Instant	Periodically fresh	E-commerce products
SSR	Slower	Always fresh	Dashboards, admin
CSR	Slower	Fresh	SPAs, highly interactive

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philijs-ssr/src/index.ts

Public API

- Direct exports: (none detected)
- Re-exported names: \$\$, closure, //AppTypesResumableAppOptions, //ClosureSerializationserializeClosure, //ComponentBoundariesboundary, //DevelopmentToolsgetResumabilityStats, //EventListe - Lazy, - loadableReferencesqrl, //QRQLTypesQRQL, //ResumableAppWrappercreateResumableApp, //SSRIntegrationinjectResumableState, //StateSerializationresumable, //StateTypes - loadable, MemoryPPRCache, MemoryRateLimitStore, PPRBuildConfig, PPRBuildError, PPRBuildResult, PPRBuilder, PPRCache, PPRConfig, PPRContext, PPRManifest, PPRRouteEntry, PPRStreamController, PPRStreamOptions, PPSuspenseProps, PPR_FALLBACK_END, PPR_FALLBACK_START, PPR_PLACEHOLDER_END, PPR_PLACEHOLDER_START, PPR_VERSION, PhilSServerOptions, RateLimitConfig, RateLimitInfo, RateLimitStore, RateLimiter, RedisSRCache, RedisPPRCache, RedisRateLimitStore, RenderMode, RenderOptions, RenderToStreamOptions, RequestContext, RequestTimeData, ResumabilityOptions, ResumableApp, ResumableContext, ResumableListener, RevalidationOptions, RouteConfig, RouteModule, SSRSuperJSONOptions, SUPERPERSON_LOADER, ShellAssets, SlidingWindowRateLimiter, StaticGenerator, StaticPage, StaticShell, StreamContext, Suspense, SuspenseBoundary, TimedChunk, apiKeyRateLimit, apiRateLimit, authRateLimit, autoHydrateIslands, buildPPr, buildPPrRoute, buildStaticSite, clearIslands, clearSerializedState, configureRoute, create304Response, createBufferedStream, createCompressionStream, createDynamic, createExpressMiddleware, createFetchHandler, createFilterStream, createLoaderDataAccessor, createLoaderDataSerializer, createMultipleStream, createNodeHttpHandler, createPPRContext, createPPRDevServer, createPPRStream, createRateLimitedStream, createRenderingMiddleware, createStreamingLoaderSerializer, createThroughputMeasurer, createTimingStream, createViteMiddleware, createWorkerHandler, csr, csrfField, csrfProtection, deserializeClosureVars, deserializeLoaderData, deserializeState, dynamic, dynamicDeferred, dynamicForUser, dynamicIf, dynamicPriority, dynamicWithDependencies, dynamicWithRevalidation, enableResumability, extractBoundaryId, extractCSRFToken, extractHydrationData, extractResumableState, generateCSRFToken, generateCacheHeaders, generateHydrationRestoreScript, generateHydrationScript, generatePPRResponse, getBoundary, getDynamicBoundaryId, getIslandStatus, getSuperJSONLoaderOptions, handleRequest, handleRevalidation, hasResumableState, hasResumed, hasSuperJSONLoader, hashContent, hydrateAllIslands, hydrateIslands, hydrateIslandOnIdle, hydrateIslandOnInteraction, hydrateIslandOnVisible, injectDynamicContent, injectLoaderData, isDynamic, isQRL, isr, loadPPRManifest, loadStaticShell, logResumabilityInfo, makeDynamic, mergeStreams, nodeStreamToWebStream, onResume, parseConditionalRequest, pipeWebStreamToNode, pprVitePlugin, preloadIsland, qrlChunk, qrlRegistry, rateLimit, registerDynamicBoundary, registerIsland, renderAllDynamicContent, renderDynamicContent, renderToStaticShell, renderToStream, renderToStreamingResponse, resolveQRL, resumableComputed, resumeComputed, resumeContext, resumeFromState, resumeListeners, serializeBoundaries, serializeContext, serializeListeners, serializeLoaderData, serializeState, serverOnly, shouldReturn304, ssg, ssr, streamPPRResponse, superJSONAction, superJSONLoader, teeStream, useResumable, userRateLimit, webStreamToNodeStream, wrapActionWithSuperJSON, wrapLoaderWithSuperJSON
- Re-exported modules: ./adapters.js, ./csrf.js, ./dynamic.js, ./hints.js, ./hydrate-island.js, ./loader.js, ./ppr-build.js, ./ppr-cache.js, ./ppr-streaming.js, ./ppr-types.js, ./ppr.js, ./rate-limit.js, ./render-to-stream.js, ./request-handler.js, ./resumability.js, ./resume.js, ./security.js, ./static-generation.js, ./stream-adapters.js, ./stream.js, ./streaming.js, ./superjson.js, ./types.js

License

MIT

```
## @philijs/storage - Type: Node package - Purpose: Cloud storage abstraction layer for PhilJS with S3, GCS, Azure, and local providers - Version: 0.1.0 - Location: packages/philijs-storage - Entry points: packages/philijs-storage/src/index.ts, ./s3, ./gcs, ./azure, ./local, ./memory, packages/philijs-storage/src/hooks.ts, ./utils - Keywords: philijs, storage, s3, gcs, azure, blob, file-upload, cloud-storage
```

@philijs/storage

File storage abstractions for PhilJS applications. Unified API for local filesystem, S3, Google Cloud Storage, and other storage providers.

Requirements

- **Node.js 24** or higher
- **TypeScript 6** or higher
- **ESM only** - CommonJS is not supported

Installation

```
pnpm add @philijs/storage
```

Basic Usage

```

import { createStorage, StorageProvider } from '@philjs/storage';

const storage = createStorage({
  provider: 's3',
  bucket: 'my-bucket',
  region: 'us-east-1',
});

// Upload file
await storage.put('uploads/image.png', fileBuffer, {
  contentType: 'image/png',
  public: true,
});

// Download file
const file = await storage.get('uploads/image.png');

// Get public URL
const url = await storage.getUrl('uploads/image.png');

// Delete file
await storage.delete('uploads/image.png');

```

Features

- **Multiple Providers** - S3, GCS, Azure Blob, local filesystem
- **Unified API** - Same interface across all providers
- **Streaming** - Stream large files efficiently
- **Signed URLs** - Generate temporary access URLs
- **File Metadata** - Store and retrieve file metadata
- **Directory Operations** - List, copy, move directories
- **Image Processing** - Resize, crop, optimize images
- **CDN Integration** - CloudFront, Cloudflare CDN support
- **Chunked Uploads** - Resume interrupted uploads
- **Access Control** - Fine-grained file permissions
- **Versioning** - File version management
- **Encryption** - Server-side and client-side encryption

Providers

Provider	Configuration
AWS S3	provider: 's3'
Google Cloud	provider: 'gcs'
Azure Blob	provider: 'azure'
Local	provider: 'local'
Memory	provider: 'memory'
Cloudflare R2	provider: 'r2'

React Components

```

import { FileUpload, fileList, ImagePreview } from '@philjs/storage/react';

<FileUpload
  storage={storage}
  path="uploads/"
  onUpload={(file) => console.log('Uploaded:', file)}
/>

<fileList storage={storage} path="uploads/" />

```

Hooks

Hook	Description
useStorage	Access storage instance
useUpload	File upload with progress
useFileList	List files in path
useSignedUrl	Generate signed URLs

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./s3`, `./gcs`, `./azure`, `./local`, `./memory`, `./hooks`, `./utils`
- Source files: `packages/philjs-storage/src/index.ts`, `packages/philjs-storage/src/hooks.ts`

Public API

- Direct exports: AzureConfig, CopyOptions, DownloadOptions, DownloadProgress, DownloadState, FileListState, GCSConfig, ListOptions, ListResult, LocalConfig, MemoryConfig, MoveOptions, S3Config, SignedUrlOptions, StorageConfig, StorageFile, StorageProviderType, UploadOptions, UploadProgress, UploadState, createStorageClient, useDownload, useFileList, useUpload
- Re-exported names: AzureStorageClient, GCSStorageClient, LocalStorageClient, MemoryStorageClient, ResizeOptions, S3StorageClient, StreamingUploadOptions, bufferToStream, createStreamingUpload, detectMimeType, getMimeTypeFromExtension, resizeImage, streamToBuffer, useDownload, useFileList, useUpload
- Re-exported modules: ./hooks.js, ./providers/azure.js, ./providers/gcs.js, ./providers/local.js, ./providers/memory.js, ./providers/s3.js, ./utils/mime.js, ./utils/resize.js, ./utils/stream.js

License

MIT

```
## @philjs/storybook - Type: Node package - Purpose: Storybook integration for PhilJS - component development and testing - Version: 0.1.0 - Location: packages/philjs-storybook - Entry points: packages/philjs-storybook/src/index.ts, packages/philjs-storybook/src/preset.ts, ./addons/signal-inspector, ./addons/route-tester, ./addons/theme-switcher, ./addons/viewport, packages/philjs-storybook/src/decorators/index.ts, packages/philjs-storybook/src/mocks/index.ts - Keywords: philjs, storybook, component-development, testing, ui
```

PhilJS Storybook

Official Storybook integration for PhilJS - develop, test, and document your components in isolation.

Features

- **PhilJS Renderer:** Seamlessly render PhilJS components in Storybook
- **Signal Inspector:** Debug and manipulate signals in real-time
- **Route Tester:** Test route components with mock data
- **Theme Switcher:** Toggle between light/dark themes
- **Viewport Helper:** Test responsive designs
- **Story Helpers:** Type-safe story creation utilities
- **Mocking Utilities:** Mock signals, routers, and APIs
- **CLI Integration:** Generate stories from the command line

Installation

```
# Install PhilJS Storybook
npm add -D philjs-storybook storybook

# Initialize Storybook
philjs storybook init
```

Quick Start

1. Initialize Storybook

```
philjs storybook init
```

This will create a `.storybook` directory with the necessary configuration files.

2. Start Storybook

```
philjs storybook dev
```

or use the npm script:

```
npm run storybook
```

3. Create Your First Story

Generate a story for a component:

```
philjs storybook generate Button
```

Or create one manually:

```
// Button.stories.tsx
import type { Meta, StoryObj } from 'philjs-storybook';
import { Button } from './Button';

const meta: Meta<typeof Button> = {
  title: 'Components/Button',
  component: Button,
  tags: ['autodocs'],
};

export default meta;
type Story = StoryObj<typeof Button>;

export const Primary: Story = {
  args: {
    variant: 'primary',
    children: 'Click me',
  },
};
```

Story Examples

Component Story

```

import type { Meta, StoryObj } from 'philjs-storybook';
import { Card } from './Card';

const meta: Meta<typeof Card> = {
  title: 'Components/Card',
  component: Card,
  tags: ['autodocs'],
  argTypes: {
    variant: {
      control: 'select',
      options: ['default', 'outlined', 'elevated'],
    },
  },
};

export default meta;
type Story = StoryObj<typeof Card>

export const Default: Story = {
  args: {
    title: 'Card Title',
    content: 'Card content goes here',
  },
};

```

Route Story

```

import type { Meta, StoryObj } from 'philjs-storybook';
import { UserProfile } from './UserProfile';
import { withRouter } from 'philjs-storybook/decorators';
import { createMockLoader } from 'philjs-storybook/mocks';

const meta: Meta<typeof UserProfile> = {
  title: 'Routes/UserProfile',
  component: UserProfile,
  decorators: [withRouter],
  parameters: {
    router: {
      pathname: '/users/[id]',
      params: { id: '123' },
    },
  },
};

export default meta;
type Story = StoryObj<typeof UserProfile>

export const Default: Story = {};

export const Loading: Story = {
  render: () => {
    const loader = createMockLoader(async () => {
      await new Promise(resolve => setTimeout(resolve, 2000));
      return { name: 'John Doe', email: 'john@example.com' };
    });
  },
  return <UserProfile loader={loader} />;
};

```

Form Story with Interactions

```

import type { Meta, StoryObj } from 'philjs-storybook';
import { LoginForm } from './LoginForm';
import { within, userEvent, expect } from '@storybook/test';

const meta: Meta<typeof LoginForm> = {
  title: 'Forms/LoginForm',
  component: LoginForm,
};

export default meta;
type Story = StoryObj<typeof LoginForm>

export const FilledForm: Story = {
  play: async ({ canvasElement }) => {
    const canvas = within(canvasElement);

    await userEvent.type(canvas.getByLabelText('Email'), 'user@example.com');
    await userEvent.type(canvas.getByLabelText('Password'), 'password123');
    await userEvent.click(canvas.getByRole('button', { name: /login/i }));
  },
};

```

Island Story with Signals

```

import type { Meta, StoryObj } from 'philjs-storybook';
import { Counter } from './Counter';
import { withSignals } from 'philjs-storybook/decorators';

const meta: Meta<typeof Counter> = {
  title: 'Islands/Counter',
  component: Counter,
  decorators: [withSignals],
  parameters: {
    signals: {
      count: 0,
    },
  },
};

export default meta;
type Story = StoryObj<typeof Counter>;

export const Default: Story = {};

export const StartingAtTen: Story = {
  parameters: {
    signals: {
      count: 10,
    },
  },
};

```

Decorators

withRouter

Wraps stories with a mock router context.

```

import { withRouter } from 'philjs-storybook/decorators';

export default {
  decorators: [withRouter],
  parameters: {
    router: {
      pathname: '/products/[id]',
      params: { id: '123' },
      searchParams: '?tab=reviews',
    },
  },
};

```

withSignals

Provides signal state management for stories.

```

import { withSignals } from 'philjs-storybook/decorators';

export default {
  decorators: [withSignals],
  parameters: {
    signals: {
      isOpen: false,
      selectedItem: null,
    },
  },
};

```

withTheme

Enables theme switching for stories.

```

import { withTheme } from 'philjs-storybook/decorators';

export default {
  decorators: [withTheme],
  parameters: {
    theme: 'dark',
  },
};

```

withLayout

Controls story layout.

```

import { withLayout } from 'philjs-storybook/decorators';

export default {
  decorators: [withLayout],
  parameters: {
    layout: 'centered', // 'centered' | 'fullscreen' | 'padded' | 'none'
  },
};

```

Mocking Utilities

Signal Mocks

```
import { createMockSignal, createMockComputed } from 'philjs-storybook/mock';

// Create a mock signal
const count$ = createMockSignal(0);
count$.set(5);

// Check how many times it was accessed
console.log(count$.getSetCount()); // 1
console.log(count$.getGetCount()); // 0

// Reset the signal
count$.reset();
```

Router Mocks

```
import { createMockRouter } from 'philjs-storybook/mock';

const router = createMockRouter('/home');

router.navigate('/about');
router.back();
router.push('/contact');

// Check navigation history
console.log(router.getCalls());
// [
//   { method: 'navigate', args: ['/about'], timestamp: ... },
//   { method: 'back', args: [], timestamp: ... },
//   { method: 'push', args: ['/contact'], timestamp: ... }
// ]
```

API Mocks

```
import { createMockAPI } from 'philjs-storybook/mock';

export const handlers = createMockAPI([
  {
    method: 'GET',
    path: '/api/users/:id',
    response: { id: 1, name: 'John Doe' },
  },
  {
    method: 'POST',
    path: '/api/users',
    response: { id: 2, name: 'Jane Doe' },
    status: 201,
  },
]);
// Use in story parameters
export default {
  parameters: {
    msw: {
      handlers,
    },
  },
};
```

Route Mocks

```
import { createMockLoader, createMockAction } from 'philjs-storybook/mock';

// Mock loader
const loader = createMockLoader(async (params) => {
  return { user: { id: params.id, name: 'John' } };
});

await loader.load({ id: '123' });
console.log(loader.data()); // { user: { id: '123', name: 'John' } }

// Mock action
const action = createMockAction(async (data) => {
  return { success: true };
});

await action.submit({ name: 'John' });
console.log(action.data()); // { success: true }
```

Addons

Signal Inspector

View and manipulate signals in real-time:

1. Open the "Signal Inspector" panel in Storybook
2. See all registered signals and their values
3. Click a signal to edit its value

- Changes are reflected immediately in your component

Route Tester

Test route components with different parameters:

- Open the "Route Tester" panel
- Configure path, params, and search params
- Set loader/action data
- View navigation history

Theme Switcher

Switch between themes:

- Use the theme dropdown in the toolbar
- Or open the "Theme Switcher" panel
- Create custom themes with CSS variables

Viewport Helper

Test responsive designs:

- Use the viewport dropdown in the toolbar
- Or open the "Viewport" panel
- Select preset devices or create custom viewports

CLI Commands

Initialize Storybook

```
philjs storybook init
```

Start Dev Server

```
philjs storybook dev
# or
philjs storybook dev -p 6007
```

Build Storybook

```
philjs storybook build
# or
philjs storybook build -o dist/storybook
```

Generate Story

```
# Generate component story
philjs storybook generate Button

# Generate route story
philjs storybook generate UserProfile --type route

# Generate form story
philjs storybook generate ContactForm --type form

# Generate island story
philjs storybook generate Counter --type island
```

Configuration

.storybook/main.ts

```
import type { StorybookConfig } from 'philjs-storybook';

const config: StorybookConfig = {
  stories: ['../src/**/*.{stories.(ts|tsx)}'],
  addons: [
    '@storybook/addon-essentials',
    '@storybook/addon-interactions',
    'philjs-storybook/addons/signal-inspector',
    'philjs-storybook/addons/route-tester',
    'philjs-storybook/addons/theme-switcher',
    'philjs-storybook/addons/viewport',
  ],
  framework: {
    name: 'philjs-storybook',
    options: {},
  },
};

export default config;
```

.storybook/preview.ts

```

import type { Preview } from 'philjs-storybook';
import { withRouter, withTheme, withLayout } from 'philjs-storybook/decorators';

const preview: Preview = {
  decorators: [withRouter, withTheme, withLayout],
  parameters: {
    layout: 'padded',
    backgrounds: {
      default: 'light',
      values: [
        { name: 'light', value: '#ffffff' },
        { name: 'dark', value: '#000000' },
      ],
    },
  },
};

export default preview;

```

TypeScript Support

PhilJS Storybook is fully typed. All helpers, decorators, and mocks include comprehensive TypeScript definitions.

```

import type { Meta, StoryObj } from 'philjs-storybook';

interface ButtonProps {
  variant: 'primary' | 'secondary';
  size: 'small' | 'medium' | 'large';
  onClick?: () => void;
}

const meta: Meta<ButtonProps> = {
  // Fully typed
};

type Story = StoryObj<ButtonProps>;

```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./preset`, `/addons/signal-inspector`, `/addons/route-tester`, `/addons/theme-switcher`, `/addons/viewport`, `/decorators`, `/mocks`
- Source files: `packages/philjs-storybook/src/index.ts`, `packages/philjs-storybook/src/preset.ts`, `packages/philjs-storybook/src/decorators/index.ts`, `packages/philjs-storybook/src/mocks/index.ts`

Public API

- Direct exports: `addons`, `core`, `docs`, `framework`, `presetConfig`, `typescript`, `viteFinal`
- Re-exported names: `MockAPIHandler`, `MockRouter`, `RenderContext`, `StoryConfig`, `StoryContext`, `createMockAPI`, `createMockAction`, `createMockComputed`, `createMockLoader`, `createMockRouter`, `createMockSignal`, `createStory`, `presetConfig`, `renderer`, `withLayout`, `withMockData`, `withRouter`, `withSignals`, `withTheme`
- Re-exported modules: `/api-mocks.js`, `/decorators/index.js`, `/mocks/index.js`, `./preset.js`, `./renderer.js`, `./route-mocks.js`, `./router-mocks.js`, `./signal-mocks.js`, `./story-helpers.js`, `./with-layout.js`, `./with-mock-data.js`, `./with-router.js`, `./with-signals.js`, `./with-theme.js`

License

MIT PhilJS Team

@philjs/studio - Type: Node package - Purpose: Visual component builder with drag-and-drop for PhilJS - Version: 0.1.0 - Location: packages/philjs-studio - Entry points: .. -
Keywords: philjs, studio, visual-builder, drag-and-drop, no-code

@philjs/studio

Visual development studio for PhilJS applications. A comprehensive IDE-like environment for building, previewing, and deploying PhilJS components and applications.

Installation

```

npm install @philjs/studio
# or
yarn add @philjs/studio
# or
pnpm add @philjs/studio

```

Basic Usage

```

# Start the visual studio
npx philjs-studio

# Or with configuration
npx philjs-studio --port 4000 --project ./my-app

```

```
// Embed studio in your app
import { Studio, StudioProvider } from '@philjs/studio';

function DevEnvironment() {
  return (
    <StudioProvider project="./src">
      <Studio />
    </StudioProvider>
  );
}
```

Features

- **Visual Editor** - Drag-and-drop component composition
- **Code Editor** - Integrated code editor with IntelliSense
- **Component Library** - Browse and preview available components
- **Props Panel** - Visual editing of component properties
- **Live Preview** - Real-time preview of changes
- **State Inspector** - Debug component state and props
- **Design Tokens** - Manage colors, typography, spacing
- **Responsive Testing** - Test across device sizes
- **Component Docs** - Auto-generated documentation
- **Version Control** - Built-in Git integration
- **Deploy** - One-click deployment to cloud
- **Collaboration** - Real-time collaborative editing

Panels

Panel	Description
Component Tree	Hierarchical view of components
Props Editor	Edit component properties
Code View	Source code editor
Preview	Live component preview
Console	Debug output and logs
Assets	Manage images and files
Styles	CSS and design tokens

Keyboard Shortcuts

Shortcut	Action
Ctrl+S	Save changes
Ctrl+Z	Undo
Ctrl+Shift+Z	Redo
Ctrl+P	Quick file open
Ctrl+Space	Autocomplete
F5	Refresh preview

Configuration

```
// philjs-studio.config.json
{
  "port": 4000,
  "project": "./src",
  "components": "./src/components",
  "theme": "dark",
  "plugins": ["@philjs/studio-plugin-tailwind"]
}
```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: (none detected)

Public API

- Direct exports: (none detected)
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/styles - Type: Node package - Purpose: Scoped styles, CSS Modules, and CSS-in-JS for PhilJS - Version: 0.1.0 - Location: packages/philjs-styles - Entry points: packages/philjs-styles/src/index.ts, packages/philjs-styles/src/css-modules.ts, packages/philjs-styles/src/scoped.ts, packages/philjs-styles/src/css-in-js.ts, ./vite - Keywords: philjs, css, styles, scoped, css-modules, css-in-js

philjs-styles

Scoped styles, CSS Modules, and CSS-in-JS for PhilJS.

Features

- **Scoped Styles** - Component-scoped CSS with zero conflicts
- **CSS Modules** - Import CSS as JavaScript objects
- **CSS-in-JS** - Runtime and compile-time styling
- **Type Safety** - TypeScript support for CSS
- **SSR Compatible** - Server-side rendering support
- **Vite Plugin** - Optimized Vite integration
- **Theme Support** - Built-in theming utilities
- **No Runtime Overhead** - Optional zero-runtime CSS-in-JS

Installation

```
pnpm add philjs-styles
```

Quick Start

Scoped Styles

```
import { css } from 'philjs-styles/scoped';

export default function Button({ children }) {
  const styles = css`
    button {
      background: #3b82f6;
      color: white;
      padding: 0.5rem 1rem;
      border-radius: 0.5rem;
      border: none;
      cursor: pointer;
    }

    button:hover {
      background: #2563eb;
    }
  `;

  return (
    <>
      <style>{styles}</style>
      <button>{children}</button>
    </>
  );
}
```

CSS Modules

Create `Button.module.css`:

```
.button {
  background: #3b82f6;
  color: white;
  padding: 0.5rem 1rem;
  border-radius: 0.5rem;
}

.button:hover {
  background: #2563eb;
}

.primary {
  background: #3b82f6;
}

.secondary {
  background: #8b5cf6;
}
```

Use in component:

```

import styles from './Button.module.css';

export default function Button({ variant = 'primary', children }) {
  return (
    <button className={`${styles.button} ${styles[variant]}`}>
      {children}
    </button>
  );
}

```

CSS-in-JS

```

import { styled } from 'philjs-styles/css-in-js';

const Button = styled('button', {
  base: {
    padding: '0.5rem 1rem',
    borderRadius: '0.5rem',
    border: 'none',
    cursor: 'pointer',
    transition: 'background 0.2s'
  },
  variants: {
    variant: {
      primary: {
        background: '#3b82f6',
        color: 'white',
        '&:hover': {
          background: '#2563eb'
        }
      },
      secondary: {
        background: '#8b5cf6',
        color: 'white',
        '&:hover': {
          background: '#7c3aed'
        }
      }
    },
    size: {
      sm: { padding: '0.25rem 0.5rem', fontSize: '0.875rem' },
      md: { padding: '0.5rem 1rem', fontSize: '1rem' },
      lg: { padding: '0.75rem 1.5rem', fontSize: '1.125rem' }
    }
  }
});

// Usage
<Button variant="primary" size="lg">Click me</Button>

```

Vite Plugin

Update vite.config.ts:

```

import { defineConfig } from 'vite';
import philjs from 'philjs-cli/vite';
import styles from 'philjs-styles/vite';

export default defineConfig({
  plugins: [
    philjs(),
    styles({
      modules: true,           // Enable CSS Modules
      scoped: true,           // Enable scoped styles
      cssInJs: 'compile'     // 'compile' or 'runtime'
    })
  ]
});

```

Theming

```

import { createTheme, ThemeProvider } from 'philjs-styles/css-in-js';

const theme = createTheme({
  colors: {
    primary: '#3b82f6',
    secondary: '#8b5cf6',
    success: '#10b981',
    error: '#ef4444'
  },
  spacing: {
    sm: '0.5rem',
    md: '1rem',
    lg: '2rem'
  },
  fonts: {
    body: 'system-ui, sans-serif',
    mono: 'monospace'
  }
});

export default function App() {
  return (
    <ThemeProvider theme={theme}>
      <YourApp />
    </ThemeProvider>
  );
}

```

Use theme in components:

```

import { styled, useTheme } from 'philjs-styles/css-in-js';

const Button = styled('button', ({ theme }) => ({
  background: theme.colors.primary,
  padding: theme.spacing.md,
  fontFamily: theme.fonts.body
}));

```

Documentation

For more information, see the PhilJS documentation.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./css-modules`, `./scoped`, `./css-in-js`, `/vite`
- Source files: `packages/philjs-styles/src/index.ts`, `packages/philjs-styles/src/css-modules.ts`, `packages/philjs-styles/src/scoped.ts`, `packages/philjs-styles/src/css-in-js.ts`

Public API

- Direct exports: `CSSModuleClasses`, `ThemeProvider`, `bindStyles`, `createClassNames`, `createGlobalStyle`, `createStyled`, `createTheme`, `css`, `cssModules`, `cva`, `getCSSModuleConfig`, `importCSSModule`, `keyframes`, `setTheme`, `styled`, `subscribeToTheme`, `useCSSModule`, `useTheme`
- Re-exported names: `CSSProperties`, `StyleObject`, `StyleVariant`, `Theme`, `ThemeConfig`, `ThemeProvider`, `bindStyles`, `classNames`, `cx`, `createGlobalStyle`, `createStyled`, `createTheme`, `css`, `cssModules`, `cx`, `extractCriticalCSS`, `injectStyles`, `keyframes`, `mergeStyles`, `philjsStylesPlugin`, `styled`, `useCSSModule`, `useTheme`
- Re-exported modules: `./css-in-js.js`, `./css-modules.js`, `./scoped.js`, `./types.js`, `./utils.js`, `./vite-plugin.js`

License

MIT

`## @philjs/swift - Type: Node package - Purpose: Swift bindings for PhilJS - iOS and macOS native app development - Version: 0.1.0 - Location: packages/philjs-swift - Entry points: packages/philjs-swift/src/index.ts, ./ios, ./macos - Keywords: philjs, swift, ios, macos, native, swiftui, uikit, appkit`

philjs-swift

Swift bindings for PhilJS - iOS and macOS native app development.

Requirements

- Node.js 24** or higher
- TypeScript 6** or higher
- ESM only** - CommonJS is not supported
- Swift 5.9+** for Swift-side features
- Xcode 15+** for iOS/macOS development

Installation

```
pnpm add @philjs/swift
```

Features

- iOS Integration** - Build iOS apps with PhilJS
- macOS Support** - Native macOS application development
- SwiftUI** - Reactive UI with SwiftUI interop
- UIKit Bridge** - Access UIKit APIs from TypeScript

- **AppKit Bridge** - Access AppKit APIs for macOS
- **Type Bridge** - Automatic TypeScript <-> Swift type conversion

Quick Start

iOS Setup

```
import { createIOSApp } from '@philjs/swift/ios';

const app = createIOSApp({
  name: 'MyApp',
  bundleId: 'com.example.myapp',
  deploymentTarget: '17.0',
});

// Generate iOS project
await app.scaffold('./ios');
```

macOS Setup

```
import { createMacOSApp } from '@philjs/swift/macos';

const app = createMacOSApp({
  name: 'MyDesktopApp',
  bundleId: 'com.example.mydesktopapp',
  deploymentTarget: '14.0',
  category: 'public.app-category.developer-tools',
});

// Generate macOS project
await app.scaffold('./macos');
```

SwiftUI Integration

```
import { createSwiftUIBinding } from '@philjs/swift/ios';
import { signal } from '@philjs/core';

const count = signal(0);

// Create SwiftUI-compatible state
const swiftState = createSwiftUIBinding(count, {
  name: 'counter',
  type: 'Int',
});

// Generated Swift code:
// @State private var counter: Int = 0
```

Swift Bridge

```
import { createSwiftBridge } from '@philjs/swift';

const bridge = createSwiftBridge({
  module: './swift/Shared.swift',
  exports: ['calculateSum', 'formatDate', 'UserData'],
});

// Call Swift functions from TypeScript
const sum = await bridge.invoke('calculateSum', [1, 2, 3]);
console.log(sum); // 6

// Use Swift structs
const user = await bridge.create('UserData', {
  name: 'John',
  age: 30,
});
```

ES2024 Features

Promise.withResolvers() for iOS APIs

```

import { createIOSBridge } from '@philjs/swift/ios';

const ios = createIOSBridge();

// Wrap completion handler-based iOS APIs with Promise.withResolvers()
function requestNotificationPermission() {
  const { promise, resolve, reject } = Promise.withResolvers<boolean>();

  ios.requestNotificationAuthorization({
    onGranted: () => resolve(true),
    onDenied: () => resolve(false),
    onError: (error) => reject(error),
  });
}

return promise;
}

const granted = await requestNotificationPermission();

```

Object.groupBy() for UI Components

```

import { createSwiftUIBinding } from '@philjs/swift/ios';

interface View {
  id: string;
  type: 'Text' | 'Button' | 'TextField' | 'Image';
  props: Record<string, unknown>;
}

// Group views by type using ES2024 Object.groupBy()
function organizeViews(views: View[]) {
  const grouped = Object.groupBy(views, v => v.type);

  return {
    textViews: grouped.Text ?? [],
    buttonViews: grouped.Button ?? [],
    textFieldViews: grouped.TextField ?? [],
    imageViews: grouped.Image ?? [],
  };
}

```

Resource Management with using

```

import { createSwiftBridge } from '@philjs/swift';

// Automatic cleanup with TypeScript 6 explicit resource management
async function runSwiftCode() {
  await using bridge = await createSwiftBridge({
    module: './swift/Process.swift',
    [Symbol.asyncDispose]: async () => {
      await bridge.cleanup();
      console.log('Swift bridge disposed');
    }
  });

  // Bridge automatically cleaned up when scope exits
  return bridge.invoke('processData', { input: 'test' });
}

```

CLI

```

# Initialize iOS project
philjs-swift init ios my-app

# Initialize macOS project
philjs-swift init macos my-app

# Build Swift module
philjs-swift build ./src/Shared.swift

# Generate TypeScript types from Swift
philjs-swift types ./swift/Models.swift

# Run iOS simulator
philjs-swift ios:run

# Run macOS app
philjs-swift macos:run

```

API Reference

ios

Function	Description
createIOSApp(config)	Create iOS app config
createIOSBridge()	Bridge to iOS APIs
scaffold(path)	Generate iOS project

macOS

Function	Description
createMacOSApp(config)	Create macOS app config
createMacOSBridge()	Bridge to macOS APIs
scaffold(path)	Generate macOS project

SwiftUI

Function	Description
createSwiftUIBinding(signal, options)	Bind PhilJS signal to SwiftUI state
generateSwiftUIView(component)	Generate SwiftUI code

Swift Bridge

Function	Description
createSwiftBridge(config)	Create Swift bridge
bridge.invoke(fn, args)	Call Swift function
bridge.create(struct, props)	Instantiate Swift struct

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `,`, `/ios`, `/macos`
- Source files: `packages/philjs-swift/src/index.ts`

Public API

- Direct exports: (none detected)
- Re-exported names: (none detected)
- Re-exported modules: `./bridge.js`, `./codegen.js`, `./types.js`

License

MIT

@philjs/table - Type: Node package - Purpose: Headless, type-safe table component for PhilJS - TanStack Table inspired - Version: 0.1.0 - Location: packages/philjs-table - Entry points: packages/philjs-table/src/index.ts - Keywords: table, datagrid, headless, sorting, filtering, pagination, philjs

@philjs/table

Headless, type-safe table component for PhilJS - TanStack Table inspired

Overview

Headless, type-safe table component for PhilJS - TanStack Table inspired

Focus Areas

- table, datagrid, headless, sorting, filtering, pagination, philjs

Entry Points

- `packages/philjs-table/src/index.ts`

Quick Start

```
import { Cell, CellContext, Column } from '@philjs/table';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- Cell
- CellContext
- Column
- ColumnDef
- ColumnFiltersState
- ColumnHelper
- ColumnVisibilityState
- ExpandedState
- FilterFn
- Header
- HeaderContext

- HeaderGroup

Install

```
pnpm add @philjs/table
```

Usage

```
import { Cell, CellContext, Column } from '@philjs/table';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-table/src/index.ts

Public API

- Direct exports: Cell, CellContext, Column, ColumnDef, ColumnFiltersState, ColumnHelper, ColumnVisibilityState, ExpandedState, FilterFn, Header, HeaderContext, HeaderGroup, PaginationState, Row, RowModel, RowSelectionState, SortDirection, SortingFn, SortingState, Table, TableOptions, TableState, createColumnHelper, createTable, filterFns, flexRender, getCoreRowModel, getFilteredRowModel, getPaginatedRowModel, getSortedRowModel, sortingFns
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/tailwind - Type: Node package - Purpose: Tailwind CSS integration for PhilJS with IntelliSense and optimizations - Version: 0.1.0 - Location: packages/philjs-tailwind - Entry points: packages/philjs-tailwind/src/index.ts, packages/philjs-tailwind/src/preset.ts, packages/philjs-tailwind/src/plugin.ts, ./vite - Keywords: philjs, tailwind, tailwindcss, css, utility-first

philjs-tailwind

Tailwind CSS integration for PhilJS with IntelliSense and optimizations.

Features

- **Zero Configuration** - Works out of the box with PhilJS
- **IntelliSense Support** - Full autocomplete in VS Code
- **JIT Mode** - Just-in-Time compilation for faster builds
- **Custom Preset** - PhilJS-optimized Tailwind preset
- **SSR Compatible** - Works with server-side rendering
- **Vite Plugin** - Optimized for Vite builds
- **Dark Mode** - Class-based dark mode utilities

Installation

```
pnpm add -D philjs-tailwind tailwindcss autoprefixer postcss
```

Quick Start

1. Create Tailwind Config

Create `tailwind.config.js`:

```
import philJSPreset from 'philjs-tailwind/preset';

export default {
  content: ['./src/**/*.{ts,tsx}'],
  presets: [philJSPreset],
  darkMode: 'class'
};
```

2. Add Vite Plugin

Update `vite.config.ts`:

```
import { defineConfig } from 'vite';
import philjs from 'philjs-cli/vite';
import tailwind from 'philjs-tailwind/vite';

export default defineConfig({
  plugins: [
    philjs(),
    tailwind()
  ]
});
```

3. Import Styles

In your `src/entry-client.ts`:

```
import 'tailwindcss/tailwind.css';
```

4. Use Tailwind Classes

```
export default function Button({ children }) {
  return (
    <button className="px-4 py-2 bg-blue-500 text-white rounded-lg hover:bg-blue-600 transition-colors">
      {children}
    </button>
  );
}
```

PhilJS Preset

The PhilJS preset includes:

- **Custom Colors** - PhilJS brand colors
- **Extended Spacing** - Additional spacing utilities
- **Typography** - Optimized font scales
- **Animations** - Smooth transitions and animations
- **Container Queries** - Container query utilities

Dark Mode

```
import { signal } from 'philjs-core';

export default function App() {
  const isDark = signal(false);

  return (
    <div className={isDark() ? 'dark' : ''}>
      <div className="bg-white dark:bg-gray-900 text-gray-900 dark:text-white">
        <button onClick={() => isDark.set(!isDark())}>
          Toggle Dark Mode
        </button>
      </div>
    </div>
  );
}
```

Custom Plugin

Create custom Tailwind utilities:

```
import plugin from 'tailwindcss/plugin';

export default {
  plugins: [
    plugin(({ addUtilities }) => {
      addUtilities({
        '.glass': {
          'background': 'rgba(255, 255, 255, 0.1)',
          'backdrop-filter': 'blur(10px)',
          'border': '1px solid rgba(255, 255, 255, 0.2)'
        }
      });
    })
  ]
};
```

Documentation

For more information, see the PhilJS documentation.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./preset`, `./plugin`, `./vite`
- Source files: `packages/philjs-tailwind/src/index.ts`, `packages/philjs-tailwind/src/preset.ts`, `packages/philjs-tailwind/src/plugin.ts`

Public API

- Direct exports: PhilJSPluginOptions, PhilJPresetOptions, createPhilJSPlugin, createPhilJPreset, createTailwindConfig, philjsPreset, philjsTailwindPlugin
- Re-exported names: ClassValue, PhilJSPluginOptions, PhilJPresetOptions, PhilSTailwindViteOptions, VariantProps, cjsx, cn, createPhilJSPlugin, createPhilJPreset, cva, philjsPreset, philjsTailwindPlugin, philjsTailwindVite, tw, twJoin, twMerge
- Re-exported modules: ./plugin.js, ./preset.js, ./types.js, ./utils.js, ./vite-plugin.js

License

MIT

@philjs/templates - Type: Node package - Purpose: Starter templates for PhilJS applications - Version: 0.1.0 - Location: packages/philjs-templates - Entry points: packages/philjs-templates/src/index.ts - Keywords: philjs, templates, starter, create-philjs, scaffolding

philjs-templates

Starter templates for PhilJS applications.

Installation

This package is used automatically by `create-philjs`. No manual installation needed.

```
pnpm create philjs my-app
```

Available Templates

Basic Template

A minimal PhilJS application with core features:

```
pnpm create philjs my-app --template basic
```

Features: - PhilJS Core (signals, memo, effect) - File-based routing - Hot Module Replacement - TypeScript support - Basic styling

Full Template (Default)

Complete PhilJS application with all features:

```
pnpm create philjs my-app --template full
```

Features: - Everything in Basic template - Server-Side Rendering (SSR) - Islands Architecture - API routes - Database integration (DrizzleORM) - Authentication setup - Testing setup (Vitest) - E2E tests (Playwright)

SSR Template

Optimized for server-side rendering:

```
pnpm create philjs my-app --template ssr
```

Features: - SSR-first architecture - Streaming SSR support - SEO optimization - Meta tags management - Sitemap generation - RSS feed support

API Template

Backend-focused template:

```
pnpm create philjs my-app --template api
```

Features: - API routes only - Database integration - Authentication - Rate limiting - CORS configuration - OpenAPI documentation

SPA Template

Single-Page Application template:

```
pnpm create philjs my-app --template spa
```

Features: - Client-side only - Client-side routing - No SSR overhead - Optimized bundle size - PWA support

Component Library Template

For building component libraries:

```
pnpm create philjs my-app --template library
```

Features: - Component development setup - Storybook integration - Documentation generator - NPM publishing config - Bundle size optimization

Template Structure

Basic Template

```
my-app/
src/
  routes/
    index.tsx
  components/
    Counter.tsx
  styles/
    global.css
  entry-client.tsx
public/
  favicon.ico
package.json
vite.config.ts
tsconfig.json
```

Full Template

```
my-app/
src/
  routes/
    index.tsx
    about.tsx
    api/
      users.ts
  components/
    Counter.tsx
    TodoList.tsx
  lib/
    db.ts
    auth.ts
  styles/
    global.css
  entry-client.tsx
  entry-server.tsx
public/
  assets/
tests/
  unit/
  e2e/
package.json
vite.config.ts
tsconfig.json
playwright.config.ts
.env.example
```

Programmatic Usage

Use templates programmatically in your own scaffolding tools:

```
import { getTemplate, scaffoldProject } from 'philjs-templates';

// Get template files
const template = await getTemplate('full');

// Scaffold a new project
await scaffoldProject({
  template: 'full',
  projectName: 'my-app',
  targetDir: './my-app',
  typescript: true,
  git: true
});
```

Template Features

TypeScript

All templates support TypeScript by default. Use `--js` flag for JavaScript:

```
pnpm create philjs my-app --js
```

Package Manager

Choose your preferred package manager:

```
pnpm create philjs my-app --pm npm
pnpm create philjs my-app --pm yarn
pnpm create philjs my-app --pm pnpm # default
```

Git Integration

Initialize Git repository automatically:

```
pnpm create philjs my-app --git
```

Styling Options

Choose a styling solution:

```
pnpm create philjs my-app --style tailwind
pnpm create philjs my-app --style css-modules
pnpm create philjs my-app --style styled-components
pnpm create philjs my-app --style vanilla # default
```

Customization

Custom Templates

Create your own templates by adding a directory in `templates/`:

```
templates/
  my-custom-template/
    template.json
    package.json
  src/
  public/
```

template.json:

```
{
  "name": "my-custom-template",
  "description": "My custom PhilJS template",
  "features": ["custom-feature"],
  "prompts": [
    {
      "name": "includeAuth",
      "type": "confirm",
      "message": "Include authentication?"
    }
  ]
}
```

Template Variables

Use variables in template files:

```
// src/config.ts
export const config = {
  appName: "{{PROJECT_NAME}}",
  version: "{{VERSION}}",
  // Variables are replaced during scaffolding
};
```

API

Functions

- `getTemplate(name)` - Get template by name
- `listTemplates()` - List all available templates
- `scaffoldProject(options)` - Create project from template
- `validateTemplate(name)` - Validate template structure

Options

```
interface ScaffoldOptions {
  template: string;
  projectName: string;
  targetDir: string;
  typescript?: boolean;
  git?: boolean;
  packageManager?: 'npm' | 'yarn' | 'pnpm';
  install?: boolean;
}
```

Examples

Create Full Stack App

```
pnpm create philjs my-app --template full --pm pnpm --git
cd my-app
pnpm dev
```

Create Component Library

```
pnpm create philjs ui-components --template library
cd ui-components
pnpm dev
pnpm build
```

Create API Server

```
pnpm create philjs api-server --template api
cd api-server
pnpm dev
```

Scripts

All templates include these npm scripts:

```
{
  "scripts": {
    "dev": "philjs dev",
    "build": "philjs build",
    "preview": "philjs preview",
    "test": "vitest",
    "test:e2e": "playwright test",
    "typecheck": "tsc --noEmit"
  }
}
```

Documentation

For more information, see the PhilJS documentation and Getting Started guide.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-templates/src/index.ts

Public API

- Direct exports: Template, getAllTags, getTemplate, getTemplates, getTemplatesByDifficulty, getTemplatesByTag, searchTemplates, templates
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/testing - Type: Node package - Purpose: Official testing library for PhilJS - render, queries, events, and utilities - Version: 0.1.0 - Location: packages/philjs-testing - Entry points: packages/philjs-testing/src/index.ts, packages/philjs-testing/src/vitest.ts, packages/philjs-testing/src/jest.ts - Keywords: philjs, testing, test, vitest, testing-library

philjs-testing

Official testing library for PhilJS - render, queries, events, and utilities for testing components and signals.

Requirements

- **Node.js 24** or higher
- **TypeScript 6** or higher
- **ESM only** - CommonJS is not supported

Features

- **Component Rendering** - Render components in a test environment
- **DOM Queries** - Find elements using accessible queries
- **User Events** - Simulate user interactions
- **Signal Testing** - Test reactive state and computed values
- **Async Utilities** - Wait for conditions and async updates
- **Custom Matchers** - Assertion helpers for common checks
- **Hook Testing** - Test custom hooks in isolation
- **Debug Helpers** - Inspect DOM, signals, and accessibility
- **Vitest Integration** - Works seamlessly with Vitest

Installation

```
pnpm add -D philjs-testing vitest jsdom
```

Quick Start

Basic Component Test

```
import { render, screen, fireEvent } from 'philjs-testing';
import { describe, it, expect } from 'vitest';
import Counter from './Counter';

describe('Counter', () => {
  it('increments count on button click', () => {
    render();
    const button = screen.getByRole('button', { name: /increment/i });
    const count = screen.getByText(/count: 0/i);

    expect(count).toBeInTheDocument();

    fireEvent.click(button);
    expect(screen.getByText(/count: 1/i)).toBeInTheDocument();

    fireEvent.click(button);
    expect(screen.getByText(/count: 2/i)).toBeInTheDocument();
  });
});
```

Testing Signals

```

import { signal } from 'philjs-core';
import { waitForSignal, signalValue } from 'philjs-testing';
import { describe, it, expect } from 'vitest';

describe('Signal Tests', () => {
  it('waits for signal to update', async () => {
    const count = signal(0);

    setTimeout(() => count.set(10), 100);

    await waitForSignal(count, (value) => value === 10);
    expect(signalValue(count)).toBe(10);
  });

  it('tracks signal history', async () => {
    const count = signal(0);
    const history = [];

    count.subscribe((value) => history.push(value));

    count.set(1);
    count.set(2);
    count.set(3);

    expect(history).toEqual([0, 1, 2, 3]);
  });
});

```

User Events

Use userEvent for more realistic interactions:

```

import { render, screen, userEvent } from 'philjs-testing';
import LoginForm from './LoginForm';

it('submits the form with user credentials', async () => {
  const handleSubmit = vi.fn();
  render(<LoginForm onSubmit={handleSubmit} />);

  const user = userEvent.setup();

  await user.type(screen.getByLabelText(/email/i), 'user@example.com');
  await user.type(screen.getByLabelText(/password/i), 'password123');
  await user.click(screen.getByRole('button', { name: /sign in/i }));

  expect(handleSubmit).toHaveBeenCalledWith({
    email: 'user@example.com',
    password: 'password123'
  });
});

```

Async Utilities

Wait for elements and conditions:

```

import { render, screen, waitFor, waitForElementToBeRemoved } from 'philjs-testing';
import UserProfile from './UserProfile';

it('loads and displays user data', async () => {
  render(<UserProfile userId="123" />);

  // Wait for loading spinner to disappear
  await waitForElementToBeRemoved(() => screen.queryByText(/loading/i));

  // Wait for data to appear
  await waitFor(() => {
    expect(screen.getByText(/john doe/i)).toBeInTheDocument();
  });

  expect(screen.getByText(/john@example.com/i)).toBeInTheDocument();
});

```

Testing Hooks

Test custom hooks in isolation:

```

import { renderHook, act } from 'philjs-testing';
import { useCounter } from './useCounter';

describe('useCounter', () => {
  it('increments counter', () => {
    const { result } = renderHook(() => useCounter(0));

    expect(result.current.count()).toBe(0);

    act(() => {
      result.current.increment();
    });

    expect(result.current.count()).toBe(1);
  });

  it('accepts initial value', () => {
    const { result } = renderHook(() => useCounter(10));
    expect(result.current.count()).toBe(10);
  });
});

```

Custom Matchers

Use built-in matchers for common assertions:

```

import { render, screen } from 'philjs-testing';

it('uses custom matchers', () => {
  render(<button disabled>Click me</button>);

  const button = screen.getByRole('button');

  expect(button).toBeInTheDocument();
  expect(button).toBeDisabled();
  expect(button).toHaveTextContent('Click me');
  expect(button).not.toBeVisible(); // if hidden
  expect(button).toHaveAttribute('disabled');
});

```

Debug Utilities

Inspect rendered output and signals:

```

import { render, screen, debug, debugSignals } from 'philjs-testing';
import { signal } from 'philjs-core';

it('debugs component output', () => {
  const count = signal(5);
  const { container } = render(<Counter count={count} />);

  // Print the DOM
  debug(container);

  // Print specific element
  debug(screen.getByRole('button'));

  // Debug signal values
  debugSignals({ count });
  // Outputs: { count: 5 }
});

```

API Reference

Rendering

```
render(component, options?)
```

Renders a component for testing.

Returns: RenderResult with utilities and container

Options: - container - Custom container element - wrapper - Wrapper component (e.g., providers) - hydrate - Use hydration instead of render

Example:

```
const { container, rerender, unmount } = render(<App />);
```

```
cleanup()
```

Unmounts all rendered components. Called automatically after each test.

Queries

All queries from @testing-library/dom are available:

- getByRole, queryByRole, findByRole
- getByText, queryByText, findByText
- getLabelText, queryByLabelText, findByLabelText
- getPlaceholderText, getByAltText, getByTitle

- `getById`, `queryById`, `findByTestId`
- Plus `getAllBy*`, `queryAllBy*`, `findAllBy*` variants

screen - Query the entire document:

```
screen.getByRole('button', { name: /submit/i });
```

within - Query within a specific element:

```
const form = screen.getByRole('form');
const button = within(form). getByRole('button');
```

Events

`fireEvent.{eventType}(element, options?)`

Fire DOM events on elements:

```
fireEvent.click(button);
fireEvent.change(input, { target: { value: 'new value' } });
fireEvent.submit(form);
```

userEvent

More realistic user interactions:

```
const user = userEvent.setup();

await user.click(button);
await user.type(input, 'Hello');
await user.clear(input);
await user.selectOptions(select, 'option-1');
await user.upload(fileInput, file);
await user.keyboard('{Enter}');
```

Async Utilities

`waitFor(callback, options?)`

Wait for a condition to be true:

```
await waitFor(() => {
  expect(screen.getByText(/success/i)).toBeInTheDocument();
}, { timeout: 3000 });
```

waitForElementToBeRemoved(callback)

Wait for element to be removed from DOM:

```
await waitForElementToBeRemoved(() => screen.queryByText(/loading/i));
```

findBy* Queries

Async versions of queries that wait for elements:

```
const button = await screen.findByRole('button');
```

waitForSignal(signal, condition, options?)

Wait for signal to meet a condition:

```
await waitForSignal(count, (value) => value > 5);
```

Signal Testing

`createMockSignal(initialValue)`

Create a mock signal for testing:

```
const mockCount = createMockSignal(0);
mockCount.set(5);
expect(mockCount()).toBe(5);
```

signalValue(signal)

Get current signal value safely:

```
const value = signalValue(count);
```

waitForSignalValue(signal, expectedValue, options?)

Wait for signal to have a specific value:

```
await waitForSignalValue(count, 10);
```

Matchers

- `toBeInTheDocument()` - Element is in the DOM
- `toBeVisible()` - Element is visible
- `toBeDisabled() / toBeEnabled()` - Element disabled state
- `toHaveTextContent(text)` - Element contains text
- `toHaveAttribute(attr, value?)` - Element has attribute
- `toHaveClass(className)` - Element has CSS class
- `toHaveStyle(styles)` - Element has inline styles
- `toHaveFocus()` - Element has focus
- `toHaveValue(value)` - Input has value
- `toBeChecked()` - Checkbox/radio is checked
- `toBeEmptyDOMElement()` - Element has no children

Hooks

```
renderHook(hook, options?)
```

Render a hook for testing:

```
const { result, rerender, unmount } = renderHook(() => useMyHook());
expect(result.current.value).toBe(42);
```

```
act(callback)
```

Wrap state updates in `act`:

```
act(() => {
  result.current.increment();
});
```

Debug & Snapshots

- `debug(element?)` - Print DOM to console
- `logDOM(element?)` - Same as debug
- `prettyDOM(element)` - Return formatted DOM string
- `debugSignals(signals)` - Print signal values
- `debugA11y(element)` - Check accessibility tree
- `snapshot(element)` - Create DOM snapshot
- `compareSnapshots(snap1, snap2)` - Compare snapshots

```
takeSnapshot(element, options?)
```

Create a snapshot of DOM or component state:

```
const result = takeSnapshot(container, {
  maxLength: 5000,
  includeSignals: true,
  serializer: (el) => el.outerHTML
});

if (!result.toMatch(expectedSnapshot)) {
  console.log(result.diff(expectedSnapshot));
}
```

```
createSnapshotMatcher(options?)
```

Create a snapshot matcher for testing:

```
const matcher = createSnapshotMatcher({ updateMode: false });

it('matches snapshot', () => {
  const { container } = render(<Component />);
  matcher.matchSnapshot('component-default', container);
});
```

```
snapshotSignalState(signals)
```

Capture signal state as JSON:

```
const snapshot = snapshotSignalState({
  count: signal(5),
  name: signal('test')
});
// Returns: { "count": 5, "name": "test" }
```

```
compareSignalSnapshots(actual, expected)
```

Compare signal states:

```

const result = compareSignalSnapshots(
  { count: signal(5) },
  { count: 5 }
);

expect(result.match).toBe(true);

```

Vitest Configuration

Add to your `vitest.config.ts`:

```

import { defineConfig } from 'vitest/config';

export default defineConfig({
  test: {
    environment: 'jsdom',
    globals: true,
    setupFiles: ['./test/setup.ts']
  }
});

```

Create `test/setup.ts`:

```

import { cleanup } from 'philjs-testing';
import { afterEach } from 'vitest';

// Cleanup after each test
afterEach(() => {
  cleanup();
});

```

Best Practices

1. **Use accessible queries** - Prefer `getByRole` and `getByText` over `getById`
2. **Test behavior, not implementation** - Test what users see and do
3. **Use userEvent for interactions** - More realistic than `fireEvent`
4. **Wait for async changes** - Use `waitFor` and `findBy*` queries
5. **Clean up after tests** - Use `cleanup()` to avoid test pollution
6. **Test signal behavior** - Verify reactive state changes
7. **Debug when needed** - Use `debug()` and `screen.debug()`

Examples

See the `examples` directory for full working tests.

Documentation

For more information, see the PhilJS documentation.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./vitest`, `./jest`
- Source files: `packages/philjs-testing/src/index.ts`, `packages/philjs-testing/src/vitest.ts`, `packages/philjs-testing/src/jest.ts`

Public API

- Direct exports: (none detected)
- Re-exported names: // Re-export everything for convenience `render`, `A11yIncomplete`, `A11yPass`, `A11yReport`, `A11yViolation`, `APITestHelper`, `ActionArgs`, `BoundFunctions`, `ComponentQueries`, `ComponentTestConfig`, `ComponentTestResult`, `GraphQLMock`, `HookResult`, `IntegrationTestContext`, `IntegrationTestOptions`, `LoaderArgs`, `MatcherResult`, `MockAction`, `MockHandler`, `MockRequest`, `MockResponse`, `MockResponseConfig`, `MockRouteOptions`, `MockSignal`, `NavigateOptions`, `NavigationState`, `NetworkMock`, `NetworkStats`, `PerformanceMetrics`, `Queries`, `QueryOptions`, `RenderOptions`, `RenderResult`, `RouteTestContext`, `SnapshotMatcher`, `SnapshotOptions`, `SnapshotResult`, `SubmitOptions`, `TestContext`, `TestFixture`, `VisualDiff`, `VisualTestOptions`, `WaitForOptions`, `WebSocketMock`, `act`, `assertNavigationState`, `assertRouteParams`, `assertSearchParams`, `assertSignalHistory`, `benchmarkRoute`, `cleanup`, `cleanupHooks`, `compareSignalSnapshots`, `compareSnapshots`, `componentFixture`, `createAPITestHelper`, `createAuthTestHelper`, `createDatabaseTestHelper`, `createEvent`, `createFixture`, `createGraphQLMock`, `createIntegrationTest`, `createMockAction`, `createMockComputed`, `createMockFormData`, `createMockLoader`, `createMockRequest`, `createMockRoute`, `createMockSignal`, `createNetworkMock`, `createSnapshotMatcher`, `createVisualSnapshot`, `createWebSocketMock`, `debug`, `debugA11y`, `debugForm`, `debugSignals`, `delay`, `delayed`, `expectActionToReturn`, `expectActionToThrow`, `expectHTMLSnapshot`, `expectJSONSnapshot`, `expectLoaderToReturn`, `expectLoaderToThrow`, `expectNoA11yViolations`, `expectNoRequests`, `expectRenderWithinBudget`, `expectRequest`, `expectRequestCount`, `findByRole`, `findByText`, `fireEvent`, `flaky`, `getNetworkStats`, `integrationWaitFor`, `interactionTest`, `json`, `logDOM`, `measureRenderPerformance`, `measureResponseTime`, `networkError`, `paginated`, `prettyDOM`, `queries`, `render`, `renderHook`, `retry`, `screen`, `setup`, `signalValue`, `snapshot`, `snapshotSignalState`, `takeSnapshot`, `testAction`, `testComponent`, `testLoader`, `testLoaderWithParams`, `testNavigation`, `testPostAction`, `testRouteFlow`, `toBeChecked`, `toBeDisabled`, `toBeEmptyDOMElement`, `toBeEnabled`, `toBelnTheDocument`, `toBeVisible`, `toHaveAttribute`, `toHaveClass`, `toHaveFocus`, `toHaveStyle`, `toHaveTextContent`, `toHaveValue`, `updateVisualSnapshot`, `user`, `userEvent`, `visualTest`, `waitFor`, `waitForElementToBeRemoved`, `waitForLoaderData`, `waitForLoadingToFinish`, `waitForNavigation`, `waitForNetworkIdle`, `waitForSignal`, `waitForSignalValue`, `withBody`, `withHeaders`, `withQuery`, `within`
- Re-exported modules: `./async.js`, `./component-testing.js`, `./debug.js`, `./events.js`, `./hooks.js`, `./index.js`, `./integration.js`, `./matchers.js`, `./network-mocking.js`, `./queries.js`, `./render.js`, `./route-testing.js`, `./signals.js`, `./snapshot.js`, `./user-event.js`

License

MIT

@philjs/time-travel - Type: Node package - Purpose: Time-travel debugging for PhilJS - Elm-style state history with visual debugger - Version: 0.1.0 - Location: packages/philjs-time-travel - Entry points: packages/philjs-time-travel/src/index.ts - Keywords: philjs, time-travel, debugging, devtools, state-management, elm-style, replay

@philjs/time-travel

Time-travel debugging for PhilJS - Elm-style state history with visual debugger

Overview

Time-travel debugging for PhilJS - Elm-style state history with visual debugger

Focus Areas

- philjs, time-travel, debugging, devtools, state-management, elm-style, replay

Entry Points

- packages/philjs-time-travel/src/index.ts

Quick Start

```
import { ActionInfo, ComponentSnapshot, ConsoleLog } from '@philjs/time-travel';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- ActionInfo
- ComponentSnapshot
- ConsoleLog
- NetworkRequest
- SnapshotMetadata
- StateDiff
- StateSnapshot
- TimeTravelConfig
- TimeTravelEngine
- TimeTravelState
- deepEqual
- diffStates

Install

```
pnpm add @philjs/time-travel
```

Usage

```
import { ActionInfo, ComponentSnapshot, ConsoleLog } from '@philjs/time-travel';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-time-travel/src/index.ts

Public API

- Direct exports: ActionInfo, ComponentSnapshot, ConsoleLog, NetworkRequest, SnapshotMetadata, StateDiff, StateSnapshot, TimeTravelConfig, TimeTravelEngine, TimeTravelState, deepEqual, diffStates, getTimeTravelEngine, initTimeTravel, useStateDiff, useTimeTravel, useStateTravelState
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/trpc - Type: Node package - Purpose: Type-safe API layer for PhilJS - RPC client/server, middleware, platform adapters - Version: 0.1.0 - Location: packages/philjs-trpc - Entry points: packages/philjs-trpc/src/index.ts, packages/philjs-trpc/src/client/index.ts, packages/philjs-trpc/src/server/index.ts - Keywords: philjs, trpc, api, type-safe, rpc, server

@philjs/trpc

Type-safe API layer for PhilJS - RPC client/server, middleware, platform adapters

Overview

Type-safe API layer for PhilJS - RPC client/server, middleware, platform adapters

Focus Areas

- philjs, trpc, api, type-safe, rpc, server

Entry Points

- packages/philijs-trpc/src/index.ts
- packages/philijs-trpc/src/client/index.ts
- packages/philijs-trpc/src/server/index.ts

Quick Start

```
import { ErrorCodes, ProcedureDefinition, RPCError } from '@philijs/trpc';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- ErrorCodes
- ProcedureDefinition
- RPCError
- createAuthMiddleware
- createBatchedClient
- createCacheMiddleware
- createCachedQuery
- createClient
- createLoggingMiddleware
- createQueryCache
- createRateLimitMiddleware
- createRoleMiddleware

Install

```
pnpm add @philijs/trpc
```

Usage

```
import { ErrorCodes, ProcedureDefinition, RPCError } from '@philijs/trpc';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: ., ./client, ./server
- Source files: packages/philijs-trpc/src/index.ts, packages/philijs-trpc/src/client/index.ts, packages/philijs-trpc/src/server/index.ts

Public API

- Direct exports: ErrorCodes, ProcedureDefinition, RPCError, createAuthMiddleware, createBatchedClient, createCacheMiddleware, createCachedQuery, createClient, createLoggingMiddleware, createQueryCache, createRateLimitMiddleware, createRoleMiddleware, createRouter, validateInput
- Re-exported names: AdapterConfig, AdapterType, AuthContext,BaseContext, BatchConfig, ClientConfig, DataTransformer, ErrorCodes, ErrorHandler, LinkConfig, MiddlewareFunction, ProcedureConfig, RPCError, RouterConfig, Session, SubscriptionCallbacks, SubscriptionConfig, User, createAuthMiddleware, createBatchedClient, createCacheMiddleware, createCachedQuery, createClient, createCloudflareAdapter, createExpressAdapter, createFastifyAdapter, createHonoAdapter, createLambdaAdapter, createLoggingMiddleware, createQueryCache, createRateLimitMiddleware, createRoleMiddleware, createRouter, createStandaloneServer, validateInput
- Re-exported modules: ./adapters/index.js, ./client/index.js, ./server/index.js, ./types.js

License

MIT

@philijs/ui - Type: Node package - Purpose: Official UI component library for PhilJS with dark mode, accessibility, and theming - Version: 0.1.0 - Location: packages/philijs-ui - Entry points: packages/philijs-ui/src/index.ts, ./theme, ./styles.css - Keywords: philijs, ui, components, design-system, accessibility

philijs-ui

Official UI component library for PhilJS with dark mode, accessibility, and theming.

Requirements

- **Node.js 24** or higher
- **TypeScript 6** or higher
- **ESM only** - CommonJS is not supported

Features

- **Pre-built Components** - 40+ accessible, production-ready components
- **Dark Mode** - Built-in theme switching with system preference detection

- **Fully Accessible** - WCAG 2.1 compliant with ARIA support
- **Customizable** - CSS variables and theme tokens
- **TypeScript** - Fully typed component props
- **Responsive** - Mobile-first design
- **Zero Dependencies** - Built with PhilJS signals
- **Tree-shakeable** - Import only what you need

Installation

```
pnpm add philjs-ui
```

Quick Start

Setup Theme Provider

```
import { ThemeProvider } from 'philjs-ui';
import 'philjs-ui/styles.css';

export default function App() {
  return (
    <ThemeProvider defaultTheme="light">
      <YourApp />
    </ThemeProvider>
  );
}
```

Use Components

```
import { Button, Input, Card } from 'philjs-ui';

export default function LoginForm() {
  return (
    <Card>
      <h2>Sign In</h2>
      <Input
        type="email"
        placeholder="Email"
        label="Email Address"
      />
      <Input
        type="password"
        placeholder="Password"
        label="Password"
      />
      <Button variant="primary" size="lg">
        Sign In
      </Button>
    </Card>
  );
}
```

Dark Mode

```
import { useTheme, Button } from 'philjs-ui';

export default function ThemeToggle() {
  const { theme, setTheme } = useTheme();

  return (
    <Button
      onClick={() => setTheme(theme() === 'dark' ? 'light' : 'dark')}
    >
      Toggle Theme
    </Button>
  );
}
```

Available Components

Layout

- Container, Grid, Flex, Stack, Spacer

Forms

- Input, Textarea, Select, Checkbox, Radio, Switch, Label

Buttons

- Button, IconButton, ButtonGroup

Display

- Card, Badge, Avatar, Tooltip, Modal, Drawer

Feedback

- Alert, Toast, Spinner, Progress, Skeleton

Navigation

- Tabs, Breadcrumbs, Pagination, Menu, Dropdown

Data Display

- Table, List, Accordion, Divider

Theming

Customize the theme with CSS variables:

```
:root {
  --color-primary: #3b82f6;
  --color-secondary: #8b5cf6;
  --color-success: #10b981;
  --color-warning: #f59e0b;
  --color-error: #ef4444;

  --radius-sm: 0.25rem;
  --radius-md: 0.5rem;
  --radius-lg: 1rem;

  --spacing-sm: 0.5rem;
  --spacing-md: 1rem;
  --spacing-lg: 2rem;
}
```

Documentation

For more information, see the PhilJS documentation.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `.. ./theme, ./styles.css`
- Source files: `packages/philjs-ui/src/index.ts`

Public API

- Direct exports: (none detected)
- Re-exported names: Accordion, AccordionButton, AccordionButtonProps, AccordionItem, AccordionItemProps, AccordionPanel, AccordionPanelProps, AccordionProps, Alert, AlertDescription, AlertProps, AlertStatus, AlertTitle, AlertVariant, Avatar, AvatarBadge, AvatarBadgeProps, AvatarGroup, AvatarGroupProps, AvatarProps, AvatarSize, Badge, BadgeColor, BadgeProps, BadgeSize, BadgeVariant, Breadcrumb, BreadcrumbIcons, BreadcrumbItem, BreadcrumbItemProps, BreadcrumbLink, BreadcrumbProps, BreadcrumbSeparator, BreadcrumbSeparatorProps, Button, ButtonColor, ButtonGroup, ButtonProps, ButtonSize, ButtonVariant, Card, CardBody, CardFooter, CardHeader, CardImage, CardProps, CardTitle, CardVariant, Checkbox, CheckboxGroup, CheckboxGroupProps, CheckboxProps, CheckboxSize, CircularProgress, CircularProgressProps, ConfirmDialog, ConfirmDialogProps, Drawer, DrawerBody, DrawerFooter, DrawerHeader, DrawerPlacement, DrawerProps, DrawerSize, Dropdown, DropdownDivider, DropdownItem, DropdownItemProps, DropdownLabel, DropdownPlacement, DropdownProps, IconButton, Input, InputProps, InputSize, InputVariant, Modal, ModalBody, ModalFooter, ModalHeader, ModalProps, ModalSize, MultiSelect, MultiSelectProps, NotificationBadge, NotificationBadgeProps, Popover, PopoverProps, Progress, ProgressColor, ProgressProps, ProgressSize, Radio, RadioGroup, RadioGroupProps, RadioProps, RadioSize, Select, SelectOption, SelectProps, SelectSize, Skeleton, SkeletonProps, Spinner, SpinnerProps, SpinnerSize, StatusIndicator, StatusIndicatorProps, StatusIndicatorStatus, Switch, SwitchProps, SwitchSize, Tab, TabList, TabListProps, TabPanel, TabPanelProps, TabPanelProps, TabPanelsProps, TabProps, Table, TableCaption, TableEmpty, TableProps, TableSize, TableVariant, Tabs, TabsProps, TabsSize, TabsVariant, Tbody, Td, TdProps, Textarea, TextareaProps, Tfoot, Th, ThProps, Thead, Theme, ThemeProvider, ToastContainer, ToastOptions, ToastPosition, ToastStatus, Tooltip, TooltipPlacement, TooltipProps, Tr, TrProps, borderRadius, boxShadow, breakpoints, colors, defaultTheme, fontFamily, fontSize, fontWeight, generateCSSVariables, spacing, toast, transition, useColorMode, useTheme, useToast, zIndex
- Re-exported modules: `./components/Accordion.js, ./components/Alert.js, ./components/Avatar.js, ./components/Badge.js, ./components/Breadcrumb.js, ./components/Button.js, ./components/Card.js, ./components/Checkbox.js, ./components/Drawer.js, ./components/Dropdown.js, ./components/Input.js, ./components/Modal.js, ./components/Radio.js, ./components/Select.js, ./components/Spinner.js, ./components/Switch.js, ./components/Table.js, ./components/Tabs.js, ./components/Toast.js, ./components/Tooltip.js, ./theme/ThemeProvider.js, ./theme/tokens.js`

License

MIT

@philjs/vector - Type: Node package - Purpose: Vector embeddings and RAG pipelines for PhilJS - semantic search, document chunking, hybrid search - Version: 0.1.0 - Location: packages/philjs-vector - Entry points: packages/philjs-vector/src/index.ts - Keywords: philjs, vector, embeddings, rag, semantic-search, openai, cohore, indexeddb

@philjs/vector

Vector embeddings and RAG pipelines for PhilJS - semantic search, document chunking, hybrid search

Overview

Vector embeddings and RAG pipelines for PhilJS - semantic search, document chunking, hybrid search

Focus Areas

- philjs, vector, embeddings, rag, semantic-search, openai, cohore, indexeddb

Entry Points

- `packages/philjs-vector/src/index.ts`

Quick Start

```
import { // Core classes
  RAGPipeline, // Document Loaders
  TextLoader, // Embedding providers
  OpenAIEmbeddings } from '@philjs/vector';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- // Core classes RAGPipeline
- // Document loaders TextLoader
- // Embedding providers OpenAIEmbeddings
- // Hooks useRAG
- // Types type Document
- ChunkOptions
- CohereEmbeddings
- DocumentLoader
- Embedding
- EmbeddingProvider
- EmbeddingProviderConfig
- IndexedDBVectorStore

Install

```
pnpm add @philjs/vector
```

Usage

```
import { // Core classes
  RAGPipeline, // Document Loaders
  TextLoader, // Embedding providers
  OpenAIEmbeddings } from '@philjs/vector';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-vector/src/index.ts

Public API

- Direct exports: // Core classes RAGPipeline, // Document loaders TextLoader, // Embedding providers OpenAIEmbeddings, // Hooks useRAG, // Types type Document, ChunkOptions, CohereEmbeddings, DocumentLoader, Embedding, EmbeddingProvider, EmbeddingProviderConfig, IndexedDBVectorStore, JSONLoader, LocalEmbeddings, MarkdownLoader, MemoryVectorStore, RAGConfig, RAGQuery, RAGResult, SearchResult, TextChunker, URLLoader, UseRAGResult, VectorStore, VectorStoreConfig, useEmbeddings
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/vector-store - Type: Node package - Purpose: High-performance vector database for PhilJS with WASM-powered semantic search - Version: 0.1.0 - Location: packages/philjs-vector-store - Entry points: packages/philjs-vector-store/src/index.ts - Keywords: vector-database, semantic-search, embeddings, wasm, hnsw, similarity-search, philjs, ai

@philjs/vector-store

High-performance vector database for PhilJS with WASM-powered semantic search.

Built on [VecStore](#) - "The SQLite of vector search".

Features

- **HNSW-based indexing** - Fast approximate nearest neighbor search
- **Multiple distance metrics** - Cosine, Euclidean, dot product, and more
- **Metadata filtering** - SQL-like filter expressions
- **Hybrid search** - Combine vector similarity with BM25 keyword matching
- **WASM-powered** - Runs entirely in the browser, no server required
- **TypeScript-first** - Full type definitions included

Installation

```
npm install @philjs/vector-store
# or
pnpm add @philjs/vector-store
```

Quick Start

```

import { VectorStore } from '@philjs/vector-store';

// Create a store with 384 dimensions (e.g., for sentence transformers)
const store = await VectorStore.create({ dimensions: 384 });

// Add vectors with metadata
await store.upsert('doc-1', embedding1, { title: 'Hello World', category: 'greeting' });
await store.upsert('doc-2', embedding2, { title: 'Goodbye', category: 'farewell' });

// Query for similar vectors
const results = await store.query(queryEmbedding, { k: 5 });
console.log(results);
// [
//   { id: 'doc-1', score: 0.95, metadata: { title: 'Hello World', ... } },
//   { id: 'doc-2', score: 0.82, metadata: { title: 'Goodbye', ... } },
// ]

// Query with metadata filter
const filtered = await store.query(queryEmbedding, {
  k: 5,
  filter: "category = 'greeting'"
});

```

API Reference

`VectorStore.create(config)`

Create a new vector store instance.

```

const store = await VectorStore.create({
  dimensions: 384,           // Required: vector dimensions
  metric: 'cosine',          // Optional: 'cosine' | 'euclidean' | 'dot' | etc.
  m: 16,                     // Optional: HNSW M parameter
  efConstruction: 200,       // Optional: HNSW ef_construction
  efSearch: 50,              // Optional: HNSW ef_search
});

```

`store.upsert(id, vector, metadata?)`

Insert or update a vector.

```

await store.upsert(
  'doc-1',
  [0.1, 0.2, 0.3, ...], // number[] or Float32Array
  { title: 'Document 1', author: 'John' } // optional metadata
);

```

`store.upsertBatch(items)`

Batch insert multiple vectors.

```

await store.upsertBatch([
  { id: 'doc-1', vector: embedding1, metadata: { title: 'First' } },
  { id: 'doc-2', vector: embedding2, metadata: { title: 'Second' } },
]);

```

`store.query(vector, options?)`

Search for similar vectors.

```

const results = await store.query(queryVector, {
  k: 10,                    // Number of results
  filter: "author = 'John'", // Metadata filter
  includeVectors: false,    // Include vectors in results
  includeMetadata: true,    // Include metadata in results
  minScore: 0.5,             // Minimum similarity score
});

```

`store.hybridSearch(vector, options?)`

Combine vector similarity with keyword matching.

```

const results = await store.hybridSearch(queryVector, {
  text: 'search query',
  vectorWeight: 0.7,
  keywordWeight: 0.3,
  k: 10,
});

```

`store.delete(id)`

Delete a vector by ID.

```
const deleted = await store.delete('doc-1');
```

`store.stats()`

Get store statistics.

```

const stats = await store.stats();
// { count: 1000, dimensions: 384, metric: 'cosine', memoryBytes: 1234567 }

```

Utility Functions

```
import {
  cosineSimilarity,
  euclideanDistance,
  normalizeVector
} from '@philjs/vector-store';

// Compute similarity between two vectors
const sim = cosineSimilarity(vectorA, vectorB);

// Compute distance
const dist = euclideanDistance(vectorA, vectorB);

// Normalize to unit length
const normalized = normalizeVector(vector);
```

Distance Metrics

Metric	Description	Best For
cosine	Cosine similarity (default)	Text embeddings
euclidean	Euclidean distance (L2)	General purpose
dot	Dot product	Normalized vectors
manhattan	Manhattan distance (L1)	Sparse vectors
hamming	Hamming distance	Binary vectors
jaccard	Jaccard similarity	Set comparison

Performance

VecStore is optimized for fast similarity search:

- 0.2ms search latency for 100K vectors
- Sub-linear scaling with HNSW indexing
- Efficient memory usage with quantization support

Use Cases

- **Semantic Search** - Find documents similar to a query
- **Recommendation Systems** - Find similar items
- **RAG (Retrieval-Augmented Generation)** - Retrieve context for LLMs
- **Image Search** - Find visually similar images
- **Anomaly Detection** - Find outliers in vector space

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-vector-store/src/index.ts

Public API

- Direct exports: DistanceMetric, HybridSearchOptions, QueryOptions, SearchResult, StoreStats, VectorMetadata, VectorStore, VectorStoreConfig, cosineSimilarity, euclideanDistance, initVectorStore, normalizeVector
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/video-chat - Type: Node package - Purpose: Full-featured video conferencing for PhilJS - virtual backgrounds, recording, transcription, breakout rooms - Version: 0.1.0
- Location: packages/philjs-video-chat - Entry points: packages/philjs-video-chat/src/index.ts - Keywords: philjs, video-chat, webrtc, conferencing, virtual-background, transcription, recording

@philjs/video-chat

Full-featured video conferencing for PhilJS applications. Includes multi-party video rooms, virtual backgrounds, noise suppression, recording, transcription, breakout rooms, and more.

Installation

```
npm install @philjs/video-chat
```

Requirements

- Nodejs >= 24
- Peer dependencies:
 - @philjs/core
 - @philjs/webrtc
- Browser with WebRTC support
- For virtual backgrounds: MediaPipe Selfie Segmentation (loaded dynamically)

Basic Usage

```
import { VideoRoom, useVideoRoom } from '@philjs/video-chat';

// Using the hook-based API
const {
  room,
  participants,
  localParticipant,
  join,
  leave,
  mute,
  unmute,
  startScreenShare
} = useVideoRoom({
  roomId: 'my-room-123',
  displayName: 'John Doe',
  signalingUrl: 'wss://your-signaling-server.com',
  enableRecording: true,
  enableTranscription: true
});

// Join the room
await join();

// Control your media
mute();
unmute();
startScreenShare();

// Send a chat message
sendMessage('Hello everyone!');

// Leave when done
await leave();
```

Virtual Backgrounds

```
import { VideoRoom, VirtualBackgroundProcessor } from '@philjs/video-chat';

const room = new VideoRoom({
  roomId: 'my-room',
  displayName: 'Jane',
  signalingUrl: 'wss://server.com',
  virtualBackground: {
    type: 'blur',
    blurStrength: 0.7,
    segmentationModel: 'mediapipe'
  }
});

// Change background during the call
room.setVirtualBackground({
  type: 'image',
  backgroundUrl: '/backgrounds/beach.jpg'
});

// Remove background (transparent)
room.setVirtualBackground({ type: 'remove' });
```

Audio Enhancement

```
import { AudioEnhancer } from '@philjs/video-chat';

const room = new VideoRoom({
  roomId: 'my-room',
  displayName: 'User',
  signalingUrl: 'wss://server.com',
  audioEnhancement: {
    noiseSuppression: true,
    echoCancellation: true,
    autoGainControl: true,
    noiseGate: true,
    noiseGateThreshold: -50,
    compressor: true,
    equalizer: [0, 0, 2, 4, 4, 2, 0, -2, -2, 0] // 10-band EQ
  }
});
```

Recording and Transcription

```

const room = new VideoRoom({
  roomId: 'meeting-001',
  displayName: 'Host',
  signalingUrl: 'wss://server.com',
  enableRecording: true,
  enableTranscription: true
});

await room.join();

// Start recording
await room.startRecording();

// Get live transcription
room.on('transcription', (segment) => {
  console.log(`${segment.speakerName}: ${segment.text}`);
});

// Stop recording and get the video blob
const recordingBlob = room.stopRecording();

// Get full transcription history
const transcript = room.getTranscription();

```

Breakout Rooms

```

// Create breakout rooms (host only)
const breakoutId = room.createBreakoutRoom('Discussion Group 1');

// Assign participants
room.assignToBreakoutRoom(participantId, breakoutId);

// Close all breakout rooms
room.closeBreakoutRooms();

```

API Reference

Classes

VideoRoom

Main video conferencing room class.

Constructor Options (RoomConfig): - roomId: string - Unique room identifier - displayName: string - Your display name - signalingUrl: string - WebSocket signaling server URL - iceServers?: RTCIceServer[] - STUN/TURN servers - maxParticipants?: number - Maximum participants allowed - enableRecording?: boolean - Enable recording capability - enableTranscription?: boolean - Enable live transcription - enableChat?: boolean - Enable in-room chat - virtualBackground?: VirtualBackgroundConfig - Virtual background settings - audioEnhancement?: AudioEnhancementConfig - Audio processing settings

Methods: - join(): Promise<Participant> - Join the room - leave(): Promise<void> - Leave the room - mute() / unmute() - Control microphone - hideVideo() / showVideo() - Control camera - startScreenShare() / stopScreenShare() - Screen sharing - sendChatMessage(text: string) - Send chat message - sendFile(file: File) - Share a file - react(emoji: string) - Send reaction - raiseHand() / lowerHand() - Hand raising - setLayout(mode: LayoutMode) - Change video layout - pinParticipant(id) / unpinParticipant() - Pin a participant - startRecording() / stopRecording() - Recording controls - createBreakoutRoom(name) - Create breakout room - setVirtualBackground(config) - Change background

Events: - joined - You joined the room - left - You left the room - participantJoined - A participant joined - participantLeft - A participant left - chatMessage - New chat message - transcription - Transcription segment - recordingStarted / recordingStopped - screenShareStarted / screenShareStopped - layoutChanged

Participant

Represents a participant in the room.

Properties: - id: string - Unique identifier - name: string - Display name - isLocal: boolean - Is this the local user - stream: MediaStream | null - Video/audio stream - screenStream: MediaStream | null - Screen share stream

Methods: - mute() / unmute() - hideVideo() / showVideo() - raiseHand() / lowerHand() - react(emoji: string) - getState(): ParticipantState

VirtualBackgroundProcessor

Handles virtual background processing.

AudioEnhancer

Handles audio processing and enhancement.

VideoGridLayout

Pre-built video grid layout component.

Hooks

- useVideoRoom(config: RoomConfig) - Main hook for video room functionality
- useParticipant(participant: Participant) - Get participant state
- useActiveSpeaker(room: VideoRoom) - Get the current active speaker
- useVirtualBackground(config?) - Manage virtual background settings

Types

- RoomConfig - Room configuration options
- VirtualBackgroundConfig - Virtual background settings
- AudioEnhancementConfig - Audio enhancement settings

- ParticipantState - Participant status and metadata
- ChatMessage - Chat message structure
- TranscriptionSegment - Transcription data
- LayoutMode - 'grid' | 'speaker' | 'sidebar' | 'spotlight' | 'presentation'
- VideoGridConfig - Grid layout configuration

Layout Modes

- **grid** - Equal-sized video tiles in a grid
- **speaker** - Active speaker large, others small
- **sidebar** - Main video with sidebar of participants
- **spotlight** - Single large video
- **presentation** - Screen share focused layout

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-video-chat/src/index.ts

Public API

- Direct exports: // Core classes VideoRoom, // Hooks useVideoRoom, // Types type RoomConfig, AudioEnhancementConfig, AudioEnhancer, ChatMessage, LayoutMode, Participant, ParticipantState, TranscriptionSegment, UseVideoRoomResult, VideoGridConfig, VideoGridLayout, VirtualBackgroundConfig, VirtualBackgroundProcessor, useActiveSpeaker, useParticipant, useVirtualBackground
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/virtual - Type: Node package - Purpose: High-performance list virtualization for PhilJS - render millions of items efficiently - Version: 0.1.0 - Location: packages/philjs-virtual - Entry points: packages/philjs-virtual/src/index.ts - Keywords: virtualization, virtual-list, virtual-scroll, windowing, philjs

@philjs/virtual

High-performance list virtualization for PhilJS - render millions of items efficiently

Overview

High-performance list virtualization for PhilJS - render millions of items efficiently

Focus Areas

- virtualization, virtual-list, virtual-scroll, windowing, philjs

Entry Points

- packages/philjs-virtual/src/index.ts

Quick Start

```
import { DEFAULT_OVERSCAN, DEFAULT_SCROLL_DEBOUNCE, ScrollToOptions } from '@philjs/virtual';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- DEFAULT_OVERSCAN
- DEFAULT_SCROLL_DEBOUNCE
- ScrollToOptions
- VirtualGrid
- VirtualGridProps
- VirtualItem
- VirtualList
- VirtualListProps
- Virtualizer
- VirtualizerOptions
- calculateVisibleRange
- createSmoothScroller

Install

```
pnpm add @philjs/virtual
```

Usage

```
import { DEFAULT_OVERSCAN, DEFAULT_SCROLL_DEBOUNCE, ScrollToOptions } from '@philjs/virtual';
```

Scripts

- pnpm run build

- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-virtual/src/index.ts

Public API

- Direct exports: DEFAULT_OVERSCAN, DEFAULT_SCROLL_DEBOUNCE, ScrollToOptions, VirtualGrid, VirtualGridProps, VirtualItem, VirtualList, VirtualListProps, Virtualizer, VirtualizerOptions, calculateVisibleRange, createSmoothScroller, createVirtualizer, createWindowScroller, findIndexAtOffset, useVirtualizer, useWindowVirtualizer
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/voice - Type: Node package - Purpose: Voice UI primitives for PhilJS - speech recognition, synthesis, and voice commands - Version: 0.1.0 - Location: packages/philjs-voice - Entry points: packages/philjs-voice/src/index.ts - Keywords: philjs, voice, speech, recognition, synthesis, voice-ui, accessibility

@philjs/voice

Voice UI primitives for PhilJS - speech recognition, synthesis, and voice commands

Overview

Voice UI primitives for PhilJS - speech recognition, synthesis, and voice commands

Focus Areas

- philjs, voice, speech, recognition, synthesis, voice-ui, accessibility

Entry Points

- packages/philjs-voice/src/index.ts

Quick Start

```
import { ConversationTurn, IntentParser, SpeechRecognitionEngine } from '@philjs/voice';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- ConversationTurn
- IntentParser
- SpeechRecognitionEngine
- SpeechResult
- SpeechSynthesisEngine
- VoiceAssistant
- VoiceAssistantConfig
- VoiceCommand
- VoiceCommandSystem
- VoiceConfig
- VoiceMatch
- VoiceNavigation

Install

```
pnpm add @philjs/voice
```

Usage

```
import { ConversationTurn, IntentParser, SpeechRecognitionEngine } from '@philjs/voice';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-voice/src/index.ts

Public API

- Direct exports: ConversationTurn, IntentParser, SpeechRecognitionEngine, SpeechResult, SpeechSynthesisEngine, VoiceAssistant, VoiceAssistantConfig, VoiceCommand, VoiceCommandSystem, VoiceConfig, VoiceMatch, VoiceNavigation, VoiceState, useSpeechRecognition, useSpeechSynthesis, useVoiceAssistant, useVoiceCommands, useVoiceNavigation
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/vscode - Type: Node package - Purpose: Official VS Code extension for PhilJS - snippets, IntelliSense, and tooling - Version: 0.1.0 - Location: packages/philjs-vscode -
Entry points: . - Keywords: philjs, javascript, typescript, jsx, tsx, signals, reactive

PhilJS for VS Code

Official VS Code extension for PhilJS - Snippets, IntelliSense, and tooling.

Installation

Install from the VS Code Marketplace:

1. Open VS Code
2. Go to Extensions (Ctrl+Shift+X / Cmd+Shift+X)
3. Search for "PhilJS"
4. Click Install

Or install from command line:

```
code --install-extension philjs.philjs-vscode
```

Features

IntelliSense

Smart code completion for PhilJS APIs:

- Signal creation and methods
- Memo and effect functions
- Router hooks and components
- SSR utilities
- Component props

Snippets

Quick code generation with intelligent snippets:

Trigger	Description
psignal	Create a signal
pmemo	Create a memo
peffect	Create an effect
pcomponent	Create a component
proute	Create a route component
ploader	Create a route loader
papi	Create an API route
pisland	Create an island component

Commands

Access PhilJS commands from the Command Palette (Ctrl+Shift+P / Cmd+Shift+P):

- **PhilJS: Create Component** - Generate a new component
- **PhilJS: Create Route** - Generate a new route
- **PhilJS: Create Page** - Generate a new page
- **PhilJS: Create Hook** - Generate a custom hook
- **PhilJS: Create Store** - Generate a state store
- **PhilJS: Open DevTools** - Open PhilJS DevTools

Signal Highlighting

Visual indicators for signal usage in your code:

- Signal declarations highlighted
- Signal reads highlighted differently from writes
- Memo and effect dependencies tracked

Diagnostics

Real-time error detection:

- Unused signals
- Missing effect cleanup
- Invalid signal usage
- Type errors in PhilJS APIs

Go to Definition

Navigate to signal and component definitions with F12 or Ctrl+Click.

Hover Information

View documentation and type information on hover.

Usage

Creating Components

1. Open Command Palette (Ctrl+Shift+P)
2. Type "PhilJS: Create Component"
3. Enter component name
4. Choose options (TypeScript, with tests, with styles)

Or use the snippet:

```
pcomponent<Tab>
```

Expands to:

```
import { signal } from 'philjs-core';

export function ComponentName() {
  return (
    <div>
      /* Component content */
    </div>
  );
}
```

Creating Signals

Type psignal and press Tab:

```
const name = signal(initialValue);
```

Creating Effects

Type peffect and press Tab:

```
effect(() => {
  // Effect body

  return () => {
    // Cleanup
  };
});
```

Creating Routes

1. Command Palette > "PhilJS: Create Route"
2. Enter route name
3. Auto-generates route file with loader

```
// src/routes/users.tsx
import { RouteLoader } from 'philjs-router';

export const loader: RouteLoader = async ({ params, request }) => {
  const users = await fetchUsers();
  return { users };
};

export default function UsersPage({ data }) {
  return (
    <div>
      <h1>Users</h1>
      <ul>
        {data.users.map(user => (
          <li key={user.id}>{user.name}</li>
        ))}
      </ul>
    </div>
  );
}
```

Configuration

Configure the extension in VS Code settings:

```
{
  // Enable/disable PhilJS IntelliSense
  "philjs.enableIntelliSense": true,

  // Highlight signal usage
  "philjs.signalHighlighting": true,

  // Default directory for components
  "philjs.componentDirectory": "src/components",

  // Default directory for routes
  "philjs.routesDirectory": "src/routes",

  // Format on save with PhilJS formatter
  "philjs.formatOnSave": true,

  // Enable experimental features
  "philjs.experimental": false
}
```

Snippets Reference

Core Snippets

Signal:

```
psignal<Tab>
// Creates:
const name = signal(initialValue);
```

Memo:

```
pmemo<Tab>
// Creates:
const computed = memo(() => {
  return calculation;
});
```

Effect:

```
peffect<Tab>
// Creates:
effect(() => {
  // Effect logic

  return () => {
    // Cleanup
  };
});
```

Component Snippets

Component:

```
pcomponent<Tab>
// Creates:
export function ComponentName() {
  return (
    <div>
      {/* Component content */}
    </div>
  );
}
```

Island Component:

```
pisland<Tab>
// Creates:
import { island } from 'philjs-islands';

export default island(function ComponentName() {
  const state = signal('');

  return (
    <div>
      {/* Interactive island content */}
    </div>
  );
});
```

Router Snippets

Route:

```

proute<Tab>
// Creates:
export default function RouteName() {
  return (
    <div>
      {/* Route content */}
    </div>
  );
}

```

Loader:

```

ploader<Tab>
// Creates:
export const loader: RouteLoader = async ({ params, request }) => {
  // Load data
  return { data };
};

```

API Route:

```

papi<Tab>
// Creates:
export async function GET(request: Request) {
  return new Response(JSON.stringify({ data }), {
    headers: { 'Content-Type': 'application/json' }
  });
}

```

Keyboard Shortcuts

Shortcut	Command
Ctrl+Shift+C	Create Component
Ctrl+Shift+R	Create Route
Ctrl+Shift+H	Create Hook
F12	Go to Definition
Shift+F12	Find References

Troubleshooting

IntelliSense Not Working

1. Ensure TypeScript version >= 5.0
2. Restart VS Code
3. Run "TypeScript: Restart TS Server" from Command Palette

Snippets Not Triggering

1. Check that snippet suggestions are enabled in settings
2. Verify file extension is .tsx or .ts
3. Try typing trigger and pressing Ctrl+Space

Commands Not Appearing

1. Ensure extension is activated
2. Check Output panel for errors (Help > Toggle Developer Tools)
3. Reinstall the extension

Contributing

Found a bug or want to suggest a feature? Open an issue on the [PhilJS repository](#).

Documentation

For more information about PhilJS, see the PhilJS documentation.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: (none detected)

Public API

- Direct exports: (none detected)
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

```
## @philjs/wasm - Type: Node package - Purpose: Rust/WebAssembly integration for PhilJS with reactive signals - Version: 0.1.0 - Location: packages/philjs-wasm - Entry points: packages/philjs-wasm/src/index.ts, ./vite, packages/philjs-wasm/src/codegen.ts, packages/philjs-wasm/src/cl.ts - Keywords: philjs, wasm, webassembly, rust, wasm-bindgen, signals,
```

reactivity, component, codegen, bindings

philjs-wasm

Rust/WebAssembly integration for PhilJS with reactive signals. This package makes PhilJS the best choice for Rust developers wanting to use WASM in their web applications.

Features

- **Seamless WASM Loading** - Load WASM modules with streaming compilation and caching
- **Reactive Rust Signals** - Create signals that can be read/written from both Rust and JavaScript
- **Component Integration** - Use WASM modules as PhilJS components
- **Vite Plugin** - First-class Vite integration for WASM loading and HMR
- **Type Safety** - Full TypeScript support with Rust type mappings
- **Memory Management** - Automatic memory allocation/deallocation helpers

Installation

```
npm install philjs-wasm philjs-core
```

Quick Start

Loading a WASM Module

```
import { loadWasm, bindRustFunctions } from 'philjs-wasm';

// Load a WASM module
const module = await loadWasm('/my-module.wasm');

// Bind all exported Rust functions
const rust = bindRustFunctions(module);

// Call Rust functions from JavaScript
const result = rust.calculateSum(1, 2, 3);
```

Using the useWasm Hook

```
import { useWasm } from 'philjs-wasm';

function MyComponent() {
  const wasm = useWasm('/crypto.wasm');

  if (wasm.loading) {
    return <div>Loading WASM...</div>;
  }

  if (wasm.error) {
    return <div>Error: {wasm.error.message}</div>;
  }

  const hash = (wasm.data!.exports.sha256 as Function)('hello');
  return <div>Hash: {hash}</div>;
}
```

Creating WASM Components

```
import { createWasmComponent } from 'philjs-wasm';

// Create a component from a WASM module
const Counter = await createWasmComponent('/counter.wasm', {
  renderFn: 'render_counter',
  props: { initialCount: 0 }
});

// Render the component
const html = Counter.render({ count: 5 });
```

Shared Rust Signals

Create signals that can be read/written from both Rust and JavaScript:

```

import { loadWasm, createI32Signal, createF64Signal } from 'philjs-wasm';

const module = await loadWasm('/counter.wasm');

// Create a shared i32 signal
const count = createI32Signal(module, 0);

// Use from JavaScript
console.log(count()); // 0
count.set(5);

// Pass pointer to Rust for direct memory access
// const ptr = count.ptr();
// module.exports.increment_counter(ptr);

// Sync value from Rust memory
count.syncFromRust();
console.log(count()); // Value updated by Rust

```

WasmProvider

For application-wide WASM module management:

```

import { WasmProvider, getWasmContext } from 'philjs-wasm';

// In your app root
function App() {
  return (
    <WasmProvider preload={['/crypto.wasm', '/image.wasm']}>
      <MyApp />
    </WasmProvider>
  );
}

// In any component
function MyComponent() {
  const ctx = getWasmContext();
  const module = ctx.getModule('/crypto.wasm');
  // ...
}

```

Vite Plugin

Add the Vite plugin for seamless WASM integration:

```

// vite.config.ts
import { defineConfig } from 'vite';
import { viteWasmPlugin } from 'philjs-wasm/vite';

export default defineConfig({
  plugins: [
    viteWasmPlugin({
      wasmDir: 'src/wasm',
      generateTypes: true,
      streaming: true,
      cache: true,
      hmr: true, // Enable HMR for WASM in development
      optimize: {
        wasmOpt: true,
        level: 'Os'
      }
    })
  ]
});

```

API Reference

Loading Functions

```
loadWasm(url, options?)
```

Load a WASM module from a URL.

```

const module = await loadWasm('/module.wasm', {
  cache: true,           // Cache the loaded module
  timeout: 30000,         // Loading timeout in ms
  initFn: 'init',         // Initialization function to call
  imports: {}             // Custom import object
});

```

```
unloadWasm(url)
```

Unload a cached WASM module.

```
isWasmLoaded(url)
```

Check if a WASM module is loaded.

```
getWasmModule(url)
```

Get a loaded WASM module.

Component Functions

```
createWasmComponent(url, options?)
```

Create a PhilJS component from a WASM module.

```
const component = await createWasmComponent('/component.wasm', {
  renderFn: 'render',
  props: { /* initial props */ }
});

component.render({ /* new props */ });
component.update({ /* partial props */ });
component.dispose();
```

Hook Functions

```
useWasm(url, options?)
```

Hook to load and use WASM modules with reactive state.

```
const { data, loading, error, reload } = useWasm('/module.wasm');
```

Binding Functions

```
bindRustFunctions(module, options?)
```

Bind Rust functions from a WASM module to JavaScript.

```
const rust = bindRustFunctions(module, {
  include: ['add', 'multiply'], // Only bind these functions
  exclude: ['internal_fn'], // Exclude these functions
  transformNames: snakeToCamel, // Transform function names
  wrapErrors: true // Wrap errors with context
});
```

Signal Functions

```
createRustSignal(options)
```

Create a signal that can be shared between Rust and JavaScript.

```
const signal = createRustSignal({
  module,
  initialValue: 0,
  serialize: (v) => /* ... */,
  deserialize: (data) => /* ... */,
  byteSize: 4
});
```

Specialized Signal Creators

```
// Integer signals
const i32Signal = createI32Signal(module, 0);
const i64Signal = createI64Signal(module, 0n);

// Float signals
const f32Signal = createF32Signal(module, 0.0);
const f64Signal = createF64Signal(module, 0.0);

// Boolean signal
const boolSignal = createBoolSignal(module, false);
```

Context Functions

```
initWasmProvider()
```

Initialize the WASM provider context.

```
getWasmContext()
```

Get the current WASM context.

Rust Type Utilities

```

import { Ok, Err, Some, None, unwrapResult, unwrapOption } from 'philjs-wasm';

// Result type
type RustResult<T, E> = { ok: true; value: T } | { ok: false; error: E };

const result: RustResult<number, string> = Ok(42);
const value = unwrapResult(result); // 42

// Option type
type RustOption<T> = { some: true; value: T } | { some: false };

const option: RustOption<string> = Some('hello');
const str = unwrapOption(option); // 'hello'

```

Rust Side Integration

Example Rust code for use with philjs-wasm:

```

use wasm_bindgen::prelude::*;

#[wasm_bindgen]
pub fn add(a: i32, b: i32) -> i32 {
    a + b
}

#[wasm_bindgen]
pub fn render(props_ptr: *const u8, props_len: usize) -> *mut u8 {
    // Read props JSON from JavaScript
    let props_slice = unsafe {
        std::slice::from_raw_parts(props_ptr, props_len)
    };
    let props_json = std::str::from_utf8(props_slice).unwrap();

    // Generate HTML
    let html = format!(<div>Hello from Rust!</div>);

    // Return HTML with length prefix
    let mut result = Vec::with_capacity(4 + html.len());
    result.extend_from_slice(&(html.len() as u32).to_le_bytes());
    result.extend_from_slice(html.as_bytes());

    let ptr = result.as_mut_ptr();
    std::mem::forget(result);
    ptr
}

// Shared memory signal access
#[wasm_bindgen]
pub fn increment_counter(ptr: *mut i32) {
    unsafe {
        *ptr += 1;
    }
}

```

TypeScript Types

```

import type {
    WasmModule,
    WasmExports,
    WasmLoadOptions,
    WasmComponentOptions,
    WasmComponent,
    BoundRustFunction,
    BindOptions,
    RustSignal,
    RustSignalOptions,
    WasmContextValue,
    WasmMemoryManager,
    RustResult,
    RustOption
} from 'philjs-wasm';

```

Memory Management

The package includes a memory manager for safe WASM memory operations:

```

const ctx = getWasmContext();
const { memory } = ctx;

// Allocate memory
const ptr = memory.alloc(module, 1024);

// Copy string to WASM memory
const { ptr: strPtr, len } = memory.copyString(module, 'Hello');

// Read string from WASM memory
const str = memory.readString(module, strPtr, len);

// Copy typed array
const { ptr: arrPtr, len: arrLen } = memory.copyTypedArray(module, new Float32Array([1, 2, 3]));

// Read typed array
const arr = memory.readTypedArray(module, arrPtr, arrLen, Float32Array);

// Free memory
memory.free(module, ptr, 1024);

```

Best Practices

1. **Preload WASM modules** - Use WasmProvider with preload for critical modules
2. **Use caching** - Enable caching to avoid reloading modules
3. **Handle errors** - Always handle loading errors gracefully
4. **Clean up** - Call dispose() on components when done
5. **Use typed signals** - Use specialized signal creators for better performance
6. **Sync carefully** - Call syncFromRust() after Rust modifies shared memory

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./vite`, `./codegen`, `./cli`
- Source files: `packages/philjs-wasm/src/index.ts`, `packages/philjs-wasm/src/codegen.ts`, `packages/philjs-wasm/src/cli.ts`

Public API

- Direct exports: `BindOptions`, `BoundFunctions`, `BoundRustFunction`, `CodegenCLIOptions`, `CodegenOptions`, `CodegenResult`, `Err`, `None`, `Ok`, `RustEnum`, `RustEnumVariant`, `RustField`, `RustFunction`, `RustImpl`, `RustModule`, `RustOption`, `RustParam`, `RustResult`, `RustSignal`, `RustSignalOptions`, `RustStruct`, `RustToJSMapper`, `RustType`, `Some`, `WasmBindgenAttrs`, `WasmComponent`, `WasmComponentOptions`, `WasmContextValue`, `WasmExports`, `WasmLoadOptions`, `WasmMemoryManager`, `WasmModule`, `WasmProvider`, `WasmProviderProps`, `WasmResourceState`, `WbFree`, `WbMalloc`, `WbRealloc`, `bindRustFunctions`, `createBoolSignal`, `createF32Signal`, `createF64Signal`, `createI32Signal`, `createI64Signal`, `createRustSignal`, `createWasmComponent`, `generateBindings`, `generateCommand`, `generateComponentBinding`, `generateJSWrapper`, `generateSignalBinding`, `generateTypeScriptTypes`, `getWasmContext`, `getWasmModule`, `initWasmProvider`, `isWasmLoaded`, `loadWasm`, `main`, `parseArgs`, `parseRustModule`, `parseRustType`, `showHelp`, `showVersion`, `unloadWasm`, `unwrapOption`, `unwrapResult`, `useWasm`, `watchCommand`
- Re-exported names: `// Code generation functions generateTypeScriptTypes`, `// Parser functions parseRustType`, `// Type definitions type RustType`, `// Type mapper RustToJSMapper`, `CodegenCLIOptions`, `CodegenOptions`, `CodegenResult`, `RustEnum`, `RustEnumVariant`, `RustField`, `RustFunction`, `RustImpl`, `RustModule`, `RustParam`, `RustStruct`, `ViteWasmPluginOptions`, `WasmBindgenAttrs`, `generateBindings`, `generateComponentBinding`, `generateJSWrapper`, `generateSignalBinding`, `parseRustModule`, `viteWasmPlugin`
- Re-exported modules: `./codegen.js`, `./vite-plugin.js`

License

MIT

@philjs/webgpu - Type: Node package - Purpose: WebGPU Integration - GPU-accelerated rendering, compute shaders, parallel DOM diffing, and real-time effects - Version: 0.1.0 - Location: packages/philjs-webgpu - Entry points: packages/philjs-webgpu/src/index.ts - Keywords: philjs, webgpu, gpu, compute-shaders, rendering, graphics, wgsl, parallel, high-performance

@philjs/webgpu

WebGPU Integration for PhilJS - Leverage next-generation GPU APIs for high-performance rendering, compute shaders, parallel DOM diffing, and real-time effects.

Installation

```
npm install @philjs/webgpu
```

Requirements

- Node.js >= 24
- Browser with WebGPU support (Chrome 113+, Edge 113+, Firefox Nightly)
- TypeScript 6.x (for development)
- `@webgpu/types` for TypeScript type definitions

Basic Usage

```

import {
  initWebGPU,
  useWebGPU,
  isWebGPUSupported,
  WebGPUContext
} from '@philjs/webgpu';

// Check for WebGPU support
if (await isWebGPUSupported()) {
  // Initialize WebGPU
  const ctx = await initWebGPU({
    powerPreference: 'high-performance',
    enableCompute: true
  });

  if (ctx) {
    const device = ctx.getDevice();
    console.log('WebGPU initialized!');
  }
}

// Use the hook
function MyComponent() {
  const { supported, context, device } = useWebGPU();

  if (!supported) {
    return <div>WebGPU is not supported</div>;
  }

  // Use the GPU device
}

```

GPU Canvas Rendering

```

import { GPUCanvas, useGPUCanvas, BuiltInShaders } from '@philjs/webgpu';

const canvas = document.getElementById('canvas') as HTMLCanvasElement;
const gpuCanvas = new GPUCanvas(canvas, {
  powerPreference: 'high-performance',
  antialias: true
});

await gpuCanvas.initialize();

// Set up frame callback
gpuCanvas.onFrame((deltaTime) => {
  const ctx = gpuCanvas.getContext();
  const result = ctx.beginRenderPass();
  if (result) {
    const { encoder, pass } = result;
    // Render your content
    pass.end();
    ctx.submit(encoder);
  }
});

gpuCanvas.start();

// Or use the hook
const { initialize, context, start, stop, onFrame } = useGPUCanvas(canvasElement);
await initialize();
onFrame((dt) => { /* render */ });
start();

```

Compute Shaders

```

import { WebGPUContext, BuiltInShaders } from '@philjs/webgpu';

const ctx = new WebGPUContext({ enableCompute: true });
await ctx.initialize();

// Create a compute pipeline
const shaderModule = ctx.createShaderModule(BuiltInShaders.parallelSum, 'sum');
const pipeline = ctx.createComputePipeline({
  layout: 'auto',
  compute: {
    module: shaderModule,
    entryPoint: 'main'
  }
});

// Create buffers
const inputData = new Float32Array([1, 2, 3, 4, 5, 6, 7, 8]);
const inputBuffer = ctx.createBuffer(inputData, GPUBufferUsage.STORAGE);

// Run compute pass
const result = ctx.beginComputePass();
if (result) {
  const { encoder, pass } = result;
  pass.setPipeline(pipeline);
  // Set bind groups...
  pass.dispatchWorkgroups(inputData.length / 256);
  pass.end();
  ctx.submit(encoder);
}

```

GPU Effects

```

import { GPUEffects, WebGPUContext } from '@philjs/webgpu';

const ctx = new WebGPUContext();
await ctx.initialize();

const effects = new GPUEffects(ctx);
await effects.initialize();

// Apply blur effect
const inputTexture = ctx.createTexture(1920, 1080, 'rgba8unorm');
const outputTexture = ctx.createTexture(1920, 1080, 'rgba8unorm');

effects.blur(inputTexture, outputTexture, 1920, 1080);

```

GPU Animations

```

import { GPUAnimator, WebGPUContext } from '@philjs/webgpu';

const ctx = new WebGPUContext();
await ctx.initialize();

const animator = new GPUAnimator(ctx);
await animator.initialize();

// Add animations
animator.addAnimation({
  id: 'fade-in',
  duration: 1000,
  easing: (t) => t * t, // ease-in
  update: (progress) => new Float32Array([progress]) // opacity
});

// Update all animations
const results = animator.update(performance.now());
const opacity = results.get('fade-in')?.[0];

```

Parallel DOM Diffing

```

import { GPUDiffer, WebGPUContext } from '@philjs/webgpu';

const ctx = new WebGPUContext({ enableCompute: true });
await ctx.initialize();

const differ = new GPUDiffer(ctx);
await differ.initialize();

const oldTree = [
  { type: 'div', children: ['Hello'] },
  { type: 'span', children: ['World'] }
];

const newTree = [
  { type: 'div', children: ['Hello', 'There'] },
  { type: 'p', children: ['World'] }
];

const { patches, computeTime } = differ.diff(oldTree, newTree);
console.log(`Diff computed in ${computeTime}ms`);
console.log(`Patches:`, patches);

```

API Reference

Initialization

- `initWebGPU(config?: WebGPUConfig): Promise<WebGPUContext | null>` - Initialize WebGPU globally
- `getWebGPUContext(): WebGPUContext | null` - Get the global context
- `isWebGPUSupported(): Promise<boolean>` - Async check for WebGPU support
- `isWebGPUSupportedSync(): boolean` - Sync check (less accurate)

Hooks

- `useWebGPU()` - Get WebGPU support status and context
- `useGPUCanvas(canvas, config?)` - Create a GPU-accelerated canvas
- `useGPUEffects()` - Access GPU effects (blur, etc.)
- `useGPUAnimator()` - Create a GPU animator instance

Classes

WebGPUContext

Core WebGPU context manager.

- Methods:**
- `initialize(canvas?: boolean): Promise<boolean>` - Initialize WebGPU - `getDevice(): GPUDevice | null` - Get the GPU device - `getContext(): GPUCanvasContext | null` - Get canvas context - `getFormat(): GPUTextureFormat` - Get preferred texture format - `createBuffer(data, usage, label?)` - Create a GPU buffer - `getBuffer(label)` - Get cached buffer by label - `createTexture(width, height, format?, usage?)` - Create a texture - `createShaderModule(code, label?)` - Create shader module - `createRenderPipeline(descriptor, label?)` - Create render pipeline - `createComputePipeline(descriptor, label?)` - Create compute pipeline - `beginRenderPass(colorAttachments?)` - Start render pass - `beginComputePass()` - Start compute pass - `submit(encoder)` - Submit command buffer - `destroy()` - Clean up resources

GPUCanvas

GPU-accelerated canvas component.

- Methods:**
- `initialize(): Promise<boolean>` - `getContext(): WebGPUContext` - `onFrame(callback: (dt: number) => void)` - `start() / stop()` - `destroy()`

GPUEffects

GPU-powered image effects.

- Methods:**
- `initialize(): Promise<void>` - `blur(input, output, width, height)`

GPUAnimator

GPU-accelerated animation system.

- Methods:**
- `initialize(): Promise<void>` - `addAnimation(animation: GPUAnimation)` - `removeAnimation(id: string)` - `update(currentTime): Map<string, Float32Array>`

GPUDiffer

Parallel DOM diffing using compute shaders.

- Methods:**
- `initialize(): Promise<void>` - `diff(oldTree, newTree): ParallelDiffResult`

Built-in Shaders

```

import { BuiltInShaders } from '@philjs/webgpu';

BuiltInShaders.basic2D      // Simple 2D colored vertices
BuiltInShaders.texturedQuad // Textured quad rendering
BuiltInShaders.blur         // Gaussian blur compute shader
BuiltInShaders.parallelSum // Parallel reduction
BuiltInShaders.animate      // Animation interpolation

```

Configuration

```

interface WebGPUConfig {
  canvas?: HTMLCanvasElement | string;
  powerPreference?: 'low-power' | 'high-performance';
  enableCompute?: boolean;
  antialias?: boolean;
  alphaMode?: 'opaque' | 'premultiplied';
  maxBufferSize?: number; // bytes, default 256MB
}

```

Types

- GPURenderingContext - Render pass context with helper methods
- GPUComputeContext - Compute pass context with dispatch helper
- ShaderModule - Vertex/fragment/compute shader code
- GPUAnimation - Animation definition with easing
- ParallelDiffResult - Diff result with patches and timing
- DOMPatch - Single DOM operation (insert/remove/update/move)

WGSL Shader Example

```

// Custom vertex shader
struct VertexOutput {
  @builtin(position) position: vec4f,
  @location(0) color: vec4f,
};

@vertex
fn vertexMain(@location(0) pos: vec2f) -> VertexOutput {
  var output: VertexOutput;
  output.position = vec4f(pos, 0.0, 1.0);
  output.color = vec4f(1.0, 0.0, 0.0, 1.0);
  return output;
}

@fragment
fn fragmentMain(input: VertexOutput) -> @location(0) vec4f {
  return input.color;
}

```

Browser Support

WebGPU is currently supported in: - Chrome 113+ (enabled by default) - Edge 113+ (enabled by default) - Firefox Nightly (behind flag) - Safari Technology Preview (partial)
Use `isWebGPUSupported()` to check availability and provide fallbacks.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: `packages/philjs-webgpu/src/index.ts`

Public API

- Direct exports: `BuiltinShaders`, `DOMPatch`, `GPUAnimation`, `GPUAnimator`, `GPUCanvas`, `GPUComponentProps`, `GPUComputeContext`, `GPU Differ`, `GPUEffects`, `GPURenderingContext`, `ParallelDiffResult`, `ShaderModule`, `WebGPUConfig`, `WebGPUContext`, `getWebGPUContext`, `initWebGPU`, `isWebGPUSupported`, `isWebGPUSupportedSync`, `useGPUAnimator`, `useGPUCanvas`, `useGPUEffects`, `useWebGPU`
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/webrtc - Type: Node package - Purpose: Full-Featured WebRTC Framework for PhilJS - Version: 0.1.0 - Location: packages/philjs-webrtc - Entry points: packages/philjs-webrtc/src/index.ts - Keywords: philjs, webrtc, peer-to-peer, real-time, video-chat

@philjs/webrtc

Full-Featured WebRTC Framework for PhilJS

Overview

Full-Featured WebRTC Framework for PhilJS

Focus Areas

- philjs, webrtc, peer-to-peer, real-time, video-chat

Entry Points

- `packages/philjs-webrtc/src/index.ts`

Quick Start

```

import { ChunkedDataChannel1, DEFAULT_ICE_SERVERS, DataChannelConfig } from '@philjs/webrtc';

```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- ChunkedDataChannel
- DEFAULT_ICE_SERVERS
- DataChannelConfig
- DataChannelMessage
- NetworkQuality
- NetworkQualityMonitor
- PeerConnection
- PeerConnectionOptions
- PeerStats
- RTCConfig
- RTCRoom
- SignalMessage

Install

```
pnpm add @philjs/webrtc
```

Usage

```
import { ChunkedDataChannel, DEFAULT_ICE_SERVERS, DataChannelConfig } from '@philjs/webrtc';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-webrtc/src/index.ts

Public API

- Direct exports: ChunkedDataChannel, DEFAULT_ICE_SERVERS, DataChannelConfig, DataChannelMessage, NetworkQuality, NetworkQualityMonitor, PeerConnection, PeerConnectionOptions, PeerStats, RTCConfig, RTCRoom, SignalMessage, SignalingClient, SignalingConfig, SignalingHandlers, useDataChannel, useNetworkQuality, usePeerConnection, useWebRTC
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/workers - Type: Node package - Purpose: Thread pool Web Workers for PhilJS - parallel computing, shared state, task queuing - Version: 0.1.0 - Location: packages/philjs-workers - Entry points: packages/philjs-workers/src/index.ts - Keywords: philjs, workers, web-workers, parallel, thread-pool, shared-memory

@philjs/workers

Thread pool Web Workers for PhilJS - parallel computing, shared state, task queuing

Overview

Thread pool Web Workers for PhilJS - parallel computing, shared state, task queuing

Focus Areas

- philjs, workers, web-workers, parallel, thread-pool, shared-memory

Entry Points

- packages/philjs-workers/src/index.ts

Quick Start

```
import { Channel, ParallelIterator, PoolStats } from '@philjs/workers';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- Channel
- ParallelIterator
- PoolStats
- PoolWorker
- SharedState

- Task
- TaskResult
- WorkerMessage
- WorkerPool
- WorkerPoolConfig
- useParallel
- useSharedState

Install

```
pnpm add @philjs/workers
```

Usage

```
import { Channel, ParallelIterator, PoolStats } from '@philjs/workers';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-workers/src/index.ts

Public API

- Direct exports: Channel, ParallelIterator, PoolStats, PoolWorker, SharedState, Task, TaskResult, WorkerMessage, WorkerPool, WorkerPoolConfig, useParallel, useSharedState, useWorkerPool
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/workflow - Type: Node package - Purpose: Visual workflow engine for PhilJS - node-based execution, human tasks, parallel/conditional flows - Version: 0.1.0 - Location: packages/philjs-workflow - Entry points: packages/philjs-workflow/src/index.ts - Keywords: philjs, workflow, bpmn, orchestration, human-task, state-machine

@philjs/workflow

Visual workflow engine for PhilJS - node-based execution, human tasks, parallel/conditional flows

Overview

Visual workflow engine for PhilJS - node-based execution, human tasks, parallel/conditional flows

Focus Areas

- philjs, workflow, bpmn, orchestration, human-task, state-machine

Entry Points

- packages/philjs-workflow/src/index.ts

Quick Start

```
import { ExecutionContext, ExecutionHistoryEntry, FormField } from '@philjs/workflow';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- ExecutionContext
- ExecutionHistoryEntry
- FormField
- HumanTask
- NodeConfig
- NodeHandler
- NodeType
- PortDefinition
- TriggerConfig
- TriggerDefinition
- VariableDefinition
- WorkflowBuilder

Install

```
pnpm add @philjs/workflow
```

Usage

```
import { ExecutionContext, ExecutionHistoryEntry,FormField } from '@philjs/workflow';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-workflow/src/index.ts

Public API

- Direct exports: ExecutionContext, ExecutionHistoryEntry, FormField, HumanTask, NodeConfig, NodeHandler, NodeType, PortDefinition, TriggerConfig, TriggerDefinition, VariableDefinition, WorkflowBuilder, WorkflowDefinition, WorkflowEdge, WorkflowEngine, WorkflowInstance, WorkflowNode, WorkflowStatus, useHumanTasks, useWorkflowEngine, useWorkflowInstance
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/xr - Type: Node package - Purpose: WebXR Components - VR/AR/MR experiences with hand tracking, spatial UI, gestures, and 3D reactive primitives - Version: 0.1.0 - Location: packages/philjs-xr - Entry points: packages/philjs-xr/src/index.ts - Keywords: philjs, webxr, vr, ar, mr, xr, virtual-reality, augmented-reality, hand-tracking, spatial-ui, gestures, immersive

@philjs/xr

WebXR Components for building immersive VR/AR/MR experiences with PhilJS. Features hand tracking, spatial UI, gesture recognition, and 3D reactive primitives.

Installation

```
npm install @philjs/xr
```

Requirements

- Node.js >= 24
- Browser with WebXR support (Chrome, Edge, Firefox, or compatible VR/AR headset browser)
- TypeScript 6.x (for development)

Basic Usage

```

import {
  initXR,
  useXR,
  useXRControllers,
  useXRHands,
  useGesture,
  Vec3,
  Quat
} from '@philjs/xr';

// Initialize XR with configuration
const manager = initXR({
  mode: 'immersive-vr',
  referenceSpace: 'local-floor',
  handTracking: true,
  frameRate: 72
});

// Start an XR session
const { startSession, endSession, session } = useXR();

async function enterVR() {
  const xrSession = await startSession();
  if (xrSession) {
    console.log('VR session started!');
  }
}

// Access controllers
const controllers = useXRControllers();
const leftController = useXRController(0);

// Access hand tracking
const hands = useXRHands();
const leftHand = useXRHand('left');

// Register gesture callbacks
useGesture('pinch', (event) => {
  console.log(`Pinch detected on ${event.hand} hand with confidence ${event.confidence}`);
});

```

Spatial UI Components

```

import {
  createXRPanel,
  createXRButton,
  createXRSlider,
  createXRText,
  createXRModel,
  Vec3
} from '@philjs/xr';

// Create a floating panel
const panel = createXRPanel({
  position: Vec3.create(0, 1.5, -2),
  width: 1,
  height: 0.5,
  backgroundColor: '#ffffff',
  billboard: true
});

// Create an interactive button
const button = createXRButton({
  position: Vec3.create(0, 1.6, -2),
  label: 'Click Me',
  onClick: () => console.log('Button clicked!'),
  hapticFeedback: true
});

// Create a slider control
const slider = createXRSlider({
  position: Vec3.create(0, 1.4, -2),
  min: 0,
  max: 100,
  value: 50,
  onChange: (value) => console.log(`Slider value: ${value}`)
});

```

API Reference

Initialization

- `initXR(config?: XRConfig): XRSessionManager` - Initialize the XR system
- `getXRManager(): XRSessionManager | null` - Get the global XR manager instance

Hooks

- `useXR()` - Access XR session controls (isSupported, startSession, endSession, session, referenceSpace)

- `useXRControllers()` - Get all connected controllers
- `useXRController(index: number)` - Get a specific controller by index
- `useXRHands()` - Get all tracked hands
- `useXRHand(handedness: 'left' | 'right')` - Get a specific hand
- `useGesture(gesture: GestureType, callback)` - Register a gesture callback
- `useXRFrame(callback)` - Register a per-frame callback
- `useHitTest()` - Perform AR hit testing
- `useAnchors()` - Create and manage AR anchors

Classes

- `XRSessionManager` - Core XR session management
- `GestureRecognizer` - Hand gesture detection

Spatial UI Factories

- `createXRPanel(props: XRPanelProps)` - Create a floating UI panel
- `createXRButton(props: XRButtonProps)` - Create an interactive button
- `createXRSlider(props: XRSliderProps)` - Create a slider control
- `createXRText(props: XRTextProps)` - Create 3D text
- `createXRModel(props: XRModelProps)` - Load and display 3D models

Math Utilities

- `Vec3` - 3D vector operations (create, add, sub, scale, normalize, dot, cross, distance, lerp)
- `Quat` - Quaternion operations (identity, fromEuler, multiply, slerp)

Types

- `XRConfig` - Configuration options for XR sessions
- `XRSessionMode` - 'inline' | 'immersive-vr' | 'immersive-ar'
- `XRReferenceSpaceType` - 'viewer' | 'local' | 'local-floor' | 'bounded-floor' | 'unbounded'
- `XRControllerState` - Controller position, rotation, buttons, and haptics
- `XRHandState` - Hand tracking data with joint positions
- `XRHandJoint` - All 25 hand joint names
- `GestureType` - 'pinch' | 'grab' | 'point' | 'thumbs-up' | 'peace' | 'fist' | 'open-palm'
- `Vector3` - {x, y, z}
- `Quaternion` - {x, y, z, w}
- `SpatialUIProps` - Common props for spatial UI components

Supported Gestures

The gesture recognizer supports the following hand gestures:

- `pinch` - Thumb and index finger touching
- `grab` - Closed fist gripping gesture
- `point` - Index finger extended
- `thumbs-up` - Thumb extended upward
- `peace` - Index and middle fingers extended
- `fist` - Fully closed hand
- `open-palm` - All fingers extended

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philijs-xr/src/index.ts

Public API

- Direct exports: `GestureEvent`, `GestureRecognizer`, `GestureType`, `Quat`, `Quaternion`, `SpatialUIProps`, `Vec3`, `XRAnchor`, `XRButtonProps`, `XRButtonState`, `XRConfig`, `XRControllerState`, `XRHandJoint`, `XRHandState`, `XRHitTestResultData`, `XRJointState`, `XRModelProps`, `XRPanelProps`, `XRReferenceSpaceType`, `XRSessionManager`, `XRSessionMode`, `XRSessionState`, `XRSliderProps`, `XRTextProps`, `createXRButton`, `createXRModel`, `createXRPanel`, `createXRSlider`, `createXRText`, `getXRManager`, `initXR`, `useAnchors`, `useGesture`, `useHitTest`, `useXR`, `useXRController`, `useXRControllers`, `useXRFrame`, `useXRHand`, `useXRHands`
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philijs/xstate - Type: Node package - Purpose: XState-inspired state machines for PhilJS with signals - Version: 0.1.0 - Location: packages/philijs-xstate - Entry points: packages/philijs-xstate/src/index.ts - Keywords: philijs, state-machine, xstate, signals, reactive, fsm

philijs-xstate

XState-inspired state machines for PhilJS with signal-based reactivity.

Features

- Finite state machines with clear state transitions

- Signal-based reactivity for optimal performance
- Guards and actions
- Entry/exit actions
- Context management
- Actor model support
- Delayed transitions
- Invoked services
- State visualization (Mermaid diagrams)
- TypeScript support

Installation

```
npm install philjs-xstate philjs-core
```

Basic Usage

Create a State Machine

```
import { createMachine } from 'philjs-xstate';

const toggleMachine = createMachine({
  id: 'toggle',
  initial: 'inactive',
  states: {
    inactive: {
      on: { TOGGLE: 'active' }
    },
    active: {
      on: { TOGGLE: 'inactive' }
    }
  }
});
```

Use in Components

```
import { useMachine } from 'philjs-xstate';

function Toggle() {
  const [state, send] = useMachine(toggleMachine);

  return (
    <div>
      <p>State: {state.value}</p>
      <button onClick={() => send('TOGGLE')}>
        {state.value === 'active' ? 'Turn Off' : 'Turn On'}
      </button>
    </div>
  );
}
```

Advanced Examples

Traffic Light Machine

```
const trafficLightMachine = createMachine({
  id: 'trafficlight',
  initial: 'red',
  states: {
    red: {
      after: {
        3000: 'green' // Auto-transition after 3 seconds
      }
    },
    yellow: {
      after: {
        1000: 'red'
      }
    },
    green: {
      after: {
        2000: 'yellow'
      }
    }
  }
});
```

Fetch Machine with Context

```
import { createMachine, assign } from 'philjs-xstate';

interface FetchContext {
  data: any;
  error: Error | null;
  retries: number;
}

const fetchMachine = createMachine<FetchContext>({
  id: 'fetch',
  initial: 'idle',
  context: {
    data: null,
    error: null,
    retries: 0,
  },
  states: {
    idle: {
      on: { FETCH: 'loading' }
    },
    loading: {
      invoke: {
        src: async (ctx, evt) => {
          const response = await fetch(evt.url);
          return response.json();
        },
        onDone: {
          target: 'success',
          actions: assign((ctx, evt) => ({ data: evt.data }))
        },
        onError: {
          target: 'failure',
          actions: assign((ctx, evt) => ({
            error: evt.error,
            retries: ctx.retries + 1
          }))
        }
      }
    },
    success: {
      on: { FETCH: 'loading' }
    },
    failure: {
      on: {
        RETRY: {
          target: 'loading',
          cond: (ctx) => ctx.retries < 3
        }
      }
    }
  });
});
```

Form Validation Machine

```

import { createMachine, assign, guard } from 'philjs-xstate';

const formMachine = createMachine({
  id: 'form',
  initial: 'editing',
  context: {
    email: '',
    password: '',
    errors: [],
  },
  states: {
    editing: {
      on: {
        CHANGE: {
          actions: assign((ctx, evt) => ({
            [evt.field]: evt.value
          }))
        },
        SUBMIT: {
          target: 'validating',
          cond: guard((ctx) =>
            ctx.email.length > 0 && ctx.password.length > 0
          )
        }
      }
    },
    validating: {
      entry: assign((ctx) => {
        const errors = [];
        if (!ctx.email.includes('@')) {
          errors.push('Invalid email');
        }
        if (ctx.password.length < 8) {
          errors.push('Password too short');
        }
        return { errors };
      }),
      always: [
        {
          target: 'submitting',
          cond: (ctx) => ctx.errors.length === 0
        },
        {
          target: 'editing'
        }
      ]
    },
    submitting: {
      invoke: {
        src: async (ctx) => {
          const response = await fetch('/api/login', {
            method: 'POST',
            body: JSON.stringify(ctx)
          });
          return response.json();
        },
        onDone: 'success',
        onError: {
          target: 'editing',
          actions: assign((ctx, evt) => ({
            errors: [evt.error.message]
          }))
        }
      }
    },
    success: {
      type: 'final'
    }
  }
});

```

Entry and Exit Actions

```

const doorMachine = createMachine({
  initial: 'closed',
  states: {
    closed: {
      on: { OPEN: 'open' },
      entry: (ctx, evt) => console.log('Door is now closed'),
      exit: (ctx, evt) => console.log('Opening door...')
    },
    open: {
      on: { CLOSE: 'closed' },
      entry: (ctx, evt) => console.log('Door is now open'),
      exit: (ctx, evt) => console.log('Closing door...')
    }
  }
});

```

Guards (Conditional Transitions)

```

const ageMachine = createMachine({
  initial: 'checkAge',
  context: { age: 0 },
  states: {
    checkAge: {
      on: {
        SUBMIT: [
          {
            target: 'adult',
            cond: (ctx) => ctx.age >= 18
          },
          {
            target: 'minor',
            cond: (ctx) => ctx.age > 0
          },
          {
            target: 'invalid'
          }
        ]
      },
      adult: {},
      minor: {},
      invalid: {}
    }
  }
});

```

Visualization

Generate Mermaid Diagram

```

import { createMachine, toMermaid } from 'philjs-xstate';

const machine = createMachine({
  initial: 'idle',
  states: {
    idle: {
      on: { START: 'running' }
    },
    running: {
      on: {
        PAUSE: 'paused',
        STOP: 'stopped'
      }
    },
    paused: {
      on: {
        RESUME: 'running',
        STOP: 'stopped'
      }
    },
    stopped: {
      type: 'final'
    }
  }
});

console.log(toMermaid(machine));

```

Output:

```

stateDiagram-v2
[*] --> idle
idle --> running: START
running --> paused: PAUSE
running --> stopped: STOP
paused --> running: RESUME
paused --> stopped: STOP
stopped --> [*]

```

Generate Visualization Data

```
import { visualize } from 'philjs-xstate';

const graph = visualize(machine);

console.log(graph.nodes); // Array of state nodes
console.log(graph.edges); // Array of transitions

// Use with visualization libraries like D3, Cytoscape, etc.
```

Actor API

Create and Control Actors

```
const machine = createMachine({
  initial: 'idle',
  states: {
    idle: {
      on: { START: 'active' }
    },
    active: {
      on: { STOP: 'idle' }
    }
  }
});

const actor = machine.createActor();

// Subscribe to state changes
const unsubscribe = actor.subscribe((state) => {
  console.log('State changed:', state.value);
});

// Send events
actor.send('START');
actor.send({ type: 'STOP', data: 'custom data' });

// Get current state
const snapshot = actor.getSnapshot();
console.log(snapshot.value);

// Clean up
actor.stop();
unsubscribe();
```

State Matching

```
const [state, send] = useMachine(machine);

if (state().matches('loading')) {
  return <Spinner />;
}

if (state().matches('success')) {
  return <Success data={state().context.data} />;
}

if (state().matches('error')) {
  return <Error error={state().context.error} />;
}
```

Context Updates

```
import { assign } from 'philjs-xstate';

const counterMachine = createMachine({
  context: { count: 0 },
  initial: 'active',
  states: {
    active: {
      on: {
        INCREMENT: {
          actions: assign((ctx) => ({ count: ctx.count + 1 }))
        },
        DECREMENT: {
          actions: assign((ctx) => ({ count: ctx.count - 1 }))
        },
        ADD: {
          actions: assign((ctx, evt) => ({
            count: ctx.count + evt.value
          }))
        }
      }
    }
  }
});
```

TypeScript

Full TypeScript support with type inference:

```
import { createMachine, EventObject } from 'philjs-xstate';

interface ToggleContext {
  count: number;
}

type ToggleEvent =
  | { type: 'TOGGLE' }
  | { type: 'RESET' };

const machine = createMachine<ToggleContext, ToggleEvent>({
  context: { count: 0 },
  initial: 'inactive',
  states: {
    inactive: {
      on: {
        TOGGLE: {
          target: 'active',
          actions: assign((ctx) => ({ count: ctx.count + 1 }))
        }
      }
    },
    active: {
      on: { TOGGLE: 'inactive' }
    }
  }
});
```

API Reference

`createMachine(config)`

Creates a state machine.

Config: - `id` - Machine identifier (optional) - `initial` - Initial state (required) - `context` - Initial context (optional) - `states` - State definitions (required)

`useMachine(machine)`

Creates a signal-based actor for use in components.

Returns: [state: Signal<State>, send: (event) => void]

`machine.createActor()`

Creates an actor instance.

Returns: ActorRef with methods: - `send(event)` - Send event to machine - `subscribe(listener)` - Subscribe to state changes - `getSnapshot()` - Get current state - `stop()` - Stop actor and clean up

`assign(assigner)`

Creates an action that updates context.

`guard(predicate)`

Creates a guard condition for transitions.

`visualize(machine)`

Generates visualization data.

Returns: { nodes, edges }

`toMermaid(machine)`

Generates Mermaid diagram syntax.

Returns: String

Best Practices

1. Keep state machines focused on a single concern
2. Use guards for conditional logic
3. Use context for data, states for behavior
4. Name events with SCREAMING_CASE
5. Use final states to mark completion
6. Leverage visualization for documentation
7. Test state machines thoroughly

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-xstate/src/index.ts

Public API

- Direct exports: Action, ActorRef, EventObject, Guard, Machine, MachineConfig, Service, ServiceConfig, State, StateNode, StateValue, TransitionConfig, VisualizationGraph, VisualizationNode, assign, createMachine, guard, toMermaid, useMachine, visualize
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

@philjs/zig - Type: Node package - Purpose: Zig bindings for PhilJS - Ultra-fast low-level performance tooling (Bun-level speed) - Version: 0.1.0 - Location: packages/philjs-zig - Entry points: packages/philjs-zig/src/index.ts, packages/philjs-zig/src/runtime.ts, packages/philjs-zig/src/wasm.ts - Keywords: philjs, zig, performance, wasm, simd, fast, low-level, bun, runtime

@philjs/zig

Zig bindings for PhilJS - Ultra-fast low-level performance tooling (Bun-level speed)

Overview

Zig bindings for PhilJS - Ultra-fast low-level performance tooling (Bun-level speed)

Focus Areas

- philjs, zig, performance, wasm, simd, fast, low-level, bun, runtime

Entry Points

- packages/philjs-zig/src/index.ts
- packages/philjs-zig/src/runtime.ts
- packages/philjs-zig/src/wasm.ts

Quick Start

```
import { SIMDOPs, ZigRuntime, buildZig } from '@philjs/zig';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- SIMDOPs
- ZigRuntime
- buildZig
- checkZigInstalled
- createSIMDOPs
- initZigProject
- loadWasmModule
- simdSupported
- streamWasmModule

Install

```
pnpm add @philjs/zig
```

Usage

```
import { SIMDOPs, ZigRuntime, buildZig } from '@philjs/zig';
```

Scripts

- pnpm run build
- pnpm run test

Compatibility

- Node >=24
- TypeScript 6

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: `./runtime`, `./wasm`
- Source files: packages/philjs-zig/src/index.ts, packages/philjs-zig/src/runtime.ts, packages/philjs-zig/src/wasm.ts

Public API

- Direct exports: SIMDOPs, ZigRuntime, buildZig, checkZigInstalled, createSIMDOPs, initZigProject, loadWasmModule, simdSupported, streamWasmModule
- Re-exported names: (none detected)
- Re-exported modules: `./runtime.js`, `./types.js`, `./wasm.js`

License

MIT

@philjs/zustand - Type: Node package - Purpose: Zustand-style state management for PhilJS with signals - Version: 0.1.0 - Location: packages/philjs-zustand - Entry points: packages/philjs-zustand/src/index.ts - Keywords: philjs, state-management, zustand, signals, reactive

philjs-zustand

Zustand-style state management for PhilJS with fine-grained reactive signals.

Features

- Minimal, unopinionated API
- Signal-based reactivity for optimal performance
- Middleware support (persist, devtools, immer)
- TypeScript support
- No boilerplate
- Easy to test

Installation

```
npm install philjs-zustand philjs-core
```

Basic Usage

Create a Store

```
import { createStore } from 'philjs-zustand';

const useStore = createStore((set) => ({
  count: 0,
  increment: () => set((state) => ({ count: state.count + 1 })),
  decrement: () => set((state) => ({ count: state.count - 1 })),
  reset: () => set({ count: 0 }),
}));
```

Use in Components

```
function Counter() {
  const count = useStore(state => state.count);
  const increment = useStore(state => state.increment);
  const decrement = useStore(state => state.decrement);

  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={increment}>+</button>
      <button onClick={decrement}>-</button>
    </div>
  );
}
```

Access State Outside Components

```
// Get current state
const currentState = useStore.getState();
console.log(currentState.count);

// Update state
useStore.setState({ count: 10 });

// Subscribe to changes
const unsubscribe = useStore.subscribe((state, prevState) => {
  console.log('Count changed from', prevState.count, 'to', state.count);
});

// Later...
unsubscribe();
```

Middleware

Persist - Save to LocalStorage

```
import { createStore, persist } from 'philjs-zustand';

const useStore = createStore(
  persist(
    (set) => ({
      name: '',
      age: 0,
      setName: (name) => set({ name }),
      setAge: (age) => set({ age }),
    }),
    {
      name: 'user-storage', // LocalStorage key
      storage: localStorage, // or sessionStorage
      partialize: (state) => ({ name: state.name }), // Only persist name
    }
  )
);
```

Persist Options

- name - LocalStorage key (required)
- storage - Storage object (default: localStorage)
- serialize - Custom serializer (default: JSON.stringify)
- deserialize - Custom deserializer (default: JSON.parse)

- `partialize` - Select which fields to persist
- `version` - Version number for migrations
- `migrate` - Migration function for version updates

DevTools - Redux DevTools Integration

```
import { createStore, devtools } from 'philjs-zustand';

const useStore = createStore(
  devtools(
    (set) => ({
      count: 0,
      increment: () => set((s) => ({ count: s.count + 1 }), false, 'increment'),
      decrement: () => set((s) => ({ count: s.count - 1 }), false, 'decrement'),
    }),
    { name: 'CounterStore' }
  )
);
```

Note: Pass action name as third argument to `set()` for better DevTools tracking.

Immer - Mutable Updates

```
import { createStore, immer } from 'philjs-zustand';

const useStore = createStore(
  immer((set) => ({
    nested: { deeply: { value: 0 } },
    items: [1, 2, 3],
    updateNested: () => set((state) => {
      state.nested.deeply.value++; // Mutable syntax!
    }),
    addItem: (item) => set((state) => {
      state.items.push(item); // Mutable array operations!
    }),
  })),
);
```

Combining Middleware

```
import { createStore, persist, devtools, immer } from 'philjs-zustand';

const useStore = createStore(
  devtools(
    persist(
      immer((set) => ({
        todos: [],
        addTodo: (text) => set((state) => {
          state.todos.push({ id: Date.now(), text, done: false });
        }),
      })),
      { name: 'todos-storage' }
    ),
    { name: 'TodoStore' }
  )
);
```

Advanced Patterns

Slices Pattern

```

import { createStore } from 'philjs-zustand';

// Define slices
const createUserSlice = (set, get) => ({
  user: null,
  setUser: (user) => set({ user }),
  clearUser: () => set({ user: null }),
});

const createCartSlice = (set, get) => ({
  items: [],
  addItem: (item) => set((state) => ({
    items: [...state.items, item]
  })),
  removeItem: (id) => set((state) => ({
    items: state.items.filter(item => item.id !== id)
  })),
  total: () => {
    const { items } = get();
    return items.reduce((sum, item) => sum + item.price, 0);
  },
});

// Combine slices
const useStore = createStore((set, get, api) => ({
  ...createUserSlice(set, get, api),
  ...createCartSlice(set, get, api),
}));

```

Combining Multiple Stores

```

import { createStore, combine } from 'philjs-zustand';

const useUserStore = createStore(() => ({ name: 'John' }));
const useCartStore = createStore(() => ({ items: [] }));

const useAppStore = combine({
  user: useUserStore,
  cart: useCartStore,
});

// Access combined state
const { user, cart } = useAppStore.getState();

```

Selectors with Equality

```

import { shallow } from 'philjs-zustand';

// Prevent unnecessary re-renders with shallow equality
const { firstName, lastName } = useStore(
  (state) => ({ firstName: state.firstName, lastName: state.lastName }),
  shallow
);

```

Async Actions

```

const useStore = createStore((set, get) => ({
  user: null,
  loading: false,
  error: null,

  fetchUser: async (id) => {
    set({ loading: true, error: null });
    try {
      const response = await fetch(`api/users/${id}`);
      const user = await response.json();
      set({ user, loading: false });
    } catch (error) {
      set({ error: error.message, loading: false });
    }
  },
));

```

API Reference

`createStore(createState)`

Creates a new store.

Parameters: - `createState(set, get, api)` - Function that returns initial state

Returns: Store hook with attached methods

`set(partial, replace?, actionName?)`

Updates the store state.

Parameters: - `partial` - New state or updater function - `replace` - If true, replace entire state (default: false) - `actionName` - Action name for DevTools (optional)

`get()`

Returns current state.

```

subscribe(listener)
Subscribes to state changes.

Parameters: - listener(state, prevState) - Callback function

Returns: Unsubscribe function

destroy()
Destroys the store and clears all listeners.

```

TypeScript

Full TypeScript support with type inference:

```

interface BearState {
  bears: number;
  increase: (by: number) => void;
}

const useStore = createStore<BearState>((set) => ({
  bears: 0,
  increase: (by) => set((state) => ({ bears: state.bears + by })),
}));

```

Performance Tips

1. Use selectors to prevent unnecessary re-renders
2. Memoize selector functions for better performance
3. Use `shallow` for object selectors
4. Batch updates with `batch()` from `philjs-core`
5. Use slices to organize large stores

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-zustand/src/index.ts

Public API

- Direct exports: Destroy, DevToolsOptions, GetState, Middleware, PersistOptions, SetState, StateCreator, StoreApi, Subscribe, UseStore, combine, createSlice, createStore, devtools, immer, persist, shallow
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

cargo-philjs - Type: Rust crate - Purpose: Cargo subcommand for PhilJS - create, build, and develop PhilJS applications with best-in-class DX - Version: 0.1.0 - Location: packages/cargo-philjs - Entry points: packages/cargo-philjs/src/main.rs - Keywords: cargo, philjs, cli, wasm, web, rust-ui

cargo-philjs

The ultimate CLI for building PhilJS applications - Best-in-class developer experience for Rust web development.

cargo-philjs is a powerful Cargo subcommand that provides everything you need to create, develop, build, and deploy PhilJS applications with maximum productivity.

Features

- **Project Scaffolding** - Multiple templates (SPA, SSR, Fullstack, API, Static Site, Component Library)
- **Hot Reload** - Lightning-fast development with incremental compilation
- **Optimized Builds** - Production builds with WASM optimization and tree-shaking
- **Code Generation** - Scaffold components, pages, API routes, and more
- **Type Checking** - Integrated Clippy and rustfmt support
- **Testing** - Run unit and integration tests with coverage
- **Deployment** - One-command deploy to Vercel, Netlify, Cloudflare, and more
- **Beautiful CLI** - Polished terminal output with progress indicators

Installation

From crates.io (Recommended)

```
cargo install cargo-philjs
```

From source

```
git clone https://github.com/anthropics/philjs
cd philjs/packages/cargo-philjs
cargo install --path .
```

Verify installation

```
cargo philjs --version
```

Quick Start

Create a new PhilJS application:

```
# Create a new SPA
cargo philjs new my-app

# Create with a specific template
cargo philjs new my-api --template=api
```

Start development server:

```
cd my-app
cargo philjs dev
```

Build for production:

```
cargo philjs build --release
```

CLI Commands

new - Create a new project

Create a new PhilJS project from a template.

```
cargo philjs new <name> [OPTIONS]
```

Options: -t, --template <TEMPLATE> - Project template (default: spa) - spa - Single-page application (client-only) - ssr - Server-side rendered application - fullstack - Full-stack with API routes and SSR - api - REST API service - static - Static site generator - component-library - Reusable component library - liveview - Phoenix LiveView-style server-driven UI - minimal - Minimal starter template - --no-git - Skip git initialization - --no-install - Skip installing dependencies - --philjs-version <VERSION> - Use specific PhilJS version

Examples:

```
# Create SPA with default template
cargo philjs new my-app

# Create fullstack application
cargo philjs new my-app --template=fullstack

# Create API service
cargo philjs new my-api --template=api

# Create without git
cargo philjs new my-app --no-git

# Use specific PhilJS version
cargo philjs new my-app --philjs-version=2.1.0
```

init - Initialize in existing project

Add PhilJS to an existing Rust project.

```
cargo philjs init [OPTIONS]
```

Options: -t, --template <TEMPLATE> - Template to use (default: spa)

Example:

```
cd my-existing-project
cargo philjs init --template=spa
```

dev - Development server

Start development server with hot reload.

```
cargo philjs dev [OPTIONS]
```

Options: -p, --port <PORT> - Port to run on (default: 3000, env: PHILJS_PORT) - --host <HOST> - Host to bind to (default: 127.0.0.1, env: PHILJS_HOST) - --open - Open browser automatically - --https - Enable HTTPS with auto-generated certificate - --watch <DIRS> - Watch additional directories (comma-separated) - --no-hot-reload - Disable hot module replacement

Examples:

```
# Start dev server on default port 3000
cargo philjs dev

# Custom port and auto-open browser
cargo philjs dev --port=8080 --open

# Enable HTTPS for testing
cargo philjs dev --https

# Watch additional directories
cargo philjs dev --watch=assets,config

# Disable hot reload
cargo philjs dev --no-hot-reload
```

build - Production build

Build optimized production bundle.

```
cargo philjs build [OPTIONS]
```

Options: - -r, --release - Enable release optimizations - -o, --out-dir <DIR> - Output directory (default: dist) - --target <TARGET> - Build target (default: browser) - browser - WebAssembly for browser - node - Node.js target - deno - Deno runtime - cloudflare - Cloudflare Workers - --ssr - Enable server-side rendering - --source-map - Generate source maps - --no-optimize - Skip WASM optimization (faster builds) - --analyze - Analyze bundle size - --minify - Minify output (default for release)

Examples:

```
# Production build
cargo philjs build --release

# Build with SSR enabled
cargo philjs build --release --ssr

# Analyze bundle size
cargo philjs build --release --analyze

# Custom output directory
cargo philjs build --release --out-dir=public

# Build for Cloudflare Workers
cargo philjs build --release --target=cloudflare

# Development build with source maps
cargo philjs build --source-map

# Skip optimization for faster builds
cargo philjs build --release --no-optimize

# Build with minification
cargo philjs build --release --minify
```

check - Type checking and linting

Type check and lint your project.

```
cargo philjs check [OPTIONS]
```

Options: - --clippy - Run Clippy lints - --fmt - Check code formatting - --fix - Auto-fix issues where possible

Examples:

```
# Basic type check
cargo philjs check

# Run with Clippy
cargo philjs check --clippy

# Check formatting
cargo philjs check --fmt

# Auto-fix issues
cargo philjs check --fix
```

test - Run tests

Run unit and integration tests.

```
cargo philjs test [OPTIONS] [PATTERN]
```

Options: - --watch - Run in watch mode - --browser - Run browser/WASM tests - --coverage - Generate coverage report

Examples:

```
# Run all tests
cargo philjs test

# Run specific tests
cargo philjs test user

# Watch mode
cargo philjs test --watch

# Run browser tests
cargo philjs test --browser

# Generate coverage
cargo philjs test --coverage
```

generate - Code generation

Generate boilerplate code for components, pages, and more.

```
cargo philjs generate <SUBCOMMAND>
```

Subcommands:

component - Generate a component

```
cargo philjs generate component <NAME> [OPTIONS]
```

Options: - -d, --dir <DIR> - Directory to create in (default: src/components) - --tests - Include test file (default: true) - --props - Generate with props boilerplate - --styled - Include CSS module

Examples:

```
# Generate basic component
cargo philjs generate component Button

# Generate in custom directory
cargo philjs generate component Button --dir=src/ui

# Generate with props and styles
cargo philjs generate component Card --props --styled

# Skip tests
cargo philjs generate component Simple --tests=false
```

page - Generate a page/route

```
cargo philjs generate page <NAME> [OPTIONS]
```

Options: - --loader - Include data loader function

Examples:

```
# Generate basic page
cargo philjs generate page About

# Generate with data Loader
cargo philjs generate page Dashboard --loader
```

server - Generate server function

```
cargo philjs generate server <NAME>
```

Example:

```
cargo philjs generate server GetUser
```

api - Generate API route

```
cargo philjs generate api <NAME>
```

Example:

```
cargo philjs generate api users
```

store - Generate state store

```
cargo philjs generate store <NAME>
```

Example:

```
cargo philjs generate store UserStore
```

hook - Generate custom hook

```
cargo philjs generate hook <NAME>
```

Example:

```
cargo philjs generate hook useAuth
```

add - Add component or page

Quick command to add components and pages.

```
cargo philjs add <SUBCOMMAND> <NAME>
```

Subcommands: - component <NAME> - Add a new component - page <NAME> - Add a new page

Examples:

```
cargo philjs add component Header
cargo philjs add page Profile
```

deploy - Deploy to platforms

Deploy your application to various hosting platforms.

```
cargo philjs deploy [OPTIONS]
```

Options: - -p, --platform <PLATFORM> - Target platform - vercel - Vercel - netlify - Netlify - cloudflare - Cloudflare Pages - railway - Railway - fly - Fly.io - aws - AWS Lambda - docker - Docker container - --preview - Create preview deployment (not production) - --no-build - Skip build step

Examples:

```
# Deploy to Vercel
cargo philjs deploy --platform=vercel

# Preview deployment
cargo philjs deploy --platform=netlify --preview

# Deploy without rebuilding
cargo philjs deploy --platform=cloudflare --no-build
```

update - Update dependencies

Update PhilJS and dependencies to latest versions.

```
cargo philjs update [OPTIONS]
```

Options: - --all - Update all dependencies - --check - Check for updates without installing

Examples:

```
# Update PhilJS
cargo philjs update

# Update all dependencies
cargo philjs update --all

# Check for updates
cargo philjs update --check
```

info - Project information

Display project information and diagnostics.

```
cargo philjs info [OPTIONS]
```

Options: - --json - Output as JSON

Examples:

```
# Show project info
cargo philjs info

# JSON output
cargo philjs info --json
```

clean - Clean build artifacts

Remove build artifacts and caches.

```
cargo philjs clean [OPTIONS]
```

Options: - --all - Also clean node_modules and .philjs cache

Examples:

```
# Clean build artifacts
cargo philjs clean

# Deep clean
cargo philjs clean --all
```

Templates

Single-Page Application (SPA)

Client-side rendered application.

Features: - Client-side routing - Component-based architecture - Hot reload development - Optimized production builds

```
cargo philjs new my-spa --template=spa
```

Server-Side Rendering (SSR)

Server-rendered application with client hydration.

Features: - SEO-friendly pre-rendering - Fast initial page loads - Progressive enhancement - Streaming SSR support

```
cargo philjs new my-ssr-app --template=ssr
```

Fullstack

Complete fullstack application with API routes and SSR.

Features: - Server functions with `#[server]` macro - API routes with Axum - Database integration ready - Type-safe client-server communication

```
cargo philjs new my-fullstack --template=fullstack
```

REST API

Backend API service with Axum.

Features: - RESTful API endpoints - PostgreSQL integration with SQLx - JWT authentication - Input validation - Health checks

```
cargo philjs new my-api --template=api
```

Static Site Generator

Build-time rendered static sites.

Features: - Markdown content with frontmatter - Blog support out of the box - Zero runtime JavaScript (optional) - Deploy anywhere (Netlify, Vercel, GitHub Pages)

```
cargo philjs new my-blog --template=static
```

Component Library

Shareable UI component library.

Features: - Reusable components - Built-in theming system - Storybook-style showcase - Tree-shakeable exports - Documentation ready

```
cargo philjs new my-components --template=component-library
```

LiveView

Phoenix LiveView-style server-driven UI.

Features: - Real-time updates over WebSocket - No client-side JavaScript needed - Server-side state management - Live form validation

```
cargo philjs new my-liveview --template=liveview
```

Minimal

Bare-bones starter template.

Features: - Minimal dependencies - Maximum flexibility - Perfect starting point

```
cargo philjs new my-minimal --template=minimal
```

Configuration

Environment Variables

- `PHILJS_PORT` - Development server port (default: 3000)
- `PHILJS_HOST` - Development server host (default: 127.0.0.1)
- `RUST_LOG` - Logging level (e.g. debug, info, warn, error)

Project Configuration

Create a `philjs.toml` in your project root:

```
[dev]  
port = 3000  
host = "127.0.0.1"  
open_browser = true  
watch_dirs = ["assets", "config"]  
  
[build]  
target = "browser"  
out_dir = "dist"  
minify = true  
source_maps = false  
  
[deploy]  
platform = "vercel"  
build_command = "cargo philjs build --release"
```

Best Practices

Project Structure

```

my-app/
src/
    lib.rs          # Entry point
    components/    # Reusable components
        mod.rs
        button.rs
        card.rs
    pages/         # Route pages
        mod.rs
        home.rs
        about.rs
    hooks/         # Custom hooks
    stores/        # State management
    utils/         # Utilities
static/
    styles.css
    images/
tests/          # Integration tests
Cargo.toml
philjs.toml    # PhilJS configuration

```

Development Workflow

1. **Start dev server:** cargo philjs dev
2. **Make changes:** Edit source files
3. **Hot reload:** Changes appear instantly
4. **Check types:** cargo philjs check --clippy
5. **Run tests:** cargo philjs test
6. **Build:** cargo philjs build --release
7. **Deploy:** cargo philjs deploy --platform=vercel

Performance Tips

- Use --release for production builds
- Enable --minify for smaller bundles
- Run --analyze to identify large dependencies
- Use code splitting for large applications
- Enable compression on your hosting platform

Troubleshooting

Common Issues

Issue: wasm-pack not found

Solution: Install wasm-pack:

```
cargo install wasm-pack
```

Issue: wasm32-unknown-unknown target not installed

Solution: Add the target:

```
rustup target add wasm32-unknown-unknown
```

Issue: Hot reload not working

Solution: Check that watch directories exist and try:

```
cargo philjs dev --watch=src,static
```

Issue: Build fails with optimization errors

Solution: Skip optimization during development:

```
cargo philjs build --no-optimize
```

Contributing

We welcome contributions! Please see CONTRIBUTING.md for details.

API Snapshot

This section is generated from the package source. Run node scripts/generate-package-atlas.mjs to refresh.

Entry Points

- Source files: packages/cargo-philjs/src/main.rs

Public API

- Public modules: (none detected)
- Public items: BuildTarget, DeployPlatform, ProjectTemplate
- Re-exports: (none detected)

License

MIT License - see LICENSE for details.

Links

- [PhilJS Website](#)
- [Documentation](#)
- [GitHub Repository](#)
- [Discord Community](#)
- [Examples](#)

Support

- [GitHub Issues](#)
- [Discord](#)
- [Twitter](#)

Built with love by the PhilJS community

create-philjs - Type: Node package - Purpose: Create PhilJS apps with one command - Version: 0.1.0 - Location: packages/create-philjs - Entry points: packages/create-philjs/src/index.ts - Keywords: philjs, create, cli, scaffold, rust, wasm

create-philjs

Scaffolding tool for PhilJS apps - The framework that thinks ahead.

Requirements

- [Node.js 24](#) or higher
- [TypeScript 6](#) or higher
- [ESM only](#) - CommonJS is not supported

Features

- [Interactive CLI](#) - Guided project setup
- [Multiple Templates](#) - Starter templates for different use cases
- [TypeScript or JavaScript](#) - Choose your preferred language
- [Package Manager Detection](#) - Automatically detects pnpm, npm, or yarn
- [Git Integration](#) - Initialize git repository
- [Dependencies Installation](#) - Automatic dependency installation
- [Ready to Code](#) - Fully configured project out of the box

Usage

Create a new project

```
pnpm create philjs my-app
```

Or use npm/yarn:

```
npm create philjs@latest my-app
npx create-philjs my-app
yarn create philjs my-app
```

Interactive Mode

Run without a project name for interactive mode:

```
pnpm create philjs
```

You'll be prompted to: 1. Enter a project name 2. Select a template 3. Choose TypeScript or JavaScript 4. Select features (routing, SSR, testing, etc.) 5. Choose a package manager

Templates

Basic

A minimal PhilJS app with routing and basic styling.

```
pnpm create philjs my-app --template basic
```

Includes: - File-based routing - Basic components - CSS support - Development server

Full-Stack

Complete full-stack application with database and authentication.

```
pnpm create philjs my-app --template full-stack
```

Includes: - SSR and SSG support - Database integration (Prisma) - Authentication - API routes - Testing setup

SaaS Starter

Production-ready SaaS application template.

```
pnpm create philjs my-app --template saas
```

Includes: - User authentication - Subscription billing - Admin dashboard - Email integration - Multi-tenancy support - Complete UI components

E-commerce

Online store template with cart and checkout.

```
pnpm create philjs my-app --template ecommerce
```

Includes: - Product catalog - Shopping cart - Checkout flow - Payment integration - Order management

Blog

Content-focused blog template.

```
pnpm create philjs my-app --template blog
```

Includes: - Markdown support - Blog post listing - Categories and tags - RSS feed - SEO optimization

Dashboard

Admin dashboard template with charts and tables.

```
pnpm create philjs my-app --template dashboard
```

Includes: - Analytics charts - Data tables - User management - Settings pages - Dark mode

CLI Options

```
pnpm create philjs [project-name] [options]
```

Options: - --template <name> - Template to use (basic, full-stack, saas, ecommerce, blog, dashboard) - --typescript - Use TypeScript (default) - --javascript - Use JavaScript - --git - Initialize git repository (default: true) - --install - Install dependencies (default: true) - --package-manager <pmm> - Package manager (pnpm, npm, yarn)

Examples

Create with specific options

```
# TypeScript with full-stack template
pnpm create philjs my-app --template full-stack --typescript

# JavaScript with basic template, skip install
pnpm create philjs my-app --template basic --javascript --no-install

# SaaS template with specific package manager
pnpm create philjs my-saas --template saas --package-manager npm
```

After Creation

```
cd my-app
pnpm install # if --no-install was used
pnpm dev      # start development server
```

Project Structure

All templates create a similar structure:

```
my-app/
src/
  components/    # Reusable components
  routes/        # Route components
  styles/        # Global styles
  entry-client.ts # Client entry point
  entry-server.ts # Server entry point (SSR templates)
public/          # Static assets
tests/           # Test files (full-stack+)
package.json
vite.config.ts   # Vite configuration
tsconfig.json    # TypeScript config (TS templates)
README.md
```

What's Included

Every template includes:

- **Development Server** - Fast HMR with Vite
- **Production Build** - Optimized builds
- **Routing** - File-based or config routing
- **Styling** - CSS, CSS Modules, or Tailwind
- **TypeScript** - Full TypeScript support (optional)
- **ESLint & Prettier** - Code quality tools (optional)

Full-stack templates also include:

- **SSR/SSG** - Server-side rendering and static generation
- **API Routes** - Backend API endpoints
- **Database** - Database integration
- **Testing** - Vitest and PhilJS Testing Library
- **Deployment** - Deploy configs for Vercel, Netlify, etc.

Next Steps

After creating your project:

1. **Read the README** - Each template has specific instructions
2. **Explore the code** - Understand the project structure
3. **Start developing** - Run `pnpm dev` and start coding

4. Check the docs - Visit [PhilJS documentation](#)

Troubleshooting

Command not found

Make sure you're using a recent version of Node.js (24+):

```
node --version # Should be 24 or higher
```

Permission errors

On macOS/Linux, you may need to use `sudo`:

```
sudo pnpm create philjs my-app
```

Or fix npm permissions: `npm docs`

Port already in use

Change the port in `vite.config.ts`:

```
export default defineConfig({
  server: {
    port: 3001 // Change to available port
  }
});
```

Documentation

For more information, see the [PhilJS documentation](#).

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/create-philjs/src/index.ts

Public API

- Direct exports: AppLayout, Counter, DocsRoute, HomeRoute, routeManifest, routes
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

create-philjs-plugin - Type: Node package - Purpose: Plugin SDK and scaffolding tool for PhilJS plugins - Version: 0.1.0 - Location: packages/create-philjs-plugin - Entry points: packages/create-philjs-plugin/src/index.ts - Keywords: philjs, plugin, sdk, create, scaffold

create-philjs-plugin

Scaffold new PhilJS plugins with comprehensive templates and interactive CLI.

Features

- Interactive CLI with prompts
- Multiple plugin templates (Basic, Vite, Transform, UI Addon)
- TypeScript support
- Automatic testing setup with Vitest
- Feature-based scaffolding
- Auto-generated documentation
- Example files included
- Git initialization
- Plugin SDK with testing utilities

Quick Start

```
# Using npm
npm create philjs-plugin

# Using yarn
yarn create philjs-plugin

# Using pnpm
pnpm create philjs-plugin

# Using bun
bun create philjs-plugin
```

Usage

Interactive Mode

Simply run the command and follow the prompts:

```
npm create philjs-plugin
```

You'll be asked about:

1. **Plugin name** - Must start with `philjs-plugin-`
2. **Plugin type** - Choose from:
 - o Basic Plugin - Simple plugin with lifecycle hooks
 - o Vite Plugin - Vite integration with transform hooks
 - o Transform Plugin - Code transformation plugin
 - o UI Addon - UI components and styles
3. **Description** - Brief description of your plugin
4. **Author** - Your name or organization
5. **License** - MIT, Apache-2.0, BSD-3-Clause, ISC, or GPL-3.0
6. **Features** - Select from type-specific features
7. **TypeScript** - Use TypeScript (recommended)
8. **Testing** - Include Vitest testing setup
9. **Git** - Initialize git repository

Plugin Templates

Basic Plugin

Simple plugin with lifecycle hooks perfect for adding custom build logic or runtime behavior.

Use cases: Custom build steps, File processing, Environment configuration

Vite Plugin

Integration with Vite build tool for advanced bundling and transformation.

Use cases: Virtual modules, Custom transformations, Build optimizations

Transform Plugin

Code transformation plugin with AST support and source maps.

Use cases: Code generation, Syntax transformations, Import rewrites

UI Addon

Component library and UI utilities plugin.

Use cases: Design systems, Component libraries, Theme providers

Plugin SDK

PluginBuilder

Fluent API for building plugins:

```
import { createBuilder } from 'create-philjs-plugin';

const plugin = createBuilder()
  .meta({ name: 'my-plugin', version: '0.1.0' })
  .setup(async (config, ctx) => {
    ctx.logger.info('Setting up...');
  })
  .hook('init', async (ctx) => {
    ctx.logger.info('Initialized!');
  })
  .build();
```

PluginTester

Testing utilities:

```
import { createTester } from 'create-philjs-plugin';
import myPlugin from './index.js';

const tester = createTester(myPlugin);
await tester.testSetup({ enabled: true });
await tester.testHook('init');
```

Examples

See the `examples` directory for complete plugin examples:

- `basic-plugin.ts` - Hello world plugin
- `vite-plugin.ts` - Virtual modules
- `transform-plugin.ts` - Code transformation
- `ui-addon-plugin.tsx` - UI components

Development Workflow

```
cd philjs-plugin-awesome
npm install
npm run dev      # Watch mode
npm test         # Run tests
npm run build   # Build for production
```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/create-philjs-plugin/src/index.ts

Public API

- Direct exports: Plugin, PluginBuilder, PluginConfigSchema, PluginContext, PluginFileSystem, PluginHooks, PluginLogger, PluginMetadata, PluginTester, PluginUtils, createBuilder, createTester, pluginHelpers, pluginPublisher, pluginValidator
- Re-exported names: PluginOptions, createPlugin
- Re-exported modules: ./generator.js, ./template-engine.js

License

MIT

```
## eslint-config-philjs - Type: Node package - Purpose: ESLint config with a11y and security plugins for PhilJS - Version: 0.1.0 - Location: packages/eslint-config-philjs - Entry points: packages/eslint-config-philjs/src/index.ts
```

eslint-config-philjs

ESLint configuration with accessibility and security plugins for PhilJS projects.

Overview

ESLint config with a11y and security plugins for PhilJS

Entry Points

- packages/eslint-config-philjs/src/index.ts

Quick Start

```
import * as eslint_config_philjs from 'eslint-config-philjs';
```

Wire the exported helpers into your app-specific workflow. See the API snapshot for the full surface.

Exports at a Glance

- (none detected)

Features

- **Accessibility Rules** - JSX a11y plugin for WCAG compliance
- **Security Rules** - Security plugin to catch vulnerabilities
- **PhilJS Best Practices** - Framework-specific linting rules
- **TypeScript Support** - Works with TypeScript projects

Installation

```
pnpm add -D eslint-config-philjs eslint
```

Usage

Create `.eslintrc.js`:

```
export default {
  extends: ['eslint-config-philjs']
};
```

Or in `package.json`:

```
{
  "eslintConfig": {
    "extends": ["eslint-config-philjs"]
  }
}
```

Documentation

For more information, see the PhilJS documentation.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/eslint-config-philjs/src/index.ts

Public API

- Direct exports: (none detected)
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

```
## eslint-plugin-philjs - Type: Node package - Purpose: ESLint plugin for PhilJS - Detect anti-patterns and optimize signal usage - Version: 0.1.0 - Location: packages/philjs-eslint - Entry points: packages/philjs-eslint/src/index.ts - Keywords: eslint, eslintplugin, philjs, signals, reactivity, linting
```

eslint-plugin-philjs

ESLint plugin for PhilJS - Detect anti-patterns and optimize signal usage.

Installation

```
pnpm add -D eslint-plugin-philjs
```

Usage

Add philjs to your ESLint configuration:

Flat Config (ESLint 9+)

```
import philjs from 'eslint-plugin-philjs';

export default [
  {
    plugins: {
      philjs
    },
    rules: {
      'philjs/no-unused-signals': 'error',
      'philjs/effect-cleanup-required': 'warn',
      'philjs/prefer-memo-for-expensive': 'warn'
    }
  }
];
```

Legacy Config (eslintrc)

```
{
  "plugins": ["philjs"],
  "rules": {
    "philjs/no-unused-signals": "error",
    "philjs/effect-cleanup-required": "warn",
    "philjs/prefer-memo-for-expensive": "warn"
  }
}
```

Recommended Configuration

Use the recommended preset for optimal PhilJS development:

```
import philjs from 'eslint-plugin-philjs';

export default [
  philjs.configs.recommended
];
```

Rules

no-unused-signals

Detects signals that are created but never used. Helps prevent memory leaks and unnecessary reactivity.

```
// Bad
const count = signal(0); // Signal created but never read

// Good
const count = signal(0);
console.log(count());
```

effect-cleanup-required

Warns when effects might need cleanup functions to prevent memory leaks.

```
// Bad
effect(() => {
  const interval = setInterval(() => {}, 1000);
  // Missing cleanup!
});

// Good
effect(() => {
  const interval = setInterval(() => {}, 1000);
  return () => clearInterval(interval);
});
```

prefer-memo-for-expensive

Suggests using `memo()` for expensive computations that derive from signals.

```
// Bad
const doubled = () => expensiveCalculation(count());

// Good
const doubled = memo(() => expensiveCalculation(count));
```

API

Rules

- no-unused-signals - Detect unused signal declarations
- effect-cleanup-required - Ensure effects cleanup side effects
- prefer-memo-for-expensive - Recommend memoization for expensive computations

Configurations

- configs.recommended - Recommended rule configuration for PhilJS projects

Configuration Options

Each rule can be configured with severity levels: - 'off' or 0 - Disable the rule - 'warn' or 1 - Warning (doesn't affect exit code) - 'error' or 2 - Error (exits with error code)

Examples

Complete Setup

```
// eslint.config.js
import philjs from 'eslint-plugin-philjs';
import typescriptParser from '@typescript-eslint/parser';

export default [
  {
    files: ['**/*.ts', '**/*.tsx'],
    languageOptions: {
      parser: typescriptParser,
      parserOptions: {
        ecmaVersion: 'latest',
        sourceType: 'module'
      }
    },
    plugins: {
      philjs
    },
    rules: {
      ...philjs.configs.recommended.rules,
      // Override specific rules
      'philjs/prefer-memo-for-expensive': 'off'
    }
  }
];
```

Documentation

For more information about PhilJS best practices and patterns, see the PhilJS documentation.

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Export keys: .
- Source files: packages/philjs-eslint/src/index.ts

Public API

- Direct exports: configs, rules
- Re-exported names: (none detected)
- Re-exported modules: (none detected)

License

MIT

philjs - Type: Rust crate - Purpose: Pure Rust UI framework with fine-grained reactivity - write components in Rust, render anywhere - Version: 0.1.0 - Location: packages/philjs-rust - Entry points: packages/philjs-rust/src/lib.rs - Keywords: ui, framework, wasm, reactive, signals

PhilJS for Rust

Pure Rust UI framework with fine-grained reactivity. Write components in Rust, render anywhere.

Features

- **Fine-grained Reactivity** - Signals, memos, and effects with automatic dependency tracking
- **JSX-like Syntax** - `view!` macro for ergonomic UI authoring
- **Component Model** - Props, children, and composition
- **SSR Support** - Server-side rendering with hydration
- **WASM-first** - Optimized for WebAssembly deployment
- **Type-safe** - Full Rust type safety

Installation

```
[dependencies]
philjs = "0.1.0"
```

Quick Start

```

use philjs::prelude::*;

#[component]
fn Counter(initial: i32) -> impl IntoView {
    let count = signal!(initial);

    view! {
        <div class="counter">
            <h1>"Count: " {count}</h1>
            <button on:click=move |_| count.set(count.get() + 1)>
                "+"
            </button>
            <button on:click=move |_| count.set(count.get() - 1)>
                "-"
            </button>
        </div>
    }
}

fn main() {
    mount(|| view! { <Counter initial=0 /> });
}

```

Reactive Primitives

Signals

```

use philjs::prelude::*;

let count = signal!(0);

// Read
let value = count.get();

// Write
count.set(5);

// Update
count.update(|n| *n += 1);

```

Memos (Computed Values)

```

use philjs::prelude::*;

let count = signal!(0);
let doubled = memo!(count.get() * 2);

count.set(5);
assert_eq!(doubled.get(), 10);

```

Effects

```

use philjs::prelude::*;

let count = signal!(0);

let _effect = Effect::new(move || {
    println!("Count changed to: {}", count.get());
});

count.set(1); // Prints: "Count changed to: 1"

```

Resources (Async Data)

```

use philjs::prelude::*;

let user_id = signal!(1);

let user = Resource::new(
    move || user_id.get(),
    |id| async move {
        fetch_user(id).await
    }
);

// In your view
view! {
    {match user.state().get() {
        ResourceState::Loading => view! { <div>"Loading..."</div> },
        ResourceState::Ready(user) => view! { <div>{user.name}</div> },
        ResourceState::Error(e) => view! { <div>"Error: " {e}</div> },
        _ => view! { <div></div> },
    }}
}

```

Components

Defining Components

```

use philjs::prelude::*;

#[component]
fn Button(
    /// Button label
    label: String,
    /// Click handler
    #[prop(optional)]
    on_click: Option<Box<dyn Fn()>>,
    /// Children
    children: Children,
) -> impl IntoView {
    view! {
        <button on:click=move |_| {
            if let Some(handler) = &on_click {
                handler();
            }
        }>
            {label}
            {children}
        </button>
    }
}

```

Using Components

```

view! {
    <Button label="Click me" on_click={|| println!("Clicked!")}>
        <span>Icon</span>
    </Button>
}

```

Control Flow

Conditional Rendering

```

use philjs::prelude::*;

let show = signal!(true);

view! {
    {Show::new(move || show.get(), || view! { <div>"Visible!"</div> })}
}

```

Lists

```

use philjs::prelude::*;

let items = signal!(vec!["a", "b", "c"]);

view! {
    <ul>
        {For::new(
            move || items.get(),
            |item| view! { <li>{item}</li> }
        )}
    </ul>
}

```

Context (Dependency Injection)

```

use philjs::prelude::*;

#[derive(Clone)]
struct Theme {
    primary: String,
    secondary: String,
}

// Provide
provide_context(Theme {
    primary: "#007bff".to_string(),
    secondary: "#6c757d".to_string(),
});

// Consume
let theme: Option<Theme> = use_context();

```

Server-Side Rendering

```

use philjs::prelude::*;

let html = render_to_string(|| view! {
    <html>
        <head><title>"My App"</title></head>
        <body>
            <div id="app">
                <Counter initial=0 />
            </div>
            {HydrationScript::new().to_html()}
        </body>
    </html>
});

```

Stores (Complex State)

```

use philjs::prelude::*;

#[derive(Store)]
struct AppState {
    count: i32,
    user: Option<User>,
    items: Vec<Item>,
}

let store = AppStateStore::new(AppState {
    count: 0,
    user: None,
    items: vec![],

});

// Access fields as signals
store.count.set(5);
let count = store.count.get();

```

Comparison with Other Rust Frameworks

Feature	PhilJS	Leptos	Dioxus	Yew
Fine-grained Reactivity				
view! Macro			(rsx!)	(html!)
SSR				
Hydration				
Islands				
JS Interop	Best			
Bundle Size	~50KB	~60KB	~80KB	~100KB

Integration with PhilJS (JavaScript)

PhilJS Rust can interop seamlessly with the JavaScript PhilJS:

```

// Rust component
#[component]
fn RustCounter() -> impl IntoView {
    let count = signal!(0);
    view! { <div>{count}</div> }
}

// JavaScript
import { useRustComponent } from 'philjs-wasm';

const RustCounter = useRustComponent('RustCounter');

// Use in JavaScript view
<RustCounter />

```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Source files: packages/philjs-rust/src/lib.rs

Public API

- Public modules: dom, liveview, meta, prelude, query, reactive, router, runtime, server, ssr, store, view, wasm
- Public items: spread_attrs
- Re-exports: crate::dom::(node_ref::NodeRef, event::Event, mount::mount,), crate::reactive::(signal::Signal, memo::Memo, effect::Effect, resource::Resource, batch::batch, context::(provide_context, use_context),), crate::ssr::(render_to_string, render_to_stream), crate::view::(element::Element, text::Text, fragment::Fragment, dynamic::Dynamic, children::Children, into_view::IntoView,), dom::(HydrationMode, HydrationContext, HydrationState, HydrationError, generate_hydration_script,), dom::(node_ref::NodeRef, event::Event, mount::mount,), dom::(hydrate, hydrate_to, hydrate_to_body), meta::(Title, TitleTemplate, Meta, Link, Style, Script, Html, Body, MetaContext, use_meta_context, with_meta_context,), philjs_macros::(component, effect, memo, resource, signal, view, Store), reactive::(Action, MultiAction, ActionError, create_action, create_server_action, create_multi_action, RwSignal, create_rw_signal, StoredValue, create_stored_value, Trigger, create_trigger, on_cleanup,), reactive::(signal::Signal,

```
memo::Memo, effect::Effect, resource::Resource, batch::batch, context::(provide_context, use_context, Context), router::form::(Form, FormMethod, FormData, ActionForm, MultiActionForm), server::functions::(ServerResult, ServerError, ServerFnConfig), ssr::(render_to_string, render_to_stream, render_to_stream_async, StreamingConfig, HydrationScript), store::(Store, StoreField, StoreVec, StoreMap, create_store, produce), view::(Transition, TransitionConfig, use_transition, use_deferred_value, AnimatedShow, AnimatedShowConfig, AnimationState, fade, slide, scale), view::(element::Element, text::Text, fragment::Fragment, dynamic::Dynamic, children::Children, into_view::IntoView, view::View, )
```

License

MIT

philjs-actix - Type: Rust crate - Purpose: Actix-web integration for PhilJS - SSR, API routes, WebSocket, and session management - Version: 0.1.0 - Location: packages/philjs-actix - Entry points: packages/philjs-actix/src/lib.rs - Keywords: actix-web, philjs, ssr, websocket, rust

PhilJS Actix-web Integration

Production-ready Actix-web integration for PhilJS applications with server-side rendering, WebSocket support, and comprehensive middleware.

Features

- **Server-Side Rendering:** Full SSR support with streaming and hydration
- **Custom Extractors:** Type-safe request data extraction with better error messages
- **Pre-built Handlers:** Common route handlers for health checks, pagination, and more
- **WebSocket Support:** Real-time communication for LiveView components
- **Session Management:** Secure session handling out of the box
- **SEO Optimization:** Built-in SEO helpers and meta tag builders

Installation

Add to your `Cargo.toml`:

```
[dependencies]
philjs-actix = "2.0"
actix-web = "4.4"
```

Quick Start

```
use actix_web::{web, App, HttpServer};
use philjs_actix::prelude::*;

#[actix_web::main]
async fn main() -> std::io::Result<()> {
    HttpServer::new(|| {
        App::new()
            .route("/", web::get().to(index))
            .route("/health", web::get().to(health_check))
    })
    .bind("127.0.0.1:8080")?
    .run()
    .await
}

async fn index() -> impl Responder {
    render_to_response(|| view! {
        <h1>"Welcome to PhilJS with Actix!"</h1>
        <p>"Server-side rendered with love"</p>
    })
}
```

Server-Side Rendering

Basic SSR

```
use philjs_actix::prelude::*;

async fn home() -> impl Responder {
    render_to_response(|| view! {
        <div>
            <h1>"Home Page"</h1>
            <p>"This is server-side rendered"</p>
        </div>
    })
}
```

SSR with Hydration

```

use philjs_actix::prelude::*;
use serde::Serialize;

#[derive(Serialize)]
struct PageData {
    user: String,
    items: Vec<String>,
}

async fn dashboard() -> impl Responder {
    let data = PageData {
        user: "John Doe".to_string(),
        items: vec!["Item 1".to_string(), "Item 2".to_string()],
    };

    render_with_data(
        || view! {
            <div id="dashboard">
                <h1>"Dashboard"</h1>
            </div>
        },
        data,
    )
}

```

Full HTML Documents

```

use philjs_actix::ssr::{HtmlDocument, MetaTag, Script, SeoBuilder};

async fn landing_page() -> impl Responder {
    let seo = SeoBuilder::new("My Awesome App")
        .description("The best app ever built")
        .keywords(vec!["rust", "web", "philjs"])
        .og("image", "https://example.com/og-image.jpg")
        .og("type", "website")
        .twitter("card", "summary_large_image")
        .build();

    let mut doc = HtmlDocument::new("My Awesome App")
        .lang("en")
        .stylesheet("/static/styles.css")
        .script(Script::src("/static/app.js").module().defer())
        .body(render_to_string(|| view! {
            <div class="app">
                <h1>"Welcome"</h1>
            </div>
        }));
    for tag in seo {
        doc = doc.meta(tag);
    }

    doc.respond()
}

```

Custom Extractors

PhilJS Actix provides custom extractors with improved error handling:

JSON Extractor

```

use philjs_actix::extractors::Json;
use serde::Deserialize;

#[derive(Deserialize)]
struct CreateUser {
    name: String,
    email: String,
}

async fn create_user(Json(user): Json<CreateUser>) -> impl Responder {
    // Validation happens automatically with better error messages
    HttpResponse::Ok().json(user)
}

```

Query Parameters

```

use philjs_actix::extractors::Query;

#[derive(Deserialize)]
struct SearchQuery {
    q: String,
    page: Option<u32>,
}

async fn search(Query(query): Query<SearchQuery>) -> impl Responder {
    let page = query.page.unwrap_or(1);
    HttpResponse::Ok().body(format!("Searching for: {}", query.q))
}

```

SSR Context

```

use philjs_actix::extractors::SsrContext;

async fn about(ctx: SsrContext) -> impl Responder {
    let user_agent = ctx.user_agent().unwrap_or("unknown");

    render_to_response(|| view! {
        <div>
            <h1>"About"</h1>
            <p>"Your browser: " {user_agent}</p>
        </div>
    })
}

```

Pre-built Handlers

Health Check

```

use philjs_actix::handlers::health_check;

App::new()
    .route("/health", web::get().to(health_check))

```

Pagination

```

use philjs_actix::handlers::{PaginationParams, PaginatedResponse};

async fn list_users(params: web::Query<PaginationParams>) -> impl Responder {
    let users = get_users(params.offset(), params.limit()).await;
    let total = count_users().await;

    PaginatedResponse::new(users, total, params.page, params.per_page).build()
}

```

API Responses

```

use philjs_actix::handlers::ApiResponse;

async fn create_item(data: web::Json<Item>) -> impl Responder {
    match save_item(&data).await {
        Ok(item) => ApiResponse::success(item)
            .with_message("Item created successfully")
            .build(),
        Err(e) => ApiResponse::error(e.to_string())
            .with_error("Validation failed")
            .build(),
    }
}

```

Redirects

```

use philjs_actix::handlers::redirect;

async fn old_route() -> impl Responder {
    redirect("/new-route", false) // temporary redirect
}

async fn moved_permanently() -> impl Responder {
    redirect("/new-location", true) // permanent redirect
}

```

WebSocket LiveView

```

#[cfg(feature = "websocket")]
use philjs_actix::websocket::LiveViewSocket;

#[derive(Default)]
struct Counter {
    count: i32,
}

async fn counter_liveview(
    req: HttpRequest,
    stream: web::Payload,
) -> Result<HttpResponse, actix_web::Error> {
    LiveViewSocket::new(Counter::default())
        .upgrade(req, stream)
}

App::new()
    .route("/counter", web::get().to(counter_liveview))

```

Middleware

SSR Middleware

```

use philjs_actix::middleware::SsrMiddleware;

App::new()
    .wrap(SsrMiddleware::new())
    .route("/", web::get().to(index))

```

Compression

```

#[cfg(feature = "compression")]
use philjs_actix::middleware::CompressionMiddleware;

App::new()
    .wrap(CompressionMiddleware::default())

```

Static Files

```

#[cfg(feature = "static-files")]
use philjs_actix::handlers::StaticFileHandler;

App::new()
    .service(
        StaticFileHandler::new("./public")
            .with_cache_duration(3600)
            .build()
    )

```

Error Handling

```

use philjs_actix::handlers::ErrorHandler;

App::new()
    .default_service(web::to(not_found))

```

Custom errors:

```

use philjs_actix::handlers::ErrorHandler;

async fn custom_error() -> impl Responder {
    ErrorHandler::new(500, "Something went wrong").html()
}

async fn api_error() -> impl Responder {
    ErrorHandler::new(400, "Invalid request").json()
}

```

Advanced SSR

Streaming SSR

```

use philjs_actix::ssr::{SsrRenderer, SsrConfig};

async fn stream_page() -> impl Responder {
    let config = SsrConfig {
        streaming: true,
        hydration: true,
        ..Default::default()
    };

    let renderer = SsrRenderer::new(config);
    renderer.to_response(|| view! {
        <div>"Streamed content"</div>
    })
}

```

Complete Example

```
use actix_web::{web, App, HttpServer, HttpResponse, Responder};
use philjs_actix::prelude::*;
use philjs_actix::handlers::{health_check, PaginationParams, ApiResponse};
use philjs_actix::extractors::{Json, SsrContext};
use serde::{Deserialize, Serialize};

#[derive(Debug, Serialize, Deserialize)]
struct User {
    id: i64,
    name: String,
    email: String,
}

async fn home(ctx: SsrContext) -> impl Responder {
    render_to_response(|| view! {
        <div>
            <h1>"Welcome to My App"</h1>
            <p>"Path: " {ctx.path()}</p>
        </div>
    })
}

async fn create_user(Json(user): Json<User>) -> impl Responder {
    ApiResponse::success(user)
        .with_message("User created")
        .build()
}

async fn list_users(params: web::Query<PaginationParams>) -> impl Responder {
    let users = vec![]; // fetch from database
    PaginatedResponse::new(users, 0, params.page, params.per_page).build()
}

#[actix_web::main]
async fn main() -> std::io::Result<()> {
    HttpServer::new(|| {
        App::new()
            .route("/", web::get().to(home))
            .route("/health", web::get().to(health_check))
            .route("/api/users", web::post().to(create_user))
            .route("/api/users", web::get().to(list_users))
    })
    .bind("127.0.0.1:8080")?
    .run()
    .await
}
```

Features

Enable optional features in your Cargo.toml:

```
[dependencies]
philjs-actix = { version = "2.0", features = ["websocket", "session", "static-files"] }
```

Available features: - **ssr** (default): Server-side rendering support - **websocket**: WebSocket and LiveView support - **session**: Session management - **static-files**: Static file serving - **compression**: Response compression - **tls**: TLS/SSL support

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Source files: packages/philjs-actix/src/lib.rs

Public API

- Public modules: config, cors, error, extractors, handlers, middleware, prelude, service, session, ssr, websocket
- Public items: api_error, api_response, render_document, render_stream, render_to_response, render_with_data
- Re-exports: actix_web::web, App, HttpServer, HttpRequest, HttpResponse, http::StatusCode, Responder, }, config::PhilJsConfig, crate::config::PhilJsConfig, crate::error::PhilJsError, crate::extractors::Json, Form, Path, Query, SsrContext, ConnectionInfo, crate::handlers::{ health_check, not_found, cors_preflight, redirect, ApiResponse, ErrorHandler, PaginationParams, PaginatedResponse, }, crate::middleware::{SsrMiddleware, CompressionMiddleware, TracingMiddleware}, crate::service::PhilJsService, crate::session::{SessionManager, SessionConfig}, crate::ssr::{ SsrRenderer, SsrConfig, HtmlDocument, MetaTag, Script, SeoBuilder, }, crate::websocket::{LiveViewSocket, WebSocketHandler}, crate::render_to_response, render_with_data, render_stream, api_response, error::PhilJsError, middleware::{SsrMiddleware, CompressionMiddleware, TracingMiddleware}, philjs::prelude::*, service::PhilJsService, session::{SessionManager, SessionConfig}, websocket::{LiveViewSocket,

License

MIT

philjs-axum - Type: Rust crate - Purpose: Axum web framework integration for PhilJS - SSR, middleware, and typed extractors - Version: 0.1.0 - Location: packages/philjs-axum
- Entry points: packages/philjs-axum/src/lib.rs - Keywords: axum, philjs, ssr, web, rust

PhilJS Axum Integration

Modern Axum web framework integration for PhilJS applications.

Overview

Axum web framework integration for PhilJS - SSR, middleware, and typed extractors

Focus Areas

- axum, philjs, ssr, web, rust

Entry Points

- packages/philjs-axum/src/lib.rs

Quick Start

```
[dependencies]
philjs-axum = "0.1.0"
```

```
use philjs_axum::{PhilJsHtml, api_error, api_response};
```

Use the exported items above as building blocks in your application.

Exports at a Glance

- PhilJsHtml
- api_error
- api_response
- render_document
- render_to_response
- render_with_hydration

Features

- **Server-Side Rendering:** Full SSR support
- **Type-safe Extractors:** Custom extractors with better error handling
- **Middleware:** PhilJS-specific middleware
- **Route Handlers:** Pre-built handlers for common patterns

Installation

```
[dependencies]
philjs-axum = "2.0"
axum = "0.7"
```

Quick Start

```
use axum::{Router, routing::get};
use philjs_axum::prelude::*;

#[tokio::main]
async fn main() {
    let app = Router::new()
        .route("/", get(index))
        .route("/health", get(health_check));

    let listener = tokio::net::TcpListener::bind("127.0.0.1:3000")
        .await
        .unwrap();

    axum::serve(listener, app).await.unwrap();
}

async fn index() -> Html<String> {
    Html("Hello from PhilJS!".to_string())
}
```

Extractors

```
use philjs_axum::extractors::{PhilJsJson, PhilJsQuery};

async fn create_user(PhilJsJson(user): PhilJsJson<User>) -> impl IntoResponse {
    ApiResponse::success(user)
}
```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Source files: packages/philjs-axum/src/lib.rs

Public API

- Public modules: extractors, handlers, middleware, prelude, ssr, state, tower, websocket
- Public items: PhilJsHtml, api_error, api_response, render_document, render_to_response, render_with_hydration
- Re-exports: axum::extract::(State, Path, Query), axum::(Router, routing::(get, post, put, patch, delete), response::(Html, Json, IntoResponse), http::StatusCode), crate::extractors::(PhilJsJson, PhilJsQuery, SsrContext), crate::handlers::(health_check, not_found, ApiResponse, PaginationParams), crate::middleware::PhilJsLayer, crate::ssr::

```
{HtmlDocument, MetaTag, Script, SeoBuilder}, crate::state::{AppState, AppStateBuilder, Environment}, crate::tower::{TracingLayer, TimeoutLayer, RequestIdLayer, SecurityHeadersLayer, RateLimitLayer}, crate::websocket::{LiveViewSocket, LiveViewHandler, BroadcastChannel, PresenceTracker}, extractors::{PhilJSon, PhilJSQuery, SsrContext}, handlers::{health_check, not_found, ApiResponse}, middleware::PhilJSLayer, philjs::prelude::*, ssr::{HtmlDocument, MetaTag, Script}, state::{AppState, AppStateBuilder, Environment, CacheStats}, tower::{TracingLayer, TimeoutLayer, RequestIdLayer, SecurityHeadersLayer, RateLimitLayer}, websocket::{LiveViewSocket, LiveViewHandler, BroadcastChannel, PresenceTracker}
```

License

MIT

philjs-macros - Type: Rust crate - Purpose: Proc-macro crate for PhilJS - Rust procedural macros for building reactive web applications - Version: 0.1.0 - Location: packages/philjs-macros - Entry points: packages/philjs-macros/src/lib.rs - Keywords: philjs, macro, web, reactive, ui

philjs-macros

Procedural macros for PhilJS - ergonomic Rust macros for building reactive web applications.

Features

- `#[component]` - Transform functions into PhilJS components with automatic props handling
- `#[signal]` - Create reactive signals from struct fields with generated getters/setters
- `#[derive(Props)]` - Derive Props trait with builder pattern and validation
- `view!` - JSX-like syntax for building UI in Rust
- `#[server]` - Mark functions as server-only with automatic RPC generation

Installation

Add to your `Cargo.toml`:

```
[dependencies]
philjs-macros = "0.1.0"
```

Usage

Component Macro

Transform functions into reusable components:

```
use philjs_macros::component;

#[component]
fn Button(text: String, disabled: bool) -> impl IntoView {
    view! {
        <button disabled={disabled}>
            {text}
        </button>
    }
}

// Use the component
let props = ButtonProps {
    text: "Click me!".to_string(),
    disabled: false,
};
let button = Button(props);
```

Signal Macro

Create reactive state management:

```
use philjs_macros::signal;

#[signal]
struct AppState {
    count: i32,
    user: Option<String>,
}

let state = AppState::new(0, None);

// Use generated methods
state.set_count(5);
state.update_count(|c| *c += 1);
println!("Count: {}", state.count());
```

Props Macro

Derive builder pattern for component props:

```

use philjs_macros::Props;

#[derive(Props)]
struct CardProps {
    title: String,
    #[prop(default = "primary")]
    variant: &'static str,
    #[prop(optional)]
    description: Option<String>,
    #[prop(into)]
    content: String,
}

let props = CardProps::builder()
    .title("My Card".to_string())
    .content("Hello") // Accepts &str with #[prop(into)]
    .build();

```

View Macro

JSX-like syntax for building UI:

```

use philjs_macros::view;

let name = "World";
let items = vec!["Apple", "Banana", "Cherry"];

let ui = view! {
    <div class="container">
        <h1>Hello, " {name}</h1>
        <ul>
            {items.iter().map(|item| view! {
                <li>{item}</li>
            }).collect::<Vec<_>>()}
        </ul>
    </div>
};

```

Features: - Self-closing tags: - Namespaced attributes: <button on:click={handler}> - Expressions: {count}, {format!("x = {}", x)} - Conditionals: {show.then(|| view! { <div>Visible</div> })} - Loops: {items.iter().map(|i| ...).collect::<Vec<_>>()}

Server Macro

Create server functions with automatic client-side RPC:

```

use philjs_macros::server;

#[server]
async fn fetch_user(id: u32) -> Result<User, ServerError> {
    let db = get_database().await;
    db.get_user(id).await
}

// On the client, this automatically becomes an RPC call:
let user = fetch_user(42).await?;

```

Custom endpoints:

```

#[server(endpoint = "/api/custom")]
async fn custom_function(data: String) -> Result<(), Error> {
    // Server-only code
    Ok(())
}

#[server(prefix = "/v1")]
async fn versioned_api(param: i32) -> Result<i32, Error> {
    // Will be available at /v1/versioned_api
    Ok(param * 2)
}

```

Advanced Usage

Component with Generics

```

#[component]
fn List<T: Display>(items: Vec<T>) -> impl IntoView {
    view! {
        <ul>
            {items.iter().map(|item| view! {
                <li>{item}</li>
            }).collect::<Vec<_>>()}
        </ul>
    }
}

```

Transparent Components

For components that don't need props structs:

```
#[component(transparent)]
fn Title(text: String) -> impl IntoView {
    view! { <h1>{text}</h1> }
}

// Use directly without props struct
let title = Title("Hello".to_string());
```

Signal with Complex Types

```
#[derive(Clone)]
struct User {
    id: u32,
    name: String,
}

#[signal]
struct UserManager {
    current: Option<User>,
    all_users: Vec<User>,
}

let manager = UserManager::new(None, vec![]);
manager.update_all_users(|users| {
    users.push(User { id: 1, name: "Alice".to_string() });
});
```

Testing

Run the test suite:

```
cargo test
```

The test suite includes: - Component macro tests - Signal macro tests - Props derive tests - View macro tests - Server function tests

Architecture

This crate uses: - **syn** - Parsing Rust syntax - **quote** - Code generation - **proc-macro2** - Procedural macro utilities - **darling** - Attribute parsing

All macros are designed to: - Provide helpful error messages - Generate efficient code - Support generics and complex types - Match the ergonomics of Leptos/Dioxus

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Source files: packages/philjs-macros/src/lib.rs

Public API

- Public modules: (none detected)
- Public items: action, api, component, derive_props, derive_store, effect, into_store, layout, loader, memo, navigate, redirect, resource, route, server, signal, store, use_params, use_query, view
- Re-exports: (none detected)

License

MIT

philjs-mobile - Type: Rust crate - Purpose: Native mobile support for PhilJS - build iOS and Android apps with Rust - Version: 0.1.0 - Location: packages/philjs-mobile - Entry points: packages/philjs-mobile/src/lib.rs - Keywords: mobile, ios, android, rust, ui

@philjs/mobile

React Native and mobile support for PhilJS applications. Share components and logic between web and mobile with platform-specific adaptations.

Installation

```
npm install @philjs/mobile
# or
yarn add @philjs/mobile
# or
pnpm add @philjs/mobile
```

Basic Usage

```

import { View, Text, Button, useResponsive } from '@philjs/mobile';

function App() {
  const { isMobile, isTablet } = useResponsive();

  return (
    <View style={{ padding: isMobile ? 16 : 32 }}>
      <Text variant="heading">Welcome</Text>
      <Button onPress={() => console.log('Pressed!')}>
        Get Started
      </Button>
    </View>
  );
}

```

Features

- **Cross-Platform Components** - Write once, run on web and native
- **Platform Adapters** - Automatic platform-specific styling
- **Navigation** - Unified navigation for web and mobile
- **Gestures** - Touch gestures and animations
- **Device APIs** - Camera, location, notifications
- **Offline Storage** - AsyncStorage and SQLite support
- **Push Notifications** - FCM and APNs integration
- **Deep Linking** - Handle deep links and universal links
- **Responsive Design** - Adapt layouts for different screens
- **Native Modules** - Easy integration with native code
- **Hot Reloading** - Fast refresh during development
- **Over-the-Air Updates** - Deploy updates without app store

Components

Component	Description
View	Cross-platform container
Text	Typography component
Button	Touchable button
Image	Responsive images
Input	Text input field
ScrollView	Scalable container
FlatList	Optimized list
Modal	Modal dialogs

Hooks

Hook	Description
useResponsive	Screen size detection
usePlatform	Current platform info
useKeyboard	Keyboard visibility
useAppState	App foreground/background
useDeviceInfo	Device information

Platform Detection

```

import { Platform } from '@philjs/mobile';

if (Platform.OS === 'ios') {
  // iOS-specific code
} else if (Platform.OS === 'android') {
  // Android-specific code
} else {
  // Web code
}

```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Source files: packages/philjs-mobile/src/lib.rs

Public API

- Public modules: android, animation, biometrics, camera, components, gestures, haptics, in_app_purchase, ios, location, navigation, notifications, permissions, platform, prelude, renderer, runtime, sensors, share, storage
- Public items: AppState, BLACK, BLUE, Color, Constraints, Edgesets, FontWeight, GREEN, IntoView, KeyboardType, NativeComponent, NativeView, Orientation, Point, RED,

Rect, SafeArea, Size, TRANSPARENT, TextAlign, WHITE, all, fn, from_hex, from_origin_size, horizontal, loose, new, run, run_with_config, symmetric, tight, unbounded, vertical, zero

- Re-exports: animation::(AnimatedValue, SpringAnimation, TimingAnimation), components::*, gestures::(GestureRecognizer, Gesture, GestureState), haptics::(HapticFeedback, HapticStyle), navigation::(Navigator, Route, NavigationStack), notifications::(LocalNotification, PushNotification, NotificationHandler), permissions::(Permission, PermissionStatus, request_permission), platform::(Platform, PlatformInfo, DeviceInfo), renderer::(NativeRenderer, RenderContext), runtime::(MobileApp, MobileConfig, run, run_with_config), storage::(SecureStorage, AsyncStorage, FileSystem)

License

MIT

philjs-seaorm - Type: Rust crate - Purpose: SeaORM integration for PhilJS - entity-based database queries with active record pattern - Version: 0.1.0 - Location: packages/philjs-seaorm - Entry points: packages/philjs-seaorm/src/lib.rs - Keywords: seaorm, philjs, orm, database, entity

PhilJS SeaORM Integration

Entity-based database queries with active record pattern for PhilJS applications.

Features

- **Reactive Entity Queries:** Integrate SeaORM with PhilJS's reactive system
- **Entity Lifecycle Hooks:** Automatic validation and auditing
- **Pagination Helpers:** Offset and cursor-based pagination
- **Type-safe ORM:** Compile-time checked entity relationships
- **Migration Support:** Database schema management

Installation

```
[dependencies]
philjs-seaorm = { version = "2.0", features = ["sqlx-postgres"] }
```

Quick Start

```
use philjs_seaorm::prelude::*;
use sea_orm::entity::prelude::*;

#[derive(Clone, Debug, PartialEq, DeriveEntityModel)]
#[sea_orm(table_name = "users")]
pub struct Model {
    #[sea_orm(primary_key)]
    pub id: i64,
    pub name: String,
    pub email: String,
}

#[derive(Copy, Clone, Debug, EnumIter, DeriveRelation)]
pub enum Relation {}

impl ActiveModelBehavior for ActiveModel {}

// Query users
let users = Entity::find()
    .filter(Column::Active.eq(true))
    .all(&db)
    .await;
```

Reactive Queries

```
use philjs_seaorm::reactive::ReactiveEntity;

let users = ReactiveEntity::users::Entity::new(&db)
    .filter(users::Column::Active.eq(true))
    .order_by_asc(users::Column::Name)
    .limit(10);

// Use with PhilJS resources
let data = create_resource(
    || (),
    move |_| async move {
        users.all().await
    }
);
```

Lifecycle Hooks

```
use philjs_seaorm::hooks::HookedEntity, BeforeHook, LoggingHook;
use std::sync::Arc;

let hooked = HookedEntity::users::Entity::new(&db)
    .with_before_hook(Arc::new(LoggingHook::new("users")))
    .with_after_hook(Arc::new(AuditHook));

let user = hooked.insert(user_model).await;
```

Pagination

Offset Pagination

```

use philjs_seaorm::pagination::{Paginator, PaginationParams};

let params = PaginationParams::new(1, 20);
let result = Paginator::new($db, users::Entity::find())
    .paginate(params)
    .await?;

println!("Total: {}, Pages: {}", result.pagination.total, result.pagination.total_pages);

```

Cursor Pagination

```

use philjs_seaorm::pagination::{CursorPaginator, CursorParams};

let params = CursorParams { cursor: None, limit: 20 };
let result = CursorPaginator::new($db, posts::Entity::find())
    .by_column(posts::Column::Id)
    .paginate(params)
    .await?;

```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Source files: packages/philjs-seaorm/src/lib.rs

Public API

- Public modules: context, entity, error, hooks, migration, pagination, prelude, query, reactive
- Public items: ConnectionBuilder, build_options, connect, connect_timeout_secs, connect_with_options, idle_timeout_secs, max_connections, min_connections, new, sqlx_logging
- Re-exports: context::(provide_db, use_db, DbProvider), crate::context::(provide_db, use_db, DbProvider), crate::entity::(EntityHelpers, Pagination, SortOrder), crate::error::(OrmError, OrmResult), crate::hooks::(HookedEntity, BeforeHook, AfterHook, ValidationHook), crate::pagination::(Paginator, PaginationParams, PaginationMeta, PaginatedResult, CursorPaginator, CursorParams, CursorResult, PaginateExt,), crate::query::(QueryHelpers, FilterBuilder, RelationLoader), crate::reactive::(ReactiveEntity, ReactiveQueryBuilder, EntityResource), entity::(EntityHelpers, Pagination, SortOrder), error::(OrmError, OrmResult), migration::(Migrator, MigrationStatus), philjs::prelude::, query::(QueryHelpers, FilterBuilder, RelationLoader), sea_orm::entity::prelude::, sea_orm::entity::entity::, query::, ActiveModelTrait, ActiveValue, ColumnTrait, EntityTrait, ModelTrait, Set, DatabaseConnection, DatabaseTransaction, ConnectionTrait, TransactionTrait, FromQueryResult, IntoActiveModel, Related, RelationDef, RelationTrait, Condition, Order,), sea_orm::entity::, query::, ActiveModelTrait, ActiveValue, ColumnTrait, EntityTrait, ModelTrait, PrimaryKeyTrait, DatabaseConnection, DatabaseTransaction, ConnectOptions, ConnectionTrait, TransactionTrait, Statement, DbErr, DbBackend, ExecResult, FromQueryResult, IntoActiveModel, TryIntoModel, Related, RelationDef, RelationTrait, Condition, Order, Value,), sea_query::(Expr, Func, SimpleExpr)

License

MIT

philjs-sqlx - Type: Rust crate - Purpose: SQLx integration for PhilJS - type-safe SQL queries with compile-time verification - Version: 0.1.0 - Location: packages/philjs-sqlx -
Entry points: packages/philjs-sqlx/src/lib.rs - Keywords: sqlx, philjs, database, postgres, mysql

PhilJS SQLx Integration

Type-safe SQL queries with compile-time verification for PhilJS applications.

Features

- **Reactive Query Wrappers:** Integrate SQLx queries with PhilJS's reactive system
- **Transaction Helpers:** Safe transaction handling with automatic rollback
- **Migration Utilities:** Database schema migration support
- **Compile-time Verification:** Type-safe SQL with compile-time checks
- **Connection Pooling:** Efficient database connection management

Installation

```

[dependencies]
philjs-sqlx = { version = "2.0", features = ["postgres"] }

```

Quick Start

```

use philjs_sqlx::prelude::*;
use sqlx::FromRow;

#[derive(FromRow, Serialize, Deserialize)]
struct User {
    id: i64,
    name: String,
    email: String,
}

#[tokio::main]
async fn main() -> Result<(), Box

```

Reactive Queries

Using with PhilJS Resources

```

use philjs_sqlx::reactive::DbResource;

#[derive(FromRow)]
struct Post {
    id: i64,
    title: String,
    content: String,
}

// Create a reactive database resource
let posts = DbResource::new(&pool, "SELECT * FROM posts ORDER BY created_at DESC");

// Use in components
view! {
    <Suspense fallback=|| view! { <div>"Loading..."</div> }>
        {move || {
            posts.fetch().await.map(|posts| view! {
                <For each=move || posts.clone()
                    key=|post| post.id
                    children=|post| view! {
                        <div>
                            <h2>{post.title}</h2>
                            <p>{post.content}</p>
                        </div>
                    }
                />
            })
        }}
    </Suspense>
}

```

Query Builder

```

use philjs_sqlx::reactive::ReactiveQueryBuilder;

let users = ReactiveQueryBuilder::new("users")
    .select(&["id", "name", "email"])
    .where_clause("active = true")
    .where_clause("verified = true")
    .order_by("name ASC")
    .limit(10)
    .offset(0)
    .fetch_all(&pool)
    .await?;

```

Transactions

Simple Transactions

```

use philjs_sqlx::transaction::with_transaction;

with_transaction(&pool, |tx| async move {
    sqlx::query("INSERT INTO users (name, email) VALUES ($1, $2)")
        .bind("Alice")
        .bind("alice@example.com")
        .execute(&mut *tx)
        .await?;

    sqlx::query("INSERT INTO profiles (user_id, bio) VALUES ($1, $2)")
        .bind(1)
        .bind("Software developer")
        .execute(&mut *tx)
        .await?;

    Ok(())
}).await?;

```

Transaction with Savepoints

```

use philjs_sqlx::transaction::{with_transaction, Savepoint};

with_transaction(&pool, |tx| async move {
    // Insert user
    sqlx::query("INSERT INTO users (name) VALUES ($1)")
        .bind("Bob")
        .execute(&mut *tx)
        .await?;

    // Try to create profile with savepoint
    let sp = Savepoint::new(tx, "profile_creation").await?;

    match sqlx::query("INSERT INTO profiles (user_id, bio) VALUES ($1, $2)")
        .bind(2)
        .bind("Engineer")
        .execute(&mut *sp.transaction())
        .await
    {
        Ok(_) => sp.release().await?,
        Err(_) => sp.rollback().await?,
    }
}

Ok(())
).await?;

```

Retry on Transient Errors

```

use philjs_sqlx::transaction::retry_transaction;

retry_transaction(&pool, 3, |tx| async move {
    sqlx::query("UPDATE counters SET value = value + 1 WHERE id = $1")
        .bind(1)
        .execute(&mut *tx)
        .await?;

    Ok(())
}).await?;

```

Batch Operations

```

use philjs_sqlx::transaction::BatchOperation;

let users = vec![
    ("Alice", "alice@example.com"),
    ("Bob", "bob@example.com"),
    ("Carol", "carol@example.com"),
];

BatchOperation::new(&pool)
    .batch_size(100)
    .execute(users, |tx, (name, email)| async move {
        sqlx::query("INSERT INTO users (name, email) VALUES ($1, $2)")
            .bind(name)
            .bind(email)
            .execute(&mut *tx)
            .await?;

        Ok(())
    })
    .await?;

```

Migrations

Running Migrations

```

use philjs_sqlx::migrate::{run_migrations, MigrationRunner};

// Simple migration run
run_migrations(&pool, "./migrations").await?;

// With migration runner
let runner = MigrationRunner::new(&pool, "./migrations");
runner.run().await?;

```

Reverting Migrations

```

let runner = MigrationRunner::new(&pool, "./migrations");
runner.revert().await?;

```

Server Functions

```

use philjs_sqlx::prelude::*;

#[server]
async fn get_users() -> Result<Vec<User>, ServerFnError> {
    let pool = use_pool()?;
    let users = sqlx::query_as!(
        User,
        "SELECT id, name, email FROM users WHERE active = $1",
        true
    )
    .fetch_all(&pool)
    .await
    .map_err(|e| ServerFnError::ServerError(e.to_string()))?;
    Ok(users)
}

#[server]
async fn create_user(name: String, email: String) -> Result<User, ServerFnError> {
    let pool = use_pool()?;
    let user = sqlx::query_as!(
        User,
        "INSERT INTO users (name, email) VALUES ($1, $2) RETURNING id, name, email",
        name,
        email
    )
    .fetch_one(&pool)
    .await
    .map_err(|e| ServerFnError::ServerError(e.to_string()))?;
    Ok(user)
}

```

Connection Pooling

Custom Pool Configuration

```

use philjs_sqlx::pool::PoolConfig;

let config = PoolConfig::new("postgres://localhost/mydb")
    .max_connections(20)
    .min_connections(5)
    .connect_timeout_secs(30)
    .idle_timeout_secs(600)
    .test_on_acquire(true);

let pool = config.create().await?;

```

Context Integration

```

use philjs_sqlx::context::{provide_pool, use_pool};

// Provide pool in your app setup
provide_pool(pool.clone());

// Use in components or server functions
let pool = use_pool()?;

```

Helper Macros

```

use philjs_sqlx::{db_query_as, db_query_one, db_scalar};

// Query all
let users: Vec<User> = db_query_as!(User, &pool, "SELECT * FROM users")?;

// Query one
let user: User = db_query_one!(User, &pool, "SELECT * FROM users WHERE id = $1", 1)?;

// Query scalar
let count: i64 = db_scalar!(i64, &pool, "SELECT COUNT(*) FROM users")?;

```

Features

```

[dependencies]
philjs-sqlx = { version = "2.0", features = ["postgres", "migrate", "uuid", "chrono"] }

```

Available features: - postgres (default): PostgreSQL support - mysql: MySQL support - sqlite: SQLite support - mssql: Microsoft SQL Server support - migrate: Migration support - uuid: UUID support - chrono: DateTime support - json: JSON support

Complete Example

```

use philjs_sqlx::prelude::*;
use philjs_sqlx::transaction::with_transaction;
use philjs_sqlx::reactive::ReactiveQueryBuilder;
use sqlx::FromRow;
use serde::{Deserialize, Serialize};

#[derive(Debug, Clone, FromRow, Serialize, Deserialize)]
struct User {
    id: i64,
    name: String,
    email: String,
}

#[tokio::main]
async fn main() -> Result<(), Box<dyn std::error::Error>> {
    // Create pool
    let pool = create_pool("postgres://localhost/mydb").await?;

    // Run migrations
    run_migrations(&pool, "./migrations").await?;

    // Insert user in transaction
    with_transaction(&pool, |tx| async move {
        sqlx::query("INSERT INTO users (name, email) VALUES ($1, $2)")
            .bind("Alice")
            .bind("alice@example.com")
            .execute(&mut *tx)
            .await?;
        Ok(())
    }).await?;

    // Query users
    let users = ReactiveQueryBuilder::new("users")
        .select(<[<"id", "name", "email">]>)
        .where_clause("active = true")
        .order_by("name ASC")
        .fetch_all::<_, _, User>(&pool)
        .await?;

    println!("Found {} users", users.len());

    Ok(())
}

```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Source files: packages/philjs-sqlx/src/lib.rs

Public API

- Public modules: context, error, migrate, migration, pool, prelude, query, reactive, transaction
- Public items: (none detected)
- Re-exports: context::(provide_pool, use_pool, DbContext), create::context::(provide_pool, use_pool, DbContext), create::error::(DbError, DbResult), create::migrate::(run_migrations, MigrationRunner), create::pool::(DbPool, PoolConfig, create_pool), create::query::(Query, QueryBuilder, Executor), create::reactive::(ReactiveQuery, DbResource, ReactiveQueryBuilder), create::transaction::(TransactionHelper, with_transaction, Savepoint, BatchOperation, retry_transaction,), error::(DbError, DbResult), philjs::prelude::*, pool::(DbPool, PoolConfig, create_pool), query::(Query, QueryBuilder, Executor), sqlx::(FromRow, Row, Column, TypeInfo, ValueRef, query, query_as, query_scalar, Encode, Decode, Type,), sqlx::(FromRow, Row, query, query_as, query_scalar), sqlx::(MySql, MySqlPool, MySqlConnection, MySqlRow), sqlx::(MySql, MySqlPool), sqlx::(Postgres, PgPool, PgConnection, PgRow), sqlx::(Postgres, PgPool), sqlx::(SQLite, SqlitePool, SqliteConnection, SqliteRow), sqlx::(SQLite, SqlitePool)

License

MIT

philjs-tauri - Type: Rust crate - Purpose: Tauri integration for PhilJS - build desktop apps with web technologies and Rust - Version: 0.1.0 - Location: packages/philjs-tauri -

Entry points: packages/philijs-tauri/src/lib.rs - Keywords: tauri, desktop, philijs, rust, webview

@philijs/tauri

Tauri desktop application integration for PhiliJS. Build native desktop apps with web technologies using Rust for the backend and PhiliJS for the frontend.

Installation

```
npm install @philijs/tauri
# or
yarn add @philijs/tauri
# or
pnpm add @philijs/tauri
```

Rust (Cargo.toml):

```
[dependencies]
philijs-tauri = "0.1"
tauri = "1.5"
```

Basic Usage

TypeScript:

```
import { invoke, useTauriCommand, useWindow } from '@philijs/tauri';

function App() {
  const { data: config } = useTauriCommand('get_config');
  const { minimize, maximize, close } = useWindow();

  const handleSave = async () => {
    await invoke('save_file', { path: '/tmp/data.json', content: data });
  };

  return (
    <div>
      <TitleBar onMinimize={minimize} onMaximize={maximize} onClose={close} />
      <button onClick={handleSave}>Save</button>
    </div>
  );
}
```

Rust:

```
use philijs_tauri::prelude::*;

#[tauri::command]
async fn save_file(path: String, content: String) -> Result<(), String> {
    std::fs::write(&path, &content).map_err(|e| e.to_string())
}

fn main() {
    tauri::Builder::default()
        .invoke_handler(tauri::generate_handler![save_file])
        .run(tauri::generate_context!())
        .expect("error running app");
}
```

Features

- **Native Performance** - Rust backend for speed and efficiency
- **Small Bundle** - Tiny app size compared to Electron
- **System Tray** - Native system tray integration
- **Window Management** - Multi-window and custom titlebars
- **File System** - Native file dialogs and operations
- **Notifications** - System notifications
- **Clipboard** - Read/write clipboard
- **Auto Updates** - Built-in update mechanism
- **Deep Links** - Custom protocol handlers
- **Native Menus** - OS-native application menus
- **Security** - Sandboxed by default
- **Cross-Platform** - Windows, macOS, Linux

Hooks

Hook	Description
useTauriCommand	Call Rust commands
useWindow	Window controls
useFilesystem	File operations
useNotification	System notifications
useClipboard	Clipboard access
useAppUpdate	Auto-update state

Components

Component	Description
TitleBar	Custom window titlebar
TrayMenu	System tray menu
NativeMenu	Application menu
Dialog	Native file dialogs

CLI

```
# Create new Tauri + PhilJS project
npx create-philjs-tauri my-app

# Development mode
npm run tauri dev

# Build for production
npm run tauri build
```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Source files: packages/philjs-tauri/src/lib.rs

Public API

- Public modules: clipboard, commands, dialog, fs, menu, notification, prelude, process, shell, state, tray, updater, window
- Public items: Builder, PhilJSApp, TauriConfig, TauriError, clipboard, config, create_state, create_window, exit, get_window, handle, init, invoke_handler, main_window, new, notify, restart, run, setup, title
- Re-exports: crate:(Builder, PhilJSApp, TauriConfig, commands:*, window:{Window, WindowBuilder, WindowConfig}, tray:{TrayIcon, TrayBuilder}, menu:{MenuItem, MenuBuilder}, dialog:{Dialog, FileDialog, MessageDialog}, clipboard:{Clipboard, notification:Notification, }, tauri:{self, AppHandle, Manager, State, Wry})

License

MIT

philjs-tokio - Type: Rust crate - Purpose: Tokio integration for PhilJS - async runtime configuration, task spawning, and channel utilities - Version: 0.1.0 - Location: packages/philjs-tokio - Entry points: packages/philjs-tokio/src/lib.rs - Keywords: tokio, philjs, async, runtime, concurrency

@philjs/tokio

Tokio runtime integration for PhilJS applications. Connect async Rust backends powered by Tokio with PhilJS frontends for high-performance, concurrent applications.

Installation

```
npm install @philjs/tokio
# or
yarn add @philjs/tokio
# or
pnpm add @philjs/tokio
```

Rust (Cargo.toml):

```
[dependencies]
philjs-tokio = "0.1"
tokio = { version = "1", features = ["full"] }
```

Basic Usage

TypeScript:

```

import { TokioClient, useTokioStream, useTokioTask } from '@philjs/tokio';

const client = new TokioClient('ws://localhost:9000');

function Dashboard() {
  // Subscribe to real-time data stream
  const { data: metrics } = useTokioStream('system_metrics');

  // Execute async task
  const { execute, isRunning } = useTokioTask('process_data');

  return (
    <div>
      <MetricsChart data={metrics} />
      <button onClick={() => execute({ batch: 1000 })} disabled={isRunning}>
        Process Data
      </button>
    </div>
  );
}

```

Rust:

```

use philjs_tokio::prelude::*;
use tokio::sync::broadcast;

#[philjs_stream]
async fn system_metrics(tx: broadcast::Sender<Metrics>) {
    loop {
        let metrics = collect_metrics().await;
        tx.send(metrics).unwrap();
        tokio::time::sleep(Duration::from_secs(1)).await;
    }
}

#[philjs_task]
async fn process_data(batch: usize) -> Result<ProcessResult, Error> {
    let data = fetch_batch(batch).await?;
    process_batch(data).await
}

#[tokio::main]
async fn main() {
    PhilJSRuntime::new()
        .stream("system_metrics", system_metrics)
        .task("process_data", process_data)
        .serve("0.0.0.0:9000")
        .await;
}

```

Features

- **Async Streams** - Subscribe to real-time Tokio channels
- **Task Execution** - Run async tasks from the frontend
- **Connection Pool** - Efficient WebSocket connection management
- **Backpressure** - Handle fast producers gracefully
- **Reconnection** - Automatic reconnection with backoff
- **Type Safety** - Generated TypeScript types from Rust
- **Concurrency** - Leverage Tokio's async runtime
- **Cancellation** - Cancel running tasks from frontend
- **Progress** - Track long-running task progress
- **Batching** - Batch multiple requests efficiently

Hooks

Hook	Description
useTokioStream	Subscribe to async streams
useTokioTask	Execute async tasks
useTokioState	Shared state synchronization
useTokioChannel	Bidirectional channels

Macros

```

#[philjs_stream] // Define a subscribable stream
#[philjs_task] // Define an executable task
#[philjs_state] // Define synchronized state
#[philjs_channel] // Define bidirectional channel

```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Source files: packages/philjs-tokio/src/lib.rs

Public API

- Public modules: channel, interval, prelude, runtime, sync, task, timeout
- Public items: (none detected)
- Re-exports: channel::(channel, broadcast, watch, Channel, Sender, Receiver), crate::channel::(channel, broadcast, watch), crate::interval::(spawn_interval, IntervalHandle), crate::runtime::(RuntimeBuilder, RuntimeConfig), crate::sync::(Mutex, RwLock, Semaphore), crate::task::(spawn_task, spawn_blocking_task, TaskHandle, TaskManager), crate::timeout::(with_timeout, TimeoutError), interval::(spawn_interval, IntervalHandle), philjs::prelude::*, runtime::(RuntimeBuilder, RuntimeConfig), sync::(Mutex, RwLock, Semaphore), task::(spawn_task, spawn_blocking_task, TaskHandle, TaskManager), timeout::(with_timeout, TimeoutError), tokio::(spawn, time::(sleep, Duration, Instant), select, join, try_join,), tokio::(spawn, task::JoinHandle, time::(sleep, Duration, Instant, interval as tokio_interval), sync::(mpsc, oneshot, broadcast as tokio_broadcast, watch as tokio_watch), select, join, try_join,)

License

MIT

philjs-tui - Type: Rust crate - Purpose: Terminal UI rendering for PhilJS - build TUI apps with the same component model - Version: 0.1.0 - Location: packages/philjs-tui - Entry points: packages/philjs-tui/src/lib.rs - Keywords: tui, terminal, cli, philjs, ratatui

@philjs/tui

Terminal UI components for building CLI applications with Rust. Create rich, interactive terminal interfaces with a React-like component model.

Installation

Rust (Cargo.toml):

```
[dependencies]
philjs-tui = "0.1"
```

Basic Usage

```
use philjs_tui::prelude::*;

fn main() -> Result<()> {
    let app = App::new()
        .title("My TUI App")
        .component(Dashboard);

    app.run()
}

#[component]
fn Dashboard() -> Element {
    let (count, set_count) = use_state(0);

    column![
        text!("Counter: {}", count).bold(),
        row![
            button!("Increment", || set_count(count + 1)),
            button!("Decrement", || set_count(count - 1)),
        ],
        progress_bar!(count as f64 / 100.0),
    ]
}
```

Features

- **Component Model** - React-like functional components
- **State Management** - Hooks for state and effects
- **Layout System** - Flexbox-inspired layouts
- **Widgets** - Pre-built UI components
- **Styling** - Colors, borders, and text styles
- **Input Handling** - Keyboard and mouse events
- **Async Support** - Tokio integration for async operations
- **Animations** - Smooth transitions and animations
- **Unicode** - Full Unicode and emoji support
- **Responsive** - Adapts to terminal size
- **Accessibility** - Screen reader friendly

Components

Component	Description
text!	Styled text display
input!	Text input field
button!	Clickable button
select!	Dropdown selection
checkbox!	Checkbox toggle
progress_bar!	Progress indicator
table!	Data table
list!	Scrollable list
tabs!	Tab navigation
modal!	Modal dialog
chart!	ASCII charts

Layout

```
// Vertical stack
column![
    text!("Header"),
    content,
    text!("Footer"),
]

// Horizontal row
row![
    sidebar,
    main_content,
]

// Grid Layout
grid!(2, 2>[
    cell1, cell2,
    cell3, cell4,
]
```

Hooks

Hook	Description
use_state	Component state
use_effect	Side effects
use_async	Async operations
use_input	Input handling
use_size	Terminal dimensions

Styling

```
text!("Hello")
    .bold()
    .fg(Color::Green)
    .bg(Color::Black)
    .padding(1)
    .border(Border::Rounded)
```

API Snapshot

This section is generated from the package source. Run `node scripts/generate-package-atlas.mjs` to refresh.

Entry Points

- Source files: packages/philijs-tui/src/lib.rs

Public API

- Public modules: app, components, event, layout, prelude, render, style, widgets
- Public items: TuiApp, TuiConfig, TuiError, draw, new, quit, run, run_with_config, should_quit, size
- Re-exports: crate::{run, run_with_config, TuiApp, TuiConfig, TuiError, components::, layout::, style::, event::, widgets::*}, ratatui::style::{Color, Modifier, Style}

License

MIT

PhilJS Platform Adapters - Complete Implementation

This document provides a comprehensive overview of all platform adapters implemented for PhilJS.

Overview

PhilJS now includes SvelteKit-style comprehensive platform adapters for major deployment targets:

1. **Cloudflare Pages** - Complete edge platform with KV, D1, R2, Durable Objects
2. **Vercel** - Enhanced Edge and Serverless with ISR, KV, Blob
3. **Netlify** - Edge Functions, Serverless, Blob storage
4. **AWS Lambda** - Full AWS integration with multiple IaC options
5. **Railway** - Docker and Nixpacks deployment

Cloudflare Pages Adapter

Location: `src/cloudflare-pages/index.ts`

Features

- **KV Namespace**: Global key-value storage with automatic replication
- **D1 Database**: SQLite at the edge with SQL query support
- **R2 Storage**: S3-compatible object storage without egress fees
- **Durable Objects**: Stateful serverless objects with coordination
- **Service Bindings**: Connect multiple Workers together
- **Queue Bindings**: Producer/consumer pattern support
- **Analytics Engine**: Real-time analytics data collection
- **TypeScript Bindings**: Auto-generated types for all bindings

Usage

```
import { cloudflarePagesAdapter } from '@philjs/adapters/cloudflare-pages';

export default cloudflarePagesAdapter({
  kv: [
    {
      binding: 'CACHE',
      id: 'production-kv-id',
      preview_id: 'preview-kv-id'
    }
  ],
  d1: [
    {
      binding: 'DB',
      database_id: 'xxxx',
      database_name: 'production'
    }
  ],
  r2: [
    {
      binding: 'UPLOADS',
      bucket_name: 'user-uploads'
    }
  ],
  durableObjects: [
    {
      binding: 'COUNTER',
      class_name: 'Counter'
    }
  ],
  queues: {
    producers: [
      { binding: 'TASKS', queue: 'background-tasks' }
    ]
  }
});
```

Generated Files

- `_worker.js` - Main Worker script
- `_routes.json` - Route configuration for static assets
- `wrangler.toml` - Local development configuration
- `env.d.ts` - TypeScript type definitions

Helper Functions

```
// KV helpers
const kv = createKVNamespace(env.CACHE);
await kv.put('key', 'value', { expirationTtl: 3600 });

// D1 helpers
const db = createD1Database(env.DB);
const results = await db.prepare('SELECT * FROM users').all();

// R2 helpers
const bucket = createR2Bucket(env.UPLOADS);
await bucket.put('file.txt', 'content');
```

Vercel Adapter

Location: [src/vercel/adapter.ts](#)

Features

- **Edge Runtime:** Deploy to Vercel's global edge network
- **Serverless Functions:** Traditional Node.js functions
- **ISR:** Incremental Static Regeneration with on-demand revalidation
- **Vercel KV:** Redis-compatible key-value storage
- **Vercel Blob:** Object storage for files and media
- **Edge Config:** Ultra-low latency configuration
- **Image Optimization:** Automatic image format conversion and resizing
- **Cron Jobs:** Scheduled function execution

Usage

```
import { vercelAdapter } from '@philjs/adapters/vercel/adapter';

export default vercelAdapter({
  edge: true,
  regions: ['iad1', 'sfo1'],
  maxDuration: 30,
  isr: {
    expiration: 60,
    bypassToken: process.env.REVALIDATE_TOKEN
  },
  kv: {
    database: 'production'
  },
  blob: true,
  edgeConfig: {
    id: process.env.EDGE_CONFIG_ID
  },
  images: {
    domains: ['cdn.example.com'],
    formats: ['image/avif', 'image/webp'],
    deviceSizes: [640, 750, 828, 1080, 1200]
  },
  crons: [
    {
      path: '/api/cron/daily',
      schedule: '0 0 * * *'
    }
  ]
});
```

Generated Files

- .vercel/output/config.json - Build Output API v3 configuration
- .vercel/output/functions/index.func/ - Function handler
- .vercel/output/static/ - Static assets
- types.d.ts - TypeScript definitions

Revalidation API

```
import { revalidatePath, revalidateTag } from '@philjs/adapters/vercel/adapter';

// Revalidate specific path
await revalidatePath('/blog/post-1');

// Revalidate by tag
await revalidateTag('blog-posts');
```

Netlify Adapter

Location: [src/netlify/adapter.ts](#)

Features

- **Edge Functions:** Deploy to Netlify's edge network with Deno runtime
- **Netlify Functions:** Node.js serverless functions
- **Blob Storage:** Key-value blob storage
- **Form Handling:** Built-in form processing
- **Redirects & Rewrites:** Advanced routing with geo/role conditions
- **Split Testing:** A/B testing support
- **Image CDN:** Automatic image optimization

Usage

```

import { netlifyAdapter } from '@philjs/adapters/netlify/adapter';

export default netlifyAdapter({
  edge: true,
  blob: true,
  redirects: [
    {
      from: '/old-path',
      to: '/new-path',
      status: 301,
      conditions: {
        country: ['US', 'CA']
      }
    }
  ],
  headers: [
    {
      for: '/api/*',
      values: {
        'Access-Control-Allow-Origin': '*',
        'Cache-Control': 'no-cache'
      }
    }
  ],
  splitTesting: [
    {
      path: '/landing',
      branches: [
        { branch: 'main', weight: 50 },
        { branch: 'variant-a', weight: 50 }
      ]
    }
  ]
});

```

Generated Files

- `netlify.toml` - Build and deployment configuration
- `_redirects` - Redirect rules
- `_headers` - HTTP headers configuration
- `edge-functions/` or `functions/` - Function handlers

Image Optimization

```

import { netlifyImageCDN } from '@philjs/adapters/netlify/adapter';

const url = netlifyImageCDN('/image.jpg', {
  width: 800,
  height: 600,
  fit: 'cover',
  quality: 80
});

```

AWS Lambda Adapter

Location: `src/aws-lambda/index.ts`

Features

- **Lambda Functions:** Serverless compute on AWS
- **API Gateway:** REST and HTTP APIs
- **CloudFront:** Global CDN integration
- **S3 Static Assets:** Static file hosting with caching
- **Lambda@Edge:** Edge computing for CloudFront
- **ALB Support:** Application Load Balancer integration
- **Multiple IaC Options:** SAM, Serverless Framework, Terraform

Usage

```

import { awsLambdaAdapter } from '@philjs/adapters/aws-lambda';

export default awsLambdaAdapter({
  region: 'us-east-1',
  runtime: 'nodejs24.x',
  memorySize: 1024,
  timeout: 30,
  integration: 'http-api',
  s3: {
    bucket: 'my-static-assets',
    region: 'us-east-1',
    cacheControl: 'public, max-age=31536000'
  },
  cloudfront: {
    distributionId: 'E1234567890ABC'
  },
  vpc: {
    subnetIds: ['subnet-1', 'subnet-2'],
    securityGroupIds: ['sg-1']
  },
  generateSAM: true,
  generateServerless: true,
  generateTerraform: true
});

```

Generated Files

- lambda/index.js - Lambda handler (API Gateway, HTTP API, ALB, or Lambda@Edge)
- template.yaml - AWS SAM template
- serverless.yml - Serverless Framework configuration
- main.tf - Terraform configuration
- deploy.sh - Deployment script

Integration Types

1. **API Gateway REST API:** Full REST API with stage management
2. **HTTP API:** Simplified HTTP API with lower costs
3. **ALB:** Application Load Balancer integration
4. **Lambda@Edge:** CloudFront edge functions
5. **CloudFront Functions:** Lightweight edge functions

Railway Adapter

Location: src/railway/index.ts

Features

- **Docker Support:** Automated Dockerfile generation
- **Nixpacks:** Automatic buildpack detection
- **Railway.toml:** Configuration management
- **Health Checks:** Built-in health monitoring
- **Graceful Shutdown:** Clean process termination
- **Static Files:** Optimized static asset serving
- **Compression:** gzip compression support

Usage

```

import { railwayAdapter } from '@philjs/adapters/railway';

export default railwayAdapter({
  docker: {
    baseImage: 'node:24-alpine',
    nodeVersion: '24',
    packages: ['python3', 'make', 'g++'],
    buildArgs: {
      NODE_ENV: 'production'
    }
  },
  railway: {
    buildCommand: 'npm install && npm run build',
    startCommand: 'npm start',
    healthCheckPath: '/health',
    healthCheckInterval: 300,
    restartPolicy: 'on-failure',
    region: 'us-west1'
  },
  nixpacks: {
    packages: ['nodejs', 'npm'],
    buildCommand: 'npm run build'
  },
  gracefulShutdown: {
    timeout: 30000,
    signals: ['SIGTERM', 'SIGINT']
  }
});

```

Generated Files

- Dockerfile - Docker configuration
- .dockerrcignore - Docker ignore patterns
- railway.toml - Railway configuration
- nixpacks.toml - Nixpacks configuration
- server.ts - Node.js server with health checks (TypeScript entry)
- DEPLOY.md - Deployment guide

Common Features

All adapters include:

1. **TypeScript Support**: Full type definitions
2. **Build Output**: Optimized production builds
3. **Static Assets**: Efficient static file handling
4. **Environment Variables**: Secure configuration management
5. **Source Maps**: Optional source map generation
6. **Examples**: Working example projects
7. **Documentation**: Comprehensive README files

Package Exports

All adapters are properly exported in package.json:

```
{
  "exports": {
    "./cloudflare-pages": "./dist/cloudflare-pages/index.js",
    "./vercel/adapter": "./dist/vercel/adapter.js",
    "./netlify/adapter": "./dist/netlify/adapter.js",
    "./aws-lambda": "./dist/aws-lambda/index.js",
    "./railway": "./dist/railway/index.js"
  }
}
```

Adapter Presets

New presets added to the main package:

```

import { createAdapter } from '@philjs/adapters';

const adapter = createAdapter('railway');
const dockerAdapter = createAdapter('railway-docker');

```

Examples

Each adapter includes complete example projects:

- examples/cloudflare-pages/ - Cloudflare Pages with KV, D1, R2
- examples/vercel/ - Vercel with Edge Functions and ISR
- examples/netlify/ - Netlify with Edge Functions and Blob

- `examples/aws-lambda/` - AWS Lambda with SAM template
- `examples/railway/` - Railway with Docker configuration

Testing

All adapters support local development:

```
# Cloudflare Pages
wrangler pages dev .cloudflare

# Vercel
vercel dev

# Netlify
netlify dev

# AWS Lambda
sam local start-api

# Railway
railway run npm start
```

Migration Guide

From Basic Cloudflare to Cloudflare Pages

```
// Before
import { cloudflareAdapter } from '@philjs/adapters/cloudflare';

// After
import { cloudflarePagesAdapter } from '@philjs/adapters/cloudflare-pages';
```

From Basic Vercel to Enhanced Vercel

```
// Before
import { vercelAdapter } from '@philjs/adapters/vercel';

// After
import { vercelAdapter } from '@philjs/adapters/vercel/adapter';
```

Performance Optimizations

Each adapter includes platform-specific optimizations:

1. **Cloudflare**: Global edge caching, KV for session storage
2. **Vercel**: Edge caching, ISR for dynamic content
3. **Netlify**: Edge Functions for low latency
4. **AWS Lambda**: Provisioned concurrency, Lambda layers
5. **Railway**: Docker multi-stage builds, compression

Security Features

All adapters implement security best practices:

- HTTPS-only deployment
- Security headers (CSP, HSTS, etc.)
- Environment variable encryption
- VPC support (AWS)
- Authentication middleware hooks

Monitoring & Logging

Built-in support for platform monitoring:

- **Cloudflare**: Analytics Engine, Workers Analytics
- **Vercel**: Analytics, Web Vitals
- **Netlify**: Analytics, Function logs
- **AWS**: CloudWatch, X-Ray tracing
- **Railway**: Built-in metrics and logs

Cost Optimization

Each adapter optimizes for platform pricing:

- Efficient bundling to reduce cold starts
- Static asset caching to reduce function invocations
- Edge caching to minimize origin requests
- Resource sizing recommendations

Next Steps

1. Review individual adapter documentation
2. Choose the appropriate adapter for your deployment target
3. Configure environment variables
4. Deploy your PhilJS application
5. Monitor performance and optimize as needed

Support

For issues or questions: - GitHub Issues: <https://github.com/yourusername/philjs/issues> - Documentation: <https://philjs.dev> - Discord: <https://discord.gg/philjs>

PhilJS Edge Middleware

Next.js-style edge middleware for PhilJS applications. Run at the edge for optimal performance with Cloudflare Workers, Vercel Edge, Deno Deploy, and other edge runtimes.

Features

- **Edge Middleware System:** Request/response rewriting at edge with middleware chaining
- **Geolocation Routing:** Country/region/city detection with geo-based redirects
- **A/B Testing at Edge:** Cookie-based variant assignment with zero layout shift
- **Edge Caching:** Cache API integration with stale-while-revalidate

Installation

```
npm install @philjs/api
```

Basic Usage

Edge Middleware

```
import { executeEdgeMiddleware, type EdgeMiddleware } from '@philjs/api/edge-middleware';

// Simple middleware
const loggingMiddleware: EdgeMiddleware = async (context) => {
  console.log(`${context.request.method} ${context.request.url.pathname}`);
  return context.next();
};

// Execute middleware
export default {
  async fetch(request: Request) {
    return executeEdgeMiddleware(request, loggingMiddleware);
  },
};
```

URL Rewrites

```
import { rewriteMiddleware } from '@philjs/api/edge-middleware';

const rewrites = rewriteMiddleware({
  // Rewrite old paths to new paths
  '/old-api/*': '/api/v2/$1',
  '/blog/:slug': '/posts/$1',
});
```

Redirects

```
import { redirectMiddleware } from '@philjs/api/edge-middleware';

const redirects = redirectMiddleware({
  '/old-page': 'new-page',
  '/legacy/*': { destination: '/modern/$1', permanent: true },
});
```

Header Manipulation

```

import { addHeadersMiddleware, securityHeadersMiddleware } from '@philjs/api/edge-middleware';

// Add custom headers
const customHeaders = addHeadersMiddleware({
  'X-Custom-Header': 'value',
  'X-API-Version': '0.1.0',
});

// Security headers
const security = securityHeadersMiddleware({
  csp: "default-src 'self'", 
  hsts: true,
  nosniff: true,
  frameOptions: 'DENY',
});

```

Geolocation

Auto-detect Location

```

import { detectGeolocation } from '@philjs/api/geolocation';

// Works with Cloudflare Workers, Vercel Edge, etc.
const geo = await detectGeolocation(request);
console.log(geo.country, geo.city); // "US", "San Francisco"

```

Geo-based Redirects

```

import { redirectByCountry } from '@philjs/api/geolocation';

const geoRedirect = redirectByCountry({
  'GB,IE': '/uk',
  'FR,BE,LU': '/fr',
  'DE,AT,CH': '/de',
});

```

Language Detection

```

import { languageDetectionMiddleware } from '@philjs/api/geolocation';

const langDetection = languageDetectionMiddleware({
  cookieName: 'preferred-language',
  headerName: 'X-Detected-Language',
});

```

Localized Redirects

```

import { localizedRedirectMiddleware } from '@philjs/api/geolocation';

const localized = localizedRedirectMiddleware({
  supportedLocales: ['en', 'fr', 'de', 'es'],
  defaultLocale: 'en',
});

// Automatically redirects /products to /fr/products for French users

```

Client-side Hook

```

import { useGeolocation } from '@philjs/api/geolocation';

function MyComponent() {
  const { geo, language } = useGeolocation();

  return (
    <div>
      <p>Country: {geo?.country}</p>
      <p>Language: {language}</p>
    </div>
  );
}

```

A/B Testing

Define Experiments

```

import { abTestingMiddleware } from '@philjs/api/edge-ab-testing';

const abTesting = abTestingMiddleware({
  experiments: [
    {
      id: 'checkout-flow',
      name: 'Checkout Flow Test',
      variants: [
        { id: 'control', name: 'Control', weight: 50 },
        { id: 'new-design', name: 'New Design', weight: 50 },
      ],
      targeting: {
        countries: ['US', 'CA'],
        urlPatterns: ['/checkout/*'],
      },
    },
  ],
  onAssignment: async (assignment, context) => {
    // Track to analytics
    console.log(`Assigned variant: ${assignment.variantName}`);
  },
});

```

Variant-based Rendering

```

import { variantMiddleware } from '@philjs/api/edge-ab-testing';

const variantRouting = variantMiddleware('checkout-flow', {
  control: async (context) => {
    context.rewrite('/checkout/control');
    return context.next();
  },
  'new-design': async (context) => {
    context.rewrite('/checkout/new-design');
    return context.next();
  },
});

```

Multivariate Testing

```

import { multivariateTestingMiddleware } from '@philjs/api/edge-ab-testing';

const mvt = multivariateTestingMiddleware({
  id: 'homepage',
  name: 'Homepage MVT',
  factors: [
    {
      id: 'headline',
      name: 'Headline',
      variants: [
        { id: 'h1', name: 'Headline 1', weight: 50 },
        { id: 'h2', name: 'Headline 2', weight: 50 },
      ],
    },
    {
      id: 'cta',
      name: 'Call to Action',
      variants: [
        { id: 'c1', name: 'Learn More', weight: 50 },
        { id: 'c2', name: 'Get Started', weight: 50 },
      ],
    },
  ],
});

```

Client-side Hook

```

import { useVariant } from '@philjs/api/edge-ab-testing';

function CheckoutPage() {
  const { variant } = useVariant('checkout-flow');

  return (
    <div>
      {variant === 'Control' ? <ControlCheckout /> : <NewCheckout />}
    </div>
  );
}

```

Statistical Significance

```

import { calculateSignificance } from '@philjs/api/edge-ab-testing';

const result = calculateSignificance(
{
  variantId: 'control',
  impressions: 1000,
  conversions: 100,
  conversionRate: 0.1,
},
{
  variantId: 'variant',
  impressions: 1000,
  conversions: 150,
  conversionRate: 0.15,
}
);

console.log(`Significant: ${result.isSignificant}`);
console.log(`Confidence: ${result.confidence}%`);
console.log(`P-value: ${result.pValue}`);

```

Edge Caching

Basic Caching

```

import { edgeCacheMiddleware } from '@philjs/api/edge-cache';

const cache = edgeCacheMiddleware({
  ttl: 300, // 5 minutes
  swr: 3600, // 1 hour stale-while-revalidate
  tags: ['api-v2'],
});

```

Cache Control Headers

```

import { cacheControlMiddleware } from '@philjs/api/edge-cache';

const cacheControl = cacheControlMiddleware({
  maxAge: 3600,
  staleWhileRevalidate: 86400,
  visibility: 'public',
  immutable: true,
});

```

ETags for Conditional Requests

```

import { etagMiddleware } from '@philjs/api/edge-cache';

const etag = etagMiddleware();
// Automatically returns 304 for matching If-None-Match

```

Cache Presets

```

import { staticAssetCache, apiCache, pageCache } from '@philjs/api/edge-cache';

// Static assets (1 year)
const staticCache = staticAssetCache();

// API responses (60s with 5min SWR)
const apiCaching = apiCache(60, 300);

// Pages (5min with 1hr SWR)
const pageCaching = pageCache(300, 3600);

```

Cache Purging

```

import { purgeCacheTags, purgeCacheKey } from '@philjs/api/edge-cache';

// Purge by tags
await purgeCacheTags(['api-v2', 'users']);

// Purge specific key
await purgeCacheKey('/api/users/123');

```

Vary Headers

```

import { varyMiddleware } from '@philjs/api/edge-cache';

const vary = varyMiddleware(['Accept-Language', 'Cookie']);
// Cache varies by language and authentication

```

Complete Example

Cloudflare Worker

```
import {
  executeEdgeMiddleware,
  composeEdgeMiddleware,
  securityHeadersMiddleware,
} from '@philjs/api/edge-middleware';
import { redirectByCountry, languageDetectionMiddleware } from '@philjs/api/geolocation';
import { abTestingMiddleware } from '@philjs/api/edge-ab-testing';
import { edgeCacheMiddleware, cacheControlMiddleware } from '@philjs/api/edge-cache';

const middleware = composeEdgeMiddleware(
  // Security
  securityHeadersMiddleware({
    csp: "default-src 'self'",  

    hsts: true,  

  }),
  // Geolocation
  redirectByCountry({  

    'GB,IE': '/uk',  

    'FR': '/fr',  

}),
languageDetectionMiddleware(),

// A/B Testing
abTestingMiddleware({
  experiments: [
    {
      id: 'pricing',
      name: 'Pricing Page Test',
      variants: [
        { id: 'control', name: 'Control', weight: 50 },
        { id: 'new', name: 'New Pricing', weight: 50 },
      ],
    },
  ],
}),
  // Caching
  edgeCacheMiddleware({
    ttl: 300,
    swr: 3600,
  }),
  cacheControlMiddleware({
    maxAge: 300,
    staleWhileRevalidate: 3600,
  })
);

export default {
  async fetch(request: Request, env: any, ctx: any) {
    return executeEdgeMiddleware(request, middleware, {
      platform: { env, ctx },
    });
  },
};
```

Vercel Edge Function

```

import { executeEdgeMiddleware } from '@philjs/api/edge-middleware';
import { geoRedirectMiddleware } from '@philjs/api/geolocation';
import { abTestingMiddleware } from '@philjs/api/edge-ab-testing';

export const config = {
  runtime: 'edge',
};

const middleware = [
  geoRedirectMiddleware([
    {
      countries: ['CN'],
      destination: '/cn',
    },
  ]),
  abTestingMiddleware({
    experiments: [
      {
        id: 'hero',
        name: 'Hero Section',
        variants: [
          { id: 'a', name: 'Variant A', weight: 50 },
          { id: 'b', name: 'Variant B', weight: 50 },
        ],
      },
    ],
  }),
];
];

export default async function handler(request: Request) {
  return executeEdgeMiddleware(request, middleware);
}

```

Advanced Features

Custom Cache Store

```

import { edgeCacheMiddleware, type CacheStore } from '@philjs/api/edge-cache';

class RedisCacheStore implements CacheStore {
  async get(key: string) {
    // Fetch from Redis
  }

  async put(key: string, response: Response, options?: {
    // Store in Redis
  })

  async delete(key: string) {
    // Delete from Redis
  }

  async purge(tags: string[]) {
    // Purge by tags
  }
}

const cache = edgeCacheMiddleware({
  store: new RedisCacheStore(),
  ttl: 300,
});

```

Custom Analytics Provider

```

import { createAnalyticsProvider, abTestingMiddleware } from '@philjs/api/edge-ab-testing';

const analytics = createAnalyticsProvider({
  endpoint: 'https://analytics.example.com/track',
  headers: {
    'Authorization': 'Bearer token',
  },
});

const abTesting = abTestingMiddleware({
  experiments: /* ... */,
  onAssignment: async (assignment, context) => {
    await analytics.trackExperiment(assignment, context);
  },
});

```

Deterministic Variant Selection

```
import { selectVariantDeterministic } from '@philjs/api/edge-ab-testing';

// Same user always gets same variant
const userId = getUserId(request);
const variant = selectVariantDeterministic(
  [
    { id: 'a', name: 'A', weight: 50 },
    { id: 'b', name: 'B', weight: 50 },
  ],
  userId
);
```

Runtime Compatibility

All edge middleware is compatible with:

- **Cloudflare Workers**: Full support including cf object
- **Vercel Edge**: Full support including geo headers
- **Deno Deploy**: Full support
- **Fasty Compute@Edge**: Full support
- **Node.js**: Works as standard middleware

Performance

- **Zero overhead**: Edge middleware runs before your application
- **No layout shift**: A/B testing variants assigned before rendering
- **Optimal caching**: Stale-while-revalidate ensures fresh content
- **Geo-optimized**: Route users to closest region automatically

Best Practices

1. **Chain middleware efficiently**: Order matters! Put security first, then geo/routing, then caching
2. **Use SWR liberally**: Stale-while-revalidate ensures users always get fast responses
3. **Tag your cache**: Use cache tags for efficient purging
4. **Monitor experiments**: Track significance before making decisions
5. **Test edge locally**: Use Miniflare or Wrangler for local development

License

MIT

Flash Messages & Enhanced Session Management

Complete guide to using Flash Messages and Enhanced Session Management in PhilJS.

Table of Contents

- [Flash Messages](#)
- [Enhanced Cookie Sessions](#)
- [Session Utilities](#)
- [Complete Examples](#)

Flash Messages

Remix-style flash messages with session-based one-time messages that are automatically cleared after being read.

Basic Usage

```

import { createCookieSessionStorage } from '@philjs/api/session';
import { setFlashSuccess, getFlashMessages } from '@philjs/api/flash';

const sessionStorage = createCookieSessionStorage({
  secret: process.env.SESSION_SECRET!,
});

// In your action/handler
export async function handleFormSubmit(request: Request) {
  const session = await sessionStorage.getSession(request);

  // Set a flash message
  setFlashSuccess(session, 'Your changes have been saved!');

  // Commit and redirect
  return new Response(null, {
    status: 302,
    headers: {
      'Location': '/dashboard',
      'Set-Cookie': await sessionStorage.commitSession(session),
    },
  });
}

// In your page Loader
export async function loadPage(request: Request) {
  const session = await sessionStorage.getSession(request);

  // Get flash messages (auto-cleared after read)
  const messages = getFlashMessages(session);

  return {
    messages,
    headers: {
      'Set-Cookie': await sessionStorage.commitSession(session),
    },
  };
}

```

Flash Categories

```

import {
  setFlashSuccess,
  setFlashError,
  setFlashWarning,
  setFlashInfo,
} from '@philjs/api/flash';

// Success message
setFlashSuccess(session, 'Operation completed successfully!');

// Error message
setFlashError(session, 'Something went wrong!');

// Warning message
setFlashWarning(session, 'Please verify your email address');

// Info message
setFlashInfo(session, 'You have 3 new notifications');

```

Flash with Metadata

```

import { setFlash } from '@philjs/api/flash';

setFlash(session, 'success', 'User created', {
  userId: 123,
  username: 'johndoe',
  action: 'create',
});

```

Flash Utilities

```

import { createFlashUtils } from '@philjs/api/flash';

const flash = createFlashUtils(session);

// Set messages
flash.success('Success!');
flash.error('Error!');
flash.warning('Warning!');
flash.info('Info!');

// Get messages (auto-cleared)
const messages = flash.get();

// Get by category (auto-cleared)
const errors = flash.getByCategory('error');

// Peek without clearing
const peeked = flash.peek();

// Check if has messages
if (flash.has()) {
  // Handle messages
}

// Clear all messages
flash.clear();

```

Advanced Usage

```

import {
  getFlashMessagesByCategory,
  peekFlashMessages,
  hasFlashMessages,
} from '@philjs/api/flash';

// Get messages by category
const errors = getFlashMessagesByCategory(session, 'error');

// Peek without clearing
const messages = peekFlashMessages(session);

// Check for messages
if (hasFlashMessages(session)) {
  // Display toast notifications
}

```

Flash Middleware

```

import { flashMiddleware } from '@philjs/api/flash';

const middleware = flashMiddleware(sessionStorage);

// Use in your request handler
const response = await middleware(request, async () => {
  // Your handler Logic
  return new Response('OK');
});

```

Enhanced Cookie Sessions

Secure cookie-based sessions with signing, encryption, rotation, and CSRF protection.

Basic Setup

```

import { createCookieSessionStorage } from '@philjs/api/cookie-session';

const sessionStorage = createCookieSessionStorage({
  name: 'my_session',
  secret: process.env.SESSION_SECRET!, // Min 32 chars
  path: '/',
  secure: true,
  httpOnly: true,
  sameSite: 'lax',
  maxAge: 60 * 60 * 24 * 7, // 7 days
});

```

With Encryption

```

const sessionStorage = createCookieSessionStorage({
  name: 'my_session',
  secret: process.env.SESSION_SECRET!, // For signing
  encryptionSecret: process.env.ENCRYPTION_SECRET!, // For encryption (min 32 chars)
  secure: true,
  httpOnly: true,
  sameSite: 'strict',
});

```

Session Rotation

```

const sessionStorage = createCookieSessionStorage({
  secret: process.env.SESSION_SECRET!,
  rotate: true,
  rotateInterval: 3600, // Rotate every hour
});

// Manual rotation
sessionStorage.rotateSession(session);

```

CSRF Protection

```

const sessionStorage = createCookieSessionStorage({
  secret: process.env.SESSION_SECRET!,
  csrf: true,
  csrfFieldName: 'csrf_token',
});

// Generate CSRF token
const csrfToken = sessionStorage.generateCSRF(session);

// Verify CSRF token
const isValid = await sessionStorage.verifyCSRF(request, token);

```

CSRF Middleware

```

import { csrfMiddleware } from '@philjs/api/cookie-session';

const middleware = csrfMiddleware(sessionStorage);

// Protects POST, PUT, PATCH, DELETE requests
const response = await middleware(request, async () => {
  return new Response('OK');
});

```

Session Rotation Middleware

```

import { sessionRotationMiddleware } from '@philjs/api/cookie-session';

const middleware = sessionRotationMiddleware(sessionStorage);

const response = await middleware(request, async () => {
  return new Response('OK');
});

```

Session Utilities

Helper functions and middleware for session management.

Basic Helpers

```

import {
  commitSession,
  destroySession,
  getOrCreateSession,
  requireSession,
} from '@philjs/api/session-utils';

// Commit session
const setCookie = await commitSession(storage, session);

// Destroy session
const setCookie = await destroySession(storage, session);

// Get or create session
const session = await getOrCreateSession(storage, request);

// Require session (throws if not found)
const session = await requireSession(storage, request, 'Login required');

```

Session Middleware

```

import { sessionMiddleware } from '@philjs/api/session-utils';

const middleware = sessionMiddleware({
  storage: sessionStorage,
  autoCommit: true, // Auto-commit on response
  contextKey: 'session',
});

const response = await middleware(request, async (req) => {
  // Session is attached to request
  const session = (req as RequestWithSession).session;

  session?.set('userId', '123');

  return new Response('OK');
});

```

Session Value Helpers

```

import {
  getSessionValue,
  setSessionValue,
  mergeSessionData,
  clearSessionData,
} from '@philjs/api/session-utils';

// Get with default
const userId = getSessionValue(session, 'userId', 'anonymous');

// Set with validation
const success = setSessionValue(
  session,
  'count',
  5,
  (value) => value > 0
);

// Merge data
mergeSessionData(session, {
  username: 'john',
  email: 'john@example.com',
});

// Clear all data
clearSessionData(session);

```

Typed Session Utilities

```

import { createTypedSessionUtils } from '@philjs/api/session-utils';

interface MySessionData {
  userId: string;
  username: string;
  role: 'admin' | 'user';
}

const utils = createTypedSessionUtils<MySessionData>(sessionStorage);

// All methods are fully typed
const session = await utils.get(request);
session.set('role', 'admin'); // Type-safe

const userId = utils.getValue(session, 'userId', 'anonymous');

utils.merge(session, {
  username: 'john',
  role: 'user',
});

```

Session Timeout Middleware

```

import { sessionTimeoutMiddleware } from '@philjs/api/session-utils';

const middleware = sessionTimeoutMiddleware(
  sessionStorage,
  3600 // 1 hour timeout
);

const response = await middleware(request, async () => {
  return new Response('OK');
});

```

Session Validator Middleware

```

import { sessionValidatorMiddleware } from '@philjs/api/session-utils';

const middleware = sessionValidatorMiddleware(
  sessionStorage,
  (session) => {
    // Validate session
    return session.get('userId') !== undefined;
  }
);

const response = await middleware(request, async () => {
  return new Response('OK');
});

```

Session Regeneration

```

import { regenerateSession } from '@philjs/api/session-utils';

// After Login or privilege escalation
const newSession = await regenerateSession(storage, oldSession);

// Destroy old session
await storage.destroySession(oldSession);

// Commit new session
const setCookie = await storage.commitSession(newSession);

```

Complete Examples

Login with Flash Messages

```

import { createCookieSessionStorage } from '@philjs/api/cookie-session';
import { setFlashSuccess, setFlashError } from '@philjs/api/flash';
import { regenerateSession } from '@philjs/api/session-utils';

const sessionStorage = createCookieSessionStorage({
  secret: process.env.SESSION_SECRET!,
  encryptionSecret: process.env.ENCRYPTION_SECRET!,
  csrf: true,
  rotate: true,
});

export async function handleLogin(request: Request) {
  const formData = await request.formData();
  const email = formData.get('email');
  const password = formData.get('password');

  const session = await sessionStorage.getSession(request);

  // Verify credentials
  const user = await verifyCredentials(email, password);

  if (!user) {
    setFlashError(session, 'Invalid credentials');

    return new Response(null, {
      status: 302,
      headers: {
        'Location': '/login',
        'Set-Cookie': await sessionStorage.commitSession(session),
      },
    });
  }

  // Regenerate session after login
  const newSession = await regenerateSession(sessionStorage, session);

  // Set user data
  newSession.set('userId', user.id);
  newSession.set('role', user.role);

  // Set success message
  setFlashSuccess(newSession, `Welcome back, ${user.name}!`);

  return new Response(null, {
    status: 302,
    headers: {
      'Location': '/dashboard',
      'Set-Cookie': await sessionStorage.commitSession(newSession),
    },
  });
}

```

Protected Route with Session Validation

```

import { sessionValidatorMiddleware, requireSession } from '@philjs/api/session-utils';
import { setFlashWarning } from '@philjs/api/flash';

const authMiddleware = sessionValidatorMiddleware(
  sessionStorage,
  async (session) => {
    // Check if user is authenticated
    const userId = session.get('userId');

    if (!userId) {
      setFlashWarning(session, 'Please log in to continue');
      return false;
    }

    // Additional validation
    const user = await getUserById(userId);
    return user !== null && user.active;
  }
);

export async function handleProtectedRoute(request: Request) {
  return authMiddleware(request, async () => {
    const session = await requireSession(sessionStorage, request);
    const userId = session.get('userId');

    // Handle protected route
    return new Response(`Hello, user ${userId}`);
  });
}

```

Form with CSRF Protection

```

import { csrfMiddleware } from '@philjs/api/cookie-session';
import { setFlashSuccess } from '@philjs/api/flash';

const csrf = csrfMiddleware(sessionStorage);

export async function handleForm(request: Request) {
  return csrf(request, async () => {
    const session = await sessionStorage.getSession(request);
    const formData = await request.formData();

    // Process form
    await saveFormData(formData);

    setFlashSuccess(session, 'Form submitted successfully!');

    return new Response(null, {
      status: 302,
      headers: {
        'Location': '/success',
        'Set-Cookie': await sessionStorage.commitSession(session),
      },
    });
  });
}

// In your HTML template
export async function renderForm(request: Request) {
  const session = await sessionStorage.getSession(request);
  const csrfToken = sessionStorage.generateCSRF(session);

  return `
    <form method="POST">
      <input type="hidden" name="csrf_token" value="${csrfToken}" />
      <!-- form fields -->
      <button type="submit">Submit</button>
    </form>
  `;
}

```

Complete Middleware Stack

```

import { composeMiddleware } from '@philjs/api/middleware';
import { sessionMiddleware, sessionTimeoutMiddleware } from '@philjs/api/session-utils';
import { csrfMiddleware, sessionRotationMiddleware } from '@philjs/api/cookie-session';
import { flashMiddleware } from '@philjs/api/flash';

const middleware = composeMiddleware([
  sessionMiddleware({ storage: sessionStorage }),
  sessionTimeoutMiddleware(sessionStorage, 3600),
  sessionRotationMiddleware(sessionStorage),
  csrfMiddleware(sessionStorage),
  flashMiddleware(sessionStorage),
]);

export async function handleRequest(request: Request) {
  return middleware(request, async (req) => {
    // Your app logic
    return new Response('OK');
  });
}

```

Type Safety

All utilities are fully typed and work with TypeScript:

```

import type { TypedSession } from '@philjs/api/session-utils';
import type { FlashSessionData } from '@philjs/api/flash';

interface MySessionData extends FlashSessionData {
  userId: string;
  username: string;
  role: 'admin' | 'user';
  preferences: {
    theme: 'light' | 'dark';
    language: string;
  };
}

const session: TypedSession<MySessionData> = await sessionStorage.getSession(request);

// Fully typed
session.set('role', 'admin'); // ✓
session.set('role', 'invalid'); // ✗ Type error

```

Security Best Practices

1. **Use strong secrets:** At least 32 characters, randomly generated
2. **Enable encryption:** For sensitive session data
3. **Enable CSRF protection:** For state-changing operations
4. **Use session rotation:** For long-lived sessions
5. **Set secure cookie options:** secure, httpOnly, sameSite
6. **Implement session timeout:** For inactive sessions
7. **Regenerate after login:** Prevent session fixation
8. **Validate session data:** On each request

```

const sessionStorage = createCookieSessionStorage({
  secret: process.env.SESSION_SECRET!, // Strong secret
  encryptionSecret: process.env.ENCRYPTION_SECRET!, // Enable encryption
  csrf: true, // Enable CSRF
  rotate: true, // Enable rotation
  rotateInterval: 3600, // Rotate hourly
  secure: true, // HTTPS only
  httpOnly: true, // No JS access
  sameSite: 'strict', // CSRF protection
  maxAge: 60 * 60 * 24 * 7, // 7 days
});

```

Migration Guide: Enhanced Sessions

Guide for migrating from basic session management to enhanced cookie sessions with flash messages.

Overview

PhilJS 0.1.0 introduces enhanced session management features:

- **Flash Messages:** One-time messages with auto-cleanup
- **Enhanced Cookie Sessions:** Encryption, signing, CSRF protection, rotation
- **Session Utilities:** Timeout, validation, and helper functions

Migration Steps

Step 1: Update Imports

Before:

```
import { createCookieSessionStorage } from '@philjs/api/session';
```

After:

```
// For basic sessions (backward compatible)
import { createCookieSessionStorage } from '@philjs/api/session';

// For enhanced sessions (new)
import { createEnhancedCookieSessionStorage } from '@philjs/api/cookie-session';

// For flash messages
import { setFlashSuccess, getFlashMessages } from '@philjs/api/flash';

// For utilities
import { sessionMiddleware, requireSession } from '@philjs/api/session-utils';
```

Step 2: Update Session Configuration

Before:

```
const sessionStorage = createCookieSessionStorage({
  cookieName: 'session',
  secret: process.env.SESSION_SECRET,
  cookie: {
    path: '/',
    secure: true,
    httpOnly: true,
    sameSite: 'lax',
    maxAge: 60 * 60 * 24 * 7,
  },
});
```

After (Enhanced):

```
const sessionStorage = createEnhancedCookieSessionStorage({
  name: 'session', // Renamed from cookieName
  secret: process.env.SESSION_SECRET,
  encryptionSecret: process.env.ENCRYPTION_SECRET, // NEW: Enable encryption
  csrf: true, // NEW: Enable CSRF protection
  rotate: true, // NEW: Enable session rotation
  rotateInterval: 3600, // NEW: Rotate every hour
  path: '/',
  secure: true,
  httpOnly: true,
  sameSite: 'lax',
  maxAge: 60 * 60 * 24 * 7,
});
```

Step 3: Add Flash Messages

Before:

```
// Storing temporary messages in session
session.set('message', 'Success!');
session.set('messageType', 'success');

// In Loader
const message = session.get('message');
const messageType = session.get('messageType');
session.delete('message');
session.delete('messageType');
```

After:

```
import { setFlashSuccess, getFlashMessages } from '@philjs/api/flash';

// In action
setFlashSuccess(session, 'Success!');

// In Loader
const messages = getFlashMessages(session); // Auto-cleared
// messages = [{ category: 'success', message: 'Success!', ... }]
```

Step 4: Add CSRF Protection

Before:

```
// Manual CSRF handling
const csrfToken = generateRandomToken();
session.set('csrfToken', csrfToken);

// In form handler
const submittedToken = formData.get('csrf_token');
if (submittedToken !== session.get('csrfToken')) {
  throw new Error('Invalid CSRF token');
}
```

After:

```
import { csrfMiddleware } from '@philjs/api/cookie-session';

// Enable in session config
const sessionStorage = createEnhancedCookieSessionStorage({
  secret: process.env.SESSION_SECRET,
  csrf: true,
});

// Generate token for form
const csrfToken = sessionStorage.generateCSRF(session);

// Protect route with middleware
const csrf = csrfMiddleware(sessionStorage);

export async function handleForm(request: Request) {
  return csrf(request, async () => {
    // CSRF verified automatically
    const formData = await request.formData();
    // Process form...
  });
}
```

Step 5: Add Session Timeout

Before:

```
// Manual timeout tracking
const lastActivity = session.get('lastActivity') as number;
if (Date.now() - lastActivity > TIMEOUT) {
  await destroySession(sessionStorage, session);
  return redirect('/login');
}
session.set('lastActivity', Date.now());
```

After:

```
import { sessionTimeoutMiddleware } from '@philjs/api/session-utils';

const timeoutMw = sessionTimeoutMiddleware(
  sessionStorage,
  3600 // 1 hour
);

export async function handleRequest(request: Request) {
  return timeoutMw(request, async () => {
    // Timeout handled automatically
    return new Response('OK');
  });
}
```

Step 6: Add Session Validation

Before:

```
// Manual validation
const userId = session.get('userId');
if (!userId) {
  return redirect('/login');
}

const user = await db.user.findUnique({ where: { id: userId } });
if (!user || !user.active) {
  await destroySession(sessionStorage, session);
  return redirect('/login');
}
```

After:

```

import { sessionValidatorMiddleware } from '@philjs/api/session-utils';

const validator = sessionValidatorMiddleware(
  sessionStorage,
  async (session) => {
    const userId = session.get('userId');
    if (!userId) return false;

    const user = await db.user.findUnique({ where: { id: userId } });
    return user !== null && user.active;
  }
);

export async function handleProtectedRoute(request: Request) {
  return validator(request, async () => {
    // Session validated automatically
    return new Response('Protected content');
  });
}

```

Step 7: Update Login Flow

Before:

```

export async function handleLogin(request: Request) {
  const { email, password } = await request.json();
  const user = await authenticate(email, password);

  if (!user) {
    return json({ error: 'Invalid credentials' }, { status: 401 });
  }

  const session = await sessionStorage.getSession(request);
  session.set('userId', user.id);

  return json(
    { success: true },
    {
      headers: {
        'Set-Cookie': await sessionStorage.commitSession(session),
      },
    }
  );
}

```

After:

```

import { regenerateSession } from '@philjs/api/session-utils';
import { setFlashSuccess, setFlashError } from '@philjs/api/flash';

export async function handleLogin(request: Request) {
  const { email, password } = await request.json();
  const session = await sessionStorage.getSession(request);

  const user = await authenticate(email, password);

  if (!user) {
    setFlashError(session, 'Invalid credentials');
    return redirect('/login', {
      headers: {
        'Set-Cookie': await sessionStorage.commitSession(session),
      },
    });
  }

  // Regenerate to prevent session fixation
  const newSession = await regenerateSession(sessionStorage, session);
  newSession.set('userId', user.id);

  setFlashSuccess(newSession, `Welcome back, ${user.name}!`);

  return redirect('/dashboard', {
    headers: {
      'Set-Cookie': await sessionStorage.commitSession(newSession),
    },
  });
}

```

Backward Compatibility

The original `createCookieSessionStorage` from `@philjs/api/session` remains unchanged and fully compatible. You can migrate incrementally:

1. Keep using basic sessions for existing code
2. Use enhanced sessions for new features
3. Gradually migrate existing code as needed

Type Safety

Enhanced sessions support full TypeScript typing:

```
interface MySessionData {
  userId: string;
  role: 'admin' | 'user';
  preferences: {
    theme: 'light' | 'dark';
  };
}

const sessionStorage = createEnhancedCookieSessionStorage<MySessionData>({
  secret: process.env.SESSION_SECRET,
});

const session = await sessionStorage.getSession(request);
session.set('role', 'admin'); // ✓ Type-safe
session.set('role', 'invalid'); // ✗ Type error
```

Performance Considerations

Encryption

Encryption adds minimal overhead (~1-2ms per request). Use it for sensitive data:

```
const sessionStorage = createEnhancedCookieSessionStorage({
  secret: process.env.SESSION_SECRET,
  encryptionSecret: process.env.ENCRYPTION_SECRET, // Enable encryption
});
```

Session Rotation

Rotation frequency affects performance. Recommended intervals:

- **High security:** 900s (15 minutes)
- **Normal security:** 3600s (1 hour)
- **Long-lived sessions:** 86400s (24 hours)

```
const sessionStorage = createEnhancedCookieSessionStorage({
  secret: process.env.SESSION_SECRET,
  rotate: true,
  rotateInterval: 3600, // 1 hour
});
```

Security Checklist

- Use strong secrets (min 32 characters, randomly generated)
- Enable encryption for sensitive data
- Enable CSRF protection for state-changing operations
- Set appropriate cookie options (`secure`, `httpOnly`, `sameSite`)
- Implement session timeout for inactive sessions
- Regenerate session after login/privilege changes
- Validate session data on each request
- Use session rotation for long-lived sessions

Common Patterns

Pattern 1: Protected Route

```
import { sessionValidatorMiddleware, requireSession } from '@philjs/api/session-utils';

const auth = sessionValidatorMiddleware(
  sessionStorage,
  (session) => session.get('userId') !== undefined
);

export async function handleRoute(request: Request) {
  return auth(request, async () => {
    const session = await requireSession(sessionStorage, request);
    const userId = session.get('userId');
    // Handle authenticated request
  });
}
```

Pattern 2: Form with Flash

```

import { setFlashSuccess, setFlashError, getFlashMessages } from '@philjs/api/flash';

export async function handleForm(request: Request) {
  const session = await sessionStorage.getSession(request);
  const formData = await request.formData();

  try {
    await processForm(formData);
    setFlashSuccess(session, 'Form submitted successfully!');
  } catch (error) {
    setFlashError(session, 'Failed to submit form');
  }

  return redirect('/form', {
    headers: {
      'Set-Cookie': await sessionStorage.commitSession(session),
    },
  });
}

export async function showForm(request: Request) {
  const session = await sessionStorage.getSession(request);
  const messages = getFlashMessages(session);

  return render({ messages });
}

```

Pattern 3: Middleware Stack

```

import { composeMiddleware } from '@philjs/api/middleware';
import { sessionMiddleware, sessionTimeoutMiddleware } from '@philjs/api/session-utils';
import { csrfMiddleware, sessionRotationMiddleware } from '@philjs/api/cookie-session';

const middleware = composeMiddleware([
  sessionMiddleware({ storage: sessionStorage }),
  sessionTimeoutMiddleware(sessionStorage, 3600),
  sessionRotationMiddleware(sessionStorage),
  csrfMiddleware(sessionStorage),
]);

export async function handleRequest(request: Request) {
  return middleware(request, async () => {
    // Your app Logic
  });
}

```

Troubleshooting

Issue: Session not persisting

Cause: Cookie not being set correctly

Solution: Ensure you're committing the session and setting the cookie header

```

const setCookie = await sessionStorage.commitSession(session);

return new Response('OK', {
  headers: {
    'Set-Cookie': setCookie,
  },
});

```

Issue: CSRF token invalid

Cause: Token mismatch or missing

Solution: Ensure token is generated and submitted correctly

```

// Generate in form
const csrfToken = sessionStorage.generateCSRF(session);

// Submit in header or form field
headers.set('X-CSRF-Token', csrfToken);
// OR
formData.append('csrf_token', csrfToken);

```

Issue: Session timeout too aggressive

Cause: Timeout interval too short

Solution: Adjust timeout interval

```

const timeoutMw = sessionTimeoutMiddleware(
  sessionStorage,
  7200 // 2 hours instead of 1 hour
);

```

Need Help?

- [Flash Messages Documentation](#)
- [Enhanced Sessions Documentation](#)
- [Session Utilities Documentation](#)
- [Examples](#)

PhilJS Authentication Guide

Complete guide to authentication in PhilJS applications.

Table of Contents

- [Quick Start](#)
- [Generators](#)
- [Providers](#)
- [Hooks](#)
- [Protected Routes](#)
- [Session Management](#)
- [Advanced Usage](#)

Quick Start

1. Generate Authentication Setup

Use the PhilJS CLI to generate authentication setup for your preferred provider:

```
# Generate Clerk authentication
philjs generate auth clerk

# Generate Auth0 authentication
philjs generate auth auth0

# Generate Supabase authentication
philjs generate auth supabase

# Generate NextAuth authentication
philjs generate auth nextauth

# Generate custom authentication
philjs generate auth custom
```

This will create: - Authentication configuration - Provider integration - Login/Signup forms - Password reset flow - Profile management - Protected route utilities - Example pages

2. Install Dependencies

```
# For Clerk
npm install @clerk/clerk-react

# For Auth0
npm install @auth0/auth0-react

# For Supabase
npm install @supabase/supabase-js

# For NextAuth
npm install next-auth

# For Custom
npm install @philjs/auth jsonwebtoken
```

3. Configure Environment Variables

Copy `.env.example` to `.env` and fill in your credentials:

```

# Clerk
VITE_CLERK_PUBLISHABLE_KEY=pk_test_...
CLERK_SECRET_KEY=sk_test_...

# Auth0
VITE_AUTH0_DOMAIN=your-domain.auth0.com
VITE_AUTH0_CLIENT_ID=your-client-id
AUTH0_CLIENT_SECRET=your-client-secret

# Supabase
VITE_SUPABASE_URL=https://your-project.supabase.co
VITE_SUPABASE_ANON_KEY=your-anon-key

# NextAuth
NEXTAUTH_SECRET=your-secret-key

# Custom
VITE_API_URL=http://localhost:3000/api
JWT_SECRET=your-jwt-secret

```

4. Wrap Your App

```

import { AuthProvider } from './auth/AuthProvider';

export function App() {
  return (
    <AuthProvider>
      <YourApp />
    </AuthProvider>
  );
}

```

Generators

Auth Generator Options

```

philjs generate auth <provider> [options]

Options:
  -d, --directory <dir>          Target directory (default: "src")
  -no-ui                  Skip UI components generation
  -no-middleware           Skip middleware generation
  -no-protected-routes     Skip protected route utilities

```

Generated Files

```

src/
└── auth/
    ├── config.ts          # Provider configuration
    ├── AuthProvider.tsx   # Provider wrapper component
    ├── hooks.ts           # useAuth, useUser hooks
    ├── protected.tsx      # Protected route utilities
    ├── AuthGuard.tsx      # Conditional rendering component
    └── components/
        ├── LoginForm.tsx   # Login form component
        ├── SignupForm.tsx   # Signup form component
        ├── PasswordReset.tsx # Password reset component
        └── ProfileForm.tsx   # Profile management component
    └── pages/
        └── auth/
            ├── sign-in.tsx    # Sign in page
            ├── sign-up.tsx    # Sign up page
            ├── forgot-password.tsx # Password reset page
            └── profile.tsx      # Profile page
    └── middleware.ts       # Auth middleware

```

Providers

Provider Abstraction

All auth providers implement a common interface:

```

interface AuthProvider {
  readonly name: string;
  readonly user: Signal<User | null>;
  readonly session: Signal<AuthSession | null>;
  readonly loading: Signal<boolean>;

  initialize(): Promise<void>;
  signInWithEmail(email: string, password: string): Promise<User>;
  signUpWithEmail(email: string, password: string, metadata?: Record<string, unknown>): Promise<User>;
  signInWithAuth?(provider: string): Promise<void>;
  signOut(): Promise<void>;
  getToken(): Promise<string | null>;
  refreshToken?(): Promise<string>;
  updateUser?(updates: Partial<User>): Promise<User>;
  sendPasswordReset?(email: string): Promise<void>;
  resetPassword?(token: string, newPassword: string): Promise<void>;
}

```

Custom Provider

Create a custom authentication provider:

```

import { BaseAuthProvider, setAuthProvider } from '@philjs/auth';
import { signal } from '@philjs/core/signals';

class MyAuthProvider extends BaseAuthProvider {
  readonly name = 'my-auth';
  readonly user = signal<User | null>(null);
  readonly session = signal<AuthSession | null>(null);
  readonly loading = signal(false);

  async initialize(): Promise<void> {
    // Initialize your auth system
  }

  async signInWithEmail(email: string, password: string): Promise<User> {
    // Implement sign in logic
  }

  async signUpWithEmail(email: string, password: string): Promise<User> {
    // Implement sign up logic
  }

  async signOut(): Promise<void> {
    // Implement sign out logic
  }

  async getToken(): Promise<string | null> {
    // Return access token
  }
}

// Initialize and set as global provider
const authProvider = new MyAuthProvider(config);
await authProvider.initialize();
setAuthProvider(authProvider);

```

Hooks

useAuth()

Access authentication state and methods:

```

import { useAuth } from '@philjs/auth/hooks';

function MyComponent() {
  const {
    user, // Current user
    session, // Current session
    isAuthenticated, // Authentication status
    isloading, // Loading state
    signIn, // Sign in method
    signUp, // Sign up method
    signOut, // Sign out method
    getToken, // Get access token
    refreshToken, // Refresh token
    updateUser, // Update user profile
  } = useAuth();

  return (
    <div>
      {isAuthenticated ? (
        <p>Welcome, {user?.name}!</p>
      ) : (
        <button onClick={() => signIn(email, password)}>
          Sign In
        </button>
      )}
    </div>
  );
}

```

useUser()

Get the current authenticated user:

```

import { useUser } from '@philjs/auth/hooks';

function Profile() {
  const user = useUser();

  if (!user) {
    return <div>Not logged in</div>;
  }

  return (
    <div>
      <img src={user.avatar} alt={user.name} />
      <h1>{user.name}</h1>
      <p>{user.email}</p>
    </div>
  );
}

```

useHasPermission()

Check user permissions or roles:

```

import { useHasPermission } from '@philjs/auth/hooks';

function AdminPanel() {
  const isAdmin = useHasPermission('admin');

  if (!isAdmin) {
    return <div>Access denied</div>;
  }

  return <div>Admin panel content</div>;
}

```

useRequireAuth()

Redirect if not authenticated:

```

import { useRequireAuth } from '@philjs/auth/hooks';

function ProtectedPage() {
  useRequireAuth('/login'); // Redirects to /Login if not authenticated

  return <div>Protected content</div>;
}

```

Protected Routes

ProtectedRoute Component

Wrap components that require authentication:

```

import { ProtectedRoute } from '@philjs/auth/protected-routes';

function Dashboard() {
  return (
    <ProtectedRoute>
      <div>Dashboard content</div>
    </ProtectedRoute>
  );
}

```

With custom redirect and fallback:

```

<ProtectedRoute
  redirectTo="/login"
  fallback={<div>Loading...</div>}
>
  <div>Protected content</div>
</ProtectedRoute>

```

withAuth HOC

Higher-order component for protecting components:

```

import { withAuth } from '@philjs/auth/protected-routes';

function DashboardComponent() {
  return <div>Dashboard content</div>;
}

export const Dashboard = withAuth(DashboardComponent);

```

Role-Based Protection

Protect routes based on user roles:

```

import { withRole } from '@philjs/auth/protected-routes';

function AdminPanelComponent() {
  return <div>Admin panel</div>;
}

export const AdminPanel = withRole(AdminPanelComponent, {
  role: 'admin',
  redirectTo: '/unauthorized',
});

```

Multiple roles (OR logic):

```

import { withAnyRole } from '@philjs/auth/protected-routes';

export const ModeratorPanel = withAnyRole(ModeratorPanelComponent, {
  roles: ['admin', 'moderator'],
  redirectTo: '/unauthorized',
});

```

AuthGuard Component

Conditionally render content based on auth state:

```

import { AuthGuard, ShowForAuth, ShowForGuest, ShowForRole } from '@philjs/auth/protected-routes';

function Navigation() {
  return (
    <nav>
      <ShowForAuth>
        <button onClick={logout}>Logout</button>
      </ShowForAuth>

      <ShowForGuest>
        <a href="/login">Login</a>
      </ShowForGuest>

      <ShowForRole role="admin">
        <a href="/admin">Admin Panel</a>
      </ShowForRole>
    </nav>
  );
}

```

Session Management

Automatic Token Refresh

Start automatic token refresh:

```
import { startSessionRefresh } from '@philjs/auth/session-refresh';

startSessionRefresh({
  refreshBeforeExpiry: 5 * 60 * 1000, // Refresh 5 minutes before expiry
  checkInterval: 60 * 1000,           // Check every minute
  refreshOnFocus: true,             // Refresh when window gains focus
  refreshOnReconnect: true,         // Refresh on network reconnect
  onRefreshFailed: (error) => {
    console.error('Token refresh failed:', error);
    // Optionally redirect to Login
  },
  onRefreshSuccess: (token) => {
    console.log('Token refreshed successfully');
  },
});
```

Session Persistence

Save and load sessions:

```
import { SessionPersistence } from '@philjs/auth/session-refresh';

// Save session
SessionPersistence.save(session, 'local'); // or 'session'

// Load session
const session = SessionPersistence.load('local');

// Clear session
SessionPersistence.clear('local');

// Clear all sessions
SessionPersistence.clearAll();
```

Logout Everywhere

Revoke all sessions across all devices:

```
import { logoutEverywhere } from '@philjs/auth/session-refresh';

async function handleLogoutEverywhere() {
  try {
    await logoutEverywhere();
    // User is Logged out everywhere
  } catch (error) {
    console.error('Failed to logout everywhere:', error);
  }
}
```

Advanced Usage

Custom Auth Flow

Implement a custom authentication flow:

```

import { useAuth } from '@philjs/auth/hooks';

function LoginForm() {
  const { signIn } = useAuth();
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState<string | null>(null);

  const handleSubmit = async (e: Event) => {
    e.preventDefault();
    setError(null);

    try {
      await signIn(email, password);
      // Redirect to dashboard
      window.location.href = '/dashboard';
    } catch (err) {
      setError(err instanceof Error ? err.message : 'Login failed');
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      {error && <div className="error">{error}</div>

      <input
        type="email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
        placeholder="Email"
        required
      />

      <input
        type="password"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
        placeholder="Password"
        required
      />

      <button type="submit">Sign In</button>
    </form>
  );
}

```

OAuth Sign In

Sign in with OAuth providers:

```

import { useAuth } from '@philjs/auth/hooks';

function SocialLogin() {
  const { signInWithOAuth } = useAuth();

  return (
    <div>
      <button onClick={() => signInWithOAuth('google')}>
        Sign in with Google
      </button>

      <button onClick={() => signInWithOAuth('github')}>
        Sign in with GitHub
      </button>

      <button onClick={() => signInWithOAuth('facebook')}>
        Sign in with Facebook
      </button>
    </div>
  );
}

```

Token Management

Get and use access tokens:

```

import { useAuth } from '@philjs/auth/hooks';

async function fetchProtectedData() {
  const { getToken } = useAuth();

  const token = await getToken();

  const response = await fetch('/api/protected', {
    headers: {
      'Authorization': `Bearer ${token}`,
    },
  });

  return response.json();
}

```

Multi-Factor Authentication

Implement MFA flow:

```

import { useAuth } from '@philjs/auth/hooks';

function MFASetup() {
  const { user } = useAuth();

  const enableMFA = async () => {
    // Call your MFA setup endpoint
    const response = await fetch('/api/auth/mfa/setup', {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${await getToken()}`,
      },
    });

    const { qrCode, secret } = await response.json();

    // Display QR code for user to scan
  };

  return (
    <div>
      <h2>Two-Factor Authentication</h2>
      {user?.metadata?.mfaEnabled ? (
        <p>MFA is enabled</p>
      ) : (
        <button onClick={enableMFA}>Enable MFA</button>
      )}
    </div>
  );
}

```

Best Practices

1. **Always use HTTPS in production** - Never send credentials over HTTP
2. **Validate tokens on the server** - Never trust client-side validation alone
3. **Use secure token storage** - Prefer httpOnly cookies for sensitive tokens
4. **Implement rate limiting** - Prevent brute force attacks
5. **Use strong password policies** - Enforce minimum length and complexity
6. **Enable MFA for sensitive operations** - Add extra security layer
7. **Refresh tokens before expiry** - Use automatic token refresh
8. **Handle errors gracefully** - Show user-friendly error messages
9. **Log security events** - Monitor authentication activity
10. **Keep dependencies updated** - Regularly update auth libraries

Troubleshooting

"Auth provider not initialized" error

Make sure you've wrapped your app with `AuthProvider` and called `setAuthProvider()`:

```

import { AuthProvider } from './auth/AuthProvider';

export function App() {
  return (
    <AuthProvider>
      <YourApp />
    </AuthProvider>
  );
}

```

Tokens not refreshing

Ensure your provider supports token refresh and you've started the refresh manager:

```
import { startSessionRefresh } from '@philjs/auth/session-refresh';

startSessionRefresh({
  refreshBeforeExpiry: 5 * 60 * 1000,
});
```

Redirect loops

Check your redirect logic and ensure protected routes don't redirect to themselves:

```
// Bad - creates infinite loop
<ProtectedRoute redirectTo="/dashboard">
  <Dashboard />
</ProtectedRoute>

// Good - redirects to login
<ProtectedRoute redirectTo="/login">
  <Dashboard />
</ProtectedRoute>
```

Examples

- Authentication and Authorization
- Security Authentication
- API Error Handling
- Routing Middleware

API Reference

See [Authentication and Authorization](#) for the current API reference.

PhilJS Database Migrations

A comprehensive, type-safe database migration system for PhilJS applications with support for multiple databases and ORMs.

Features

- **Multi-Database Support:** PostgreSQL, MySQL, SQLite, MongoDB
- **Migration Versioning:** Track and manage migration history
- **Up/Down Migrations:** Full rollback support
- **Transaction Support:** Automatic transaction wrapping
- **Schema Diff:** Automatic schema comparison
- **Auto-Migration:** Generate migrations from model changes
- **CLI Tools:** Full-featured command-line interface
- **ORM Integration:** Prisma and Drizzle support
- **Backup & Restore:** Automatic database backups
- **Conflict Detection:** Detect migration conflicts
- **Dry-Run Mode:** Preview changes before applying

Installation

```
npm install @philjs/db
```

Quick Start

1. Configuration

Create `@philjs/db.config.ts` in your project root:

```
export default {
  type: 'postgres',
  connection: process.env.DATABASE_URL,
  migrationsDir: './migrations',
  tableName: 'migrations',
  transactional: true,
  backup: true,
};
```

2. Initialize

```
npx philjs db migrate:create --name create_users_table --template table
```

3. Write Migration

Edit the generated migration file:

```

import type { Migration } from '@philjs/db/migrations';

export default {
  name: 'create_users_table',

  async up(context) {
    context.schema.createTable('users', (table) => {
      table.increments('id').primary();
      table.string('email').unique().notNullable();
      table.string('name').notNullable();
      table.timestamps(true);
    });
  },

  async down(context) {
    context.schema.dropTable('users');
  },
} as Migration;

```

4. Run Migrations

```
npx philjs db migrate
```

CLI Commands

Run Migrations

```

# Run all pending migrations
npx philjs db migrate

# Run specific number of migrations
npx philjs db migrate --step 1

# Run migrations up to specific version
npx philjs db migrate --to 20240101000000

# Dry run (preview without executing)
npx philjs db migrate --dry-run

```

Create Migration

```

# Create blank migration
npx philjs db migrate:create --name my_migration

# Create table migration
npx philjs db migrate:create --name create_posts --template table

# Create alter migration
npx philjs db migrate:create --name add_column --template alter

# Create data migration
npx philjs db migrate:create --name seed_data --template data

```

Rollback Migrations

```

# Rollback last batch
npx philjs db migrate:rollback

# Rollback specific number of migrations
npx philjs db migrate:rollback --step 1

# Rollback to specific version
npx philjs db migrate:rollback --to 20240101000000

# Rollback specific batch
npx philjs db migrate:rollback --batch 1

```

Migration Status

```

# Check migration status
npx philjs db migrate:status

# Verbose status
npx philjs db migrate:status --verbose

```

Reset & Fresh

```

# Reset (rollback all and re-run)
npx philjs db migrate:reset

# Fresh (drop all tables and re-run)
npx philjs db migrate:fresh --yes

```

Schema Diff

```
# Generate schema diff
npx philjs db migrate:diff

# Auto-generate migration from changes
npx philjs db migrate:auto --name auto_migration
```

Migration API

Schema Builder

Table Operations

```
// Create table
context.schema.createTable('users', (table) => {
  // columns here
});

// Drop table
context.schema.dropTable('users');

// Alter table
context.schema.alterTable('users', (table) => {
  // changes here
});

// Rename table
context.schema.renameTable('users', 'accounts');

// Check if table exists
const exists = await context.schema.hasTable('users');

// Raw SQL
context.schema.raw('CREATE INDEX...');
```

Column Types

```
table.increments('id')           // Auto-incrementing ID
table.integer('count')           // Integer
table.bigInteger('large_num')    // Big integer
table.string('name', 255)        // VARCHAR
table.text('description')       // TEXT
table.boolean('is_active')       // Boolean
table.date('birth_date')         // Date
table.datetime('created_at')     //Datetime
table.timestamp('updated_at')    // Timestamp
table.json('data')               // JSON
table.jsonb('metadata')          // JSONB (PostgreSQL)
table.uuid('id')                // UUID
table.decimal('price', 10, 2)     // Decimal
table.float('rating', 3, 2)       // Float
table.enum('status', ['active', 'inactive']) // Enum
```

Column Modifiers

```
column.primary()                 // Primary key
column.nullable()                // Allow NULL
column.notNulliable()            // NOT NULL
column.unique()                  // UNIQUE constraint
column.unsigned()                // Unsigned (MySQL)
column.defaultTo(value)          // Default value
column.index()                   // Create index
column.comment('...')            // Column comment
```

Constraints

```
// Primary key
table.primary('id');
table.primary(['user_id', 'post_id']); // Composite

// Unique
table.unique('email');
table.unique(['first_name', 'last_name']);

// Index
table.index('email');
table.index(['status', 'created_at']);

// Foreign key
table.foreign('user_id')
  .references('id')
  .inTable('users')
  .onDelete('CASCADE')
  .onUpdate('CASCADE');
```

Timestamps

```
// Add created_at and updated_at
table.timestamps(true);
```

Data Helpers

```
// Insert single row
await context.data.insert('users', {
  name: 'John',
  email: 'john@example.com',
});

// Insert multiple rows
await context.data.insert('users', [
  { name: 'John', email: 'john@example.com' },
  { name: 'Jane', email: 'jane@example.com' },
]);

// Update data
await context.data.update(
  'users',
  { id: 1 },
  { name: 'Updated Name' }
);

// Delete data
await context.data.delete('users', { id: 1 });

// Batch insert
await context.data.batchInsert('users', largeArray, 100);

// Raw query
const result = await context.data.raw('SELECT * FROM users');
```

SQL Execution

```
// Execute raw SQL
await context.sql('CREATE INDEX...');

// With parameters
await context.sql(
  'SELECT * FROM users WHERE email = $1',
  ['john@example.com']
);
```

Configuration Options

```
interface MigrationConfig {
  // Database type
  type: 'postgres' | 'mysql' | 'sqlite' | 'mongodb';

  // Connection string
  connection: string | object;

  // Migrations directory (default: ./migrations)
  migrationsDir?: string;

  // Migration table name (default: migrations)
  tableName?: string;

  // Enable transactions (default: true)
  transactional?: boolean;

  // Backup before migrate (default: false)
  backup?: boolean;

  // Seeds directory (default: ./seeds)
  seedsDir?: string;

  // Schema file path
  schemaFile?: string;
}
```

Database-Specific Features

PostgreSQL

```
// JSONB
table.jsonb('metadata');

// Array
table.raw('tags TEXT[]');

// Full-text search
context.sql(`
    ALTER TABLE posts
    ADD COLUMN search_vector tsvector
`);

// Triggers and functions
context.sql(`
    CREATE FUNCTION update_timestamp()...
`);


```

MySQL

```
// Engine and charset
// Automatically added to CREATE TABLE statements

// Unsigned integers
table.integer('count').unsigned();

// Enum
table.enum('status', ['active', 'inactive']);


```

SQLite

```
// Limited ALTER TABLE support
// Automatically handles with table recreation

// Check constraints
table.integer('age').check('age >= 18');


```

ORM Integration

Prisma

```
import { PrismaMigrationIntegration } from '@philjs/db/migrations';

// Import Prisma migrations
const migrations = await PrismaMigrationIntegration.importPrismaMigrations('./prisma/migrations');

// Export to Prisma schema
const schema = await PrismaMigrationIntegration.exportToPrismaSchema(migrations);


```

Drizzle

```
import { DrizzleMigrationIntegration } from '@philjs/db/migrations';

// Import Drizzle migrations
const migrations = await DrizzleMigrationIntegration.importDrizzleMigrations('./drizzle');

// Export to Drizzle schema
const schema = await DrizzleMigrationIntegration.exportToDrizzleSchema(migrations);


```

Advanced Features

Auto-Migration

Generate migrations automatically from schema changes:

```
import { AutoMigrationGenerator } from '@philjs/db/migrations';

const generator = new AutoMigrationGenerator(config);
const result = await generator.generate({
    compare: true,
    name: 'auto_migration',
    dryRun: true,
});

console.log('SQL:', result.sql);
console.log('Warnings:', result.warnings);


```

Schema Diff

Compare schemas and generate diff:

```

import { SchemaDiffGenerator } from '@philjs/db/migrations';

const generator = new SchemaDiffGenerator(config);
const diff = await generator.generate();

console.log('Created tables:', diff.tables.created);
console.log('Dropped tables:', diff.tables.dropped);
console.log('Modified tables:', diff.tables.modified);

```

Backup & Restore

```

import { BackupManager } from '@philjs/db/migrations';

const backup = new BackupManager(config);

// Create backup
const filename = await backup.createBackup();

// Restore backup
await backup.restoreBackup(filename);

```

Data Migration Helpers

```

import { DataMigrationHelper } from '@philjs/db/migrations';

// Transform data
await DataMigrationHelper.transformData(
  'users',
  (row) => ({
    ...row,
    email: row.email.toLowerCase(),
  })
);

// Copy data
await DataMigrationHelper.copyData('old_table', 'new_table', ['id', 'name']);

// Validate data
await DataMigrationHelper.migrateWithValidation(
  'users',
  (row) => row.email.includes('@'),
  (row) => console.error('Invalid:', row)
);

```

Best Practices

1. **Always test migrations:** Use dry-run mode first
2. **Use transactions:** Enable for data integrity
3. **Backup production:** Always backup before migrating
4. **Version control:** Commit migration files
5. **Down migrations:** Always implement rollback
6. **Batch data:** Use batch operations for large datasets
7. **Index carefully:** Add indexes for frequently queried columns
8. **Foreign keys:** Use CASCADE appropriately
9. **Test rollback:** Verify down migrations work
10. **Document changes:** Add comments to complex migrations

Examples

See examples/migration-examples.ts for comprehensive examples including:

- Table creation with foreign keys
- Column alterations
- Data migrations
- Batch operations
- Full-text search
- Soft deletes
- Junction tables
- Raw SQL migrations
- And more...

Troubleshooting

Migration conflicts

```
# Check status for conflicts
npx philjs db migrate:status

# Resolution: Ensure all executed migrations have corresponding files
```

Transaction errors

```
# Disable transactions for specific migration
export default {
  transaction: false,
  async up(context) { ... }
}
```

Rollback failures

```
# Use step-by-step rollback
npx philjs db migrate:rollback --step 1

# Check migration code for errors
```

License

MIT

Quick Start Guide - Multi-Framework Islands

Get started with PhilJS multi-framework islands in 5 minutes.

Installation

```
# Install @philjs/islands
pnpm add @philjs/islands

# Install frameworks you want to use (pick one or more)
pnpm add react react-dom      # React
pnpm add vue                   # Vue
pnpm add svelte                # Svelte
pnpm add preact                # Preact
pnpm add solid-js               # Solid
```

Step 1: Configure Vite

Create or update vite.config.ts:

```
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';
import vue from '@vitejs/plugin-vue';
import { viteMultiFramework } from '@philjs/islands/vite-multi-framework';

export default defineConfig({
  plugins: [
    react(), // Add framework plugins
    vue(),
    viteMultiFramework({ // Add PhilJS plugin
      frameworks: ['react', 'vue'],
      splitByFramework: true,
      generateManifest: true
    })
  ]
});
```

Step 2: Create Island Components

React Component (`islands/Counter.tsx`)

```
import { useState } from 'react';

export default function Counter({ initial = 0 }) {
  const [count, setCount] = useState(initial);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(c => c + 1)}>+</button>
    </div>
  );
}
```

Vue Component (`islands/TodoList.vue`)

```

<script setup lang="ts">
import { ref } from 'vue';

const todos = ref(['Learn PhilJS', 'Build Islands']);
const newTodo = ref('');

function addTodo() {
  if (newTodo.value) {
    todos.value.push(newTodo.value);
    newTodo.value = '';
  }
}
</script>

<template>
<div>
  <input v-model="newTodo" @keyup.enter="addTodo" />
  <button @click="addTodo">Add</button>
  <ul>
    <li v-for="todo in todos" :key="todo">{{ todo }}</li>
  </ul>
</div>
</template>

```

Step 3: Register Islands

In your main app file:

```

import { registerIslandComponent } from '@philjs/islands';

// Register components
registerIslandComponent('react', 'Counter', () => import('./islands/Counter.tsx'));
registerIslandComponent('vue', 'TodoList', () => import('./islands/TodoList.vue'));

```

Step 4: Use Islands

```

import { MultiFrameworkIsland } from '@philjs/islands';

export default function App() {
  return (
    <div>
      <h1>My App</h1>

      {/* React Island - Hydrate when visible */}
      <MultiFrameworkIsland
        framework="react"
        component="Counter"
        props={{ initial: 0 }}
        hydration={{ strategy: 'visible' }}
      />

      {/* Vue Island - Hydrate when idle */}
      <MultiFrameworkIsland
        framework="vue"
        component="TodoList"
        props={{ items: [] }}
        hydration={{ strategy: 'idle' }}
      />
    </div>
  );
}

```

Done!

That's it! You now have a multi-framework application with:

- Automatic code splitting per framework
- Intelligent hydration strategies
- Type-safe component props

Next Steps

Share State Between Islands

```

import { createSharedState } from '@philjs/islands';

const userState = createSharedState('user', {
  name: 'Guest',
  isLoggedIn: false
});

// Update from any island
userState.updateState({ name: 'John', isLoggedIn: true });

```

Use Event Bus

```

import { eventBus } from '@philjs/islands';

// Emit from one island
eventBus.emit('user-login', { userId: 123 });

// Listen in another island
eventBus.on('user-login', (data) => {
  console.log(`User logged in: ${data.userId}`);
});

```

Framework-Specific Hooks

```

// React
import { frameworkHooks } from '@philjs/islands';

function MyComponent() {
  const [state, setState] = frameworkHooks.react.useState('app', {});
  const bus = frameworkHooks.react.useEventBus();
  // ...
}

```

Hydration Strategies

Choose the right strategy for each island:

Strategy	When to Use	Example
immediate	Critical, above-the-fold	Navigation, search
visible	Scroll-dependent content	Comments, feeds
idle	Non-critical features	Analytics, chat
interaction	On-demand features	Modals, dropdowns
media	Device-specific	Desktop-only widgets

Common Patterns

Multiple Frameworks on Same Page

```

<div>
  <MultiFrameworkIsland framework="react" component="Header" />
  <MultiFrameworkIsland framework="vue" component="Sidebar" />
  <MultiFrameworkIsland framework="svelte" component="Footer" />
</div>

```

Sharing Data

```

// Create shared state
const cartState = createSharedState('cart', { items: [], total: 0 });

// Island 1 (React) - Add to cart
function ProductCard() {
  const [cart, setCart] = frameworkHooks.react.useState('cart', {});
  // ...
}

// Island 2 (Vue) - Display cart
const { state: cart } = frameworkHooks.vue.useState('cart', {});

```

Cross-Island Events

```

// Island 1 - Emit event
eventBus.emit('product-added', { id: 123, price: 29.99 });

// Island 2 - Listen for event
eventBus.on('product-added', (product) => {
  showNotification(`Added ${product.id}`);
});

```

Troubleshooting

Island not hydrating?

- Check that component is registered
- Verify framework plugin is in vite.config.ts
- Check browser console for errors

Props not working?

- Ensure props are JSON-serializable
- Check prop names match component expectations

- Verify data-props attribute in HTML

Framework not detected?

- Add framework to frameworks array in vite config
- Check that component imports the framework
- Try explicit framework="react" instead of "auto"

Learn More

- [Full Documentation](#)
- [Islands Guide](#)
- [SSR Overview](#)

PhilJS Multi-Framework Islands

PhilJS-first island architecture with optional legacy-framework wrappers. Prefer PhilJS components; use React/Vue/Svelte/Preact/Solid islands only when migrating.

Features

- Legacy Framework Support:** Optional React, Vue, Svelte, Preact, and Solid wrappers
- Selective Hydration:** Only hydrate interactive components when needed
- Multiple Strategies:** Immediate, visible, idle, interaction, and media-based hydration
- Framework Auto-Detection:** Automatically detect component frameworks
- Code Splitting:** Automatic framework-specific code splitting
- Shared State:** Cross-framework state management
- Event Bus:** Inter-island communication
- Props Normalization:** Automatic props conversion between frameworks
- Vite Plugin:** Build-time optimization and manifest generation
- TypeScript Support:** Full type safety across PhilJS components and wrappers

Installation

```
pnpm add @philjs/islands
```

Optional Framework Dependencies

Install only the legacy frameworks you need for migration:

```
# React
pnpm add react react-dom

# Vue
pnpm add vue

# Svelte
pnpm add svelte

# Preact
pnpm add preact

# Solid
pnpm add solid-js
```

Quick Start

1. Configure Vite

```
// vite.config.ts
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';
import vue from '@vitejs/plugin-vue';
import { svelte } from '@sveltejs/vite-plugin-svelte';
import { viteMultiFramework } from '@philjs/islands/vite-multi-framework';

export default defineConfig({
  plugins: [
    react(),
    vue(),
    svelte(),
    viteMultiFramework({
      frameworks: ['react', 'vue', 'svelte'],
      generateManifest: true,
      splitByFramework: true
    })
  ]
});
```

2. Register Island Components

```
import { registerIslandComponent } from '@philjs/islands';

// Register components from different frameworks
registerIslandComponent('react', 'Counter', () => import('./islands/Counter.tsx'));
registerIslandComponent('vue', 'TodoList', () => import('./islands/TodoList.vue'));
registerIslandComponent('svelte', 'Timer', () => import('./islands/Timer.svelte'));
```

3. Use Islands in Your App

```
import { MultiFrameworkIsland } from '@philjs/islands';

function App() {
  return (
    <div>
      <h1>Multi-Framework App</h1>

      {/* React Island */}
      <MultiFrameworkIsland
        framework="react"
        component="Counter"
        props={{ initial: 0 }}
        hydration={{ strategy: 'visible' }}
      />

      {/* Vue Island */}
      <MultiFrameworkIsland
        framework="vue"
        component="TodoList"
        props={{ items: [] }}
        hydration={{ strategy: 'idle' }}
      />

      {/* Svelte Island */}
      <MultiFrameworkIsland
        framework="svelte"
        component="Timer"
        props={{ autoStart: true }}
        hydration={{ strategy: 'immediate' }}
      />
    </div>
  );
}
```

Hydration Strategies

Immediate

Hydrate as soon as possible:

```
<MultiFrameworkIsland
  framework="react"
  component="CriticalComponent"
  hydration={{ strategy: 'immediate' }}
/>
```

Visible

Hydrate when scrolled into viewport:

```
<MultiFrameworkIsland
  framework="vue"
  component="LazyComponent"
  hydration={{{
    strategy: 'visible',
    rootMargin: '50px',
    threshold: 0.1
  }}}
/>
```

Idle

Hydrate when browser is idle:

```
<MultiFrameworkIsland
  framework="svelte"
  component="NonCriticalComponent"
  hydration={{{
    strategy: 'idle',
    timeout: 2000
  }}}
/>
```

Interaction

Hydrate on user interaction:

```
<MultiFrameworkIsland
  framework="preact"
  component="InteractiveWidget"
  hydration={{
    strategy: 'interaction',
    events: ['click', 'mouseenter']
  }}
/>
```

Media Query

Hydrate based on media query:

```
<MultiFrameworkIsland
  framework="solid"
  component="DesktopOnlyComponent"
  hydration={{
    strategy: 'media',
    media: '(min-width: 768px)'
  }}
/>
```

Shared State

Share state between islands from different frameworks:

```
import { createSharedState } from '@philjs/islands';

// Create shared state
const userState = createSharedState('user', {
  name: 'John Doe',
  isLoggedIn: true
});

// Read state
const currentUser = userState.getState();

// Update state
userState.setState({ name: 'Jane Doe', isLoggedIn: true });

// Partial update
userState.updateState({ isLoggedIn: false });

// Subscribe to changes
const unsubscribe = userState.subscribe((state) => {
  console.log('User changed:', state);
});
```

Framework-Specific Hooks

React

```
import { frameworkHooks } from '@philjs/islands';

function Counter() {
  const [count, setCount] = frameworkHooks.react.useSharedState('counter', 0);
  const eventBus = frameworkHooks.react.useEventBus();

  const increment = () => {
    setCount(c => c + 1);
    eventBus.emit('counter-changed', { value: count + 1 });
  };

  return <button onClick={increment}>Count: {count}</button>;
}
```

Vue

```
<script setup>
import { frameworkHooks } from '@philjs/islands';

const { state: count, setState: setCount } = frameworkHooks.vue.useSharedState('counter', 0);
const eventBus = frameworkHooks.vue.useEventBus();

function increment() {
  setCount(c => c + 1);
  eventBus.emit('counter-changed', { value: count.value + 1 });
}
</script>

<template>
  <button @click="increment">Count: {{ count }}</button>
</template>
```

Svelte

```
<script>
import { frameworkHooks } from '@philjs/islands';

const count = frameworkHooks.svelte.createSharedStore('counter', 0);
const eventBus = frameworkHooks.svelte.useEventBus();

function increment() {
  count.update(c => c + 1);
  eventBus.emit('counter-changed', { value: $count + 1 });
}
</script>

<button on:click={increment}>Count: {$count}</button>
```

Solid

```
import { frameworkHooks } from '@philjs/islands';

function Counter() {
  const [count, setCount] = frameworkHooks.solid.createSharedSignal('counter', 0);
  const eventBus = frameworkHooks.solid.useEventBus();

  const increment = () => {
    setCount(c => c + 1);
    eventBus.emit('counter-changed', { value: count() + 1 });
  };

  return <button onClick={increment}>Count: {count()}</button>;
}
```

Event Bus

Communicate between islands:

```
import { eventBus } from '@philjs/islands';

// Emit event
eventBus.emit('user-logged-in', { userId: 123 });

// Listen to event
const unsubscribe = eventBus.on('user-logged-in', (data) => {
  console.log('User logged in:', data.userId);
});

// Listen once
eventBus.once('app-initialized', () => {
  console.log('App ready!');
});

// Remove listener
eventBus.off('user-logged-in', handler);

// Add middleware
eventBus.use((event, data) => {
  console.log('Event:', event, data);
  return data; // Can transform data
});
```

Island Bridges

Create typed communication channels between specific islands:

```
import { createIslandBridge } from '@philjs/islands';

const bridge = createIslandBridge(
  { framework: 'react', id: 'island-1' },
  { framework: 'vue', id: 'island-2' }
);

// Send data from React island
bridge.send({ message: 'Hello from React!', count: 42 });

// Receive in Vue island
bridge.receive((data) => {
  console.log('Received:', data);
});
```

Props Normalization

Automatically convert props between framework conventions:

```

import { PropsNormalizer } from '@philjs/islands';

// React props
const reactProps = {
  className: 'btn',
  onClick: () => {},
  htmlFor: 'input-1'
};

// Normalize
const normalized = PropsNormalizer.normalize(reactProps, 'react');
// Result: { class: 'btn', 'on:click': Function, for: 'input-1' }

// Convert to Vue
const vueProps = PropsNormalizer.denormalize(normalized, 'vue');
// Result: { class: 'btn', '@click': Function, for: 'input-1' }

```

Framework Adapters

Custom Adapter

Create your own framework adapter:

```

import { registerAdapter, type FrameworkAdapter } from '@philjs/islands';

const customAdapter: FrameworkAdapter = {
  name: 'custom-framework',

  detect(component) {
    // Detect if component belongs to this framework
    return component.__customFramework === true;
  },

  async hydrate(element, component, props, strategy) {
    // Hydrate the component
    const instance = new CustomFramework.Component(component, props);
    instance.mount(element);
  },

  async unmount(element) {
    // Clean up
    const instance = element.__customInstance;
    if (instance) instance.destroy();
  },

  serializeProps(props) {
    return JSON.stringify(props);
  },

  deserializeProps(serialized) {
    return JSON.parse(serialized);
  },

  getPeerDependencies() {
    return ['custom-framework'];
  }
};

registerAdapter(customAdapter);

```

Vite Plugin Options

```

import { viteMultiFramework } from '@philjs/islands/vite-multi-framework';

viteMultiFramework({
  // Frameworks to support
  frameworks: ['react', 'vue', 'svelte', 'preact', 'solid'],

  // Islands directory
  islandsDir: 'src/islands',

  // File patterns to scan
  include: ['**/*.tsx', '**/*.jsx', '**/*.vue', '**/*.svelte'],

  // Files to exclude
  exclude: ['node_modules/**', 'dist/**'],

  // Generate manifest.json
  generateManifest: true,

  // Island detection pattern
  islandPattern: /<Island\s+framework=['"](\w+)[""]\//g,

  // Code splitting per framework
  splitByFramework: true,

  // Auto-inject directives
  autoInjectDirectives: true,

  // Debug mode
  debug: true
})

```

API Reference

Multiframework Islands

```

// Island component (SSR)
Island(config: MultiFrameworkIslandConfig): string

// Hydrate single island
hydrateMultiFrameworkIsland(element: HTMLElement): Promise<void>

// Hydrate all islands
hydrateAllMultiFrameworkIslands(root?: HTMLElement): void

// Unmount island
unmountIsland(id: string): Promise<void>

// Get island instance
getIsland(id: string): IslandInstance | undefined

// Get all islands
getAllIslands(): IslandInstance[]

// Register component
registerIslandComponent(framework: string, name: string, loader: () => Promise<any>): void

// Initialize islands
initMultiFrameworkIslands(root?: HTMLElement): void

```

Framework Adapters

```

// Get adapter by name
getAdapter(framework: string): FrameworkAdapter | undefined

// Auto-detect framework
detectFramework(component: any): FrameworkAdapter | undefined

// Register custom adapter
registerAdapter(adapter: FrameworkAdapter): void

// Check support
isFrameworkSupported(framework: string): boolean

// List frameworks
getSupportedFrameworks(): string[]

```

Shared State

```
// Create shared state
createSharedState<T>(name: string, initial: T): SharedStateStore<T>

// Get existing state
getSharedState<T>(name: string): SharedStateStore<T> | undefined

// Remove state
removeSharedState(name: string): void

// State store methods
store.getState(): T
store.setState(newState: T | (prev: T) => T): void
store.updateState(partial: Partial<T>): void
store.subscribe(callback: (state: T) => void): () => void
store.use(middleware: (state: T, next: T) => T): void
```

Event Bus

```
// Emit event
eventBus.emit<T>(event: string, data?: T): void

// Listen to event
eventBus.on<T>(event: string, callback: (data: T) => void): () => void

// Listen once
eventBus.once<T>(event: string, callback: (data: T) => void): void

// Remove listener
eventBus.off(event: string, callback?: Function): void

// Add middleware
eventBus.use(middleware: (event: string, data: any) => any): void
```

Performance Tips

1. **Use Lazy Hydration:** Default to visible or idle strategies
2. **Minimize Immediate Islands:** Only critical components should use immediate
3. **Code Split by Framework:** Enable splitByFramework in Vite plugin
4. **Share State Sparingly:** Only for truly global state
5. **Batch Updates:** Group state changes together
6. **Use Media Queries:** Load components only on appropriate devices
7. **Monitor Bundle Size:** Check framework chunks in build output

Examples

For full examples, see:

- [Islands](#)
- [SSR Overview](#)
- [WebAssembly](#)

Browser Support

- Modern browsers with ES Modules support
- IntersectionObserver (with fallback)
- requestIdleCallback (with fallback)
- matchMedia (with fallback)

License

MIT

PhilJS Router - Advanced Features

This document provides an overview of the advanced routing features available in PhilJS Router.

Table of Contents

1. [Router DevTools](#)
2. [Route Groups](#)
3. [Route Masking](#)
4. [Router Context](#)

Router DevTools

TanStack Router-style developer tools for debugging and monitoring your routes.

Features

- **Visual Route Tree Display:** See your entire route hierarchy at a glance
- **State Inspector:** Inspect params, search params, and loader data
- **Navigation History:** Track all navigations with timestamps and metrics
- **Performance Metrics:** Monitor load times per route
- **Route Matching Debugger:** Debug why routes match or don't match
- **Live Updates:** Real-time updates as you navigate

Usage

```
import { RouterDevTools } from '@philjs/router';

function App() {
  return (
    <>
      <RouterView />
      {/* Add DevTools - only shows in development */}
      <RouterDevTools />
    </>
  );
}
```

Configuration

```
import { initRouterDevTools } from '@philjs/router';

initRouterDevTools({
  position: 'bottom',           // 'bottom' | 'top' | 'left' | 'right'
  size: 400,                   // Height/width in pixels
  minimized: false,            // Start minimized
  maxHistoryEntries: 100,       // Max navigation history entries
  showPerformance: true,        // Enable performance tracking
  autoTrack: true,              // Auto-track navigations
  theme: 'dark',                // 'light' | 'dark' | 'system'
});
```

Programmatic Access

```
import {
  trackNavigation,
  completeNavigation,
  trackLoader,
  updateRouteTree,
  exportState,
} from '@philjs/router';

// Track navigation
trackNavigation('/users/123', { id: '123' }, searchParams);

// Track loader performance
trackLoader('users-route', 50); // 50ms

// Complete navigation with metrics
completeNavigation({
  total: 150,
  matching: 10,
  dataLoading: 100,
  rendering: 40,
});

// Export state for debugging
const state = exportState();
console.log(state);
```

Route Groups

SolidStart-style route groups for organizing routes without affecting URLs.

Features

- **URL-agnostic Organization:** Group routes without adding path segments
- **Shared Layouts:** Apply layouts to all routes in a group
- **Group-level Middleware:** Authentication, permissions, logging, rate limiting
- **Better Code Organization:** Keep related routes together

Basic Usage

```

routes/
  (marketing)/
    about.tsx      -> /about
    contact.tsx   -> /contact
    layout.tsx     -> shared layout
  (dashboard)/
    settings.tsx -> /settings
    profile.tsx   -> /profile
    layout.tsx     -> shared layout

```

Creating Route Groups

```

import { createRouteGroup, addRouteToGroup } from '@philjs/router';

const marketingGroup = createRouteGroup('marketing', {
  layout: MarketingLayout,
  meta: {
    displayName: 'Marketing Pages',
    description: 'Public marketing content',
  },
  routes: [
    { path: '/about', component: AboutPage },
    { path: '/contact', component: ContactPage },
  ],
});

```

Middleware

```

import {
  createAuthMiddleware,
  createPermissionMiddleware,
  createLoggingMiddleware,
  createRateLimitMiddleware,
} from '@philjs/router';

const adminGroup = createRouteGroup('admin', {
  middleware: [
    // Require authentication
    createAuthMiddleware(checkAuth, '/login'),

    // Require admin permission
    createPermissionMiddleware(
      ['admin'],
      getPermissions,
      '/unauthorized'
    ),

    // Log all accesses
    createLoggingMiddleware(),

    // Rate limit
    createRateLimitMiddleware({
      maxRequests: 100,
      windowMs: 60000, // 1 minute
    }),
  ],
});

```

Processing Groups

```

import { processRouteGroups } from '@philjs/router';

const groups = [marketingGroup, dashboardGroup, adminGroup];
const routes = processRouteGroups(groups);

// Routes now have group layouts and middleware applied

```

File-based Discovery

```

import { discoverRouteGroups } from '@philjs/router';

const files = import.meta.glob('./(*)/*.tsx', { eager: true });
const groups = discoverRouteGroups(files);

```

Route Masking

Display a different URL in the browser than the actual route being rendered.

Features

- **Modal Routes:** Show modals with clean URLs
- **Drawer Routes:** Side panels with URL preservation
- **Parallel Routes:** Multiple routes active simultaneously

- **Stack Management:** Nested modals/drawers
- **History Integration:** Works with browser back/forward

Basic Usage

```
import { navigateWithMask } from '@philjs/router';

// Navigate to photo detail but show /photos in URL
navigateWithMask('/photos/123', {
  masks: '/photos',
  state: { photoId: '123' },
});
```

Modal Pattern

```
import { navigateAsModal, closeOverlay } from '@philjs/router';

// Open modal
function openPhotoModal(id: string) {
  navigateAsModal('/photos/${id}', {
    backgroundRoute: '/gallery',
    state: { photoId: id },
  });
}

// Close modal
function closePhotoModal() {
  closeOverlay({ navigate: true });
}
```

Drawer Pattern

```
import { navigateAsDrawer } from '@philjs/router';

function openSettings() {
  navigateAsDrawer('/settings/detail', {
    backgroundRoute: '/app',
    side: 'right', // 'left' | 'right' | 'top' | 'bottom'
    state: { drawer: true },
  });
}
```

Nested Masking

```
import { pushMask, popMask } from '@philjs/router';

// Push masks onto stack
const mask1 = createRouteMask('/modal1', '/background');
const mask2 = createRouteMask('/modal2', '/modal1');

pushMask(mask1);
pushMask(mask2); // Modal on top of modal

// Pop to go back
popMask(); // Returns to modal1
popMask(); // Returns to background
```

Hooks

```
import {
  useRouteMask,
  useIsRouteMasked,
  useActualRoute,
  useMaskedUrl,
} from '@philjs/router';

function MyComponent() {
  const mask = useRouteMask();
  const isMasked = useIsRouteMasked();
  const actualRoute = useActualRoute();
  const maskedUrl = useMaskedUrl();

  if (isMasked) {
    return <Modal actualRoute={actualRoute} />;
  }

  return <NormalView />;
}
```

Router Context

Global context available to all routes with type-safe injection.

Features

- **Type-safe Context:** Full TypeScript support
- **Provider Pattern:** Async data providers with caching
- **Route Overrides:** Different context per route
- **Middleware:** Transform context before use
- **Validation:** Ensure context integrity

Basic Usage

```
import { initRouterContext, setGlobalContext } from '@philjs/router';

// Initialize with context
initRouterContext({
  initialContext: {
    user: currentUser,
    theme: 'dark',
    api: apiClient,
  },
});

// Access in loader
export async function loader({ context }) {
  const { user, api } = context;
  return api.fetchData(user.id);
}

// Access in component
function MyComponent() {
  const context = useRouterContext();
  return <div>Hello {context.user.name}</div>;
}
```

Context Providers

```
import {
  registerContextProvider,
  defineContextProvider,
  createUserContextProvider,
} from '@philjs/router';

// Define provider
const apiProvider = defineContextProvider(
  'api',
  async () => createApiClient(),
  { cache: true, scope: 'global' }
);

registerContextProvider(apiProvider);

// Built-in providers
registerContextProvider(
  createUserContextProvider(fetchCurrentUser)
);
```

Route-specific Context

```
import { registerRouteContextOverride } from '@philjs/router';

// Override context for specific routes
registerRouteContextOverride({
  route: '/admin/*',
  context: {
    isAdmin: true,
    permissions: ['read', 'write', 'delete'],
  },
  merge: true, // Merge with global context
});
```

Context Middleware

```
import {
  addContextMiddleware,
  defineContextMiddleware,
} from '@philjs/router';

const loggingMiddleware = defineContextMiddleware((ctx, { route }) => {
  console.log(`Route ${route} accessing context:`, Object.keys(ctx));
  return ctx;
});

addContextMiddleware(loggingMiddleware);
```

Typed Context

```

import { createTypedContext } from '@philjs/router';

type ApplicationContext = {
  user: User;
  theme: 'light' | 'dark';
  api: ApiClient;
};

const typedContext = createTypedContext<ApplicationContext>();

// Fully typed
const context = typedContext.useContext();
const user = typedContext.useValue('user'); // Type: User

```

Validation

```

import { initRouterContext } from '@philjs/router';

initRouterContext({
  validators: {
    user: {
      validate: (value) => value && typeof value === 'object',
      errorMessage: 'User must be an object',
    },
    theme: {
      validate: (value) => ['light', 'dark'].includes(value),
      errorMessage: 'Theme must be light or dark',
    },
  },
});

```

Built-in Helpers

```

import {
  createUserContextProvider,
  createThemeContextProvider,
  createLocaleContextProvider,
  createApiClientProvider,
} from '@philjs/router';

// User context
registerContextProvider(
  createUserContextProvider(async () => {
    return await fetchUser();
  })
);

// Theme context
registerContextProvider(
  createThemeContextProvider(() => 'dark')
);

// Locale context
registerContextProvider(
  createLocaleContextProvider(() => 'en-US')
);

// API client context
registerContextProvider(
  createApiClientProvider(() => new ApiClient())
);

```

Complete Example

Here's a complete example using all features:

```

import {
  createAppRouter,
  RouterView,
  RouterDevTools,
  initRouterContext,
  createUserContextProvider,
  createRouteGroup,
  processRouteGroups,
  createAuthMiddleware,
  navigateAsModal,
  closeOverlay,
} from '@philjs/router';

// Set up context
initRouterContext({
  initialContext: {
    appName: 'MyApp',
    theme: 'dark',
  },
}

```

```

    providers: [
      createUserContextProvider(fetchUser),
    ],
  });

// Create route groups
const publicGroup = createRouteGroup('public', {
  routes: [
    { path: '/', component: HomePage },
    { path: '/about', component: AboutPage },
  ],
});

const dashboardGroup = createRouteGroup('dashboard', {
  layout: DashboardLayout,
  middleware: [
    createAuthMiddleware(checkAuth, '/login'),
  ],
  routes: [
    { path: '/dashboard', component: DashboardPage },
    { path: '/settings', component: SettingsPage },
  ],
});

// Process groups
const routes = processRouteGroups([publicGroup, dashboardGroup]);

// Create router
const router = createAppRouter({
  routes,
  prefetch: true,
  transitions: true,
});

// App component
function App() {
  return (
    <>
      <RouterView />
      <RouterDevTools />
    </>
  );
}

// Modal example
function PhotoGrid() {
  const handlePhotoClick = (id: string) => {
    navigateAsModal('/photos/${id}', {
      backgroundRoute: '/gallery',
      state: { photoid: id },
    });
  };

  return (
    <div>
      {photos.map(photo => (
        <img onClick={() => handlePhotoClick(photo.id)} />
      ))}
    </div>
  );
}

function PhotoModal() {
  const mask = useRouteMask();

  if (!mask?.state?.photoId) return null;

  return (
    <Modal onClose={() => closeOverlay()}>
      <PhotoDetail id={mask.state.photoId} />
    </Modal>
  );
}
}

```

API Reference

For detailed API documentation, see the TypeScript definitions in each module:

- [devtools.ts](#) - Router DevTools
- [route-groups.ts](#) - Route Groups
- [route-masking.ts](#) - Route Masking
- [router-context.ts](#) - Router Context

All exports are available from the main package:

```
import {
  // DevTools
  RouterDevTools,
  initRouterDevTools,

  // Route Groups
  createRouteGroup,
  processRouteGroups,

  // Route Masking
  navigateAsModal,
  navigateAsDrawer,

  // Router Context
  useRouterContext,
  initRouterContext,
} from '@philjs/router';
```

Testing

All features include comprehensive test suites:

- `devtools.test.ts` - DevTools tests
- `route-groups.test.ts` - Route Groups tests
- `route-masking.test.ts` - Route Masking tests
- `router-context.test.ts` - Router Context tests

Run tests with:

```
npm test
```

Performance Considerations

DevTools

- DevTools are lightweight and only track when enabled
- Set `autoTrack: false` to manually control tracking
- Clear history periodically with `clearHistory()`

Route Groups

- Groups are processed once during initialization
- Middleware runs on every navigation - keep them fast
- Use caching in providers when possible

Route Masking

- Mask stack is limited by `maxStackDepth`
- History is automatically trimmed to `maxHistorySize`
- Old masks are not restored if past `maxAge`

Router Context

- Providers with `cache: true` only run once
- Context middleware runs on every route change
- Clear cache with `clearContextCache()` when needed

Browser Support

All features work in modern browsers with:

- ES2020+ support
- URL API
- History API
- Signal/reactive primitives

For older browsers, use appropriate polyfills.

License

MIT

CI/CD for PhilJS

Automate builds, tests, and deploys to keep quality high and releases fast.

Pipelines

- **Install:** pnpm install --frozen-lockfile
- **Static checks:** pnpm lint, pnpm typecheck, pnpm size
- **Tests:** pnpm test -- --runInBand, pnpm --filter @philjs/core vitest bench
- **Build:** pnpm build (client + server)
- **E2E:** pnpm exec playwright test --reporter=line
- **Artifacts:** upload build outputs, Playwright traces on failure

Caching

- Cache pnpm store and optionally Playwright browsers.
- Avoid caching node_modules; prefer pnpm store for determinism.

Preview environments

- Deploy PR previews (edge/serverless) for review.
- Run smoke tests against previews to catch environment-specific issues.

Branching and gating

- Protect main with required checks (lint, typecheck, test, size).
- Use feature flags for risky changes; toggle in production without redeploy.

Secrets and env

- Inject via CI secrets; never commit.
- Validate required env in build/test; fail fast if missing.

Observability hooks

- Emit build id/commit hash into responses for traceability.
- Post metrics (test duration, failure counts) to dashboards.

Rollbacks

- Keep previous build artifacts; support quick rollbacks in deploy platform.
- Automate cache invalidation on rollback if HTML/data changed.

Checklist

- CI runs lint/typecheck/tests/size/build/E2E.
- Caches configured (pnpm store, browsers).
- Previews deployed per PR with smoke tests.
- Secrets managed via CI; env validated.
- Rollback plan documented and tested.

Release Management

Plan and execute PhilJS releases with confidence and traceability.

Versioning and change control

- Use semantic versioning for your app/packages (even if PhilJS stays at 0.1.0).
- Generate changelogs from commits/changesets; highlight breaking changes and migrations.
- Tag releases in git; attach artifacts (build hashes, docs links).

Pre-release checklist

- CI green: lint, typecheck, tests, size, build, E2E.
- Benchmarks stable vs baseline; bundle sizes within budget.
- Docs updated (API changes, migration notes).
- Observability dashboards updated for new metrics/routes.
- Feature flags ready for risky features.

Release process

- Cut a release branch; freeze non-critical changes.
- Build and deploy to staging; run smoke/E2E.
- Promote to production; monitor metrics (errors, TTFB, cache hits, business KPIs).
- Gradual rollout if platform supports it; ramp traffic while watching alerts.

Post-release

- Announce changes (internal/external) with highlights and upgrade steps.
- Track regressions; open follow-up issues with owners and deadlines.
- Clean up feature flags when stabilized.

Rollback

- Keep previous build artifacts; enable quick rollback in platform.
- Invalidate caches to clear bad HTML/data.
- Document rollback steps per platform (edge/serverless/static).

Compliance/audit notes

- Keep release notes and build hashes for audit trails.
- Record who approved and when.

Checklist

- Changelog and version tags updated.
- CI + staging smoke green.
- Observability/alerts in place for new release.
- Rollout plan and rollback steps ready.
- Feature flags toggled appropriately post-release.

DevOps Checklists

Pre-merge

- `pnpm lint`, `pnpm typecheck`, `pnpm test`, `pnpm size` pass.
- Key Playwright smoke tests green; traces captured on failure.
- Bundle sizes within budget; no regression in benches.
- Security headers/CSP unaffected by changes (if relevant).
- Docs updated for behavior changes.

Pre-release

- Staging deploy green; SSR/edge preview tested.
- Observability dashboards updated (new routes/metrics).
- Feature flags set for risky features; defaults safe.
- Backups/rollbacks tested or ready.
- Changelog/release notes prepared.

Post-release

- Monitor TTFB/error rate/cache hit ratio for 24–48h.
- Address alerts quickly; roll back if necessary.
- Clean up feature flags after stabilization.
- Update docs with any hotfixes.

Infrastructure hygiene

- Env/secret validation at startup.
- Least privilege for edge/serverless roles.
- Dependency scanning scheduled.
- Build artifacts reproducible; tagged per release.

Publishing and Formats (PDF/EPUB/Kindle)

How to turn the PhilJS book into distributable formats.

Source layout

- Markdown lives in `docs/philjs-book/src`.
- `SUMMARY.md` defines navigation and export order.
- Visuals live in `docs/philjs-book/visuals` (SVG/PNG).
- Cover page is in `docs/philjs-book/src/cover.md`.

PDF and EPUB (pandoc)

Use the export script:

```
node scripts/export-book.mjs
```

Outputs:

- docs/philjs-book/dist/philjs-book.epub
- docs/philjs-book/dist/philjs-book.pdf

The script:

- Reads file order from SUMMARY.md.
- Includes docs/philjs-book/visuals via --resource-path.
- Applies docs/philjs-book/src/epub.css for EPUB styling and docs/philjs-book/src/pdf.css for PDF styling.
- Uses cover.png (if present) for EPUB cover metadata.
- Converts SVG visuals to PNG via Inkscape for EPUB/PDF compatibility.
- Uses wkhtmltopdf when available for PDF; falls back to weasyprint/prince, then pdflatex or PDF_ENGINE.

Kindle (KDP)

- Convert the EPUB with Kindle Previewer to generate .kpf.
- Validate layout and internal links on multiple device profiles.

Optional upgrades

- Use mdbook-pdf or Prince for higher-fidelity PDFs.
- Add a print-specific CSS file if you need headers/footers or page numbers.

Link correctness

- Keep links relative; pandoc preserves them in PDF/EPUB.
- Run a link checker before export (node scripts/book-links.mjs or pnpm book:links).

Image handling

- Prefer SVG for diagrams; provide PNG if Kindle rendering needs it.
- Add alt text and captions for every visual.

QA before publish

- Verify TOC and index links.
- Spot-check code blocks and diagrams in PDF/EPUB.
- Confirm cover and front matter appear in the output.

eBook Export Checklist

- SUMMARY.md up to date; all chapters reachable.
- Links verified (run node scripts/book-links.mjs or similar).
- Images optimized; SVG preferred, PNG fallbacks where needed; alt text provided.
- Inkscape installed (or PNG versions available) for SVG conversion in exports.
- epub.css and pdf.css present for typography.
- Cover page in cover.md renders; add visuals/cover.png for EPUB metadata cover.
- Metadata set: title "PhilJS Book", author, version 0.1.0.
- Code blocks have info strings for syntax highlighting.
- No absolute file:// links; all links relative.
- Run node scripts/export-book.mjs to generate PDF/EPUB.
- Validate EPUB with epubcheck; preview Kindle via Kindle Previewer.

Publishing Workflow (End-to-End)

Goal: produce a single artifact (cover, front matter, chapters, appendices, index) and export to PDF/EPUB/Kindle reliably.

Layout

- Content: docs/philjs-book/src (markdown, cover, front matter, chapters, appendices, index).
- Visuals: docs/philjs-book/visuals (SVG/PNG, cover art, diagram sources).
- Styles: docs/philjs-book/src/epub.css (EPUB styling) and docs/philjs-book/src/pdf.css (PDF styling).
- Export script: scripts/export-book.mjs (uses SUMMARY order).
- Package atlas generator: scripts/generate-package-atlas.mjs (syncs packages into the book and refreshes API snapshots in package READMEs).
- Link checker: scripts/book-links.mjs (internal links and fragments).

Steps

1. Ensure `SUMMARY.md` is up to date (drives TOC and export order).
2. Regenerate the package atlas and API snapshots (keeps the book + package docs aligned with `packages/`): `node scripts/generate-package-atlas.mjs`.
3. Run link check (optional but recommended): `node scripts/book-links.mjs` or `pnpm book:links`.
4. Build exports:

```
node scripts/export-book.mjs
```

Outputs `docs/philjs-book/dist/philjs-book.{epub,pdf}` with cover and front matter.
5. Kindle: import `philjs-book.epub` into Kindle Previewer to generate `.kpf` for KDP.

Cover and front matter

- `cover.md` embeds `visuals/cover.svg` for a printable cover page.
- To embed an EPUB cover image, add `docs/philjs-book/visuals/cover.png` (the export script picks it up).
- Front matter can be expanded with extra markdown files listed early in `SUMMARY.md`.

Tooling notes

- `export-book.mjs` auto-detects pandoc; set `PANDOC=/path/to/pandoc` if it is not on PATH.
- `export-book.mjs` prefers `wkhtmltopdf` (set `WKHTMLTOPDF=/path/to/wkhtmltopdf` if needed), then `weasyprint/prince`, and falls back to `pdflatex` or `PDF_ENGINE`.
- `export-book.mjs` converts SVG visuals to PNG using Inkscape; set `INKSCAPE=/path/to/inkscape` if needed.
- `book-links.mjs` uses `lychee`; set `LYCHEE=/path/to/lychee` if it is not on PATH.

Automation

- Add npm scripts:

```
"scripts": {  
  "book:packages": "node scripts/generate-package-atlas.mjs",  
  "book:export": "node scripts/export-book.mjs",  
  "book:links": "node scripts/book-links.mjs"  
}
```

- CI can run `book:links` and `book:export` to publish artifacts.

Clickable links

- Keep links relative (as in the repo); pandoc preserves them in PDF/EPUB.
- Verify TOC and index links after export; Kindle Previewer to confirm internal links.

Checklist

- SUMMARY updated; all chapters/applications listed.
- Link check passed.
- Exports generated (PDF/EPUB); cover present.
- Kindle preview verified; internal links clickable.
- Visuals optimized (SVG/PNG) with alt text.

Mobile: Capacitor and Native

PhilJS can target mobile with Capacitor while reusing your web stack. This chapter covers project setup, plugins, performance tips, and a few mobile-only considerations. Use it alongside `packages/philjs-native/templates/capacitor/README.md` and `docs/platforms` for deeper platform notes.

Project setup (template)

The repo includes `packages/philjs-native/templates/capacitor`:

- Node 24, TypeScript 6, PhilJS 0.1.0
- Vite for web build, Capacitor CLI for native shells
- Prewired scripts for `cap:init`, `cap:add:ios`, `cap:add:android`, `cap:sync`

Flow:

1. `pnpm install`
 2. `pnpm dev` (web preview)
 3. `pnpm cap:init` (set app id/name)
 4. `pnpm cap:add:ios` or `cap:add:android`
 5. `pnpm cap:sync` then `pnpm cap:open:ios`
- For CI, run `pnpm build` then `cap sync` to prepare artifacts; prefer building native shells on the target OS (macOS for iOS).

Using plugins

Import Capacitor plugins in PhilJS components:

```

import { Camera } from '@capacitor/camera';

const takePhoto = async () => {
  const photo = await Camera.getPhoto({ resultType: 'uri' });
  return photo.webPath;
};

```

- Keep plugin calls inside effects triggered by user intent.
- Guard for platform support (web vs native).
- Handle permissions up front and provide graceful fallbacks if declined.

Performance tips

- Keep bundle size lean; mobile networks magnify payload cost.
- Use edge rendering + caching for APIs to reduce round trips.
- Prefer signals/memos over heavy stores to cut allocations.
- Avoid layout thrash: precompute styles, use CSS animations where possible.
- Use `prefetch()` for navigation targets to reduce perceived latency.
- Defer non-critical plugin initialization until after first paint.

Offline and sync

- Combine `@philjs/offline` with persistent stores (IndexedDB) for drafts.
- Use background sync (where supported) and conflict resolution on reconnect.
- Show optimistic UI with rollback for mutations.
- Persist drafts carefully; encrypt sensitive data and avoid secrets in storage.
- Consider "airplane mode" banners and queue mutations for replay.

Native UX polish

- Match platform patterns (pull-to-refresh, gestures) via `@philjs/gesture` and native status bar controls.
- Use haptics sparingly (`@capacitor/haptics`).
- Handle permissions explicitly with clear error states.
- Tune splash screen and status bar for dark/light modes.
- Use gestures (`@philjs/gesture`) for swipe navigation; respect platform norms.

Debugging

- Use browser devtools for web preview; for device debugging attach Safari/Chrome devtools to webviews.
- Log storage usage; clear caches between test runs when profiling.
- For crashes, collect native logs (Xcode/adb) and correlate with PhilJS routes.
- Verify native splash/icon assets per platform; keep adaptive icons for Android.
- Watch bundle size; run `pnpm build` and inspect output before syncing to devices.

Try it now: camera + offline draft

```

import { Camera } from '@capacitor/camera';
import { createStore } from '@philjs/core';

const [state, setState, store] = createStore({ draftPhoto: null });
store.persist({ driver: 'localStorage', paths: ['draftPhoto'] });

async function capture() {
  const photo = await Camera.getPhoto({ resultType: 'uri', quality: 70 });
  setState('draftPhoto', photo.webPath);
}

```

Add a preview component that renders `state.draftPhoto`, and confirm it persists across reloads/offline.

Edge and Serverless Targets

PhilJS adapters let you run at the edge (Vercel/Netlify/Cloudflare/Bun/Deno) or serverless (AWS, Bun, Deno, Node). The goal: push logic closer to users while keeping SSR, data loaders, and caching predictable. For detailed recipes, see `docs/deployment` and `packages/philjs-adapters/README.md`; this chapter distills the essentials.

Choosing a target

- **Edge (Vercel/Netlify/Cloudflare/Bun/Deno)**: lowest latency, great for SSR and streaming, but with tighter limits (CPU, memory, cold starts).
- **Serverless (AWS Lambda, Node)**: more flexibility, bigger timeouts/memory; good for heavy tasks and regional deployments.
- Choose per-route: marketing and dashboards at edge; exports/PDF/image tasks in regional Node/AWS.

Adapters overview

- `philjs-adapters/vercel` - Edge/Serverless + ISR, KV, Blob.
- `philjs-adapters/netlify` - Edge Functions + Functions + blob/form support.

- `philjs-adapters/cloudflare-pages` - KV/D1/R2/DO bindings.
- `philjs-adapters/aws-lambda` - SAM template and bridge handler.
- `philjs-adapters/bun` / `philjs-adapters/deno` - native runtimes.
- `philjs-adapters/node` - generic Node server adapter for long-lived processes.
- `philjs-adapters/static` - prerender/SSG option for routes that can be fully static.
- `philjs-adapters/cdn` - helpers for cache headers and manifests.

Server entry checklist

- Keep exports ESM; avoid CommonJS.
- Use `philjs build` to emit manifest and server bundle.
- In your adapter config, set:
 - `edge: true` where needed,
 - `isr/revalidate` for cached pages,
 - `kv/d1/r2` bindings for data.
- Keep environment access minimal; prefer passing config via adapter options.

Caching and revalidation

- Use loader cache tags and `revalidate` hints to control freshness.
- On Vercel/Netlify, pair ISR/edge cache with router invalidation (`cache.invalidate(['entity', id])`).
- On Cloudflare, keep HTML small; stream responses to reduce TTFB.
- Avoid cache stampedes: dedupe loader fetches and use jittered revalidation.
- For authenticated pages, cache per-session or disable HTML caching; rely on data-layer caching instead.

Env and secrets

- Read secrets from platform env; never bundle them.
- On Edge, avoid Node-only APIs; stick to Web APIs (`fetch`, `crypto`).
- Validate env at startup; fail fast with clear error messages.
- For AWS, keep `template.yaml` lean and avoid oversized Layers; for CF Workers, keep bundle small.
- For Bun/Deno, prefer Web APIs; avoid Node-only modules to keep edge compatibility.

Deployment steps (example: Vercel Edge)

```
// vite.config.ts
import { defineConfig } from 'vite';
import philjs from 'philjs-cli/vite';
import vercel from 'philjs-adapters/vercel';

export default defineConfig({
  plugins: [philjs(), vercel({ edge: true, isr: { expiration: 60 } })]
});

pnpm build
vercel deploy --prod
```

AWS/Lambda specifics

- Prefer HTTP API over REST for latency and cost.
- Keep cold starts low: small bundles, few dependencies, no large Layers.
- Use `packages/philjs-adapters/.aws/template.yaml` as a starting point.
- Set memory/timeouts per route; separate heavy functions from lightweight ones.

Bun/Deno edge specifics

- Ship pure ESM; rely on built-in `fetch`, `crypto`, and Web Streams.
- Keep bundles tiny; Bun and Deno reward minimal polyfills.
- Use `philjs-adapters/bun` and `philjs-adapters/deno` to wire SSR without Node shims.

Observability

- Emit logs per request and include trace ids.
- Export basic metrics (TTFB, render time, cache hits/misses).
- Capture unhandled errors and surface in platform logs (Vercel/Netlify dashboards, Cloudflare Workers logs).
- Forward platform request ids (e.g., `x-vercel-id`) into your logs for cross-correlation.

Try it now: Cloudflare Pages with KV and ISR-like revalidate

1. Configure adapter:

```

import cloudflarePages from 'philjs-adapters/cloudflare-pages';

cloudflarePages({
  kv: [{ binding: 'CACHE', id: 'my-kv' }],
  db: [{ binding: 'DB', database_id: 'my-db' }],
  revalidate: 60
});

```

2. In a loader, tag caches and set revalidate hints:

```

export const postLoader = loader(async ({ params, cache }) => {
  const post = await getPost(params.slug);
  cache.tag(['post', params.slug]);
  cache.revalidate(60);
  return { post };
});

```

3. Deploy with wrangler pages deploy and verify TTFB + cache headers via curl -I.

Desktop: Tauri and Native Shells

Build PhilJS apps for desktop with Tauri (or Electron if required). This chapter covers project setup, file access, native menus, and performance considerations.

Why Tauri

- Tiny binaries, Rust-backed security, and fast startup.
- Direct filesystem access with scoped permissions.
- Native menus/notifications; system tray support.

Project setup

1. Create a PhilJS app (Vite).

2. Add Tauri:

```

pnpm add -D @tauri-apps/cli
pnpm tauri init

```

3. Point Tauri distDir to your PhilJS build output.

4. Configure tauri.conf.json for CSP, allowlists, and bundle settings.

File access

- Use Tauri APIs for filesystem; scope paths in tauri.conf.json.
- Avoid bundling secrets; read from env or secure storage.
- For large files, stream reads/writes; keep UI responsive via signals.

Native shell features

- Menus/shortcuts: map to intents and update UI state via signals/stores.
- Notifications: use sparingly; respect user preferences.
- Auto-update: wire Tauri updater; guard with feature flags.

Security

- Restrict allowlist; disable the default open if not needed.
- Content-Security-Policy: disallow remote code unless required.
- Validate all IPC; never trust unvalidated payloads.

Performance

- Keep bundle small; desktop doesn't excuse bloat.
- Use windowing options to defer heavy windows until requested.
- For GPU-heavy tasks, consider WebGPU/WASM modules (see platforms/webgpu + platforms/wasm).

Testing

- Unit/integration as usual with PhilJS testing.
- E2E: Playwright with Tauri driver or Spectron-equivalent; cover menus and file ops.

Checklist

- Tauri config locked down (allowlist, CSP, updater).
- File access scoped and validated.
- Menus/shortcuts wired to intents; no UI-only logic.
- Bundle size audited; heavy deps lazy-loaded.

E2E smoke covers launch, file open/save, and updates.

Progressive Web Apps (PWA)

Turn PhilJS apps into installable, offline-capable PWAs.

Core setup

- Use `philjs-plugin-pwa` (or Vite PWA plugin) to generate service workers and manifests.
- Add `manifest.webmanifest` with icons/splash screens.
- Configure service worker strategies: app shell precache, network-first for APIs.

Service worker strategies

- **App shell:** precache static assets and core routes.
- **Data:** network-first with fallback to cache for read-heavy endpoints.
- **Images:** cache-first with expirations.
- **Background sync:** queue mutations and retry when online (where supported).

Installability

- Serve over HTTPS.
- Provide 192x192 and 512x512 icons; theme color and background color set.
- Prompt install thoughtfully (not on first visit); respect user choice.

Offline UX

- Show offline banner and cached content.
- Persist drafts (forms/notes) to IndexedDB; replay on reconnect.
- Provide retry and conflict resolution flows.

Updates

- Notify users when a new service worker is ready; let them refresh when convenient.
- Avoid breaking changes in cached shells; version assets and bust caches correctly.

Testing PWAs

- Lighthouse PWA audit.
- Playwright offline mode: verify shell loads, cached routes render, drafts persist.
- Test update flow (new SW) to ensure users see refreshed content.

Security

- Restrict external resource loading; set CSP.
- Validate data stored offline; encrypt sensitive items.

Checklist

- Manifest + icons present.
- Service worker caches shell/assets and handles data sensibly.
- Offline banner + draft persistence.
- Install prompt timing controlled.
- Update flow tested.

WebGPU and High-Performance Rendering

Use WebGPU for advanced graphics and compute workloads. PhilJS can orchestrate UI + control flows while WebGPU handles heavy rendering.

When to use WebGPU

- 3D scenes, data viz beyond Canvas2D/WebGL.
- GPGPU compute (image processing, ML inference) when WASM alone isn't enough.
- Offloading CPU-heavy work to the GPU.

Setup

- Ensure browser support; feature-detect and provide fallbacks.
- Use `@philjs/webgpu` helpers (if available) or raw WebGPU API.
- Keep shaders in dedicated `.wgs1` files and load asynchronously.

Basic pipeline (sketch)

```
const adapter = await navigator.gpu.requestAdapter();
const device = await adapter.requestDevice();
const context = canvas.getContext('webgpu');
context.configure({ device, format: navigator.gpu.getPreferredCanvasFormat() });
```

Use PhilJS signals to manage UI state (camera params, toggles) and pass them into uniform buffers.

Performance tips

- Reuse pipelines; avoid recreating per frame.
- Double-buffer uniforms; batch updates.
- Avoid large JS<->GPU copies; keep data on GPU as long as possible.
- Throttle renders when hidden; use `requestAnimationFrame` wisely.

SSR/Islands with WebGPU

- Render UI server-side; hydrate a small island that owns the canvas.
- Lazy-load WebGPU code on interaction/visibility.
- Provide a “fallback image” or WebGL/Canvas2D version for unsupported devices.

Testing

- Feature-detect in tests; mock WebGPU interfaces where needed.
- Visual diff key frames with Playwright screenshots for critical scenes.
- Benchmark compute shaders with small, deterministic inputs.

Safety and compatibility

- Guard for unsupported browsers; show a friendly message and fallbacks.
- Keep shaders deterministic; validate inputs before sending to GPU.
- For privacy/security, avoid exposing precise timing data unnecessarily.

WebAssembly (WASM)

Use WASM to speed up CPU-heavy tasks (parsing, math, image/voice processing) while PhilJS handles UI and orchestration.

When to reach for WASM

- Hot loops where JS perf isn’t enough.
- Reusing existing Rust/Go/C++ logic.
- Deterministic compute (e.g., codecs, parsers).

Build and load

- Compile with target `wasm32-unknown-unknown` (Rust) or Emscripten/Go as needed.
- Use `import` with `WebAssembly.instantiateStreaming` where supported.
- Keep WASM modules small; split features if needed.

Passing data

- Use `Uint8Array/Float32Array` for buffers.
- Avoid round-tripping large blobs; batch work per call.
- For strings, standardize encoding (UTF-8) and helpers.

Threading and SIMD

- Enable WASM threads (`SharedArrayBuffer`) where allowed; check COOP/COEP headers.
- Use SIMD flags for vectorized workloads if supported by target browsers.

SSR and islands

- Avoid executing WASM during SSR; load in islands after hydration or on interaction.
- For edge runtimes, ensure WASM bundle is small and supported by the platform.

Testing

- Unit-test WASM functions with deterministic inputs/outputs.
- Benchmark critical functions and compare to JS fallbacks.
- In Playwright, ensure WASM loads and runs on target browsers; provide fallbacks when blocked.

Security

- Validate untrusted inputs before passing to WASM.
- Keep WASM modules signed/versioned; avoid dynamic code fetch from untrusted origins.
- Beware of large memory allocations; set sane limits and monitor in performance tools.

Native Bridges (Android/iOS)

Bridge PhilJS apps to native capabilities when a pure web approach isn't enough.

When to use a native bridge

- Deep platform integrations: background services, sensors beyond Web APIs.
- Performance-sensitive features that need native modules.
- Integrating with existing native apps via webviews.

Options

- Capacitor plugins (preferred with PhilJS native template).
- Custom native modules exposed to webviews (Android: JSInterface; iOS: WKScriptMessageHandler).
- React Native/NativeScript bridges if mixing with native stacks (heavier).

Patterns

- Keep bridge APIs small and typed; version them.
- Validate inputs/outputs at the boundary.
- Avoid chatty bridges; batch requests or stream events.
- Handle permission prompts natively; reflect state in PhilJS UI via signals.

Security

- Restrict which origins can call the bridge.
- Validate messages; never eval arbitrary JS.
- For Android, disable addJavascriptInterface on untrusted content; prefer modern APIs.

Testing

- Unit-test bridge wrappers in PhilJS with mocks.
- Device/instrumentation tests for native modules.
- E2E: drive webview + native interactions (Appium/Playwright for webview contexts).

Checklist

- Bridge surface area minimal and typed.
- Permissions handled explicitly with clear UX.
- Inputs/outputs validated; no arbitrary eval.
- Tests cover bridge calls and failure modes.

Streaming and Progressive Rendering

Streaming reduces TTFB and keeps users engaged while data loads. PhilJS supports streaming both on the server (SSR) and client (resources + Suspense-like patterns). Use this with the performance and SSR chapters to build responsive dashboards and long-running flows.

When to stream

- Above-the-fold content should ship immediately; stream secondary panels and dashboards.
- Long-running loaders (reports, analytics) should yield partial HTML and continue as data arrives.
- Use for chat feeds, logs, and incremental search results.
- Use for personalization: stream common shell, then personalized panels.

Server-side streaming

```
import { renderToStream } from '@philjs/ssr';

export default async function handler(req) {
  const stream = await renderToStream(<App />);
  return new Response(stream, {
    headers: { 'content-type': 'text/html; charset=utf-8' }
  });
}
```

Tips:

- Keep initial shell small; inline critical CSS or use streaming-friendly styles.
- Flush early with renderToStream to improve TTFB.

- Avoid blocking promises in root layout; push them into resources.
- Combine with edge adapters to reduce round-trip latency further.

Client-side progressive data

Use resources to represent async data and show fallbacks:

```
const user = resource(() => fetch('/api/me').then(r => r.json()));

<Show when={user()}>
  {(data) => <Profile user={data} />}
</Show>
<Show when={!user()}>
  <SkeletonProfile />
</Show>
```

Error and retry

- Wrap streamed sections in error boundaries; render friendly fallbacks.
- Provide retry buttons for individual panels; avoid full page reloads.
- Log streaming errors to observability pipeline (see observability chapter).
- Consider backoff on repeated failures; avoid hot loops.

Performance and SEO

- Streaming improves TTFB and FID; keep head tags stable to avoid cumulative layout shift.
- For SEO, ensure critical meta tags are present in the first chunk.
- Use `<link rel="preload">` for critical CSS/fonts in the first flush.

Testing streaming

- Use integration tests (Playwright) to assert first paint speed and fallback presence.
- In Vitest + jsdom, simulate slow fetches to ensure fallbacks render before data arrives.
- Add smoke tests that capture HTML chunks to verify critical meta and shell render in chunk 1.

Try it now: streamed dashboard panel

```
import { resource } from '@philjs/core';
import { renderAsStream } from '@philjs/ssr';

const slowStats = resource(() =>
  fetch('/api/stats?slow=1').then(r => r.json())
);

function Dashboard() {
  return (
    <Layout>
      <Hero />
      <Show when={slowStats()} fallback={<Skeleton />}>
        {(stats) => <StatsPanel stats={stats} />}
      </Show>
    </Layout>
  );
}

export default async function handler() {
  return new Response(await renderAsStream(<Dashboard />), {
    headers: { 'content-type': 'text/html; charset=utf-8' }
  });
}
```

Test with slow network throttling and confirm the hero renders immediately while stats fill in later.

Offline and Resilient UX

Build experiences that keep working without a network and recover gracefully when connectivity returns.

Core principles

- Local-first: write locally, sync later.
- Predictable caches: explicit lifetimes, no silent staleness.
- Clear status: show offline/online banners and sync state.

Data strategy

- Use loaders with cache tags and `staleTime`; pair with `@philjs/offline` for IndexedDB storage.
- Queue mutations when offline; replay on reconnect with backoff.
- Detect conflicts and show resolution UI (keep both / pick latest / merge).

Persistence

- Persist drafts (forms, documents) to IndexedDB.
- Encrypt sensitive payloads; avoid storing secrets.
- Keep history bounded to prevent unbounded growth.

UI patterns

- Banner/toast when offline; avoid modal blocks.
- Disable actions that truly require network; allow drafts otherwise.
- Show sync progress and errors; let users retry.

Service workers/PWA

- Use `philjs-plugin-pwa` for caching strategies (app shell, assets, API fallback).
- Precache core routes; use network-first for dynamic data.
- Background sync where supported; otherwise retry on focus/reconnect.

Testing

- Simulate offline in Playwright; ensure pages render from cache and drafts persist.
- Unit-test mutation queue logic with fake timers and reconnect events.
- Ensure conflict handling paths are covered with fixtures.

Checklist

- Offline banner + status.
- Draft persistence for key flows.
- Mutation queue with retry/backoff.
- Conflict resolution path.
- PWA/service worker configured for core routes and assets.

Accessibility Patterns

Ship accessible experiences by default and catch regressions early.

Principles

- Semantics first: use native elements and correct roles.
- Labels everywhere: form controls, buttons, inputs.
- Keyboard-first: everything reachable via Tab/Shift+Tab; Escape closes overlays.
- Respect user preferences: reduced motion, contrast, font size.

Forms

- Use `<label for>` with matching ids.
- Add `aria-invalid` and `aria-describedby` for errors.
- Group related fields with `<fieldset>` and `<legend>`.

Dialogs and menus

- Trap focus inside dialogs; restore on close.
- Close on Escape and outside click; keep state in signals/stores.
- Menus: roving tab index, arrow key navigation, `aria-expanded` on trigger.

Live regions

- Announce async events (toast, form errors) with `role="status"` or `role="alert"`.
- Throttle announcements to avoid spam.

Motion and contrast

- Honor `prefers-reduced-motion`: disable heavy animations.
- Ensure color contrast meets WCAG AA; bake into design tokens.

Testing

- Use PhilJS testing with role/label queries.
- Add axe checks to key pages in CI (sparingly to keep noise low).
- Playwright keyboard navigation tests for dialogs/menus/forms.

Checklist

- Labels and roles on interactive elements.
- Focus management for overlays.
- Keyboard nav for menus/lists.
- Reduced-motion respected.
- Contrast verified for tokens and components.
- Live regions for async feedback.

Internationalization (i18n) Patterns

Localize PhilJS apps with predictable fallbacks and performance in mind.

Principles

- Extract all user-facing strings; avoid concatenation that breaks grammar.
- Keep locale data lean; lazy-load per locale/namespace.
- Provide sane fallbacks and error handling for missing translations.

Setup

- Store messages in JSON per locale/namespace.
- Use `@philjs/i18n` (if available) or a lightweight library; avoid bloated runtimes.
- Detect locale from URL, headers, or user settings; persist choice.

Usage

```
import { t } from '@philjs/i18n';

function Welcome() {
  return <h1>{t('welcome.title', 'Welcome')}</h1>;
}
```

Loading strategy

- Load base locale on first render; lazy-load additional namespaces per route.
- Cache translations per locale; invalidate on deploy/version bump.
- For SSR, prefetch required namespaces and serialize them.

Plurals and formatting

- Use ICU/messageformat for plurals, gender, and select cases.
- Handle dates/numbers with Intl APIs; memoize formatters per locale.

Forms and validation

- Localize validation messages; keep schemas and messages together.
- Respect locale for number/date parsing.

Testing

- Snapshot critical pages in multiple locales for layout regressions (rtl/ltr).
- Unit-test formatter outputs.
- Ensure missing keys surface warnings in dev; have fallbacks in prod.

Performance tips

- Avoid shipping all locales; tree-shake unused ones.
- Split locale bundles by route/namespace.
- Use streaming to send base UI fast, then hydrate localized pieces if needed.

Checklist

- Locale detection + persistence.
- Lazy-loaded namespaces per route.
- Pluralization and formatting handled via Intl/ICU.
- RTL support tested.
- Missing key warnings in dev; fallbacks in prod.

Forms UX Patterns

Design forms that are fast, clear, and resilient.

Clarity and flow

- Single-column layouts; logical grouping with headings.
- Show progress for multi-step flows; allow back/forward without losing data.
- Keep labels always visible; avoid placeholder-only labels.

Validation UX

- Validate on blur or submit; avoid noisy per-keystroke errors.
- Show inline errors near fields; summary at top with anchors for accessibility.
- Offer examples (e.g., email format) and constraints (password rules) before submission.

Performance and responsiveness

- Debounce expensive validation (e.g., username availability).
- Optimistic UI for non-critical saves; autosave drafts.
- Prefill known data; cache form state between steps/pages.

Accessibility

- Proper labels, descriptions, and `aria-*` attributes.
- Keyboard-friendly: tab order logical, Enter submits, Escape cancels modals.
- High contrast and visible focus states.

File uploads

- Show file size/type constraints early.
- Display progress, success/failure; allow cancel/retry.
- Handle large files with chunking where necessary.

Errors and recovery

- Keep data on error; never clear the form.
- Offer retry and contact/support options when server fails.
- For partial saves, show what succeeded and what did not.

Testing

- Component tests for validation messages and disabled states.
- Integration tests for server errors and retries (MSW).
- E2E for multi-step flows, back/forward navigation, and drafts.

Checklist

- Labels + descriptions always visible.
- Validation timed sensibly; helpful messages.
- Autosave/drafts for longer flows.
- File uploads with progress and retry.
- Errors keep data and offer recovery.

Feature Flags and Experiments

Control rollouts and experiments safely with PhilJS.

Goals

- Ship incrementally; reduce blast radius.
- Run A/B or multivariate tests with clear assignment and analytics.
- Kill-switch risky features instantly.

Implementation

- Central flag provider (server-driven or config).
- Evaluate flags in loaders/actions; pass to components as props/signals.
- For experiments, assign variants server-side; persist in cookies/local storage with expiry.

Data and cache impact

- Tag caches with flag/variant to avoid cross-contamination.
- Avoid caching HTML for user-specific flag combinations unless scoped.

Observability

- Log flag/variant values with events and errors.
- Analyze impact on performance and business metrics.

Testing

- Unit: flag evaluation logic with overrides.
- Integration: simulate variants and assert UI changes.
- E2E: ensure consistent variant across navigation/session.

Checklist

- Flag evaluation centralized; defaults defined.
- Variants persisted and consistent per user/session.
- Caches scoped to flags/variants as needed.
- Metrics/analytics include flag/variant dimensions.

Analytics and Instrumentation

Measure what matters while respecting performance and privacy.

Principles

- Minimal, purposeful events; avoid noisy firehose.
- Sample where appropriate to limit cost and overhead.
- Respect user privacy and consent; avoid sensitive data.

Events to capture

- Page/route views with route id and cache status.
- Key actions (sign-ups, conversions, feature use).
- Performance milestones (TTFB, hydration, LCP) if allowed by your APM.
- Experiment/feature flag assignments.

Implementation

- Central analytics client; queue events and batch send.
- Ensure edge-safe API usage if sending from server/SSR.
- Avoid blocking navigation; sendBeacon/fetch with timeout.

Privacy and compliance

- Honor consent (GDPR/CCPA); allow opt-out.
- Redact PII; avoid raw user content in events.
- Short retention for sensitive metrics; anonymize where possible.

Testing

- Unit-test event builders for correct shape and redaction.
- Integration: assert key events fire on actions in jsdom/Playwright.
- Verify sampling works: e.g., 10% sample for heavy events.

Checklist

- Event schema defined and versioned.
- Batching + backoff implemented; non-blocking.
- Consent/opt-out respected; PII redacted.
- Events include route/flag/variant context where relevant.

Architecture Decision Records (ADRs)

Track key technical decisions for PhilJS projects.

Why ADRs

- Capture context, options, and consequences.
- Onboard new team members quickly.
- Avoid re-litigating past decisions.

Template (suggested)

```

ADR-00X: Title
Date: YYYY-MM-DD
Status: Proposed | Accepted | Superseded

Context:
- Background and constraints.

Decision:
- Chosen approach.

Consequences:
- Positive/negative trade-offs.

Alternatives:
- Considered and why rejected.

```

What to record

- Routing/data strategy (loaders/actions vs client-only).
- SSR/islands strategy per surface.
- State management choices (signals/stores/resources boundaries).
- Caching strategy (tags, stale times, revalidate).
- Security/auth approach.
- Observability stack (APM/logs/metrics).
- Deployment targets (edge vs regional vs static).

Practices

- Keep ADRs in repo (docs/adr/); link from README.
- Reference ADRs in PRs when touching related code.
- Supersede old ADRs when direction changes; keep history.

Checklist

- ADR created for significant decisions.
- Status updated; superseded ADRs linked.
- Linked from relevant docs/code (e.g., router setup, caching config).

CMS and Content Workflows

Integrate PhilJS with headless CMSs while keeping performance and editorial velocity high.

Choosing a CMS

- Headless options (Contentful, Sanity, Strapi, Ghost) for structured content.
- Consider editorial UX, API performance, preview support, and webhook capabilities.

Data flow

- Fetch content via loaders; cache by slug and locale.
- Use incremental/static generation for stable pages; revalidate on webhook.
- For previews, bypass cache and hit draft endpoints; protect with auth tokens.

Modeling content

- Keep content schemas stable; version fields for breaking changes.
- Use references/blocks for rich pages; keep components in PhilJS aligned with CMS blocks.
- Validate CMS data in loaders; handle missing fields gracefully.

Images and media

- Use CMS image CDN transforms for responsive sizes.
- Cache media URLs; set proper Cache-Control headers.
- Lazy-load below-the-fold media.

Webhooks and revalidation

- On publish/update, trigger cache invalidation for affected routes.
- For ISR-like setups, call revalidate endpoints; ensure authentication on webhooks.

Localization

- Store locales per entry; use hreflang in rendered pages.
- Fallback strategy for missing translations.

Testing

- Stub CMS API responses with MSW; include draft/published variants.
- E2E preview flows: ensure draft content renders and is not cached.
- Monitor 404s for missing slugs.

Checklist

- Loaders fetch and cache content by slug/locale.
- Preview flow protected; draft content not cached.
- Webhooks trigger precise invalidation.
- CMS data validated; fallbacks for missing fields.
- Responsive images via CMS CDN.

Static vs Live Content

Choose the right rendering and caching strategy for each content type.

Static content (docs, marketing)

- Pre-render (SSG) and cache aggressively (long TTL).
- Use ISR/edge cache for infrequent updates; trigger revalidate on CMS webhook.
- Keep payloads small; stream hero + critical CSS.

Semi-static content (blogs with comments, listings)

- Pre-render the shell; hydrate live sections (comments, counts) as islands.
- Cache base HTML; fetch live data client-side with caches and revalidate hints.
- Invalidate tags when new content is published or updated.

Live content (dashboards, chat, analytics)

- SSR for first paint; stream partials for slow data.
- Hydrate live panels; use WebSockets/SSE for updates.
- Aggressive caching for static parts; short staleTime or no HTML cache for live panels.

Mixed routes

- Split route into sections: hero/static, live widgets, user-specific panels.
- Tag caches per section; hydrate only what needs interactivity.
- Use prefetch and background revalidate for near-real-time freshness.

Testing

- For static routes, assert cache headers and no unexpected network calls.
- For live routes, simulate slow data and ensure fallbacks render quickly.
- Verify revalidate/webhook flows update pages correctly.

Checklist

- Identify content type per route.
- Set caching (SSG/ISR/edge) accordingly.
- Live sections hydrated as islands/resources.
- Webhooks or invalidation rules for updates.

SEO and Discoverability

Optimize PhilJS apps for search engines and social sharing without sacrificing performance.

Core practices

- Server-render critical pages; stream for speed.
- Unique, descriptive titles and meta descriptions per route.
- Canonical URLs to avoid duplicate content.
- Structured data (JSON-LD) for rich results when applicable.

Meta handling

- Set <title>, <meta name="description">, and canonical links in layouts.
- Add Open Graph/Twitter tags for social cards.
- Avoid duplicate titles; include brand suffix/prefix consistently.

Routing and links

- Use clean URLs; avoid hash routing.
- Ensure internal links are crawlable; avoid JS-only navigation for critical content.
- Provide sitemaps and robots.txt; update on deploy.

Performance and crawl budget

- Keep TTFB low with edge SSR; stream content.
- Defer non-critical scripts; minimize JS for landing pages.
- Use lazy loading for below-the-fold media; supply `loading="lazy"` and `width/height` to avoid CLS.

Internationalization

- Use `hreflang` for localized pages.
- Ensure language-specific content has distinct URLs.

Accessibility overlap

- Semantic HTML improves SEO; headings in order, alt text for images.
- Avoid hiding primary content behind interactions for bots.

Testing

- Run Lighthouse SEO checks.
- Validate structured data with Google's Rich Results Test.
- Inspect rendered HTML (SSR output) to ensure meta tags are present before hydration.

Checklist

- Titles/descriptions set per route.
- Canonical tags present.
- OG/Twitter cards configured.
- Sitemap/robots generated on deploy.
- Structured data added where relevant.
- Performance budgets met on landing pages.

Charts, Graphics, and Diagrams

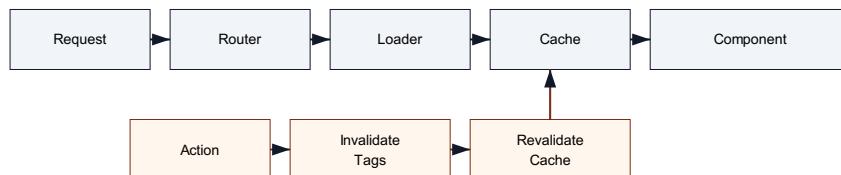
This chapter provides the visuals referenced throughout the book. Use them as anchors when reviewing routing flow, SSR pipelines, state graphs, and performance metrics.

Architecture and routing



Data flows from router to loaders/actions, through caching, into SSR/islands, and finally hydration.

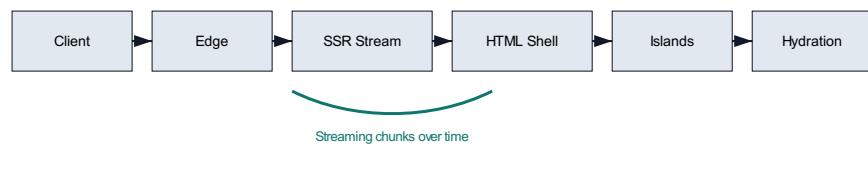
Architecture overview



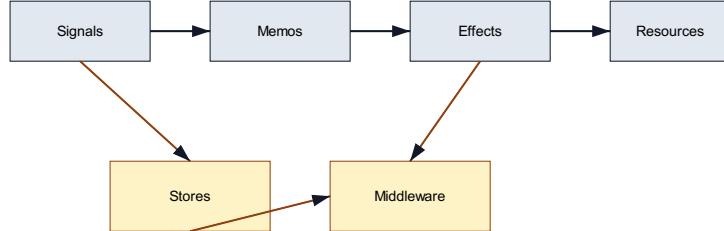
Top row: read path. Bottom row: mutation and cache invalidation.

Routing data flow

SSR and state

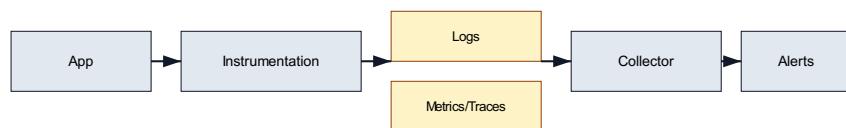


SSR streaming pipeline

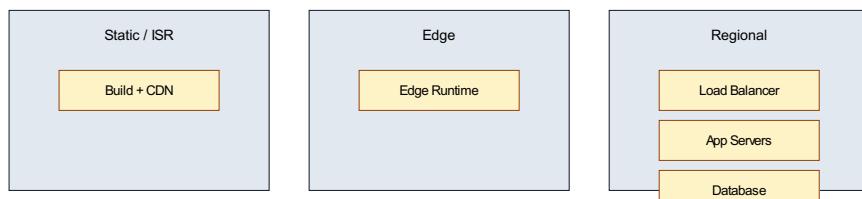


State graph

Observability and deployment

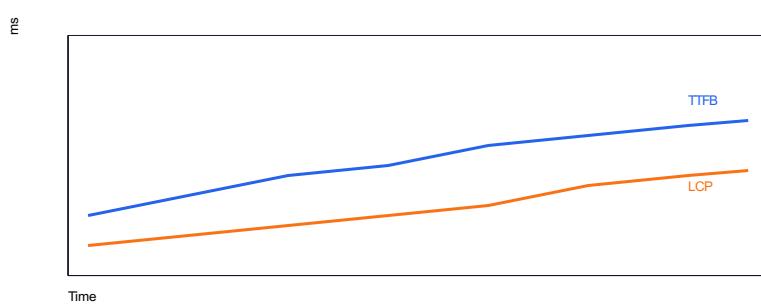


Observability pipeline



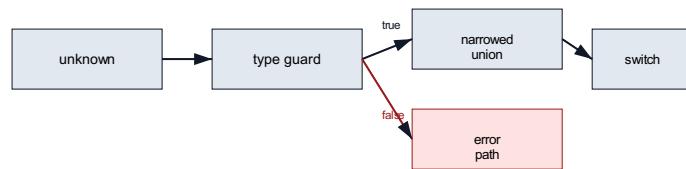
Deployment topologies

Performance charts

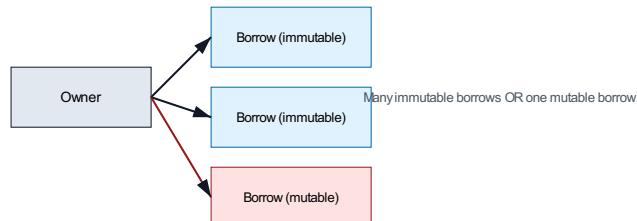


Performance metrics

Appendix visuals



Type narrowing flow



Rust ownership

Notes

- All visuals are SVG for clarity in PDF and EPUB exports.
- Keep alt text in place for accessibility and searchability.
- Update diagrams in `docs/philjs-book/visuals` if any workflows change.

Local-First and GenUI

Nexus architecture combines local-first data with generative UI. PhilJS is built to support both without sacrificing performance or control.

Local-first foundation

- Treat the client as the source of truth.
- Use embedded storage (SQLite, CRDTs) and sync engines.
- Render from local data, not round-trips.

Generative UI readiness

- UI can be composed dynamically from a registry of components.
- Agent output must be validated against schemas.
- Interactive components must be allow-listed.

PhilJS alignment

- Signals for fine-grained updates
- Web Components for interoperable primitives
- SSR + islands to keep payloads small

Nexus checklist

- Local data reads are synchronous and reactive
- AI-generated layouts are schema-validated
- Security boundaries prevent UI injection
- Accessibility is enforced at render time

Nexus, GenUI, and AI-assisted Flows

Nexus is PhilJS's opinionated architecture for local-first, AI-assisted applications. GenUI sits on top, composing intent capture, guarded AI actions, and collaborative state. Lean on this plus `docs/nexus` and `docs/ai` to keep safety and UX aligned.

Principles

- **Local-first:** optimistic updates with CRDT-friendly data shapes; sync later.
- **Intent over clicks:** capture user intent explicitly, feed it to AI safely.
- **Guardrails:** validate AI outputs against schemas and policies before applying.
- **Collaboration:** multi-user presence and conflict resolution baked in.

- **Cost-aware:** track token usage and cap spend per user/workflow.

Capturing intent

Use `@philjs/intent` to declare intents with types and policies:

```
import { defineIntent } from '@philjs/intent';

export const createDoc = defineIntent({
  name: 'create-doc',
  input: z.object({ title: z.string().min(1) }),
  policy: ({ user }) => user.role === 'editor'
});
```

Bind to UI:

```
const intent = useIntent(createDoc);
intent.submit({ title });
```

Enrich intent with context (feature flags, locale, user role) to guide AI behavior and policy checks.

AI actions with guardrails

- Call LLMs via `@philjs/ai` with typed prompts and expected JSON schemas.
- Validate outputs; reject or repair before mutating state.
- Log prompts/responses for audit (redact user secrets).
- Add safety filters (toxicity/PII) and enforce deterministic output with JSON modes where possible.
- Keep temperature low for deterministic workflows; higher for ideation surfaces only.

Collaboration and presence

- Use `@philjs/collab` for presence, cursors, and live cursors in editors.
- Sync via WebSockets or SharedWorkers; merge changes with CRDTs where possible.
- Show conflict indicators; prefer intent replay over last-write-wins.
- For large docs, chunk state and sync per chunk to avoid bandwidth spikes.

Offline/rehydration

- Persist draft intents locally; replay when back online.
- Use resumability in SSR to avoid redoing AI calls on hydration.
- Keep AI caches separate from user data; drop stale AI suggestions aggressively.
- Show "AI stale" indicators when cached suggestions exceed freshness windows.

Testing and safety

- Unit-test intent policies and AI validators.
- Add integration tests that simulate AI failures and ensure UI degrades gracefully.
- Rate-limit AI calls; enforce spend budgets server-side.
- Fuzz-test prompt outputs against schemas to catch drift after model updates.
- Add red-team tests for prompt injection and ensure sanitization holds.

Observability for AI flows

- Emit metrics: prompt latency, token usage, success/failure rates.
- Trace intent → AI call → state mutation to debug regressions.
- Add user-facing "why" strings for AI decisions to build trust.
- Log model version and feature flags to correlate changes with behavior.

Try it now: guarded AI intent

```

import { defineIntent, useIntent } from '@philjs/intent';
import { callAI } from '@philjs/ai';
import { z } from 'zod';

const summarize = defineIntent({
  name: 'summarize',
  input: z.object({ text: z.string().min(1) }),
  policy: ({ user }) => user.role !== 'banned'
});

const schema = z.object({ summary: z.string().max(500) });

function useSummarize() {
  const intent = useIntent(summarize);
  return async (text: string) => {
    const res = await callAI({
      prompt: `Summarize:\n${text}`,
      schema
    });
    schema.parse(res); // guardrail
    intent.submit({ text }); // Log the intent
    return res.summary;
  };
}

```

Add tests that mock `callAI` to return invalid shapes and assert they are rejected, and track token usage per request to enforce budgets.

Collaboration Patterns (Nexus)

Build real-time, multi-user experiences on top of Nexus and PhilJS.

Presence

- Use `@philjs/collab` or WebSockets to broadcast presence (online, active doc, cursor).
- Keep payloads small: user id, display name, color, position.
- Expire stale presence with heartbeats; drop disconnected users quickly.

CRDTs and conflict resolution

- Prefer CRDT-backed data structures for shared documents/boards.
- If using patches/logs, record intent and order; replay to reach consistency.
- Show conflicts explicitly (e.g., “other user editing this section”).

Editing models

- Text/blocks: CRDTs for high concurrency.
- Forms/dashboards: optimistic updates with conflict checks on commit.
- Media: lock or chunked uploads with coordination via server/worker.

Sync channels

- WebSockets for low latency; fallback to SSE/long-polling if needed.
- Batch updates and coalesce rapid events to reduce churn.
- Authenticate channels with short-lived tokens; scope access per document/tenant.

Offline + reconnect

- Queue edits locally; replay on reconnect with ordering preserved.
- Detect divergence and request full resync when conflicts are too large.
- Keep UI showing “synced/unsynced” state; allow manual retry.

Observability

- Log sync lag, drop/reconnect counts, and conflict rates.
- Trace from intent → network event → state merge to debug drift.

Testing

- Simulate multiple clients (in tests) with different event orders; assert convergence.
- E2E: run two browsers and verify presence/cursors and conflict handling.
- Fuzz merge logic with randomized patches.

Checklist

- Presence broadcast with heartbeats and expiry.
- Authenticated channels per doc/tenant.

- Conflict strategy defined (CRDT or explicit merge).
- Offline queue and replay tested.
- Metrics for lag, conflicts, reconnects.

Migrating from React

Move React apps to PhilJS with minimal friction and safer incremental steps.

Strategy

- Start with leaf components; migrate routes/features gradually.
- Keep routers side-by-side during transition (PhilJS Router + existing router) if needed.
- Replace `useState/useEffect` with signals/memos; avoid 1:1 rewrites of effect-heavy code.

JSX runtime

- Update `tsconfig.json` to `jsxImportSource: "@philjs/core"`.
- Swap React imports with PhilJS equivalents (`createContext -> signals/context helpers`).

State and effects

- Signals instead of `useState`.
- `memo` instead of derived state in `useMemo`.
- Replace `useEffect` data-fetching with loaders/resources; side effects stay in `effect`.
- Remove dependency arrays; PhilJS tracks dependencies automatically.

Components and props

- Functional components stay largely the same; drop React-specific hooks.
- Event handling is similar; avoid synthetic event assumptions.

Routing

- Map React Router routes to PhilJS Router.
- Move data fetching to loaders; mutations to actions.
- Use `Link` and `prefetch` for perf; add `errorBoundary` equivalents.

Context

- Prefer signals/stores over heavy context.
- When needed, use PhilJS context utilities; keep values serializable for SSR.

Porting hooks

- Many custom hooks become plain functions with signals.
- Derive state with memos; avoid effect-driven synchronization.

Styling and assets

- Reuse CSS/SCSS/Tailwind; adjust build plugins as needed.
- Replace React-specific styling libraries if they rely on React internals.

Testing

- Swap React Testing Library for `@philjs/testing`.
- Reuse MSW for network mocks; adapt render helpers.
- Keep Playwright E2E mostly unchanged.

SSR/Islands

- Move SSR entry to PhilJS SSR adapters.
- Split heavy components into islands for hydration control.

Checklist

- `tsconfig` updated (`jsxImportSource`).
- State/effects migrated to signals/memos/resources.
- Routes converted to PhilJS Router with loaders/actions.
- Tests updated to `@philjs/testing`.
- SSR/adapter configured; hydration verified.

Migrating from Solid/Svelte

Solid and Svelte already embrace reactivity; moving to PhilJS is mostly about routing, SSR, and API boundaries.

Mapping concepts

- Signals/Stores → PhilJS signals/stores (similar semantics).
- Derived values → `memo`.
- Side effects → `effect` (avoid `async` inside).
- Async data → `resource` (instead of Svelte load functions/Solid resources, align with loaders).

JSX and templates

- Solid users keep JSX; update `jsxImportSource: "@philjs/core"`.
- Svelte users rewrite templates to JSX; start with leaf components.

Routing

- Replace existing routers with PhilJS Router; move load/mutation logic to loaders/actions.
- Add `prefetch` and `cache` tags for data consistency.

SSR and islands

- Switch to PhilJS SSR adapters; stream by default.
- Hydrate only necessary islands; map Svelte/Solid hydration strategies to PhilJS equivalents.

Stores and context

- Solid signals map closely; Svelte stores map to PhilJS stores/resources.
- Avoid global writable stores; prefer scoped signals/stores per route.

Styling

- Reuse CSS/SCSS/Tailwind; adjust build plugins as needed.
- For Svelte scoped styles, translate to CSS modules/scoped styles in PhilJS.

Testing

- Swap to `@philjs/testing`; MSW remains for network mocks.
- Playwright E2E flows stay similar.

Checklist

- `jsxImportSource` updated (Solid).
- Templates ported to JSX (Svelte).
- Router + loaders/actions in place.
- SSR streaming/hydration verified.
- Tests migrated to PhilJS tooling.

Configuration

PhilJS projects are configured with `philjs.config.ts`.

```
import { defineConfig } from "philjs-cli";

export default defineConfig({
  routes: "./src/routes",
  publicDir: "./public",
  build: {
    outDir: "dist",
    ssg: false,
  },
  dev: {
    port: 3000,
    host: "localhost",
  },
});
```

Route modules

Each route can export a `config` object for caching or runtime hints.

```
export const config = {
  revalidate: 60,
  runtime: "edge",
};
```

TypeScript 6.x Deep Dive (Appendix)

A practical, in-depth TypeScript 6.x guide for PhilJS developers. Use this appendix as a language reference, a style guide, and a set of patterns you can apply directly in PhilJS apps.

Table of contents

- [TypeScript mental model](#)
- [Type operators and assignability](#)
- [Compiler configuration](#)
- [Types, unions, and intersections](#)
- [Enums and literal alternatives](#)
- [Control-flow narrowing](#)
- [Functions and overloads](#)
- [Generics and inference](#)
- [Object types and safety flags](#)
- [Tuples, arrays, and readonly data](#)
- [Classes, interfaces, and the type system](#)
- [Modules and the type/value split](#)
- [Declaration merging and module augmentation](#)
- [Utility, mapped, and conditional types](#)
- [Template literal types](#)
- [Async typing and error models](#)
- [JSX and PhilJS typing](#)
- [Interop with JavaScript](#)
- [Migration from JavaScript](#)
- [Testing types](#)
- [Build performance and project references](#)
- [Real-world patterns for PhilJS](#)
- [TS6 using and disposal](#)
- [Common pitfalls and fixes](#)
- [Checklists and exercises](#)

TypeScript mental model

TypeScript is a structural type system layered on top of JavaScript. The compiler checks your code at build time, then emits JavaScript without runtime type metadata.

Key ideas:

- Structural typing: if the shape matches, the types are compatible.
- Inference first: the compiler infers types and you constrain them where needed.
- Type/value split: types exist only at compile time; values exist at runtime.
- Practical safety: TypeScript favors ergonomics over strict soundness.

In PhilJS projects, strictness matters because you ship type-checked loaders, actions, and SSR boundaries. Treat the type system as an API contract.

Widening vs narrowing

TypeScript widens values unless you tell it not to:

```
const mode = 'edge';           // type: string (widened)
const mode2 = 'edge' as const; // type: 'edge'
```

Use `as const` or `satisfies` for config objects to keep literals narrow:

```
const env = {
  runtime: 'edge',
  cache: 'stale-while-revalidate'
} satisfies { runtime: 'edge' | 'node'; cache: string };
```

Type/value split

Types do not exist at runtime. If you need runtime checks, use functions or schemas.

```
type User = { id: string };

function isUser(value: unknown): value is User {
  return typeof value === 'object' && value !== null && 'id' in value;
}
```

Type operators and assignability

TypeScript is structural. Assignability is based on whether the required fields and signatures exist, not on nominal names.

```

type User = { id: string; email: string };
type MinimalUser = { id: string };

const u: User = { id: 'u1', email: 'a@b.com' };
const m: MinimalUser = u; // ok, u has at least the required shape

```

Common type operators:

- `keyof` produces a union of keys.
- `typeof` reads a value's type.
- indexed access (`T[K]`) reads a property type.

```

type UserKeys = keyof User;           // 'id' | 'email'
type UserEmail = User['email'];       // string

const config = { cache: 'stale', ttl: 30 };
type Config = typeof config;         // { cache: string; ttl: number }

```

Assignability gotchas:

- Function parameters follow variance rules; keep callback types explicit.
- Optional properties are not the same as `undefined` when `exactOptionalPropertyTypes` is enabled.

When in doubt, use `satisfies` to validate shape without widening:

```

const pipeline = [
  { name: 'lint', run: true },
  { name: 'typecheck', run: true }
] satisfies Array<{ name: string; run: boolean }>;

```

Compiler configuration

A good `tsconfig.json` is the foundation for a reliable codebase.

Recommended base for PhilJS apps:

```
{
  "compilerOptions": {
    "target": "ES2024",
    "module": "ESNext",
    "moduleResolution": "bundler",
    "jsx": "preserve",
    "jsxiImportSource": "@philjs/core",
    "strict": true,
    "noUncheckedIndexedAccess": true,
    "exactOptionalPropertyTypes": true,
    "verbatimModuleSyntax": true,
    "useDefineForClassFields": true,
    "isolatedModules": true,
    "isolatedDeclarations": true,
    "skipLibCheck": false,
    "types": ["@philjs/core"],
    "lib": ["ES2024", "DOM", "DOM.Iterable"]
  },
  "include": ["src"]
}
```

Config notes:

- `moduleResolution: bundler` matches modern toolchains and avoids CJS edge cases.
- `noUncheckedIndexedAccess` forces you to handle missing keys.
- `exactOptionalPropertyTypes` prevents `undefined` from silently slipping in.
- `isolatedModules` keeps you compatible with fast transpilers and Vite.
- `isolatedDeclarations` improves declaration emit for libraries.

For libraries:

- Set `declaration: true` and `declarationMap: true`.
- Emit to `dist` and keep `types` in `package.json` updated.

For apps:

- `noEmit: true` is fine because Vite handles emit.
- Keep `types` narrow to avoid slow editor performance.

Monorepo tips:

- Use a shared `tsconfig.base.json`.
- Add project references for faster incremental builds.
- Keep paths aligned with bundler aliases.

Strictness flags that matter

These flags prevent subtle runtime bugs:

- `strictNullChecks`: forces you to handle `null` and `undefined`.

- `noImplicitOverride`: ensures overrides are intentional.
- `noUncheckedIndexedAccess`: forces safe access for `obj[key]`.
- `exactOptionalPropertyTypes`: optional does not mean `undefined` unless declared.

Together, they prevent the common problems of “missing data” and “wrong shape” that show up most often in loaders, actions, and SSR payloads.

Types, unions, and intersections

TypeScript supports primitives, literals, unions, and intersections.

```
type Status = 'idle' | 'loading' | 'success' | 'error';

type ApiError = {
  code: 'UNAUTHORIZED' | 'NOT_FOUND' | 'RATE_LIMIT';
  message: string;
};

type Result<T> = { ok: true; value: T } | { ok: false; error: ApiError };
```

Intersections merge shapes:

```
type WithMeta = { createdAt: string; updatedAt: string };

type User = { id: string; email: string } & WithMeta;
```

Special types to know:

- `unknown`: safest top type. You must narrow before use.
- `never`: indicates an impossible value; great for exhaustive checks.
- `void`: for functions that return nothing.
- `any`: opt-out; avoid for public APIs.

Literal unions for state

```
type Theme = 'light' | 'dark' | 'system';
```

Literal unions make UI state explicit and easy to exhaustively handle.

Enums and literal alternatives

TypeScript enums exist at runtime, while literal unions are compile-time only. Prefer literal unions for most PhilJS code because they are tree-shakeable and align with serialized SSR data.

```
// Prefer this
type Env = 'edge' | 'node';

// Over this, unless you need runtime enum values
enum EnvEnum {
  Edge = 'edge',
  Node = 'node'
}
```

If you do use enums, avoid `const enum` in libraries unless your build pipeline inlines them consistently.

Control-flow narrowing

TypeScript narrows types based on control flow. This is essential for safe loaders, route params, and SSR payloads.

```
function formatUser(user: User | null) {
  if (!user) return 'Guest';
  return user.email.toLowerCase();
}
```

Discriminated unions:

```
type LoadState =
  | { status: 'idle' }
  | { status: 'loading' }
  | { status: 'success'; data: User }
  | { status: 'error'; error: ApiError };

function render(state: LoadState) {
  switch (state.status) {
    case 'idle':
      return 'Idle';
    case 'loading':
      return 'Loading';
    case 'success':
      return state.data.email;
    case 'error':
      return state.error.message;
  }
}
```

Other narrowing tools:

```

if ('id' in value) {
  // value is now narrowed to an object with id
}

if (value instanceof Error) {
  // value is Error
}

```

User-defined type guards:

```

function isUser(value: unknown): value is User {
  return typeof value === 'object' && value !== null && 'email' in value;
}

```

Assertion functions (use sparingly):

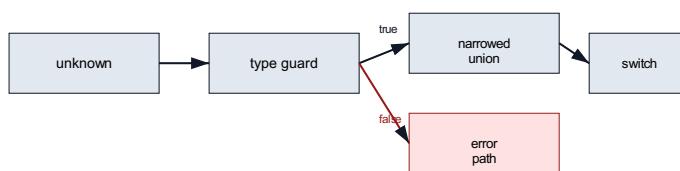
```

function assertUser(value: unknown): asserts value is User {
  if (!isUser(value)) throw new Error('Not a user');
}

```

Type assertions should be rare. Prefer narrowing via guards or schema validation.

Visual: Type narrowing flow



Type narrowing flow diagram

Functions and overloads

Function types include parameters, return type, and optional `this` context.

```

type Loader<T> = (params: { request: Request }) => Promise<T>;

```

Overloads should be minimal. Use a single signature with unions when possible.

```

function parse(input: string): number;
function parse(input: string, radix: number): number;
function parse(input: string, radix?: number) {
  return Number.parseInt(input, radix ?? 10);
}

```

If you need `this` typing:

```

function onClick(this: HTMLButtonElement, ev: MouseEvent) {
  this.disabled = true;
}

```

Avoid `Function` or `any` in public APIs. Keep return types explicit for loaders and actions.

Generics and inference

Generics allow you to write reusable, type-safe utilities.

```

function identity<T>(value: T): T {
  return value;
}

function pick<T, K extends keyof T>(obj: T, key: K): T[K] {
  return obj[key];
}

```

Useful patterns:

- Constraints: `<T extends { id: string }>`
- Defaults: `<T = string>`
- Inference helpers: `ReturnType`, `Parameters`, `Awaited`

Const type parameters (TS6) preserve literal inference:

```

function defineRoute<const T extends string>(path: T) {
  return { path } as const;
}

```

Use `satisfies` for configs to validate shape without widening:

```
const routes = [
  { path: '/', auth: false },
  { path: '/account', auth: true }
] satisfies Array<{ path: string; auth: boolean }>;
```

Indexed access and keyof

```
type UserEmail = User['email'];

function getField<T, K extends keyof T>(value: T, key: K): T[K] {
  return value[key];
}
```

Conditional inference patterns

Conditional types with `infer` let you extract parts of a type without duplicating it:

```
type LoaderReturn<T> = T extends (...args: any[]) => Promise<infer R> ? R : never;

type ParamsOf<T> = T extends (params: infer P) => any ? P : never;
```

This is useful for keeping component props aligned with loader return types. Use it to reduce duplication, but avoid overusing it when it makes the code harder to read. ## Object types and safety flags

Object types are the backbone of PhilJS state, props, and data contracts.

```
type User = {
  id: string;
  email: string;
  name?: string;
};
```

Important flags:

- `noUncheckedIndexedAccess`: `obj[key]` becomes `T | undefined`.
- `exactOptionalPropertyTypes`: optional props do not accept `undefined` unless specified.

Use index signatures sparingly. Prefer explicit property types.

```
type HeaderMap = Record<string, string>;
```

Readonly and tuples:

```
type Point = readonly [number, number];
const origin: Point = [0, 0];
```

Excess property checks prevent accidental typos:

```
const user: User = { id: 'u1', email: 'a@b.com', extra: true }; // error
```

Tuples, arrays, and readonly data

Tuples encode fixed-length arrays with typed positions. They are useful for route params, cache keys, and small coordinate pairs.

```
type CacheKey = readonly ['user', string];
const key: CacheKey = ['user', 'u1'];
```

Optional tuple elements:

```
type Range = [start: number, end?: number];
```

Readonly arrays help prevent accidental mutations in SSR payloads and shared caches:

```
type ReadonlyUsers = ReadonlyArray<User>;
```

Classes, interfaces, and the type system

Classes are optional in PhilJS, but they appear in adapters and external libraries.

```
interface Cache {
  get(key: string): string | null;
  set(key: string, value: string): void;
}

class MemoryCache implements Cache {
  private store = new Map<string, string>();
  get(key: string) { return this.store.get(key) ?? null; }
  set(key: string, value: string) { this.store.set(key, value); }
}
```

Prefer `private` fields for simple encapsulation. Use `#private` when you need runtime privacy.

Use `implements` to validate shape and `override` to ensure correct method signatures.

Modules and the type/value split

TypeScript distinguishes types from values. Use `type` imports for clarity and better tree-shaking.

```
import type { User } from './types';
import { fetchUser } from './api';
```

verbatimModuleSyntax forces you to be explicit about value vs type imports and aligns with ESM tooling.

Common ESM patterns:

- export type { User } from './types';
- export { fetchUser } from './api';
- Avoid export = or namespace in modern projects.

When building libraries, set "types" in package.json and keep ESM/CJS exports consistent.

Module resolution and exports

Modern packages should declare exports and types in package.json so TypeScript and bundlers agree:

```
{
  "name": "@philjs/core",
  "type": "module",
  "exports": {
    ".": {
      "types": "./dist/index.d.ts",
      "import": "./dist/index.js"
    }
  },
  "types": "./dist/index.d.ts"
}
```

If you ship multiple entry points, keep a types file for each entry. Avoid mixed CJS/ESM unless you must support legacy tooling.

Declaration merging and module augmentation

TypeScript can merge declarations with the same name. This is useful for extending global types or augmenting third-party modules.

```
declare global {
  interface Window {
    __PHILJS_VERSION__: string;
  }
}
```

Module augmentation lets you extend existing packages:

```
declare module '@philjs/router' {
  export interface RouteMeta {
    requiresAuth?: boolean;
  }
}
```

Use augmentation sparingly and keep it close to the consuming code so the intent is clear.

Utility, mapped, and conditional types

Utility types help you reshape existing types safely.

```
type PartialUser = Partial<User>;
type RequiredUser = Required<User>;
type UserPreview = Pick<User, 'id' | 'email'>;
type WithoutName = Omit<User, 'name'>;
```

Mapped types:

```
type DeepReadonly<T> = {
  readonly [K in keyof T]: DeepReadonly<T[K]>;
};
```

Conditional types:

```
type ExtractId<T> = T extends { id: infer I } ? I : never;
```

Distribution happens over unions:

```
type ToArray<T> = T extends unknown ? T[] : never;
```

Type-level programming patterns

These patterns show up in real codebases:

```
type DeepPartial<T> = {
  [K in keyof T]?: T[K] extends object ? DeepPartial<T[K]> : T[K];
};

type XOR<A, B> =
  | (A & { [K in keyof B]?: never })
  | (B & { [K in keyof A]?: never });
```

Use these sparingly. When types become hard to read, prefer runtime validation and simpler static types.

Template literal types

Template literal types are powerful for routes and cache tags.

```
type RoutePath = `/users/${string}` | `/teams/${string}`;
type CacheTag = `user:${string}` | `team:${string}`;
```

Use them to encode invariants the compiler can enforce.

Async typing and error models

Prefer explicit result types for async boundaries.

```
type ApiResult<T> =
  { ok: true; value: T }
  { ok: false; error: ApiError };

async function fetchUser(id: string): Promise<ApiResult<User>> {
  // ...
}
```

Use `Promise.allSettled` for batch loaders with partial success:

```
const results = await Promise.allSettled(tasks);
results.forEach((r) => {
  if (r.status === 'fulfilled') {
    // r.value
  } else {
    // r.reason
  }
});
```

Avoid throwing raw errors across loader/action boundaries. Convert to typed results or normalized error shapes.

Error modeling patterns

Two common styles work well for PhilJS:

1. **Result unions** (best for predictable UI state)
2. **Thrown errors** at the boundary (best for infrastructure failures)

Example with error codes:

```
type ErrorCode = 'UNAUTHORIZED' | 'NOT_FOUND' | 'RATE_LIMIT';

type ApiError = { code: ErrorCode; message: string };
type ApiResult<T> = { ok: true; value: T } | { ok: false; error: ApiError };
```

Use `never` to enforce exhaustive switches:

```
function assertNever(x: never): never {
  throw new Error(`Unexpected: ${x}`);
}
```

For loaders that support cancellation, include `AbortSignal` in your types and pass it to `fetch`:

```
type LoaderContext = { request: Request; signal: AbortSignal };
```

JSX and PhilJS typing

JSX types drive props, children, and event handling.

```
import type { ComponentProps } from '@philjs/core';

type ButtonProps = ComponentProps<'button'> & {
  variant?: 'primary' | 'ghost';
};
```

Event typing:

```
const onInput = (ev: InputEvent & { target: HTMLInputElement }) => {
  const value = ev.target.value;
};
```

PhilJS loader and action types should be explicit and reusable:

```
type UserLoader = () => Promise<{ user: User } | { user: null }>;
```

If you build component libraries, publish `ComponentProps` helpers to keep type inference intact.

JSX namespace and intrinsic elements

JSX typing relies on the `JSX` namespace. In PhilJS, intrinsic elements map to DOM elements, and custom components are inferred from function signatures.

If you need to extend intrinsic props, use module augmentation:

```
declare namespace JSX {
  interface IntrinsicElements {
    'x-chip': { label: string };
  }
}
```

Prefer `ComponentProps<'button'>` for native props and `PropsWithChildren` for components that accept children.

DOM and event typing

DOM events are generic in TypeScript. Narrow the event target to get safe access to `value`, `checked`, or `files`:

```
const onSubmit = (ev: SubmitEvent & { target: HTMLFormElement }) => {
  ev.preventDefault();
  const form = new FormData(ev.target);
  const email = String(form.get('email') ?? '');
};

const onToggle = (ev: Event & { target: HTMLInputElement }) => {
  const enabled = ev.target.checked;
};
```

For keyboard and mouse events, prefer built-in DOM types like `KeyboardEvent` and `MouseEvent` to avoid any.

Interop with JavaScript

Use `unknown` at boundaries and validate with schemas.

```
function parseJson<T>(raw: string, validate: (v: unknown) => v is T): T {
  const data: unknown = JSON.parse(raw);
  if (!validate(data)) throw new Error('Invalid payload');
  return data;
}
```

For JS packages without types, add minimal `*.d.ts` files instead of using any everywhere.

```
declare module 'legacy-lib' {
  export function doThing(input: string): string;
}
```

For globals:

```
declare global {
  interface Window {
    __PHILJS_VERSION__: string;
  }
}
```

Migration from JavaScript

For existing JS projects, migrate incrementally:

- Start with `allowJs: true` and `checkJs: true` to get type checking without converting files.
- Convert leaf modules first (utility and domain logic), then move toward UI and routing.
- Replace `// @ts-ignore` with `// @ts-expect-error` once you understand the error.

You can also annotate JS with JSDoc:

```
// @ts-check
/** @typedef {{ id: string, email: string }} User */
/** @param {User} user */
function greet(user) {
  return `Hello ${user.email}`;
}
```

Use JSDoc to narrow types before you fully convert files to TS. ## Testing types

Type-level tests prevent API drift.

```
import { expectTypeOf } from 'vitest';

expectTypeOf<ReturnType<typeof loader>>().toMatchTypeOf<{ user: User }>();
```

You can also use `tsd` for dedicated `.test-d.ts` files.

Use `// @ts-expect-error` to document intentional errors in tests. Avoid `// @ts-ignore` when possible.

Debugging type errors

When the compiler reports a complex type error:

- Inline the type with `type X = ...` and hover in the editor.
- Break large unions into smaller named types.
- Use `as const` to avoid literal widening.
- Add intermediate variables to force inference. If an error mentions a massive inferred type, simplify the function signature with explicit generics or return types.

Build performance and project references

Large repos need fast builds:

- Enable `incremental` and `composite` for project references.
- Split huge type declarations into smaller modules.
- Avoid massive union types or `any`-like escape hatches.
- Use `skipLibCheck: true` only when performance is critical and libs are trusted.

Project reference example:

```
{
  "references": [
    { "path": "../packages/core" },
    { "path": "../packages/router" }
  ]
}
```

Use `tsc -b` for multi-project builds and `tsc --watch` for incremental dev feedback.

Editor performance tips:

- Keep types lists small to avoid loading unnecessary globals.
- Split large `*.d.ts` files into modules.
- Prefer `skipLibCheck` only when you trust dependencies and need faster CI.

Real-world patterns for PhilJS

Typed route params

```
type Params = { id: string };

function parseParams(params: Record<string, string | undefined>): Params {
  if (!params.id) throw new Error('Missing id');
  return { id: params.id };
}
```

Loader results as discriminated unions

```
type LoaderResult =
  | { status: 'ok'; user: User }
  | { status: 'not-found' };
```

Safe config validation

```
type AppConfig = { apiUrl: string; featureFlags: string[] };

function parseConfig(env: Record<string, string | undefined>): AppConfig {
  if (!env.API_URL) throw new Error('Missing API_URL');
  return { apiUrl: env.API_URL, featureFlags: (env.FLAGS ?? '').split(',') };
}
```

Schema-style validation

If you use a schema library, keep the inferred type close to the schema and export it for reuse:

```
// pseudo-code pattern
const UserSchema = {
  id: (v: unknown) => typeof v === 'string',
  email: (v: unknown) => typeof v === 'string'
};

type User = { id: string; email: string };
```

The goal is one source of truth: validate at runtime, type-check at compile time.

Branded IDs

```
type Brand<T, B> = T & { __brand: B };

type UserId = Brand<string, 'UserId'>;
```

Cache tags with template literals

```
type CacheTag = `user:${string}` | `project:${string}`;

function tagUser(id: string): CacheTag {
  return `user:${id}`;
}
```

Typed loaders and actions

Keep loader and action shapes explicit to avoid inference drift:

```
type LoaderData = { user: User | null; lastLogin?: string };

type Loader = (ctx: { request: Request }) => Promise<LoaderData>;
type Action = (ctx: { request: Request }) => Promise<{ ok: true } | { ok: false; error: ApiError }>;
```

If you expose loader data to components, export the type so component props stay aligned.

Store selectors with keyof

```
type StoreState = { theme: 'light' | 'dark'; locale: string };

function select<K extends keyof StoreState>(state: StoreState, key: K): StoreState[K] {
    return state[key];
}
```

TS6 using and disposal

TypeScript 6 adds using to model deterministic cleanup when types implement `Symbol.dispose` or `Symbol.asyncDispose`.

```
class Tempfile {
    constructor(private path: string) {}
    [Symbol.dispose]() {
        // cleanup
    }
}

using file = new Tempfile('/tmp/data');
```

Use using for resource cleanup in tooling, tests, or server utilities.

Common pitfalls and fixes

These issues show up frequently in PhilJS apps:

- **Widened literals:** a config field becomes `string` instead of `'edge'`. Fix with `as const` or `satisfies`.
- **any from JSON:** `JSON.parse` returns `any` by default. Wrap it in a parser that returns `unknown` and validates.
- **Optional vs undefined:** with `exactOptionalPropertyTypes`, `name?: string` does not accept `undefined`. Use `name?: string | undefined` if you need it.
- **Unchecked index access:** `obj[key]` becomes `T | undefined` with `noUncheckedIndexedAccess`. Handle missing keys explicitly.
- **Excess property checks:** inline object literals get checked more strictly. Assign to a typed variable first if needed.
- **Accidental as:** using assertions to silence the compiler can hide bugs. Prefer guards or schemas.
- **Non-serializable SSR data:** `Date`, `Map`, and `Set` do not serialize cleanly. Convert to strings or arrays.
- **Type/value confusion:** `import type` vs `import` matters with `verbatimModuleSyntax`. Use `import type` when you only need types.

Treat compiler errors as design feedback. If something is hard to type, it might be hard to reason about. ## Checklists and exercises

Checklist: safer types

- Use `unknown` at boundaries and narrow early.
- Prefer discriminated unions for state machines.
- Keep loader/action return types explicit.
- Avoid `any` in public surfaces.
- Use `satisfies` for config objects.
- Avoid `as` assertions in shared libraries.

Exercises

1. Create a union type for loader states and make an exhaustive switch.
2. Write a `defineRoute` helper using `const` type parameters.
3. Add `expectTypeOf` tests for a component that forwards props.
4. Build a branded ID type for `ProjectId` and use it in two functions.
5. Model an API error type and enforce it in a loader.
6. Add a type guard for `CacheTag` values and use it in invalidation helpers.

Links

- [TypeScript Handbook](#)
- [TSConfig Reference](#)
- [PhilJS Installation](#)

Rust 1.92 Quickstart (Appendix)

A concise on-ramp to Rust 1.92 for PhilJS developers. This appendix focuses on the minimum you need for tooling, small services, and WASM helpers. The deep Rust coverage lives in the separate Rust with PhilJS book.

Table of contents

- [Toolchain and Cargo](#)
- [Rust basics](#)
- [Traits and lifetimes](#)
- [Ownership and borrowing](#)
- [Error handling](#)
- [WASM interop](#)
- [Testing and linting](#)

- Practical crates
- Command quick reference

Toolchain and Cargo

Install Rust with rustup and pin to 1.92:

```
rustup default 1.92.0
rustup component add rustfmt clippy
```

Create a new project:

```
cargo new my_tool
cd my_tool
cargo run
```

Project layout:

```
my_tool/
|-- Cargo.toml
`-- src/
    '-- main.rs
```

Rust basics

Rust is strict about mutability and ownership.

```
fn main() {
    let name = String::from("Phil");
    let mut count = 0;
    count += 1;
    println!("{} {}", name, count);
}
```

Structs and enums:

```
struct User {
    id: String,
    email: String,
}

enum LoadState {
    Idle,
    Loading,
    Loaded(User),
    Error(String),
}
```

Pattern matching:

```
match state {
    LoadState::Loaded(user) => println!("{} {}", user.id, user.email),
    LoadState::Error(err) => eprintln!("{} {}", err),
    _ => {}
}
```

Traits and lifetimes

Traits describe shared behavior. Most Rust libraries expose traits so you can write generic functions.

```
trait Identified {
    fn id(&self) -> &str;
}

impl Identified for User {
    fn id(&self) -> &str { &self.id }
}
```

Lifetimes describe how long references are valid. In many cases they are elided automatically. When you return references, you may need an explicit lifetime:

```
fn first<'a>(items: &'a [String]) -> Option<&'a String> {
    items.first()
}
```

Ownership and borrowing

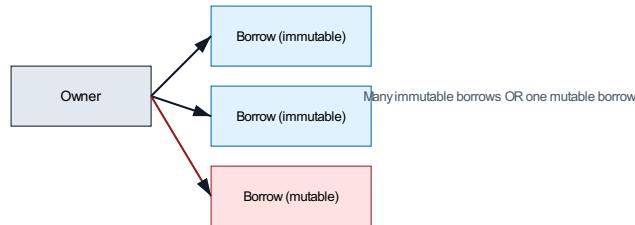
Rust enforces that each value has a single owner. References (`&T` or `&mut T`) borrow without taking ownership.

```
fn len(s: &String) -> usize {
    s.len()
}
```

If you need to mutate, borrow mutably:

```
fn push_exclamation(s: &mut String) {
    s.push('!');
}
```

Visual: Ownership and borrowing



Rust ownership diagram

Rules to remember:

- Many immutable borrows OR one mutable borrow at a time.
- A mutable borrow prevents all other borrows until it ends.
- Values are dropped when they go out of scope.

Error handling

Use `Result<T, E>` and the `? operator` for propagation.

```
use std::fs;

fn read_config(path: &str) -> Result<String, std::io::Error> {
    let content = fs::read_to_string(path)?;
    Ok(content)
}
```

Use `thiserror` for structured errors in libraries, and `anyhow` for simple tools.

WASM interop

For simple JS interop, use `wasm-bindgen`.

```
use wasm_bindgen::prelude::*;

#[wasm_bindgen]
pub fn add(a: i32, b: i32) -> i32 {
    a + b
}
```

Build:

```
cargo build --release --target wasm32-unknown-unknown
wasm-bindgen --target web --out-dir pkg target/wasm32-unknown-unknown/release/my_tool.wasm
```

Keep exported functions simple: numbers, strings, and typed arrays.

Testing and linting

Run tests and lint:

```
cargo test
cargo clippy
cargo fmt
```

Use `#[cfg(test)]` modules for unit tests and `tests/` for integration tests.

Practical crates

Common picks for PhilJS projects:

- `serde` and `serde_json` for serialization
- `reqwest` for HTTP (native)
- `wasm-bindgen`, `js-sys`, `web-sys` for WASM
- `thiserror` or `anyhow` for errors
- `tracing` for logs and spans

Command quick reference

```
rustup update
cargo build --release
cargo test
cargo clippy
cargo fmt
```

Links

- [The Rust Book](#)
- [Rust By Example](#)
- [wasm-bindgen Guide](#)

Glossary

A quick reference for terms used throughout the PhilJS book.

- **Action**: Route-bound mutation handler; runs on server/edge, can invalidate caches.
- **Adapter**: Platform-specific integration for SSR/edge/serverless (Vercel, Netlify, CF, AWS, Bun, Deno, Node, static).
- **Cache tag**: Identifier used to scope loader caches and invalidations (e.g., `['user', id]`).
- **CSR**: Client-side rendering; PhilJS prefers SSR + islands for speed.
- **Effect**: PhilJS reactive side-effect; should perform side effects only.
- **Hydration**: Attaching event handlers/state to server-rendered markup.
- **Island**: An interactive component hydrated separately from the rest of the page.
- **Loader**: Route-bound data fetcher; runs before render, supports caching and revalidation.
- **Memo**: Derived value that recomputes when dependencies change.
- **Resource**: Async data primitive with loading/error states.
- **SSR**: Server-side rendering; in PhilJS, often streamed and edge-friendly.
- **Store**: Structured state container with middleware, history, persistence.
- **Stale time**: Duration a cached value is considered fresh before revalidation.
- **TTFB**: Time to first byte; key SSR/edge performance metric.
- **Untrack**: Read a signal without subscribing to its changes.
- **Variant/Flag**: Value used to control experiments or feature toggles.

Index (Quick Links)

Use these jump links to reach key topics quickly in PDF/EPUB (links are relative and stay clickable after export).

Getting Started

- [Installation](#)
- [Quick Start](#)
- [Thinking in PhilJS](#)
- [Project Structure](#)
- [Tutorial: Todo App](#)

Core and Rendering

- [Components](#)
- [Signals](#)
- [Effects and Memos](#)
- [JSX Basics](#)

Routing and Data

- [Routing Overview](#)
- [Navigation](#)
- [Route Guards](#)
- [Data Layer Overview](#)
- [Caching](#)
- [Server Functions](#)
- [Forms](#)

SSR and Islands

- [SSR Overview](#)
- [Islands](#)
- [SSR/Islands Playbook](#)

State and Styling

- [Store and History](#)
- [State Middleware](#)
- [Styling Options](#)
- [Design Systems](#)
- [Motion](#)

Testing and Performance

- [Testing Overview](#)
- [E2E Testing](#)
- [Performance Overview](#)
- [Performance Budgets](#)

- [Code Splitting](#)
- [Profiling](#)
- [Web Vitals](#)

Best Practices

- [Best Practices Overview](#)
- [Component Patterns](#)
- [Security Practices](#)
- [Testing Practices](#)

Deployment and DevOps

- [Deployment Overview](#)
- [CI/CD](#)
- [Publishing Workflow](#)

Security and Observability

- [Security Overview](#)
- [API Security](#)
- [Logging and Tracing](#)
- [Runbooks](#)

Architecture

- [Architecture Overview](#)
- [Runtime Internals](#)
- [Data Modeling](#)

Platforms

- [Edge and Serverless](#)
- [PWA](#)
- [Desktop](#)
- [Mobile](#)
- [WebGPU](#)
- [WebAssembly](#)

Integrations

- [OpenAPI](#)
- [GraphQL](#)
- [WebSockets](#)
- [AI](#)
- [Payments](#)

Patterns and Content

- [Streaming and Progressive Rendering](#)
- [Offline UX](#)
- [Accessibility Patterns](#)
- [Internationalization](#)
- [SEO](#)
- [Charts and Diagrams](#)

Packages

- [Package Atlas](#)

Appendices

- [Glossary](#)
- [TypeScript 6.x Deep Dive](#)
- [Rust 1.92 Quickstart](#)