Featuring PAN Communication Bus

Learning LARC

Modern Web Architecture Using Open Standards.

Christopher Robison

Learning LARC

A Hands-On Guide to Modern Web Development

Christopher Robison

December 2025

Praise for Learning LARC

“LARC represents a return to web fundamentals while embracing modern capabilities. This book beautifully explains why that matters.” — **David B. - Software Engineer**

“Finally, a framework that respects the browser. Learning LARC shows you how to build without fighting the platform.” — **Jon W. - App Developer**

“The PAN bus architecture is elegant and powerful. This book makes it accessible to everyone.” — **Mary S. - Designer / Artist**

Copyright

Printed in the United States of America.

Published by LARC Press.

Table of Contents

- Setting Up Your Development Environment
- Your First LARC Application
- Project Structure
- Import Maps Explained
- Development Workflow
- Common Patterns

Part II: Building Components

Chapter 4: Creating Web Components

- Anatomy of a LARC Component
- Shadow DOM Deep Dive
- Attributes and Properties
- Component Styling
- Lifecycle Methods
- Testing Components

Chapter 5: The PAN Bus

- Understanding Pub/Sub Architecture
- Topics and Namespaces
- Publishing Messages
- Subscribing to Events
- Message Patterns
- Debugging PAN Communication

Chapter 6: State Management

- Component-Local State
- Shared State Patterns
- The pan-store Component
- IndexedDB Integration
- Persistence Strategies
- Offline-First Applications

Chapter 7: Advanced Component Patterns

- Compound Components
- Higher-Order Components
- Component Composition
- Slots and Content Projection
- Dynamic Component Loading
- Performance Optimization

Part III: Building Applications

Chapter 8: Routing and Navigation

- Client-Side Routing
- The pan-router Component
- Route Parameters
- Nested Routes
- Route Guards
- History Management

Chapter 9: Forms and Validation

- Form Components
- Two-Way Data Binding
- Validation Strategies
- Error Handling
- File Uploads
- Form Submission

Chapter 10: Data Fetching and APIs

- The pan-fetch Component
- REST API Integration
- GraphQL Support
- WebSocket Communication
- Server-Sent Events

- Error Handling and Retry Logic

Chapter 11: Authentication and Security

- Authentication Patterns
- The pan-auth Component
- JWT Token Management
- Protected Routes
- CORS Considerations
- Security Best Practices

Part IV: Advanced Topics

Chapter 12: Server Integration

- Backend Architecture
- Node.js Integration
- PHP Connector
- Python/Django Integration
- Database Patterns
- Real-Time Communication

Chapter 13: Testing

- Unit Testing Components
- Integration Testing
- End-to-End Testing
- Visual Regression Testing
- Performance Testing
- Continuous Integration

Chapter 14: Performance and Optimization

- Loading Strategies
- Code Splitting
- Lazy Loading Components
- Caching Strategies

- Bundle Size Optimization
- Performance Monitoring

Chapter 15: Deployment

- Static Hosting
- CDN Configuration
- Environment Variables
- CI/CD Pipelines
- Monitoring and Analytics
- Production Best Practices

Part V: Ecosystem

Chapter 16: Component Library

- Using the Component Registry
- Contributing Components
- Creating a Component Library
- Documentation Strategies
- Versioning and Releases

Chapter 17: Tooling

- Development Tools
- CLI Tools
- VS Code Integration
- Browser DevTools
- Debugging Techniques

Chapter 18: Real-World Applications

- Case Study: E-Commerce Platform
- Case Study: Dashboard Application
- Case Study: Blog/CMS
- Lessons Learned
- Best Practices

Appendices

Appendix A: Web Components API Reference

- Custom Elements
- Shadow DOM
- HTML Templates
- ES Modules

Appendix B: PAN Bus API Reference

- Core Methods
- Message Formats
- Topic Patterns
- Configuration Options

Appendix C: Component API Reference

- Built-in Components
- Component Properties
- Events and Methods

Appendix D: Migration Guides

- From React
- From Vue
- From Angular
- From jQuery

Appendix E: Resources

- Official Documentation
- Community Resources
- Video Tutorials
- Example Projects

Index

About the Author

Christopher Robison is a veteran software engineer and architect with nearly three decades of experience building systems that range from biotech and online trading platforms to complex web applications and AI-driven tools. A lifelong maker with a deep appreciation for open standards, he has spent his career exploring the boundaries of what the web can do when you stop fighting the platform and start embracing it.

He is the creator of LARC.js and the PAN message bus, a browser-native architecture inspired by the elegant simplicity of the automotive CAN bus. His work blends engineering pragmatism with a playful curiosity that has led him to design everything from 3D printers and robotics to interactive music systems and decentralized applications.

Christopher currently lives in San Francisco, where he continues to build things that bridge the digital and physical worlds — and occasionally sneaks off to play punk rock shows with his band.

Website: https://larcjs.com

Foreword

by Christopher Robison

I didn't set out to build a framework. I set out to escape one — or at least escape the gravitational pull of the endless build pipeline.

After decades of building things for the web, my machine had become a storage exhibit of Node versions, Python versions, shims, wrappers, and dependency folders with the mass of small moons. Not because any of it was bad. Build tools are fine. For big projects, they're downright amazing. But somewhere along the way, we normalized the idea that even the simplest experiment needed a pipeline, a bundler, a transpiler, and a twelve-step hydration ritual before it could say "Hello, World."

That friction bothered me.

I wanted the feeling I had back in the early days: the joy of dropping a `<script>` tag into an HTML file and instantly seeing something come alive. No ceremony. No yak shaving. Just a browser, a file, and an idea.

Web Components felt close to that spirit — native modules, shadow DOM, real encapsulation — but they were oddly isolated. Each component was a self-contained island. Reusable, yes. Architecturally composable? Not really. Nothing tied them together except whatever glue code you wrote yourself. It felt like someone had shipped LEGO bricks without the ability to click them together.

That's when I remembered the CAN bus in cars.

The CAN bus is this beautifully simple ecosystem: dozens of systems — sensors, motors, controllers — all sharing a single communication line. Anybody can broadcast. Anybody can listen. Nobody needs to know who else exists. A message goes out, and the parts that care respond. It's loosely coupled machinery at its finest.

I wanted that for the web.

So I built the PAN bus — the Page Area Network — and started experimenting. Not with the

intention of making A Real Framework™, but out of curiosity. How far could I push this idea? What could I build if every component on the page could talk over a shared bus, using nothing but browser-native APIs and one tiny script include?

That little experiment got out of hand in the best way.

I kept pushing it, partly out of stubbornness, partly out of sheer delight. I wanted to see if I could build real, full-blown applications with no build process at all — just a single script tag pointing to LARC and a page full of components chatting over the bus. And it turned out to be… fun. Refreshing. Capable. Liberating, even. A loose, elegant architecture emerged almost on its own.

Along the way, I realized something important: I'm not anti-build-tool. They solve real problems, especially at scale. But they shouldn't be mandatory for everything. And they shouldn't overshadow the fact that the browser today is powerful enough to build serious applications with a simple HTML page, a few components, and a shared message bus.

React, Angular, Vue — they solved problems that absolutely needed solving at the time. The web platform in 2015 was missing big pieces: templating, reactivity, routing, structured components, coherence. These frameworks carried the industry through that era. But the web has evolved since then. Many of those features now exist natively — standardized, built-in, fast, and universally available.

LARC isn't here to replace those frameworks. It complements them. It fills in the 20% Web Components never standardized — the shared communication fabric. The glue that lets components coexist instead of siloing themselves off. It also makes bundle sizes smaller and architectures cleaner, whether you're going framework-free or integrating with your existing stack.

If this book succeeds, you'll see what I saw: the thrill of rediscovering the browser as a first-class app platform. The joy of building big things out of small, decoupled pieces. And the surprising power of an architecture that starts with a simple HTML file and one script include.

The web grew up. Now we get to build like it.

— Christopher Robison

1 Philosophy and Background

1.1 The Problem with Modern Web Development

If you've been building web applications for the past decade, you've likely experienced what many developers call "JavaScript fatigue." The modern web development landscape has become increasingly complex, with countless tools, frameworks, and build processes standing between you and shipping working code.

Consider a typical modern web project setup:

1. Initialize your project with a framework CLI (`create-react-app` , `vue create` , etc.)
2. Install hundreds or thousands of npm dependencies
3. Configure webpack, Babel, TypeScript, ESLint, Prettier
4. Set up build scripts for development, production, testing
5. Wait for builds to complete (sometimes minutes)
6. Debug build configuration issues when something breaks
7. Update dependencies regularly to patch security vulnerabilities
8. Repeat the cycle when frameworks release breaking changes

This complexity wasn't always necessary. In the early days of the web, you could create an HTML file, add some CSS and JavaScript, and open it directly in a browser. No build step. No toolchain. No configuration. Just code that runs.

What happened?

1.1.1 The Rise of Complexity

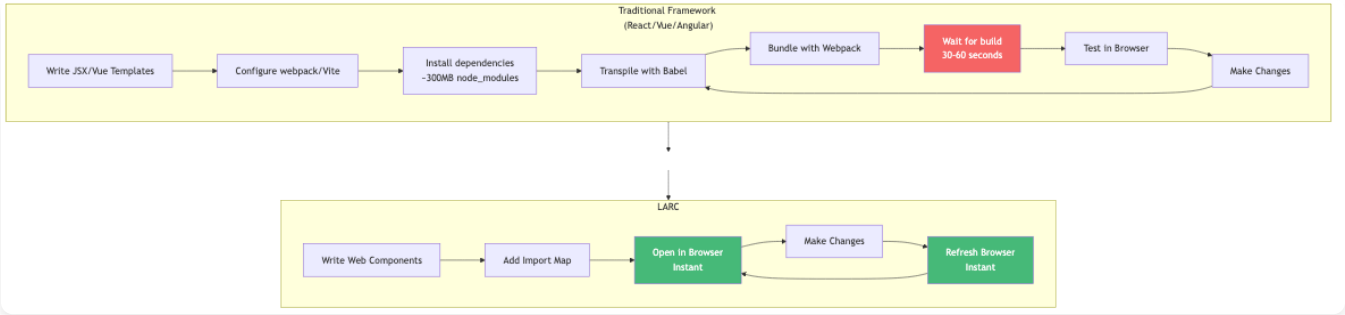

Figure 1.1: Development Workflow - Traditional vs LARC

The web platform evolved, but it didn't evolve fast enough for ambitious developers. We wanted:

- **Component-based architecture** — but HTML didn't have custom elements yet
- **Module systems** — but JavaScript didn't have native imports
- **Reactive data binding** — but the DOM wasn't designed for it
- **Advanced syntax** — like JSX, TypeScript, or class properties

Frameworks filled these gaps by building abstractions on top of the web platform. But these abstractions came with costs:

- **Build toolchains** became mandatory to transpile code
- **Bundle sizes** grew as framework code was shipped to browsers
- **Learning curves** steepened as developers had to learn both the framework and the tools
- **Debugging** became harder with source maps and transpiled code
- **Performance** suffered from unnecessary abstraction layers

The irony? While we were busy building these elaborate toolchains, the web platform itself was evolving to support many of the features we wanted natively.

1.1.2 The Platform Has Caught Up

Today's web platform is remarkably capable. Modern browsers support:

- **Custom Elements** — native component definition
- **Shadow DOM** — true style encapsulation
- **ES Modules** — native JavaScript modules with imports
- **Import Maps** — dependency management without bundlers
- **Template Literals** — dynamic HTML without JSX
- **Proxy and Reflect** — reactive data patterns
- **CSS Custom Properties** — themeable components
- **Web Components** — standards-based component architecture

These aren't polyfills or experimental features. They're stable, well-supported standards that work across all modern browsers. Yet most web frameworks continue to build elaborate abstractions on top of the platform, ignoring these native capabilities.

1.1.3 A Common Scenario

Let's look at a real-world example. Imagine you're building a simple dashboard with a few interactive components: a card, a button, and a data table. Here's what this might look like in a typical React project:

The Setup:

```
npx create-react-app my-dashboard
cd my-dashboard
npm install styled-components react-router axios redux
# Wait 5-10 minutes for installation
# Project size: ~300MB, ~1000+ dependencies
```

The Code:

```
// Card.jsx
import React from 'react';
import styled from 'styled-components';

const StyledCard = styled.div`
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 2px 4px rgba(0,0,0,0.1);
`;

export default function Card({ title, children }) {
  return (
    <StyledCard>
      <h2>{title}</h2>
      {children}
    </StyledCard>
  );
}
```

The Build:

```
npm run build
# Wait 30-60 seconds
# Output: Minified, bundled, transpiled code
# Bundle size: 200-500KB (before your actual code)
```

Now, here's the same thing with native Web Components and LARC:

The Setup:

```
<!DOCTYPE html>
<html>
<head>
  <script type="module">
  {
    "imports": {
      "@larcjs/ui": "https://cdn.jsdelivr.net/npm/@larcjs/
        components@2.0.0/pan-card.mjs"
    }
  }
  </script>
</head>
<body>
  <pan-card title="Dashboard">
    <p>Your content here</p>
  </pan-card>

</body>
</html>
```

The Build:

```
# There is no build step. Open the HTML file. It works.
```

Same functionality. Zero dependencies. No build process. No toolchain. Just HTML, CSS, and JavaScript working together as the platform intended.

1.2 A Return to Fundamentals

LARC (Lightweight Autonomous Reactive Components) represents a philosophical shift back to web fundamentals. But this isn't about going backward—it's about recognizing that the platform has evolved to the point where many of our abstractions are no longer necessary.

1.2.1 What LARC Is

LARC is a set of conventions, patterns, and utilities for building modern web applications using native web standards:

- **Web Components** for encapsulated, reusable UI elements
- **ES Modules** for code organization and imports
- **Import Maps** for dependency management
- **The PAN Bus** for component communication
- **Native APIs** for state, routing, and data fetching

LARC provides guidance and utilities, but it doesn't abstract away the platform. When you write LARC code, you're writing standard JavaScript, HTML, and CSS that runs directly in the browser.

1.2.2 What LARC Is Not

LARC is deliberately minimal. It is **not**:

- A framework with proprietary APIs you must learn
- A template language that requires compilation
- A state management system with complex rules
- A build tool that transforms your code
- A runtime that interprets your components

If you know HTML, CSS, and JavaScript, you already know most of LARC.

1.2.3 Core Principles

LARC is built on several core principles:

1.2.3.1 1. Standards First

LARC embraces web standards rather than fighting them. Every LARC component is a valid Web Component. Every LARC module is a valid ES Module. If you understand the standards, you understand LARC.

1.2.3.2 2. Zero Build for Development

During development, you should be able to edit a file and refresh the browser. No build step. No waiting. No configuration. The browser is your development environment.

This doesn't mean builds are forbidden—you can still optimize for production if needed. But they should be optional enhancements, not requirements.

1.2.3.3 3. Progressive Enhancement

Start simple and add complexity only when needed. A basic component can be a few lines of JavaScript. As requirements grow, add features incrementally: state management, routing, server integration, etc.

You're never locked into architectural decisions made at project initialization. LARC applications evolve naturally.

1.2.3.4 4. Local First, Network Aware

Components should work independently with local state. Network communication happens through explicit, observable patterns (the PAN bus). This makes components:

- Easier to test (no mocking required)

- More reusable (fewer dependencies)
- More resilient (graceful degradation)

1.2.3.5 5. Developer Experience Through Simplicity

Good DX doesn't require complex tooling. It comes from:

- Clear, predictable patterns
- Minimal abstractions
- Fast feedback loops
- Easy debugging
- Comprehensive documentation

When something breaks in LARC, you can open browser DevTools and debug standard JavaScript. No source maps. No transpiled code. No framework internals.

1.3 The LARC Philosophy

At its heart, LARC is about **respecting the platform**. The web is incredibly powerful, yet we've spent years building layers of abstraction that hide its capabilities. LARC removes those layers.

1.3.1 Composition Over Configuration

Rather than configuring a framework through JSON or CLI flags, LARC applications are composed from standard parts:

```
<!-- Composition: Combine standard elements -->
<pan-router>
  <pan-route path="/" component="home-page"></pan-route>
  <pan-route path="/dashboard" component="dashboard-page"></pan-route>
</pan-router>
```

Each element is understandable in isolation. There's no magic configuration file that controls behavior across your entire application.

1.3.2 Convention Over Prescription

LARC suggests patterns but doesn't enforce them. There's no "one true way" to structure a LARC application. The conventions exist to make common tasks easier, but you can always drop down to standard APIs when needed.

For example, LARC recommends the PAN bus for component communication, but you can also use:

- Custom events
- Direct property access
- Shared state objects
- URL parameters
- LocalStorage
- Any other standard browser API

Choose the right tool for your specific use case.

1.3.3 Explicit Over Implicit

LARC favors explicitness. When a component fetches data, you see the fetch call. When state changes, you see the assignment. When events are dispatched, you see the dispatch.

Compare these two approaches:

Implicit (typical framework):

```
function UserProfile() {
  const [user, loading, error] = useUser(userId);

  if (loading) return <Spinner />;
  if (error) return <Error message={error} />;

  return <ProfileCard user={user} />;
}
```

Magic happens in `useUser`. Where does the data come from? When does it refetch? What triggers updates? You need to understand the framework's mental model.

Explicit (LARC):

```
class UserProfile extends HTMLElement {
  async connectedCallback() {
    this.render({ loading: true });

    try {
      const response = await fetch(`/api/users/${this.userId}`);
      const user = await response.json();
      this.render({ user });
    } catch (error) {
      this.render({ error: error.message });
    }
  }

  render(state) {
    if (state.loading) {
      this.innerHTML = '<loading-spinner></loading-spinner>';
    } else if (state.error) {
      this.innerHTML = `<error-message text="${state.error}"></error-
        message>`;
    } else {
      this.innerHTML = `<profile-card .user="${state.user}"></profile-
        card>`;
    }
  }
}
```

Every step is visible. You can trace exactly what happens and when. Debugging is straightforward because you're working with standard JavaScript.

1.4 Why "No Build" Matters

The "no build" philosophy isn't about being purist or rejecting tools. It's about removing unnecessary complexity and its associated costs.

1.4.1 Development Speed

Without a build step, your development cycle is:

1. Edit code
2. Refresh browser
3. See changes

That's it. No waiting for webpack to rebuild. No watching file watchers fail. No debugging build configurations.

This might seem like a small thing, but it compounds. Over a day of development, those 10-30 second build times add up to significant lost productivity. More importantly, they break flow state.

1.4.2 Debugging Simplicity

When you open browser DevTools in a LARC application, you see your actual code. No source maps needed. No transpiled output. No minified framework internals.

Set a breakpoint in your component's `connectedCallback`. It stops exactly where you expect. The call stack is readable. Variables are named as you wrote them.

This makes debugging accessible to junior developers and reduces time spent fighting tools.

1.4.3 Deployment Simplicity

A LARC application can be deployed to any static host:

- GitHub Pages
- Netlify
- Vercel
- Amazon S3
- Any web server

No server-side rendering. No Node.js runtime. No build artifacts to manage. Just upload

HTML, CSS, and JavaScript files.

Want to deploy to a CDN? Your entire application is already CDN-friendly because it's just static files.

1.4.4 Lower Barrier to Entry

New developers can learn web development by:

1. Creating an HTML file
2. Adding some CSS and JavaScript
3. Opening it in a browser

No installation. No environment setup. No project configuration. This is how the web should work.

With build tools, new developers face:

1. Install Node.js
2. Learn npm/yarn
3. Understand package.json
4. Configure webpack/Babel
5. Troubleshoot build errors
6. Learn framework-specific tooling

Before writing a single line of application code, they've already encountered dozens of concepts unrelated to actual web development.

1.4.5 Sustainability

Build tools and frameworks change rapidly. A React application from 2015 likely needs significant updates to run today. Build configurations break. Dependencies become unmaintained. Migration guides are incomplete.

LARC applications use web standards. A LARC application from 2025 will still run in 2035 because it's built on stable browser APIs, not framework-specific abstractions.

This doesn't mean LARC applications never need updates—APIs evolve, best practices change. But the core architecture is built on a foundation that changes slowly and deliberately through standards processes.

1.5 When to Use LARC

LARC isn't the right choice for every project. Understanding when to use it (and when not to) helps you make informed decisions.

1.5.1 LARC Excels At

Small to Medium Applications Projects with 10-100 components where simplicity and maintainability matter more than framework ecosystem size.

Dashboard and Admin Panels Internal tools where the development team controls the environment and values fast iteration.

Progressive Web Apps Applications that benefit from offline-first architecture and minimal JavaScript overhead.

Learning Projects Teaching web development without the complexity of modern toolchains.

Embedded Widgets Reusable components that need to work in any environment without framework dependencies.

Prototypes and MVPs Quickly validating ideas without upfront tooling investment.

1.5.2 Consider Alternatives When

Very Large Teams If you have 50+ developers working on a single codebase, framework opinions and tooling might provide valuable guardrails.

Heavy Framework Ecosystem Dependencies If your project critically relies on a specific framework's ecosystem (e.g., React Native integration, specific UI libraries), switching costs may be prohibitive.

Server-Side Rendering is Critical While LARC supports SSR, frameworks like Next.js have more mature SSR/SSG ecosystems.

Team Expertise If your entire team is deeply experienced in React/Vue/Angular and inexperienced with Web Components, the learning curve might slow initial development.

That said, LARC's simplicity often means the learning curve is shorter than expected. Most experienced developers can become productive with LARC in days, not weeks.

1.5.3 Hybrid Approaches

You don't have to go all-in on LARC. Consider hybrid approaches:

Progressive Migration Build new features in LARC while maintaining existing framework code. Web Components can coexist with React, Vue, or Angular.

Micro-frontends Use LARC for some micro-frontends and other frameworks for others. Web Components provide clean boundaries.

Component Libraries Build a LARC component library that can be consumed by any framework or vanilla JavaScript.

1.6 What You'll Build

Throughout this book, you'll build several progressively complex applications:

1.6.1 Chapter Examples

Each chapter includes focused examples demonstrating specific concepts:

- A **counter component** (Chapter 4) to understand component basics
- A **todo list** (Chapter 5) to learn PAN bus communication
- A **user profile form** (Chapter 9) to master form handling
- A **data table** (Chapter 10) to work with APIs and data

1.6.2 Capstone Project: TaskFlow

In the final chapters, you'll build **TaskFlow**, a complete project management application featuring:

- User authentication and authorization
- Real-time collaboration via WebSockets
- Offline-first architecture with IndexedDB
- Drag-and-drop task boards
- File attachments and comments
- Search and filtering
- Data visualization
- Mobile-responsive design

TaskFlow will demonstrate how LARC patterns scale to production applications while remaining maintainable and performant.

1.6.3 What You'll Learn

By the end of this book, you'll be able to:

- Build complex, maintainable applications using Web Components
- Design effective component communication patterns with the PAN bus
- Manage application state without external frameworks
- Integrate with backend APIs and real-time services
- Handle routing, forms, and authentication
- Write testable, reusable components
- Optimize performance and bundle size
- Deploy LARC applications to production
- Make informed decisions about when to use LARC vs. other approaches

1.7 Looking Ahead

The next chapter dives into LARC's core concepts: Web Components, the PAN bus, and

event-driven architecture. You'll learn the fundamental patterns that make LARC applications work.

But before we get technical, take a moment to consider what drew you to this book. Perhaps you're tired of build tool complexity. Perhaps you want to understand how the web really works. Perhaps you're curious about a different approach.

Whatever your motivation, LARC offers something increasingly rare in modern web development: simplicity without sacrificing capability. You're about to learn how to build serious web applications using the platform itself, not abstractions on top of it.

Let's begin.

1.8 Summary

- Modern web development has become unnecessarily complex with build tools, frameworks, and abstractions
- The web platform has evolved to support features natively that once required frameworks
- LARC uses web standards (Web Components, ES Modules, Import Maps) to build applications without build steps
- Core principles: standards first, zero build for development, progressive enhancement, local first
- "No build" matters for development speed, debugging simplicity, deployment, and sustainability
- LARC works best for small-to-medium applications, dashboards, PWAs, and prototypes
- You'll build real applications throughout this book, culminating in a production-ready project management app

---

1.9 Further Reading

For comprehensive reference: - *Building with LARC* Chapter 1: Introduction - Reference manual overview and conventions - *Building with LARC* Chapter 2: Core Concepts - Deep dive into LARC architecture - *Building with LARC* Appendix D: Migration Guide - Migrating from React, Vue, Angular

2 Core Concepts

Now that you understand LARC's philosophy, let's explore the technical foundation that makes it work. This chapter introduces the core concepts you'll use throughout the book: Web Components, the PAN bus, event-driven architecture, and the component lifecycle.

Don't worry if some of these concepts are new to you. We'll build understanding progressively, starting with the basics and working toward more sophisticated patterns.

2.1 Web Components Refresher

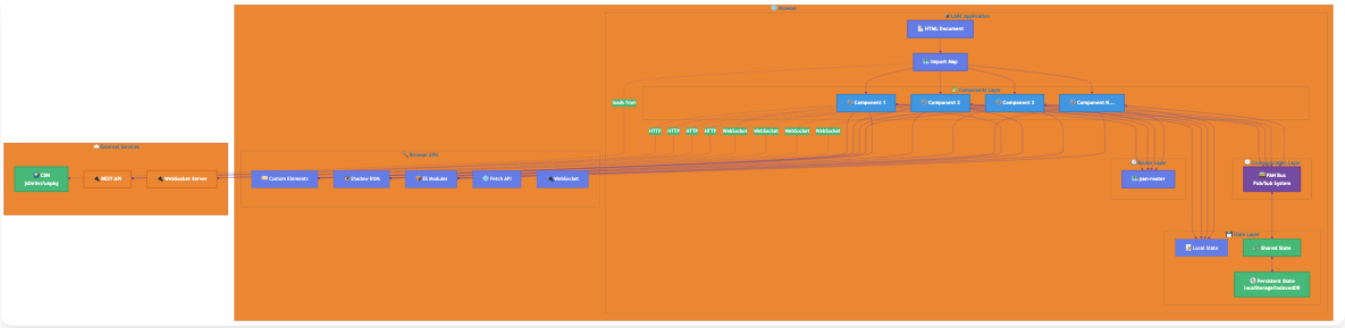

Figure 2.1: LARC High-Level Architecture

Web Components are a suite of browser APIs that let you create custom, reusable HTML elements. Unlike framework components, Web Components are browser standards supported natively across all modern browsers.

2.1.1 The Three Pillars

Web Components rest on three main technologies:

2.1.1.1 1. Custom Elements

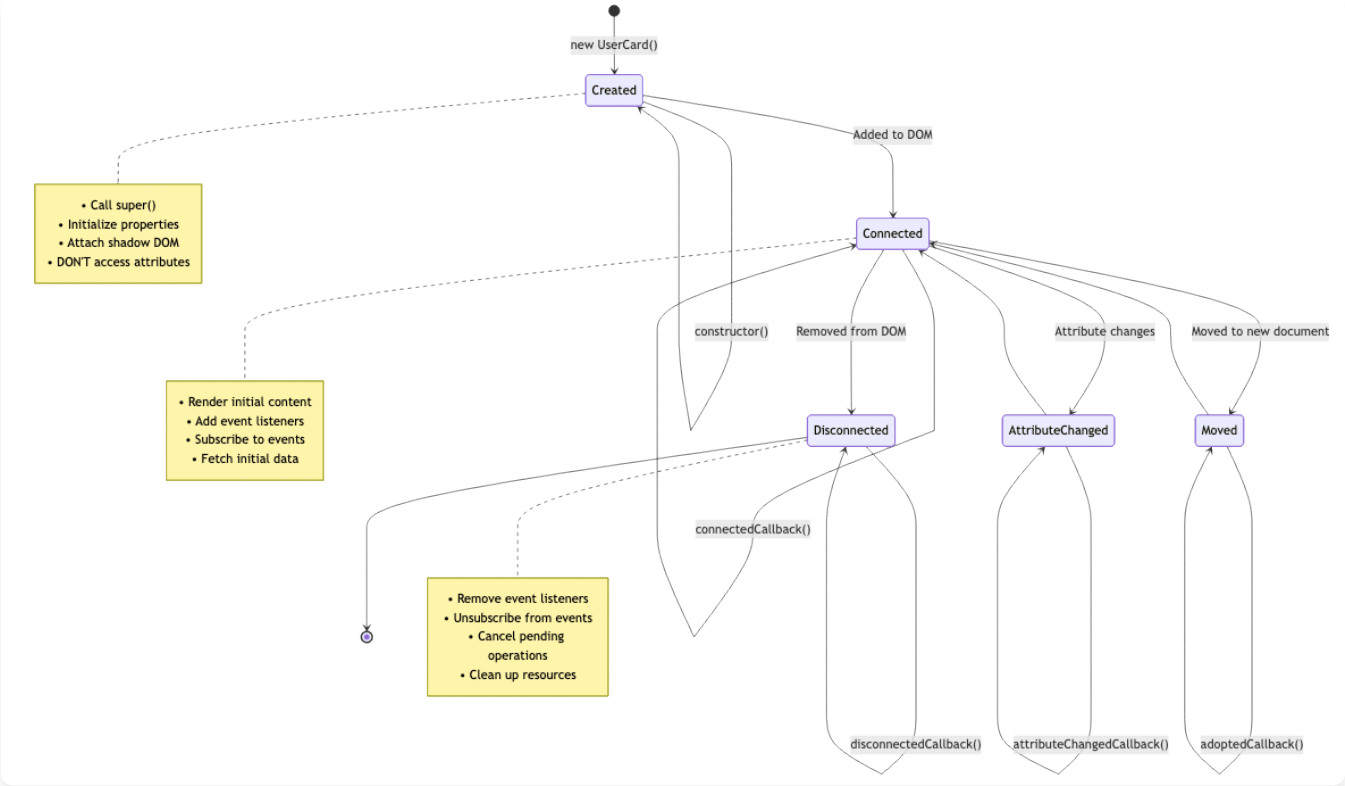

Figure 2.2: Web Component Anatomy

Custom Elements let you define new HTML tags with custom behavior:

```
// Define a custom element
class HelloWorld extends HTMLElement {
  connectedCallback() {
    this.textContent = 'Hello, World!';
  }
}

// Register it
customElements.define('hello-world', HelloWorld);
```

Now you can use `<hello-world></hello-world>` in your HTML, and it works like any built-in element.

Key Points:

- Element names must contain a hyphen (e.g., `my-component`, not `mycomponent`)
- Custom elements inherit from `HTMLElement` or another HTML element
- They have lifecycle callbacks for creation, connection, and removal

2.1.1.2 2. Shadow DOM

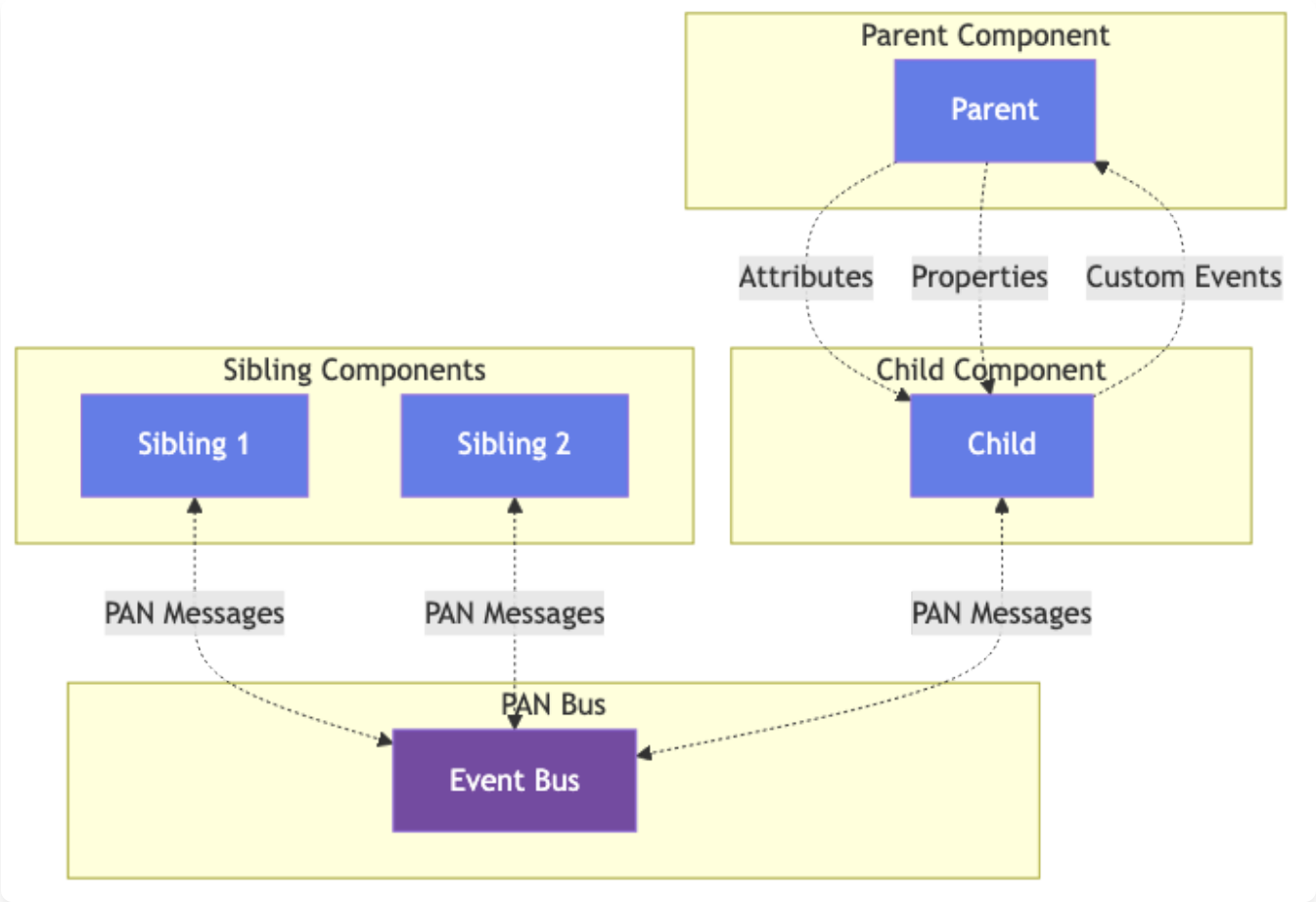

Figure 2.3: Shadow DOM Tree Structure

Shadow DOM provides style and markup encapsulation:

```
class FancyButton extends HTMLElement {
  constructor() {
    super();
    // Create shadow root
    this.attachShadow({ mode: 'open' });
  }

  connectedCallback() {
    this.shadowRoot.innerHTML = `
      <style>
        button {
          background: blue;
          color: white;
          border: none;
          padding: 10px 20px;
          border-radius: 4px;
        }
      </style>
      <button>
        <slot></slot>
      </button>
    `;
  }
}

customElements.define('fancy-button', FancyButton);
```

The styles inside Shadow DOM don’t leak out, and external styles don’t leak in:

```
<!-- This button is blue (from shadow DOM) -->
<fancy-button>Click Me</fancy-button>

<!-- This button is not affected by fancy-button's styles -->
<button>Regular Button</button>

<style>
  /* This won't affect fancy-button's internal button */
  button { background: red; }
</style>
```

Key Points:

- Shadow DOM creates an isolated scope for styles and DOM
- Use `<slot>` elements to project content from light DOM into shadow DOM
- `mode: 'open'` makes shadow root accessible via `element.shadowRoot`

2.1.1.3 3. HTML Templates

Templates define reusable chunks of markup that aren't rendered until activated:

```
<template id="card-template">
  <style>
    .card {
      border: 1px solid #ddd;
      border-radius: 8px;
      padding: 16px;
    }
  </style>
  <div class="card">
    <h2 class="title"></h2>
    <p class="content"></p>
  </div>
</template>

<script>
  class SimpleCard extends HTMLElement {
    connectedCallback() {
      const template = document.getElementById('card-template');
      const clone = template.content.cloneNode(true);

      clone.querySelector('.title').textContent =
        this.getAttribute('title');
      clone.querySelector('.content').textContent =
        this.getAttribute('content');

      this.attachShadow({ mode: 'open' });
      this.shadowRoot.appendChild(clone);
    }
  }

  customElements.define('simple-card', SimpleCard);
</script>
```

Key Points:

- Template content is inert (scripts don't run, images don't load)
- Templates can be defined in HTML or created programmatically
- Clone template content before using it

2.1.2 Web Components vs Framework Components

It's worth understanding how Web Components differ from framework components:

Aspect	Web Components	React Components
Definition	Browser standard	Library-specific
Syntax	JavaScript classes	JSX or functions
Lifecycle	Native callbacks	Virtual DOM lifecycle
Reusability	Works everywhere	Requires React
Build step	Optional	Required (for JSX)
Encapsulation	Shadow DOM	CSS Modules/CSS-in-JS

Both approaches have their place. Web Components excel at true reusability and standards-based development. Framework components often provide better ergonomics within their specific ecosystem.

LARC chooses Web Components because they align with the "standards first" principle.

2.2 The Page Area Network (PAN)

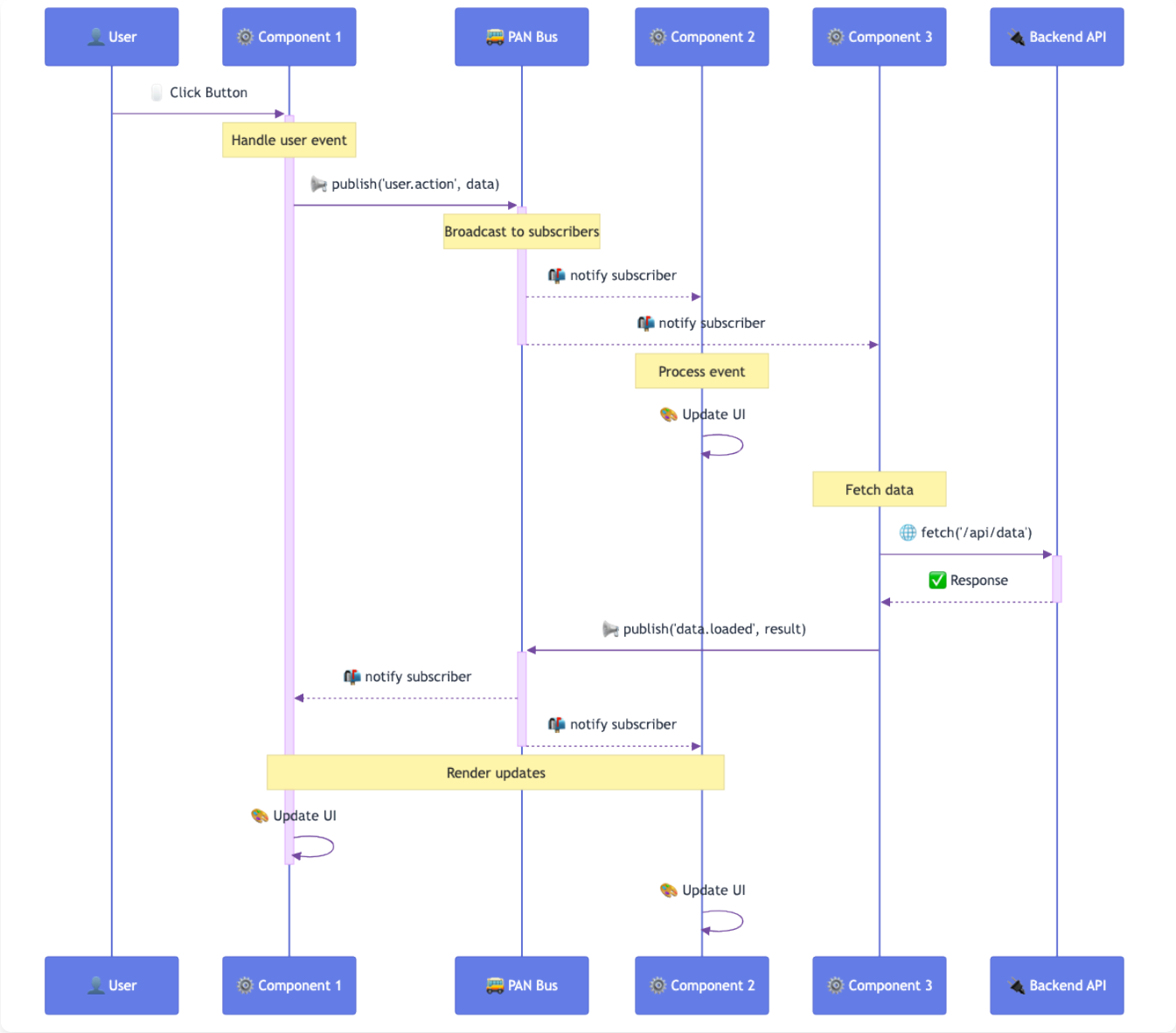

Figure 2.4: Component Communication Flow

The Page Area Network, or PAN bus, is LARC's event-driven communication system. It's inspired by microservices architecture but designed for browser components.

2.2.1 The Problem It Solves

In a traditional component tree, communication flows up and down:

```
App
├── Header
|    └── UserMenu
|         └── LogoutButton
└── Content
     └── UserProfile
```

If `LogoutButton` needs to notify `UserProfile` that the user logged out, you have several options:

1. **Pass callbacks down** through props (prop drilling)
2. **Lift state up** to a common ancestor
3. **Use context** or global state
4. **Dispatch custom events** that bubble up

Each approach has tradeoffs. Prop drilling creates tight coupling. Global state makes testing harder. Event bubbling is limited by DOM structure.

2.2.2 The PAN Bus Approach

The PAN bus provides a **decoupled pub/sub system**:

```
// LogoutButton publishes an event
pan.publish('user.logout', { userId: 123 });

// UserProfile subscribes to events (anywhere in the app)
pan.subscribe('user.logout', (data) => {
  console.log('User logged out:', data.userId);
  this.clearUserData();
});
```

Components don't need to know about each other. They communicate through topics (like `'user.logout'`) with no direct coupling.

2.2.3 Topic Namespaces

Topics use dot notation for organization:

```
'user.login'          // User logged in
'user.logout'         // User logged out
'user.profile.update' // Profile was updated

'cart.item.add'       // Item added to cart
'cart.item.remove'    // Item removed
'cart.checkout'       // Checkout initiated

'app.theme.change'    // Theme changed
'app.error'           // Application error
```

You can subscribe to specific topics or use wildcards:

```
// Specific topic
pan.subscribe('user.login', handler);

// Wildcard (all user events)
pan.subscribe('user.*', handler);

// All events (useful for debugging)
pan.subscribe('*', handler);
```

2.2.4 Message Patterns

The PAN bus supports several messaging patterns:

2.2.4.1 1. Fire and Forget

Most common pattern. Publish a message and continue:

```
pan.publish('notification.show', {
  type: 'success',
  message: 'Saved successfully'
});
```

2.2.4.2 2. Request/Response

Publish a message and wait for a response:

```
const result = await pan.request('api.fetch', {
  url: '/api/users',
  method: 'GET'
});
```

A subscriber handles the request and returns data:

```
pan.respond('api.fetch', async (data) => {
  const response = await fetch(data.url, { method: data.method });
  return response.json();
});
```

2.2.4.3 3. State Broadcast

Publish state changes that multiple components need:

```
// Theme switcher publishes
pan.publish('app.theme.change', { theme: 'dark' });

// Multiple components subscribe
class Header extends HTMLElement {
  connectedCallback() {
    pan.subscribe('app.theme.change', ({ theme }) => {
      this.applyTheme(theme);
    });
  }
}

class Sidebar extends HTMLElement {
  connectedCallback() {
    pan.subscribe('app.theme.change', ({ theme }) => {
      this.applyTheme(theme);
    });
  }
}
```

2.2.5 Why PAN Bus?

The PAN bus provides several advantages:

Loose Coupling Components don't need references to each other. Add or remove components without changing others.

Testability Test components in isolation. Mock the bus or test actual pub/sub behavior.

Debuggability Subscribe to `'*'` to log all messages. Visualize message flow easily.

Scalability Add new features by subscribing to existing topics. No need to modify existing code.

Flexibility Mix different communication patterns (events, requests, broadcasts) as needed.

2.3 Event-Driven Architecture

LARC applications use event-driven architecture (EDA) at multiple levels:

2.3.1 Browser Events

Standard DOM events for user interaction:

```
class ClickCounter extends HTMLElement {
  constructor() {
    super();
    this.count = 0;
  }

  connectedCallback() {
    this.innerHTML = `
      <button id="btn">Clicked ${this.count} times</button>
    `;

    this.querySelector('#btn').addEventListener('click', () => {
      this.count++;
      this.querySelector('#btn').textContent = `Clicked ${this.count}
        times`;
    });
  }
}
```

2.3.2 Custom Events

Components can dispatch custom events for parent components:

```
class ColorPicker extends HTMLElement {
  selectColor(color) {
    // Dispatch custom event
    this.dispatchEvent(new CustomEvent('colorchange', {
      detail: { color },
      bubbles: true,
      composed: true  // Cross shadow DOM boundary
    }));
  }
}

// Parent can listen
document.querySelector('color-picker').addEventListener('colorchange',
        (e) => {
  console.log('Selected color:', e.detail.color);
});
```

2.3.3 PAN Bus Events

For cross-component communication:

```
class SearchBox extends HTMLElement {
  handleInput(value) {
    pan.publish('search.query', { query: value });
  }
}

class SearchResults extends HTMLElement {
  connectedCallback() {
    pan.subscribe('search.query', ({ query }) => {
      this.search(query);
    });
  }
}
```

2.3.4 When to Use Each

Use DOM Events when:

- Handling user interactions (click, input, focus, etc.)
- Communication is parent-child relationship
- Following HTML semantics matters

Use Custom Events when:

- Component needs to notify parent/ancestors
- Event should bubble up the DOM tree
- Mimicking native element behavior

Use PAN Bus when:

- Components are not in parent-child relationship
- Multiple unrelated components need the same data
- Decoupling is more important than DOM semantics
- Building cross-cutting concerns (logging, analytics, etc.)

2.4 State Management Philosophy

LARC takes a pragmatic approach to state management: use the simplest solution that works, then scale up if needed.

2.4.1 State Hierarchy

State can exist at different levels:

2.4.1.1 1. Component-Local State

State that only matters to one component:

```
class TodoItem extends HTMLElement {
  constructor() {
    super();
    this.completed = false;  // Local state
  }

  toggle() {
    this.completed = !this.completed;
    this.render();
  }

  render() {
    this.classList.toggle('completed', this.completed);
  }
}
```

When to use: UI state, temporary values, component-specific configuration.

2.4.1.2 2. Shared State

State that multiple components need:

```
// Simple shared state object
const appState = {
  user: null,
  theme: 'light',
  notifications: []
};

// Components read from it
class UserMenu extends HTMLElement {
  connectedCallback() {
    this.render(appState.user);
  }
}

// Components write to it and notify via PAN
function updateTheme(theme) {
  appState.theme = theme;
  pan.publish('app.theme.change', { theme });
}
```

When to use: Application-wide settings, user data, feature flags.

2.4.1.3 3. Persistent State

State that survives page reloads:

```
class TodoList extends HTMLElement {
  loadTodos() {
    const saved = localStorage.getItem('todos');
    return saved ? JSON.parse(saved) : [];
  }

  saveTodos(todos) {
    localStorage.setItem('todos', JSON.stringify(todos));
  }
}
```

When to use: User preferences, draft content, offline data.

2.4.1.4 4. Server State

State that comes from and syncs with a server:

```
class UserProfile extends HTMLElement {
  async loadProfile() {
    const response = await fetch('/api/profile');
    this.profile = await response.json();
    this.render();
  }

  async saveProfile(updates) {
    await fetch('/api/profile', {
      method: 'PUT',
      body: JSON.stringify(updates)
    });
  }
}
```

When to use: Database records, API data, real-time updates.

2.4.2 Reactive State (Optional)

For more complex state needs, LARC provides reactive patterns using JavaScript Proxies:

```
function createStore(initialState) {
  const listeners = new Set();

  const state = new Proxy(initialState, {
    set(target, property, value) {
      target[property] = value;
      listeners.forEach(fn => fn(property, value));
      return true;
    }
  });

  return {
    state,
    subscribe(fn) {
      listeners.add(fn);
      return () => listeners.delete(fn);
    }
  };
}

// Usage
const store = createStore({ count: 0 });

class Counter extends HTMLElement {
  connectedCallback() {
    // Subscribe to changes
    this.unsubscribe = store.subscribe((prop, value) => {
      if (prop === 'count') this.render();
    });

    this.render();
  }

  disconnectedCallback() {
```

```
    this.unsubscribe();
  }

  render() {
    this.textContent = `Count: ${store.state.count}`;
  }
}

// Update state (automatically notifies subscribers)
store.state.count++;
```

This is similar to MobX or Vue's reactivity, but built with standard JavaScript.

2.5 Module System

LARC uses ES Modules, the native JavaScript module system.

2.5.1 Import/Export Basics

Export from a module:

```
// components/button.js
export class PanButton extends HTMLElement {
  // ...
}

export const BUTTON_TYPES = ['primary', 'secondary', 'danger'];

export default PanButton;
```

Import into another module:

```
// app.js
import PanButton, { BUTTON_TYPES } from './components/button.js';

// Or import everything
import * as Button from './components/button.js';
```

2.5.2 Import Maps

Import Maps let you define aliases for module paths:

```
<script type="importmap">
{
  "imports": {
    "@larcjs/core": "https://cdn.jsdelivr.net/npm/@larcjs/core@2.0.0/
        pan.mjs",
    "@larcjs/ui": "https://cdn.jsdelivr.net/npm/@larcjs/components@2.0.0/
        pan-card.mjs",
    "app/": "/src/",
    "components/": "/"
  }
}
</script>

<script type="module">
  // Use aliases
  import { pan } from '@larcjs/core';
  import { PanButton } from '@larcjs/ui';
  import { Header } from 'components/header.js';
</script>
```

This is similar to webpack's resolve aliases, but it's a browser standard.

2.5.3 Module Organization

A typical LARC project structure:

```
src/
├── components/
|   ├── header.js
|   ├── footer.js
|   └── sidebar.js
├── lib/
|   ├── api.js
|   ├── auth.js
|   └── utils.js
├── pages/
|   ├── home.js
|   ├── dashboard.js
|   └── profile.js
└── app.js
```

Each file is a module with clear responsibilities:

```
// src/lib/api.js
export async function fetchJSON(url, options = {}) {
  const response = await fetch(url, {
    ...options,
    headers: {
      'Content-Type': 'application/json',
      ...options.headers
    }
  });

  if (!response.ok) {
    throw new Error(`API error: ${response.status}`);
  }

  return response.json();
}

// src/components/user-list.js
import { fetchJSON } from '../lib/api.js';

export class UserList extends HTMLElement {
  async connectedCallback() {
    const users = await fetchJSON('/api/users');
    this.render(users);
  }
}

customElements.define('user-list', UserList);
```

2.6 The Component Lifecycle

Understanding the component lifecycle is essential for building robust LARC applications.

2.6.1 Lifecycle Callbacks

Web Components provide several lifecycle callbacks:

2.6.1.1 constructor()

Called when an instance is created:

```
class MyComponent extends HTMLElement {
  constructor() {
    // MUST call super() first
    super();

    // Initialize instance properties
    this.count = 0;
    this.data = null;

    // Attach shadow DOM if needed
    this.attachShadow({ mode: 'open' });

    // DON'T access attributes or children here
    // They might not be set yet
  }
}
```

Best practices:

- Always call `super()` first
- Initialize instance properties
- Attach shadow DOM
- Don’t access attributes, children, or parent elements
- Don’t render here (use `connectedCallback` instead)

2.6.1.2 connectedCallback()

Called when the element is inserted into the DOM:

```
connectedCallback() {
  // Now it's safe to access attributes, children, parent
  const title = this.getAttribute('title');

  // Render initial content
  this.render();

  // Add event listeners
  this.addEventListener('click', this.handleClick);

  // Fetch data
  this.loadData();

  // Subscribe to PAN events
  this.unsubscribe = pan.subscribe('data.update', this.handleUpdate);
}
```

Best practices:

- Render initial content
- Add event listeners
- Subscribe to events
- Fetch initial data
- Can be called multiple times if element is moved

2.6.1.3 disconnectedCallback()

Called when the element is removed from the DOM:

```
disconnectedCallback() {
  // Clean up event listeners
  this.removeEventListener('click', this.handleClick);

  // Unsubscribe from PAN events
  if (this.unsubscribe) {
    this.unsubscribe();
  }

  // Cancel pending operations
  if (this.fetchController) {
    this.fetchController.abort();
  }

  // Clear timers
  if (this.timer) {
    clearInterval(this.timer);
  }
}
```

Best practices:

- Remove event listeners to prevent memory leaks
- Unsubscribe from PAN events
- Cancel pending async operations
- Clear timers and intervals

2.6.1.4 attributeChangedCallback(name, oldValue, newValue)

Called when observed attributes change:

```
static get observedAttributes() {
  return ['title', 'count', 'active'];
}

attributeChangedCallback(name, oldValue, newValue) {
  // Called for each observed attribute that changes
  if (name === 'title') {
    this.updateTitle(newValue);
  } else if (name === 'count') {
    this.updateCount(Number(newValue));
  } else if (name === 'active') {
    this.updateActive(newValue !== null);
  }
}
```

Best practices:

- Only observe attributes you actually use
- Convert string values to appropriate types
- Handle null/undefined values
- Update only what changed (don't re-render everything)

2.6.1.5 adoptedCallback()

Called when the element is moved to a new document (rare):

```
adoptedCallback() {
  // Usually not needed
  // Called when element is moved between documents
  // (e.g., iframe scenarios)
}
```

2.6.2 Complete Lifecycle Example

Here’s a full component showing proper lifecycle management:

```
class DataTable extends HTMLElement {
  // Define which attributes to observe
  static get observedAttributes() {
    return ['url', 'page-size'];
  }

  constructor() {
    super();
    this.attachShadow({ mode: 'open' });

    // Initialize state
    this.data = [];
    this.pageSize = 10;
    this.currentPage = 1;
  }

  async connectedCallback() {
    // Initial render
    this.render();

    // Load data if URL is set
    const url = this.getAttribute('url');
    if (url) {
      await this.loadData(url);
    }

    // Subscribe to events
    this.unsubscribePan = pan.subscribe('table.refresh', () => {
      this.refresh();
    });

    // Set up event listeners
    this.addEventListener('page-change', this.handlePageChange);
  }
```

```
disconnectedCallback() {
  // Clean up subscriptions
  if (this.unsubscribePan) {
    this.unsubscribePan();
  }

  // Remove event listeners
  this.removeEventListener('page-change', this.handlePageChange);

  // Cancel pending fetch
  if (this.fetchController) {
    this.fetchController.abort();
  }
}

attributeChangedCallback(name, oldValue, newValue) {
  if (oldValue === newValue) return;

  if (name === 'url' && newValue) {
    this.loadData(newValue);
  } else if (name === 'page-size') {
    this.pageSize = Number(newValue) || 10;
    this.render();
  }
}

async loadData(url) {
  // Cancel previous fetch if any
  if (this.fetchController) {
    this.fetchController.abort();
  }

  this.fetchController = new AbortController();
```

```
    try {
      const response = await fetch(url, {
        signal: this.fetchController.signal
      });
      this.data = await response.json();
      this.render();
    } catch (error) {
      if (error.name !== 'AbortError') {
        console.error('Failed to load data:', error);
      }
    }
  }

  render() {
    // Render logic here
    this.shadowRoot.innerHTML = `
      <style>
        table { width: 100%; border-collapse: collapse; }
        th, td { padding: 8px; text-align: left; border-bottom: 1px solid
        #ddd; }
      </style>
      <table>
        <thead>
          <tr><th>ID</th><th>Name</th><th>Status</th></tr>
        </thead>
        <tbody>
          ${this.data.map(row => `
            <tr>
              <td>${row.id}</td>
              <td>${row.name}</td>
              <td>${row.status}</td>
            </tr>
          `).join('')}
        </tbody>
```

```
      </table>
    `;
  }

  handlePageChange = (event) => {
    this.currentPage = event.detail.page;
    this.render();
  }

  async refresh() {
    const url = this.getAttribute('url');
    if (url) {
      await this.loadData(url);
    }
  }
}

customElements.define('data-table', DataTable);
```

2.7 Summary

This chapter introduced LARC's core concepts:

- **Web Components** provide standard, reusable elements with Custom Elements, Shadow DOM, and Templates
- **The PAN Bus** enables decoupled pub/sub communication between components
- **Event-Driven Architecture** uses DOM events, custom events, and PAN messages for different scenarios
- **State Management** starts simple (local state) and scales to shared, persistent, and server state
- **ES Modules** organize code with standard imports/exports and import maps
- **Component Lifecycle** provides callbacks for creation, connection, attribute changes, and cleanup

In the next chapter, we'll put these concepts into practice by setting up a development environment and building your first LARC application.

2.8 Key Takeaways

- Web Components are browser standards, not framework abstractions
- Shadow DOM provides true style encapsulation
- The PAN bus decouples components through pub/sub messaging
- Use the simplest state management that works, then scale up
- ES Modules and Import Maps replace build-time bundling
- Proper lifecycle management prevents bugs and memory leaks
- Components should be self-contained but composable

2.9 Further Reading

For detailed technical reference: - *Building with LARC* Chapter 2: Core Concepts - Architecture patterns and message flow reference - *Building with LARC* Appendix A: Message Topics Reference - Standard topic conventions - *Building with LARC* Appendix B: Event Envelope Specification - Message format details - *Building with LARC* Appendix F: Glossary - Technical terminology reference

3 Getting Started

Theory is important, but there's no substitute for hands-on experience. In this chapter, you'll set up your development environment and build your first LARC application. By the end, you'll have a working project and understand the basic development workflow.

3.1 Setting Up Your Development Environment

One of LARC's strengths is minimal setup requirements. You don't need complex tooling or configuration—just a browser, a text editor, and a way to serve files.

3.1.1 Requirements

Essential:

- **Modern browser** — Chrome, Firefox, Safari, or Edge (latest version)
- **Text editor** — VS Code, Sublime Text, Atom, or any editor you prefer
- **Local web server** — Python's SimpleHTTPServer, Node's `http-server`, or VS Code's Live Server extension

Optional but Recommended:

- **VS Code** with the LARC extension for snippets and IntelliSense
- **Browser DevTools** familiarity for debugging
- **Git** for version control

3.1.2 Quick Start with create-larc-app

The fastest way to start is using the LARC CLI:

```
# Install globally
npm install -g create-larc-app

# Create a new project
create-larc-app my-first-app

# Start development server
cd my-first-app
larc dev
```

Open `http://localhost:3000` and you'll see your new LARC application running.

3.1.3 Manual Setup (No CLI)

Don't want to install Node.js? You can set up a LARC project manually:

1. Create project structure:

```
mkdir my-first-app
cd my-first-app
mkdir src
mkdir src/components
mkdir public
```

2. 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>My First LARC App</title>

  <!-- Import Map for dependencies -->
  <script type="importmap">
  {
    "imports": {
      "@larcjs/core": "https://cdn.jsdelivr.net/npm/@larcjs/core@2.0.0/
        pan.mjs"
    }
  }
  </script>

  <style>
    body {
      font-family: system-ui, -apple-system, sans-serif;
      margin: 0;
      padding: 20px;
      background: #f5f5f5;
    }
  </style>
</head>
<body>
  <div id="app"></div>

  <script type="module" src="src/app.js"></script>
</body>
</html>
```

3. Create `src/app.js` **:**

```
import { pan } from '@larcjs/core';

// Import your components
import './components/hello-world.js';

// Initialize app
console.log('LARC app initialized');
pan.publish('app.ready');

// Add component to page
document.getElementById('app').innerHTML = '<hello-world></hello-world>';
```

4. Create `src/components/hello-world.js` **:**

```
class HelloWorld extends HTMLElement {
  connectedCallback() {
    this.innerHTML = `
      <div style="
        background: white;
        padding: 40px;
        border-radius: 8px;
        box-shadow: 0 2px 8px rgba(0,0,0,0.1);
        text-align: center;
      ">
        <h1>Hello, LARC!</h1>
        <p>Welcome to your first LARC application.</p>
      </div>
    `;
  }
}

customElements.define('hello-world', HelloWorld);
```

5. Serve the files:

```
# Python 3
python3 -m http.server 3000

# Or Python 2
python -m SimpleHTTPServer 3000

# Or with Node.js
npx http-server -p 3000

# Or use VS Code Live Server extension
# (right-click index.html → "Open with Live Server")
```

Open `http://localhost:3000` and you should see “Hello, LARC!” displayed.

That’s it. No build step. No transpilation. No bundling. Just HTML, CSS, and JavaScript.

3.1.4 Development Tools

3.1.4.1 VS Code Extensions

Install these extensions for the best experience:

LARC Extension:

- Snippets for components and PAN patterns
- IntelliSense for LARC APIs
- Commands for creating components

Install: Search “LARC” in VS Code extensions marketplace

Live Server:

- Auto-reload when files change

- Simple local web server
- Right-click HTML file to start

Install: Search “Live Server” by Ritwick Dey

ES6 String HTML:

- Syntax highlighting for template literals
- Makes component templates more readable

Install: Search “ES6 String HTML”

3.1.4.2 Browser DevTools

Learn these DevTools features for LARC development:

Elements Panel:

```
- Inspect shadow DOM (enable "Show user agent shadow DOM" in settings)
- View Custom Elements with their properties
- Debug CSS in shadow roots
```

Console:

```
- Subscribe to all PAN messages: `pan.subscribe('*', console.log)`
- Test components directly: `document.querySelector('my-component')`
- Check Custom Elements registry: `customElements.get('my-component')`
```

Network Panel:

```
- Verify ES modules load correctly
- Check import map resolution
- Monitor API calls
```

Sources Panel:

```
- Set breakpoints in your source code (no source maps needed!)
- Step through component lifecycle
- Watch variables and state
```

3.2 Your First LARC Application

Let's build something more interesting than "Hello World"—a simple counter application with multiple components communicating via the PAN bus.

3.2.1 Project Goal

We'll create:

```
- A counter display component
- Increment and decrement buttons
- A reset button
- Communication via PAN bus (no prop drilling!)
```

3.2.2 Step 1: Update index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Counter App - LARC</title>

  <script type="importmap">
  {
    "imports": {
      "@larcjs/core": "https://cdn.jsdelivr.net/npm/@larcjs/core@2.0.0/
        pan.mjs"
    }
  }
  </script>

  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    body {
      font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI',
        Roboto, sans-serif;
      background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
      min-height: 100vh;
      display: flex;
      justify-content: center;
      align-items: center;
    }

    #app {
      background: white;
```

```
      padding: 40px;
      border-radius: 16px;
      box-shadow: 0 20px 60px rgba(0, 0, 0, 0.3);
      min-width: 400px;
    }
  </style>
</head>
<body>
  <div id="app">
    <counter-display></counter-display>
    <counter-controls></counter-controls>
  </div>

  <script type="module" src="src/app.js"></script>
</body>
</html>
```

3.2.3 Step 2: Create app.js

```
// src/app.js
import { pan } from '@larcjs/core';

// Import components
import './components/counter-display.js';
import './components/counter-controls.js';

// Initialize application state
let count = 0;

// Listen for increment requests
pan.subscribe('counter.increment', () => {
  count++;
  pan.publish('counter.updated', { count });
});

// Listen for decrement requests
pan.subscribe('counter.decrement', () => {
  count--;
  pan.publish('counter.updated', { count });
});

// Listen for reset requests
pan.subscribe('counter.reset', () => {
  count = 0;
  pan.publish('counter.updated', { count });
});

// Publish initial state
pan.publish('counter.updated', { count });

console.log('Counter app initialized');
```

3.2.4 Step 3: Create counter-display.js

```
// src/components/counter-display.js
import { pan } from '@larcjs/core';

class CounterDisplay extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.count = 0;
  }

  connectedCallback() {
    // Subscribe to count updates
    this.unsubscribe = pan.subscribe('counter.updated', ({ count }) => {
      this.count = count;
      this.render();
    });

    this.render();
  }

  disconnectedCallback() {
    this.unsubscribe();
  }

  render() {
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: block;
          text-align: center;
          margin-bottom: 30px;
        }

        .display {
```

```
          font-size: 72px;
          font-weight: bold;
          color: #667eea;
          margin-bottom: 10px;
          font-variant-numeric: tabular-nums;
        }

        .label {
          font-size: 18px;
          color: #666;
          text-transform: uppercase;
          letter-spacing: 2px;
        }
      </style>

      <div class="display">${this.count}</div>
      <div class="label">Current Count</div>
    `;
  }
}

customElements.define('counter-display', CounterDisplay);
```

3.2.5 Step 4: Create counter-controls.js

```
// src/components/counter-controls.js
import { pan } from '@larcjs/core';

class CounterControls extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
  }

  connectedCallback() {
    this.render();
    this.attachEventListeners();
  }

  attachEventListeners() {
    this.shadowRoot.querySelector('#increment').addEventListener('click', ()
        => {

      pan.publish('counter.increment');
    });

    this.shadowRoot.querySelector('#decrement').addEventListener('click', ()
        => {

      pan.publish('counter.decrement');
    });

    this.shadowRoot.querySelector('#reset').addEventListener('click', ()
        => {

      pan.publish('counter.reset');
    });
  }

  render() {
```

```
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: block;
        }

        .controls {
          display: flex;
          gap: 10px;
          margin-bottom: 15px;
        }

        button {
          flex: 1;
          padding: 15px;
          font-size: 16px;
          font-weight: 600;
          border: none;
          border-radius: 8px;
          cursor: pointer;
          transition: all 0.2s;
        }

        button:hover {
          transform: translateY(-2px);
          box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15);
        }

        button:active {
          transform: translateY(0);
        }

        #increment {
          background: #48bb78;
```

```
      color: white;
    }

    #increment:hover {
      background: #38a169;
    }

    #decrement {
      background: #f56565;
      color: white;
    }

    #decrement:hover {
      background: #e53e3e;
    }

    #reset {
      background: #4a5568;
      color: white;
      width: 100%;
    }

    #reset:hover {
      background: #2d3748;
    }
  </style>

  <div class="controls">
    <button id="decrement">- Decrement</button>
    <button id="increment">+ Increment</button>
  </div>
  <button id="reset">Reset</button>
`;
}
```

```
}

customElements.define('counter-controls', CounterControls);
```

3.2.6 Step 5: Test Your App

Start your local server and open the page. You should see:

```
- A large counter display showing "0"
- Increment and decrement buttons
- A reset button
```

Click the buttons. Notice how:

```
- Components update immediately
- State is managed centrally in `app.js`
- Components don't reference each other directly
- Adding new components is trivial (just subscribe to `counter.updated`)
```

3.2.7 What Just Happened?

Let’s examine the architecture:

Data Flow:

```
User clicks button
     ↓
Controls component publishes event
     ↓
App.js receives event and updates state
     ↓
App.js publishes updated state
     ↓
Display component receives update and re-renders
```

Key Points:

1. **Decoupled Components:** Display and controls don't know about each other
2. **Central State:** State lives in `app.js`, not in components
3. **Pub/Sub Communication:** All communication via PAN bus topics
4. **No Props:** No prop drilling or lifting state up
5. **Easy Testing:** Each component can be tested in isolation

3.3 Project Structure

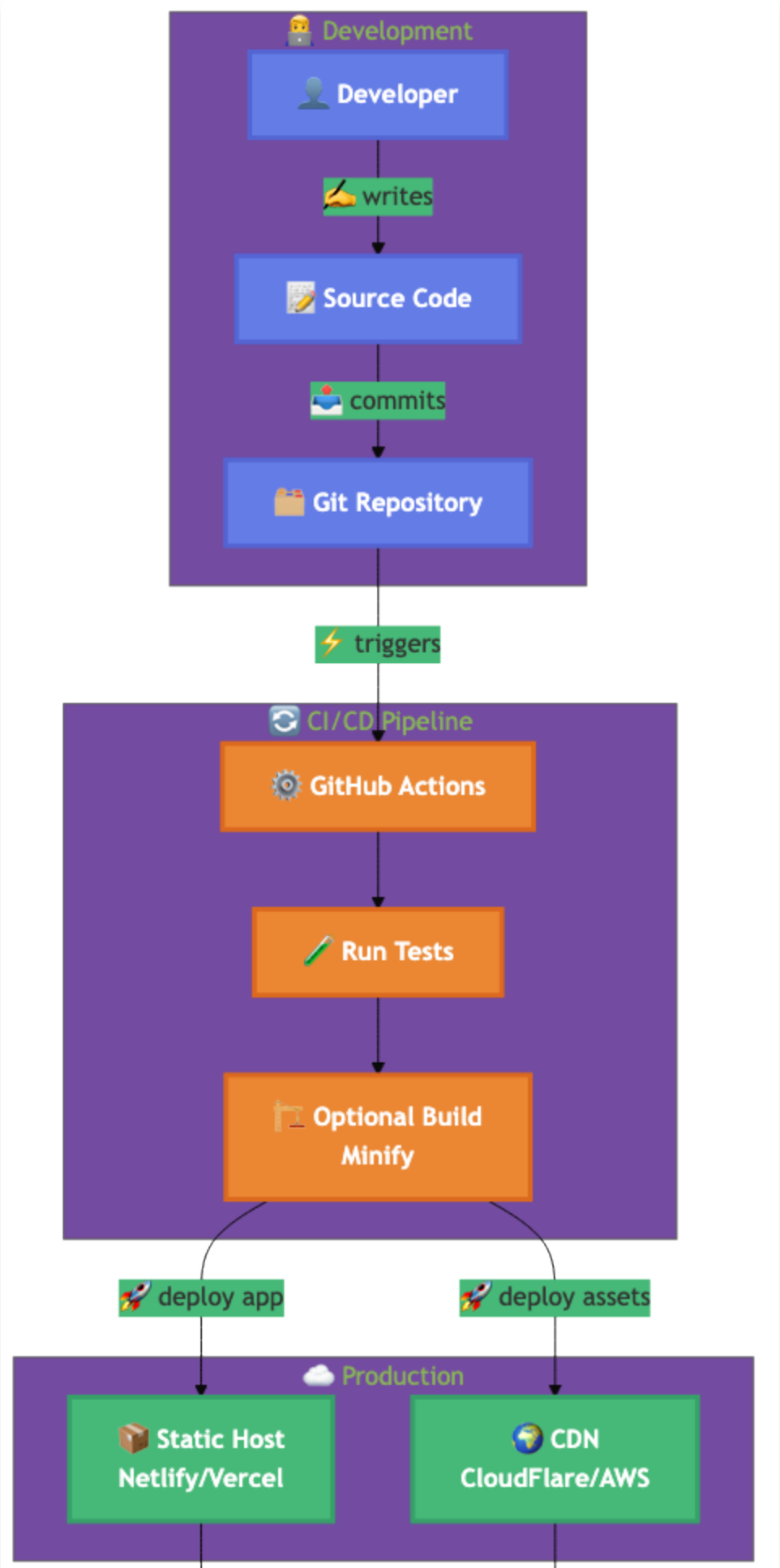
Development
Developer
writes
Source Code
commits
Git Repository
triggers
CI/CD Pipeline
GitHub Actions
Run Tests
Optional Build
Minify
deploy app
deploy assets
Production
Static Host
Netlify/Vercel
CDN
CloudFlare/AWS

Figure 3.1: LARC Deployment Architecture

As your application grows, organization becomes important. Here’s a recommended structure:

```
my-app/
├── index.html              # Entry point
├── larc.config.json        # Optional config
├── src/
|   ├── app.js              # Main application logic
|   ├── components/         # Reusable components
|   |   ├── ui/             # Generic UI components
|   |   |   ├── button.js
|   |   |   ├── card.js
|   |   |   └── modal.js
|   |   ├── features/       # Feature-specific components
|   |   |   ├── user-profile.js
|   |   |   ├── todo-list.js
|   |   |   └── dashboard.js
|   |   └── layout/         # Layout components
|   |       ├── header.js
|   |       ├── sidebar.js
|   |       └── footer.js
|   ├── lib/                # Utilities and helpers
|   |   ├── api.js          # API client
|   |   ├── auth.js         # Authentication
|   |   ├── router.js       # Routing logic
|   |   └── utils.js        # General utilities
|   ├── pages/              # Page-level components
|   |   ├── home.js
|   |   ├── dashboard.js
|   |   └── settings.js
|   └── styles/             # Global styles
|       ├── reset.css
|       ├── variables.css
|       └── utilities.css
├── public/                 # Static assets
|   ├── images/
|   ├── fonts/
```

```
|   └── icons/
└── tests/                  # Test files
    ├── components/
    └── integration/
```

3.3.1 File Organization Principles

Components:

```
- One component per file
- File name matches component name: `user-profile.js` defines `<user-
profile>`
- Keep related components together in subdirectories
```

Lib:

```
- Utilities that don't render UI
- API clients, helpers, formatters
- Pure functions when possible
```

Pages:

```
- Top-level route components
- Compose smaller components
- Handle page-specific logic
```

Styles:

```
- Global styles in `styles/`
- Component-specific styles in Shadow DOM
- CSS custom properties for theming
```

3.4 Import Maps Explained

Figure 3.2: Module Loading with Import Maps

Import Maps are a browser standard that replaces the need for bundlers to resolve module paths.

3.4.1 Basic Import Map

```
<script type="importmap">
{
  "imports": {
    "lodash": "https://cdn.jsdelivr.net/npm/lodash-es@4/lodash.js",
    "dayjs": "https://cdn.jsdelivr.net/npm/dayjs@1/dayjs.min.js"
  }
}
</script>

<script type="module">
  // Use package names instead of URLs
  import _ from 'lodash';
  import dayjs from 'dayjs';

  console.log(dayjs().format('YYYY-MM-DD'));
</script>
```

3.4.2 Path Aliases

Create shortcuts for your own modules:

```
<script type="importmap">
{
  "imports": {
    "@/": "/src/",
    "components/": "/",
    "lib/": "/src/lib/",
    "@larcjs/core": "https://cdn.jsdelivr.net/npm/@larcjs/core@2.0.0/
        pan.mjs"
  }
}
</script>

<script type="module">
  // Instead of: import { api } from '../../../lib/api.js';
  import { api } from 'lib/api.js';

  // Instead of: import Button from '../components/ui/button.js';
  import Button from 'components/ui/button.js';

  // Instead of: import something from '../../../src/utils.js';
  import something from '@/utils.js';
</script>
```

3.4.3 Version Management

Pin dependencies to specific versions:

```
{
  "imports": {
    "@larcjs/core": "https://cdn.jsdelivr.net/npm/@larcjs/core@2.0.0/
        dist/index.js",
    "@larcjs/ui": "https://cdn.jsdelivr.net/npm/@larcjs/ui@2.0.1/dist/
        index.js"
  }
}
```

Or use version ranges for automatic updates:

```
{
  "imports": {
    "@larcjs/core": "https://cdn.jsdelivr.net/npm/@larcjs/core@2.0.0/
        pan.mjs",
    "@larcjs/ui": "https://cdn.jsdelivr.net/npm/@larcjs/ui@2/dist/
        index.js"
  }
}
```

3.4.4 Multiple CDNs

Add fallbacks for reliability:

```
{
  "imports": {
    "react": "https://esm.sh/react@18",
    "react-fallback": "https://cdn.skypack.dev/react@18"
  }
}
```

Then in code:

```
let React;
try {
  React = await import('react');
} catch {
  React = await import('react-fallback');
}
```

3.4.5 Development vs Production

Use different import maps for different environments:

development.importmap.json:

```
{
  "imports": {
    "@larcjs/core": "/node_modules/@larcjs/core/dist/index.js",
    "app/": "/src/"
  }
}
```

production.importmap.json:

```
{
  "imports": {
    "@larcjs/core": "https://cdn.jsdelivr.net/npm/@larcjs/core@2.0.0/
        dist/index.js",
    "app/": "/assets/js/"
  }
}
```

Load the appropriate map:

```
<script type="importmap" src="/config/production.importmap.json"></script>
```

3.5 Development Workflow

3.5.1 Daily Development

A typical development session:

1. Start dev server:

```
larc dev
```

This starts a local server with hot reload.

2. Edit files: Open your editor and make changes. The browser automatically reloads when you save.

3. Check the console: Open browser DevTools and check for errors or warnings.

4. Test in browser: Interact with your app, verify behavior, check responsive design.

5. Debug as needed: Set breakpoints, inspect elements, monitor network requests.

6. Repeat: The edit-refresh cycle is instant with no build step.

3.5.2 Debugging Tips

Log all PAN messages:

```
pan.subscribe('*', (topic, data) => {
  console.log(`[PAN] ${topic}:`, data);
});
```

Inspect custom elements:

```
// Get element
const el = document.querySelector('my-component');

// Check if defined
console.log(customElements.get('my-component'));

// Access shadow root
console.log(el.shadowRoot);

// Call methods directly
el.someMethod();
```

Monitor attribute changes:

```
// Create observer
const observer = new MutationObserver((mutations) => {
  mutations.forEach(mutation => {
    console.log('Attribute changed:', mutation.attributeName);
  });
});

// Watch element
observer.observe(element, { attributes: true });
```

3.5.3 Testing

Run tests without a build step:

```html
<!-- tests/counter.test.html -->
<!DOCTYPE html>
<html>
<head>
  <title>Counter Tests</title>
  <script type="importmap">
  {
    "imports": {
      "@larcjs/core": "../node_modules/@larcjs/core/dist/index.js"
    }
  }
  </script>
</head>
<body>
  <div id="test-container"></div>

  <script type="module">
    import { pan } from '@larcjs/core';
    import '../counter-display.js';

    // Simple test framework
    function test(name, fn) {
      try {
        fn();
        console.log(`✓ ${name}`);
      } catch (error) {
        console.error(`✗ ${name}:`, error);
      }
    }

    function assert(condition, message) {
      if (!condition) throw new Error(message || 'Assertion failed');
    }
```

```
    // Tests
    test('counter-display renders initial count', () => {
      const el = document.createElement('counter-display');
      document.getElementById('test-container').appendChild(el);

      const display = el.shadowRoot.querySelector('.display');
      assert(display.textContent === '0', 'Initial count should be 0');

      el.remove();
    });

    test('counter-display updates on PAN message', async () => {
      const el = document.createElement('counter-display');
      document.getElementById('test-container').appendChild(el);

      // Wait for component to connect
      await new Promise(resolve => setTimeout(resolve, 10));

      // Publish update
      pan.publish('counter.updated', { count: 42 });

      // Wait for render
      await new Promise(resolve => setTimeout(resolve, 10));

      const display = el.shadowRoot.querySelector('.display');
      assert(display.textContent === '42', 'Count should update to 42');

      el.remove();
    });

    console.log('All tests complete');
  </script>
</body>
</html>
```

Open `tests/counter.test.html` in your browser to run tests.

3.6 Common Patterns

3.6.1 Pattern 1: Loading States

```
class DataComponent extends HTMLElement {
  async connectedCallback() {
    this.render({ loading: true });

    try {
      const data = await this.fetchData();
      this.render({ data });
    } catch (error) {
      this.render({ error: error.message });
    }
  }

  render(state) {
    if (state.loading) {
      this.innerHTML = '<loading-spinner></loading-spinner>';
    } else if (state.error) {
      this.innerHTML = `<error-message>${state.error}</error-message>`;
    } else {
      this.innerHTML = `<data-display .data="${state.data}"></data-
        display>`;
    }
  }
}
```

3.6.2 Pattern 2: Form Handling

```
class LoginForm extends HTMLElement {
  connectedCallback() {
    this.innerHTML = `
      <form>
        <input type="email" name="email" required>
        <input type="password" name="password" required>
        <button type="submit">Login</button>
      </form>
    `;

    this.querySelector('form').addEventListener('submit', async (e) => {
      e.preventDefault();

      const formData = new FormData(e.target);
      const data = Object.fromEntries(formData);

      pan.publish('auth.login', data);
    });
  }
}
```

3.6.3 Pattern 3: Conditional Rendering

```
class UserMenu extends HTMLElement {
  constructor() {
    super();
    this.user = null;
  }

  connectedCallback() {
    pan.subscribe('auth.user.changed', ({ user }) => {
      this.user = user;
      this.render();
    });

    this.render();
  }

  render() {
    if (this.user) {
      this.innerHTML = `
        <div class="logged-in">
          <span>Hello, ${this.user.name}</span>
          <button id="logout">Logout</button>
        </div>
      `;

      this.querySelector('#logout').addEventListener('click', () => {
        pan.publish('auth.logout');
      });
    } else {
      this.innerHTML = `
        <button id="login">Login</button>
      `;

      this.querySelector('#login').addEventListener('click', () => {
        pan.publish('app.navigate', { path: '/login' });
```

```
      });
    }
  }
}
```

3.6.4 Pattern 4: Lists and Iteration

```
class TodoList extends HTMLElement {
  constructor() {
    super();
    this.todos = [];
  }

  connectedCallback() {
    pan.subscribe('todos.updated', ({ todos }) => {
      this.todos = todos;
      this.render();
    });

    this.render();
  }

  render() {
    this.innerHTML = `
      <ul>
        ${this.todos.map(todo => `
          <li>
            <input type="checkbox"
                   ${todo.completed ? 'checked' : ''}
                   data-id="${todo.id}">
            <span class="${todo.completed ? 'completed' : ''}">
              ${todo.text}
            </span>
          </li>
        `).join('')}
      </ul>
    `;

    // Attach event listeners after rendering
    this.querySelectorAll('input[type="checkbox"]').forEach(checkbox => {
      checkbox.addEventListener('change', (e) => {
```

```
        const id = e.target.dataset.id;
        pan.publish('todos.toggle', { id });
      });
    });
  }
}
```

3.7 Summary

In this chapter, you:

```
- Set up a LARC development environment (CLI or manual)
- Built your first multi-component application
- Learned project structure best practices
- Mastered Import Maps for dependency management
- Established an efficient development workflow
- Explored common component patterns
```

You now have a solid foundation for building LARC applications. The next chapter dives deeper into creating sophisticated Web Components with proper lifecycle management, styling, and interactivity.

3.8 Exercises

1. Enhance the Counter App:

```
- Add a history component that shows past values
- Add increment/decrement by custom amounts
- Persist count to localStorage
```

2. Build a Todo List:

```
- Add/remove todos
- Mark as complete/incomplete
- Filter by status (all/active/completed)
- Use PAN bus for state management
```

3. Create a Theme Switcher:

```
- Light/dark theme toggle
- Publish theme changes via PAN
- Multiple components respond to theme changes
- Persist theme preference
```

4. Experiment with Import Maps:

```
- Try different CDNs (jsDelivr, unpkg, esm.sh)
- Add path aliases for your components
- Import an external library (lodash, dayjs, etc.)
```

Take your time with these exercises. Understanding these patterns now will make the rest of the book much easier.

3.9 Further Reading

For detailed API reference and configuration options: - *Building with LARC* Chapter 3: Getting Started - Complete installation options and troubleshooting - *Building with LARC* Chapter 2: Core Concepts - Architecture and messaging patterns reference - *Building with LARC* Appendix C: Configuration Options - All configuration parameters

4 Creating Web Components

Now that you've built your first LARC application, it's time to master the art of creating robust, reusable Web Components. This chapter covers everything from basic component anatomy to advanced patterns like composition, slots, and performance optimization.

By the end of this chapter, you'll be able to build production-quality components that are maintainable, testable, and performant.

4.1 Anatomy of a LARC Component

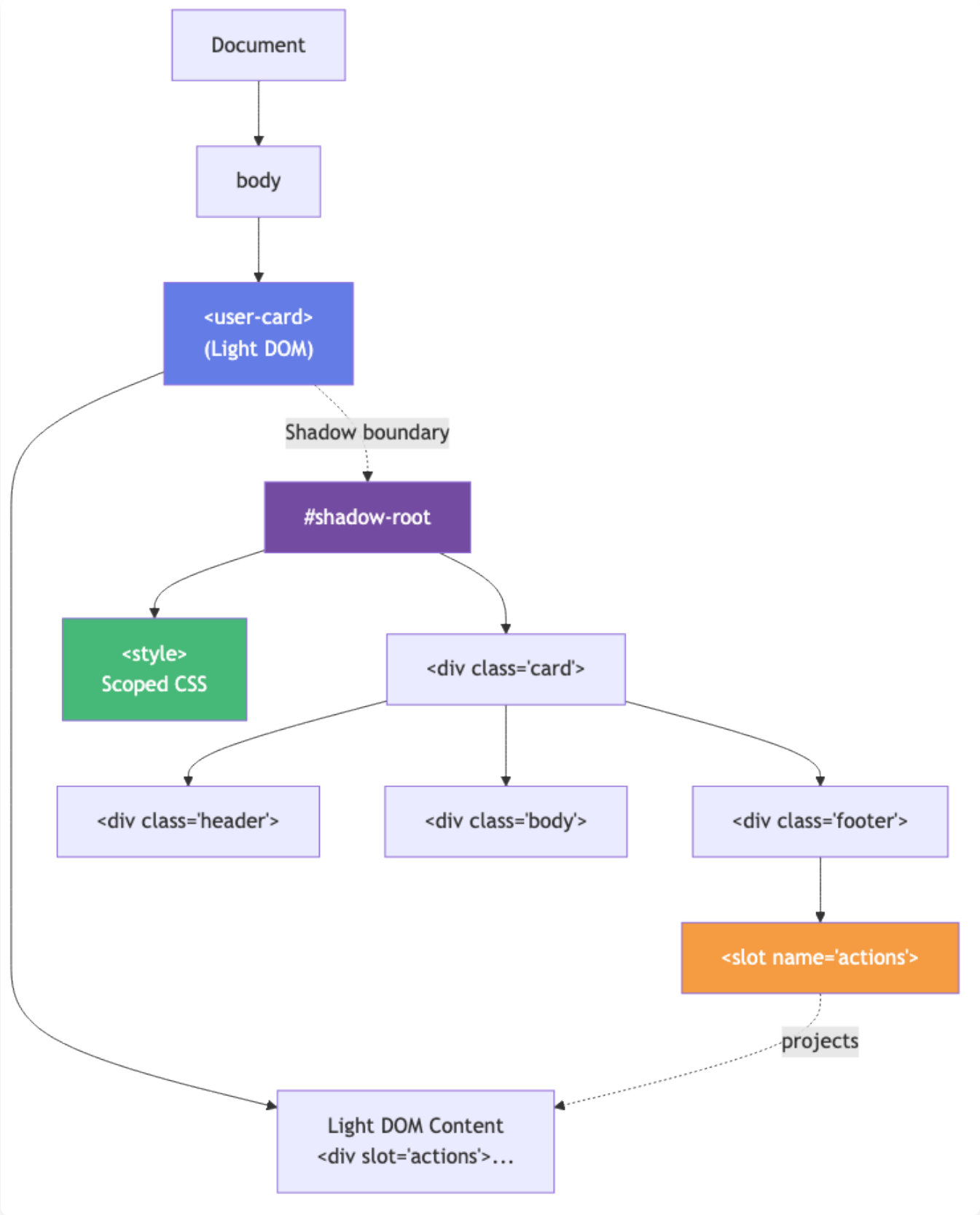

Figure 4.1: Component Lifecycle Flow

Let’s dissect a well-structured LARC component to understand its parts:

```
// Import dependencies
import { pan } from '@larcjs/core';
import { formatDate } from '../lib/utils.js';

/**
 * A card component for displaying user information.
 *
 * @element user-card
 *
 * @attr {string} user-id - The ID of the user to display
 * @attr {boolean} compact - Display in compact mode
 *
 * @fires user-selected - Dispatched when card is clicked
 *
 * @slot - Default slot for additional content
 * @slot actions - Slot for action buttons
 */
class UserCard extends HTMLElement {
  // 1. Define observed attributes
  static get observedAttributes() {
    return ['user-id', 'compact'];
  }

  // 2. Constructor - initialize instance
  constructor() {
    super();

    // Attach shadow DOM
    this.attachShadow({ mode: 'open' });

    // Initialize private state
    this._user = null;
    this._loading = false;
    this._error = null;
```

```
    // Bind event handlers
    this.handleClick = this.handleClick.bind(this);
  }

  // 3. Lifecycle: connected to DOM
  connectedCallback() {
    this.render();

    // Load user data if ID is provided
    const userId = this.getAttribute('user-id');
    if (userId) {
      this.loadUser(userId);
    }

    // Subscribe to PAN events
    this.unsubscribe = pan.subscribe('user.updated',
        this.handleUserUpdate);

    // Add event listeners
    this.shadowRoot.addEventListener('click', this.handleClick);
  }

  // 4. Lifecycle: disconnected from DOM
  disconnectedCallback() {
    // Clean up subscriptions
    if (this.unsubscribe) {
      this.unsubscribe();
    }

    // Remove event listeners
    this.shadowRoot.removeEventListener('click', this.handleClick);
  }

  // 5. Lifecycle: attributes changed
```

```
  attributeChangedCallback(name, oldValue, newValue) {
    if (oldValue === newValue) return;

    if (name === 'user-id' && newValue) {
      this.loadUser(newValue);
    } else if (name === 'compact') {
      this.render();
    }
  }

  // 6. Public properties with getters/setters
  get user() {
    return this._user;
  }

  set user(value) {
    this._user = value;
    this.render();
  }

  get loading() {
    return this._loading;
  }

  // 7. Public methods
  async loadUser(userId) {
    this._loading = true;
    this._error = null;
    this.render();

    try {
      const response = await fetch(`/api/users/${userId}`);
      if (!response.ok) throw new Error('Failed to load user');
```

```
      this._user = await response.json();
      this._loading = false;
      this.render();
    } catch (error) {
      this._error = error.message;
      this._loading = false;
      this.render();
    }
  }

  refresh() {
    const userId = this.getAttribute('user-id');
    if (userId) {
      this.loadUser(userId);
    }
  }

  // 8. Private methods
  handleClick(event) {
    if (!this._user) return;

    this.dispatchEvent(new CustomEvent('user-selected', {
      detail: { user: this._user },
      bubbles: true,
      composed: true
    }));
  }

  handleUserUpdate = (data) => {
    if (data.userId === this.getAttribute('user-id')) {
      this._user = data.user;
      this.render();
    }
  }
```

```
// 9. Render method
render() {
  const compact = this.hasAttribute('compact');

  if (this._loading) {
    this.shadowRoot.innerHTML = this.renderLoading();
    return;
  }

  if (this._error) {
    this.shadowRoot.innerHTML = this.renderError();
    return;
  }

  if (!this._user) {
    this.shadowRoot.innerHTML = this.renderEmpty();
    return;
  }

  this.shadowRoot.innerHTML = compact
    ? this.renderCompact()
    : this.renderFull();
}

renderLoading() {
  return `
    <style>${this.styles()}</style>
    <div class="card loading">
      <div class="spinner"></div>
      <p>Loading...</p>
    </div>
  `;
}
```

```
  renderError() {
    return `
      <style>${this.styles()}</style>
      <div class="card error">
        <p class="error-message">${this._error}</p>
        <button class="retry">Retry</button>
      </div>
    `;
  }

  renderEmpty() {
    return `
      <style>${this.styles()}</style>
      <div class="card empty">
        <p>No user data</p>
      </div>
    `;
  }

  renderCompact() {
    return `
      <style>${this.styles()}</style>
      <div class="card compact">
        <img src="${this._user.avatar}" alt="${this._user.name}">
        <div class="info">
          <h3>${this._user.name}</h3>
          <slot name="actions"></slot>
        </div>
      </div>
    `;
  }

  renderFull() {
```

```
    return `
      <style>${this.styles()}</style>
      <div class="card">
        <div class="header">
          <img src="${this._user.avatar}" alt="${this._user.name}"
       class="avatar">
          <div class="header-content">
            <h2>${this._user.name}</h2>
            <p class="email">${this._user.email}</p>
          </div>
        </div>
        <div class="body">
          <p class="bio">${this._user.bio || 'No bio available'}</p>
          <div class="meta">
            <span>Joined ${formatDate(this._user.createdAt)}</span>
          </div>
          <slot></slot>
        </div>
        <div class="footer">
          <slot name="actions"></slot>
        </div>
      </div>
    `;
  }

  // 10. Styles
  styles() {
    return `
      :host {
        display: block;
        cursor: pointer;
      }

      .card {
        background: white;
```

```
  border-radius: 8px;
  box-shadow: 0 2px 4px rgba(0,0,0,0.1);
  padding: 16px;
  transition: box-shadow 0.2s;
}

.card:hover {
  box-shadow: 0 4px 12px rgba(0,0,0,0.15);
}

.header {
  display: flex;
  gap: 12px;
  margin-bottom: 16px;
}

.avatar {
  width: 48px;
  height: 48px;
  border-radius: 50%;
  object-fit: cover;
}

h2 {
  margin: 0;
  font-size: 18px;
  color: #333;
}

.email {
  margin: 4px 0 0 0;
  font-size: 14px;
  color: #666;
}
```

```
.bio {
  color: #444;
  line-height: 1.5;
}

.meta {
  font-size: 12px;
  color: #999;
  margin-top: 12px;
}

.loading, .error, .empty {
  text-align: center;
  padding: 40px 20px;
  color: #666;
}

.spinner {
  border: 3px solid #f3f3f3;
  border-top: 3px solid #667eea;
  border-radius: 50%;
  width: 40px;
  height: 40px;
  animation: spin 1s linear infinite;
  margin: 0 auto 16px;
}

@keyframes spin {
  to { transform: rotate(360deg); }
}

.error-message {
  color: #e53e3e;
```

```
      }

      .compact {
        display: flex;
        align-items: center;
        gap: 12px;
        padding: 12px;
      }

      .compact img {
        width: 40px;
        height: 40px;
        border-radius: 50%;
      }

      .compact h3 {
        margin: 0;
        font-size: 14px;
      }
    `;
  }
}

// 11. Register the custom element
customElements.define('user-card', UserCard);

// 12. Export for use in other modules
export default UserCard;
```

4.1.1 Component Structure Breakdown

1. Documentation:

- JSDoc comments explain usage

- Attribute, property, event, and slot documentation
- Helps other developers understand the component

2. Static Properties:

- `observedAttributes` defines which attributes trigger `attributeChangedCallback`
- Keep this list minimal for performance

3. Constructor:

- Initialize instance variables
- Attach shadow DOM
- Bind methods (for event handlers)
- Don't access attributes or DOM here

4. Lifecycle Methods:

- `connectedCallback`: Setup when added to DOM
- `disconnectedCallback`: Cleanup when removed
- `attributeChangedCallback`: Respond to attribute changes

5. Properties:

- Use private fields (`_user`) for internal state
- Provide getters/setters for public API
- Setters can trigger re-renders

6. Methods:

- Public methods for external use
- Private methods (conventionally start with `_` or use `#` private fields)
- Keep methods focused and single-purpose

7. Rendering:

- Separate render logic from state management
- Multiple render methods for different states
- Extract styles to a separate method

4.2 Shadow DOM Deep Dive

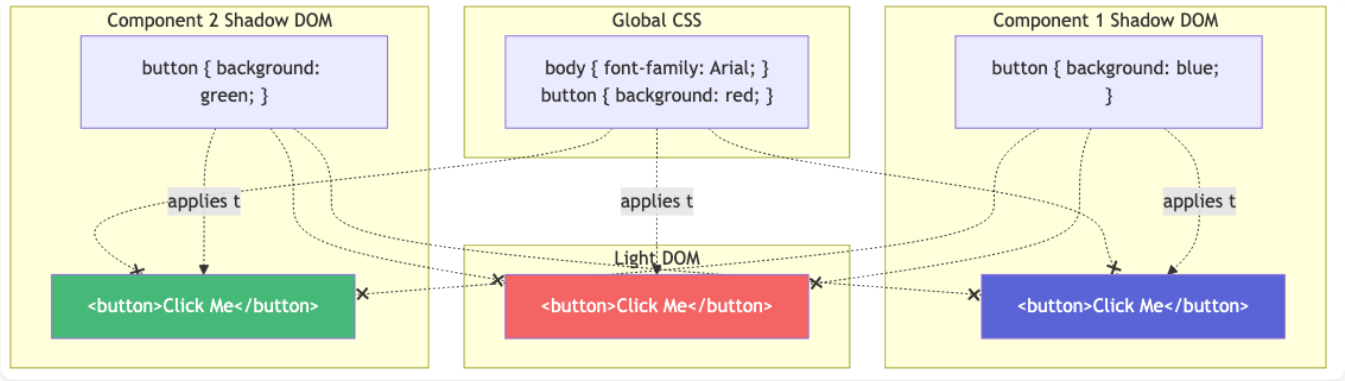

Figure 4.2: Slots and Content Projection

Shadow DOM is one of the most powerful features of Web Components. It provides true encapsulation for both markup and styles.

4.2.1 Creating Shadow DOM

```
class MyComponent extends HTMLElement {
  constructor() {
    super();

    // Create shadow root
    this.attachShadow({ mode: 'open' });

    // mode: 'open' - shadow root accessible via element.shadowRoot
    // mode: 'closed' - shadow root not accessible (rarely used)
  }
}
```

4.2.2 Shadow DOM vs Light DOM

```
<my-component>
  <!-- This is Light DOM (regular DOM) -->
  <p>Visible content</p>
</my-component>

<script>
  class MyComponent extends HTMLElement {
    constructor() {
      super();
      this.attachShadow({ mode: 'open' });

      // This is Shadow DOM
      this.shadowRoot.innerHTML = `
        <div class="shadow-content">
          <h2>Shadow DOM Content</h2>
          <slot></slot>
        </div>
      `;
    }
  }

  customElements.define('my-component', MyComponent);
</script>
```

Result:

- Light DOM (`<p>Visible content</p>`) is projected into the `<slot>`
- Shadow DOM provides the structure and styling
- Styles in shadow DOM don’t leak out
- Styles from light DOM don’t leak in

4.2.3 Style Encapsulation

```
class StyledButton extends HTMLElement {
  connectedCallback() {
    this.attachShadow({ mode: 'open' });

    this.shadowRoot.innerHTML = `
      <style>
        /* These styles only affect this component */
        button {
          background: blue;
          color: white;
          border: none;
          padding: 10px 20px;
          border-radius: 4px;
          cursor: pointer;
        }

        button:hover {
          background: darkblue;
        }
      </style>
      <button><slot></slot></button>
    `;
  }
}
```

Key Points:

- Styles inside shadow DOM are scoped
- No conflicts with global styles
- No CSS class name collisions
- True component encapsulation

4.2.4 The :host Selector

Style the component itself:

```
:host {
  display: block;
  margin: 16px 0;
}

/* Style host when it has a class */
:host(.highlighted) {
  border: 2px solid gold;
}

/* Style host when it has an attribute */
:host([disabled]) {
  opacity: 0.5;
  pointer-events: none;
}

/* Style host in specific contexts */
:host-context(.dark-theme) {
  background: #333;
  color: white;
}
```

4.2.5 CSS Custom Properties (Variables)

CSS variables pierce the shadow DOM boundary:

```
// Component defines and uses variables
class ThemedCard extends HTMLElement {
  connectedCallback() {
    this.attachShadow({ mode: 'open' });

    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: block;
          background: var(--card-bg, white);
          color: var(--card-text, black);
          border: 1px solid var(--card-border, #ddd);
          border-radius: var(--card-radius, 8px);
          padding: var(--card-padding, 16px);
        }
      </style>
      <slot></slot>
    `;
  }
}
```

Usage:

```
<style>
  /* Override component variables from outside */
  themed-card {
    --card-bg: #f0f0f0;
    --card-text: #333;
    --card-border: #ccc;
    --card-radius: 12px;
  }

  themed-card.dark {
    --card-bg: #333;
    --card-text: #fff;
    --card-border: #555;
  }
</style>

<themed-card>Normal theme</themed-card>
<themed-card class="dark">Dark theme</themed-card>
```

This pattern allows theming while maintaining encapsulation.

4.2.6 Parts and ::part()

Expose specific shadow DOM elements for styling:

```
class FancyButton extends HTMLElement {
  connectedCallback() {
    this.attachShadow({ mode: 'open' });

    this.shadowRoot.innerHTML = `
      <style>
        button { /* default styles */ }
        .icon { /* icon styles */ }
      </style>
      <button part="button">
        <span part="icon" class="icon">→</span>
        <slot></slot>
      </button>
    `;
  }
}
```

Style from outside:

```
fancy-button::part(button) {
  background: linear-gradient(135deg, #667eea, #764ba2);
}

fancy-button::part(icon) {
  color: gold;
}
```

This gives consumers more control while maintaining encapsulation.

4.3 Attributes and Properties

Understanding the difference between attributes and properties is crucial for component

design.

4.3.1 Attributes vs Properties

Attributes:

- HTML attributes (`<my-el foo="bar">`)
- Always strings
- Visible in HTML
- Trigger `attributeChangedCallback`

Properties:

- JavaScript properties (`element.foo = 123`)
- Any type (string, number, object, etc.)
- Not visible in HTML
- Direct access, no callback

4.3.2 Reflecting Properties to Attributes

```
class ToggleButton extends HTMLElement {
  static get observedAttributes() {
    return ['checked'];
  }

  constructor() {
    super();
    this._checked = false;
  }

  // Property getter
  get checked() {
    return this._checked;
  }

  // Property setter - reflects to attribute
  set checked(value) {
    const isChecked = Boolean(value);

    if (isChecked) {
      this.setAttribute('checked', '');
    } else {
      this.removeAttribute('checked');
    }
  }

  // Attribute changed - updates property
  attributeChangedCallback(name, oldValue, newValue) {
    if (name === 'checked') {
      this._checked = newValue !== null;
      this.render();
    }
  }
```

```
  render() {
    this.innerHTML = `
      <button class="${this._checked ? 'checked' : ''}">
        ${this._checked ? '✓' : '○'}
      </button>
    `;
  }
}
```

Usage:

```
<!-- Set via attribute -->
<toggle-button checked></toggle-button>

<script>
  const toggle = document.querySelector('toggle-button');

  // Set via property
  toggle.checked = true;

  // Get property
  console.log(toggle.checked); // true

  // Check attribute
  console.log(toggle.hasAttribute('checked')); // true
</script>
```

4.3.3 When to Use Each

Use Attributes for:

- Simple configuration (strings, numbers, booleans)
- Values that should be visible in HTML
- Initial configuration from HTML

- Values that need to work with CSS selectors

Use Properties for:

- Complex data (objects, arrays, functions)
- Data that changes frequently
- Large data that shouldn't serialize to HTML
- Callback functions

4.3.4 Type Conversion

Attributes are always strings, so convert appropriately:

```
attributeChangedCallback(name, oldValue, newValue) {
  if (name === 'count') {
    this._count = Number(newValue) || 0;
  } else if (name === 'enabled') {
    this._enabled = newValue !== null; // Boolean attribute
  } else if (name === 'options') {
    try {
      this._options = JSON.parse(newValue);
    } catch {
      this._options = {};
    }
  }
}
```

4.3.5 Boolean Attributes

Follow HTML conventions:

```
// Boolean attribute: presence = true, absence = false
if (this.hasAttribute('disabled')) {
  // Is disabled
}

// Set boolean attribute
this.setAttribute('disabled', ''); // value doesn't matter

// Remove boolean attribute
this.removeAttribute('disabled');
```

4.4 Component Styling

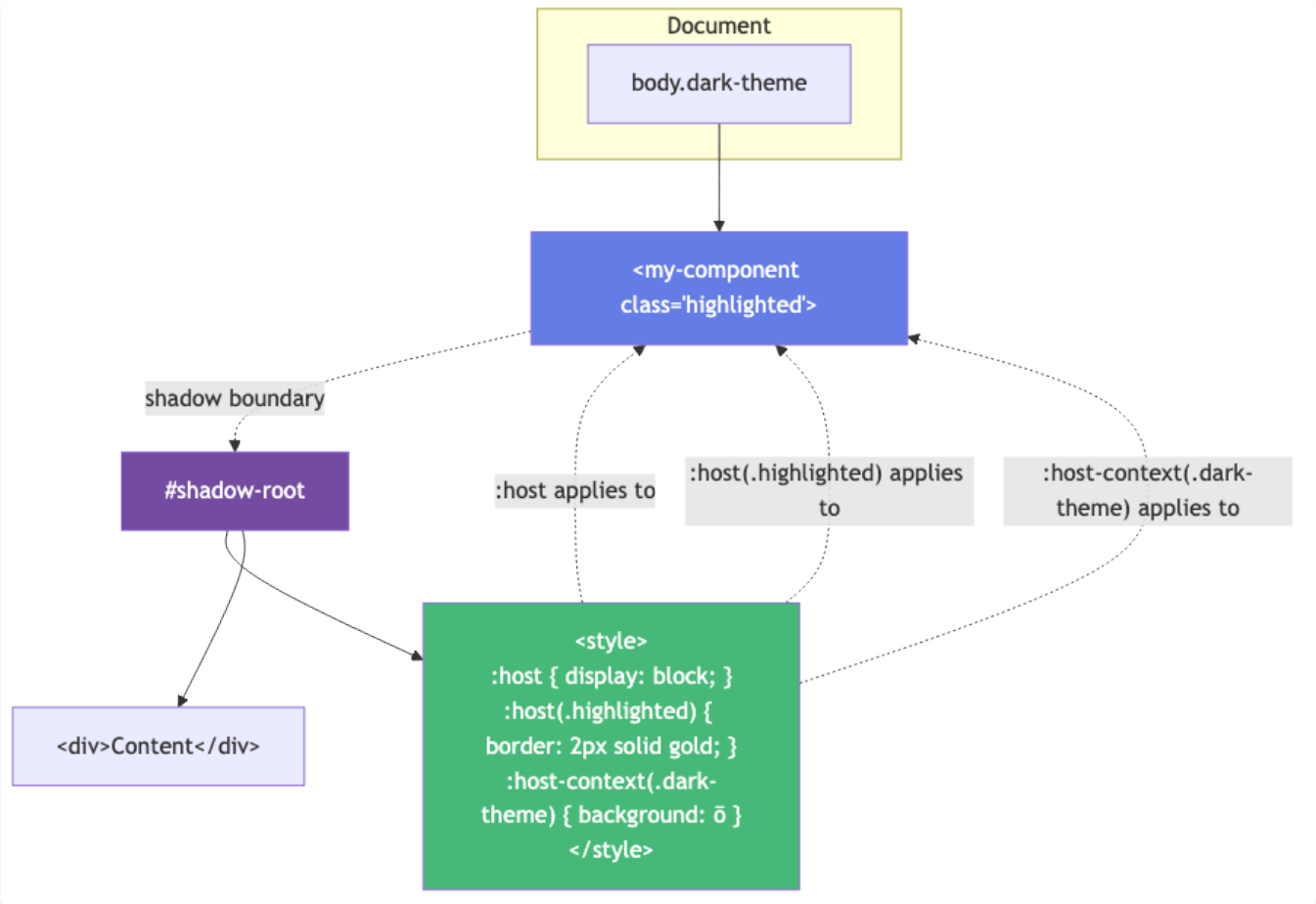

Figure 4.3: CSS Encapsulation with Shadow DOM

4.4.1 Internal Styles

Most styles should be in shadow DOM:

```
styles() {
  return `
    :host {
      display: block;
    }

    .container {
      padding: 16px;
    }

    /* All your component styles */
  `;
}
```

4.4.2 External Stylesheets

For larger components, link external styles:

```
connectedCallback() {
  this.attachShadow({ mode: 'open' });

  this.shadowRoot.innerHTML = `
    <link rel="stylesheet" href="/styles/components/user-card.css">
    <div class="user-card">
      <!-- content -->
    </div>
  `;
}
```

4.4.3 Adoptable Stylesheets

Share styles between component instances:

```
// Create shared stylesheet once
const sheet = new CSSStyleSheet();
sheet.replaceSync(`
  .card {
    padding: 16px;
    border-radius: 8px;
    background: white;
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);
  }
`);

class CardComponent extends HTMLElement {
  connectedCallback() {
    this.attachShadow({ mode: 'open' });

    // Adopt shared stylesheet (very fast)
    this.shadowRoot.adoptedStyleSheets = [sheet];

    this.shadowRoot.innerHTML = `
      <div class="card">
        <slot></slot>
      </div>
    `;
  }
}
```

Benefits:

- Styles parsed once, shared across instances
- Better performance with many components
- Modify shared styles dynamically

4.4.4 Theming Strategies

Strategy 1: CSS Custom Properties

```
class ThemedComponent extends HTMLElement {
  styles() {
    return `
      :host {
        --primary-color: var(--app-primary, #667eea);
        --background: var(--app-bg, white);
        --text: var(--app-text, #333);
      }

      .content {
        background: var(--background);
        color: var(--text);
      }

      button {
        background: var(--primary-color);
      }
    `;
  }
}
```

Strategy 2: Class-Based Themes

```
class ThemeAwareComponent extends HTMLElement {
  connectedCallback() {
    // Observe theme changes on documentElement
    const observer = new MutationObserver(() => {
      this.updateTheme();
    });

    observer.observe(document.documentElement, {
      attributes: true,
      attributeFilter: ['data-theme']
    });

    this.updateTheme();
  }

  updateTheme() {
    const theme = document.documentElement.dataset.theme || 'light';
    this.setAttribute('theme', theme);
  }

  styles() {
    return `
      :host([theme="light"]) {
        background: white;
        color: black;
      }

      :host([theme="dark"]) {
        background: #333;
        color: white;
      }
    `;
  }
}
```

Strategy 3: PAN-Based Themes

```
import { pan } from '@larcjs/core';

class PanThemedComponent extends HTMLElement {
  connectedCallback() {
    this.unsubscribe = pan.subscribe('app.theme.changed', ({ theme }) =>
        {
      this.applyTheme(theme);
    });

    // Request current theme
    pan.request('app.theme.get').then(theme => {
      this.applyTheme(theme);
    });
  }

  applyTheme(theme) {
    this.setAttribute('data-theme', theme);
  }
}
```

4.5 Lifecycle Methods (Advanced Patterns)

4.5.1 Deferred Rendering

Wait for dependencies before rendering:

```
class DataDisplay extends HTMLElement {
  async connectedCallback() {
    // Wait for dependencies to load
    await customElements.whenDefined('loading-spinner');
    await customElements.whenDefined('error-message');

    // Now render
    this.render();
  }
}
```

4.5.2 Preventing Memory Leaks

```
class WebSocketComponent extends HTMLElement {
  connectedCallback() {
    this.ws = new WebSocket('wss://api.example.com');

    this.ws.onmessage = (event) => {
      this.handleMessage(event.data);
    };

    this.ws.onerror = (error) => {
      console.error('WebSocket error:', error);
    };
  }

  disconnectedCallback() {
    // Clean up WebSocket connection
    if (this.ws) {
      this.ws.close();
      this.ws = null;
    }
  }
}
```

4.5.3 Handling Rapid Reconnection

Components can be disconnected and reconnected quickly:

```
class RobustComponent extends HTMLElement {
  connectedCallback() {
    // Might be called multiple times
    // Use a guard to prevent duplicate setup
    if (this._initialized) {
      return;
    }

    this._initialized = true;
    this.setup();
  }

  disconnectedCallback() {
    // Use setTimeout to debounce
    this._cleanupTimer = setTimeout(() => {
      this.cleanup();
      this._initialized = false;
    }, 100);
  }

  connectedCallback() {
    // Cancel cleanup if reconnected quickly
    if (this._cleanupTimer) {
      clearTimeout(this._cleanupTimer);
      this._cleanupTimer = null;
    }

    if (this._initialized) {
      return;
    }

    this._initialized = true;
    this.setup();
  }
```

```
}
```

4.6 Testing Components

4.6.1 Unit Testing

Test components in isolation:

```
// tests/user-card.test.js
import { expect } from '@open-wc/testing';
import '../user-card.js';

describe('UserCard', () => {
  let element;

  beforeEach(() => {
    element = document.createElement('user-card');
    document.body.appendChild(element);
  });

  afterEach(() => {
    element.remove();
  });

  it('renders empty state by default', () => {
    const emptyText = element.shadowRoot.querySelector('.empty');
    expect(emptyText).to.exist;
  });

  it('loads user when user-id attribute is set', async () => {
    // Mock fetch
    global.fetch = async () => ({
      ok: true,
      json: async () => ({ id: 1, name: 'John Doe', email:
        'john@example.com' })
    });

    element.setAttribute('user-id', '1');

    // Wait for async operations
    await new Promise(resolve => setTimeout(resolve, 100));
```

```
    const name = element.shadowRoot.querySelector('h2');
    expect(name.textContent).to.equal('John Doe');
  });

  it('handles loading state', async () => {
    element.setAttribute('user-id', '1');

    const spinner = element.shadowRoot.querySelector('.spinner');
    expect(spinner).to.exist;
  });

  it('dispatches user-selected event on click', async () => {
    element._user = { id: 1, name: 'John' };
    element.render();

    let eventData = null;
    element.addEventListener('user-selected', (e) => {
      eventData = e.detail;
    });

    element.shadowRoot.querySelector('.card').click();

    expect(eventData).to.deep.equal({ user: { id: 1, name: 'John' } });
  });
});
```

4.6.2 Integration Testing

Test components working together:

```
// tests/counter-integration.test.js
describe('Counter Integration', () => {
  beforeEach(() => {
    document.body.innerHTML = `
      <counter-display></counter-display>
      <counter-controls></counter-controls>
    `;
  });

  it('updates display when controls are clicked', async () => {
    const display = document.querySelector('counter-display');
    const controls = document.querySelector('counter-controls');

    const incrementBtn = controls.shadowRoot.querySelector('#increment');
    incrementBtn.click();

    await new Promise(resolve => setTimeout(resolve, 50));

    const displayValue =
        display.shadowRoot.querySelector('.display').textContent;
    expect(displayValue).to.equal('1');
  });
});
```

4.6.3 Visual Regression Testing

Catch visual bugs:

```
// tests/visual.test.js
import puppeteer from 'puppeteer';
import pixelmatch from 'pixelmatch';

describe('Visual Regression', () => {
  let browser, page;

  beforeAll(async () => {
    browser = await puppeteer.launch();
    page = await browser.newPage();
  });

  afterAll(async () => {
    await browser.close();
  });

  it('user-card matches snapshot', async () => {
    await page.goto('http://localhost:3000/tests/user-card.html');

    const screenshot = await page.screenshot({ fullPage: true });
    const baseline = fs.readFileSync('tests/snapshots/user-card.png');

    const diff = pixelmatch(screenshot, baseline, null, 800, 600, {
      threshold: 0.1
    });

    expect(diff).to.be.lessThan(100); // Allow small differences
  });
});
```

4.7 Summary

This chapter covered:

- **Component Anatomy**: Structure, lifecycle, and organization
- **Shadow DOM**: Encapsulation, slots, and styling
- **Attributes vs Properties**: When to use each and how to reflect them
- **Component Styling**: Internal styles, theming, and CSS custom properties
- **Lifecycle Patterns**: Memory management and robust connection handling
- **Testing**: Unit, integration, and visual regression testing

You now know how to build production-quality Web Components. The next chapter explores the PAN bus in depth, showing you how to orchestrate component communication at scale.

4.8 Best Practices

1. **Always clean up in `disconnectedCallback`**
    - Remove event listeners
    - Cancel pending operations
    - Unsubscribe from events
2. **Use Shadow DOM for encapsulation**
    - Keep styles scoped
    - Avoid global style pollution
    - Use `:host` and CSS custom properties for theming
3. **Reflect important properties to attributes**
    - Makes state visible in HTML
    - Enables CSS selectors
    - Improves debugging
4. **Keep components focused**
    - Single responsibility principle
    - Compose larger components from smaller ones
    - Extract shared logic to utilities

5. **Test early and often**
    - Write tests as you build components
    - Test both happy paths and error cases
    - Use integration tests for component interaction

4.9 Further Reading

For complete Web Components reference: - *Building with LARC* Chapter 2: Core Concepts - Web Components architecture and lifecycle - *Building with LARC* Chapters 17-21: Component Reference - Complete API documentation - *Building with LARC* Chapter 13: Testing Strategies - Component testing patterns

5 The PAN Bus

The Page Area Network (PAN) bus is LARC's event-driven communication backbone. It enables decoupled, scalable component architectures by providing a pub/sub messaging system that works across your entire application.

In this chapter, you'll master the PAN bus: from basic publish/subscribe patterns to advanced message routing, error handling, and debugging techniques. By the end, you'll be able to build complex applications where components communicate seamlessly without tight coupling.

5.1 Understanding Pub/Sub Architecture

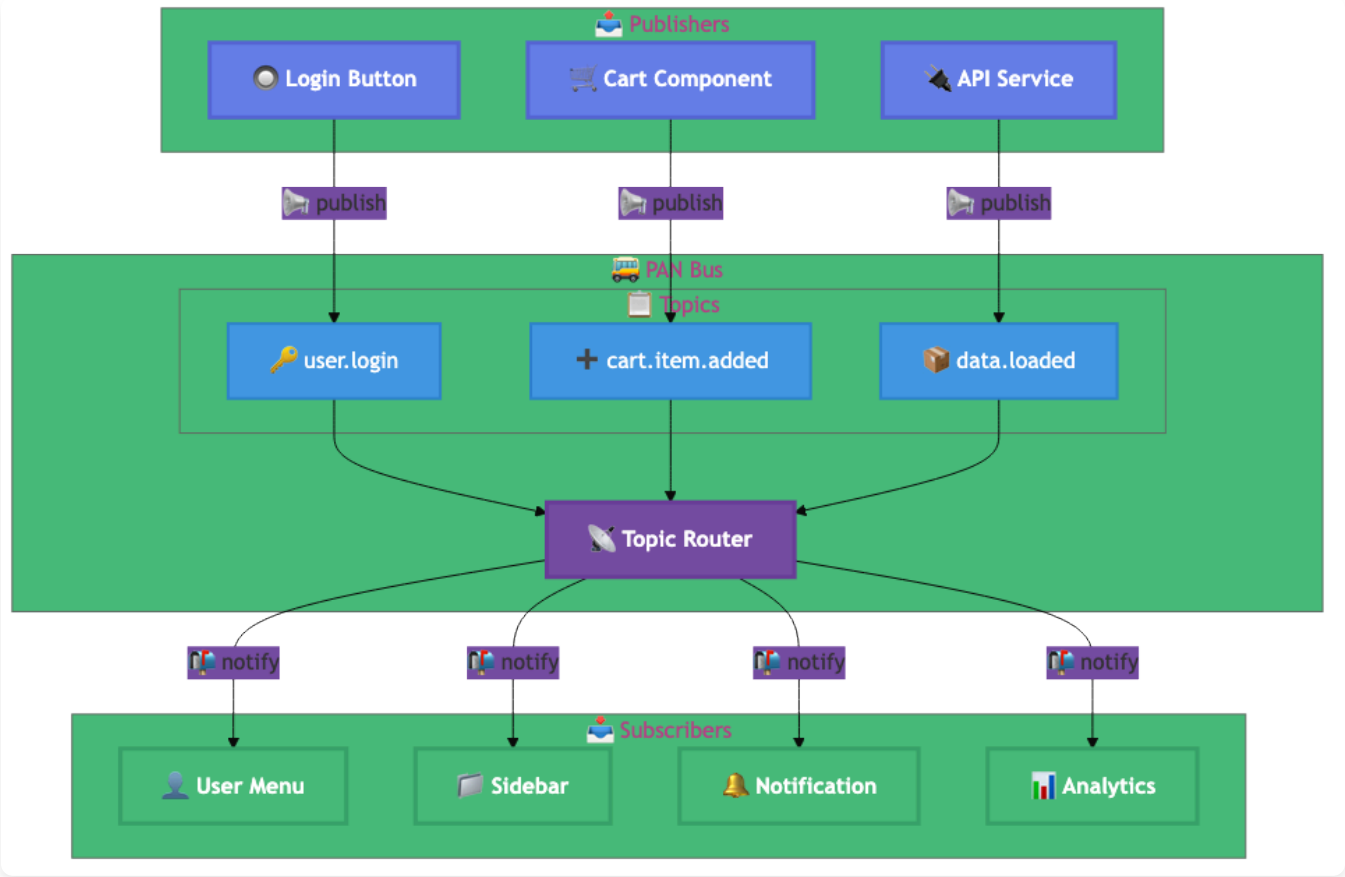

Figure 5.1: PAN Bus Pub/Sub Architecture

Publish/Subscribe (pub/sub) is a messaging pattern where senders (publishers) don't directly target specific receivers (subscribers). Instead, messages are sent to topics, and any component interested in those topics receives them.

5.1.1 Traditional Communication

Without pub/sub, components need direct references:

```
// ❌ Tight coupling
class LoginButton {
  handleLogin() {
    const user = this.authenticate();

    // Direct reference to other components
    document.querySelector('user-menu').updateUser(user);
    document.querySelector('sidebar').showUserPanel();
    document.querySelector('notification').show('Welcome!');
  }
}
```

Problems:

- LoginButton must know about all dependent components
- Adding new components requires modifying LoginButton
- Components can't work independently
- Testing requires mocking all dependencies

5.1.2 Pub/Sub Communication

With the PAN bus:

```
// ✓ Loose coupling
class LoginButton {
  handleLogin() {
    const user = this.authenticate();

    // Publish event - don't care who listens
    pan.publish('user.logged-in', { user });
  }
}

// Separate components subscribe independently
class UserMenu {
  connectedCallback() {
    pan.subscribe('user.logged-in', ({ user }) => {
      this.updateUser(user);
    });
  }
}

class Sidebar {
  connectedCallback() {
    pan.subscribe('user.logged-in', () => {
      this.showUserPanel();
    });
  }
}

class Notification {
  connectedCallback() {
    pan.subscribe('user.logged-in', () => {
      this.show('Welcome!');
    });
  }
}
```

Benefits:

- LoginButton doesn't know about consumers
- Add new subscribers without changing publishers
- Components work independently
- Easy to test in isolation

5.1.3 The PAN Bus API

The PAN bus provides three core operations:

```
import { pan } from '@larcjs/core';

// 1. Publish - send a message to a topic
pan.publish('topic.name', { data: 'value' });

// 2. Subscribe - listen for messages on a topic
const unsubscribe = pan.subscribe('topic.name', (data) => {
  console.log('Received:', data);
});

// 3. Unsubscribe - stop listening
unsubscribe();
```

That's the foundation. Everything else builds on these three operations.

5.2 Topics and Namespaces

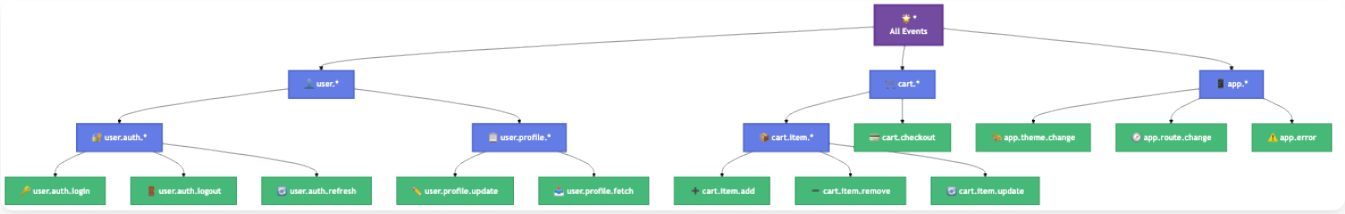

Figure 5.2: Topic Namespace Structure

Topics are the routing keys for messages. Well-designed topics make your application's data flow clear and maintainable.

5.2.1 Topic Naming Conventions

Use dot notation to create hierarchies:

```
domain.entity.action
```

Examples:

```
user.profile.updated
user.auth.login
user.auth.logout
user.settings.changed

cart.item.added
cart.item.removed
cart.total.calculated
cart.checkout.started
cart.checkout.completed

notification.info.show
notification.warning.show
notification.error.show

app.theme.changed
app.language.changed
app.route.changed
```

5.2.2 Namespace Structure

Organize topics by domain:

User Domain:

```
user.auth.login
user.auth.logout
user.auth.refresh
user.profile.fetch
user.profile.update
user.settings.fetch
user.settings.update
```

Shopping Cart Domain:

```
cart.init
cart.item.add
cart.item.remove
cart.item.update
cart.clear
cart.checkout
```

Application Domain:

```
app.ready
app.error
app.navigate
app.theme.change
app.modal.open
app.modal.close
```

5.2.3 Wildcards

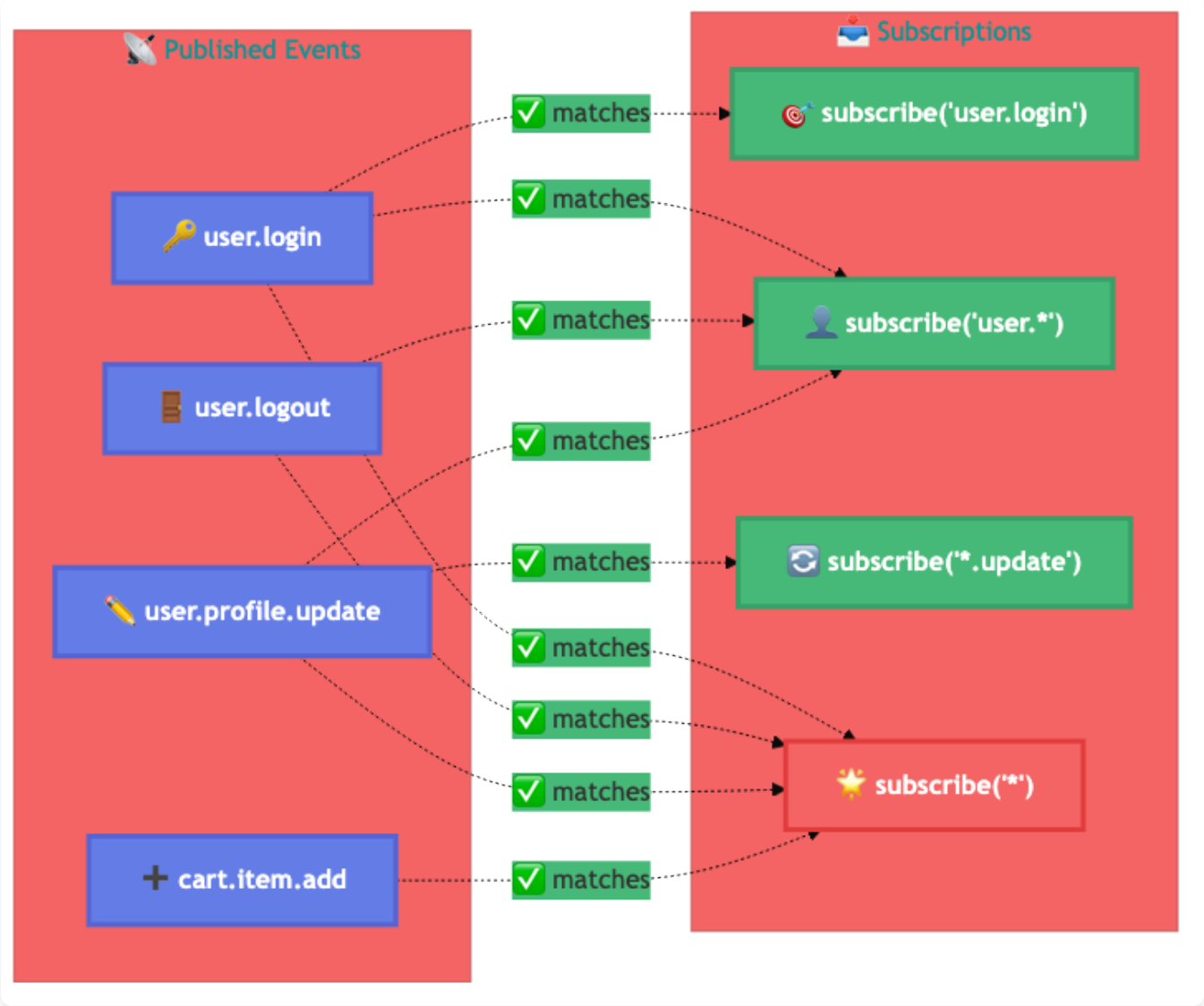

Figure 5.3: Wildcard Subscription Matching

Subscribe to multiple topics using wildcards:

```
// Subscribe to all user events
pan.subscribe('user.*', (data) => {
  console.log('User event:', data);
});

// Subscribe to all auth events across domains
pan.subscribe('*.auth.*', (data) => {
  console.log('Auth event:', data);
});

// Subscribe to ALL events (debugging)
pan.subscribe('*', (topic, data) => {
  console.log(`[${topic}]`, data);
});
```

Wildcard Patterns:

- `user.*` - All user events (user.login, user.logout, etc.)
- `*.created` - All create events (user.created, post.created, etc.)
- `user.*.updated` - All user update events (user.profile.updated, user.settings.updated, etc.)
- `*` - All events

5.2.4 Topic Best Practices

1. Be Specific:

```
// ✓ Good - clear intent
pan.publish('cart.item.added', { item, quantity });

// ✗ Bad - vague
pan.publish('cart.update', { type: 'add', item, quantity });
```

2. Use Consistent Tense:

```
// ✓ Good - past tense for events that happened
pan.publish('user.logged-in', { user });
pan.publish('data.loaded', { data });

// ✗ Bad - mixed tense
pan.publish('user.login', { user });  // Is this a command or event?
```

3. Include Context:

```
// ✓ Good - data includes context
pan.publish('task.completed', {
  taskId: 123,
  userId: 456,
  completedAt: new Date()
});

// ✗ Bad - missing context
pan.publish('task.done', { id: 123 });
```

4. Avoid Over-Nesting:

```
// ✓ Good - clear and concise
pan.publish('user.profile.updated', { user });

// ✗ Bad - too nested
pan.publish('app.domain.user.entity.profile.action.updated', { user });
```

5.3 Publishing Messages

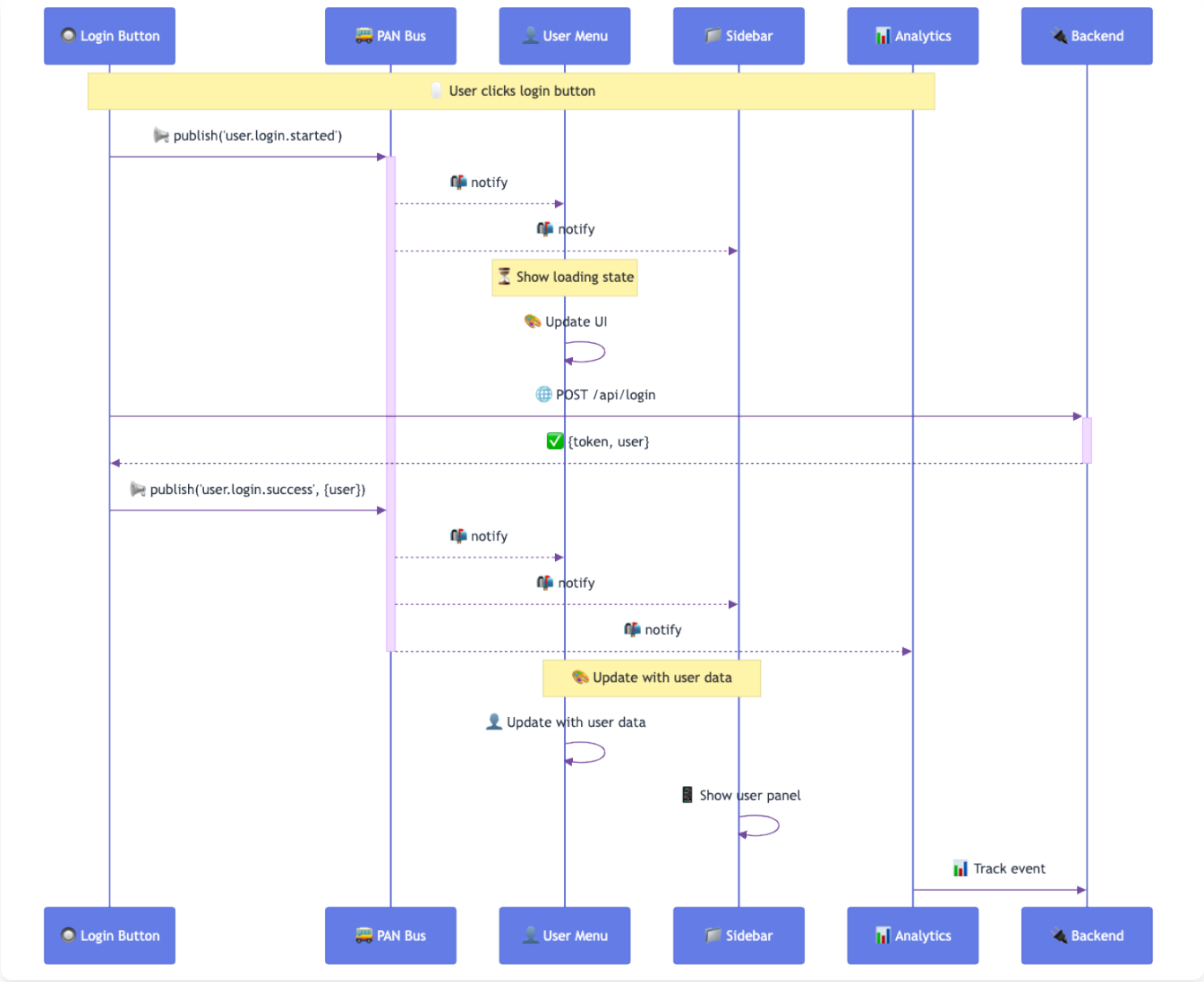

Figure 5.4: Message Flow Sequence

Publishing is straightforward, but there are patterns and options to understand.

5.3.1 Basic Publishing

```
pan.publish('event.name', { any: 'data' });
```

The data can be anything JSON-serializable:

```
// Simple value
pan.publish('counter.updated', 42);

// Object
pan.publish('user.logged-in', {
  userId: 123,
  username: 'john',
  email: 'john@example.com'
});

// Array
pan.publish('items.loaded', [
  { id: 1, name: 'Item 1' },
  { id: 2, name: 'Item 2' }
]);

// Null/undefined
pan.publish('data.cleared', null);
```

5.3.2 Publishing from Components

Publish in response to user actions or state changes:

```
class AddToCartButton extends HTMLElement {
  connectedCallback() {
    this.addEventListener('click', this.handleClick);
  }

  async handleClick() {
    const productId = this.getAttribute('product-id');
    const quantity = parseInt(this.getAttribute('quantity') || 1);

    // Publish intent
    pan.publish('cart.item.add-requested', { productId, quantity });

    try {
      // Perform action
      await this.addToCart(productId, quantity);

      // Publish success
      pan.publish('cart.item.added', {
        productId,
        quantity,
        timestamp: Date.now()
      });
    } catch (error) {
      // Publish failure
      pan.publish('cart.item.add-failed', {
        productId,
        quantity,
        error: error.message
      });
    }
  }

  async addToCart(productId, quantity) {
    const response = await fetch('/api/cart/items', {
```

```
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ productId, quantity })
    });

    if (!response.ok) {
      throw new Error('Failed to add item to cart');
    }

    return response.json();
  }
}
```

5.3.3 Event Metadata

Include metadata for debugging and auditing:

```
function publishWithMetadata(topic, data) {
  pan.publish(topic, {
    ...data,
    _meta: {
      timestamp: Date.now(),
      source: 'UserComponent',
      userId: currentUser?.id,
      sessionId: sessionId
    }
  });
}

// Usage
publishWithMetadata('order.placed', {
  orderId: 12345,
  total: 99.99
});
```

5.3.4 Batch Publishing

Publish multiple events efficiently:

```
function syncLocalChanges(changes) {
  changes.forEach(change => {
    switch (change.type) {
      case 'add':
        pan.publish('data.item.added', change.item);
        break;
      case 'update':
        pan.publish('data.item.updated', change.item);
        break;
      case 'delete':
        pan.publish('data.item.deleted', { id: change.id });
        break;
    }
  });

  // Publish batch complete
  pan.publish('data.sync.completed', {
    changesCount: changes.length,
    timestamp: Date.now()
  });
}
```

5.4 Subscribing to Events

Subscriptions are how components react to events they care about.

5.4.1 Basic Subscription

```
const unsubscribe = pan.subscribe('event.name', (data) => {
  console.log('Received:', data);
});

// Later, when done
unsubscribe();
```

5.4.2 Component Lifecycle Integration

Subscribe in `connectedCallback`, unsubscribe in `disconnectedCallback`:

```
class NotificationDisplay extends HTMLElement {
  connectedCallback() {
    // Subscribe to notification events
    this.unsubscribeInfo = pan.subscribe('notification.info',
        this.showInfo);
    this.unsubscribeWarning = pan.subscribe('notification.warning',
        this.showWarning);
    this.unsubscribeError = pan.subscribe('notification.error',
        this.showError);
  }

  disconnectedCallback() {
    // Clean up subscriptions
    this.unsubscribeInfo();
    this.unsubscribeWarning();
    this.unsubscribeError();
  }

  showInfo = (data) => {
    this.showNotification('info', data.message);
  }

  showWarning = (data) => {
    this.showNotification('warning', data.message);
  }

  showError = (data) => {
    this.showNotification('error', data.message);
  }

  showNotification(type, message) {
    // Render notification UI
  }
}
```

5.4.3 Multiple Subscriptions Helper

Manage multiple subscriptions easily:

```
class SubscriptionManager {
  constructor() {
    this.subscriptions = [];
  }

  subscribe(topic, handler) {
    const unsubscribe = pan.subscribe(topic, handler);
    this.subscriptions.push(unsubscribe);
    return unsubscribe;
  }

  unsubscribeAll() {
    this.subscriptions.forEach(unsubscribe => unsubscribe());
    this.subscriptions = [];
  }
}

// Usage in component
class MyComponent extends HTMLElement {
  constructor() {
    super();
    this.subs = new SubscriptionManager();
  }

  connectedCallback() {
    this.subs.subscribe('user.login', this.handleLogin);
    this.subs.subscribe('user.logout', this.handleLogout);
    this.subs.subscribe('app.theme.changed', this.handleThemeChange);
  }

  disconnectedCallback() {
    this.subs.unsubscribeAll();
  }
```

```
  handleLogin = (data) => { /* ... */ }
  handleLogout = (data) => { /* ... */ }
  handleThemeChange = (data) => { /* ... */ }
}
```

5.4.4 Conditional Subscriptions

Subscribe only when conditions are met:

```
class UserDashboard extends HTMLElement {
  connectedCallback() {
    // Subscribe to user-specific events only when user is logged in
    this.unsubscribeAuth = pan.subscribe('auth.state.changed', ({
        isAuthenticated, user }) => {
      if (isAuthenticated) {
        this.subscribeToUserEvents(user.id);
      } else {
        this.unsubscribeFromUserEvents();
      }
    });
  }

  subscribeToUserEvents(userId) {
    this.unsubscribeUserActivity = pan.subscribe('user.activity', (data)
        => {
      if (data.userId === userId) {
        this.updateActivity(data);
      }
    });

    this.unsubscribeUserNotifications =
        pan.subscribe('user.notifications', (data) => {
      if (data.userId === userId) {
        this.showNotification(data);
      }
    });
  }

  unsubscribeFromUserEvents() {
    if (this.unsubscribeUserActivity) {
      this.unsubscribeUserActivity();
      this.unsubscribeUserActivity = null;
    }
```

```
    if (this.unsubscribeUserNotifications) {
      this.unsubscribeUserNotifications();
      this.unsubscribeUserNotifications = null;
    }
  }
}
```

5.4.5 Filtering Events

Filter events in the subscriber:

```
pan.subscribe('task.updated', (task) => {
  // Only handle tasks assigned to current user
  if (task.assignedTo === currentUser.id) {
    this.updateTaskDisplay(task);
  }
});

pan.subscribe('notification.*', (notification) => {
  // Only show high-priority notifications
  if (notification.priority >= 3) {
    this.showNotification(notification);
  }
});
```

5.5 Message Patterns

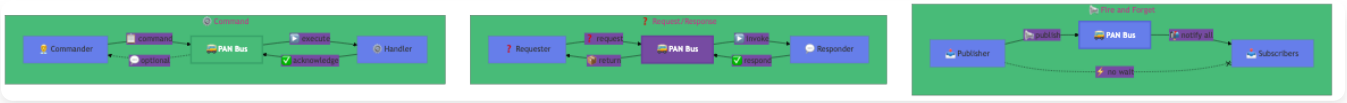

Figure 5.5: Event Pattern Comparison

The PAN bus supports several messaging patterns for different use cases.

5.5.1 1. Fire and Forget

Most common pattern. Publish and continue without waiting:

```
// Publisher
function saveSettings(settings) {
  localStorage.setItem('settings', JSON.stringify(settings));
  pan.publish('settings.saved', settings);
}

// Subscriber
pan.subscribe('settings.saved', (settings) => {
  console.log('Settings updated:', settings);
  updateUI(settings);
});
```

Use when:

- Multiple components may react
- You don’t need confirmation
- Action is non-critical

5.5.2 2. Request/Response

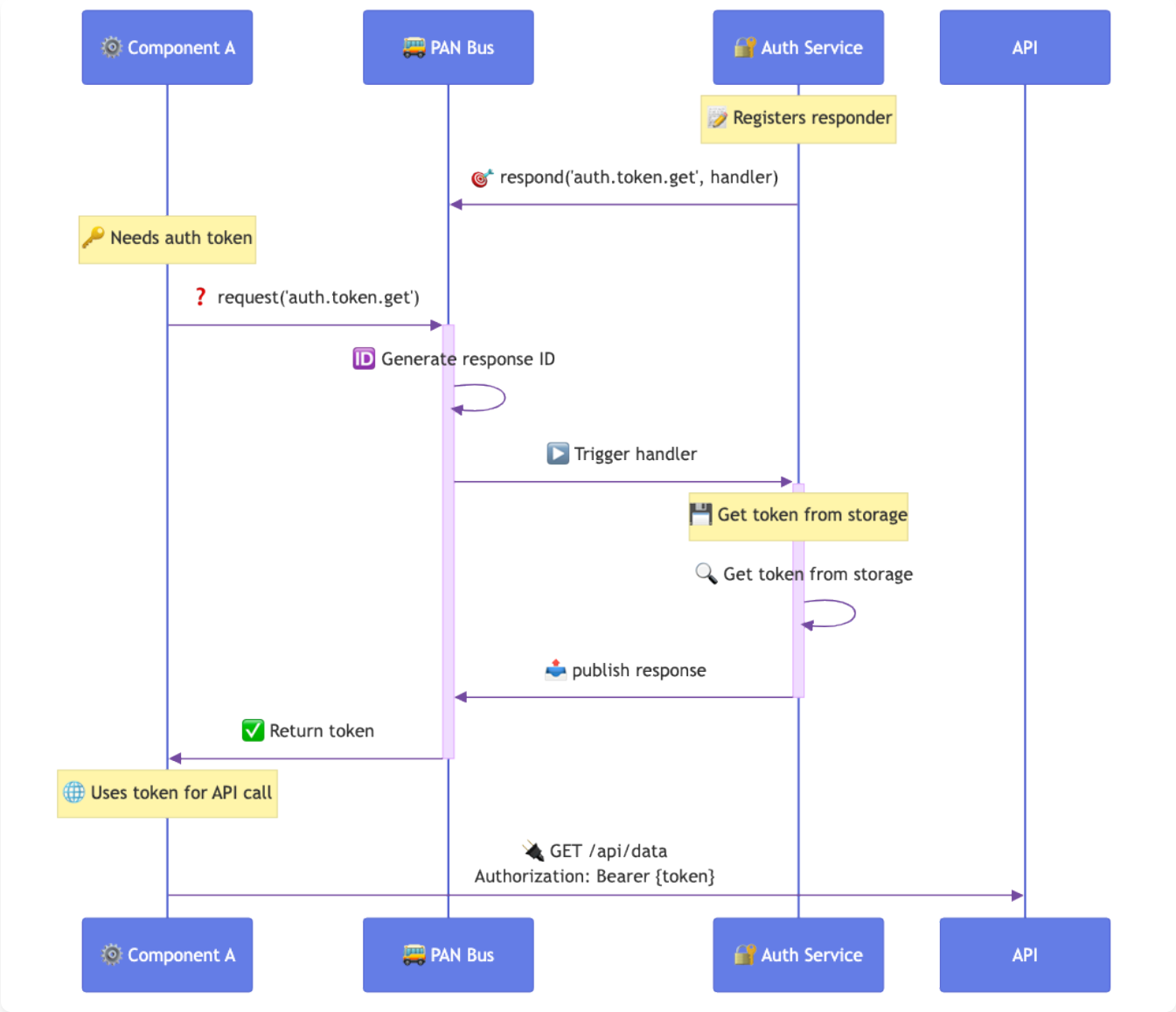

Figure 5.6: Request/Response Pattern

Request data and wait for a response:

```
// Responder
pan.respond('auth.token.get', async () => {
  return localStorage.getItem('authToken');
});

// Requester
const token = await pan.request('auth.token.get');
console.log('Token:', token);
```

Implementation:

```
// In PAN library
class PAN {
  request(topic, data, timeout = 5000) {
    return new Promise((resolve, reject) => {
      const responseId = `${topic}:${Date.now()}:${Math.random()}`;

      // Subscribe to response
      const unsubscribe =
        this.subscribe(`${topic}:response:${responseId}`, (response) => {
        unsubscribe();
        clearTimeout(timer);
        resolve(response);
      });

      // Set timeout
      const timer = setTimeout(() => {
        unsubscribe();
        reject(new Error(`Request timeout: ${topic}`));
      }, timeout);

      // Publish request
      this.publish(`${topic}:request`, {
        ...data,
        _responseId: responseId
      });
    });
  }

  respond(topic, handler) {
    return this.subscribe(`${topic}:request`, async (data) => {
      try {
        const result = await handler(data);
        this.publish(`${topic}:response:${data._responseId}`, result);
      } catch (error) {
```

```
        this.publish(`${topic}:response:${data._responseId}`, {
          error: error.message
        });
      }
    });
  }
}
```

Use when:

- Need data from another component
- Waiting for response is acceptable
- Asynchronous operations

5.5.3 3. Command Pattern

Issue commands that components execute:

```
// Command issuer
pan.publish('modal.open', {
  component: 'user-profile',
  props: { userId: 123 }
});

// Command handler
pan.subscribe('modal.open', ({ component, props }) => {
  const modal = document.createElement('app-modal');
  modal.component = component;
  modal.props = props;
  document.body.appendChild(modal);
});
```

Use when:

- Triggering actions in other components
- Implementing undo/redo
- Building command palette UIs

5.5.4 4. Event Sourcing

Store events for replay or auditing:

```
const eventStore = [];

// Store all events
pan.subscribe('*', (topic, data) => {
  eventStore.push({
    topic,
    data,
    timestamp: Date.now()
  });
});

// Replay events
function replayEvents(fromTimestamp) {
  eventStore
    .filter(event => event.timestamp >= fromTimestamp)
    .forEach(event => {
      pan.publish(event.topic, event.data);
    });
}

// Get events for debugging
function getEventHistory(topic) {
  return eventStore.filter(event =>
    event.topic === topic || event.topic.startsWith(topic + '.')
  );
}
```

Use when:

- Debugging complex interactions
- Implementing undo/redo
- Auditing user actions
- Syncing state across sessions

5.5.5 5. Aggregation Pattern

Collect multiple events before acting:

```
class DataAggregator extends HTMLElement {
  constructor() {
    super();
    this.pendingUpdates = new Set();
    this.debounceTimer = null;
  }

  connectedCallback() {
    pan.subscribe('data.item.updated', ({ id }) => {
      this.pendingUpdates.add(id);
      this.scheduleRefresh();
    });
  }

  scheduleRefresh() {
    clearTimeout(this.debounceTimer);

    this.debounceTimer = setTimeout(() => {
      this.refreshItems(Array.from(this.pendingUpdates));
      this.pendingUpdates.clear();
    }, 500);
  }

  async refreshItems(ids) {
    const items = await fetchItems(ids);
    this.render(items);
  }
}
```

Use when:

- Avoiding excessive updates
- Batching API requests
- Debouncing rapid events

5.5.6 6. Saga Pattern

Coordinate multi-step processes:

```
class CheckoutSaga {
  constructor() {
    this.setupListeners();
  }

  setupListeners() {
    pan.subscribe('checkout.started', this.handleCheckoutStart);
    pan.subscribe('payment.completed', this.handlePaymentComplete);
    pan.subscribe('order.created', this.handleOrderCreated);
  }

  handleCheckoutStart = async ({ cart }) => {
    try {
      // Step 1: Validate cart
      pan.publish('checkout.validating', { cart });
      await this.validateCart(cart);

      // Step 2: Calculate totals
      pan.publish('checkout.calculating', { cart });
      const totals = await this.calculateTotals(cart);

      // Step 3: Request payment
      pan.publish('payment.requested', { totals });
    } catch (error) {
      pan.publish('checkout.failed', { error: error.message });
    }
  }

  handlePaymentComplete = async ({ paymentId, totals }) => {
    try {
      // Step 4: Create order
      pan.publish('order.creating', { paymentId });
      const order = await this.createOrder(paymentId, totals);
```

```
      pan.publish('order.created', { order });
    } catch (error) {
      // Compensating transaction: refund payment
      pan.publish('payment.refund-requested', { paymentId });
      pan.publish('checkout.failed', { error: error.message });
    }
  }

  handleOrderCreated = async ({ order }) => {
    // Step 5: Send confirmation
    pan.publish('order.confirmation-sending', { order });
    await this.sendConfirmation(order);

    // Step 6: Complete checkout
    pan.publish('checkout.completed', { order });
  }
}
```

Use when:

- Complex multi-step workflows
- Need to handle failures and rollbacks
- Coordinating multiple services

5.6 Debugging PAN Communication

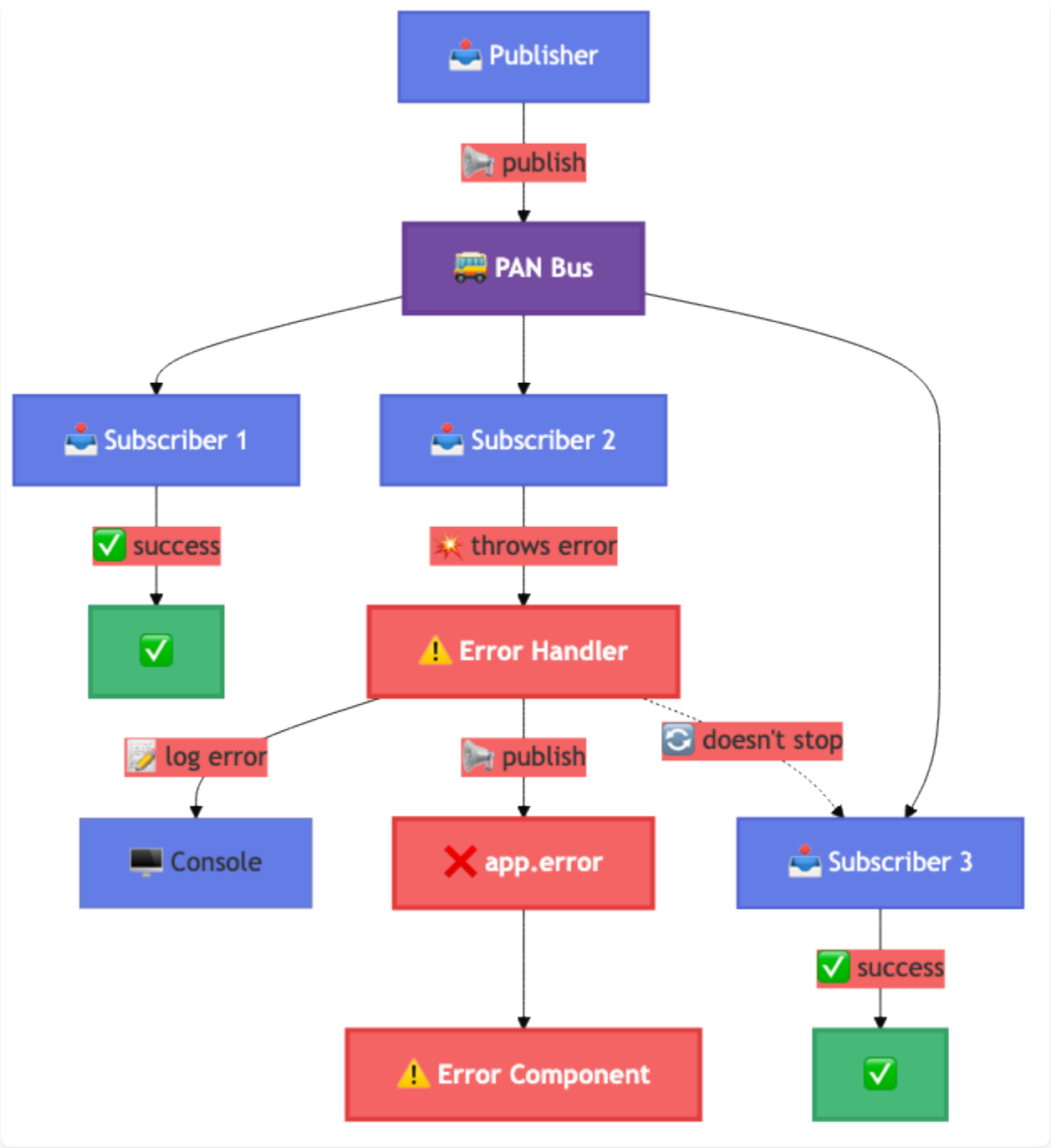

Figure 5.7: PAN Bus Internal Architecture

Debugging event-driven systems requires different techniques than traditional debugging.

5.6.1 Logging All Events

```
// Enable debug mode
pan.debug(true);

// Or manually subscribe to all events
pan.subscribe('*', (topic, data) => {
  console.group(`[PAN] ${topic}`);
  console.log('Data:', data);
  console.log('Timestamp:', new Date().toISOString());
  console.trace('Stack trace');
  console.groupEnd();
});
```

5.6.2 Event Inspector

Build a visual event inspector:

```
class PanInspector extends HTMLElement {
  constructor() {
    super();
    this.events = [];
    this.maxEvents = 100;
  }

  connectedCallback() {
    this.render();

    pan.subscribe('*', (topic, data) => {
      this.logEvent(topic, data);
    });
  }

  logEvent(topic, data) {
    this.events.unshift({
      topic,
      data,
      timestamp: Date.now()
    });

    if (this.events.length > this.maxEvents) {
      this.events.pop();
    }

    this.render();
  }

  render() {
    this.innerHTML = `
      <style>
        .pan-inspector {
          position: fixed;
```

```
      bottom: 0;
      right: 0;
      width: 400px;
      height: 300px;
      background: white;
      border: 1px solid #ccc;
      overflow: auto;
      font-family: monospace;
      font-size: 12px;
    }

    .event {
      padding: 8px;
      border-bottom: 1px solid #eee;
    }

    .event:hover {
      background: #f5f5f5;
    }

    .topic {
      font-weight: bold;
      color: #667eea;
    }

    .timestamp {
      color: #999;
      font-size: 10px;
    }

    .data {
      margin-top: 4px;
      color: #333;
    }
```

```
      </style>

      <div class="pan-inspector">
        <h3>PAN Event Inspector</h3>
        ${this.events.map(event => `
          <div class="event">
            <div class="topic">${event.topic}</div>
            <div class="timestamp">${new
        Date(event.timestamp).toLocaleTimeString()}</div>
            <div class="data">${JSON.stringify(event.data, null,
        2)}</div>
          </div>
        `).join('')}
      </div>
    `;
  }
}

customElements.define('pan-inspector', PanInspector);
```

5.6.3 Event Filtering

Filter events for specific topics:

```
function filterEvents(pattern) {
  const regex = new RegExp(pattern.replace('*', '.*'));

  pan.subscribe('*', (topic, data) => {
    if (regex.test(topic)) {
      console.log(`[FILTERED] ${topic}:`, data);
    }
  });
}

// Usage
filterEvents('user.*');     // Only user events
filterEvents('*.error');    // All error events
filterEvents('cart|order'); // Cart or order events
```

5.6.4 Performance Monitoring

Track event frequency and performance:

```
class PanMonitor {
  constructor() {
    this.stats = new Map();

    pan.subscribe('*', (topic) => {
      const stat = this.stats.get(topic) || { count: 0, timestamps: [] };

      stat.count++;
      stat.timestamps.push(Date.now());

      // Keep only last 100 timestamps
      if (stat.timestamps.length > 100) {
        stat.timestamps.shift();
      }

      this.stats.set(topic, stat);
    });
  }

  getStats(topic) {
    const stat = this.stats.get(topic);
    if (!stat) return null;

    const timestamps = stat.timestamps;
    const duration = timestamps[timestamps.length - 1] - timestamps[0];
    const frequency = timestamps.length / (duration / 1000);

    return {
      topic,
      count: stat.count,
      frequency: frequency.toFixed(2) + ' events/sec',
      lastEvent: new Date(timestamps[timestamps.length - 1])
    };
  }
```

```
  getAllStats() {
    const results = [];

    this.stats.forEach((_, topic) => {
      results.push(this.getStats(topic));
    });

    return results.sort((a, b) => b.count - a.count);
  }

  reset() {
    this.stats.clear();
  }
}

// Usage
const monitor = new PanMonitor();

// Later, check stats
console.table(monitor.getAllStats());
```

5.6.5 Event Replay

Capture and replay events for testing:

```
class EventRecorder {
  constructor() {
    this.recording = false;
    this.events = [];
  }

  start() {
    this.recording = true;
    this.events = [];

    this.unsubscribe = pan.subscribe('*', (topic, data) => {
      if (this.recording) {
        this.events.push({ topic, data, timestamp: Date.now() });
      }
    });
  }

  stop() {
    this.recording = false;
    if (this.unsubscribe) {
      this.unsubscribe();
    }

    return this.events;
  }

  replay(events, speed = 1) {
    if (!events || events.length === 0) return;

    const startTime = events[0].timestamp;

    events.forEach((event, index) => {
      const delay = (event.timestamp - startTime) / speed;
```

```
      setTimeout(() => {
        pan.publish(event.topic, event.data);
      }, delay);
    });
  }

  save(name) {
    localStorage.setItem(`pan-recording-${name}`,
        JSON.stringify(this.events));
  }

  load(name) {
    const data = localStorage.getItem(`pan-recording-${name}`);
    return data ? JSON.parse(data) : null;
  }
}

// Usage
const recorder = new EventRecorder();

// Start recording
recorder.start();

// ... perform actions ...

// Stop and save
const events = recorder.stop();
recorder.save('my-test-scenario');

// Later, replay
const events = recorder.load('my-test-scenario');
recorder.replay(events, 2); // 2x speed
```

5.7 Summary

This chapter covered:

- **Pub/Sub Architecture**: Decoupled communication via topics
- **Topics and Namespaces**: Organizing events with hierarchical naming
- **Publishing**: Sending messages and event patterns
- **Subscribing**: Receiving and filtering events
- **Message Patterns**: Fire-and-forget, request/response, commands, sagas
- **Debugging**: Logging, inspection, monitoring, and replay tools

The PAN bus is central to LARC applications. Mastering it enables you to build scalable, maintainable applications where components collaborate without tight coupling.

5.8 Best Practices

1. **Use descriptive topic names**
    - `user.profile.updated` not `userUpdated`
    - Past tense for events that happened
    - Include context in message data
2. **Clean up subscriptions**
    - Always unsubscribe in `disconnectedCallback`
    - Use subscription managers for multiple subscriptions
    - Avoid memory leaks
3. **Avoid infinite loops**
    - Don't publish the same event you're subscribed to
    - Use different topics for input and output
    - Add loop detection in debug mode
4. **Keep handlers fast**
    - Don't block the event loop
    - Use async/await for long operations
    - Consider debouncing rapid events

5. **Include metadata**
    - Timestamp, source, user ID for debugging
    - Request IDs for tracing
    - Error details for failures
6. **Test event flows**
    - Use event recorders for integration tests
    - Mock pan.publish/subscribe in unit tests
    - Verify event contracts between components

5.9 Further Reading

For complete PAN bus API reference: - *Building with LARC* Chapter 2: Core Concepts - Message bus architecture deep dive - *Building with LARC* Appendix A: Message Topics Reference - Standard topic conventions - *Building with LARC* Appendix B: Event Envelope Specification - Message format details

6 State Management

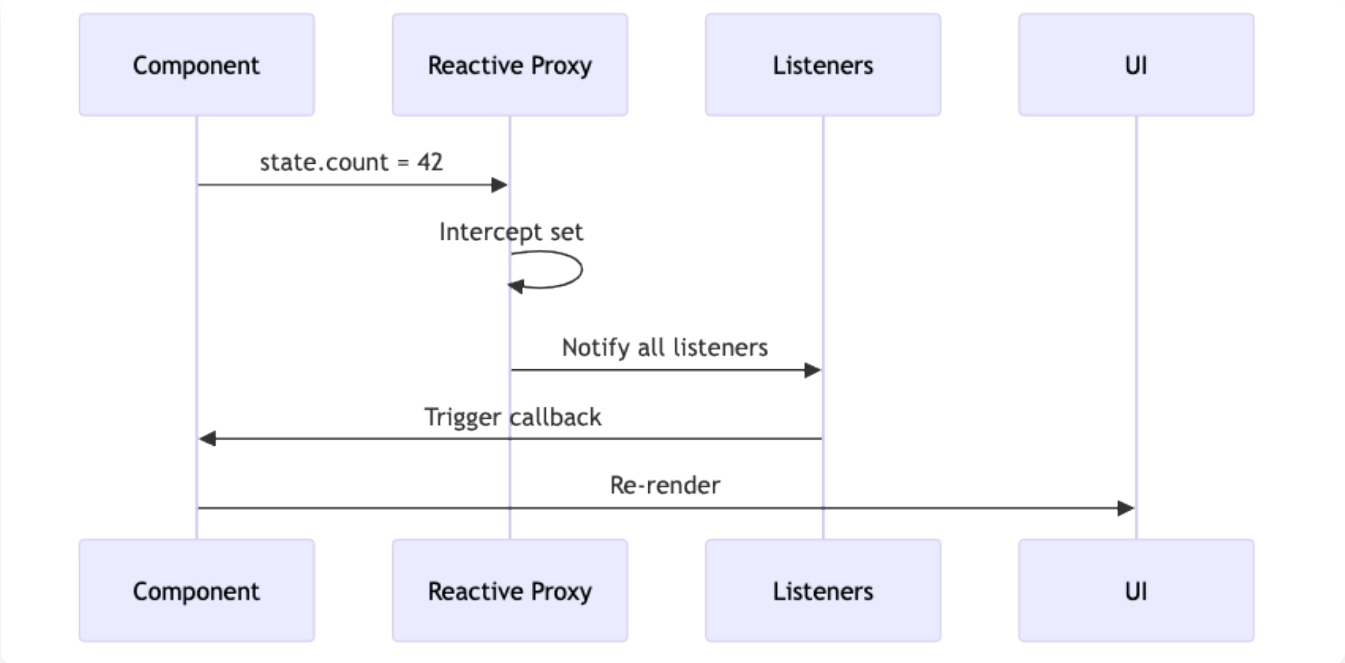

Figure 6.1: State Management Hierarchy

State management is one of the most critical aspects of application development. Poor state management leads to bugs, performance issues, and maintenance nightmares. Good state management makes applications predictable, testable, and maintainable.

LARC takes a pragmatic approach: start simple and scale complexity only when needed. This chapter explores state management at every level, from component-local state to distributed, offline-first architectures.

6.1 Component-Local State

The simplest form of state lives entirely within a single component. This is your first choice for most scenarios.

6.1.1 Instance Properties

Use instance properties for component-specific state:

```
class ToggleSwitch extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });

    // Local state
    this.isOn = false;
  }

  connectedCallback() {
    this.render();

    this.shadowRoot.querySelector('button').addEventListener('click', ()
        => {

      this.isOn = !this.isOn;  // Update state
      this.render();            // Re-render

      // Notify others
      this.dispatchEvent(new CustomEvent('toggle', {
        detail: { isOn: this.isOn }
      }));
    });
  }

  render() {
    this.shadowRoot.innerHTML = `
      <style>
        button {
          background: ${this.isOn ? '#48bb78' : '#cbd5e0'};
          color: white;
          border: none;
          padding: 8px 16px;
          border-radius: 4px;
```

```
            cursor: pointer;
          }
        </style>
        <button>${this.isOn ? 'ON' : 'OFF'}</button>
      `;
    }
}
```

When to use:

- UI state (expanded/collapsed, selected, etc.)
- Temporary values (search input, form drafts)
- Component-specific configuration

Advantages:

- Simple and straightforward
- No dependencies on external state
- Easy to reason about
- Easy to test

6.1.2 Private Fields

Use private fields (with `#`) for true encapsulation:

```
class Counter extends HTMLElement {
  // Private fields
  #count = 0;
  #max = 100;
  #min = 0;

  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
  }

  // Public getter
  get count() {
    return this.#count;
  }

  // Public setter with validation
  set count(value) {
    const newCount = Number(value);

    if (isNaN(newCount)) {
      throw new Error('Count must be a number');
    }

    if (newCount < this.#min || newCount > this.#max) {
      throw new Error(`Count must be between ${this.#min} and
        ${this.#max}`);
    }

    this.#count = newCount;
    this.render();
  }

  increment() {
```

```
    this.count = Math.min(this.#count + 1, this.#max);
  }

  decrement() {
    this.count = Math.max(this.#count - 1, this.#min);
  }

  render() {
    this.shadowRoot.innerHTML = `
      <div>${this.#count}</div>
    `;
  }
}
```

Benefits:

- True privacy (can't access from outside)
- Validation at setter boundaries
- Clear public API

6.1.3 State Objects

Organize related state in objects:

```
class UserProfile extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });

    // Group related state
    this.state = {
      user: null,
      loading: false,
      error: null,
      editMode: false
    };
  }

  setState(updates) {
    // Merge updates into state
    this.state = {
      ...this.state,
      ...updates
    };

    this.render();
  }

  async loadUser(userId) {
    this.setState({ loading: true, error: null });

    try {
      const response = await fetch(`/api/users/${userId}`);
      const user = await response.json();

      this.setState({ user, loading: false });
    } catch (error) {
      this.setState({ error: error.message, loading: false });
```

```
    }
  }

  render() {
    const { user, loading, error, editMode } = this.state;

    if (loading) {
      this.shadowRoot.innerHTML = '<div>Loading...</div>';
    } else if (error) {
      this.shadowRoot.innerHTML = `<div class="error">${error}</div>`;
    } else if (user) {
      this.shadowRoot.innerHTML = `
        <div>
          <h2>${user.name}</h2>
          ${editMode ? this.renderEditForm() : this.renderDisplay()}
        </div>
      `;
    }
  }
}
```

Benefits:

- Organized state structure
- Single method to update state
- Clear state shape
- Easier debugging (log entire state)

6.2 Shared State Patterns

When multiple components need access to the same data, you need shared state.

6.2.1 Simple Global State

Create a shared state object:

```
// lib/state.js
export const appState = {
  user: null,
  theme: 'light',
  language: 'en',
  notifications: []
};

// Update state
export function updateState(updates) {
  Object.assign(appState, updates);
  pan.publish('app.state.changed', appState);
}

// Get state
export function getState() {
  return { ...appState };
}
```

Usage in components:

```
import { appState, updateState } from '../lib/state.js';

class ThemeSwitcher extends HTMLElement {
  connectedCallback() {
    // Read initial state
    this.render(appState.theme);

    // Subscribe to changes
    this.unsubscribe = pan.subscribe('app.state.changed', (state) => {
      this.render(state.theme);
    });

    // Add event listener
    this.addEventListener('click', () => {
      const newTheme = appState.theme === 'light' ? 'dark' : 'light';
      updateState({ theme: newTheme });
    });
  }

  disconnectedCallback() {
    this.unsubscribe();
  }

  render(theme) {
    this.textContent = `Theme: ${theme}`;
  }
}
```

6.2.2 Reactive State with Proxy

Make state changes automatically trigger updates:

```
// lib/reactive-state.js
export function createReactiveState(initialState) {
  const listeners = new Set();

  const state = new Proxy(initialState, {
    set(target, property, value) {
      const oldValue = target[property];
      target[property] = value;

      // Notify listeners
      listeners.forEach(listener => {
        listener(property, value, oldValue);
      });

      // Also publish via PAN
      pan.publish('state.changed', {
        property,
        value,
        oldValue
      });

      return true;
    },

    get(target, property) {
      return target[property];
    }
  });

  return {
    state,
    subscribe(listener) {
      listeners.add(listener);
      return () => listeners.delete(listener);
```

```
    },
    getState() {
      return { ...state };
    }
  };
}
```

Usage:

```
// Create reactive state
const { state, subscribe } = createReactiveState({
  count: 0,
  user: null,
  theme: 'light'
});

// Components automatically react to changes
class CountDisplay extends HTMLElement {
  connectedCallback() {
    // Subscribe to specific property changes
    this.unsubscribe = subscribe((property, value) => {
      if (property === 'count') {
        this.textContent = `Count: ${value}`;
      }
    });

    // Initial render
    this.textContent = `Count: ${state.count}`;
  }

  disconnectedCallback() {
    this.unsubscribe();
  }
}

// Update state (automatically triggers updates)
state.count++;  // All subscribers notified
state.count = 42;  // All subscribers notified
```

6.2.3 Store Pattern

Build a more sophisticated store:

```
// lib/store.js
class Store {
  constructor(initialState = {}) {
    this.state = initialState;
    this.listeners = new Map();
    this.middleware = [];
  }

  getState() {
    return { ...this.state };
  }

  setState(updates) {
    const oldState = { ...this.state };
    this.state = { ...this.state, ...updates };

    // Run middleware
    this.middleware.forEach(fn => fn(this.state, oldState));

    // Notify listeners
    this.listeners.forEach((listeners, key) => {
      if (key === '*' || key in updates) {
        listeners.forEach(listener => {
          listener(this.state, oldState);
        });
      }
    });
  }

  subscribe(key, listener) {
    if (!this.listeners.has(key)) {
      this.listeners.set(key, new Set());
    }
```

```
    this.listeners.get(key).add(listener);

    // Return unsubscribe function
    return () => {
      const listeners = this.listeners.get(key);
      if (listeners) {
        listeners.delete(listener);
      }
    };
  }

  use(middleware) {
    this.middleware.push(middleware);
  }

  dispatch(action) {
    // Action pattern: { type, payload }
    switch (action.type) {
      case 'user/login':
        this.setState({ user: action.payload });
        break;
      case 'user/logout':
        this.setState({ user: null });
        break;
      case 'theme/change':
        this.setState({ theme: action.payload });
        break;
      default:
        console.warn(`Unknown action: ${action.type}`);
    }
  }
}

// Create store instance
```

```
export const store = new Store({
  user: null,
  theme: 'light',
  notifications: []
});

// Add logging middleware
store.use((state, oldState) => {
  console.log('State changed:', { old: oldState, new: state });
});

// Add persistence middleware
store.use((state) => {
  localStorage.setItem('app-state', JSON.stringify(state));
});
```

Usage:

```
import { store } from '../lib/store.js';

class UserMenu extends HTMLElement {
  connectedCallback() {
    // Subscribe to user changes only
    this.unsubscribe = store.subscribe('user', (state) => {
      this.render(state.user);
    });

    // Initial render
    this.render(store.getState().user);
  }

  disconnectedCallback() {
    this.unsubscribe();
  }

  render(user) {
    if (user) {
      this.innerHTML = `
        <div>Hello, ${user.name}</div>
        <button id="logout">Logout</button>
      `;

      this.querySelector('#logout').addEventListener('click', () => {
        store.dispatch({ type: 'user/logout' });
      });
    } else {
      this.innerHTML = '<button id="login">Login</button>';

      this.querySelector('#login').addEventListener('click', () => {
        // Trigger login flow
        pan.publish('auth.login.requested');
      });
```

```
    }
  }
}
```

6.3 The pan-store Component

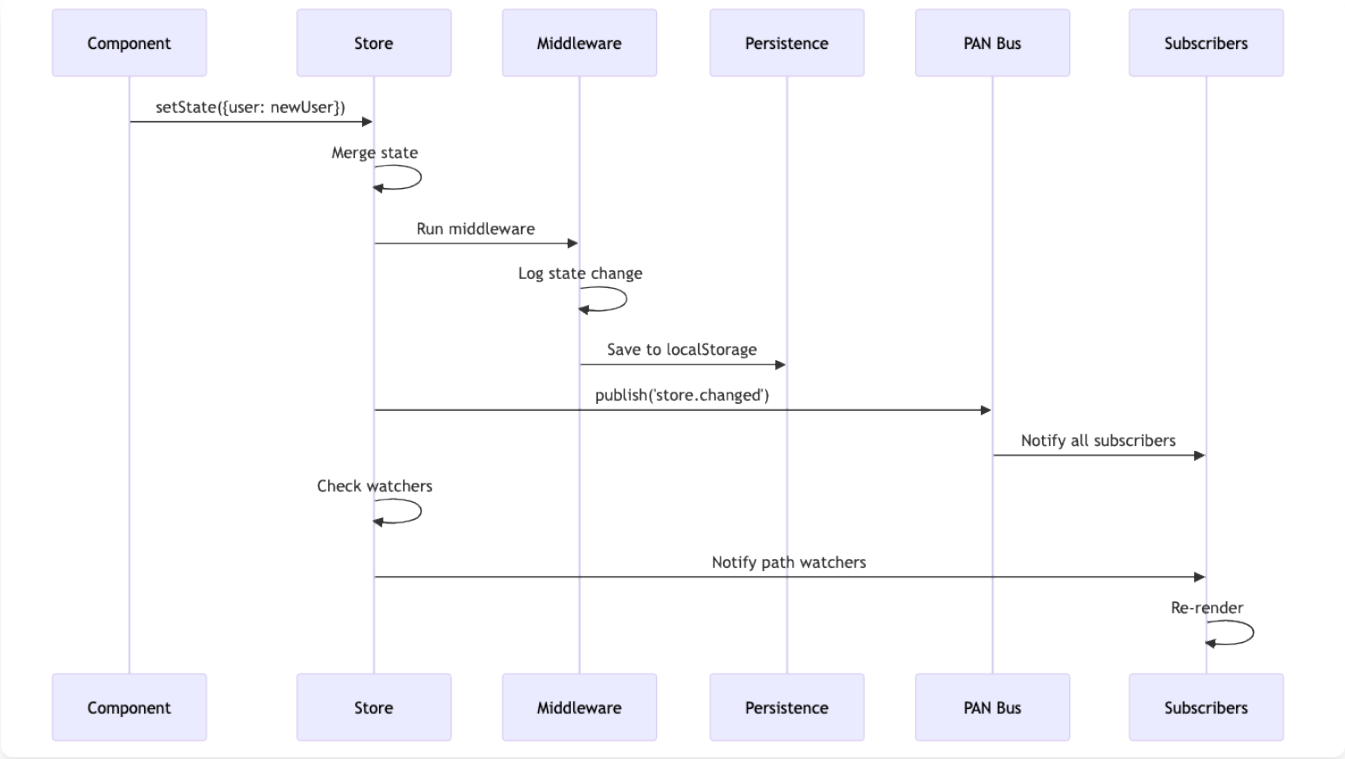

Figure 6.2: pan-store Architecture

LARC provides a built-in component for state management:

```
<pan-store id="app-store" persist="true">
  <!-- Initial state -->
  <script type="application/json">
  {
    "user": null,
    "theme": "light",
    "cart": {
      "items": [],
      "total": 0
    }
  }
  </script>
</pan-store>

<script type="module">
  const store = document.getElementById('app-store');

  // Get state
  const state = store.getState();

  // Update state
  store.setState({ theme: 'dark' });

  // Subscribe to changes
  store.addEventListener('state-changed', (e) => {
    console.log('State changed:', e.detail);
  });

  // Or use PAN bus
  pan.subscribe('store.changed', (state) => {
    console.log('State via PAN:', state);
  });
</script>
```

Features:

- Declarative state initialization
- Optional persistence to localStorage
- Integrates with PAN bus
- Supports nested state updates
- Time-travel debugging in dev mode

Advanced usage:

```
// Get nested state
const cartItems = store.getState('cart.items');

// Update nested state
store.setState('cart.items', [...items, newItem]);

// Subscribe to specific paths
store.subscribe('cart.total', (value) => {
  console.log('Cart total changed:', value);
});

// Computed properties
store.computed('cart.itemCount', (state) => {
  return state.cart.items.length;
});

// Actions
store.action('addToCart', (item) => {
  const cart = store.getState('cart');
  const items = [...cart.items, item];
  const total = items.reduce((sum, item) => sum + item.price, 0);

  store.setState({
    'cart.items': items,
    'cart.total': total
  });
});

// Use action
store.dispatch('addToCart', { id: 1, name: 'Product', price: 29.99 });
```

6.4 IndexedDB Integration

For large datasets or offline capability, use IndexedDB:

6.4.1 Basic IndexedDB Wrapper

```
// lib/db.js
class Database {
  constructor(name, version = 1) {
    this.name = name;
    this.version = version;
    this.db = null;
  }

  async open(stores) {
    return new Promise((resolve, reject) => {
      const request = indexedDB.open(this.name, this.version);

      request.onerror = () => reject(request.error);
      request.onsuccess = () => {
        this.db = request.result;
        resolve(this.db);
      };

      request.onupgradeneeded = (event) => {
        const db = event.target.result;

        stores.forEach(({ name, keyPath, indexes }) => {
          if (!db.objectStoreNames.contains(name)) {
            const store = db.createObjectStore(name, { keyPath });

            indexes?.forEach(({ name, keyPath, options }) => {
              store.createIndex(name, keyPath, options);
            });
          }
        });
      };
    });
  }
```

```
  async add(storeName, data) {
    const tx = this.db.transaction(storeName, 'readwrite');
    const store = tx.objectStore(storeName);

    return new Promise((resolve, reject) => {
      const request = store.add(data);
      request.onsuccess = () => resolve(request.result);
      request.onerror = () => reject(request.error);
    });
  }

  async get(storeName, key) {
    const tx = this.db.transaction(storeName, 'readonly');
    const store = tx.objectStore(storeName);

    return new Promise((resolve, reject) => {
      const request = store.get(key);
      request.onsuccess = () => resolve(request.result);
      request.onerror = () => reject(request.error);
    });
  }

  async getAll(storeName) {
    const tx = this.db.transaction(storeName, 'readonly');
    const store = tx.objectStore(storeName);

    return new Promise((resolve, reject) => {
      const request = store.getAll();
      request.onsuccess = () => resolve(request.result);
      request.onerror = () => reject(request.error);
    });
  }

  async update(storeName, data) {
```

```
    const tx = this.db.transaction(storeName, 'readwrite');
    const store = tx.objectStore(storeName);

    return new Promise((resolve, reject) => {
      const request = store.put(data);
      request.onsuccess = () => resolve(request.result);
      request.onerror = () => reject(request.error);
    });
  }

  async delete(storeName, key) {
    const tx = this.db.transaction(storeName, 'readwrite');
    const store = tx.objectStore(storeName);

    return new Promise((resolve, reject) => {
      const request = store.delete(key);
      request.onsuccess = () => resolve(request.result);
      request.onerror = () => reject(request.error);
    });
  }

  async clear(storeName) {
    const tx = this.db.transaction(storeName, 'readwrite');
    const store = tx.objectStore(storeName);

    return new Promise((resolve, reject) => {
      const request = store.clear();
      request.onsuccess = () => resolve(request.result);
      request.onerror = () => reject(request.error);
    });
  }
}

// Initialize database
```

```
export const db = new Database('MyApp', 1);

await db.open([
  {
    name: 'todos',
    keyPath: 'id',
    indexes: [
      { name: 'by-status', keyPath: 'status' },
      { name: 'by-created', keyPath: 'createdAt' }
    ]
  },
  {
    name: 'users',
    keyPath: 'id'
  }
]);
```

Usage:

```
import { db } from '../lib/db.js';

class TodoList extends HTMLElement {
  async connectedCallback() {
    // Load todos from IndexedDB
    this.todos = await db.getAll('todos');
    this.render();

    // Subscribe to changes
    pan.subscribe('todo.added', async ({ todo }) => {
      await db.add('todos', todo);
      this.todos = await db.getAll('todos');
      this.render();
    });

    pan.subscribe('todo.updated', async ({ todo }) => {
      await db.update('todos', todo);
      this.todos = await db.getAll('todos');
      this.render();
    });

    pan.subscribe('todo.deleted', async ({ id }) => {
      await db.delete('todos', id);
      this.todos = await db.getAll('todos');
      this.render();
    });
  }

  render() {
    this.innerHTML = `
      <ul>
        ${this.todos.map(todo => `
          <li>
            <span>${todo.text}</span>
```

```
            <button data-id="${todo.id}">Delete</button>
          </li>
        `).join('')}
      </ul>
    `;
  }
}
```

6.4.2 Cache-First Strategy

Implement cache-first data loading:

```
class DataManager {
  constructor(storeName) {
    this.storeName = storeName;
    this.cache = new Map();
  }

  async get(id) {
    // 1. Check memory cache
    if (this.cache.has(id)) {
      return this.cache.get(id);
    }

    // 2. Check IndexedDB
    const cached = await db.get(this.storeName, id);
    if (cached) {
      this.cache.set(id, cached);
      return cached;
    }

    // 3. Fetch from API
    const data = await this.fetchFromAPI(id);

    // 4. Store in cache and IndexedDB
    this.cache.set(id, data);
    await db.add(this.storeName, data);

    return data;
  }

  async fetchFromAPI(id) {
    const response = await fetch(`/api/${this.storeName}/${id}`);
    return response.json();
  }
```

```
  async refresh(id) {
    // Force refresh from API
    const data = await this.fetchFromAPI(id);

    // Update cache and IndexedDB
    this.cache.set(id, data);
    await db.update(this.storeName, data);

    return data;
  }

  async getAll() {
    // Load from IndexedDB first
    const items = await db.getAll(this.storeName);

    // Cache in memory
    items.forEach(item => {
      this.cache.set(item.id, item);
    });

    return items;
  }
}

// Usage
const userManager = new DataManager('users');

// Always returns fast (from cache if available)
const user = await userManager.get(123);

// Force refresh
const freshUser = await userManager.refresh(123);
```

6.5 Persistence Strategies

6.5.1 localStorage

Simple key-value storage:

```
class PersistentState {
  constructor(key) {
    this.key = key;
    this.state = this.load();
  }

  load() {
    try {
      const data = localStorage.getItem(this.key);
      return data ? JSON.parse(data) : {};
    } catch (error) {
      console.error('Failed to load state:', error);
      return {};
    }
  }

  save() {
    try {
      localStorage.setItem(this.key, JSON.stringify(this.state));
    } catch (error) {
      console.error('Failed to save state:', error);
    }
  }

  get(path) {
    return this.getNestedValue(this.state, path);
  }

  set(path, value) {
    this.setNestedValue(this.state, path, value);
    this.save();
  }

  getNestedValue(obj, path) {
```

```
    return path.split('.').reduce((current, key) => current?.[key], obj);
  }

  setNestedValue(obj, path, value) {
    const keys = path.split('.');
    const lastKey = keys.pop();
    const target = keys.reduce((current, key) => {
      if (!(key in current)) current[key] = {};
      return current[key];
    }, obj);
    target[lastKey] = value;
  }

  clear() {
    this.state = {};
    localStorage.removeItem(this.key);
  }
}

// Usage
const settings = new PersistentState('app-settings');

settings.set('theme', 'dark');
settings.set('user.preferences.notifications', true);

console.log(settings.get('theme'));  // 'dark'
console.log(settings.get('user.preferences.notifications'));  // true
```

6.5.2 sessionStorage

For temporary session data:

```
class SessionState {
  constructor(key) {
    this.key = key;
  }

  set(data) {
    sessionStorage.setItem(this.key, JSON.stringify(data));
  }

  get() {
    const data = sessionStorage.getItem(this.key);
    return data ? JSON.parse(data) : null;
  }

  clear() {
    sessionStorage.removeItem(this.key);
  }
}

// Usage - data persists only for the session
const sessionData = new SessionState('form-draft');

// Save form draft
sessionData.set({ email: 'user@example.com', message: 'Draft...' });

// Restore on page reload (same session)
const draft = sessionData.get();
```

6.5.3 Hybrid Strategy

Combine localStorage and IndexedDB:

```
class HybridStorage {
  constructor(namespace) {
    this.namespace = namespace;
  }

  async set(key, value) {
    const fullKey = `${this.namespace}:${key}`;

    // Store small data in localStorage
    if (this.isSmall(value)) {
      localStorage.setItem(fullKey, JSON.stringify(value));
    } else {
      // Store large data in IndexedDB
      await db.update('storage', { key: fullKey, value });
    }
  }

  async get(key) {
    const fullKey = `${this.namespace}:${key}`;

    // Try localStorage first
    const local = localStorage.getItem(fullKey);
    if (local) {
      return JSON.parse(local);
    }

    // Try IndexedDB
    const result = await db.get('storage', fullKey);
    return result?.value;
  }

  isSmall(value) {
    const str = JSON.stringify(value);
    return str.length < 1024 * 10; // 10KB threshold
```

```
  }

  async clear() {
    // Clear localStorage items
    Object.keys(localStorage).forEach(key => {
      if (key.startsWith(`${this.namespace}:`)) {
        localStorage.removeItem(key);
      }
    });

    // Clear IndexedDB items
    const all = await db.getAll('storage');
    for (const item of all) {
      if (item.key.startsWith(`${this.namespace}:`)) {
        await db.delete('storage', item.key);
      }
    }
  }
}
```

6.6 Offline-First Applications

Build applications that work without connectivity:

6.6.1 Service Worker + State Management

```
// sw.js - Service Worker
self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open('v1').then((cache) => {
      return cache.addAll([
        '/',
        '/index.html',
        '/src/app.js',
        '/',
        // Cache critical assets
      ]);
    })
  );
});

self.addEventListener('fetch', (event) => {
  event.respondWith(
    caches.match(event.request).then((response) => {
      // Return cached version or fetch
      return response || fetch(event.request);
    })
  );
});
```

6.6.2 Sync Queue

Queue operations when offline:

```
// lib/sync-queue.js
class SyncQueue {
  constructor() {
    this.queue = this.loadQueue();
    this.processing = false;

    // Listen for online events
    window.addEventListener('online', () => {
      this.process();
    });

    // Start processing if online
    if (navigator.onLine) {
      this.process();
    }
  }

  loadQueue() {
    const data = localStorage.getItem('sync-queue');
    return data ? JSON.parse(data) : [];
  }

  saveQueue() {
    localStorage.setItem('sync-queue', JSON.stringify(this.queue));
  }

  add(operation) {
    this.queue.push({
      id: Date.now() + Math.random(),
      operation,
      timestamp: Date.now(),
      attempts: 0
    });
```

```
    this.saveQueue();

    if (navigator.onLine) {
      this.process();
    }
  }

  async process() {
    if (this.processing || this.queue.length === 0) {
      return;
    }

    this.processing = true;

    while (this.queue.length > 0 && navigator.onLine) {
      const item = this.queue[0];

      try {
        await this.executeOperation(item.operation);

        // Success - remove from queue
        this.queue.shift();
        this.saveQueue();

        pan.publish('sync.success', { operation: item.operation });
      } catch (error) {
        item.attempts++;

        if (item.attempts >= 3) {
          // Max attempts - remove and report error
          this.queue.shift();
          this.saveQueue();

          pan.publish('sync.failed', {
```

```
            operation: item.operation,
            error: error.message
          });
        } else {
          // Retry later
          break;
        }
      }
    }

    this.processing = false;
  }

  async executeOperation(operation) {
    switch (operation.type) {
      case 'CREATE':
        return this.create(operation.data);
      case 'UPDATE':
        return this.update(operation.data);
      case 'DELETE':
        return this.delete(operation.id);
      default:
        throw new Error(`Unknown operation: ${operation.type}`);
    }
  }

  async create(data) {
    const response = await fetch('/api/items', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(data)
    });

    if (!response.ok) throw new Error('Create failed');
```

```
    return response.json();
  }

  async update(data) {
    const response = await fetch(`/api/items/${data.id}`, {
      method: 'PUT',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(data)
    });

    if (!response.ok) throw new Error('Update failed');
    return response.json();
  }

  async delete(id) {
    const response = await fetch(`/api/items/${id}`, {
      method: 'DELETE'
    });

    if (!response.ok) throw new Error('Delete failed');
  }

  clear() {
    this.queue = [];
    this.saveQueue();
  }

  getStatus() {
    return {
      queued: this.queue.length,
      online: navigator.onLine,
      processing: this.processing
    };
  }
```

```
}

export const syncQueue = new SyncQueue();
```

Usage:

```
import { syncQueue } from '../lib/sync-queue.js';

class TodoManager {
  async addTodo(text) {
    const todo = {
      id: Date.now(),
      text,
      completed: false,
      createdAt: new Date()
    };

    // Save locally immediately
    await db.add('todos', todo);
    pan.publish('todo.added', { todo });

    // Queue for server sync
    if (!navigator.onLine) {
      syncQueue.add({
        type: 'CREATE',
        data: todo
      });

      pan.publish('notification.info', {
        message: 'Saved locally. Will sync when online.'
      });
    } else {
      // Online - sync immediately
      try {
        await this.syncToServer(todo);
      } catch (error) {
        // Failed - add to queue
        syncQueue.add({
          type: 'CREATE',
          data: todo
```

```
        });
      }
    }
  }

  async syncToServer(todo) {
    const response = await fetch('/api/todos', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(todo)
    });

    if (!response.ok) {
      throw new Error('Sync failed');
    }

    const result = await response.json();

    // Update local copy with server ID
    await db.update('todos', { ...todo, serverId: result.id });
  }
}
```

6.7 Summary

This chapter covered state management at every level:

- **Component-Local State**: Instance properties, private fields, and state objects
- **Shared State**: Global state, reactive proxies, and store patterns
- **pan-store**: Built-in state management component
- **IndexedDB**: Large dataset storage and offline capability
- **Persistence**: localStorage, sessionStorage, and hybrid strategies
- **Offline-First**: Service workers, sync queues, and conflict resolution

Choose the simplest solution that meets your needs, then scale up complexity as requirements grow.

6.8 Best Practices

1. **Start with local state**
    - Only share state when necessary
    - Keeps components independent
    - Easier to test and debug
2. **Use IndexedDB for large data**
    - localStorage limited to ~5-10MB
    - IndexedDB can store gigabytes
    - Better performance for large datasets
3. **Implement cache-first strategies**
    - Load from cache immediately
    - Update from server in background
    - Show stale data rather than loading spinner
4. **Queue offline operations**
    - Don’t lose user data
    - Sync when connection restored
    - Show sync status to user
5. **Test offline scenarios**
    - Use DevTools to simulate offline
    - Test sync queue behavior
    - Verify conflict resolution
6. **Monitor storage usage**
    - Check quota before storing
    - Clean up old data
    - Provide clear error messages when full

6.9 Further Reading

For complete state management reference: - *Building with LARC* Chapter 4: State Management - All state patterns and strategies - *Building with LARC* Chapter 18: Data Components - pan-store and pan-idb API reference - *Building with LARC* Appendix E: Recipes and Patterns - State management recipes

7 Advanced Component Patterns

As your LARC applications grow, you'll encounter scenarios that require sophisticated component architectures. This chapter explores advanced patterns that enable code reuse, flexible composition, and optimal performance.

These patterns come from years of component-based development across frameworks. LARC implements them using web standards, making them portable and future-proof.

7.1 Compound Components

Compound components work together as a set, sharing implicit state. Think of HTML's `<select>` and `<option>` elements—they form a cohesive unit.

7.1.1 Basic Compound Component

```
// tabs.js - Container component
class TabGroup extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.activeTab = 0;
  }

  connectedCallback() {
    this.render();
    this.setupTabs();
  }

  setupTabs() {
    // Get all tab headers
    const headers = this.querySelectorAll('tab-header');
    headers.forEach((header, index) => {
      header.addEventListener('click', () => {
        this.activeTab = index;
        this.updateTabs();
      });
    });

    this.updateTabs();
  }

  updateTabs() {
    // Update headers
    const headers = this.querySelectorAll('tab-header');
    headers.forEach((header, index) => {
      header.active = index === this.activeTab;
    });

    // Update panels
```

```
    const panels = this.querySelectorAll('tab-panel');
    panels.forEach((panel, index) => {
      panel.active = index === this.activeTab;
    });
  }

  render() {
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: block;
        }
        .headers {
          display: flex;
          border-bottom: 2px solid #e2e8f0;
        }
        .panels {
          padding: 16px 0;
        }
      </style>
      <div class="headers">
        <slot name="headers"></slot>
      </div>
      <div class="panels">
        <slot name="panels"></slot>
      </div>
    `;
  }
}

// tab-header.js
class TabHeader extends HTMLElement {
  constructor() {
    super();
```

```
    this.attachShadow({ mode: 'open' });
  }

  set active(value) {
    this._active = value;
    this.render();
  }

  connectedCallback() {
    this.render();
  }

  render() {
    this.shadowRoot.innerHTML = `
      <style>
        button {
          padding: 12px 24px;
          border: none;
          background: ${this._active ? '#667eea' : 'transparent'};
          color: ${this._active ? 'white' : '#4a5568'};
          cursor: pointer;
          font-weight: ${this._active ? '600' : '400'};
          transition: all 0.2s;
        }
        button:hover {
          background: ${this._active ? '#5a67d8' : '#f7fafc'};
        }
      </style>
      <button><slot></slot></button>
    `;
  }
}

// tab-panel.js
```

```
class TabPanel extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
  }

  set active(value) {
    this._active = value;
    this.style.display = value ? 'block' : 'none';
  }

  connectedCallback() {
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: block;
        }
      </style>
      <slot></slot>
    `;
  }
}

customElements.define('tab-group', TabGroup);
customElements.define('tab-header', TabHeader);
customElements.define('tab-panel', TabPanel);
```

Usage:

```
<tab-group>
  <tab-header slot="headers">Profile</tab-header>
  <tab-header slot="headers">Settings</tab-header>
  <tab-header slot="headers">Billing</tab-header>

  <tab-panel slot="panels">
    <h2>Profile Content</h2>
    <p>User profile information...</p>
  </tab-panel>

  <tab-panel slot="panels">
    <h2>Settings Content</h2>
    <p>Application settings...</p>
  </tab-panel>

  <tab-panel slot="panels">
    <h2>Billing Content</h2>
    <p>Billing information...</p>
  </tab-panel>
</tab-group>
```

7.1.2 Context API for Compound Components

Share state without prop drilling:

```
// lib/context.js
const contexts = new WeakMap();

export function createContext(defaultValue) {
  return {
    Provider: class extends HTMLElement {
      constructor() {
        super();
        this.value = defaultValue;
        contexts.set(this, this.value);
      }

      provide(value) {
        this.value = value;
        contexts.set(this, value);
        this.notifyConsumers();
      }

      notifyConsumers() {
        const consumers = this.querySelectorAll('[data-context-
        consumer]');
        consumers.forEach(consumer => {
          if (consumer.onContextChange) {
            consumer.onContextChange(this.value);
          }
        });
      }

      connectedCallback() {
        this.innerHTML = `<slot></slot>`;
      }
    },

    Consumer: class extends HTMLElement {
```

```
      connectedCallback() {
        this.setAttribute('data-context-consumer', '');

        // Find provider up the tree
        let provider = this.closest('[data-context-provider]');
        if (provider && contexts.has(provider)) {
          this.onContextChange(contexts.get(provider));
        }
      }

      onContextChange(value) {
        // Override in subclasses
      }
    }
  };
}
```

Usage:

```
// Create context
const ThemeContext = createContext({ theme: 'light' });

// Provider component
class ThemeProvider extends ThemeContext.Provider {
  connectedCallback() {
    super.connectedCallback();
    this.setAttribute('data-context-provider', '');

    this.provide({
      theme: 'light',
      toggleTheme: () => {
        const newTheme = this.value.theme === 'light' ? 'dark' : 'light';
        this.provide({ ...this.value, theme: newTheme });
      }
    });
  }
}

// Consumer component
class ThemedButton extends ThemeContext.Consumer {
  onContextChange(context) {
    this.context = context;
    this.render();
  }

  render() {
    const { theme } = this.context || { theme: 'light' };

    this.innerHTML = `
      <button style="
        background: ${theme === 'dark' ? '#333' : '#fff'};
        color: ${theme === 'dark' ? '#fff' : '#333'};
      ">
```

```
        <slot></slot>
      </button>
    `;
  }
}

customElements.define('theme-provider', ThemeProvider);
customElements.define('themed-button', ThemedButton);
```

```
<theme-provider>
  <themed-button>Light/Dark</themed-button>
  <themed-button>Another Button</themed-button>
</theme-provider>
```

7.2 Higher-Order Components

Higher-order components (HOCs) wrap other components to add functionality.

7.2.1 Mixin Pattern

JavaScript mixins add functionality to classes:

```javascript
// mixins/observable.js
export const ObservableMixin = (Base) => class extends Base {
  constructor() {
    super();
    this._observers = new Map();
  }

  observe(property, callback) {
    if (!this._observers.has(property)) {
      this._observers.set(property, new Set());
    }
    this._observers.get(property).add(callback);

    // Return unobserve function
    return () => {
      this._observers.get(property)?.delete(callback);
    };
  }

  notify(property, value) {
    this._observers.get(property)?.forEach(callback => {
      callback(value);
    });
  }

  set(property, value) {
    this[`_${property}`] = value;
    this.notify(property, value);
  }

  get(property) {
    return this[`_${property}`];
  }
};
```

```
// mixins/resizable.js
export const ResizableMixin = (Base) => class extends Base {
  connectedCallback() {
    super.connectedCallback?.();

    this.resizeObserver = new ResizeObserver((entries) => {
      for (const entry of entries) {
        this.onResize?.(entry.contentRect);
      }
    });

    this.resizeObserver.observe(this);
  }

  disconnectedCallback() {
    super.disconnectedCallback?.();
    this.resizeObserver?.disconnect();
  }
};

// mixins/loading.js
export const LoadingMixin = (Base) => class extends Base {
  constructor() {
    super();
    this._loading = false;
  }

  startLoading() {
    this._loading = true;
    this.setAttribute('loading', '');
    this.onLoadingChange?.(true);
  }
```

```
  stopLoading() {
    this._loading = false;
    this.removeAttribute('loading');
    this.onLoadingChange?.(false);
  }

  get loading() {
    return this._loading;
  }
};
```

Usage:

```
import { ObservableMixin } from './mixins/observable.js';
import { ResizableMixin } from './mixins/resizable.js';
import { LoadingMixin } from './mixins/loading.js';

// Compose multiple mixins
class DataTable extends
        LoadingMixin(ResizableMixin(ObservableMixin(HTMLElement))) {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
  }

  async connectedCallback() {
    super.connectedCallback();

    // Use Observable mixin
    this.observe('data', (data) => {
      console.log('Data changed:', data);
      this.render();
    });

    // Use Loading mixin
    this.startLoading();
    const data = await this.fetchData();
    this.set('data', data);
    this.stopLoading();
  }

  // Use Resizable mixin
  onResize(rect) {
    console.log('Component resized:', rect.width, rect.height);
    this.updateLayout();
  }
```

```
  onLoadingChange(loading) {
    this.render();
  }

  async fetchData() {
    const response = await fetch('/api/data');
    return response.json();
  }

  render() {
    // Render based on state
  }
}

customElements.define('data-table', DataTable);
```

7.2.2 Decorator Pattern

Wrap components to enhance them:

```
// decorators/with-loading.js
export function withLoading(ComponentClass) {
  return class extends ComponentClass {
    constructor() {
      super();
      this._originalConnectedCallback = this.connectedCallback;
    }

    connectedCallback() {
      // Inject loading overlay
      const loadingOverlay = document.createElement('div');
      loadingOverlay.className = 'loading-overlay';
      loadingOverlay.style.cssText = `
        position: absolute;
        top: 0;
        left: 0;
        right: 0;
        bottom: 0;
        background: rgba(255,255,255,0.8);
        display: none;
        align-items: center;
        justify-content: center;
      `;
      loadingOverlay.innerHTML = '<div class="spinner"></div>';

      this.appendChild(loadingOverlay);
      this._loadingOverlay = loadingOverlay;

      // Call original
      if (this._originalConnectedCallback) {
        this._originalConnectedCallback.call(this);
      }
    }
```

```
    showLoading() {
      if (this._loadingOverlay) {
        this._loadingOverlay.style.display = 'flex';
      }
    }

    hideLoading() {
      if (this._loadingOverlay) {
        this._loadingOverlay.style.display = 'none';
      }
    }
  };
}

// Usage
class UserProfile extends HTMLElement {
  async connectedCallback() {
    this.showLoading();

    const user = await fetch('/api/user').then(r => r.json());
    this.render(user);

    this.hideLoading();
  }

  render(user) {
    this.innerHTML = `<h1>${user.name}</h1>`;
  }
}

// Apply decorator
const UserProfileWithLoading = withLoading(UserProfile);
customElements.define('user-profile', UserProfileWithLoading);
```

7.3 Component Composition

Build complex UIs from simple, focused components.

7.3.1 Container/Presentational Pattern

Separate logic from presentation:

```
// Presentational - no logic, just rendering
class UserCard extends HTMLElement {
  set user(value) {
    this._user = value;
    this.render();
  }

  render() {
    if (!this._user) return;

    this.innerHTML = `
      <div class="card">
        <img src="${this._user.avatar}" alt="${this._user.name}">
        <h3>${this._user.name}</h3>
        <p>${this._user.email}</p>
        <button class="follow-btn">Follow</button>
      </div>
    `;

    // Emit events, don't handle logic
    this.querySelector('.follow-btn').addEventListener('click', () => {
      this.dispatchEvent(new CustomEvent('follow', {
        detail: { userId: this._user.id }
      }));
    });
  }
}

// Container - handles logic and data
class UserCardContainer extends HTMLElement {
  async connectedCallback() {
    const userId = this.getAttribute('user-id');

    // Fetch data
```

```
    this.user = await this.fetchUser(userId);

    // Create presentational component
    const card = document.createElement('user-card');
    card.user = this.user;

    // Handle events
    card.addEventListener('follow', (e) => {
      this.followUser(e.detail.userId);
    });

    this.appendChild(card);
  }

  async fetchUser(id) {
    const response = await fetch(`/api/users/${id}`);
    return response.json();
  }

  async followUser(userId) {
    await fetch(`/api/users/${userId}/follow`, { method: 'POST' });
    pan.publish('user.followed', { userId });
  }
}

customElements.define('user-card', UserCard);
customElements.define('user-card-container', UserCardContainer);
```

7.3.2 Render Props Pattern

Pass rendering logic as a slot:

```
class DataProvider extends HTMLElement {
  async connectedCallback() {
    const url = this.getAttribute('url');

    // Render loading state
    this.innerHTML = '<slot name="loading">Loading...</slot>';

    try {
      const response = await fetch(url);
      const data = await response.json();

      // Render with data
      const renderSlot = this.querySelector('[slot="render"]');
      if (renderSlot) {
        renderSlot.data = data;
        this.innerHTML = '';
        this.appendChild(renderSlot);
      }
    } catch (error) {
      // Render error state
      this.innerHTML = `<slot name="error">Error:
        ${error.message}</slot>`;
    }
  }
}

customElements.define('data-provider', DataProvider);
```

Usage:

```
<data-provider url="/api/users">
  <div slot="loading">
    <spinner-component></spinner-component>
  </div>

  <user-list slot="render"></user-list>

  <div slot="error">
    <error-message></error-message>
  </div>
</data-provider>
```

7.4 Slots and Content Projection

Slots are powerful for flexible component composition.

7.4.1 Named Slots

```
class CardComponent extends HTMLElement {
  connectedCallback() {
    this.attachShadow({ mode: 'open' });

    this.shadowRoot.innerHTML = `
      <style>
        .card {
          border: 1px solid #e2e8f0;
          border-radius: 8px;
          overflow: hidden;
        }
        .header {
          background: #f7fafc;
          padding: 16px;
          border-bottom: 1px solid #e2e8f0;
        }
        .body {
          padding: 16px;
        }
        .footer {
          background: #f7fafc;
          padding: 12px 16px;
          border-top: 1px solid #e2e8f0;
          display: flex;
          justify-content: flex-end;
          gap: 8px;
        }
      </style>

      <div class="card">
        <div class="header">
          <slot name="header">Default Header</slot>
        </div>
        <div class="body">
```

```
          <slot></slot>
        </div>
        <div class="footer">
          <slot name="footer"></slot>
        </div>
      </div>
    `;
  }
}

customElements.define('card-component', CardComponent);
```

Usage:

```
<card-component>
  <h2 slot="header">User Profile</h2>

  <!-- Default slot -->
  <p>User profile content goes here...</p>

  <div slot="footer">
    <button>Save</button>
    <button>Cancel</button>
  </div>
</card-component>
```

7.4.2 Slot Change Detection

React to slot content changes:

```
class DynamicList extends HTMLElement {
  connectedCallback() {
    this.attachShadow({ mode: 'open' });

    this.shadowRoot.innerHTML = `
      <style>
        .count { font-weight: bold; color: #667eea; }
      </style>
      <div class="count"></div>
      <slot></slot>
    `;

    // Listen for slot changes
    const slot = this.shadowRoot.querySelector('slot');
    slot.addEventListener('slotchange', () => {
      this.updateCount();
    });

    this.updateCount();
  }

  updateCount() {
    const slot = this.shadowRoot.querySelector('slot');
    const elements = slot.assignedElements();

    const count = this.shadowRoot.querySelector('.count');
    count.textContent = `${elements.length} items`;
  }
}

customElements.define('dynamic-list', DynamicList);
```

Usage:

```
<dynamic-list>
  <div>Item 1</div>
  <div>Item 2</div>
  <div>Item 3</div>
</dynamic-list>

<script>
  const list = document.querySelector('dynamic-list');

  // Add item dynamically
  const newItem = document.createElement('div');
  newItem.textContent = 'Item 4';
  list.appendChild(newItem);
  // Count automatically updates!
</script>
```

7.4.3 Conditional Slots

Show/hide content based on slot presence:

```
class ConditionalCard extends HTMLElement {
  connectedCallback() {
    this.attachShadow({ mode: 'open' });

    const hasHeader = this.querySelector('[slot="header"]') !== null;
    const hasFooter = this.querySelector('[slot="footer"]') !== null;

    this.shadowRoot.innerHTML = `
      <style>
        .card { border: 1px solid #ddd; border-radius: 8px; }
        .header, .footer { background: #f5f5f5; padding: 16px; }
        .body { padding: 16px; }
        .hidden { display: none; }
      </style>

      <div class="card">
        <div class="header ${hasHeader ? '' : 'hidden'}">
          <slot name="header"></slot>
        </div>
        <div class="body">
          <slot></slot>
        </div>
        <div class="footer ${hasFooter ? '' : 'hidden'}">
          <slot name="footer"></slot>
        </div>
      </div>
    `;
  }
}

customElements.define('conditional-card', ConditionalCard);
```

7.5 Dynamic Component Loading

Load components on demand for better performance.

7.5.1 Lazy Loading

```
class LazyLoader extends HTMLElement {
  async connectedCallback() {
    const component = this.getAttribute('component');
    const src = this.getAttribute('src');

    // Show placeholder
    this.innerHTML = '<div>Loading component...</div>';

    try {
      // Dynamically import component
      await import(src);

      // Wait for component to be defined
      await customElements.whenDefined(component);

      // Create and append component
      const element = document.createElement(component);

      // Copy attributes
      Array.from(this.attributes).forEach(attr => {
        if (attr.name !== 'component' && attr.name !== 'src') {
          element.setAttribute(attr.name, attr.value);
        }
      });

      this.innerHTML = '';
      this.appendChild(element);
    } catch (error) {
      this.innerHTML = `<div class="error">Failed to load component:
        ${error.message}</div>`;
    }
  }
}
```

```
customElements.define('lazy-loader', LazyLoader);
```

Usage:

```
<!-- Component loads when added to DOM -->
<lazy-loader
  component="heavy-chart"
  src="/components/heavy-chart.js"
  data-url="/api/chart-data">
</lazy-loader>
```

7.5.2 Intersection Observer for Viewport Loading

Load components when they enter the viewport:

```
class ViewportLoader extends HTMLElement {
  connectedCallback() {
    this.observer = new IntersectionObserver((entries) => {
      entries.forEach(entry => {
        if (entry.isIntersecting && !this.loaded) {
          this.load();
        }
      });
    }, {
      rootMargin: '50px'  // Start loading 50px before visible
    });

    this.observer.observe(this);
  }

  disconnectedCallback() {
    this.observer?.disconnect();
  }

  async load() {
    this.loaded = true;
    const component = this.getAttribute('component');
    const src = this.getAttribute('src');

    await import(src);
    await customElements.whenDefined(component);

    const element = document.createElement(component);
    Array.from(this.attributes).forEach(attr => {
      if (!['component', 'src'].includes(attr.name)) {
        element.setAttribute(attr.name, attr.value);
      }
    });
```

```
    this.appendChild(element);
  }
}

customElements.define('viewport-loader', ViewportLoader);
```

Usage:

```
<!-- Heavy image gallery - only loads when scrolled into view -->
<viewport-loader
  component="image-gallery"
  src="/components/image-gallery.js"
  album-id="123">
</viewport-loader>
```

7.6 Performance Optimization

7.6.1 Virtual Scrolling

Render only visible items in long lists:

```
class VirtualList extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });

    this.items = [];
    this.itemHeight = 50;
    this.visibleCount = 20;
    this.scrollTop = 0;
  }

  set data(items) {
    this.items = items;
    this.render();
  }

  connectedCallback() {
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: block;
          height: 100%;
          overflow-y: auto;
          position: relative;
        }
        .viewport {
          position: relative;
        }
        .item {
          position: absolute;
          left: 0;
          right: 0;
          height: ${this.itemHeight}px;
          display: flex;
```

```
        align-items: center;
        padding: 0 16px;
        border-bottom: 1px solid #eee;
      }
    </style>
    <div class="viewport"></div>
  `;

  this.viewport = this.shadowRoot.querySelector('.viewport');

  this.addEventListener('scroll', () => {
    this.scrollTop = this.scrollTop;
    this.renderVisibleItems();
  });
}

render() {
  if (!this.viewport) return;

  // Set total height
  const totalHeight = this.items.length * this.itemHeight;
  this.viewport.style.height = `${totalHeight}px`;

  this.renderVisibleItems();
}

renderVisibleItems() {
  const startIndex = Math.floor(this.scrollTop / this.itemHeight);
  const endIndex = Math.min(
    startIndex + this.visibleCount,
    this.items.length
  );

  // Clear existing items
```

```
    this.viewport.innerHTML = '';

    // Render only visible items
    for (let i = startIndex; i < endIndex; i++) {
      const item = document.createElement('div');
      item.className = 'item';
      item.style.top = `${i * this.itemHeight}px`;
      item.textContent = this.items[i];

      this.viewport.appendChild(item);
    }
  }
}

customElements.define('virtual-list', VirtualList);
```

Usage:

```
const list = document.createElement('virtual-list');
list.data = Array.from({ length: 10000 }, (_, i) => `Item ${i + 1}`);
list.style.height = '400px';
document.body.appendChild(list);
```

7.6.2 Memoization

Cache expensive computations:

```
class MemoizedComponent extends HTMLElement {
  constructor() {
    super();
    this.cache = new Map();
  }

  memoize(fn, keyFn) {
    return (...args) => {
      const key = keyFn ? keyFn(...args) : JSON.stringify(args);

      if (this.cache.has(key)) {
        return this.cache.get(key);
      }

      const result = fn(...args);
      this.cache.set(key, result);

      return result;
    };
  }

  computeExpensiveValue = this.memoize(
    (data) => {
      // Expensive computation
      console.log('Computing...');
      return data.reduce((acc, val) => acc + val.price, 0);
    },
    (data) => data.map(d => d.id).join(',')
  );

  connectedCallback() {
    const data = [
      { id: 1, price: 100 },
      { id: 2, price: 200 }
```

```
    ];

    // First call - computes
    console.log(this.computeExpensiveValue(data));

    // Second call - cached
    console.log(this.computeExpensiveValue(data));
  }
}
```

7.6.3 Debouncing and Throttling

Limit expensive operations:

```
// lib/performance.js
export function debounce(fn, delay) {
  let timeoutId;

  return function(...args) {
    clearTimeout(timeoutId);

    timeoutId = setTimeout(() => {
      fn.apply(this, args);
    }, delay);
  };
}

export function throttle(fn, limit) {
  let inThrottle;

  return function(...args) {
    if (!inThrottle) {
      fn.apply(this, args);
      inThrottle = true;

      setTimeout(() => {
        inThrottle = false;
      }, limit);
    }
  };
}

// Usage
class SearchBox extends HTMLElement {
  constructor() {
    super();

    // Debounce search - wait for user to stop typing
```

```
    this.handleSearch = debounce(this.search.bind(this), 300);

    // Throttle scroll - limit updates
    this.handleScroll = throttle(this.onScroll.bind(this), 100);
  }

  connectedCallback() {
    this.innerHTML = '<input type="search" placeholder="Search...">';

    this.querySelector('input').addEventListener('input', (e) => {
      this.handleSearch(e.target.value);
    });

    window.addEventListener('scroll', this.handleScroll);
  }

  search(query) {
    console.log('Searching for:', query);
    // Perform search
  }

  onScroll() {
    console.log('Scrolled');
    // Update UI based on scroll
  }
}
```

7.7 Summary

This chapter explored advanced component patterns:

- **Compound Components**: Components that work together as a cohesive unit
- **Higher-Order Components**: Mixins and decorators for code reuse

- **Component Composition**: Container/presentational pattern and render props
- **Slots**: Named slots, slot change detection, and conditional rendering
- **Dynamic Loading**: Lazy loading and viewport-based loading
- **Performance**: Virtual scrolling, memoization, debouncing, and throttling

These patterns enable you to build sophisticated, performant applications while keeping code maintainable and testable.

7.8 Best Practices

1. **Favor composition over inheritance**
    - Build complex components from simple ones
    - Use slots for flexibility
    - Keep components focused
2. **Use mixins for cross-cutting concerns**
    - Observable behavior
    - Resize handling
    - Loading states
3. **Separate logic from presentation**
    - Container components handle data
    - Presentational components handle UI
    - Easier to test and reuse
4. **Lazy load heavy components**
    - Reduce initial bundle size
    - Load on demand or when visible
    - Show loading states
5. **Optimize expensive operations**
    - Memoize pure functions
    - Debounce user input
    - Throttle scroll/resize handlers
    - Use virtual scrolling for long lists
6. **Keep performance in mind**
    - Profile before optimizing

- Measure impact of changes
- Don’t over-optimize prematurely

7.9 Further Reading

For advanced patterns and optimization: - *Building with LARC* Chapter 15: Advanced Patterns - Micro-frontends, plugin systems, middleware - *Building with LARC* Chapter 12: Performance Optimization - Virtual scrolling, lazy loading, profiling - *Building with LARC* Appendix E: Recipes and Patterns - Advanced component patterns

8 Business Logic Patterns

In the previous chapters, we've learned how to build components, communicate via the PAN bus, and manage state. But when building real-world applications, you'll inevitably need to inject your own custom business logic: validation rules, pricing calculations, access control, analytics tracking, and countless other domain-specific concerns.

A common question developers ask when adopting LARC is: *"Where do I put my business logic?"* This chapter explores the architectural patterns for integrating business logic into LARC applications, helping you make informed decisions about code organization and separation of concerns.

8.1 The Philosophy: Separation of Concerns

LARC's architecture naturally encourages a clean separation between:

- **Components**: UI and interaction concerns
- **PAN Bus**: Communication layer
- **Business Logic**: Domain rules and workflows

This separation isn't just academic—it makes your code:

- **Testable**: Logic can be tested independently of UI
- **Maintainable**: Changes to business rules don't require touching components
- **Reusable**: Logic can be shared across multiple components
- **Flexible**: Easy to modify workflows without refactoring components

Let's explore the patterns that make this possible.

8.2 Pattern 1: PAN Bus Listeners (Recommended)

The most common and recommended approach is to create separate modules that listen to PAN bus events and implement your business logic. This pattern treats business logic as a **first-class concern**, separate from both UI components and state management.

8.2.1 When to Use

Use PAN bus listeners when you need to:

- Coordinate behavior across multiple components
- Implement cross-cutting concerns (analytics, logging, validation)
- Add business rules that aren't tied to a specific component
- Keep components generic and reusable

8.2.2 Basic Implementation

Let's build an e-commerce application where we need to enforce business rules around cart operations:

```
// business-logic/cart-rules.js
import { pan } from '@larcjs/core';

class CartBusinessRules {
  constructor() {
    this.maxItemsPerOrder = 50;
    this.maxQuantityPerItem = 10;
  }

  init() {
    // Subscribe to cart events
    pan.subscribe('cart.item.add', this.handleItemAdd.bind(this));
    pan.subscribe('cart.item.update', this.handleItemUpdate.bind(this));
    pan.subscribe('cart.checkout.start', this.handleCheckout.bind(this));
  }

  async handleItemAdd(data) {
    console.log('Business rule: Validating item add', data);

    // Check current cart state
    const currentCart = await pan.request('cart.get');

    // Business Rule 1: Maximum items per order
    if (currentCart.items.length >= this.maxItemsPerOrder) {
      pan.publish('cart.error', {
        code: 'MAX_ITEMS_EXCEEDED',
        message: `Cannot add more than ${this.maxItemsPerOrder} items to
         cart`
      });
      return;
    }

    // Business Rule 2: Check inventory
    const available = await this.checkInventory(data.product.id);
```

```
    if (!available || available < data.quantity) {
      pan.publish('cart.error', {
        code: 'INSUFFICIENT_INVENTORY',
        message: 'This item is currently out of stock',
        product: data.product
      });
      return;
    }

    // Business Rule 3: Apply pricing
    const pricing = await this.calculatePrice(data.product,
        data.quantity);

    // All validations passed - allow the add and publish enriched data
    pan.publish('cart.item.validated', {
      ...data,
      pricing,
      timestamp: Date.now()
    });
  }

  async handleItemUpdate(data) {
    // Business Rule: Quantity limits
    if (data.quantity > this.maxQuantityPerItem) {
      pan.publish('cart.error', {
        code: 'MAX_QUANTITY_EXCEEDED',
        message: `Maximum ${this.maxQuantityPerItem} per item`
      });
      return;
    }

    // Check inventory for new quantity
    const available = await this.checkInventory(data.productId);
    if (available < data.quantity) {
      pan.publish('cart.error', {
```

```
        code: 'INSUFFICIENT_INVENTORY',
        message: `Only ${available} available`,
        available
      });
      return;
    }

    pan.publish('cart.item.update.validated', data);
  }

  async handleCheckout(data) {
    // Business Rule: Minimum order value
    const cart = await pan.request('cart.get');
    const total = cart.items.reduce((sum, item) => sum + item.total, 0);

    if (total < 10) {
      pan.publish('checkout.error', {
        code: 'MINIMUM_ORDER_NOT_MET',
        message: 'Minimum order value is $10',
        current: total,
        required: 10
      });
      return;
    }

    // Business Rule: User must be logged in
    const user = await pan.request('auth.user.get');
    if (!user) {
      pan.publish('checkout.error', {
        code: 'AUTH_REQUIRED',
        message: 'Please log in to continue'
      });
      return;
    }
```

```
    pan.publish('checkout.validated', { cart, user });
  }

  async checkInventory(productId) {
    // In real app, this would call your backend
    const response = await fetch(`/api/inventory/${productId}`);
    const data = await response.json();
    return data.available;
  }

  async calculatePrice(product, quantity) {
    // Apply business logic: bulk discounts, promotions, etc.
    let unitPrice = product.price;

    // Bulk discount: 10% off for 5+ items
    if (quantity >= 5) {
      unitPrice = unitPrice * 0.9;
    }

    // TODO: Check for active promotions
    // TODO: Apply user-specific pricing

    return {
      unitPrice,
      quantity,
      subtotal: unitPrice * quantity,
      discount: quantity >= 5 ? (product.price - unitPrice) * quantity :
        0
    };
  }
}

// Initialize and export
const cartRules = new CartBusinessRules();
```

```
export default cartRules;
```

Now in your main application file:

```
// app.js
import { pan } from '@larcjs/core';
import cartRules from './business-logic/cart-rules.js';

// Initialize business logic
cartRules.init();

// Your components just publish events - the business logic handles the
        rest
// No business logic in components themselves!
```

Your components remain simple and focused on UI:

```
// components/product-card.js
class ProductCard extends HTMLElement {
  // ... component setup ...

  handleAddToCart() {
    // Just publish the event - business logic will validate
    pan.publish('cart.item.add', {
      product: this.product,
      quantity: this.quantity
    });

    // Show optimistic UI
    this.showAddingState();
  }

  connectedCallback() {
    super.connectedCallback();

    // Listen for validation results
    this.unsubscribers = [
      pan.subscribe('cart.item.validated', (data) => {
        if (data.product.id === this.product.id) {
          this.showSuccess();
        }
      }),

      pan.subscribe('cart.error', (error) => {
        this.showError(error.message);
      })
    ];
  }
}
```

8.2.3 Advantages

This pattern provides several key benefits:

1. **Separation of Concerns**: Components handle UI, business logic modules handle rules
2. **Easy Testing**: Test business logic without rendering components
3. **Centralized Rules**: All cart rules in one place, easy to modify
4. **Reusable**: Multiple components can trigger the same logic
5. **Flexible**: Easy to add, remove, or modify rules

8.2.4 Advanced: Composable Business Logic

For larger applications, you can compose multiple business logic modules:

```
// business-logic/index.js
import { pan } from '@larcjs/core';
import cartRules from './cart-rules.js';
import pricingRules from './pricing-rules.js';
import inventoryRules from './inventory-rules.js';
import analyticsRules from './analytics-rules.js';

export function initBusinessLogic() {
  console.log('Initializing business logic...');

  // Initialize all business logic modules
  cartRules.init();
  pricingRules.init();
  inventoryRules.init();
  analyticsRules.init();

  console.log('Business logic ready');
}
```

```
// app.js
import { initBusinessLogic } from './business-logic/index.js';

// Single call to initialize all business logic
initBusinessLogic();
```

8.3 Pattern 2: Extending Components

Sometimes you need to add business logic directly to a component, especially when:

- The logic is specific to one component type
- You need to override component behavior
- You're creating specialized versions of generic components

8.3.1 When to Use

Use component extension when:

- Logic is tightly coupled to component rendering
- You need access to component internals (Shadow DOM, private methods)
- Creating specialized variants of base components
- Logic doesn't need to be shared across different component types

8.3.2 Implementation

Let's extend a generic product card with business-specific behavior:

```
// components/base/product-card.js
export class ProductCard extends HTMLElement {
  connectedCallback() {
    this.attachShadow({ mode: 'open' });
    this.render();
  }

  render() {
    this.shadowRoot.innerHTML = `
      <style>
        /* Base styles */
      </style>
      <div class="card">
        <img src="${this.product.image}" alt="${this.product.name}">
        <h3>${this.product.name}</h3>
        <p class="price">${this.formatPrice(this.product.price)}</p>
        <button class="add-to-cart">Add to Cart</button>
      </div>
    `;

    this.shadowRoot.querySelector('.add-to-cart')
      .addEventListener('click', () => this.handleAddToCart());
  }

  handleAddToCart() {
    pan.publish('cart.item.add', {
      product: this.product,
      quantity: 1
    });
  }

  formatPrice(price) {
    return `$${price.toFixed(2)}`;
  }
```

```
  get product() {
    return JSON.parse(this.getAttribute('product'));
  }
}

customElements.define('product-card', ProductCard);
```

Now extend it with business-specific logic:

```
// components/premium-product-card.js
import { ProductCard } from './base/product-card.js';

export class PremiumProductCard extends ProductCard {
  connectedCallback() {
    super.connectedCallback();

    // Add business-specific subscriptions
    this._unsubscribers = [
      pan.subscribe('pricing.update', this.handlePriceUpdate.bind(this)),
      pan.subscribe('user.tier.changed',
        this.handleTierChange.bind(this))
    ];

    // Initialize premium features
    this.loadMemberPricing();
  }

  async loadMemberPricing() {
    const user = await pan.request('auth.user.get');
    if (user?.tier === 'premium') {
      this.applyPremiumDiscount();
    }
  }

  applyPremiumDiscount() {
    // Business Rule: 15% discount for premium members
    const discount = 0.15;
    const originalPrice = this.product.price;
    const discountedPrice = originalPrice * (1 - discount);

    this.product.price = discountedPrice;
    this.product.originalPrice = originalPrice;
```

```
    this.render(); // Re-render with new price
  }

  render() {
    // Call parent render
    super.render();

    // Add premium badge if applicable
    if (this.product.originalPrice) {
      this.addPremiumBadge();
    }
  }

  addPremiumBadge() {
    const badge = document.createElement('div');
    badge.className = 'premium-badge';
    badge.innerHTML = `
      <style>
        .premium-badge {
          position: absolute;
          top: 10px;
          right: 10px;
          background: gold;
          color: black;
          padding: 5px 10px;
          border-radius: 3px;
          font-weight: bold;
        }
        .original-price {
          text-decoration: line-through;
          color: #999;
          font-size: 0.9em;
        }
      </style>
```

```
      <span>Premium Member</span>
      <div class="original-price">
        ${this.formatPrice(this.product.originalPrice)}
      </div>
    `;

    this.shadowRoot.querySelector('.card').prepend(badge);
  }

  async handleAddToCart() {
    // Business validation before adding
    const canAddPremiumItem = await this.validatePremiumAccess();

    if (!canAddPremiumItem) {
      pan.publish('app.error', {
        message: 'Premium membership required for this product'
      });
      return;
    }

    // Track premium conversions
    this.trackPremiumConversion();

    // Call parent behavior
    super.handleAddToCart();
  }

  async validatePremiumAccess() {
    if (!this.product.premiumOnly) return true;

    const user = await pan.request('auth.user.get');
    return user?.tier === 'premium';
  }
```

```
  trackPremiumConversion() {
    pan.publish('analytics.track', {
      event: 'premium_product_add_to_cart',
      product: this.product.id,
      price: this.product.price,
      discount: this.product.originalPrice - this.product.price
    });
  }

  handlePriceUpdate(data) {
    if (data.productId === this.product.id) {
      this.product.price = data.newPrice;
      this.render();
    }
  }

  handleTierChange(data) {
    // User tier changed - recalculate pricing
    this.loadMemberPricing();
  }

  disconnectedCallback() {
    // Clean up subscriptions
    this._unsubscribers.forEach(unsub => unsub());
    super.disconnectedCallback?.();
  }
}

customElements.define('premium-product-card', PremiumProductCard);
```

8.3.3 When This Makes Sense

Component extension works well when:

1. **The logic changes how the component renders or behaves**
2. **You need multiple variants of a base component** (premium, free, guest, etc.)
3. **Business logic is closely tied to component lifecycle**

However, be cautious: overuse of extension can lead to:

- Tight coupling between business logic and UI
- Harder to test business rules independently
- Duplication if multiple components need the same logic

8.4 Pattern 3: Wrapper Components

Wrapper components let you add behavior around existing components without modifying them. This is useful when you want to:

- Add behavior to third-party components
- Keep base components pristine
- Compose behaviors dynamically

8.4.1 Implementation

```
// components/business-wrapper.js
class BusinessWrapper extends HTMLElement {
  connectedCallback() {
    this.attachShadow({ mode: 'open' });

    // Intercept events from slotted content
    this.addEventListener('add-to-cart',
        this.handleBusinessLogic.bind(this));

    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: block;
        }
        .validation-message {
          color: red;
          padding: 10px;
          background: #fee;
          border-radius: 4px;
          margin-bottom: 10px;
        }
        .validation-message.hidden {
          display: none;
        }
      </style>
      <div class="validation-message hidden"></div>
      <slot></slot>
    `;
  }

  async handleBusinessLogic(e) {
    // Stop the event from propagating immediately
    e.stopPropagation();
```

```
    // Apply business validation
    const validation = await this.validateBusinessRules(e.detail);

    if (!validation.valid) {
      this.showError(validation.message);
      return;
    }

    // Validation passed - let the event continue
    pan.publish('cart.item.add', e.detail);
  }

  async validateBusinessRules(data) {
    // Check user eligibility
    const user = await pan.request('auth.user.get');
    if (!user) {
      return {
        valid: false,
        message: 'Please log in to add items to cart'
      };
    }

    // Check age restriction
    if (data.product.ageRestricted && user.age < 21) {
      return {
        valid: false,
        message: 'This product requires age verification (21+)'
      };
    }

    // Check geographic restriction
    if (data.product.geoRestricted && !this.isAllowedRegion(user.region))
        {
      return {
        valid: false,
```

```
        message: 'This product is not available in your region'
      };
    }

    return { valid: true };
  }

  isAllowedRegion(region) {
    // Business logic for regional restrictions
    const allowedRegions = ['US', 'CA', 'UK'];
    return allowedRegions.includes(region);
  }

  showError(message) {
    const errorEl = this.shadowRoot.querySelector('.validation-message');
    errorEl.textContent = message;
    errorEl.classList.remove('hidden');

    setTimeout(() => {
      errorEl.classList.add('hidden');
    }, 5000);
  }
}

customElements.define('business-wrapper', BusinessWrapper);
```

Usage:

```
<!-- Wrap any component with business logic -->
<business-wrapper>
  <product-card product-id="123"></product-card>
</business-wrapper>

<business-wrapper>
  <quick-buy-button product-id="456"></quick-buy-button>
</business-wrapper>
```

The wrapper intercepts events and applies business logic **without modifying** the wrapped components.

8.5 Pattern 4: Behavior Mixins

Mixins let you share behavior across multiple component types. This is useful for cross-cutting concerns like analytics, logging, or validation.

8.5.1 Implementation

```
// mixins/analytics-mixin.js
export const AnalyticsMixin = (BaseClass) => class extends BaseClass {
  track(event, data = {}) {
    pan.publish('analytics.track', {
      event,
      data,
      component: this.tagName.toLowerCase(),
      timestamp: Date.now(),
      ...this.getAnalyticsContext()
    });
  }

  trackInteraction(element, action) {
    this.track(`${element}.${action}`, {
      element,
      action
    });
  }

  getAnalyticsContext() {
    // Add common context to all analytics events
    return {
      page: window.location.pathname,
      referrer: document.referrer
    };
  }

  connectedCallback() {
    super.connectedCallback?.();
    this.track('component.mounted', { id: this.id });
  }

  disconnectedCallback() {
    this.track('component.unmounted', { id: this.id });
```

```
    super.disconnectedCallback?.();
  }
};
```

```
// mixins/validation-mixin.js
export const ValidationMixin = (BaseClass) => class extends BaseClass {
  async validate(data, rules) {
    const errors = [];

    for (const [field, rule] of Object.entries(rules)) {
      const value = data[field];

      if (rule.required && !value) {
        errors.push(`${field} is required`);
      }

      if (rule.min && value < rule.min) {
        errors.push(`${field} must be at least ${rule.min}`);
      }

      if (rule.max && value > rule.max) {
        errors.push(`${field} must be at most ${rule.max}`);
      }

      if (rule.pattern && !rule.pattern.test(value)) {
        errors.push(`${field} is invalid`);
      }

      if (rule.custom) {
        const customError = await rule.custom(value, data);
        if (customError) errors.push(customError);
      }
    }

    return {
      valid: errors.length === 0,
      errors
    };
```

```
  }

  showValidationErrors(errors) {
    pan.publish('validation.errors', {
      component: this.tagName.toLowerCase(),
      errors
    });
  }
};
```

Use mixins to compose behavior:

```
import { AnalyticsMixin } from './mixins/analytics-mixin.js';
import { ValidationMixin } from './mixins/validation-mixin.js';

class CheckoutForm extends ValidationMixin(AnalyticsMixin(HTMLElement)) {
  async handleSubmit() {
    // Use validation from mixin
    const validation = await this.validate(this.formData, {
      email: {
        required: true,
        pattern: /^[^\s@]+@[^\s@]+\.[^\s@]+$/
      },
      cardNumber: {
        required: true,
        custom: async (value) => {
          const valid = await this.validateCard(value);
          return valid ? null : 'Invalid card number';
        }
      }
    });

    if (!validation.valid) {
      this.showValidationErrors(validation.errors);
      return;
    }

    // Use analytics from mixin
    this.track('checkout.submit', {
      amount: this.total,
      items: this.items.length
    });

    // Process checkout
    this.processOrder();
  }
```

```
}
```

8.6 Pattern 5: Service Layer

For complex business logic, create a dedicated service layer that components and PAN listeners can both use:

```
// services/pricing-service.js
class PricingService {
  async calculatePrice(product, quantity, user) {
    let price = product.basePrice;

    // Business Rule: Volume discounts
    if (quantity >= 10) price *= 0.85;
    else if (quantity >= 5) price *= 0.90;

    // Business Rule: Member discounts
    if (user?.tier === 'premium') {
      price *= 0.85;
    } else if (user?.tier === 'gold') {
      price *= 0.90;
    }

    // Business Rule: Active promotions
    const promotions = await this.getActivePromotions(product.id);
    for (const promo of promotions) {
      price = this.applyPromotion(price, promo);
    }

    return {
      unitPrice: price,
      quantity,
      subtotal: price * quantity,
      savings: (product.basePrice - price) * quantity
    };
  }

  async getActivePromotions(productId) {
    const response = await fetch(`/api/promotions?product=${productId}`);
    return response.json();
  }
```

```
  applyPromotion(price, promotion) {
    if (promotion.type === 'percentage') {
      return price * (1 - promotion.value / 100);
    } else if (promotion.type === 'fixed') {
      return Math.max(0, price - promotion.value);
    }
    return price;
  }

  async getTax(subtotal, region) {
    const taxRates = {
      'CA': 0.0725,
      'NY': 0.08,
      'TX': 0.0625
    };

    return subtotal * (taxRates[region] || 0);
  }
}

export default new PricingService();
```

Use the service from both components and PAN listeners:

```
// In a component
import pricingService from './services/pricing-service.js';

class ProductCard extends HTMLElement {
  async updatePrice() {
    const user = await pan.request('auth.user.get');
    const pricing = await pricingService.calculatePrice(
      this.product,
      this.quantity,
      user
    );

    this.displayPrice(pricing);
  }
}
```

```
// In business logic
import pricingService from './services/pricing-service.js';

class CartBusinessLogic {
  init() {
    pan.subscribe('cart.item.add', async (data) => {
      const user = await pan.request('auth.user.get');
      const pricing = await pricingService.calculatePrice(
        data.product,
        data.quantity,
        user
      );

      pan.publish('cart.item.priced', { ...data, pricing });
    });
  }
}
```

8.7 Decision Matrix

Here's how to choose the right pattern:

Scenario	Recommended Pattern	Why
Cross-component coordination	PAN Bus Listeners	Decoupled, flexible
Analytics/logging	Mixins	Reusable across all components
Validation before actions	PAN Bus Listeners	Centralized rules
Component-specific UI logic	Extend Component	Access to internals
Add behavior to third-party components	Wrapper	Non-invasive
Complex business calculations	Service Layer	Testable, reusable
Component variants (premium, free)	Extend Component	Clear inheritance
Feature flags / A-B testing	Wrapper or PAN Listeners	Easy to toggle

8.8 Real-World Example: E-Commerce Checkout

Let’s see how these patterns work together in a complete checkout flow:

```
// services/checkout-service.js
class CheckoutService {
  async processOrder(cart, paymentInfo, shippingInfo) {
    // Complex business logic
    const pricing = await this.calculateFinalPricing(cart);
    const shipping = await this.calculateShipping(cart, shippingInfo);
    const tax = await this.calculateTax(pricing.subtotal,
        shippingInfo.state);

    return {
      items: cart.items,
      pricing,
      shipping,
      tax,
      total: pricing.subtotal + shipping.cost + tax
    };
  }

  async calculateFinalPricing(cart) {
    // Apply all discounts, coupons, etc.
    let subtotal = 0;
    let savings = 0;

    for (const item of cart.items) {
      const itemPricing = await pricingService.calculatePrice(
        item.product,
        item.quantity,
        cart.user
      );
      subtotal += itemPricing.subtotal;
      savings += itemPricing.savings;
    }

    return { subtotal, savings };
```

```
  }

  async calculateShipping(cart, shippingInfo) {
    // Shipping business rules
    if (cart.total >= 50) {
      return { method: 'standard', cost: 0, freeShipping: true };
    }

    const weight = cart.items.reduce((sum, item) => sum + item.weight,
        0);
    const zone = this.getShippingZone(shippingInfo.state);

    return {
      method: 'standard',
      cost: this.calculateShippingCost(weight, zone),
      freeShipping: false
    };
  }

  calculateShippingCost(weight, zone) {
    const baseRate = { 1: 5, 2: 7, 3: 10 };
    return baseRate[zone] + (weight > 5 ? (weight - 5) * 0.5 : 0);
  }

  getShippingZone(state) {
    const zones = {
      1: ['CA', 'OR', 'WA'],
      2: ['NV', 'AZ', 'UT', 'ID'],
      3: [] // All other states
    };

    for (const [zone, states] of Object.entries(zones)) {
      if (states.includes(state)) return parseInt(zone);
    }
    return 3;
```

```
  }

  async calculateTax(subtotal, state) {
    return pricingService.getTax(subtotal, state);
  }
}

export default new CheckoutService();
```

```
// business-logic/checkout-rules.js
import checkoutService from '../services/checkout-service.js';

class CheckoutBusinessRules {
  init() {
    pan.subscribe('checkout.start', this.handleCheckoutStart.bind(this));
    pan.subscribe('checkout.submit',
        this.handleCheckoutSubmit.bind(this));
  }

  async handleCheckoutStart(data) {
    // Business validations
    const cart = await pan.request('cart.get');
    const user = await pan.request('auth.user.get');

    // Validation 1: Cart not empty
    if (!cart.items.length) {
      pan.publish('checkout.error', {
        code: 'EMPTY_CART',
        message: 'Your cart is empty'
      });
      return;
    }

    // Validation 2: User logged in
    if (!user) {
      pan.publish('checkout.error', {
        code: 'AUTH_REQUIRED',
        message: 'Please log in to continue'
      });
      return;
    }

    // Validation 3: Inventory check
```

```
    for (const item of cart.items) {
      const available = await this.checkInventory(item.product.id);
      if (available < item.quantity) {
        pan.publish('checkout.error', {
          code: 'INSUFFICIENT_INVENTORY',
          message: `Only ${available} of "${item.product.name}"
        available`,
          item
        });
        return;
      }
    }

    // All validations passed
    pan.publish('checkout.validated', { cart, user });
  }

  async handleCheckoutSubmit(data) {
    try {
      // Process order through service
      const order = await checkoutService.processOrder(
        data.cart,
        data.paymentInfo,
        data.shippingInfo
      );

      // Submit to backend
      const response = await fetch('/api/orders', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(order)
      });

      if (!response.ok) {
        throw new Error('Order submission failed');
```

```
      }

      const result = await response.json();

      // Success
      pan.publish('checkout.success', {
        orderId: result.orderId,
        order: result
      });

      // Clear cart
      pan.publish('cart.clear');

    } catch (error) {
      pan.publish('checkout.error', {
        code: 'SUBMISSION_FAILED',
        message: 'Unable to process order. Please try again.',
        error
      });
    }
  }

  async checkInventory(productId) {
    const response = await fetch(`/api/inventory/${productId}`);
    const data = await response.json();
    return data.available;
  }
}

export default new CheckoutBusinessRules();
```

The checkout component stays simple:

```
// components/checkout-form.js
class CheckoutForm extends HTMLElement {
  connectedCallback() {
    this.attachShadow({ mode: 'open' });
    this.render();
    this.attachEventListeners();
    this.subscribeToEvents();
  }

  subscribeToEvents() {
    this._unsubscribers = [
      pan.subscribe('checkout.validated', () => {
        this.showCheckoutForm();
      }),

      pan.subscribe('checkout.error', (error) => {
        this.showError(error.message);
      }),

      pan.subscribe('checkout.success', (data) => {
        this.showSuccess(data.orderId);
      })
    ];
  }

  handleSubmit(e) {
    e.preventDefault();

    // Just collect data and publish - business logic handles the rest
    pan.publish('checkout.submit', {
      cart: this.cart,
      paymentInfo: this.getPaymentInfo(),
      shippingInfo: this.getShippingInfo()
    });
```

```
    this.showProcessing();
  }

  // UI methods only - no business logic
  showCheckoutForm() { /* ... */ }
  showError(message) { /* ... */ }
  showSuccess(orderId) { /* ... */ }
  showProcessing() { /* ... */ }
}
```

8.9 Testing Business Logic

One of the biggest advantages of separating business logic is testability. Here's how to test each pattern:

8.9.1 Testing PAN Bus Listeners

```
// __tests__/cart-rules.test.js
import { describe, it, expect, beforeEach, vi } from 'vitest';
import { pan } from '@larcjs/core';
import cartRules from '../business-logic/cart-rules.js';

describe('Cart Business Rules', () => {
  beforeEach(() => {
    // Reset PAN bus between tests
    pan.clear();
    cartRules.init();
  });

  it('should reject adding more than max items', async () => {
    // Mock cart with max items
    pan.respond('cart.get', () => ({
      items: new Array(50).fill({})
    }));

    const errorHandler = vi.fn();
    pan.subscribe('cart.error', errorHandler);

    // Try to add another item
    await pan.publish('cart.item.add', {
      product: { id: 1, name: 'Test' },
      quantity: 1
    });

    expect(errorHandler).toHaveBeenCalledWith({
      code: 'MAX_ITEMS_EXCEEDED',
      message: expect.stringContaining('50 items')
    });
  });

  it('should apply bulk discount for 5+ items', async () => {
```

```
    const validated = vi.fn();
    pan.subscribe('cart.item.validated', validated);

    await pan.publish('cart.item.add', {
      product: { id: 1, name: 'Test', price: 100 },
      quantity: 5
    });

    expect(validated).toHaveBeenCalledWith(
      expect.objectContaining({
        pricing: expect.objectContaining({
          unitPrice: 90, // 10% discount
          discount: 50
        })
      })
    );
  });
});
```

8.9.2 Testing Services

```
// __tests__/pricing-service.test.js
import { describe, it, expect } from 'vitest';
import pricingService from '../services/pricing-service.js';

describe('Pricing Service', () => {
  it('should apply volume discount', async () => {
    const product = { basePrice: 100 };
    const pricing = await pricingService.calculatePrice(product, 10,
        null);

    expect(pricing.unitPrice).toBe(85); // 15% off for 10+
    expect(pricing.subtotal).toBe(850);
  });

  it('should stack member and volume discounts', async () => {
    const product = { basePrice: 100 };
    const user = { tier: 'premium' };

    const pricing = await pricingService.calculatePrice(product, 10,
        user);

    // 15% volume + 15% premium = 72.25
    expect(pricing.unitPrice).toBe(72.25);
  });
});
```

8.10 Best Practices

8.10.1 1. Keep Components Dumb

Components should focus on UI and user interaction. They publish events but don't implement business rules.

Good:

```
handleAddToCart() {
  pan.publish('cart.item.add', { product: this.product });
}
```

Bad:

```
async handleAddToCart() {
  // Business logic in component - hard to test and reuse
  const inventory = await fetch('/api/inventory');
  if (inventory < this.quantity) {
    alert('Out of stock');
    return;
  }

  const user = await fetch('/api/user');
  if (user.age < 21 && this.product.ageRestricted) {
    alert('Age restricted');
    return;
  }

  // ... more business logic
}
```

8.10.2 2. Use Services for Complex Logic

If business logic involves multiple steps, calculations, or external APIs, put it in a service:

```
// Good: Service handles complexity
const pricing = await pricingService.calculatePrice(product, quantity,
        user);

// Bad: Business logic scattered across components and PAN listeners
const basePrice = product.price;
const volumeDiscount = quantity >= 10 ? 0.15 : 0;
const memberDiscount = user?.tier === 'premium' ? 0.15 : 0;
// ... etc
```

8.10.3 3. Make Business Logic Observable

Use PAN bus to make business logic transparent:

```
class OrderProcessor {
  async processOrder(order) {
    pan.publish('order.processing.start', { orderId: order.id });

    try {
      await this.validateOrder(order);
      pan.publish('order.validated', { orderId: order.id });

      await this.chargePayment(order);
      pan.publish('order.charged', { orderId: order.id });

      await this.createShipment(order);
      pan.publish('order.shipped', { orderId: order.id });

      pan.publish('order.complete', { orderId: order.id });
    } catch (error) {
      pan.publish('order.failed', { orderId: order.id, error });
    }
  }
}
```

Now other parts of your app can react to these events (analytics, notifications, UI updates, etc.).

8.10.4 4. Document Business Rules

Make business rules explicit and documented:

```
/**
 * Shopping Cart Business Rules
 *
 * 1. Maximum 50 items per order
 * 2. Maximum 10 quantity per item
 * 3. Free shipping over $50
 * 4. Volume discounts:
 *    - 5-9 items: 10% off
 *    - 10+ items: 15% off
 * 5. Member discounts:
 *    - Premium: 15% off
 *    - Gold: 10% off
 * 6. Minimum order value: $10
 */
class CartBusinessRules {
  // Implementation
}
```

8.10.5 5. Use Feature Flags

Make business logic toggleable:

```
class CheckoutRules {
  constructor() {
    this.features = {
      guestCheckout: true,
      expressCheckout: false,
      digitalWallet: true
    };
  }

  async handleCheckout(data) {
    if (!this.features.guestCheckout && !data.user) {
      pan.publish('checkout.error', {
        message: 'Account required for checkout'
      });
      return;
    }

    // ... rest of logic
  }
}
```

8.11 Summary

When integrating business logic into LARC applications:

1. **Default to PAN Bus listeners** for most business logic - it's decoupled, testable, and flexible
2. **Use services** for complex calculations and workflows
3. **Extend components** only when logic is tightly coupled to UI
4. **Use mixins** for cross-cutting concerns like analytics
5. **Wrap components** when adding behavior to third-party code
6. **Keep components dumb** - they publish events, business logic handles the rest

This separation of concerns makes your application:

- **Easier to test** - business logic without rendering components
- **More maintainable** - business rules in one place
- **More flexible** - easy to change rules without touching UI
- **More reusable** - logic can be shared across components

In the next chapter, we'll explore routing and navigation, building on these patterns to create complete single-page applications.

8.12 Further Reading

For business logic and architecture patterns: - *Building with LARC* Chapter 15: Advanced Patterns - Architecture patterns and middleware - *Building with LARC* Chapter 4: State Management - State management strategies - *Building with LARC* Appendix E: Recipes and Patterns - Design patterns and anti-patterns

9 Routing and Navigation

Client-side routing enables single-page applications (SPAs) to feel like multi-page websites without full page reloads. LARC provides routing through web standards and the PAN bus, keeping things simple and framework-free.

9.1 Client-Side Routing Basics

Client-side routing intercepts link clicks and updates the URL without reloading:

```
// lib/router.js
class Router {
  constructor() {
    this.routes = new Map();
    this.currentRoute = null;

    // Intercept link clicks
    document.addEventListener('click', (e) => {
      if (e.target.matches('a[href^="/"]')) {
        e.preventDefault();
        this.navigate(e.target.getAttribute('href'));
      }
    });

    // Handle browser back/forward
    window.addEventListener('popstate', () => {
      this.handleRoute(window.location.pathname);
    });
  }

  register(path, handler) {
    this.routes.set(path, handler);
  }

  navigate(path, state = {}) {
    window.history.pushState(state, '', path);
    this.handleRoute(path);

    // Publish navigation event
    pan.publish('router.navigated', { path, state });
  }

  handleRoute(path) {
    // Find matching route
```

```
    for (const [pattern, handler] of this.routes) {
      const params = this.matchRoute(pattern, path);
      if (params) {
        this.currentRoute = { path, pattern, params };
        handler(params);
        return;
      }
    }

    // 404 - no match
    pan.publish('router.not-found', { path });
  }

  matchRoute(pattern, path) {
    // Simple pattern matching
    const patternParts = pattern.split('/').filter(Boolean);
    const pathParts = path.split('/').filter(Boolean);

    if (patternParts.length !== pathParts.length) {
      return null;
    }

    const params = {};

    for (let i = 0; i < patternParts.length; i++) {
      const patternPart = patternParts[i];
      const pathPart = pathParts[i];

      if (patternPart.startsWith(':')) {
        // Dynamic segment
        params[patternPart.slice(1)] = pathPart;
      } else if (patternPart !== pathPart) {
        // Mismatch
        return null;
```

```
      }
    }

    return params;
  }

  start() {
    this.handleRoute(window.location.pathname);
  }
}

export const router = new Router();
```

Usage:

```
import { router } from './lib/router.js';

// Register routes
router.register('/', () => {
  document.getElementById('app').innerHTML = '<home-page></home-page>';
});

router.register('/about', () => {
  document.getElementById('app').innerHTML = '<about-page></about-page>';
});

router.register('/users/:id', (params) => {
  const page = document.createElement('user-page');
  page.setAttribute('user-id', params.id);
  document.getElementById('app').innerHTML = '';
  document.getElementById('app').appendChild(page);
});

// Start router
router.start();
```

9.2 The pan-router Component

LARC provides a declarative router component:

```
<pan-router>
  <pan-route path="/" component="home-page"></pan-route>
  <pan-route path="/about" component="about-page"></pan-route>
  <pan-route path="/users/:id" component="user-page"></pan-route>
  <pan-route path="/posts/:postId/comments/:commentId"
        component="comment-page"></pan-route>
  <pan-route path="*" component="not-found-page"></pan-route>
</pan-router>
```

Implementation:

```
class PanRouter extends HTMLElement {
  connectedCallback() {
    this.routes = Array.from(this.querySelectorAll('pan-
        route')).map(route => ({
      path: route.getAttribute('path'),
      component: route.getAttribute('component'),
      guard: route.getAttribute('guard')
    }));

    // Create outlet
    this.outlet = document.createElement('div');
    this.outlet.className = 'router-outlet';
    this.appendChild(this.outlet);

    // Listen for navigation
    pan.subscribe('router.navigate', ({ path, params }) => {
      this.navigate(path, params);
    });

    // Handle browser navigation
    window.addEventListener('popstate', () => {
      this.handleRoute(window.location.pathname);
    });

    // Intercept links
    document.addEventListener('click', (e) => {
      const link = e.target.closest('a[href^="/"]');
      if (link) {
        e.preventDefault();
        this.navigate(link.getAttribute('href'));
      }
    });

    // Initial route
```

```
    this.handleRoute(window.location.pathname);
  }

  navigate(path, params = {}) {
    window.history.pushState(params, '', path);
    this.handleRoute(path);
  }

  async handleRoute(path) {
    // Find matching route
    for (const route of this.routes) {
      const params = this.matchRoute(route.path, path);

      if (params) {
        // Check route guard
        if (route.guard) {
          const canActivate = await this.runGuard(route.guard, params);
          if (!canActivate) {
            return;
          }
        }

        // Render component
        await this.renderComponent(route.component, params);
        return;
      }
    }

    // 404
    pan.publish('router.not-found', { path });
  }

  matchRoute(pattern, path) {
    if (pattern === '*') return {};
```

```
  const patternParts = pattern.split('/').filter(Boolean);
  const pathParts = path.split('/').filter(Boolean);

  if (patternParts.length !== pathParts.length) return null;

  const params = {};

  for (let i = 0; i < patternParts.length; i++) {
    if (patternParts[i].startsWith(':')) {
      params[patternParts[i].slice(1)] = pathParts[i];
    } else if (patternParts[i] !== pathParts[i]) {
      return null;
    }
  }

  return params;
}

async runGuard(guardName, params) {
  const result = await pan.request(`guard.${guardName}`, params);
  return result !== false;
}

async renderComponent(componentName, params) {
  // Wait for component to be defined
  await customElements.whenDefined(componentName);

  // Create component
  const component = document.createElement(componentName);

  // Pass route params
  Object.entries(params).forEach(([key, value]) => {
    component.setAttribute(key, value);
```

```
    });

    // Clear outlet and add component
    this.outlet.innerHTML = '';
    this.outlet.appendChild(component);

    // Publish route change
    pan.publish('router.changed', { component: componentName, params });
  }
}

customElements.define('pan-router', PanRouter);
customElements.define('pan-route', class extends HTMLElement {});
```

9.3 Route Parameters

Access route parameters in components:

```
class UserPage extends HTMLElement {
  static get observedAttributes() {
    return ['user-id'];
  }

  attributeChangedCallback(name, oldValue, newValue) {
    if (name === 'user-id' && newValue) {
      this.loadUser(newValue);
    }
  }

  async loadUser(id) {
    const response = await fetch(`/api/users/${id}`);
    const user = await response.json();
    this.render(user);
  }

  render(user) {
    this.innerHTML = `
      <h1>${user.name}</h1>
      <p>${user.email}</p>
    `;
  }
}

customElements.define('user-page', UserPage);
```

9.4 Route Guards

Protect routes with authentication checks:

```
// Respond to auth guard
pan.respond('guard.auth', async () => {
  const token = localStorage.getItem('authToken');

  if (!token) {
    // Redirect to login
    pan.publish('router.navigate', { path: '/login' });
    return false;
  }

  // Verify token
  try {
    const response = await fetch('/api/auth/verify', {
      headers: { 'Authorization': `Bearer ${token}` }
    });

    return response.ok;
  } catch {
    return false;
  }
});

// Respond to admin guard
pan.respond('guard.admin', async () => {
  const user = await pan.request('auth.user.get');
  return user?.role === 'admin';
});
```

Usage:

```
<pan-router>
  <pan-route path="/login" component="login-page"></pan-route>
  <pan-route path="/dashboard" component="dashboard-page"
        guard="auth"></pan-route>
  <pan-route path="/admin" component="admin-page" guard="admin"></pan-
        route>
</pan-router>
```

9.5 Nested Routes

Support hierarchical routing:

```
<pan-router>
  <pan-route path="/settings" component="settings-layout">
    <pan-route path="/settings/profile" component="profile-settings"></pan-
        route>
    <pan-route path="/settings/security" component="security-
        settings"></pan-route>
    <pan-route path="/settings/billing" component="billing-
        settings"></pan-route>
  </pan-route>
</pan-router>
```

9.6 Programmatic Navigation

Navigate from JavaScript:

```
// Navigate to a path
pan.publish('router.navigate', { path: '/users/123' });

// Navigate with state
pan.publish('router.navigate', {
  path: '/search',
  state: { query: 'web components' }
});

// Go back
pan.publish('router.back');

// Go forward
pan.publish('router.forward');

// Replace current route (no history entry)
pan.publish('router.replace', { path: '/new-path' });
```

9.7 Query Parameters

Parse and use query parameters:

```
class SearchPage extends HTMLElement {
  connectedCallback() {
    // Parse query params
    const params = new URLSearchParams(window.location.search);
    const query = params.get('q');
    const page = parseInt(params.get('page') || '1');

    this.performSearch(query, page);

    // Listen for query changes
    pan.subscribe('router.changed', () => {
      const params = new URLSearchParams(window.location.search);
      const newQuery = params.get('q');
      const newPage = parseInt(params.get('page') || '1');

      if (newQuery !== query || newPage !== page) {
        this.performSearch(newQuery, newPage);
      }
    });
  }

  performSearch(query, page) {
    // Search implementation
  }
}
```

Update query params:

```
function updateQuery(params) {
  const url = new URL(window.location);

  Object.entries(params).forEach(([key, value]) => {
    url.searchParams.set(key, value);
  });

  pan.publish('router.navigate', { path: url.pathname + url.search });
}

// Usage
updateQuery({ q: 'web components', page: '2' });
```

9.8 Summary

LARC routing provides:

- Client-side navigation without page reloads
- Declarative route configuration
- Route parameters and guards
- Nested routing support
- Browser history integration
- PAN bus integration

9.9 Best Practices

1. **Use declarative routing** - Prefer `<pan-router>` over imperative API
2. **Implement route guards** - Protect sensitive routes
3. **Handle 404s gracefully** - Always include catch-all route
4. **Preserve scroll position** - Restore scroll on back navigation
5. **Use query params for filters** - Makes URLs shareable

9.10 Further Reading

For complete routing reference: - *Building with LARC* Chapter 5: Routing and Navigation - All routing patterns and guards - *Building with LARC* Chapter 17: Core Components - pan-routes API reference - *Building with LARC* Appendix E: Recipes and Patterns - Routing recipes and examples

10 Forms and Validation

Forms are the primary way users input data into web applications. LARC provides patterns for building accessible, validated forms using web standards and the PAN bus.

10.1 Form Components

10.1.1 Basic Form Component

```
class ContactForm extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
  }

  connectedCallback() {
    this.render();
    this.attachEventListeners();
  }

  attachEventListeners() {
    const form = this.shadowRoot.querySelector('form');

    form.addEventListener('submit', async (e) => {
      e.preventDefault();

      if (this.validate()) {
        const data = this.getFormData();
        await this.handleSubmit(data);
      }
    });
  }

  getFormData() {
    const form = this.shadowRoot.querySelector('form');
    const formData = new FormData(form);
    return Object.fromEntries(formData);
  }

  validate() {
    const form = this.shadowRoot.querySelector('form');
    return form.checkValidity();
  }
```

```
async handleSubmit(data) {
  try {
    const response = await fetch('/api/contact', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(data)
    });

    if (response.ok) {
      pan.publish('form.submitted', { form: 'contact', data });
      this.showSuccess();
    } else {
      throw new Error('Submission failed');
    }
  } catch (error) {
    this.showError(error.message);
  }
}

showSuccess() {
  pan.publish('notification.success', { message: 'Form submitted
      successfully!' });
  this.shadowRoot.querySelector('form').reset();
}

showError(message) {
  pan.publish('notification.error', { message });
}

render() {
  this.shadowRoot.innerHTML = `
    <style>
      form { max-width: 500px; }
      .field { margin-bottom: 16px; }
```

```
    label {
      display: block;
      margin-bottom: 4px;
      font-weight: 600;
    }
    input, textarea {
      width: 100%;
      padding: 8px 12px;
      border: 1px solid #cbd5e0;
      border-radius: 4px;
    }
    input:invalid, textarea:invalid {
      border-color: #fc8181;
    }
    button {
      background: #667eea;
      color: white;
      padding: 10px 24px;
      border: none;
      border-radius: 4px;
      cursor: pointer;
    }
  </style>

  <form>
    <div class="field">
      <label for="name">Name *</label>
      <input type="text" id="name" name="name" required
    minlength="2">
    </div>

    <div class="field">
      <label for="email">Email *</label>
      <input type="email" id="email" name="email" required>
    </div>
```

```
        <div class="field">
          <label for="message">Message *</label>
          <textarea id="message" name="message" required minlength="10"
        rows="5"></textarea>
        </div>

        <button type="submit">Send Message</button>
      </form>
    `;
  }
}

customElements.define('contact-form', ContactForm);
```

10.2 Two-Way Data Binding

Sync form inputs with component state:

```
class DataBoundForm extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.state = {
      firstName: '',
      lastName: '',
      email: ''
    };
  }

  connectedCallback() {
    this.render();
    this.bindInputs();
  }

  bindInputs() {
    const inputs = this.shadowRoot.querySelectorAll('input');

    inputs.forEach(input => {
      // Update state when input changes
      input.addEventListener('input', (e) => {
        this.state[e.target.name] = e.target.value;
        pan.publish('form.state.changed', { state: this.state });
      });

      // Update input when state changes
      pan.subscribe('form.state.update', (updates) => {
        if (updates[input.name] !== undefined) {
          input.value = updates[input.name];
          this.state[input.name] = updates[input.name];
        }
      });
    });
```

```
  }

  render() {
    this.shadowRoot.innerHTML = `
      <form>
        <input type="text" name="firstName" value="${this.state.firstName}"
        placeholder="First Name">
        <input type="text" name="lastName" value="${this.state.lastName}"
        placeholder="Last Name">
        <input type="email" name="email" value="${this.state.email}"
        placeholder="Email">
      </form>
      <div class="preview">
        <p>Hello, ${this.state.firstName} ${this.state.lastName}!</p>
        <p>Email: ${this.state.email}</p>
      </div>
    `;
  }
}
```

10.3 Validation Strategies

10.3.1 Native HTML5 Validation

```
<input type="email" required>
<input type="number" min="1" max="100">
<input type="text" pattern="[A-Za-z]{3,}" title="At least 3 letters">
<input type="url" required>
```

10.3.2 Custom Validation

```
class ValidatedInput extends HTMLElement {
  connectedCallback() {
    this.innerHTML = `
      <input type="text" id="input">
      <span class="error"></span>
    `;

    const input = this.querySelector('input');
    const error = this.querySelector('.error');

    input.addEventListener('blur', () => {
      const validationResult = this.customValidate(input.value);

      if (!validationResult.valid) {
        error.textContent = validationResult.message;
        input.classList.add('invalid');
      } else {
        error.textContent = '';
        input.classList.remove('invalid');
      }
    });
  }

  customValidate(value) {
    // Custom validation logic
    if (value.length < 3) {
      return { valid: false, message: 'Must be at least 3 characters' };
    }

    if (!/^[a-zA-Z]+$/.test(value)) {
      return { valid: false, message: 'Only letters allowed' };
    }

    return { valid: true };
```

```
    }
}
```

10.3.3 Async Validation

```
class UsernameInput extends HTMLElement {
  connectedCallback() {
    this.render();

    const input = this.querySelector('input');
    let timeoutId;

    input.addEventListener('input', (e) => {
      clearTimeout(timeoutId);

      timeoutId = setTimeout(async () => {
        await this.checkAvailability(e.target.value);
      }, 500);
    });
  }

  async checkAvailability(username) {
    const status = this.querySelector('.status');

    if (username.length < 3) {
      status.textContent = '';
      return;
    }

    status.textContent = 'Checking...';

    try {
      const response = await fetch(`/api/check-
        username?username=${username}`);
      const { available } = await response.json();

      if (available) {
        status.textContent = '✓ Available';
        status.className = 'status success';
```

```
      } else {
        status.textContent = '✗ Already taken';
        status.className = 'status error';
      }
    } catch (error) {
      status.textContent = 'Could not check availability';
      status.className = 'status error';
    }
  }

  render() {
    this.innerHTML = `
      <label>Username</label>
      <input type="text" placeholder="Choose a username">
      <span class="status"></span>
    `;
  }
}
```

10.4 Error Handling

Display validation errors elegantly:

```
class FormWithErrors extends HTMLElement {
  constructor() {
    super();
    this.errors = {};
  }

  connectedCallback() {
    this.render();

    const form = this.querySelector('form');

    form.addEventListener('submit', (e) => {
      e.preventDefault();

      this.clearErrors();
      const errors = this.validateForm();

      if (Object.keys(errors).length === 0) {
        this.handleSubmit();
      } else {
        this.showErrors(errors);
      }
    });
  }

  validateForm() {
    const errors = {};
    const inputs = this.querySelectorAll('input');

    inputs.forEach(input => {
      if (!input.validity.valid) {
        errors[input.name] = this.getErrorMessage(input);
      }
    });
```

```
    return errors;
  }

  getErrorMessage(input) {
    if (input.validity.valueMissing) {
      return 'This field is required';
    }
    if (input.validity.typeMismatch) {
      return `Please enter a valid ${input.type}`;
    }
    if (input.validity.tooShort) {
      return `Must be at least ${input.minLength} characters`;
    }
    if (input.validity.tooLong) {
      return `Must be no more than ${input.maxLength} characters`;
    }
    if (input.validity.patternMismatch) {
      return input.title || 'Invalid format';
    }

    return 'Invalid input';
  }

  showErrors(errors) {
    Object.entries(errors).forEach(([fieldName, message]) => {
      const field = this.querySelector(`[name="${fieldName}"]`);
      const errorEl = field.parentElement.querySelector('.error');

      if (errorEl) {
        errorEl.textContent = message;
        field.classList.add('invalid');
      }
    });
```

```
  }

  clearErrors() {
    this.querySelectorAll('.error').forEach(el => {
      el.textContent = '';
    });

    this.querySelectorAll('.invalid').forEach(el => {
      el.classList.remove('invalid');
    });
  }

  render() {
    this.innerHTML = `
      <form>
        <div class="field">
          <label>Email</label>
          <input type="email" name="email" required>
          <span class="error"></span>
        </div>

        <div class="field">
          <label>Password</label>
          <input type="password" name="password" required minlength="8">
          <span class="error"></span>
        </div>

        <button type="submit">Submit</button>
      </form>
    `;
  }
}
```

10.5 File Uploads

Handle file uploads with progress tracking:

```
class FileUpload extends HTMLElement {
  connectedCallback() {
    this.render();

    const input = this.querySelector('input[type="file"]');
    const button = this.querySelector('button');

    input.addEventListener('change', (e) => {
      const file = e.target.files[0];
      if (file) {
        this.showPreview(file);
        button.disabled = false;
      }
    });

    button.addEventListener('click', () => {
      const file = input.files[0];
      if (file) {
        this.uploadFile(file);
      }
    });
  }

  showPreview(file) {
    const preview = this.querySelector('.preview');

    if (file.type.startsWith('image/')) {
      const reader = new FileReader();
      reader.onload = (e) => {
        preview.innerHTML = `<img src="${e.target.result}"
         alt="Preview">`;
      };
      reader.readAsDataURL(file);
    } else {
```

```
      preview.innerHTML = `
        <p>${file.name}</p>
        <p>${this.formatFileSize(file.size)}</p>
      `;
    }
  }

  async uploadFile(file) {
    const formData = new FormData();
    formData.append('file', file);

    const xhr = new XMLHttpRequest();

    xhr.upload.addEventListener('progress', (e) => {
      const percent = (e.loaded / e.total) * 100;
      this.updateProgress(percent);
    });

    xhr.addEventListener('load', () => {
      if (xhr.status === 200) {
        pan.publish('file.uploaded', {
          filename: file.name,
          response: JSON.parse(xhr.response)
        });
        this.showSuccess();
      } else {
        this.showError('Upload failed');
      }
    });

    xhr.addEventListener('error', () => {
      this.showError('Upload failed');
    });
```

```
    xhr.open('POST', '/api/upload');
    xhr.send(formData);
  }

  updateProgress(percent) {
    const progress = this.querySelector('.progress-bar');
    progress.style.width = `${percent}%`;
    progress.textContent = `${Math.round(percent)}%`;
  }

  formatFileSize(bytes) {
    if (bytes < 1024) return bytes + ' B';
    if (bytes < 1024 * 1024) return (bytes / 1024).toFixed(1) + ' KB';
    return (bytes / (1024 * 1024)).toFixed(1) + ' MB';
  }

  showSuccess() {
    this.querySelector('.status').innerHTML = '✓ Uploaded successfully';
  }

  showError(message) {
    this.querySelector('.status').innerHTML = `✗ ${message}`;
  }

  render() {
    this.innerHTML = `
      <div class="upload-container">
        <input type="file" accept="image/*">
        <div class="preview"></div>
        <div class="progress">
          <div class="progress-bar"></div>
        </div>
        <button disabled>Upload</button>
        <div class="status"></div>
```

```
      </div>
    `;
  }
}

customElements.define('file-upload', FileUpload);
```

10.6 Form Submission

Handle form submission with loading states and error recovery:

```
class SmartForm extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.submitting = false;
  }

  connectedCallback() {
    this.render();

    this.shadowRoot.querySelector('form').addEventListener('submit',
        async (e) => {
      e.preventDefault();

      if (this.submitting) return;

      this.submitting = true;
      this.disableForm();

      try {
        const data = this.getFormData();
        await this.submitForm(data);
        this.handleSuccess();
      } catch (error) {
        this.handleError(error);
      } finally {
        this.submitting = false;
        this.enableForm();
      }
    });
  }

  getFormData() {
    const form = this.shadowRoot.querySelector('form');
```

```
    const formData = new FormData(form);
    return Object.fromEntries(formData);
  }

  async submitForm(data) {
    const response = await fetch('/api/submit', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(data)
    });

    if (!response.ok) {
      const error = await response.json();
      throw new Error(error.message || 'Submission failed');
    }

    return response.json();
  }

  disableForm() {
    const inputs = this.shadowRoot.querySelectorAll('input, button,
        textarea');
    inputs.forEach(el => el.disabled = true);

    this.shadowRoot.querySelector('.loading').style.display = 'block';
  }

  enableForm() {
    const inputs = this.shadowRoot.querySelectorAll('input, button,
        textarea');
    inputs.forEach(el => el.disabled = false);

    this.shadowRoot.querySelector('.loading').style.display = 'none';
  }
```

```
  handleSuccess() {
    pan.publish('notification.success', { message: 'Form submitted
        successfully!' });
    this.shadowRoot.querySelector('form').reset();
  }

  handleError(error) {
    pan.publish('notification.error', { message: error.message });
  }

  render() {
    this.shadowRoot.innerHTML = `
      <style>
        .loading {
          display: none;
          text-align: center;
          padding: 16px;
        }
      </style>

      <form>
        <!-- Form fields -->
        <button type="submit">Submit</button>
      </form>

      <div class="loading">
        <div class="spinner"></div>
        <p>Submitting...</p>
      </div>
    `;
  }
}

customElements.define('smart-form', SmartForm);
```

10.7 Summary

This chapter covered:

- Building accessible form components
- Two-way data binding patterns
- Validation strategies (native and custom)
- Error handling and display
- File upload with progress tracking
- Form submission with loading states

10.8 Best Practices

1. **Use native validation first** - HTML5 provides powerful built-in validation
2. **Provide clear error messages** - Tell users exactly what's wrong
3. **Validate on blur** - Don't show errors while user is typing
4. **Disable during submission** - Prevent double-submission
5. **Show progress for uploads** - Users want to see progress
6. **Handle errors gracefully** - Network can fail, handle it well

---

10.9 Further Reading

For complete forms and validation reference: - *Building with LARC* Chapter 6: Forms and User Input - All form patterns and validation strategies - *Building with LARC* Chapter 19: UI Components - pan-files and pan-markdown-editor reference - *Building with LARC* Appendix E: Recipes and Patterns - Form validation recipes

11 Data Fetching and APIs

Every meaningful web application needs to communicate with servers. Whether you're loading user profiles, submitting forms, or streaming real-time updates, data fetching is the bridge between your frontend and the outside world. LARC embraces the browser's native fetch API while providing patterns that make common tasks simple and complex scenarios manageable.

11.1 The Fetch API: Your Foundation

The Fetch API is built into every modern browser, and it's genuinely excellent. Unlike the XMLHttpRequest it replaced, fetch returns Promises, works naturally with async/await, and provides a clean interface for HTTP operations.

Here's the simplest possible fetch:

```
const response = await fetch('/api/users');
const users = await response.json();
```

Two lines. No libraries. No configuration. This is the foundation everything else builds upon.

But real applications need more: error handling, loading states, retries, caching. Let's build these capabilities systematically.

11.2 Building an API Client

Rather than scattering fetch calls throughout your application, centralize them in an API client. This gives you one place to handle authentication, errors, and common patterns:

```
// api-client.js
class ApiClient {
  constructor(baseUrl = '/api') {
    this.baseUrl = baseUrl;
  }

  async fetch(endpoint, options = {}) {
    const url = `${this.baseUrl}${endpoint}`;

    const config = {
      headers: {
        'Content-Type': 'application/json',
        ...options.headers
      },
      ...options
    };

    // Add auth token if available
    const token = localStorage.getItem('authToken');
    if (token) {
      config.headers['Authorization'] = `Bearer ${token}`;
    }

    try {
      const response = await fetch(url, config);

      if (!response.ok) {
        const error = await response.json().catch(() => ({}));
        throw new ApiError(response.status, error.message || 'Request
        failed');
      }

      return response.json();
    } catch (error) {
```

```
      if (error instanceof ApiError) throw error;
      throw new ApiError(0, 'Network error');
    }
  }

  get(endpoint) {
    return this.fetch(endpoint);
  }

  post(endpoint, data) {
    return this.fetch(endpoint, {
      method: 'POST',
      body: JSON.stringify(data)
    });
  }

  put(endpoint, data) {
    return this.fetch(endpoint, {
      method: 'PUT',
      body: JSON.stringify(data)
    });
  }

  delete(endpoint) {
    return this.fetch(endpoint, { method: 'DELETE' });
  }
}

class ApiError extends Error {
  constructor(status, message) {
    super(message);
    this.status = status;
  }
}
```

```
export const api = new ApiClient();
```

Now every component in your application can import this client and make requests with consistent error handling and authentication.

11.3 Integrating with the PAN Bus

Here's where LARC shines. Instead of each component managing its own loading states and error handling, broadcast API events on the PAN bus:

```
// api-client.js (enhanced)
import { pan } from '@aspect/pan-client';

class ApiClient {
  async fetch(endpoint, options = {}) {
    const requestId = crypto.randomUUID();

    // Announce request start
    pan.publish('api.request.start', { requestId, endpoint });

    try {
      const response = await fetch(/* ... */);
      const data = await response.json();

      // Announce success
      pan.publish('api.request.success', { requestId, endpoint, data });
      return data;

    } catch (error) {
      // Announce failure
      pan.publish('api.request.error', { requestId, endpoint, error });
      throw error;
    }
  }
}
```

Now any component can listen for API events. A loading indicator component might subscribe to `api.request.start` and `api.request.success`. An error toast might listen only for `api.request.error`. Components become loosely coupled—they don't need to know about each other, just the messages they care about.

11.4 Caching Strategies

Network requests are slow and expensive. Smart caching makes your application feel instant while reducing server load.

11.4.1 Cache-First Strategy

For data that changes infrequently, serve from cache immediately and update in the background:

```
class CachedApiClient extends ApiClient {
  constructor() {
    super();
    this.cache = new Map();
  }

  async getCached(endpoint, maxAge = 60000) {
    const cached = this.cache.get(endpoint);

    if (cached && Date.now() - cached.timestamp < maxAge) {
      // Return cached data immediately
      return cached.data;
    }

    // Fetch fresh data
    const data = await this.get(endpoint);
    this.cache.set(endpoint, { data, timestamp: Date.now() });
    return data;
  }

  async getStaleWhileRevalidate(endpoint, maxAge = 60000) {
    const cached = this.cache.get(endpoint);

    // Return stale data immediately if available
    if (cached) {
      // Revalidate in background
      this.get(endpoint).then(data => {
        this.cache.set(endpoint, { data, timestamp: Date.now() });
        pan.publish(`cache.updated.${endpoint}`, { data });
      });

      return cached.data;
    }
```

```
    // No cache, must wait for network
    const data = await this.get(endpoint);
    this.cache.set(endpoint, { data, timestamp: Date.now() });
    return data;
  }
}
```

11.4.2 Network-First Strategy

For data that must be current, try the network first and fall back to cache:

```
async getNetworkFirst(endpoint, maxAge = 300000) {
  try {
    const data = await this.get(endpoint);
    this.cache.set(endpoint, { data, timestamp: Date.now() });
    return data;
  } catch (error) {
    const cached = this.cache.get(endpoint);
    if (cached && Date.now() - cached.timestamp < maxAge) {
      console.warn('Using cached data due to network error');
      return cached.data;
    }
    throw error;
  }
}
```

11.5 WebSocket Communication

When you need real-time bidirectional communication, WebSockets provide a persistent connection between browser and server:

```
// websocket-client.js
class WebSocketClient {
  constructor(url) {
    this.url = url;
    this.socket = null;
    this.reconnectAttempts = 0;
    this.maxReconnectAttempts = 5;
  }

  connect() {
    this.socket = new WebSocket(this.url);

    this.socket.onopen = () => {
      this.reconnectAttempts = 0;
      pan.publish('ws.connected');
    };

    this.socket.onmessage = (event) => {
      const message = JSON.parse(event.data);
      // Broadcast message on PAN bus
      pan.publish(`ws.message.${message.type}`, message.payload);
    };

    this.socket.onclose = () => {
      pan.publish('ws.disconnected');
      this.attemptReconnect();
    };

    this.socket.onerror = (error) => {
      pan.publish('ws.error', { error });
    };
  }

  send(type, payload) {
```

```
    if (this.socket?.readyState === WebSocket.OPEN) {
      this.socket.send(JSON.stringify({ type, payload }));
    }
  }

  attemptReconnect() {
    if (this.reconnectAttempts < this.maxReconnectAttempts) {
      this.reconnectAttempts++;
      const delay = Math.min(1000 * Math.pow(2, this.reconnectAttempts),
        30000);
      setTimeout(() => this.connect(), delay);
    }
  }
}
```

The beauty of this approach: your components don't know or care that data comes from a WebSocket. They just subscribe to PAN bus topics like `ws.message.user-joined` or `ws.message.chat-message`.

11.6 Server-Sent Events

When you only need server-to-client updates (no bidirectional communication), Server-Sent Events (SSE) are simpler and more reliable than WebSockets:

```
// sse-client.js
class SSEClient {
  constructor(url) {
    this.url = url;
    this.eventSource = null;
  }

  connect() {
    this.eventSource = new EventSource(this.url);

    this.eventSource.onmessage = (event) => {
      const data = JSON.parse(event.data);
      pan.publish('sse.message', data);
    };

    this.eventSource.addEventListener('notification', (event) => {
      const data = JSON.parse(event.data);
      pan.publish('sse.notification', data);
    });

    this.eventSource.onerror = () => {
      pan.publish('sse.error');
      // EventSource automatically reconnects
    };
  }

  disconnect() {
    this.eventSource?.close();
  }
}
```

SSE reconnects automatically, handles authentication through cookies, and works through proxies that might block WebSockets. For many real-time use cases, it's the better choice.

11.7 Retry Logic with Exponential Backoff

Network requests fail. Good applications handle failure gracefully:

```
async function fetchWithRetry(url, options = {}, maxRetries = 3) {
  let lastError;

  for (let attempt = 0; attempt < maxRetries; attempt++) {
    try {
      const response = await fetch(url, options);
      if (response.ok) return response;

      // Don't retry client errors (4xx)
      if (response.status >= 400 && response.status < 500) {
        throw new Error(`Client error: ${response.status}`);
      }

      throw new Error(`Server error: ${response.status}`);
    } catch (error) {
      lastError = error;

      if (attempt < maxRetries - 1) {
        // Exponential backoff: 1s, 2s, 4s...
        const delay = Math.pow(2, attempt) * 1000;
        await new Promise(resolve => setTimeout(resolve, delay));
      }
    }
  }

  throw lastError;
}
```

11.8 Putting It All Together

Here’s a component that fetches user data with loading states, caching, and error handling—all integrated with the PAN bus:

```
class UserList extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.users = [];
    this.loading = true;
    this.error = null;
  }

  async connectedCallback() {
    // Subscribe to cache updates
    pan.subscribe('cache.updated./api/users', ({ data }) => {
      this.users = data;
      this.render();
    });

    try {
      this.users = await api.getStaleWhileRevalidate('/users');
      this.loading = false;
    } catch (error) {
      this.error = error.message;
      this.loading = false;
    }

    this.render();
  }

  render() {
    this.shadowRoot.innerHTML = `
      <style>
        :host { display: block; }
        .loading { color: #666; }
        .error { color: red; }
        .user { padding: 8px; border-bottom: 1px solid #eee; }
```

```
      </style>

      ${this.loading ? '<p class="loading">Loading users...</p>' : ''}
      ${this.error ? `<p class="error">Error: ${this.error}</p>` : ''}

      <div class="users">
        ${this.users.map(user => `
          <div class="user">${user.name} - ${user.email}</div>
        `).join('')}
      </div>
    `;
  }
}

customElements.define('user-list', UserList);
```

11.9 GraphQL Integration

While REST APIs are common, GraphQL offers precise data fetching—request exactly what you need, nothing more. Here's how to integrate GraphQL with LARC:

```
// graphql-client.js
class GraphQLClient {
  constructor(endpoint) {
    this.endpoint = endpoint;
  }

  async query(query, variables = {}) {
    const response = await fetch(this.endpoint, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${localStorage.getItem('authToken')}`
      },
      body: JSON.stringify({ query, variables })
    });

    const { data, errors } = await response.json();

    if (errors) {
      throw new GraphQLError(errors);
    }

    return data;
  }

  async mutation(mutation, variables = {}) {
    return this.query(mutation, variables);
  }
}

class GraphQLError extends Error {
  constructor(errors) {
    super(errors.map(e => e.message).join(', '));
    this.errors = errors;
```

```
  }
}

export const graphql = new GraphQLClient('/graphql');
```

11.9.1 Using GraphQL in Components

```
class UserProfile extends HTMLElement {
  async connectedCallback() {
    const userId = this.getAttribute('user-id');

    const query = `
      query GetUser($id: ID!) {
        user(id: $id) {
          id
          name
          email
          avatar
          posts {
            id
            title
            publishedAt
          }
        }
      }
    `;

    try {
      const { user } = await graphql.query(query, { id: userId });
      this.renderUser(user);
    } catch (error) {
      this.renderError(error);
    }
  }

  renderUser(user) {
    this.innerHTML = `
      <div class="profile">
        <img src="${user.avatar}" alt="${user.name}">
        <h2>${user.name}</h2>
        <p>${user.email}</p>
```

```
          <h3>Recent Posts</h3>
          <ul>
            ${user.posts.map(post => `
              <li>${post.title} (${new
          Date(post.publishedAt).toLocaleDateString()})</li>
            `).join('')}
          </ul>
        </div>
      `;
    }
  }
```

11.10 Real-World Example: Building a Dashboard

Let’s build a complete dashboard that fetches data from multiple API endpoints, handles loading states, and updates in real-time.

11.10.1 The Dashboard Component

```
// components/dashboard.js
import { api } from '../lib/api-client.js';
import { pan } from '@larcjs/core';

class Dashboard extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.state = {
      stats: null,
      activity: [],
      loading: true,
      error: null
    };
  }

  async connectedCallback() {
    // Subscribe to real-time updates
    pan.subscribe('ws.message.stats-updated', ({ stats }) => {
      this.state.stats = stats;
      this.render();
    });

    pan.subscribe('ws.message.activity-added', ({ activity }) => {
      this.state.activity.unshift(activity);
      this.render();
    });

    await this.loadData();
    this.render();
  }

  async loadData() {
    try {
```

```
      // Load multiple endpoints in parallel
      const [stats, activity] = await Promise.all([
        api.getCached('/stats', 30000),
        api.get('/activity?limit=10')
      ]);

      this.state = {
        stats,
        activity,
        loading: false,
        error: null
      };
    } catch (error) {
      this.state = {
        ...this.state,
        loading: false,
        error: error.message
      };
    }
  }

  async refreshData() {
    this.state.loading = true;
    this.render();
    await this.loadData();
    this.render();
  }

  render() {
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: block;
          padding: 20px;
```

```
        }

        .header {
          display: flex;
          justify-content: space-between;
          align-items: center;
          margin-bottom: 20px;
        }

        .stats {
          display: grid;
          grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
          gap: 20px;
          margin-bottom: 30px;
        }

        .stat-card {
          background: white;
          padding: 20px;
          border-radius: 8px;
          box-shadow: 0 2px 4px rgba(0,0,0,0.1);
        }

        .stat-value {
          font-size: 32px;
          font-weight: bold;
          color: #667eea;
        }

        .stat-label {
          color: #666;
          margin-top: 5px;
        }
```

```
.activity {
  background: white;
  border-radius: 8px;
  box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

.activity-header {
  padding: 20px;
  border-bottom: 1px solid #eee;
  font-weight: 600;
}

.activity-item {
  padding: 15px 20px;
  border-bottom: 1px solid #eee;
}

.activity-item:last-child {
  border-bottom: none;
}

.loading {
  text-align: center;
  padding: 40px;
  color: #666;
}

.error {
  background: #fee;
  color: #c00;
  padding: 15px;
  border-radius: 4px;
}
```

```
        button {
          background: #667eea;
          color: white;
          border: none;
          padding: 10px 20px;
          border-radius: 4px;
          cursor: pointer;
        }

        button:hover {
          background: #5568d3;
        }
      </style>

      <div class="header">
        <h1>Dashboard</h1>
        <button @click="${() => this.refreshData()}">Refresh</button>
      </div>

      ${this.state.loading && !this.state.stats ? `
        <div class="loading">Loading dashboard...</div>
      ` : ''}

      ${this.state.error ? `
        <div class="error">Error: ${this.state.error}</div>
      ` : ''}

      ${this.state.stats ? `
        <div class="stats">
          <div class="stat-card">
            <div class="stat-value">${this.state.stats.totalUsers}</div>
            <div class="stat-label">Total Users</div>
          </div>
          <div class="stat-card">
```

```
            <div class="stat-value">${this.state.stats.activeUsers}</div>
            <div class="stat-label">Active Users</div>
          </div>
          <div class="stat-card">
            <div class="stat-value">${this.state.stats.revenue}</div>
            <div class="stat-label">Revenue</div>
          </div>
          <div class="stat-card">
            <div class="stat-
       value">${this.state.stats.conversionRate}%</div>
            <div class="stat-label">Conversion Rate</div>
          </div>
        </div>
      ` : ''}

      ${this.state.activity.length > 0 ? `
        <div class="activity">
          <div class="activity-header">Recent Activity</div>
          ${this.state.activity.map(item => `
            <div class="activity-item">
              <strong>${item.user}</strong> ${item.action}
              <span style="color: #999; font-size: 14px;">
                ${this.formatTime(item.timestamp)}
              </span>
            </div>
          `).join('')}
        </div>
      ` : ''}
    `;

    // Attach event listeners
    const refreshBtn = this.shadowRoot.querySelector('button');
    if (refreshBtn) {
      refreshBtn.addEventListener('click', () => this.refreshData());
    }
```

```
  }

  formatTime(timestamp) {
    const date = new Date(timestamp);
    const now = new Date();
    const diff = now - date;

    if (diff < 60000) return 'just now';
    if (diff < 3600000) return `${Math.floor(diff / 60000)}m ago`;
    if (diff < 86400000) return `${Math.floor(diff / 3600000)}h ago`;
    return date.toLocaleDateString();
  }
}

customElements.define('app-dashboard', Dashboard);
```

This dashboard demonstrates:

- **Parallel data loading** with `Promise.all`
- **Caching** for stats that don't change often
- **Real-time updates** via PAN bus
- **Loading and error states**
- **Manual refresh** capability
- **Relative time formatting**

11.11 Error Handling Patterns

11.11.1 Circuit Breaker Pattern

Prevent cascading failures by stopping requests to failing services:

```
class CircuitBreaker {
  constructor(threshold = 5, timeout = 60000) {
    this.failureCount = 0;
    this.threshold = threshold;
    this.timeout = timeout;
    this.state = 'CLOSED'; // CLOSED, OPEN, HALF_OPEN
    this.nextAttempt = Date.now();
  }

  async execute(fn) {
    if (this.state === 'OPEN') {
      if (Date.now() < this.nextAttempt) {
        throw new Error('Circuit breaker is OPEN');
      }
      this.state = 'HALF_OPEN';
    }

    try {
      const result = await fn();
      this.onSuccess();
      return result;
    } catch (error) {
      this.onFailure();
      throw error;
    }
  }

  onSuccess() {
    this.failureCount = 0;
    this.state = 'CLOSED';
  }

  onFailure() {
    this.failureCount++;
```

```
    if (this.failureCount >= this.threshold) {
      this.state = 'OPEN';
      this.nextAttempt = Date.now() + this.timeout;
      pan.publish('circuit-breaker.opened', {
        breaker: this
      });
    }
  }
}

// Usage
const userApiBreaker = new CircuitBreaker();

async function fetchUsers() {
  return userApiBreaker.execute(() => api.get('/users'));
}
```

11.11.2 Fallback Strategies

Provide graceful degradation when APIs fail:

```
class FallbackApiClient extends ApiClient {
  async getWithFallback(endpoint, fallbackData) {
    try {
      return await this.get(endpoint);
    } catch (error) {
      console.warn(`API call failed, using fallback for ${endpoint}`);
      pan.publish('api.fallback', { endpoint, error });
      return fallbackData;
    }
  }

  async getWithCacheFallback(endpoint) {
    try {
      const data = await this.get(endpoint);
      localStorage.setItem(`fallback:${endpoint}`, JSON.stringify(data));
      return data;
    } catch (error) {
      const cached = localStorage.getItem(`fallback:${endpoint}`);
      if (cached) {
        return JSON.parse(cached);
      }
      throw error;
    }
  }
}
```

11.12 Optimistic Updates

Update UI immediately before the server responds:

```
class TodoList extends HTMLElement {
  async addTodo(text) {
    const optimisticTodo = {
      id: `temp-${Date.now()}`,
      text,
      completed: false,
      pending: true
    };

    // Add to UI immediately
    this.todos.push(optimisticTodo);
    this.render();

    try {
      // Send to server
      const savedTodo = await api.post('/todos', { text });

      // Replace optimistic todo with server response
      const index = this.todos.findIndex(t => t.id === optimisticTodo.id);
      this.todos[index] = savedTodo;
      this.render();

    } catch (error) {
      // Revert on failure
      this.todos = this.todos.filter(t => t.id !== optimisticTodo.id);
      this.render();

      pan.publish('notification.error', {
        message: 'Failed to add todo'
      });
    }
  }
}
```

11.13 Infinite Scroll / Pagination

Load more data as the user scrolls:

```
class InfiniteList extends HTMLElement {
  constructor() {
    super();
    this.items = [];
    this.page = 1;
    this.loading = false;
    this.hasMore = true;
  }

  connectedCallback() {
    this.render();
    this.loadMore();

    // Intersection Observer for infinite scroll
    const sentinel = this.querySelector('.sentinel');
    const observer = new IntersectionObserver(entries => {
      if (entries[0].isIntersecting && !this.loading && this.hasMore) {
        this.loadMore();
      }
    });
    observer.observe(sentinel);
  }

  async loadMore() {
    if (this.loading || !this.hasMore) return;

    this.loading = true;
    this.render();

    try {
      const data = await api.get(`/items?page=${this.page}&limit=20`);

      this.items.push(...data.items);
      this.hasMore = data.hasMore;
```

```
      this.page++;

    } catch (error) {
      console.error('Failed to load more items:', error);
    } finally {
      this.loading = false;
      this.render();
    }
  }

  render() {
    this.innerHTML = `
      <div class="items">
        ${this.items.map(item => `
          <div class="item">${item.title}</div>
        `).join('')}
      </div>

      ${this.loading ? '<div class="loading">Loading...</div>' : ''}
      ${this.hasMore ? '<div class="sentinel"></div>' : ''}
      ${!this.hasMore ? '<div class="end">No more items</div>' : ''}
    `;
  }
}
```

11.14 Troubleshooting

11.14.1 Problem: CORS Errors

Symptom: `Access to fetch at 'https://api.example.com' from origin 'http://localhost:3000' has been blocked by CORS policy`

Solution: Configure your server to send CORS headers:

```
// Express.js example
app.use((req, res, next) => {
  res.header('Access-Control-Allow-Origin', 'http://localhost:3000');
  res.header('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE');
  res.header('Access-Control-Allow-Headers', 'Content-Type,
        Authorization');
  res.header('Access-Control-Allow-Credentials', 'true');
  next();
});
```

Or use a proxy in development:

```
// vite.config.js
export default {
  server: {
    proxy: {
      '/api': {
        target: 'https://api.example.com',
        changeOrigin: true,
        rewrite: (path) => path.replace(/^\/api/, '')
      }
    }
  }
};
```

11.14.2 Problem: Requests Timing Out

Symptom: Fetch hangs indefinitely or takes too long

Solution: Add timeout to fetch requests:

```
async function fetchWithTimeout(url, options = {}, timeout = 5000) {
  const controller = new AbortController();
  const id = setTimeout(() => controller.abort(), timeout);

  try {
    const response = await fetch(url, {
      ...options,
      signal: controller.signal
    });
    clearTimeout(id);
    return response;
  } catch (error) {
    clearTimeout(id);
    if (error.name === 'AbortError') {
      throw new Error('Request timeout');
    }
    throw error;
  }
}
```

11.14.3 Problem: Memory Leaks from Uncancelled Requests

Symptom: Component removed but requests still updating state

Solution: Cancel requests when component disconnects:

```
class UserList extends HTMLElement {
  constructor() {
    super();
    this.abortController = new AbortController();
  }

  async connectedCallback() {
    try {
      const response = await fetch('/api/users', {
        signal: this.abortController.signal
      });
      const users = await response.json();
      this.render(users);
    } catch (error) {
      if (error.name !== 'AbortError') {
        console.error(error);
      }
    }
  }

  disconnectedCallback() {
    this.abortController.abort();
  }
}
```

11.14.4 Problem: Stale Data After Navigation

Symptom: Old data briefly appears before loading new data

Solution: Clear cache or reset state on navigation:

```
pan.subscribe('navigation.changed', () => {
  // Clear API cache
  api.clearCache();

  // Or invalidate specific endpoints
  api.invalidate('/users');
});
```

11.15 Best Practices

1. **Centralize API logic** - Use a single API client, not scattered fetch calls
2. **Handle loading states** - Always show feedback during async operations
3. **Implement caching** - Reduce server load and improve perceived performance
4. **Use optimistic updates** - Make UI feel instant for common operations
5. **Fail gracefully** - Provide fallbacks, don't just show error messages
6. **Cancel requests** - Clean up when components unmount
7. **Use the PAN bus** - Decouple components from API implementation details
8. **Implement retries** - Networks are unreliable, retry with backoff
9. **Monitor API health** - Use circuit breakers for failing services
10. **Test with mocks** - Don't depend on live APIs for tests

11.16 Exercises

11.16.1 Exercise 1: Weather Dashboard

Build a weather dashboard that:

- Fetches current weather for a city
- Displays 5-day forecast
- Caches results for 30 minutes
- Updates automatically when city changes

- Shows loading spinner during fetch
- Handles API errors gracefully

Bonus: Add geolocation to auto-detect user's city.

11.16.2 Exercise 2: Infinite Scroll Blog

Create a blog list with infinite scroll that:

- Loads 10 posts initially
- Loads 10 more as user scrolls down
- Shows loading indicator at bottom
- Handles "no more posts" state
- Implements pull-to-refresh on mobile

Bonus: Add search that filters posts while maintaining infinite scroll.

11.16.3 Exercise 3: Real-Time Chat

Build a simple chat application that:

- Uses WebSockets for real-time messages
- Shows typing indicators
- Displays online/offline status
- Reconnects automatically when connection drops
- Falls back to polling if WebSockets unavailable

Bonus: Add message read receipts and "User is typing…" indicators.

11.16.4 Exercise 4: Optimistic Todo List

Create a todo list with optimistic updates:

- Add todos instantly (before server confirms)
- Revert if server rejects
- Show pending state with different styling

- Handle offline mode (queue operations)
- Sync when connection restored

Bonus: Use IndexedDB for offline storage and sync queue.

11.17 Summary

Data fetching is the bridge between your frontend and the world. LARC embraces native browser APIs (fetch, WebSocket, EventSource) while providing patterns that make common tasks simple:

- **Use fetch** for HTTP requests, enhance with retries and timeouts
- **Build an API client** to centralize authentication and error handling
- **Integrate with PAN bus** to decouple components from API details
- **Implement caching** to improve performance and reduce server load
- **Handle errors gracefully** with circuit breakers and fallbacks
- **Use optimistic updates** to make UI feel instant
- **Choose the right tool**: REST for standard APIs, GraphQL for flexible queries, WebSockets for bidirectional real-time, SSE for server push

The web platform provides excellent data fetching capabilities. LARC helps you use them effectively.

11.18 Further Reading

For complete API integration reference: - *Building with LARC* Chapter 7: Data Fetching and APIs - All patterns and strategies - *Building with LARC* Chapter 20: Integration Components - pan-data-connector, pan-websocket, pan-sse API reference - *Building with LARC* Appendix E: Recipes and Patterns - API integration recipes

12 Authentication and Security

Authentication is the bouncer at your application's door. Get it wrong, and either legitimate users can't get in, or everyone can. Security isn't a feature you add later—it's a mindset that shapes every decision from the start.

12.1 Understanding Authentication vs Authorization

These terms often get conflated, but they're distinct:

Authentication answers: "Who are you?" It's verifying identity—matching a username and password, validating a token, confirming you are who you claim to be.

Authorization answers: "What can you do?" Once we know who you are, authorization determines your permissions—can you view this page, edit this record, delete this user?

LARC applications typically handle authentication with JWT tokens and authorization with role-based or permission-based access control.

12.2 JWT Token Management

JSON Web Tokens (JWTs) are the standard for stateless authentication. A JWT contains encoded claims about the user, signed by the server:

```
// auth-service.js
class AuthService {
  constructor() {
    this.tokenKey = 'auth_token';
    this.refreshKey = 'refresh_token';
  }

  async login(email, password) {
    const response = await fetch('/api/auth/login', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ email, password })
    });

    if (!response.ok) {
      const error = await response.json();
      throw new Error(error.message || 'Login failed');
    }

    const { accessToken, refreshToken, user } = await response.json();

    this.setTokens(accessToken, refreshToken);
    pan.publish('auth.login', { user }, { retained: true });

    return user;
  }

  setTokens(accessToken, refreshToken) {
    localStorage.setItem(this.tokenKey, accessToken);
    if (refreshToken) {
      localStorage.setItem(this.refreshKey, refreshToken);
    }
  }
```

```
  getToken() {
    return localStorage.getItem(this.tokenKey);
  }

  isAuthenticated() {
    const token = this.getToken();
    if (!token) return false;

    try {
      const payload = this.decodeToken(token);
      return payload.exp * 1000 > Date.now();
    } catch {
      return false;
    }
  }

  decodeToken(token) {
    const base64Url = token.split('.')[1];
    const base64 = base64Url.replace(/-/g, '+').replace(/_/g, '/');
    return JSON.parse(atob(base64));
  }

  async refresh() {
    const refreshToken = localStorage.getItem(this.refreshKey);
    if (!refreshToken) throw new Error('No refresh token');

    const response = await fetch('/api/auth/refresh', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ refreshToken })
    });

    if (!response.ok) {
      this.logout();
```

```
      throw new Error('Token refresh failed');
    }

    const { accessToken } = await response.json();
    localStorage.setItem(this.tokenKey, accessToken);
    return accessToken;
  }

  logout() {
    localStorage.removeItem(this.tokenKey);
    localStorage.removeItem(this.refreshKey);
    pan.publish('auth.logout', {}, { retained: true });
  }

  getCurrentUser() {
    const token = this.getToken();
    if (!token) return null;

    try {
      return this.decodeToken(token);
    } catch {
      return null;
    }
  }
}

export const auth = new AuthService();
```

12.3 Automatic Token Refresh

Tokens expire. Good applications refresh them transparently:

```
// api-client.js with token refresh
class AuthenticatedApiClient {
  async fetch(endpoint, options = {}) {
    // First attempt
    try {
      return await this.doFetch(endpoint, options);
    } catch (error) {
      // If 401, try refreshing token
      if (error.status === 401) {
        try {
          await auth.refresh();
          // Retry with new token
          return await this.doFetch(endpoint, options);
        } catch (refreshError) {
          // Refresh failed, user must log in again
          auth.logout();
          throw error;
        }
      }
      throw error;
    }
  }

  async doFetch(endpoint, options) {
    const token = auth.getToken();
    const headers = {
      'Content-Type': 'application/json',
      ...options.headers
    };

    if (token) {
      headers['Authorization'] = `Bearer ${token}`;
    }
```

```
    const response = await fetch(`/api${endpoint}`, { ...options, headers
        });

    if (!response.ok) {
      throw { status: response.status, message: await response.text() };
    }

    return response.json();
  }
}
```

12.4 Protected Routes

Some pages should only be accessible to authenticated users. Here's a route guard pattern:

```
// route-guard.js
class RouteGuard extends HTMLElement {
  connectedCallback() {
    this.checkAuth();
    pan.subscribe('auth.logout', () => this.checkAuth());
  }

  checkAuth() {
    if (!auth.isAuthenticated()) {
      // Store intended destination
      sessionStorage.setItem('returnUrl', window.location.pathname);
      // Redirect to login
      pan.publish('router.navigate', { path: '/login' });
    }
  }
}

customElements.define('route-guard', RouteGuard);
```

Use it to wrap protected content:

```
<route-guard>
  <dashboard-page></dashboard-page>
</route-guard>
```

12.5 Role-Based Access Control

Different users have different permissions. A simple RBAC implementation:

```
// rbac.js
class RBAC {
  constructor() {
    this.permissions = {
      admin: ['read', 'write', 'delete', 'manage-users'],
      editor: ['read', 'write'],
      viewer: ['read']
    };
  }

  can(user, action) {
    if (!user?.role) return false;
    const allowed = this.permissions[user.role] || [];
    return allowed.includes(action);
  }
}

export const rbac = new RBAC();
```

Use it in components:

```
class AdminPanel extends HTMLElement {
  connectedCallback() {
    const user = auth.getCurrentUser();

    if (!rbac.can(user, 'manage-users')) {
      this.innerHTML = '<p>Access denied</p>';
      return;
    }

    this.render();
  }
}
```

12.6 Security Best Practices

12.6.1 Sanitize User Input

Never trust user input. Always sanitize before rendering:

```
function escapeHtml(text) {
  const div = document.createElement('div');
  div.textContent = text;
  return div.innerHTML;
}

// Safe rendering
this.innerHTML = `<p>${escapeHtml(userInput)}</p>`;
```

12.6.2 Use HTTPS

Always serve your application over HTTPS. This protects tokens in transit and enables secure cookies.

12.6.3 Secure Token Storage

LocalStorage is convenient but accessible to JavaScript. For high-security applications, consider httpOnly cookies:

```
// Server sets cookie
res.cookie('token', jwt, {
  httpOnly: true,
  secure: true,
  sameSite: 'strict'
});
```

12.6.4 Content Security Policy

Set CSP headers to prevent XSS attacks:

```
<meta http-equiv="Content-Security-Policy"
      content="default-src 'self'; script-src 'self'">
```

12.7 Complete Login/Signup Flow

Let’s build a complete authentication system with login and signup components that work together:

12.7.1 Login Component

```
// components/login-form.js
import { auth } from '../services/auth.js';
import { pan } from '@larcjs/core';

class LoginForm extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.state = {
      email: '',
      password: '',
      loading: false,
      error: null
    };
  }

  connectedCallback() {
    this.render();
  }

  async handleSubmit(e) {
    e.preventDefault();

    this.state.loading = true;
    this.state.error = null;
    this.render();

    try {
      const user = await auth.login(this.state.email,
        this.state.password);

      // Redirect to intended destination or dashboard
      const returnUrl = sessionStorage.getItem('returnUrl') ||
        '/dashboard';
      sessionStorage.removeItem('returnUrl');
```

```
      pan.publish('router.navigate', { path: returnUrl });

    } catch (error) {
      this.state.error = error.message;
      this.state.loading = false;
      this.render();
    }
  }

  render() {
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: block;
          max-width: 400px;
          margin: 50px auto;
        }

        form {
          background: white;
          padding: 30px;
          border-radius: 8px;
          box-shadow: 0 2px 10px rgba(0,0,0,0.1);
        }

        h2 {
          margin: 0 0 20px;
          color: #333;
        }

        .error {
          background: #fee;
          color: #c00;
          padding: 10px;
```

```
    border-radius: 4px;
    margin-bottom: 15px;
  }

  .form-group {
    margin-bottom: 15px;
  }

  label {
    display: block;
    margin-bottom: 5px;
    font-weight: 600;
    color: #555;
  }

  input {
    width: 100%;
    padding: 10px;
    border: 1px solid #ddd;
    border-radius: 4px;
    font-size: 14px;
  }

  input:focus {
    outline: none;
    border-color: #667eea;
  }

  button {
    width: 100%;
    padding: 12px;
    background: #667eea;
    color: white;
    border: none;
```

```
        border-radius: 4px;
        font-size: 16px;
        cursor: pointer;
      }

      button:hover:not(:disabled) {
        background: #5568d3;
      }

      button:disabled {
        opacity: 0.6;
        cursor: not-allowed;
      }

      .signup-link {
        text-align: center;
        margin-top: 15px;
        color: #666;
      }

      .signup-link a {
        color: #667eea;
        text-decoration: none;
      }
    </style>

    <form>
      <h2>Login</h2>

      ${this.state.error ? `
        <div class="error">${this.state.error}</div>
      ` : ''}

      <div class="form-group">
```

```
        <label for="email">Email</label>
        <input
          type="email"
          id="email"
          value="${this.state.email}"
          required
          autocomplete="email"
        >
      </div>

      <div class="form-group">
        <label for="password">Password</label>
        <input
          type="password"
          id="password"
          value="${this.state.password}"
          required
          autocomplete="current-password"
        >
      </div>

      <button type="submit" ?disabled="${this.state.loading}">
        ${this.state.loading ? 'Logging in...' : 'Login'}
      </button>

      <div class="signup-link">
        Don't have an account? <a href="/signup">Sign up</a>
      </div>
    </form>
  `;

  // Attach event listeners
  const form = this.shadowRoot.querySelector('form');
  const emailInput = this.shadowRoot.querySelector('#email');
```

```
    const passwordInput = this.shadowRoot.querySelector('#password');

    form.addEventListener('submit', (e) => this.handleSubmit(e));

    emailInput.addEventListener('input', (e) => {
      this.state.email = e.target.value;
    });

    passwordInput.addEventListener('input', (e) => {
      this.state.password = e.target.value;
    });

    // Handle signup link
    const signupLink = this.shadowRoot.querySelector('.signup-link a');
    signupLink?.addEventListener('click', (e) => {
      e.preventDefault();
      pan.publish('router.navigate', { path: '/signup' });
    });
  }
}

customElements.define('login-form', LoginForm);
```

12.7.2 Signup Component

```
// components/signup-form.js
class SignupForm extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.state = {
      name: '',
      email: '',
      password: '',
      confirmPassword: '',
      loading: false,
      errors: {}
    };
  }

  connectedCallback() {
    this.render();
  }

  validate() {
    const errors = {};

    if (!this.state.name || this.state.name.length < 2) {
      errors.name = 'Name must be at least 2 characters';
    }

    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(this.state.email)) {
      errors.email = 'Invalid email address';
    }

    if (this.state.password.length < 8) {
      errors.password = 'Password must be at least 8 characters';
    }
```

```
    if (this.state.password !== this.state.confirmPassword) {
      errors.confirmPassword = 'Passwords do not match';
    }

    return errors;
  }

  async handleSubmit(e) {
    e.preventDefault();

    const errors = this.validate();
    if (Object.keys(errors).length > 0) {
      this.state.errors = errors;
      this.render();
      return;
    }

    this.state.loading = true;
    this.state.errors = {};
    this.render();

    try {
      const response = await fetch('/api/auth/signup', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          name: this.state.name,
          email: this.state.email,
          password: this.state.password
        })
      });

      if (!response.ok) {
```

```
        const error = await response.json();
        throw new Error(error.message || 'Signup failed');
      }

      const { accessToken, refreshToken, user } = await response.json();

      auth.setTokens(accessToken, refreshToken);
      pan.publish('auth.login', { user }, { retained: true });
      pan.publish('router.navigate', { path: '/dashboard' });

    } catch (error) {
      this.state.errors = { general: error.message };
      this.state.loading = false;
      this.render();
    }
  }

  render() {
    this.shadowRoot.innerHTML = `
      <style>
        /* Similar styles to login-form */
        :host {
          display: block;
          max-width: 400px;
          margin: 50px auto;
        }
        /* ... (copy styles from login-form) ... */
      </style>

      <form>
        <h2>Create Account</h2>

        ${this.state.errors.general ? `
          <div class="error">${this.state.errors.general}</div>
```

```
      ` : ''}

      <div class="form-group">
        <label for="name">Name</label>
        <input type="text" id="name" value="${this.state.name}"
      required>
        ${this.state.errors.name ? `
          <span class="field-error">${this.state.errors.name}</span>
        ` : ''}
      </div>

      <div class="form-group">
        <label for="email">Email</label>
        <input type="email" id="email" value="${this.state.email}"
      required>
        ${this.state.errors.email ? `
          <span class="field-error">${this.state.errors.email}</span>
        ` : ''}
      </div>

      <div class="form-group">
        <label for="password">Password</label>
        <input type="password" id="password" required
      autocomplete="new-password">
        ${this.state.errors.password ? `
          <span class="field-error">${this.state.errors.password}</span>
        ` : ''}
      </div>

      <div class="form-group">
        <label for="confirmPassword">Confirm Password</label>
        <input type="password" id="confirmPassword" required>
        ${this.state.errors.confirmPassword ? `
          <span class="field-
      error">${this.state.errors.confirmPassword}</span>
        ` : ''}
```

```
        </div>

        <button type="submit" ?disabled="${this.state.loading}">
          ${this.state.loading ? 'Creating account...' : 'Sign Up'}
        </button>

        <div class="login-link">
          Already have an account? <a href="/login">Login</a>
        </div>
      </form>
    `;

    // Attach event listeners (similar to login-form)
    const form = this.shadowRoot.querySelector('form');
    form.addEventListener('submit', (e) => this.handleSubmit(e));

    // Update state on input
    ['name', 'email', 'password', 'confirmPassword'].forEach(field => {
      const input = this.shadowRoot.querySelector(`#${field}`);
      input?.addEventListener('input', (e) => {
        this.state[field] = e.target.value;
      });
    });
  }
}

customElements.define('signup-form', SignupForm);
```

12.8 OAuth Integration (GitHub Example)

OAuth allows users to log in with existing accounts from providers like GitHub, Google, or Facebook:

12.8.1 OAuth Flow

```
// services/oauth.js
class OAuthService {
  constructor() {
    this.providers = {
      github: {
        clientId: 'your-github-client-id',
        authUrl: 'https://github.com/login/oauth/authorize',
        scope: 'read:user user:email'
      }
    };
  }

  initiateLogin(provider) {
    const config = this.providers[provider];
    if (!config) throw new Error(`Unknown provider: ${provider}`);

    const redirectUri = `${window.location.origin}/auth/callback`;
    const state = this.generateState();

    // Store state for CSRF protection
    sessionStorage.setItem('oauth_state', state);

    const params = new URLSearchParams({
      client_id: config.clientId,
      redirect_uri: redirectUri,
      scope: config.scope,
      state
    });

    // Redirect to provider
    window.location.href = `${config.authUrl}?${params}`;
  }

  generateState() {
```

```
    return Array.from(crypto.getRandomValues(new Uint8Array(16)))
      .map(b => b.toString(16).padStart(2, '0'))
      .join('');
  }

  async handleCallback() {
    const params = new URLSearchParams(window.location.search);
    const code = params.get('code');
    const state = params.get('state');

    // Verify state (CSRF protection)
    const storedState = sessionStorage.getItem('oauth_state');
    if (state !== storedState) {
      throw new Error('Invalid state parameter');
    }

    // Exchange code for token with your backend
    const response = await fetch('/api/auth/github/callback', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ code })
    });

    if (!response.ok) {
      throw new Error('OAuth authentication failed');
    }

    const { accessToken, refreshToken, user } = await response.json();

    auth.setTokens(accessToken, refreshToken);
    pan.publish('auth.login', { user }, { retained: true });

    // Clean up
    sessionStorage.removeItem('oauth_state');
```

```
    return user;
  }
}

export const oauth = new OAuthService();
```

12.8.2 OAuth Button Component

```
class OAuthButtons extends HTMLElement {
  connectedCallback() {
    this.innerHTML = `
      <style>
        .oauth-buttons {
          display: flex;
          flex-direction: column;
          gap: 10px;
          margin: 20px 0;
        }

        .oauth-button {
          display: flex;
          align-items: center;
          justify-content: center;
          gap: 10px;
          padding: 12px;
          border: 1px solid #ddd;
          border-radius: 4px;
          background: white;
          cursor: pointer;
          transition: background 0.2s;
        }

        .oauth-button:hover {
          background: #f5f5f5;
        }

        .oauth-button img {
          width: 20px;
          height: 20px;
        }
      </style>
```

```
      <div class="oauth-buttons">
        <button class="oauth-button" data-provider="github">
          <img src="/icons/github.svg" alt="GitHub">
          <span>Continue with GitHub</span>
        </button>

        <button class="oauth-button" data-provider="google">
          <img src="/icons/google.svg" alt="Google">
          <span>Continue with Google</span>
        </button>
      </div>

      <div class="divider">
        <span>or</span>
      </div>
    `;

    this.querySelectorAll('[data-provider]').forEach(button => {
      button.addEventListener('click', () => {
        const provider = button.dataset.provider;
        oauth.initiateLogin(provider);
      });
    });
  }
}

customElements.define('oauth-buttons', OAuthButtons);
```

12.9 Session Management

Track active sessions and allow users to log out from all devices:

```
// services/session-manager.js
class SessionManager {
  async getSessions() {
    return await api.get('/auth/sessions');
  }

  async revokeSession(sessionId) {
    await api.delete(`/auth/sessions/${sessionId}`);
    pan.publish('session.revoked', { sessionId });
  }

  async revokeAllSessions() {
    await api.delete('/auth/sessions');
    pan.publish('session.all-revoked');
    auth.logout();
  }
}

export const sessionManager = new SessionManager();
```

12.9.1 Sessions Component

```
class SessionsList extends HTMLElement {
  constructor() {
    super();
    this.sessions = [];
  }

  async connectedCallback() {
    await this.loadSessions();
    this.render();

    pan.subscribe('session.revoked', () => this.loadSessions());
  }

  async loadSessions() {
    try {
      this.sessions = await sessionManager.getSessions();
      this.render();
    } catch (error) {
      console.error('Failed to load sessions:', error);
    }
  }

  async revokeSession(sessionId) {
    if (!confirm('Revoke this session?')) return;

    try {
      await sessionManager.revokeSession(sessionId);
    } catch (error) {
      alert('Failed to revoke session');
    }
  }

  async revokeAll() {
    if (!confirm('Log out from all devices?')) return;
```

```
    try {
      await sessionManager.revokeAllSessions();
    } catch (error) {
      alert('Failed to revoke sessions');
    }
  }

  render() {
    this.innerHTML = `
      <div class="sessions">
        <h3>Active Sessions</h3>

        ${this.sessions.map(session => `
          <div class="session" data-current="${session.isCurrent}">
            <div class="session-info">
              <strong>${session.device}</strong>
              <span>${session.location}</span>
              <small>Last active: ${new
        Date(session.lastActive).toLocaleString()}</small>
            </div>
            ${session.isCurrent ? `
              <span class="badge">Current</span>
            ` : `
              <button onclick="this.closest('sessions-
        list').revokeSession('${session.id}')">
                Revoke
              </button>
            `}
          </div>
        `).join('')}

        <button class="danger" onclick="this.closest('sessions-
        list').revokeAll()">
          Log out from all devices
        </button>
```

```
      </div>
    `;
  }
}

customElements.define('sessions-list', SessionsList);
```

12.10 XSS and CSRF Protection

12.10.1 Preventing XSS (Cross-Site Scripting)

Always sanitize user input before rendering:

```
// utils/sanitize.js
const ALLOWED_TAGS = ['b', 'i', 'em', 'strong', 'a', 'p', 'br'];

function sanitizeHtml(html) {
  const div = document.createElement('div');
  div.innerHTML = html;

  // Remove all scripts
  div.querySelectorAll('script').forEach(el => el.remove());

  // Remove event handlers
  div.querySelectorAll('*').forEach(el => {
    [...el.attributes].forEach(attr => {
      if (attr.name.startsWith('on')) {
        el.removeAttribute(attr.name);
      }
    });

    // Remove tags not in allowlist
    if (!ALLOWED_TAGS.includes(el.tagName.toLowerCase())) {
      el.replaceWith(...el.childNodes);
    }
  });

  return div.innerHTML;
}

export { sanitizeHtml };
```

12.10.2 CSRF Protection

Include CSRF tokens in state-changing requests:

```
// api-client.js
class SecureApiClient {
  getCsrfToken() {
    // Get from meta tag or cookie
    return document.querySelector('meta[name="csrf-token"]')?.content;
  }

  async fetch(endpoint, options = {}) {
    const headers = {
      'Content-Type': 'application/json',
      'X-CSRF-Token': this.getCsrfToken(),
      ...options.headers
    };

    return fetch(endpoint, { ...options, headers });
  }
}
```

Server should validate CSRF tokens on state-changing requests (POST, PUT, DELETE).

12.11 Password Reset Flow

```
// components/password-reset.js
class PasswordResetForm extends HTMLElement {
  async handleRequest(e) {
    e.preventDefault();
    const email = this.querySelector('#email').value;

    try {
      await fetch('/api/auth/password-reset', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ email })
      });

      this.innerHTML = `
        <p>If an account exists for ${email}, you will receive a password
        reset email.</p>
      `;
    } catch (error) {
      this.showError('Failed to send reset email');
    }
  }

  async handleReset(e) {
    e.preventDefault();
    const token = new
        URLSearchParams(window.location.search).get('token');
    const password = this.querySelector('#password').value;

    try {
      await fetch('/api/auth/password-reset/confirm', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ token, password })
      });
```

```
      pan.publish('router.navigate', { path: '/login' });
      pan.publish('notification.success', {
        message: 'Password reset successful'
      });
    } catch (error) {
      this.showError('Failed to reset password');
    }
  }
}
```

12.12 Troubleshooting

12.12.1 Problem: Token Expires Too Quickly

Symptom: Users frequently get logged out

Solution: Implement automatic token refresh in background:

```
// Auto-refresh tokens before expiry
class TokenRefreshManager {
  constructor() {
    this.refreshInterval = null;
  }

  start() {
    // Check token every minute
    this.refreshInterval = setInterval(async () => {
      const token = auth.getToken();
      if (!token) return;

      const payload = auth.decodeToken(token);
      const expiresIn = (payload.exp * 1000) - Date.now();

      // Refresh if expires in less than 5 minutes
      if (expiresIn < 5 * 60 * 1000) {
        try {
          await auth.refresh();
        } catch (error) {
          console.error('Auto-refresh failed:', error);
          auth.logout();
        }
      }
    }, 60 * 1000);
  }

  stop() {
    if (this.refreshInterval) {
      clearInterval(this.refreshInterval);
    }
  }
}
```

```
export const tokenRefresh = new TokenRefreshManager();
```

12.12.2 Problem: OAuth Callback Not Working

Symptom: After OAuth redirect, nothing happens

Solution: Ensure callback route is registered and handles the code:

```
// In your router setup
pan.subscribe('route.changed', ({ path }) => {
  if (path === '/auth/callback') {
    oauth.handleCallback()
      .then(() => {
        pan.publish('router.navigate', { path: '/dashboard' });
      })
      .catch(error => {
        console.error('OAuth callback failed:', error);
        pan.publish('router.navigate', { path: '/login' });
      });
  }
});
```

12.12.3 Problem: Users See Flash of Protected Content

Symptom: Protected page renders briefly before redirect to login

Solution: Check authentication before mounting component:

```
class ProtectedPage extends HTMLElement {
  connectedCallback() {
    // Don't render until auth check complete
    this.checkAuthThenRender();
  }

  async checkAuthThenRender() {
    if (!auth.isAuthenticated()) {
      sessionStorage.setItem('returnUrl', window.location.pathname);
      pan.publish('router.navigate', { path: '/login' });
      return;
    }

    this.render();
  }

  render() {
    // Only called if authenticated
    this.innerHTML = `
      <h1>Protected Content</h1>
      <p>Only authenticated users see this</p>
    `;
  }
}
```

12.12.4 Problem: Infinite Redirect Loop

Symptom: App keeps redirecting between login and protected route

Solution: Check for redirect loops in route guard:

```
class RouteGuard extends HTMLElement {
  checkAuth() {
    if (!auth.isAuthenticated() && window.location.pathname !== '/login')
        {
      sessionStorage.setItem('returnUrl', window.location.pathname);
      pan.publish('router.navigate', { path: '/login' });
    }
  }
}
```

12.13 Best Practices

1. **Never store passwords** - Always hash on the server
2. **Use HTTPS everywhere** - Tokens in plaintext over HTTP are vulnerable
3. **Implement token refresh** - Don't force users to re-login frequently
4. **Validate on the server** - Client-side validation is for UX, not security
5. **Use httpOnly cookies for tokens** - Protects against XSS
6. **Implement CSRF protection** - Required for cookie-based auth
7. **Rate limit authentication endpoints** - Prevent brute force attacks
8. **Log authentication events** - Track suspicious activity
9. **Use secure password requirements** - Minimum 8 characters, complexity rules
10. **Provide 2FA** - Add extra layer of security for sensitive apps

12.14 Exercises

12.14.1 Exercise 1: Add "Remember Me"

Extend the login form with a "Remember Me" checkbox that:

- Uses a longer-lived refresh token when checked
- Falls back to session-only auth when unchecked

- Persists the preference across sessions

12.14.2 Exercise 2: Implement 2FA

Add two-factor authentication:

- Generate TOTP secret on enrollment
- Display QR code for authenticator apps
- Validate TOTP codes on login
- Provide backup codes for account recovery

12.14.3 Exercise 3: Password Strength Meter

Build a password strength indicator that:

- Shows strength in real-time as user types
- Checks length, character variety, common passwords
- Provides feedback on how to improve
- Blocks weak passwords on submission

12.14.4 Exercise 4: Activity Log

Create an activity log component that:

- Tracks login attempts, password changes, session activity
- Displays timeline of security events
- Alerts on suspicious activity (new device, new location)
- Allows filtering by event type

12.15 Summary

Authentication and security are critical to any application that handles user data. LARC applications use industry-standard patterns:

- **JWT tokens** for stateless authentication
- **Automatic token refresh** to maintain sessions
- **OAuth integration** for social login
- **RBAC** for fine-grained authorization
- **XSS and CSRF protection** to prevent attacks
- **Secure token storage** with httpOnly cookies when possible
- **Route guards** to protect sensitive pages
- **Session management** to track and revoke active sessions

Security isn't a checkbox—it's an ongoing practice. Stay updated on security best practices, use HTTPS everywhere, validate all input, and treat user data with respect.

12.16 Further Reading

For complete authentication reference: - *Building with LARC* Chapter 8: Authentication and Authorization - All auth patterns and strategies - *Building with LARC* Chapter 14: Error Handling and Debugging - Security debugging - *Building with LARC* Appendix E: Recipes and Patterns - Auth implementation recipes

13 Server Integration

LARC is frontend-agnostic about backends. Whether you're using Node.js, Python, PHP, or any other server technology, the patterns remain the same: your components communicate via HTTP and WebSockets, and the PAN bus coordinates the frontend.

13.1 Node.js with Express

Express is the most popular Node.js framework, and it pairs naturally with LARC:

```
// server.js
const express = require('express');
const cors = require('cors');
const jwt = require('jsonwebtoken');

const app = express();

app.use(cors());
app.use(express.json());
app.use(express.static('public'));

// Authentication middleware
function authenticate(req, res, next) {
  const authHeader = req.headers.authorization;
  if (!authHeader?.startsWith('Bearer ')) {
    return res.status(401).json({ error: 'No token provided' });
  }

  const token = authHeader.slice(7);
  try {
    req.user = jwt.verify(token, process.env.JWT_SECRET);
    next();
  } catch {
    res.status(401).json({ error: 'Invalid token' });
  }
}

// Public routes
app.post('/api/auth/login', async (req, res) => {
  const { email, password } = req.body;

  const user = await db.users.findByEmail(email);
  if (!user || !await bcrypt.compare(password, user.password)) {
    return res.status(401).json({ error: 'Invalid credentials' });
```

```
  }

  const accessToken = jwt.sign(
    { id: user.id, email: user.email, role: user.role },
    process.env.JWT_SECRET,
    { expiresIn: '15m' }
  );

  res.json({ accessToken, user: { id: user.id, email: user.email } });
});

// Protected routes
app.get('/api/users', authenticate, async (req, res) => {
  const users = await db.users.findAll();
  res.json(users);
});

app.post('/api/users', authenticate, async (req, res) => {
  const user = await db.users.create(req.body);
  res.status(201).json(user);
});

app.listen(3000, () => console.log('Server running on port 3000'));
```

13.2 Python with Flask

Flask provides a lightweight Python backend:

```python
# app.py
from flask import Flask, jsonify, request
from flask_cors import CORS
from functools import wraps
import jwt

app = Flask(__name__, static_folder='public')
CORS(app)

def token_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        auth_header = request.headers.get('Authorization')
        if not auth_header or not auth_header.startswith('Bearer '):
            return jsonify({'error': 'No token provided'}), 401

        token = auth_header[7:]
        try:
            data = jwt.decode(token, app.config['SECRET_KEY'],
        algorithms=['HS256'])
            request.user = data
        except:
            return jsonify({'error': 'Invalid token'}), 401

        return f(*args, **kwargs)
    return decorated

@app.route('/api/users')
@token_required
def get_users():
    users = User.query.all()
    return jsonify([u.to_dict() for u in users])

@app.route('/api/users', methods=['POST'])
```

```python
@token_required
def create_user():
    data = request.get_json()
    user = User(**data)
    db.session.add(user)
    db.session.commit()
    return jsonify(user.to_dict()), 201

if __name__ == '__main__':
    app.run(debug=True)
```

13.3 PHP Integration

PHP remains popular for web backends:

```php
<?php
// api.php
header('Content-Type: application/json');
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Headers: Content-Type, Authorization');

require_once 'vendor/autoload.php';
use Firebase\JWT\JWT;
use Firebase\JWT\Key;

function authenticate() {
    $headers = getallheaders();
    $auth = $headers['Authorization'] ?? '';

    if (!preg_match('/Bearer\s+(.*)$/i', $auth, $matches)) {
        http_response_code(401);
        echo json_encode(['error' => 'No token provided']);
        exit;
    }

    try {
        return JWT::decode($matches[1], new Key($_ENV['JWT_SECRET'],
        'HS256'));
    } catch (Exception $e) {
        http_response_code(401);
        echo json_encode(['error' => 'Invalid token']);
        exit;
    }
}

$method = $_SERVER['REQUEST_METHOD'];
$path = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);

if ($path === '/api/users' && $method === 'GET') {
```

```
    $user = authenticate();
    $users = $pdo->query('SELECT id, name, email FROM users')-
        >fetchAll(PDO::FETCH_ASSOC);
    echo json_encode($users);
}
```

13.4 Real-Time with WebSockets

For real-time features, add WebSocket support. Here's Node.js with the `ws` library:

```
// websocket-server.js
const WebSocket = require('ws');
const jwt = require('jsonwebtoken');

const wss = new WebSocket.Server({ port: 8080 });

const clients = new Map();

wss.on('connection', (ws, req) => {
  // Authenticate connection
  const url = new URL(req.url, 'http://localhost');
  const token = url.searchParams.get('token');

  try {
    const user = jwt.verify(token, process.env.JWT_SECRET);
    clients.set(ws, user);
  } catch {
    ws.close(4001, 'Unauthorized');
    return;
  }

  ws.on('message', (data) => {
    const message = JSON.parse(data);
    handleMessage(ws, message);
  });

  ws.on('close', () => {
    clients.delete(ws);
  });
});

function handleMessage(sender, message) {
  switch (message.type) {
    case 'broadcast':
```

```
      // Send to all connected clients
      wss.clients.forEach(client => {
        if (client.readyState === WebSocket.OPEN) {
          client.send(JSON.stringify(message));
        }
      });
      break;

    case 'direct':
      // Send to specific user
      clients.forEach((user, client) => {
        if (user.id === message.targetUserId && client.readyState ===
        WebSocket.OPEN) {
          client.send(JSON.stringify(message));
        }
      });
      break;
  }
}
```

13.5 Database Patterns

Most backends need a database. Here's a clean repository pattern:

```
// user-repository.js
class UserRepository {
  constructor(db) {
    this.db = db;
  }

  async findAll() {
    return this.db.query('SELECT id, name, email FROM users');
  }

  async findById(id) {
    const [user] = await this.db.query(
      'SELECT id, name, email FROM users WHERE id = ?',
      [id]
    );
    return user;
  }

  async create(data) {
    const result = await this.db.query(
      'INSERT INTO users (name, email, password) VALUES (?, ?, ?)',
      [data.name, data.email, data.hashedPassword]
    );
    return this.findById(result.insertId);
  }

  async update(id, data) {
    await this.db.query(
      'UPDATE users SET name = ?, email = ? WHERE id = ?',
      [data.name, data.email, id]
    );
    return this.findById(id);
  }
```

```
  async delete(id) {
    await this.db.query('DELETE FROM users WHERE id = ?', [id]);
  }
}
```

13.6 Complete CRUD API Example

Let’s build a complete REST API for a todo application with full CRUD operations:

13.6.1 Express Backend with SQLite

```
// server/app.js
const express = require('express');
const cors = require('cors');
const sqlite3 = require('sqlite3').verbose();
const { open } = require('sqlite');

const app = express();
app.use(cors());
app.use(express.json());

// Database setup
let db;
(async () => {
  db = await open({
    filename: './database.sqlite',
    driver: sqlite3.Database
  });

  await db.exec(`
    CREATE TABLE IF NOT EXISTS todos (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      title TEXT NOT NULL,
      completed BOOLEAN DEFAULT 0,
      created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
      updated_at DATETIME DEFAULT CURRENT_TIMESTAMP
    )
  `);
})();

// Get all todos
app.get('/api/todos', async (req, res) => {
  try {
    const todos = await db.all('SELECT * FROM todos ORDER BY created_at
        DESC');
```

```
    res.json(todos);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

// Get single todo
app.get('/api/todos/:id', async (req, res) => {
  try {
    const todo = await db.get('SELECT * FROM todos WHERE id = ?',
        req.params.id);
    if (!todo) {
      return res.status(404).json({ error: 'Todo not found' });
    }
    res.json(todo);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

// Create todo
app.post('/api/todos', async (req, res) => {
  try {
    const { title } = req.body;
    if (!title) {
      return res.status(400).json({ error: 'Title is required' });
    }

    const result = await db.run(
      'INSERT INTO todos (title) VALUES (?)',
      title
    );

    const todo = await db.get('SELECT * FROM todos WHERE id = ?',
        result.lastID);
```

```
    res.status(201).json(todo);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

// Update todo
app.put('/api/todos/:id', async (req, res) => {
  try {
    const { title, completed } = req.body;

    await db.run(
      'UPDATE todos SET title = ?, completed = ?, updated_at =
        CURRENT_TIMESTAMP WHERE id = ?',
      title,
      completed ? 1 : 0,
      req.params.id
    );

    const todo = await db.get('SELECT * FROM todos WHERE id = ?',
        req.params.id);
    res.json(todo);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

// Delete todo
app.delete('/api/todos/:id', async (req, res) => {
  try {
    await db.run('DELETE FROM todos WHERE id = ?', req.params.id);
    res.status(204).send();
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
```

```
});


app.listen(3000, () => console.log('Server running on
        http://localhost:3000'));
```

13.6.2 Frontend Integration

```
// frontend/services/todo-service.js
class TodoService {
  constructor(baseUrl = 'http://localhost:3000/api') {
    this.baseUrl = baseUrl;
  }

  async getAll() {
    const response = await fetch(`${this.baseUrl}/todos`);
    if (!response.ok) throw new Error('Failed to fetch todos');
    return response.json();
  }

  async create(title) {
    const response = await fetch(`${this.baseUrl}/todos`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ title })
    });
    if (!response.ok) throw new Error('Failed to create todo');
    return response.json();
  }

  async update(id, updates) {
    const response = await fetch(`${this.baseUrl}/todos/${id}`, {
      method: 'PUT',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(updates)
    });
    if (!response.ok) throw new Error('Failed to update todo');
    return response.json();
  }

  async delete(id) {
    const response = await fetch(`${this.baseUrl}/todos/${id}`, {
```

```
      method: 'DELETE'
    });
    if (!response.ok) throw new Error('Failed to delete todo');
  }
}

export const todoService = new TodoService();
```

13.7 Using ORMs

Object-Relational Mappers simplify database operations. Here's Prisma (Node.js) and SQLAlchemy (Python):

13.7.1 Prisma (Node.js)

```
// prisma/schema.prisma
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

generator client {
  provider = "prisma-client-js"
}

model User {
  id        Int      @id @default(autoincrement())
  email     String   @unique
  name      String?
  posts     Post[]
  createdAt DateTime @default(now())
}

model Post {
  id        Int      @id @default(autoincrement())
  title     String
  content   String?
  published Boolean  @default(false)
  author    User     @relation(fields: [authorId], references: [id])
  authorId  Int
  createdAt DateTime @default(now())
}
```

```
// server/routes/posts.js
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();

app.get('/api/posts', async (req, res) => {
  const posts = await prisma.post.findMany({
    include: {
      author: {
        select: { id: true, name: true, email: true }
      }
    },
    orderBy: { createdAt: 'desc' }
  });
  res.json(posts);
});

app.post('/api/posts', authenticate, async (req, res) => {
  const { title, content } = req.body;

  const post = await prisma.post.create({
    data: {
      title,
      content,
      author: {
        connect: { id: req.user.id }
      }
    },
    include: { author: true }
  });

  res.status(201).json(post);
});

app.put('/api/posts/:id', authenticate, async (req, res) => {
```

```
  const { id } = req.params;
  const { title, content, published } = req.body;

  // Check ownership
  const post = await prisma.post.findUnique({
    where: { id: parseInt(id) }
  });

  if (post.authorId !== req.user.id) {
    return res.status(403).json({ error: 'Forbidden' });
  }

  const updated = await prisma.post.update({
    where: { id: parseInt(id) },
    data: { title, content, published },
    include: { author: true }
  });

  res.json(updated);
});
```

13.7.2 SQLAlchemy (Python)

```python
# models.py
from sqlalchemy import Column, Integer, String, Text, Boolean, DateTime,
        ForeignKey
from sqlalchemy.orm import relationship
from datetime import datetime
from database import Base

class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    email = Column(String, unique=True, index=True)
    name = Column(String)
    hashed_password = Column(String)
    posts = relationship("Post", back_populates="author")
    created_at = Column(DateTime, default=datetime.utcnow)

class Post(Base):
    __tablename__ = "posts"

    id = Column(Integer, primary_key=True, index=True)
    title = Column(String, index=True)
    content = Column(Text)
    published = Column(Boolean, default=False)
    author_id = Column(Integer, ForeignKey("users.id"))
    author = relationship("User", back_populates="posts")
    created_at = Column(DateTime, default=datetime.utcnow)

# routes.py
from fastapi import FastAPI, Depends, HTTPException
from sqlalchemy.orm import Session
from typing import List
import models, schemas
from database import get_db
```

```
app = FastAPI()

@app.get("/api/posts", response_model=List[schemas.Post])
def get_posts(db: Session = Depends(get_db)):
    return
        db.query(models.Post).order_by(models.Post.created_at.desc()).all()



@app.post("/api/posts", response_model=schemas.Post)
def create_post(post: schemas.PostCreate, db: Session = Depends(get_db),
        user=Depends(get_current_user)):
    db_post = models.Post(**post.dict(), author_id=user.id)
    db.add(db_post)
    db.commit()
    db.refresh(db_post)
    return db_post

@app.put("/api/posts/{post_id}", response_model=schemas.Post)
def update_post(post_id: int, post: schemas.PostUpdate, db: Session =
        Depends(get_db), user=Depends(get_current_user)):
    db_post = db.query(models.Post).filter(models.Post.id ==
        post_id).first()
    if not db_post:
        raise HTTPException(status_code=404, detail="Post not found")
    if db_post.author_id != user.id:
        raise HTTPException(status_code=403, detail="Not authorized")

    for key, value in post.dict(exclude_unset=True).items():
        setattr(db_post, key, value)

    db.commit()
    db.refresh(db_post)
    return db_post
```

13.8 File Upload and Download

Handle file uploads from LARC components:

13.8.1 Express File Upload

```
// server.js
const multer = require('multer');
const path = require('path');

// Configure storage
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'uploads/');
  },
  filename: (req, file, cb) => {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() *
        1E9);
    cb(null, uniqueSuffix + path.extname(file.originalname));
  }
});

const upload = multer({
  storage,
  limits: { fileSize: 5 * 1024 * 1024 }, // 5MB
  fileFilter: (req, file, cb) => {
    const allowedTypes = /jpeg|jpg|png|gif|pdf/;
    const extname =
        allowedTypes.test(path.extname(file.originalname).toLowerCase());
    const mimetype = allowedTypes.test(file.mimetype);

    if (mimetype && extname) {
      return cb(null, true);
    }
    cb(new Error('Invalid file type'));
  }
});

// Upload endpoint
app.post('/api/upload', authenticate, upload.single('file'), async (req,
        res) => {
```

```
  if (!req.file) {
    return res.status(400).json({ error: 'No file uploaded' });
  }

  // Save file metadata to database
  const file = await db.files.create({
    userId: req.user.id,
    filename: req.file.filename,
    originalName: req.file.originalname,
    mimetype: req.file.mimetype,
    size: req.file.size,
    path: req.file.path
  });

  res.json({
    id: file.id,
    filename: file.filename,
    url: `/uploads/${file.filename}`
  });
});

// Download endpoint
app.get('/api/files/:id/download', authenticate, async (req, res) => {
  const file = await db.files.findById(req.params.id);

  if (!file) {
    return res.status(404).json({ error: 'File not found' });
  }

  res.download(file.path, file.originalName);
});

// Serve uploaded files
app.use('/uploads', express.static('uploads'));
```

13.8.2 Frontend Upload Component

```
class FileUpload extends HTMLElement {
  async handleUpload(e) {
    const file = e.target.files[0];
    if (!file) return;

    const formData = new FormData();
    formData.append('file', file);

    try {
      const token = auth.getToken();
      const response = await fetch('http://localhost:3000/api/upload', {
        method: 'POST',
        headers: {
          'Authorization': `Bearer ${token}`
        },
        body: formData
      });

      if (!response.ok) throw new Error('Upload failed');

      const result = await response.json();
      pan.publish('file.uploaded', result);

      this.showSuccess(`File uploaded: ${result.filename}`);
    } catch (error) {
      this.showError(error.message);
    }
  }

  connectedCallback() {
    this.innerHTML = `
      <div class="upload-container">
        <input type="file" id="file-input">
        <button onclick="document.getElementById('file-input').click()">
```

```
          Choose File
        </button>
        <div class="message"></div>
      </div>
    `;

    this.querySelector('#file-input').addEventListener('change', (e) => {
      this.handleUpload(e);
    });
  }
}

customElements.define('file-upload', FileUpload);
```

13.9 Real-Time Chat Application

Complete example combining HTTP and WebSocket:

13.9.1 Backend (Node.js)

```
// chat-server.js
const express = require('express');
const http = require('http');
const WebSocket = require('ws');
const jwt = require('jsonwebtoken');

const app = express();
const server = http.createServer(app);
const wss = new WebSocket.Server({ server });

app.use(express.json());

// Store active connections
const connections = new Map(); // ws -> user
const rooms = new Map(); // roomId -> Set of ws

// REST API for chat history
app.get('/api/rooms/:roomId/messages', authenticate, async (req, res) =>
        {
  const messages = await db.query(
    'SELECT * FROM messages WHERE room_id = ? ORDER BY created_at ASC',
    [req.params.roomId]
  );
  res.json(messages);
});

app.post('/api/rooms/:roomId/messages', authenticate, async (req, res)
        => {
  const { content } = req.body;

  const message = await db.query(
    'INSERT INTO messages (room_id, user_id, content) VALUES (?, ?, ?)',
    [req.params.roomId, req.user.id, content]
  );
```

```
  res.status(201).json(message);
});

// WebSocket for real-time
wss.on('connection', (ws, req) => {
  const url = new URL(req.url, 'ws://localhost');
  const token = url.searchParams.get('token');

  try {
    const user = jwt.verify(token, process.env.JWT_SECRET);
    connections.set(ws, user);

    ws.on('message', (data) => {
      const message = JSON.parse(data);
      handleMessage(ws, user, message);
    });

    ws.on('close', () => {
      connections.delete(ws);
      // Remove from all rooms
      rooms.forEach(roomClients => roomClients.delete(ws));
    });

    ws.send(JSON.stringify({ type: 'connected', user }));
  } catch {
    ws.close(4001, 'Unauthorized');
  }
});

function handleMessage(ws, user, message) {
  switch (message.type) {
    case 'join-room':
      if (!rooms.has(message.roomId)) {
        rooms.set(message.roomId, new Set());
```

```
      }
      rooms.get(message.roomId).add(ws);

      // Notify room
      broadcastToRoom(message.roomId, {
        type: 'user-joined',
        user: { id: user.id, name: user.name }
      });
      break;

    case 'leave-room':
      rooms.get(message.roomId)?.delete(ws);

      broadcastToRoom(message.roomId, {
        type: 'user-left',
        user: { id: user.id, name: user.name }
      });
      break;

    case 'chat-message':
      // Save to database
      db.query(
        'INSERT INTO messages (room_id, user_id, content) VALUES (?, ?,
        ?)',
        [message.roomId, user.id, message.content]
      ).then(result => {
        // Broadcast to room
        broadcastToRoom(message.roomId, {
          type: 'chat-message',
          message: {
            id: result.insertId,
            userId: user.id,
            userName: user.name,
            content: message.content,
            createdAt: new Date()
```

```
          }
        });
      });
      break;

    case 'typing':
      broadcastToRoom(message.roomId, {
        type: 'user-typing',
        user: { id: user.id, name: user.name }
      }, ws);
      break;
  }
}

function broadcastToRoom(roomId, message, exclude = null) {
  const roomClients = rooms.get(roomId);
  if (!roomClients) return;

  roomClients.forEach(client => {
    if (client !== exclude && client.readyState === WebSocket.OPEN) {
      client.send(JSON.stringify(message));
    }
  });
}

server.listen(3000, () => console.log('Chat server running on port
        3000'));
```

13.9.2 Frontend Chat Component

```
class ChatRoom extends HTMLElement {
  constructor() {
    super();
    this.roomId = this.getAttribute('room-id');
    this.messages = [];
    this.ws = null;
    this.typingTimer = null;
  }

  async connectedCallback() {
    await this.loadHistory();
    this.connectWebSocket();
    this.render();
  }

  async loadHistory() {
    try {
      this.messages = await api.get(`/rooms/${this.roomId}/messages`);
    } catch (error) {
      console.error('Failed to load chat history:', error);
    }
  }

  connectWebSocket() {
    const token = auth.getToken();
    this.ws = new WebSocket(`ws://localhost:3000?token=${token}`);

    this.ws.onopen = () => {
      this.ws.send(JSON.stringify({
        type: 'join-room',
        roomId: this.roomId
      }));
    };
```

```
    this.ws.onmessage = (event) => {
      const data = JSON.parse(event.data);
      this.handleWebSocketMessage(data);
    };

    this.ws.onclose = () => {
      // Reconnect after 3 seconds
      setTimeout(() => this.connectWebSocket(), 3000);
    };
  }

  handleWebSocketMessage(data) {
    switch (data.type) {
      case 'chat-message':
        this.messages.push(data.message);
        this.render();
        this.scrollToBottom();
        break;

      case 'user-joined':
        this.showNotification(`${data.user.name} joined`);
        break;

      case 'user-left':
        this.showNotification(`${data.user.name} left`);
        break;

      case 'user-typing':
        this.showTypingIndicator(data.user.name);
        break;
    }
  }

  sendMessage(content) {
```

```
    if (!content.trim()) return;

    this.ws.send(JSON.stringify({
      type: 'chat-message',
      roomId: this.roomId,
      content
    }));

    this.querySelector('#message-input').value = '';
  }

  handleTyping() {
    clearTimeout(this.typingTimer);

    this.ws.send(JSON.stringify({
      type: 'typing',
      roomId: this.roomId
    }));

    this.typingTimer = setTimeout(() => {
      // Stop typing indicator after 2 seconds
    }, 2000);
  }

  render() {
    this.innerHTML = `
      <style>
        .chat-container {
          display: flex;
          flex-direction: column;
          height: 600px;
          border: 1px solid #ddd;
          border-radius: 8px;
        }
```

```
.messages {
  flex: 1;
  overflow-y: auto;
  padding: 20px;
}

.message {
  margin-bottom: 15px;
}

.message-author {
  font-weight: 600;
  color: #667eea;
}

.message-content {
  margin-top: 5px;
}

.message-time {
  font-size: 12px;
  color: #999;
}

.input-area {
  display: flex;
  gap: 10px;
  padding: 15px;
  border-top: 1px solid #ddd;
}

input {
  flex: 1;
```

```
      padding: 10px;
      border: 1px solid #ddd;
      border-radius: 4px;
    }

    button {
      padding: 10px 20px;
      background: #667eea;
      color: white;
      border: none;
      border-radius: 4px;
      cursor: pointer;
    }
  </style>

  <div class="chat-container">
    <div class="messages">
      ${this.messages.map(msg => `
        <div class="message">
          <div class="message-author">${msg.userName}</div>
          <div class="message-
   content">${this.escapeHtml(msg.content)}</div>
          <div class="message-time">${new
   Date(msg.createdAt).toLocaleTimeString()}</div>
        </div>
      `).join('')}
    </div>

    <div class="input-area">
      <input
        type="text"
        id="message-input"
        placeholder="Type a message..."
      >
      <button id="send-btn">Send</button>
```

```
        </div>
      </div>
    `;

    const input = this.querySelector('#message-input');
    const sendBtn = this.querySelector('#send-btn');

    input.addEventListener('input', () => this.handleTyping());
    input.addEventListener('keypress', (e) => {
      if (e.key === 'Enter') {
        this.sendMessage(input.value);
      }
    });

    sendBtn.addEventListener('click', () => {
      this.sendMessage(input.value);
    });
  }

  escapeHtml(text) {
    const div = document.createElement('div');
    div.textContent = text;
    return div.innerHTML;
  }

  scrollToBottom() {
    const messages = this.querySelector('.messages');
    messages.scrollTop = messages.scrollHeight;
  }

  disconnectedCallback() {
    if (this.ws) {
      this.ws.send(JSON.stringify({
        type: 'leave-room',
```

```
        roomId: this.roomId
      }));
      this.ws.close();
    }
  }
}

customElements.define('chat-room', ChatRoom);
```

13.10 Troubleshooting

13.10.1 Problem: CORS Errors in Development

Symptom: `Access-Control-Allow-Origin` errors

Solution: Configure CORS properly for development:

```
// Express
const cors = require('cors');
app.use(cors({
  origin: 'http://localhost:5173', // Your frontend URL
  credentials: true
}));

// Or for all origins in development
if (process.env.NODE_ENV === 'development') {
  app.use(cors({ origin: '*' }));
}
```

13.10.2 Problem: Database Connection Pool Exhaustion

Symptom: “Too many connections” errors

Solution: Configure connection pooling:

```
const { Pool } = require('pg');

const pool = new Pool({
  max: 20, // Maximum connections
  idleTimeoutMillis: 30000,
  connectionTimeoutMillis: 2000,
});

// Always release connections
app.get('/api/users', async (req, res) => {
  const client = await pool.connect();
  try {
    const result = await client.query('SELECT * FROM users');
    res.json(result.rows);
  } finally {
    client.release(); // Important!
  }
});
```

13.10.3 Problem: File Uploads Failing

Symptom: 413 Payload Too Large or multipart parsing errors

Solution: Increase limits and configure multer properly:

```
app.use(express.json({ limit: '10mb' }));
app.use(express.urlencoded({ limit: '10mb', extended: true }));

const upload = multer({
  limits: {
    fileSize: 10 * 1024 * 1024, // 10MB
    files: 5 // Max 5 files
  }
});
```

13.10.4 Problem: WebSocket Connection Drops

Symptom: Frequent disconnections

Solution: Implement heartbeat/ping-pong:

```
// Server
wss.on('connection', (ws) => {
  ws.isAlive = true;
  ws.on('pong', () => {
    ws.isAlive = true;
  });
});

// Ping clients every 30 seconds
const interval = setInterval(() => {
  wss.clients.forEach((ws) => {
    if (ws.isAlive === false) return ws.terminate();

    ws.isAlive = false;
    ws.ping();
  });
}, 30000);

// Client
setInterval(() => {
  if (ws.readyState === WebSocket.OPEN) {
    ws.send(JSON.stringify({ type: 'ping' }));
  }
}, 30000);
```

13.11 Best Practices

1. **Use environment variables** - Never hardcode secrets or config
2. **Validate input** - Always validate on the server, not just client
3. **Use connection pooling** - Reuse database connections
4. **Implement rate limiting** - Prevent abuse with rate limits
5. **Log errors properly** - Use structured logging (Winston, Bunyan)

6. **Handle graceful shutdown** - Close connections properly on SIGTERM
7. **Use transactions** - For operations that must succeed or fail together
8. **Sanitize user input** - Prevent SQL injection and XSS
9. **Set security headers** - Use helmet.js or similar
10. **Monitor performance** - Use APM tools (New Relic, DataDog)

13.12 Exercises

13.12.1 Exercise 1: Build a Blog API

Create a REST API with:

- User registration and authentication
- CRUD operations for blog posts
- Comments on posts
- Search functionality
- Tag/category filtering

Bonus: Add pagination and sorting options.

13.12.2 Exercise 2: Real-Time Notifications

Build a notification system with:

- WebSocket connection for real-time delivery
- Fallback to polling if WebSocket unavailable
- Mark as read/unread functionality
- Notification categories (info, warning, error)
- Persistence to database

Bonus: Add push notifications for mobile.

13.12.3 Exercise 3: File Management System

Create a file management API with:

- Upload multiple files
- Organize files in folders
- Share files with other users
- Generate temporary download links
- Thumbnail generation for images

Bonus: Implement file versioning.

13.12.4 Exercise 4: GraphQL API

Convert a REST API to GraphQL:

- Define schema for users, posts, comments
- Implement resolvers
- Add authentication to resolvers
- Implement subscriptions for real-time
- Optimize N+1 queries with DataLoader

Bonus: Add GraphQL Playground for testing.

13.13 Summary

Server integration with LARC follows standard web patterns—your frontend communicates via HTTP and WebSockets, regardless of backend technology:

- **REST APIs** with Express, Flask, FastAPI, or PHP
- **ORMs** like Prisma, SQLAlchemy for database access
- **File uploads** with multer or similar libraries
- **Real-time features** with WebSockets
- **Authentication** via JWT tokens

- **Database patterns** with repositories and connection pooling

LARC doesn't dictate your backend choices. Use whatever server technology fits your team's expertise and requirements. The PAN bus on the frontend keeps your components decoupled from implementation details.

13.14 Further Reading

For complete server integration reference: - *Building with LARC* Chapter 13: Server Integration - Backend patterns and API design - *Building with LARC* Chapter 7: Data Fetching and APIs - HTTP and WebSocket patterns - *Building with LARC* Appendix E: Recipes and Patterns - Server integration recipes

14 Testing

Testing isn't optional. It's how you know your code works, how you prevent regressions, and how you confidently refactor. LARC applications benefit from the same testing strategies as any JavaScript application, with some patterns specific to Web Components.

14.1 Unit Testing Components

The `@open-wc/testing` library provides excellent utilities for testing Web Components:

```
// user-card.test.js
import { expect, fixture, html } from '@open-wc/testing';
import '../components/user-card.js';

describe('UserCard', () => {
  it('renders user name and email', async () => {
    const el = await fixture(html`
      <user-card></user-card>
    `);

    el.user = { name: 'Alice', email: 'alice@example.com' };
    await el.updateComplete;

    const name = el.shadowRoot.querySelector('.name');
    const email = el.shadowRoot.querySelector('.email');

    expect(name.textContent).to.equal('Alice');
    expect(email.textContent).to.equal('alice@example.com');
  });

  it('dispatches follow event when button clicked', async () => {
    const el = await fixture(html`<user-card></user-card>`);
    el.user = { id: 1, name: 'Alice' };

    let eventDetail = null;
    el.addEventListener('follow', (e) => {
      eventDetail = e.detail;
    });

    el.shadowRoot.querySelector('.follow-btn').click();

    expect(eventDetail).to.deep.equal({ userId: 1 });
  });
```

```
  it('shows loading state initially', async () => {
    const el = await fixture(html`<user-card loading></user-card>`);

    const spinner = el.shadowRoot.querySelector('.spinner');
    expect(spinner).to.exist;
  });
});
```

14.2 Testing PAN Bus Integration

Mock the PAN bus to test component communication:

```
// pan-mock.js
class MockPanBus {
  constructor() {
    this.messages = [];
    this.subscriptions = new Map();
  }

  publish(topic, data) {
    this.messages.push({ topic, data });
    const handlers = this.subscriptions.get(topic) || [];
    handlers.forEach(handler => handler(data));
  }

  subscribe(topic, handler) {
    if (!this.subscriptions.has(topic)) {
      this.subscriptions.set(topic, []);
    }
    this.subscriptions.get(topic).push(handler);
    return () => this.unsubscribe(topic, handler);
  }

  unsubscribe(topic, handler) {
    const handlers = this.subscriptions.get(topic) || [];
    const index = handlers.indexOf(handler);
    if (index > -1) handlers.splice(index, 1);
  }

  clear() {
    this.messages = [];
    this.subscriptions.clear();
  }
}

export const mockPan = new MockPanBus();
```

Use it in tests:

```
import { mockPan } from './pan-mock.js';

describe('NotificationList', () => {
  beforeEach(() => mockPan.clear());

  it('displays notifications from PAN bus', async () => {
    const el = await fixture(html`<notification-list></notification-list>`);

    mockPan.publish('notification.new', {
      id: 1,
      message: 'Hello world'
    });

    await el.updateComplete;

    const notifications = el.shadowRoot.querySelectorAll('.notification');
    expect(notifications.length).to.equal(1);
    expect(notifications[0].textContent).to.include('Hello world');
  });
});
```

14.3 Integration Testing

Test components working together:

```
describe('Shopping Cart Integration', () => {
  it('updates cart when product added', async () => {
    const cart = await fixture(html`<shopping-cart></shopping-cart>`);
    const product = await fixture(html`
      <product-card .product=${{ id: 1, name: 'Widget', price: 10 }}>
      </product-card>
    `);

    // Simulate add to cart
    product.shadowRoot.querySelector('.add-btn').click();

    await cart.updateComplete;

    expect(cart.items.length).to.equal(1);
    expect(cart.total).to.equal(10);
  });
});
```

14.4 End-to-End Testing with Playwright

For full user flow testing, Playwright provides excellent browser automation:

```
// e2e/login.spec.js
import { test, expect } from '@playwright/test';

test.describe('Login Flow', () => {
  test('successful login redirects to dashboard', async ({ page }) => {
    await page.goto('/login');

    await page.fill('input[name="email"]', 'user@example.com');
    await page.fill('input[name="password"]', 'password123');
    await page.click('button[type="submit"]');

    await expect(page).toHaveURL('/dashboard');
    await expect(page.locator('h1')).toHaveText('Dashboard');
  });

  test('invalid credentials show error', async ({ page }) => {
    await page.goto('/login');

    await page.fill('input[name="email"]', 'wrong@example.com');
    await page.fill('input[name="password"]', 'wrongpassword');
    await page.click('button[type="submit"]');

    await expect(page.locator('.error')).toHaveText('Invalid
        credentials');
    await expect(page).toHaveURL('/login');
  });
});
```

14.5 Mocking Fetch Requests

Control network responses in tests:

```
// fetch-mock.js
class FetchMock {
  constructor() {
    this.mocks = new Map();
    this.originalFetch = window.fetch;
  }

  mock(url, response, options = {}) {
    this.mocks.set(url, { response, options });
  }

  enable() {
    window.fetch = async (url, config) => {
      const mock = this.mocks.get(url);
      if (mock) {
        if (mock.options.delay) {
          await new Promise(r => setTimeout(r, mock.options.delay));
        }
        return new Response(JSON.stringify(mock.response), {
          status: mock.options.status || 200,
          headers: { 'Content-Type': 'application/json' }
        });
      }
      return this.originalFetch(url, config);
    };
  }

  disable() {
    window.fetch = this.originalFetch;
    this.mocks.clear();
  }
}

export const fetchMock = new FetchMock();
```

Use in tests:

```
import { fetchMock } from './fetch-mock.js';

describe('UserList Component', () => {
  beforeEach(() => {
    fetchMock.enable();
    fetchMock.mock('/api/users', [
      { id: 1, name: 'Alice' },
      { id: 2, name: 'Bob' }
    ]);
  });

  afterEach(() => {
    fetchMock.disable();
  });

  it('loads and displays users', async () => {
    const el = await fixture(html`<user-list></user-list>`);
    await el.updateComplete;

    const users = el.shadowRoot.querySelectorAll('.user');
    expect(users.length).to.equal(2);
  });
});
```

14.6 Visual Regression Testing

Catch visual changes with screenshot comparison:

```
// visual.test.js
import { test, expect } from '@playwright/test';

test.describe('Visual Regression', () => {
  test('button should match snapshot', async ({ page }) => {
    await page.goto('/components/button');

    const button = page.locator('app-button');
    await expect(button).toHaveScreenshot('button-default.png');
  });

  test('button hover state', async ({ page }) => {
    await page.goto('/components/button');

    const button = page.locator('app-button');
    await button.hover();
    await expect(button).toHaveScreenshot('button-hover.png');
  });

  test('dark mode theme', async ({ page }) => {
    await page.goto('/');
    await page.locator('[data-theme-toggle]').click();

    await expect(page).toHaveScreenshot('homepage-dark.png', {
      fullPage: true
    });
  });
});
```

Run visual tests:

```
# First run creates baseline screenshots
npx playwright test visual.test.js

# Subsequent runs compare against baseline
npx playwright test visual.test.js

# Update baselines when changes are intentional
npx playwright test visual.test.js --update-snapshots
```

14.7 Test Coverage

Track which code is tested:

```
// web-test-runner.config.js
export default {
  coverage: true,
  coverageConfig: {
    threshold: {
      statements: 80,
      branches: 75,
      functions: 80,
      lines: 80
    },
    include: ['src/**/*.js'],
    exclude: ['src/**/*.test.js', 'src/test/**']
  }
};
```

Generate coverage reports:

```
npx wtr --coverage

# View HTML report
open coverage/index.html
```

14.7.1 What to Test

Focus on:

- **Public API** - Methods users will call
- **Edge cases** - Empty inputs, null values, errors
- **State changes** - Component updates correctly
- **User interactions** - Clicks, typing, form submission
- **Integration points** - Component communication

Skip testing:

- **Framework internals** - Trust Web Components API
- **Third-party libraries** - They have their own tests
- **Trivial code** - Simple getters/setters

14.8 Component Testing Patterns

14.8.1 Testing Async Loading

```
it('shows loading state then data', async () => {
  const el = await fixture(html`<user-profile user-id="1"></user-profile>`);

  // Should show loading initially
  expect(el.shadowRoot.querySelector('.loading')).to.exist;

  // Wait for data
  await waitUntil(() => !el.loading);

  // Should show user data
  expect(el.shadowRoot.querySelector('.user-
        name')).to.have.text('Alice');
  expect(el.shadowRoot.querySelector('.loading')).to.not.exist;
});
```

14.8.2 Testing Error States

```
it('displays error message on fetch failure', async () => {
  fetchMock.mock('/api/users', { error: 'Server error' }, { status: 500
        });

  const el = await fixture(html`<user-list></user-list>`);
  await el.updateComplete;

  expect(el.shadowRoot.querySelector('.error')).to.have.text('Failed to
        load users');

});
```

14.8.3 Testing Form Validation

```
it('validates email format', async () => {
  const el = await fixture(html`<login-form></login-form>`);

  const emailInput = el.shadowRoot.querySelector('#email');
  const form = el.shadowRoot.querySelector('form');

  // Invalid email
  emailInput.value = 'notanemail';
  emailInput.dispatchEvent(new Event('input'));

  expect(el.errors.email).to.equal('Invalid email format');

  // Valid email
  emailInput.value = 'user@example.com';
  emailInput.dispatchEvent(new Event('input'));

  expect(el.errors.email).to.be.undefined;
});
```

14.8.4 Testing Component Communication

```
it('publishes event on button click', async () => {
  const el = await fixture(html`<add-todo></add-todo>`);

  let publishedData = null;
  mockPan.subscribe('todo.added', (msg) => {
    publishedData = msg.data;
  });

  el.shadowRoot.querySelector('#todo-input').value = 'Buy milk';
  el.shadowRoot.querySelector('button').click();

  expect(publishedData).to.deep.equal({ text: 'Buy milk' });
});
```

14.9 CI/CD Integration

14.9.1 GitHub Actions

```yaml
# .github/workflows/test.yml
name: Tests

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '18'
          cache: 'npm'

      - name: Install dependencies
        run: npm ci

      - name: Run unit tests
        run: npm test

      - name: Run E2E tests
        run: npx playwright test

      - name: Upload test results
        if: failure()
        uses: actions/upload-artifact@v3
        with:
          name: test-results
          path: test-results/
```

```yaml
      - name: Upload coverage
        uses: codecov/codecov-action@v3
        with:
          files: ./coverage/lcov.info
```

14.9.2 GitLab CI

```yaml
# .gitlab-ci.yml
stages:
  - test
  - report

test:
  stage: test
  image: node:18
  cache:
    paths:
      - node_modules/
  script:
    - npm ci
    - npm test
    - npx playwright test
  artifacts:
    when: always
    paths:
      - coverage/
      - test-results/
    reports:
      coverage_report:
        coverage_format: cobertura
        path: coverage/cobertura-coverage.xml

coverage:
  stage: report
  image: node:18
  script:
    - npx nyc report --reporter=text-summary
  coverage: '/Lines\s*:\s*(\d+\.\d+)%/'
```

14.10 Test-Driven Development (TDD)

Write tests first, then implement:

14.10.1 Example: Building a Counter Component

Step 1: Write the test

```
// counter.test.js
describe('Counter', () => {
  it('starts at zero', async () => {
    const counter = await fixture(html`<app-counter></app-counter>`);
    expect(counter.shadowRoot.querySelector('.count').textContent).to.equal('0');

  });

  it('increments when plus button clicked', async () => {
    const counter = await fixture(html`<app-counter></app-counter>`);

    counter.shadowRoot.querySelector('.plus-btn').click();
    await counter.updateComplete;

    expect(counter.shadowRoot.querySelector('.count').textContent).to.equal('1');

  });

  it('decrements when minus button clicked', async () => {
    const counter = await fixture(html`<app-counter></app-counter>`);

    counter.shadowRoot.querySelector('.plus-btn').click();
    await counter.updateComplete;

    counter.shadowRoot.querySelector('.minus-btn').click();
    await counter.updateComplete;

    expect(counter.shadowRoot.querySelector('.count').textContent).to.equal('0');

  });

  it('never goes below zero', async () => {
    const counter = await fixture(html`<app-counter></app-counter>`);
```

```
    counter.shadowRoot.querySelector('.minus-btn').click();
    await counter.updateComplete;

    expect(counter.shadowRoot.querySelector('.count').textContent).to.equal('0');

  });
});
```

Step 2: Run tests (they fail)

```
npx wtr
# All tests fail - component doesn't exist yet
```

Step 3: Implement minimal code

```
// counter.js
class Counter extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.count = 0;
  }

  connectedCallback() {
    this.render();
  }

  increment() {
    this.count++;
    this.render();
  }

  decrement() {
    if (this.count > 0) {
      this.count--;
      this.render();
    }
  }

  render() {
    this.shadowRoot.innerHTML = `
      <div>
        <button class="minus-btn">-</button>
        <span class="count">${this.count}</span>
        <button class="plus-btn">+</button>
      </div>
    `;
```

```
    this.shadowRoot.querySelector('.plus-btn').addEventListener('click',
        () => this.increment());
    this.shadowRoot.querySelector('.minus-
        btn').addEventListener('click', () => this.decrement());
  }

  get updateComplete() {
    return Promise.resolve();
  }
}

customElements.define('app-counter', Counter);
```

Step 4: Run tests (they pass)

```
npx wtr
# All tests pass!
```

Step 5: Refactor with confidence

Tests ensure refactoring doesn't break functionality.

14.11 Advanced Testing Patterns

14.11.1 Testing Custom Events

```
it('dispatches custom event with detail', async () => {
  const el = await fixture(html`<product-card></product-card>`);

  let eventDetail = null;
  el.addEventListener('add-to-cart', (e) => {
    eventDetail = e.detail;
  });

  el.shadowRoot.querySelector('.add-btn').click();

  expect(eventDetail).to.deep.equal({
    productId: 123,
    quantity: 1
  });
});
```

14.11.2 Testing Slots

```
it('renders slotted content', async () => {
  const el = await fixture(html`
    <card-component>
      <h2 slot="title">My Title</h2>
      <p>My content</p>
    </card-component>
  `);

  const title = el.shadowRoot.querySelector('slot[name="title"]');
  const assignedNodes = title.assignedNodes();

  expect(assignedNodes[0].textContent).to.equal('My Title');
});
```

14.11.3 Testing Accessibility

```
import { expect } from '@open-wc/testing';

it('is accessible', async () => {
  const el = await fixture(html`<my-button>Click me</my-button>`);

  await expect(el).to.be.accessible();
});

it('has correct ARIA attributes', async () => {
  const el = await fixture(html`<dialog-box></dialog-box>`);

  const dialog = el.shadowRoot.querySelector('[role="dialog"]');
  expect(dialog).to.have.attribute('aria-modal', 'true');
  expect(dialog).to.have.attribute('aria-labelledby');
});
```

14.11.4 Testing Keyboard Navigation

```
it('navigates with arrow keys', async () => {
  const el = await fixture(html`
    <tab-panel>
      <tab-item>Tab 1</tab-item>
      <tab-item>Tab 2</tab-item>
      <tab-item>Tab 3</tab-item>
    </tab-panel>
  `);

  const tabs = el.shadowRoot.querySelectorAll('tab-item');

  // Focus first tab
  tabs[0].focus();

  // Press arrow right
  tabs[0].dispatchEvent(new KeyboardEvent('keydown', { key: 'ArrowRight'
        }));

  expect(document.activeElement).to.equal(tabs[1]);
});
```

14.12 Performance Testing

Test component performance:

```
it('renders large list efficiently', async () => {
  const items = Array.from({ length: 10000 }, (_, i) => ({
    id: i,
    name: `Item ${i}`
  }));

  const startTime = performance.now();

  const el = await fixture(html`<virtual-list .items=${items}></virtual-
        list>`);
  await el.updateComplete;

  const renderTime = performance.now() - startTime;

  // Should render in under 100ms
  expect(renderTime).to.be.lessThan(100);

  // Should only render visible items
  const renderedItems = el.shadowRoot.querySelectorAll('.item');
  expect(renderedItems.length).to.be.lessThan(50);
});
```

14.13 Troubleshooting Tests

14.13.1 Problem: Tests Pass Locally but Fail in CI

Symptom: Tests work on your machine but fail in GitHub Actions

Solution: Common issues:

```
// 1. Timing issues - add proper waits
await waitUntil(() => el.shadowRoot.querySelector('.data'));

// 2. Browser differences - use consistent browser
// playwright.config.js
projects: [
  { name: 'chromium', use: { ...devices['Desktop Chrome'] } }
]

// 3. Port conflicts - use random ports
const port = Math.floor(Math.random() * 10000) + 50000;
```

14.13.2 Problem: Flaky Tests

Symptom: Tests sometimes pass, sometimes fail

Solution:

```
// Bad: Fixed timeout
await new Promise(resolve => setTimeout(resolve, 1000));

// Good: Wait for condition
await waitUntil(() => el.dataLoaded);

// Bad: Race condition
el.loadData();
expect(el.data).to.exist;

// Good: Wait for async
await el.loadData();
expect(el.data).to.exist;
```

14.13.3 Problem: Slow Tests

Symptom: Test suite takes too long

Solution:

```
// 1. Run tests in parallel
// web-test-runner.config.js
export default {
  concurrency: 10,
  nodeResolve: true
};

// 2. Mock expensive operations
beforeEach(() => {
  fetchMock.mock('/api/expensive', cachedData);
});

// 3. Skip browser for pure logic
describe('pure functions', () => {
  it('calculates correctly', () => {
    expect(calculateTotal([1, 2, 3])).to.equal(6);
  });
});
```

14.14 Best Practices

1. **Test behavior, not implementation** - Test what users see, not internal details
2. **Keep tests focused** - One assertion per test when possible
3. **Use descriptive names** - Test names should document behavior
4. **Avoid test interdependence** - Each test should run independently
5. **Mock external dependencies** - Don't rely on real APIs
6. **Test error paths** - Test failures, not just successes

7. **Run tests often** - Catch bugs early
8. **Maintain test code** - Refactor tests like production code
9. **Use setup/teardown** - DRY principle applies to tests
10. **Achieve good coverage** - Aim for 80%+, but focus on critical paths

14.15 Exercises

14.15.1 Exercise 1: Test a Todo Component

Write comprehensive tests for a todo component:

- Renders list of todos
- Adds new todo when form submitted
- Toggles complete status on click
- Deletes todo when delete button clicked
- Shows empty state when no todos
- Validates input before adding

Bonus: Test keyboard shortcuts (Enter to submit, Escape to clear).

14.15.2 Exercise 2: E2E Shopping Flow

Create E2E tests for shopping cart:

- Browse products
- Add items to cart
- Update quantities
- Apply coupon code
- Complete checkout
- Verify order confirmation

Bonus: Test as guest and authenticated user.

14.15.3 Exercise 3: Visual Regression Suite

Set up visual regression testing:

- Capture screenshots of all components
- Test different states (hover, focus, disabled)
- Test responsive breakpoints
- Test dark/light themes
- Integrate into CI pipeline

Bonus: Add Percy or Chromatic for cloud-based visual testing.

14.15.4 Exercise 4: TDD Calculator

Build a calculator component using TDD:

- Write tests first for basic operations (+, -, ×, ÷)
- Implement each operation one at a time
- Add tests for edge cases (divide by zero, overflow)
- Add memory functions (MC, MR, M+, M-)
- Keep tests passing throughout

Bonus: Add scientific functions (sin, cos, sqrt).

14.16 Summary

Testing ensures your LARC applications work correctly and continue working as you make changes:

- **Unit tests** verify individual components in isolation
- **Integration tests** verify components work together
- **E2E tests** verify complete user workflows
- **Visual regression tests** catch unintended visual changes
- **Test coverage** ensures critical code is tested

- **CI/CD integration** catches problems before deployment
- **TDD** helps design better components

Good tests give you confidence to refactor, add features, and deploy. They're not overhead—they're insurance that saves time debugging production issues.

14.17 Further Reading

For complete testing reference: - *Building with LARC* Chapter 13: Testing Strategies - All testing patterns and tools - *Building with LARC* Appendix E: Recipes and Patterns - Testing recipes and examples - @open-wc/testing documentation - Playwright documentation

15 Performance and Optimization

Performance is a feature. Slow applications frustrate users and hurt business metrics. LARC's no-build philosophy gives you a head start—no framework overhead, no transpilation artifacts—but there's more you can do.

15.1 Lazy Loading Components

Don't load everything upfront. Load components when needed:

```
// Lazy load on route change
pan.subscribe('router.navigate', async ({ path }) => {
  if (path === '/admin') {
    await import('./components/admin-panel.js');
  }
});

// Lazy load on user interaction
document.querySelector('.show-chart').addEventListener('click', async ()
        => {
  const { ChartComponent } = await import('./components/chart.js');
  // Use component
}, { once: true });

// Lazy load when visible
const observer = new IntersectionObserver(async (entries) => {
  for (const entry of entries) {
    if (entry.isIntersecting) {
      const component = entry.target.dataset.component;
      await import(`./components/${component}.js`);
      observer.unobserve(entry.target);
    }
  }
});

document.querySelectorAll('[data-lazy]').forEach(el =>
        observer.observe(el));
```

15.2 Image Optimization

Images are often the largest assets. Optimize them:

```
class LazyImage extends HTMLElement {
  connectedCallback() {
    this.innerHTML = `
      <img
        loading="lazy"
        src="${this.getAttribute('placeholder') || 'placeholder.svg'}"
        data-src="${this.getAttribute('src')}"
        alt="${this.getAttribute('alt') || ''}"
      >
    `;

    const img = this.querySelector('img');

    const observer = new IntersectionObserver((entries) => {
      entries.forEach(entry => {
        if (entry.isIntersecting) {
          img.src = img.dataset.src;
          observer.unobserve(img);
        }
      });
    });

    observer.observe(img);
  }
}

customElements.define('lazy-image', LazyImage);
```

Use responsive images with srcset:

```
<img
  srcset="image-400.jpg 400w,
          image-800.jpg 800w,
          image-1200.jpg 1200w"
  sizes="(max-width: 400px) 400px,
         (max-width: 800px) 800px,
         1200px"
  src="image-800.jpg"
  alt="Responsive image"
  loading="lazy"
>
```

15.3 Service Worker Caching

Service workers enable offline functionality and faster loads:

```
// sw.js
const CACHE_NAME = 'app-v1';
const ASSETS = [
  '/',
  '/index.html',
  '/styles.css',
  '/app.js',
  '/components/header.js',
  '/components/footer.js'
];

self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME).then(cache => cache.addAll(ASSETS))
  );
});

self.addEventListener('fetch', (event) => {
  event.respondWith(
    caches.match(event.request).then(cached => {
      // Cache first, network fallback
      if (cached) return cached;

      return fetch(event.request).then(response => {
        // Cache successful responses
        if (response.ok) {
          const clone = response.clone();
          caches.open(CACHE_NAME).then(cache => {
            cache.put(event.request, clone);
          });
        }
        return response;
      });
    })
```

```
  );
});
```

Register it in your app:

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/sw.js');
}
```

15.4 Measuring Performance

You can't optimize what you don't measure. Use the Performance API:

```
// Measure component render time
performance.mark('render-start');
this.render();
performance.mark('render-end');
performance.measure('render', 'render-start', 'render-end');

const measure = performance.getEntriesByName('render')[0];
console.log(`Render took ${measure.duration}ms`);
```

Track Web Vitals:

```
import { getCLS, getFID, getLCP } from 'web-vitals';

getCLS(console.log);  // Cumulative Layout Shift
getFID(console.log);  // First Input Delay
getLCP(console.log);  // Largest Contentful Paint
```

15.5 Virtual Lists for Large Data

Rendering thousands of items kills performance. Virtualize:

```
class VirtualList extends HTMLElement {
  constructor() {
    super();
    this.items = [];
    this.itemHeight = 40;
    this.visibleCount = 20;
    this.scrollTop = 0;
  }

  set data(items) {
    this.items = items;
    this.render();
  }

  connectedCallback() {
    this.style.cssText = `
      display: block;
      height: 400px;
      overflow-y: auto;
    `;

    this.addEventListener('scroll', () => {
      this.scrollTop = this.scrollTop;
      this.render();
    });

    this.render();
  }

  render() {
    const startIndex = Math.floor(this.scrollTop / this.itemHeight);
    const visibleItems = this.items.slice(startIndex, startIndex +
        this.visibleCount);
```

```
    this.innerHTML = `
      <div style="height: ${this.items.length * this.itemHeight}px;
        position: relative;">
        ${visibleItems.map((item, i) => `
          <div style="
            position: absolute;
            top: ${(startIndex + i) * this.itemHeight}px;
            height: ${this.itemHeight}px;
            width: 100%;
          ">
            ${item.name}
          </div>
        `).join('')}
      </div>
    `;
  }
}

customElements.define('virtual-list', VirtualList);
```

15.6 Code Splitting and Dynamic Imports

Break your code into smaller chunks that load on demand:

```
// Route-based code splitting
class AppRouter extends HTMLElement {
  constructor() {
    super();
    this.routes = new Map([
      ['/', () => import('./pages/home.js')],
      ['/products', () => import('./pages/products.js')],
      ['/admin', () => import('./pages/admin.js')]
    ]);
  }

  async navigate(path) {
    // Show loading state
    this.innerHTML = '<div class="loading">Loading...</div>';

    try {
      const loader = this.routes.get(path);
      if (!loader) {
        throw new Error('Route not found');
      }

      // Load the module
      const module = await loader();

      // Render the page component
      this.innerHTML = `<${module.tagName}></${module.tagName}>`;

      // Track page load time
      performance.mark(`page-${path}-loaded`);
    } catch (error) {
      this.innerHTML = `<error-page message="${error.message}"></error-
        page>`;
    }
  }
```

```
}

// Feature-based code splitting
class DataGrid extends HTMLElement {
  async enableExport() {
    if (!this.exportModule) {
      // Only load export library when user needs it
      this.exportModule = await import('https://cdn.jsdelivr.net/npm/
        xlsx/+esm');
    }

    const worksheet = this.exportModule.utils.json_to_sheet(this.data);
    const workbook = this.exportModule.utils.book_new();
    this.exportModule.utils.book_append_sheet(workbook, worksheet,
        'Data');
    this.exportModule.writeFile(workbook, 'export.xlsx');
  }
}
```

15.7 Debouncing and Throttling

Control how often expensive operations run:

```
// Debounce: Wait for user to stop typing
function debounce(fn, delay = 300) {
  let timeoutId;
  return function (...args) {
    clearTimeout(timeoutId);
    timeoutId = setTimeout(() => fn.apply(this, args), delay);
  };
}

class SearchBox extends HTMLElement {
  connectedCallback() {
    this.innerHTML = `
      <input type="text" placeholder="Search...">
      <div class="results"></div>
    `;

    const input = this.querySelector('input');
    const results = this.querySelector('.results');

    // Debounce search API calls
    const searchDebounced = debounce(async (query) => {
      if (query.length < 2) {
        results.innerHTML = '';
        return;
      }

      results.innerHTML = 'Searching...';

      const data = await fetch(`/api/
        search?q=${encodeURIComponent(query)}`)
        .then(r => r.json());

      results.innerHTML = data.map(item =>
        `<div class="result">${item.title}</div>`
```

```
      ).join('');
    }, 300);

    input.addEventListener('input', (e) => {
      searchDebounced(e.target.value);
    });
  }
}

// Throttle: Limit scroll handler frequency
function throttle(fn, delay = 100) {
  let lastCall = 0;
  return function (...args) {
    const now = Date.now();
    if (now - lastCall >= delay) {
      lastCall = now;
      fn.apply(this, args);
    }
  };
}

class InfiniteScroll extends HTMLElement {
  connectedCallback() {
    const loadMore = throttle(() => {
      const scrollBottom = this.scrollTop + this.clientHeight;
      const threshold = this.scrollHeight - 200;

      if (scrollBottom >= threshold && !this.loading) {
        this.loadNextPage();
      }
    }, 200);

    this.addEventListener('scroll', loadMore);
  }
```

```
  async loadNextPage() {
    this.loading = true;
    // Load more items...
    this.loading = false;
  }
}
```

15.8 Memoization for Expensive Computations

Cache computed values to avoid redundant work:

```
class DataTable extends HTMLElement {
  constructor() {
    super();
    this.cache = new Map();
  }

  // Memoize expensive sort operation
  getSortedData(data, sortKey, direction) {
    const cacheKey = `${sortKey}-${direction}`;

    if (this.cache.has(cacheKey)) {
      console.log('Using cached sort');
      return this.cache.get(cacheKey);
    }

    console.log('Computing sort');
    const sorted = [...data].sort((a, b) => {
      const aVal = a[sortKey];
      const bVal = b[sortKey];
      const multiplier = direction === 'asc' ? 1 : -1;
      return aVal < bVal ? -multiplier : aVal > bVal ? multiplier : 0;
    });

    this.cache.set(cacheKey, sorted);
    return sorted;
  }

  // Clear cache when data changes
  set data(newData) {
    this._data = newData;
    this.cache.clear();
    this.render();
  }
}
```

```
// Memoize with WeakMap for object keys
const memoizedCalculations = new WeakMap();

function expensiveCalculation(obj) {
  if (memoizedCalculations.has(obj)) {
    return memoizedCalculations.get(obj);
  }

  const result = {
    total: obj.items.reduce((sum, item) => sum + item.price, 0),
    tax: obj.items.reduce((sum, item) => sum + item.price * 0.1, 0),
    // ... more expensive calculations
  };

  memoizedCalculations.set(obj, result);
  return result;
}
```

15.9 Bundle Size Optimization

Keep your JavaScript small:

```
// Use import maps to share dependencies
// In your HTML:
/*
<script type="importmap">
{
  "imports": {
    "lit": "https://cdn.jsdelivr.net/npm/lit@3/+esm",
    "lit/": "https://cdn.jsdelivr.net/npm/lit@3/"
  }
}
</script>
*/

// Multiple components can share the same lit import
import { LitElement, html, css } from 'lit';

// Tree-shake unused code by importing only what you need
// ❌ Bad: imports everything
import * as utils from './utils.js';

// ✅ Good: imports only what's needed
import { formatDate, formatCurrency } from './utils.js';

// Prefer native APIs over libraries
// ❌ Heavy date library (40KB+)
import dayjs from 'dayjs';
const formatted = dayjs(date).format('YYYY-MM-DD');

// ✅ Native Intl (0KB)
const formatted = new Intl.DateTimeFormat('en-US').format(date);

// Use dynamic imports for conditional features
if (user.isAdmin) {
  const { AdminPanel } = await import('./admin.js');
```

```
  // Use AdminPanel
}
```

Check your bundle size:

```
# Analyze what's being loaded
ls -lh dist/*.js

# Use browser DevTools Network tab to see:
# - Total KB transferred
# - Uncompressed size
# - Number of requests
```

15.10 Web Vitals Monitoring

Monitor real user experience:

```
// web-vitals-tracker.js
class WebVitalsTracker {
  constructor() {
    this.metrics = {};
  }

  async track() {
    // Import web-vitals library only when needed
    const { onCLS, onFID, onLCP, onFCP, onTTFB } = await import(
      'https://cdn.jsdelivr.net/npm/web-vitals@3/+esm'
    );

    onCLS((metric) => this.reportMetric(metric));
    onFID((metric) => this.reportMetric(metric));
    onLCP((metric) => this.reportMetric(metric));
    onFCP((metric) => this.reportMetric(metric));
    onTTFB((metric) => this.reportMetric(metric));
  }

  reportMetric(metric) {
    this.metrics[metric.name] = metric.value;

    // Send to analytics
    if (navigator.sendBeacon) {
      navigator.sendBeacon('/analytics', JSON.stringify({
        metric: metric.name,
        value: metric.value,
        rating: metric.rating,
        page: window.location.pathname
      }));
    }

    // Log for development
    console.log(`${metric.name}: ${metric.value} (${metric.rating})`);
```

```
  }

  getScores() {
    return {
      cls: this.metrics.CLS || 0,
      fid: this.metrics.FID || 0,
      lcp: this.metrics.LCP || 0,
      fcp: this.metrics.FCP || 0,
      ttfb: this.metrics.TTFB || 0
    };
  }
}

// Use in your app
const vitals = new WebVitalsTracker();
vitals.track();
```

Display performance scores to users:

```
class PerformanceWidget extends HTMLElement {
  async connectedCallback() {
    const { onLCP, onFID, onCLS } = await import(
      'https://cdn.jsdelivr.net/npm/web-vitals@3/+esm'
    );

    this.innerHTML = `
      <div class="vitals">
        <div class="metric">
          <span class="label">LCP</span>
          <span class="value lcp">...</span>
        </div>
        <div class="metric">
          <span class="label">FID</span>
          <span class="value fid">...</span>
        </div>
        <div class="metric">
          <span class="label">CLS</span>
          <span class="value cls">...</span>
        </div>
      </div>
    `;

    onLCP(({ value, rating }) => {
      this.querySelector('.lcp').textContent = `${Math.round(value)}ms`;
      this.querySelector('.lcp').className = `value lcp ${rating}`;
    });

    onFID(({ value, rating }) => {
      this.querySelector('.fid').textContent = `${Math.round(value)}ms`;
      this.querySelector('.fid').className = `value fid ${rating}`;
    });

    onCLS(({ value, rating }) => {
```

```
      this.querySelector('.cls').textContent = value.toFixed(3);
      this.querySelector('.cls').className = `value cls ${rating}`;
    });
  }
}

customElements.define('performance-widget', PerformanceWidget);
```

15.11 Real-World Example: Optimized Dashboard

Here's a complete dashboard with all optimization techniques applied:

```
class OptimizedDashboard extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.cache = new Map();
    this.loadedWidgets = new Set();
  }

  async connectedCallback() {
    // 1. Render shell immediately (FCP)
    this.renderShell();

    // 2. Load critical data
    await this.loadCriticalData();

    // 3. Set up lazy loading for below-fold widgets
    this.setupLazyLoading();

    // 4. Track performance
    this.trackPerformance();
  }

  renderShell() {
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: block;
          container-type: inline-size;
        }

        .grid {
          display: grid;
          grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
          gap: 1rem;
```

```
      padding: 1rem;
    }

    .widget {
      background: white;
      border-radius: 8px;
      padding: 1rem;
      min-height: 200px;
    }

    .skeleton {
      background: linear-gradient(90deg, #f0f0f0 25%, #e0e0e0 50%,
    #f0f0f0 75%);
      background-size: 200% 100%;
      animation: shimmer 1.5s infinite;
    }

    @keyframes shimmer {
      0% { background-position: 200% 0; }
      100% { background-position: -200% 0; }
    }
  </style>

  <div class="grid">
    <div class="widget" data-widget="revenue">
      <div class="skeleton"></div>
    </div>
    <div class="widget" data-widget="users">
      <div class="skeleton"></div>
    </div>
    <div class="widget" data-widget="chart" data-lazy>
      <div class="skeleton"></div>
    </div>
    <div class="widget" data-widget="table" data-lazy>
      <div class="skeleton"></div>
```

```
        </div>
      </div>
    `;
  }

  async loadCriticalData() {
    // Load above-the-fold widgets in parallel
    const criticalWidgets = ['revenue', 'users'];

    await Promise.all(
      criticalWidgets.map(widget => this.loadWidget(widget))
    );
  }

  setupLazyLoading() {
    const lazyWidgets = this.shadowRoot.querySelectorAll('[data-lazy]');

    const observer = new IntersectionObserver(
      (entries) => {
        entries.forEach(entry => {
          if (entry.isIntersecting) {
            const widgetName = entry.target.dataset.widget;
            this.loadWidget(widgetName);
            observer.unobserve(entry.target);
          }
        });
      },
      { rootMargin: '50px' }
    );

    lazyWidgets.forEach(widget => observer.observe(widget));
  }

  async loadWidget(name) {
```

```
    if (this.loadedWidgets.has(name)) return;

    performance.mark(`widget-${name}-start`);

    try {
      // Check cache first
      let data = this.cache.get(name);

      if (!data) {
        // Load data with timeout
        const controller = new AbortController();
        const timeoutId = setTimeout(() => controller.abort(), 5000);

        data = await fetch(`/api/widgets/${name}`, {
          signal: controller.signal
        }).then(r => r.json());

        clearTimeout(timeoutId);

        // Cache for 5 minutes
        this.cache.set(name, data);
        setTimeout(() => this.cache.delete(name), 5 * 60 * 1000);
      }

      // Render widget
      const widget = this.shadowRoot.querySelector(`[data-
        widget="${name}"]`);
      widget.innerHTML = this.renderWidget(name, data);

      this.loadedWidgets.add(name);

      performance.mark(`widget-${name}-end`);
      performance.measure(
        `widget-${name}`,
        `widget-${name}-start`,
```

```
        `widget-${name}-end`
      );
    } catch (error) {
      console.error(`Failed to load widget ${name}:`, error);

      const widget = this.shadowRoot.querySelector(`[data-
        widget="${name}"]`);
      widget.innerHTML = `
        <div class="error">
          <p>Failed to load ${name}</p>
          <button
        onclick="this.getRootNode().host.loadWidget('${name}')">
            Retry
          </button>
        </div>
      `;
    }
  }

  renderWidget(name, data) {
    switch (name) {
      case 'revenue':
        return `
          <h3>Revenue</h3>
          <div class="value">$${data.total.toLocaleString()}</div>
          <div class="change ${data.change >= 0 ? 'positive' :
        'negative'}">
            ${data.change >= 0 ? '↑' : '↓'} ${Math.abs(data.change)}%
          </div>
        `;

      case 'users':
        return `
          <h3>Active Users</h3>
          <div class="value">${data.count.toLocaleString()}</div>
        `;
```

```
    case 'chart':
      // Lazy load chart library only when needed
      return `<canvas id="chart-${name}"></canvas>`;

    case 'table':
      return `
        <h3>Recent Activity</h3>
        <virtual-list></virtual-list>
      `;

    default:
      return `<div>Unknown widget: ${name}</div>`;
  }
}

trackPerformance() {
  // Track load time
  window.addEventListener('load', () => {
    const loadTime = performance.timing.loadEventEnd -
                     performance.timing.navigationStart;
    console.log(`Dashboard loaded in ${loadTime}ms`);
  });

  // Track widget render times
  const observer = new PerformanceObserver((list) => {
    list.getEntries().forEach((entry) => {
      if (entry.name.startsWith('widget-')) {
        console.log(`${entry.name}: ${entry.duration}ms`);
      }
    });
  });

  observer.observe({ entryTypes: ['measure'] });
```

```
  }
}

customElements.define('optimized-dashboard', OptimizedDashboard);
```

15.12 Troubleshooting Performance Issues

15.12.1 Problem 1: Memory Leaks

Symptoms: Page gets slower over time, browser tab uses increasing memory.

Common causes:

- Event listeners not removed
- Setters/intervals not cleared
- Large objects cached indefinitely

Solution:

```
class LeakyComponent extends HTMLElement {
  connectedCallback() {
    // ❌ Memory leak: handler never removed
    window.addEventListener('resize', this.onResize);

    // ❌ Memory leak: interval never cleared
    this.intervalId = setInterval(() => this.update(), 1000);

    // ❌ Memory leak: cache grows forever
    this.cache = new Map();
  }
}

class FixedComponent extends HTMLElement {
  connectedCallback() {
    // ✅ Store handler reference
    this.onResize = () => this.handleResize();
    window.addEventListener('resize', this.onResize);

    // ✅ Store interval ID
    this.intervalId = setInterval(() => this.update(), 1000);

    // ✅ Use LRU cache with size limit
    this.cache = new Map();
    this.maxCacheSize = 100;
  }

  disconnectedCallback() {
    // ✅ Clean up listener
    window.removeEventListener('resize', this.onResize);

    // ✅ Clear interval
    clearInterval(this.intervalId);
```

```
    // ✅ Clear cache
    this.cache.clear();
  }

  addToCache(key, value) {
    if (this.cache.size >= this.maxCacheSize) {
      // Remove oldest entry
      const firstKey = this.cache.keys().next().value;
      this.cache.delete(firstKey);
    }
    this.cache.set(key, value);
  }
}
```

Use browser DevTools to detect leaks: 1. Open Performance Monitor (Cmd/Ctrl + Shift + P → “Performance Monitor”) 2. Watch JS heap size over time 3. Take heap snapshots to find retained objects

15.12.2 Problem 2: Slow Initial Render

Symptoms: Long time before page shows content, poor LCP score.

Common causes:

- Loading too much JavaScript upfront
- Synchronous data fetching
- Rendering everything at once

Solution:

```
// ❌ Bad: Wait for everything
class SlowApp extends HTMLElement {
  async connectedCallback() {
    const data = await fetch('/api/data').then(r => r.json());
    this.render(data);
  }
}

// ✅ Good: Progressive rendering
class FastApp extends HTMLElement {
  connectedCallback() {
    // 1. Show shell immediately
    this.innerHTML = '<div class="shell">Loading...</div>';

    // 2. Load data asynchronously
    this.loadData();
  }

  async loadData() {
    try {
      const data = await fetch('/api/data').then(r => r.json());
      this.render(data);
    } catch (error) {
      this.renderError(error);
    }
  }
}
```

15.12.3 Problem 3: Large Bundle Size

Symptoms: Slow initial load, poor First Contentful Paint.

Diagnosis:

```
// Analyze what's in your bundle
console.table(
  performance.getEntriesByType('resource')
    .filter(r => r.initiatorType === 'script')
    .map(r => ({
      name: r.name.split('/').pop(),
      size: `${(r.transferSize / 1024).toFixed(2)} KB`,
      time: `${r.duration.toFixed(2)}ms`
    }))
);
```

Solutions:

- Use import maps to share dependencies
- Lazy load non-critical features
- Use native APIs instead of libraries
- Tree-shake unused code

15.12.4 Problem 4: Layout Thrashing

Symptoms: Janky scrolling, slow animations, poor FPS.

Cause: Reading and writing DOM in the same frame.

```
// ❌ Bad: Forces multiple reflows
items.forEach(item => {
  const height = item.offsetHeight;  // Read (reflow)
  item.style.height = height * 2 + 'px';  // Write (reflow)
});

// ✅ Good: Batch reads and writes
const heights = items.map(item => item.offsetHeight);  // Batch reads
items.forEach((item, i) => {
  item.style.height = heights[i] * 2 + 'px';  // Batch writes
});

// ✅ Better: Use requestAnimationFrame
function updateLayout() {
  // All reads first
  const measurements = elements.map(el => ({
    width: el.offsetWidth,
    height: el.offsetHeight
  }));

  // Then all writes
  elements.forEach((el, i) => {
    el.style.width = measurements[i].width * 2 + 'px';
    el.style.height = measurements[i].height * 2 + 'px';
  });
}

requestAnimationFrame(updateLayout);
```

15.13 Performance Best Practices

1. **Load Critical Resources First**: Prioritize above-the-fold content and user-interactive

elements.

2. **Lazy Load Everything Else**: Use Intersection Observer for images, components, and heavy features.
3. **Cache Aggressively**: Use service workers, HTTP caching, and in-memory caches appropriately.
4. **Measure Real Users**: Track Web Vitals for actual user experiences, not just lab tests.
5. **Debounce User Input**: Don't make API calls on every keystroke—wait for users to finish typing.
6. **Virtualize Long Lists**: Never render more than ~50-100 items at once.
7. **Code Split by Route**: Load only the JavaScript needed for the current page.
8. **Optimize Images**: Use modern formats (WebP, AVIF), lazy loading, and responsive images.
9. **Clean Up Resources**: Always remove event listeners and clear intervals in `disconnectedCallback`.
10. **Profile Before Optimizing**: Use DevTools to find actual bottlenecks—don't guess.

15.14 Hands-On Exercises

15.14.1 Exercise 1: Optimize an Image Gallery

Create an image gallery that:

- Lazy loads images as they scroll into view
- Uses Intersection Observer
- Shows a loading placeholder
- Tracks LCP for the first visible image

Bonus: Add a “Load All” button that prefetches remaining images.

15.14.2 Exercise 2: Build a Virtual List

Implement a virtual list component that:

- Renders only visible items
- Handles variable-height items
- Supports smooth scrolling
- Works with 10,000+ items

Bonus: Add keyboard navigation and accessibility.

15.14.3 Exercise 3: Implement Request Deduplication

Create a data service that:

- Prevents duplicate API calls for the same resource
- Shares pending requests between components
- Caches responses for 1 minute
- Provides a cache invalidation API

Bonus: Add optimistic updates with rollback on error.

15.14.4 Exercise 4: Performance Dashboard

Build a performance monitoring dashboard that:

- Tracks all Core Web Vitals
- Shows performance over time
- Highlights performance regressions
- Exports data to CSV

Bonus: Add alerts when metrics exceed thresholds.

15.15 Summary

Performance optimization is about making smart tradeoffs:

- **Load less**: Code split, lazy load, tree shake
- **Cache more**: Service workers, HTTP cache, memory cache
- **Render efficiently**: Virtual lists, debouncing, memoization
- **Measure everything**: Web Vitals, Performance API, DevTools

Start with the low-hanging fruit (lazy loading, caching) and use DevTools to find the real bottlenecks. Remember: premature optimization is the root of all evil—measure first, then optimize.

15.16 Further Reading

- **Building with LARC - Chapter 17 (Performance)**: Deep dive into LARC-specific optimization techniques
- **Building with LARC - Chapter 8 (Lifecycle)**: Component cleanup and resource management
- **Building with LARC - Chapter 11 (Best Practices)**: Performance patterns and anti-patterns

16 Deployment

Deploying LARC applications is refreshingly simple. No build artifacts to manage, no complex CI/CD pipelines required. Just static files that any web server can handle.

16.1 Static Hosting Options

LARC apps are static files. Host them anywhere:

16.1.1 GitHub Pages

Free hosting for public repositories:

```yaml
# .github/workflows/deploy.yml
name: Deploy to GitHub Pages

on:
  push:
    branches: [main]

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: peaceiris/actions-gh-pages@v3
        with:
          github_token: ${{ secrets.GITHUB_TOKEN }}
          publish_dir: ./public
```

16.1.2 Netlify

Drag and drop deployment or connect to Git:

```
# netlify.toml
[build]
  publish = "public"

[[redirects]]
  from = "/*"
  to = "/index.html"
  status = 200
```

16.1.3 Vercel

Zero-config deployment:

```
{
  "rewrites": [
    { "source": "/(.*)", "destination": "/index.html" }
  ]
}
```

16.2 CDN Configuration

Serve assets from a CDN for faster global delivery:

```
<!-- Use CDN for LARC core -->
<script type="importmap">
{
  "imports": {
    "@aspect/pan-client": "https://cdn.jsdelivr.net/npm/@aspect/pan-
        client@latest/pan-client.mjs"
  }
}
</script>
```

Set proper cache headers:

```
# .htaccess for Apache
<IfModule mod_expires.c>
  ExpiresActive On

  # HTML - no cache (or short cache)
  ExpiresByType text/html "access plus 0 seconds"

  # CSS and JS - long cache (use versioned filenames)
  ExpiresByType text/css "access plus 1 year"
  ExpiresByType application/javascript "access plus 1 year"

  # Images - long cache
  ExpiresByType image/png "access plus 1 year"
  ExpiresByType image/jpeg "access plus 1 year"
  ExpiresByType image/svg+xml "access plus 1 year"
</IfModule>
```

16.3 Environment Variables

Manage configuration across environments:

```
// config.js
const configs = {
  development: {
    apiUrl: 'http://localhost:3000/api',
    debug: true
  },
  production: {
    apiUrl: 'https://api.example.com',
    debug: false
  }
};

const env = window.location.hostname === 'localhost' ? 'development' :
        'production';
export const config = configs[env];
```

Or use a build-time approach:

```
<!-- Injected by server/build -->
<script>
  window.CONFIG = {
    apiUrl: '%%API_URL%%',
    version: '%%VERSION%%'
  };
</script>
```

16.4 Pre-Deployment Checklist

Before deploying to production:

- [ ] Test in all target browsers
- [ ] Verify all API endpoints use HTTPS

- [ ] Check for console errors
- [ ] Validate accessibility (keyboard navigation, screen readers)
- [ ] Test on slow network (Chrome DevTools throttling)
- [ ] Verify error handling works
- [ ] Check mobile responsiveness
- [ ] Set up error monitoring (Sentry, LogRocket)
- [ ] Configure analytics
- [ ] Enable HTTPS
- [ ] Set security headers

16.5 Monitoring

Track errors in production:

```
// error-tracking.js
window.addEventListener('error', (event) => {
  fetch('/api/errors', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      message: event.message,
      filename: event.filename,
      lineno: event.lineno,
      colno: event.colno,
      stack: event.error?.stack,
      userAgent: navigator.userAgent,
      url: window.location.href
    })
  });
});

window.addEventListener('unhandledrejection', (event) => {
  fetch('/api/errors', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      message: event.reason?.message || 'Unhandled promise rejection',
      stack: event.reason?.stack,
      userAgent: navigator.userAgent,
      url: window.location.href
    })
  });
});
```

16.6 Netlify Deployment Walkthrough

Let's deploy a LARC app to Netlify step-by-step:

1. Prepare your project:

```
# Project structure
my-larc-app/
├── public/
|   ├── index.html
|   ├── app.js
|   ├── components/
|   └── styles/
├── netlify.toml
└── package.json
```

2. Create `netlify.toml`:

```toml
[build]
  publish = "public"

[build.environment]
  NODE_VERSION = "18"

[[redirects]]
  from = "/*"
  to = "/index.html"
  status = 200

[[headers]]
  for = "/*.js"
  [headers.values]
    Cache-Control = "public, max-age=31536000, immutable"

[[headers]]
  for = "/*.css"
  [headers.values]
    Cache-Control = "public, max-age=31536000, immutable"

[[headers]]
  for = "/index.html"
  [headers.values]
    Cache-Control = "public, max-age=0, must-revalidate"
    X-Frame-Options = "DENY"
    X-Content-Type-Options = "nosniff"
    Referrer-Policy = "strict-origin-when-cross-origin"
    Permissions-Policy = "geolocation=(), microphone=(), camera=()"
```

3. Deploy via Netlify CLI:

```
# Install Netlify CLI
npm install -g netlify-cli

# Login
netlify login

# Initialize site
netlify init

# Deploy
netlify deploy --prod
```

4. Set environment variables:

```
# Via CLI
netlify env:set API_URL "https://api.example.com"
netlify env:set SENTRY_DSN "https://..."

# Or in Netlify UI: Site Settings → Environment Variables
```

5. Access in your app:

```
// Access Netlify environment variables
const config = {
  apiUrl: window.ENV?.API_URL || 'http://localhost:3000',
  sentryDsn: window.ENV?.SENTRY_DSN
};
```

16.7 Vercel Deployment Walkthrough

Deploy to Vercel with zero configuration:

1. Install Vercel CLI:

```
npm install -g vercel
```

2. Create `vercel.json`:

```
{
  "version": 2,
  "builds": [
    {
      "src": "public/**",
      "use": "@vercel/static"
    }
  ],
  "routes": [
    {
      "src": "/(.*)",
      "dest": "/public/$1"
    },
    {
      "src": "/.*",
      "dest": "/public/index.html"
    }
  ],
  "headers": [
    {
      "source": "/public/(.*\\.js|.*\\.css)",
      "headers": [
        {
          "key": "Cache-Control",
          "value": "public, max-age=31536000, immutable"
        }
      ]
    }
  ],
  "env": {
    "API_URL": "@api-url",
    "NODE_ENV": "production"
  }
}
```

3. Deploy:

```
# First deployment
vercel

# Production deployment
vercel --prod

# Set secrets
vercel secrets add api-url "https://api.example.com"
```

4. Configure custom domain:

```
vercel domains add www.example.com
vercel alias deployment-url.vercel.app www.example.com
```

16.8 GitHub Pages Deployment

Deploy directly from your GitHub repository:

1. Create GitHub Actions workflow:

```yaml
# .github/workflows/deploy.yml
name: Deploy to GitHub Pages

on:
  push:
    branches: [main]
  workflow_dispatch:

permissions:
  contents: read
  pages: write
  id-token: write

concurrency:
  group: "pages"
  cancel-in-progress: true

jobs:
  deploy:
    environment:
      name: github-pages
      url: ${{ steps.deployment.outputs.page_url }}
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Setup Pages
        uses: actions/configure-pages@v4

      - name: Upload artifact
        uses: actions/upload-pages-artifact@v3
        with:
          path: './public'
```

```
      - name: Deploy to GitHub Pages
        id: deployment
        uses: actions/deploy-pages@v4
```

2. Configure repository:

1. Go to Settings → Pages
2. Source: “GitHub Actions”
3. Your site will be available at `https://username.github.io/repo-name/`

3. Handle base path:

```
// config.js - Handle GitHub Pages subdirectory
const basePath = window.location.pathname.includes('repo-name')
  ? '/repo-name'
  : '';

export const config = {
  basePath,
  apiUrl: `${basePath}/api`
};

// Use in router
pan.publish('router.navigate', {
  path: `${config.basePath}/about`
});
```

16.9 CI/CD Pipeline Example

Complete GitHub Actions workflow with testing and deployment:

```yaml
# .github/workflows/ci-cd.yml
name: CI/CD Pipeline

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

env:
  NODE_VERSION: '18'

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: ${{ env.NODE_VERSION }}
          cache: 'npm'

      - name: Install dependencies
        run: npm ci

      - name: Run linter
        run: npm run lint

      - name: Run tests
        run: npm test

      - name: Run E2E tests
```

```yaml
      run: npm run test:e2e

    - name: Upload test results
      if: always()
      uses: actions/upload-artifact@v3
      with:
        name: test-results
        path: test-results/

build:
  needs: test
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v4

    - name: Setup Node.js
      uses: actions/setup-node@v4
      with:
        node-version: ${{ env.NODE_VERSION }}

    - name: Create build info
      run: |
        echo "Build: ${{ github.sha }}" > public/build.txt
        echo "Date: $(date)" >> public/build.txt

    - name: Upload build artifact
      uses: actions/upload-artifact@v3
      with:
        name: public
        path: public/

deploy-staging:
  needs: build
  if: github.ref == 'refs/heads/develop'
```

```yaml
    runs-on: ubuntu-latest
    environment:
      name: staging
      url: https://staging.example.com
    steps:
      - name: Download artifact
        uses: actions/download-artifact@v3
        with:
          name: public
          path: public/

      - name: Deploy to staging
        uses: netlify/actions/cli@master
        env:
          NETLIFY_AUTH_TOKEN: ${{ secrets.NETLIFY_AUTH_TOKEN }}
          NETLIFY_SITE_ID: ${{ secrets.NETLIFY_STAGING_SITE_ID }}
        with:
          args: deploy --dir=public --prod

  deploy-production:
    needs: build
    if: github.ref == 'refs/heads/main'
    runs-on: ubuntu-latest
    environment:
      name: production
      url: https://example.com
    steps:
      - name: Download artifact
        uses: actions/download-artifact@v3
        with:
          name: public
          path: public/

      - name: Deploy to production
```

```
        uses: netlify/actions/cli@master
        env:
          NETLIFY_AUTH_TOKEN: ${{ secrets.NETLIFY_AUTH_TOKEN }}
          NETLIFY_SITE_ID: ${{ secrets.NETLIFY_PROD_SITE_ID }}
        with:
          args: deploy --dir=public --prod

      - name: Notify deployment
        run: |
          curl -X POST ${{ secrets.SLACK_WEBHOOK }} \
            -H 'Content-Type: application/json' \
            -d '{"text":"✅ Deployed to production: ${{ github.sha }}"}'
```

16.10 Docker Deployment

Package your app with Nginx for containerized deployment:

Dockerfile:

```
# Use Nginx as base
FROM nginx:alpine

# Copy app files
COPY public/ /usr/share/nginx/html/

# Copy custom Nginx config
COPY nginx.conf /etc/nginx/conf.d/default.conf

# Expose port
EXPOSE 80

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
  CMD wget --quiet --tries=1 --spider http://localhost/health || exit 1

CMD ["nginx", "-g", "daemon off;"]
```

nginx.conf:

```
server {
  listen 80;
  server_name _;
  root /usr/share/nginx/html;
  index index.html;

  # 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;

  # Gzip compression
  gzip on;
  gzip_vary on;
  gzip_types text/css application/javascript application/json image/
svg+xml;
  gzip_min_length 1024;

  # Cache static assets
  location ~* \.(js|css|png|jpg|jpeg|gif|svg|ico|woff|woff2)$ {
    expires 1y;
    add_header Cache-Control "public, immutable";
  }

  # No cache for HTML
  location ~* \.html$ {
    expires -1;
    add_header Cache-Control "no-store, no-cache, must-revalidate, proxy-
revalidate";
  }

  # SPA fallback
  location / {
```

```
    try_files $uri $uri/ /index.html;
  }

  # Health check endpoint
  location /health {
    access_log off;
    return 200 "healthy\n";
    add_header Content-Type text/plain;
  }
}
```

Build and run:

```
# Build image
docker build -t my-larc-app:latest .

# Run locally
docker run -p 8080:80 my-larc-app:latest

# Push to registry
docker tag my-larc-app:latest registry.example.com/my-larc-app:latest
docker push registry.example.com/my-larc-app:latest

# Deploy with docker-compose
cat > docker-compose.yml << EOF
version: '3.8'
services:
  app:
    image: my-larc-app:latest
    ports:
      - "80:80"
    environment:
      - NODE_ENV=production
    restart: unless-stopped
EOF

docker-compose up -d
```

16.11 Security Headers

Protect your app with proper HTTP headers:

```
// For Netlify (_headers file)
/*
  X-Frame-Options: DENY
  X-Content-Type-Options: nosniff
  X-XSS-Protection: 1; mode=block
  Referrer-Policy: strict-origin-when-cross-origin
  Permissions-Policy: geolocation=(), microphone=(), camera=()
  Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-
        inline' https://cdn.jsdelivr.net; style-src 'self' 'unsafe-
        inline'; img-src 'self' data: https:; font-src 'self' data:;
        connect-src 'self' https://api.example.com

/*.js
  Cache-Control: public, max-age=31536000, immutable

/*.css
  Cache-Control: public, max-age=31536000, immutable

/index.html
  Cache-Control: public, max-age=0, must-revalidate
```

Test security headers:

```
curl -I https://your-site.com | grep -E "X-Frame-Options|X-Content-Type"
```

Or use online tools:

- https://securityheaders.com
- https://observatory.mozilla.org

16.12 Performance Optimization for Production

1. Enable compression:

```
# Nginx
gzip on;
gzip_vary on;
gzip_min_length 1024;
gzip_types text/plain text/css text/xml text/javascript
           application/x-javascript application/xml+rss
           application/javascript application/json image/svg+xml;
```

2. Set cache headers:

```
// For Cloudflare Workers
export default {
  async fetch(request) {
    const url = new URL(request.url);
    const response = await fetch(request);

    // Clone response so we can modify headers
    const newResponse = new Response(response.body, response);

    if (url.pathname.match(/\.(js|css|png|jpg|jpeg|svg|woff2?)$/)) {
      newResponse.headers.set('Cache-Control', 'public, max-age=31536000,
        immutable');
    } else if (url.pathname.endsWith('.html')) {
      newResponse.headers.set('Cache-Control', 'public, max-age=0, must-
        revalidate');
    }

    return newResponse;
  }
};
```

3. Preload critical resources:

```html
<!DOCTYPE html>
<html>
<head>
  <!-- Preload critical resources -->
  <link rel="preload" href="/app.js" as="script">
  <link rel="preload" href="/styles/main.css" as="style">

  <!-- Preconnect to external domains -->
  <link rel="preconnect" href="https://api.example.com">
  <link rel="preconnect" href="https://cdn.jsdelivr.net">

  <!-- DNS prefetch for third-party domains -->
  <link rel="dns-prefetch" href="https://analytics.example.com">
</head>
</html>
```

16.13 Rollback Strategies

Prepare for when deployments go wrong:

1. Keep previous versions:

```
# With Netlify CLI
netlify deploy --prod  # Deploys new version

# Rollback to previous deploy
netlify rollback

# Or via UI: Deploys tab → Click on previous deploy → "Publish deploy"
```

2. Blue-green deployment:

```yaml
# Deploy to staging (green), then swap with production (blue)
name: Blue-Green Deployment

on:
  workflow_dispatch:

jobs:
  deploy-green:
    runs-on: ubuntu-latest
    steps:
      - name: Deploy to green environment
        run: |
          netlify deploy --site=${{ secrets.GREEN_SITE_ID }} --prod

      - name: Run smoke tests
        run: npm run test:smoke -- --url=https://green.example.com

      - name: Swap blue and green
        if: success()
        run: |
          # Update DNS or load balancer to point to green
          # This is provider-specific
          echo "Swapping environments..."
```

3. Feature flags for gradual rollout:

```
// feature-flags.js
class FeatureFlags {
  constructor() {
    this.flags = {};
    this.loadFlags();
  }

  async loadFlags() {
    try {
      const response = await fetch('/api/feature-flags');
      this.flags = await response.json();
    } catch (error) {
      console.error('Failed to load feature flags:', error);
    }
  }

  isEnabled(feature, userId = null) {
    const flag = this.flags[feature];
    if (!flag) return false;

    // Global enable/disable
    if (flag.enabled === false) return false;

    // Percentage rollout
    if (flag.percentage && userId) {
      const hash = this.hashUserId(userId);
      return (hash % 100) < flag.percentage;
    }

    // Whitelist
    if (flag.whitelist && userId) {
      return flag.whitelist.includes(userId);
    }
```

```
    return flag.enabled;
  }

  hashUserId(userId) {
    // Simple hash for percentage rollout
    let hash = 0;
    for (let i = 0; i < userId.length; i++) {
      hash = ((hash << 5) - hash) + userId.charCodeAt(i);
      hash = hash & hash;
    }
    return Math.abs(hash);
  }
}

export const featureFlags = new FeatureFlags();

// Usage
if (featureFlags.isEnabled('new-dashboard', user.id)) {
  await import('./components/new-dashboard.js');
} else {
  await import('./components/old-dashboard.js');
}
```

16.14 Complete Deployment Example

Let's deploy a complete app with monitoring and error tracking:

1. Project structure:

```
my-app/
├── public/
|   ├── index.html
|   ├── app.js
|   ├── config.js
|   └── components/
├── scripts/
|   └── deploy.sh
├── .github/workflows/
|   └── deploy.yml
├── netlify.toml
└── package.json
```

2. Environment-aware config:

```
// public/config.js
const environments = {
  local: {
    apiUrl: 'http://localhost:3000',
    sentryDsn: null,
    analytics: null
  },
  staging: {
    apiUrl: 'https://staging-api.example.com',
    sentryDsn: 'https://...@sentry.io/staging',
    analytics: 'UA-STAGING'
  },
  production: {
    apiUrl: 'https://api.example.com',
    sentryDsn: 'https://...@sentry.io/prod',
    analytics: 'UA-PROD'
  }
};

function detectEnvironment() {
  const hostname = window.location.hostname;
  if (hostname === 'localhost' || hostname === '127.0.0.1') {
    return 'local';
  }
  if (hostname.includes('staging')) {
    return 'staging';
  }
  return 'production';
}

export const config = environments[detectEnvironment()];
```

3. Error tracking setup:

```
// public/monitoring.js
import { config } from './config.js';

class ErrorTracker {
  constructor() {
    if (config.sentryDsn) {
      this.initSentry();
    }
    this.setupErrorHandlers();
  }

  async initSentry() {
    const Sentry = await import('https://cdn.jsdelivr.net/npm/@sentry/
        browser@7/+esm');

    Sentry.init({
      dsn: config.sentryDsn,
      environment: config.environment,
      beforeSend(event) {
        // Filter out noise
        if (event.message?.includes('ResizeObserver')) {
          return null;
        }
        return event;
      }
    });
  }

  setupErrorHandlers() {
    window.addEventListener('error', (event) => {
      this.captureError(event.error, {
        filename: event.filename,
        lineno: event.lineno,
        colno: event.colno
```

```
      });
    });

    window.addEventListener('unhandledrejection', (event) => {
      this.captureError(event.reason, {
        type: 'unhandledrejection'
      });
    });
  }

  captureError(error, context = {}) {
    console.error('Error captured:', error, context);

    // Send to your error tracking service
    if (config.sentryDsn && window.Sentry) {
      window.Sentry.captureException(error, { extra: context });
    }
  }
}

export const errorTracker = new ErrorTracker();
```

4. Deployment script:

```bash
#!/bin/bash
# scripts/deploy.sh

set -e

ENV=${1:-staging}

echo "🚀 Deploying to $ENV..."

# Run tests
echo "🧪 Running tests..."
npm test

# Check for uncommitted changes
if [[ -n $(git status -s) ]]; then
  echo "❌ Uncommitted changes detected. Commit or stash them first."
  exit 1
fi

# Get current version
VERSION=$(git rev-parse --short HEAD)
echo "📦 Version: $VERSION"

# Create build info
cat > public/build-info.json << EOF
{
  "version": "$VERSION",
  "environment": "$ENV",
  "buildDate": "$(date -u +"%Y-%m-%dT%H:%M:%SZ")",
  "branch": "$(git rev-parse --abbrev-ref HEAD)"
}
EOF

# Deploy based on environment
```

```
if [ "$ENV" = "production" ]; then
  echo "🌍 Deploying to production..."
  netlify deploy --prod --site=$NETLIFY_PROD_SITE_ID
elif [ "$ENV" = "staging" ]; then
  echo "🔧 Deploying to staging..."
  netlify deploy --prod --site=$NETLIFY_STAGING_SITE_ID
fi

echo "✅ Deployment complete!"
echo "🔗 Check deployment: https://$ENV.example.com"
```

16.15 Troubleshooting Deployment Issues

16.15.1 Problem 1: 404 on Refresh (SPA Routing)

Symptoms: Navigating directly to `/about` returns 404, but clicking links works.

Cause: Server doesn't know about client-side routes.

Solution:

```
# netlify.toml
[[redirects]]
  from = "/*"
  to = "/index.html"
  status = 200

# vercel.json
{
  "rewrites": [
    { "source": "/(.*)", "destination": "/index.html" }
  ]
}

# Apache .htaccess
<IfModule mod_rewrite.c>
  RewriteEngine On
  RewriteBase /
  RewriteRule ^index\.html$ - [L]
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteRule . /index.html [L]
</IfModule>
```

16.15.2 Problem 2: CORS Errors in Production

Symptoms: API calls work locally but fail in production with CORS errors.

Cause: API server not configured to allow your production domain.

Solution:

```
// Backend CORS configuration
app.use(cors({
  origin: [
    'https://example.com',
    'https://www.example.com',
    'https://staging.example.com'
  ],
  credentials: true
}));

// Or use environment variable
app.use(cors({
  origin: process.env.ALLOWED_ORIGINS?.split(','),
  credentials: true
}));
```

Alternative: Use a proxy in your deployment config:

```
# netlify.toml
[[redirects]]
  from = "/api/*"
  to = "https://api.example.com/:splat"
  status = 200
  force = true
```

16.15.3 Problem 3: Environment Variables Not Working

Symptoms: App can't read environment variables, falls back to defaults.

Cause: Build-time vs. runtime configuration confusion.

Solution:

For runtime configuration, inject variables into HTML:

```
<!-- Injected by build process or server -->
<script>
  window.ENV = {
    API_URL: "%%API_URL%%",
    SENTRY_DSN: "%%SENTRY_DSN%%"
  };
</script>
```

Or load from a config endpoint:

```
async function loadConfig() {
  try {
    const response = await fetch('/config.json');
    return await response.json();
  } catch (error) {
    console.error('Failed to load config:', error);
    return {
      apiUrl: 'http://localhost:3000',
      // fallback values
    };
  }
}

export const config = await loadConfig();
```

16.15.4 Problem 4: Slow Initial Load

Symptoms: First visit takes several seconds to show content.

Diagnosis:

```
// Add timing to index.html
<script>
  window.perfMetrics = {
    navigationStart: performance.timing.navigationStart,
    fetchStart: performance.timing.fetchStart,
    domainLookupEnd: performance.timing.domainLookupEnd,
    connectEnd: performance.timing.connectEnd,
    responseEnd: performance.timing.responseEnd,
    domContentLoadedEventEnd:
        performance.timing.domContentLoadedEventEnd,
    loadEventEnd: performance.timing.loadEventEnd
  };

  window.addEventListener('load', () => {
    const metrics = window.perfMetrics;
    console.log('DNS lookup:', metrics.domainLookupEnd -
        metrics.fetchStart);
    console.log('TCP connection:', metrics.connectEnd -
        metrics.domainLookupEnd);
    console.log('Response time:', metrics.responseEnd -
        metrics.connectEnd);
    console.log('DOM processing:', metrics.domContentLoadedEventEnd -
        metrics.responseEnd);
    console.log('Total load time:', metrics.loadEventEnd -
        metrics.navigationStart);
  });
</script>
```

Solutions:

- Enable compression (gzip/brotli)
- Use CDN for static assets
- Preload critical resources
- Lazy load non-critical components
- Optimize images
- Minify JavaScript and CSS

16.16 Deployment Best Practices

1. **Automate Everything**: Use CI/CD pipelines for consistent, reliable deployments.
2. **Test Before Deploying**: Run linters, unit tests, and E2E tests in your pipeline.
3. **Use Staging Environments**: Test production builds in a staging environment first.
4. **Monitor Deployments**: Track errors, performance, and user behavior after each deploy.
5. **Enable Rollbacks**: Keep previous versions and be ready to rollback quickly.
6. **Version Your Releases**: Tag releases in Git and track which version is deployed where.
7. **Secure Your Secrets**: Never commit API keys or secrets. Use environment variables or secret managers.
8. **Set Cache Headers Properly**: Long cache for assets (with versioned filenames), no cache for HTML.
9. **Use Health Checks**: Implement `/health` endpoints to verify deployments succeeded.
10. **Document Your Process**: Maintain runbooks for common deployment scenarios and troubleshooting.

16.17 Hands-On Exercises

16.17.1 Exercise 1: Deploy to Netlify

Deploy your LARC app to Netlify:

- Set up a free Netlify account

- Connect your Git repository
- Configure build settings and environment variables
- Set up a custom domain (or use Netlify subdomain)
- Configure security headers

Bonus: Set up preview deployments for pull requests.

16.17.2 Exercise 2: Create a CI/CD Pipeline

Build a complete GitHub Actions workflow that:

- Runs tests on every push
- Deploys to staging on merge to `develop`
- Deploys to production on merge to `main`
- Sends notifications on deployment success/failure

Bonus: Add automated performance testing with Lighthouse CI.

16.17.3 Exercise 3: Implement Feature Flags

Create a feature flag system that:

- Loads flags from a remote config
- Supports percentage-based rollouts
- Allows user whitelisting
- Caches flags locally
- Has a UI to toggle features

Bonus: Add A/B testing with analytics integration.

16.17.4 Exercise 4: Set Up Error Tracking

Implement production error tracking:

- Integrate Sentry or similar service
- Capture unhandled errors and promise rejections

- Track custom errors with context
- Filter out noise (ResizeObserver, etc.)
- Set up alerts for critical errors

Bonus: Add user session replay for debugging.

16.18 Summary

Deploying LARC applications is straightforward—no build process means fewer things to go wrong. Key takeaways:

- **Static hosting is simple**: Use Netlify, Vercel, GitHub Pages, or any static host
- **Automate with CI/CD**: Let GitHub Actions handle testing and deployment
- **Security matters**: Set proper headers, use HTTPS, configure CSP
- **Monitor production**: Track errors, performance, and user experience
- **Be ready to rollback**: Keep previous versions and have a rollback plan

Start with simple deployments and add sophistication as needed. The beauty of LARC's no-build approach is that deployment remains simple even as your app grows.

16.19 Further Reading

- **Building with LARC - Chapter 18 (Deployment)**: Advanced deployment patterns and strategies
- **Building with LARC - Chapter 19 (Performance)**: Production optimization techniques
- **Building with LARC - Chapter 11 (Best Practices)**: Security and reliability patterns

17 Component Library

As your application grows, you'll accumulate reusable components. A well-organized component library accelerates development and ensures consistency.

17.1 Organizing Components

Structure your library logically:

```
components/
├── core/
|   ├── pan-button.js
|   ├── pan-input.js
|   └── pan-card.js
├── layout/
|   ├── pan-header.js
|   ├── pan-sidebar.js
|   └── pan-grid.js
├── data/
|   ├── pan-table.js
|   ├── pan-list.js
|   └── pan-pagination.js
└── index.js
```

Export from a single entry point:

```
// components/index.js
export * from './core/pan-button.js';
export * from './core/pan-input.js';
export * from './core/pan-card.js';
export * from './layout/pan-header.js';
// ...
```

17.2 Documentation

Document every component:

```
/**
 * A customizable button component.
 *
 * @element pan-button
 *
 * @attr {string} variant - Button style: "primary", "secondary",
       "danger"
 * @attr {boolean} disabled - Disables the button
 * @attr {string} size - Button size: "small", "medium", "large"
 *
 * @fires click - Fired when button is clicked
 *
 * @slot - Button content
 *
 * @example
 * <pan-button variant="primary">Click me</pan-button>
 *
 * @example
 * <pan-button variant="danger" disabled>Delete</pan-button>
 */
class PanButton extends HTMLElement {
  // ...
}
```

Generate documentation automatically with tools like `web-component-analyzer`:

```
npx web-component-analyzer analyze components/**/*.js --outFile docs.json
```

17.3 Design Tokens

Use CSS custom properties for theming:

```css
/* tokens.css */
:root {
  /* Colors */
  --color-primary: #0066cc;
  --color-secondary: #6c757d;
  --color-success: #28a745;
  --color-danger: #dc3545;

  /* Spacing */
  --space-xs: 4px;
  --space-sm: 8px;
  --space-md: 16px;
  --space-lg: 24px;
  --space-xl: 32px;

  /* Typography */
  --font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto,
        sans-serif;
  --font-size-sm: 0.875rem;
  --font-size-md: 1rem;
  --font-size-lg: 1.25rem;

  /* Borders */
  --border-radius: 4px;
  --border-width: 1px;
  --border-color: #dee2e6;
}
```

Components use these tokens:

```
class PanButton extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
  }

  connectedCallback() {
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: inline-block;
        }

        button {
          font-family: var(--font-family);
          font-size: var(--font-size-md);
          padding: var(--space-sm) var(--space-md);
          border-radius: var(--border-radius);
          border: var(--border-width) solid transparent;
          cursor: pointer;
        }

        :host([variant="primary"]) button {
          background: var(--color-primary);
          color: white;
        }

        :host([variant="secondary"]) button {
          background: var(--color-secondary);
          color: white;
        }

        :host([disabled]) button {
          opacity: 0.5;
```

```
          cursor: not-allowed;
        }
      </style>
      <button><slot></slot></button>
    `;
  }
}
```

17.4 Versioning and Publishing

Use semantic versioning. Publish to npm or a private registry:

```
{
  "name": "@myorg/components",
  "version": "1.2.0",
  "type": "module",
  "exports": {
    ".": "./index.js",
    "./button": "./core/pan-button.js",
    "./card": "./core/pan-card.js"
  }
}
```

```
npm publish --access public
```

17.5 Building a Complete Component

Let’s build a production-ready dialog component from scratch:

```
// components/core/pan-dialog.js

/**
 * A modal dialog component with accessibility support.
 *
 * @element pan-dialog
 *
 * @attr {boolean} open - Controls dialog visibility
 * @attr {string} title - Dialog title
 * @attr {boolean} modal - Whether dialog is modal (blocks background)
 * @attr {boolean} close-on-escape - Close on Escape key (default: true)
 * @attr {boolean} close-on-backdrop - Close on backdrop click (default:
       true)
 *
 * @fires open - Fired when dialog opens
 * @fires close - Fired when dialog closes
 * @fires cancel - Fired when user tries to close (cancelable)
 *
 * @slot - Main dialog content
 * @slot header - Custom header content
 * @slot footer - Custom footer content
 *
 * @csspart dialog - The dialog container
 * @csspart header - The header section
 * @csspart body - The body section
 * @csspart footer - The footer section
 *
 * @cssprop --dialog-width - Dialog width (default: 500px)
 * @cssprop --dialog-max-width - Maximum dialog width (default: 90vw)
 * @cssprop --dialog-backdrop - Backdrop color (default: rgba(0,0,0,0.5))
 *
 * @example
 * <pan-dialog open title="Confirm Delete">
 *   <p>Are you sure you want to delete this item?</p>
```

```
 *   <div slot="footer">
 *     <button>Cancel</button>
 *     <button>Delete</button>
 *   </div>
 * </pan-dialog>
 */
class PanDialog extends HTMLElement {
  static observedAttributes = ['open', 'title'];

  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this._previouslyFocused = null;
  }

  connectedCallback() {
    this.render();
    this.setupAccessibility();
    this.setupEventListeners();
  }

  disconnectedCallback() {
    this.removeEventListeners();
  }

  attributeChangedCallback(name, oldValue, newValue) {
    if (oldValue === newValue) return;

    if (name === 'open') {
      newValue !== null ? this.show() : this.hide();
    } else if (name === 'title') {
      this.updateTitle();
    }
  }
```

```
render() {
  this.shadowRoot.innerHTML = `
    <style>
      :host {
        display: none;
        position: fixed;
        top: 0;
        left: 0;
        width: 100%;
        height: 100%;
        z-index: 1000;
      }

      :host([open]) {
        display: flex;
        align-items: center;
        justify-content: center;
      }

      .backdrop {
        position: absolute;
        top: 0;
        left: 0;
        width: 100%;
        height: 100%;
        background: var(--dialog-backdrop, rgba(0, 0, 0, 0.5));
        animation: fadeIn 0.2s ease-out;
      }

      .dialog {
        position: relative;
        width: var(--dialog-width, 500px);
        max-width: var(--dialog-max-width, 90vw);
```

```
  max-height: 90vh;
  background: var(--dialog-bg, white);
  border-radius: var(--border-radius, 8px);
  box-shadow: 0 10px 40px rgba(0, 0, 0, 0.2);
  display: flex;
  flex-direction: column;
  animation: slideUp 0.3s ease-out;
}

.header {
  padding: var(--space-md, 16px);
  border-bottom: 1px solid var(--border-color, #e0e0e0);
  display: flex;
  align-items: center;
  justify-content: space-between;
}

.title {
  font-size: var(--font-size-lg, 1.25rem);
  font-weight: 600;
  margin: 0;
}

.close-button {
  background: none;
  border: none;
  font-size: 1.5rem;
  cursor: pointer;
  padding: 4px 8px;
  border-radius: 4px;
  color: var(--color-text-secondary, #666);
}

.close-button:hover {
```

```
      background: var(--color-hover, #f0f0f0);
    }

    .body {
      padding: var(--space-md, 16px);
      overflow-y: auto;
      flex: 1;
    }

    .footer {
      padding: var(--space-md, 16px);
      border-top: 1px solid var(--border-color, #e0e0e0);
      display: flex;
      justify-content: flex-end;
      gap: var(--space-sm, 8px);
    }

    @keyframes fadeIn {
      from { opacity: 0; }
      to { opacity: 1; }
    }

    @keyframes slideUp {
      from {
        transform: translateY(20px);
        opacity: 0;
      }
      to {
        transform: translateY(0);
        opacity: 1;
      }
    }
  </style>
```

```
      <div class="backdrop" part="backdrop"></div>
      <div class="dialog" part="dialog" role="dialog" aria-modal="true">
        <div class="header" part="header">
          <slot name="header">
            <h2 class="title" id="dialog-
        title">${this.getAttribute('title') || ''}</h2>
          </slot>
          <button class="close-button" aria-label="Close dialog"
        type="button">
            ×
          </button>
        </div>
        <div class="body" part="body">
          <slot></slot>
        </div>
        <div class="footer" part="footer">
          <slot name="footer"></slot>
        </div>
      </div>
    `;
  }

  setupAccessibility() {
    const dialog = this.shadowRoot.querySelector('.dialog');
    dialog.setAttribute('aria-labelledby', 'dialog-title');

    // Set focus trap
    const focusableElements = dialog.querySelectorAll(
      'button, [href], input, select, textarea,
        [tabindex]:not([tabindex="-1"])'
    );

    this._firstFocusable = focusableElements[0];
    this._lastFocusable = focusableElements[focusableElements.length -
        1];
  }
```

```
setupEventListeners() {
  // Close button
  this.shadowRoot.querySelector('.close-
      button').addEventListener('click', () => {
    this.close();
  });

  // Backdrop click
  if (this.hasAttribute('close-on-backdrop') ||
      !this.hasAttribute('close-on-backdrop')) {
    this.shadowRoot.querySelector('.backdrop').addEventListener('click', ()
      => {

      this.close();
    });
  }

  // Escape key
  this._keydownHandler = (e) => {
    if (e.key === 'Escape' && (this.hasAttribute('close-on-escape') ||
      !this.hasAttribute('close-on-escape'))) {
      this.close();
    }

    // Focus trap
    if (e.key === 'Tab' && this.hasAttribute('open')) {
      if (e.shiftKey) {
        if (document.activeElement === this._firstFocusable) {
          e.preventDefault();
          this._lastFocusable?.focus();
        }
      } else {
        if (document.activeElement === this._lastFocusable) {
          e.preventDefault();
          this._firstFocusable?.focus();
```

```
        }
      }
    }
  };

  document.addEventListener('keydown', this._keydownHandler);
}

removeEventListeners() {
  document.removeEventListener('keydown', this._keydownHandler);
}

show() {
  // Store previously focused element
  this._previouslyFocused = document.activeElement;

  // Prevent body scroll
  document.body.style.overflow = 'hidden';

  // Focus first focusable element
  setTimeout(() => {
    this._firstFocusable?.focus();
  }, 100);

  // Fire event
  this.dispatchEvent(new CustomEvent('open'));
}

hide() {
  // Restore body scroll
  document.body.style.overflow = '';

  // Restore focus
  this._previouslyFocused?.focus();
```

```
    // Fire event
    this.dispatchEvent(new CustomEvent('close'));
  }

  close() {
    // Fire cancelable event
    const event = new CustomEvent('cancel', {
      cancelable: true
    });

    this.dispatchEvent(event);

    if (!event.defaultPrevented) {
      this.removeAttribute('open');
    }
  }

  updateTitle() {
    const titleEl = this.shadowRoot.getElementById('dialog-title');
    if (titleEl) {
      titleEl.textContent = this.getAttribute('title') || '';
    }
  }
}

customElements.define('pan-dialog', PanDialog);
```

17.6 Component Showcase

Build a documentation page to showcase components:

```
// docs/showcase.js
class ComponentShowcase extends HTMLElement {
  connectedCallback() {
    this.innerHTML = `
      <style>
        .showcase {
          max-width: 1200px;
          margin: 0 auto;
          padding: 2rem;
        }

        .component-demo {
          margin: 2rem 0;
          padding: 2rem;
          border: 1px solid #e0e0e0;
          border-radius: 8px;
        }

        .demo-title {
          font-size: 1.5rem;
          margin-bottom: 1rem;
        }

        .demo-example {
          padding: 1rem;
          background: #f8f8f8;
          border-radius: 4px;
          margin: 1rem 0;
        }

        .demo-code {
          background: #282c34;
          color: #abb2bf;
          padding: 1rem;
```

```
      border-radius: 4px;
      overflow-x: auto;
    }

    .demo-props {
      margin-top: 1rem;
    }

    .prop-table {
      width: 100%;
      border-collapse: collapse;
    }

    .prop-table th,
    .prop-table td {
      text-align: left;
      padding: 0.5rem;
      border-bottom: 1px solid #e0e0e0;
    }
  </style>

  <div class="showcase">
    <h1>Component Library</h1>

    <div class="component-demo">
      <h2 class="demo-title">Dialog</h2>
      <p>A modal dialog with accessibility features.</p>

      <div class="demo-example">
        <button id="open-dialog">Open Dialog</button>
        <pan-dialog id="demo-dialog" title="Example Dialog">
          <p>This is a sample dialog with default settings.</p>
          <div slot="footer">
            <button id="cancel-btn">Cancel</button>
```

```
            <button id="confirm-btn" style="background: #0066cc;
        color: white; padding: 8px 16px; border: none; border-radius:
        4px; cursor: pointer;">Confirm</button>
          </div>
        </pan-dialog>
      </div>

      <div class="demo-code">
<pre>&lt;pan-dialog open title="Example Dialog"&gt;
  &lt;p&gt;This is a sample dialog.&lt;/p&gt;
  &lt;div slot="footer"&gt;
    &lt;button&gt;Cancel&lt;/button&gt;
    &lt;button&gt;Confirm&lt;/button&gt;
  &lt;/div&gt;
&lt;/pan-dialog&gt;</pre>
      </div>

      <div class="demo-props">
        <h3>Properties</h3>
        <table class="prop-table">
          <thead>
            <tr>
              <th>Attribute</th>
              <th>Type</th>
              <th>Default</th>
              <th>Description</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td><code>open</code></td>
              <td>boolean</td>
              <td>false</td>
              <td>Controls dialog visibility</td>
            </tr>
```

```
              <tr>
                <td><code>title</code></td>
                <td>string</td>
                <td>""</td>
                <td>Dialog title</td>
              </tr>
              <tr>
                <td><code>close-on-escape</code></td>
                <td>boolean</td>
                <td>true</td>
                <td>Close on Escape key</td>
              </tr>
              <tr>
                <td><code>close-on-backdrop</code></td>
                <td>boolean</td>
                <td>true</td>
                <td>Close on backdrop click</td>
              </tr>
            </tbody>
          </table>
        </div>
      </div>
    </div>
  `;

  // Setup demo interactivity
  this.setupDemos();
}

setupDemos() {
  const openBtn = this.querySelector('#open-dialog');
  const dialog = this.querySelector('#demo-dialog');
  const cancelBtn = this.querySelector('#cancel-btn');
  const confirmBtn = this.querySelector('#confirm-btn');
```

```
    openBtn.addEventListener('click', () => {
      dialog.setAttribute('open', '');
    });

    cancelBtn.addEventListener('click', () => {
      dialog.removeAttribute('open');
    });

    confirmBtn.addEventListener('click', () => {
      alert('Confirmed!');
      dialog.removeAttribute('open');
    });
  }
}

customElements.define('component-showcase', ComponentShowcase);
```

17.7 Testing Components

Write comprehensive tests for your components:

```
// components/core/pan-dialog.test.js
import { expect, fixture, html, oneEvent } from '@open-wc/testing';
import './pan-dialog.js';

describe('PanDialog', () => {
  it('should render with default attributes', async () => {
    const el = await fixture(html`<pan-dialog></pan-dialog>`);
    expect(el).to.exist;
    expect(el.hasAttribute('open')).to.be.false;
  });

  it('should show when open attribute is set', async () => {
    const el = await fixture(html`<pan-dialog open></pan-dialog>`);
    const computedStyle = window.getComputedStyle(el);
    expect(computedStyle.display).to.equal('flex');
  });

  it('should hide when open attribute is removed', async () => {
    const el = await fixture(html`<pan-dialog open></pan-dialog>`);
    el.removeAttribute('open');
    await el.updateComplete;
    const computedStyle = window.getComputedStyle(el);
    expect(computedStyle.display).to.equal('none');
  });

  it('should fire open event when opened', async () => {
    const el = await fixture(html`<pan-dialog></pan-dialog>`);
    setTimeout(() => el.setAttribute('open', ''));
    const { detail } = await oneEvent(el, 'open');
    expect(detail).to.exist;
  });

  it('should fire close event when closed', async () => {
    const el = await fixture(html`<pan-dialog open></pan-dialog>`);
```

```
    setTimeout(() => el.removeAttribute('open'));
    const { detail } = await oneEvent(el, 'close');
    expect(detail).to.exist;
  });

  it('should close on backdrop click', async () => {
    const el = await fixture(html`<pan-dialog open close-on-
        backdrop></pan-dialog>`);
    const backdrop = el.shadowRoot.querySelector('.backdrop');

    setTimeout(() => backdrop.click());
    await oneEvent(el, 'cancel');

    expect(el.hasAttribute('open')).to.be.false;
  });

  it('should close on Escape key', async () => {
    const el = await fixture(html`<pan-dialog open close-on-escape></pan-
        dialog>`);

    setTimeout(() => {
      const event = new KeyboardEvent('keydown', { key: 'Escape' });
      document.dispatchEvent(event);
    });

    await oneEvent(el, 'cancel');
    expect(el.hasAttribute('open')).to.be.false;
  });

  it('should trap focus within dialog', async () => {
    const el = await fixture(html`
      <pan-dialog open>
        <button id="btn1">Button 1</button>
        <button id="btn2">Button 2</button>
      </pan-dialog>
```

```
    `);

    const btn1 = el.querySelector('#btn1');
    const btn2 = el.querySelector('#btn2');

    btn2.focus();
    expect(document.activeElement).to.equal(btn2);

    // Simulate Tab key on last focusable element
    const tabEvent = new KeyboardEvent('keydown', { key: 'Tab', bubbles:
        true });
    document.dispatchEvent(tabEvent);

    // Should wrap to first focusable element
    expect(document.activeElement).to.equal(btn1);
  });

  it('should support custom CSS parts', async () => {
    const el = await fixture(html`<pan-dialog open></pan-dialog>`);
    const dialog = el.shadowRoot.querySelector('[part="dialog"]');
    const header = el.shadowRoot.querySelector('[part="header"]');
    const body = el.shadowRoot.querySelector('[part="body"]');

    expect(dialog).to.exist;
    expect(header).to.exist;
    expect(body).to.exist;
  });

  it('should render slotted content', async () => {
    const el = await fixture(html`
      <pan-dialog open>
        <p>Custom content</p>
        <button slot="footer">Action</button>
      </pan-dialog>
    `);
```

```
    const paragraph = el.querySelector('p');
    const footerButton = el.querySelector('[slot="footer"]');

    expect(paragraph.textContent).to.equal('Custom content');
    expect(footerButton.textContent).to.equal('Action');
  });
});
```

17.8 Theming System

Create a comprehensive theming system:

```
// themes/theme-manager.js
class ThemeManager {
  constructor() {
    this.themes = new Map();
    this.currentTheme = 'default';
  }

  registerTheme(name, tokens) {
    this.themes.set(name, tokens);
  }

  applyTheme(name) {
    const theme = this.themes.get(name);
    if (!theme) {
      console.warn(`Theme "${name}" not found`);
      return;
    }

    // Apply CSS custom properties to :root
    Object.entries(theme).forEach(([key, value]) => {
      document.documentElement.style.setProperty(key, value);
    });

    this.currentTheme = name;

    // Store preference
    localStorage.setItem('theme', name);

    // Dispatch event
    window.dispatchEvent(new CustomEvent('theme-changed', {
      detail: { theme: name }
    }));
  }
```

```
  getCurrentTheme() {
    return this.currentTheme;
  }

  getAvailableThemes() {
    return Array.from(this.themes.keys());
  }
}

export const themeManager = new ThemeManager();

// Register default theme
themeManager.registerTheme('default', {
  '--color-primary': '#0066cc',
  '--color-secondary': '#6c757d',
  '--color-success': '#28a745',
  '--color-danger': '#dc3545',
  '--color-warning': '#ffc107',
  '--color-info': '#17a2b8',

  '--color-text': '#212529',
  '--color-text-secondary': '#6c757d',
  '--color-bg': '#ffffff',
  '--color-bg-secondary': '#f8f9fa',

  '--space-xs': '4px',
  '--space-sm': '8px',
  '--space-md': '16px',
  '--space-lg': '24px',
  '--space-xl': '32px',

  '--font-family': '-apple-system, BlinkMacSystemFont, "Segoe UI",
      Roboto, sans-serif',
  '--font-size-sm': '0.875rem',
  '--font-size-md': '1rem',
```

```
  '--font-size-lg': '1.25rem',
  '--font-size-xl': '1.5rem',

  '--border-radius': '4px',
  '--border-radius-lg': '8px',
  '--border-width': '1px',
  '--border-color': '#dee2e6',

  '--shadow-sm': '0 1px 2px rgba(0, 0, 0, 0.05)',
  '--shadow-md': '0 4px 6px rgba(0, 0, 0, 0.1)',
  '--shadow-lg': '0 10px 15px rgba(0, 0, 0, 0.1)',
});

// Register dark theme
themeManager.registerTheme('dark', {
  '--color-primary': '#4d9fff',
  '--color-secondary': '#6c757d',
  '--color-success': '#28a745',
  '--color-danger': '#dc3545',
  '--color-warning': '#ffc107',
  '--color-info': '#17a2b8',

  '--color-text': '#e9ecef',
  '--color-text-secondary': '#adb5bd',
  '--color-bg': '#212529',
  '--color-bg-secondary': '#343a40',

  '--border-color': '#495057',

  '--shadow-sm': '0 1px 2px rgba(0, 0, 0, 0.3)',
  '--shadow-md': '0 4px 6px rgba(0, 0, 0, 0.4)',
  '--shadow-lg': '0 10px 15px rgba(0, 0, 0, 0.4)',
});
```

```
// Apply saved theme or default
const savedTheme = localStorage.getItem('theme') || 'default';
themeManager.applyTheme(savedTheme);
```

Theme switcher component:

```
class ThemeSwitcher extends HTMLElement {
  connectedCallback() {
    import('./theme-manager.js').then(({ themeManager }) => {
      this.themeManager = themeManager;
      this.render();
    });
  }

  render() {
    const themes = this.themeManager.getAvailableThemes();
    const current = this.themeManager.getCurrentTheme();

    this.innerHTML = `
      <style>
        .theme-switcher {
          display: inline-flex;
          gap: 0.5rem;
        }

        .theme-button {
          padding: 0.5rem 1rem;
          border: 1px solid var(--border-color);
          border-radius: var(--border-radius);
          background: var(--color-bg);
          color: var(--color-text);
          cursor: pointer;
        }

        .theme-button.active {
          background: var(--color-primary);
          color: white;
          border-color: var(--color-primary);
        }
      </style>
```

```
      <div class="theme-switcher">
        ${themes.map(theme => `
          <button
            class="theme-button ${theme === current ? 'active' : ''}"
            data-theme="${theme}"
          >
            ${theme}
          </button>
        `).join('')}
      </div>
    `;

    // Add event listeners
    this.querySelectorAll('.theme-button').forEach(btn => {
      btn.addEventListener('click', () => {
        const theme = btn.dataset.theme;
        this.themeManager.applyTheme(theme);
        this.render();
      });
    });
  }
}

customElements.define('theme-switcher', ThemeSwitcher);
```

17.9 Publishing to npm

Prepare your library for npm:

1. package.json configuration:

```
{
  "name": "@myorg/larc-components",
  "version": "1.0.0",
  "description": "LARC component library",
  "type": "module",
  "main": "./index.js",
  "exports": {
    ".": {
      "import": "./index.js",
      "types": "./index.d.ts"
    },
    "./dialog": {
      "import": "./components/core/pan-dialog.js",
      "types": "./components/core/pan-dialog.d.ts"
    },
    "./button": {
      "import": "./components/core/pan-button.js"
    },
    "./themes/*": "./themes/*"
  },
  "files": [
    "components/",
    "themes/",
    "index.js",
    "README.md"
  ],
  "keywords": [
    "web-components",
    "larc",
    "components",
    "ui"
  ],
  "customElements": "custom-elements.json",
  "scripts": {
```

```
    "test": "web-test-runner \"components/**/*.test.js\"",
    "analyze": "wca analyze \"components/**/*.js\" --outFile custom-
        elements.json",
    "prepublishOnly": "npm test && npm run analyze"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/myorg/larc-components"
  },
  "license": "MIT"
}
```

2. Create README.md:

```
# @myorg/larc-components

A collection of accessible, themeable web components built with LARC.

## Installation

```bash
npm install @myorg/larc-components
```

17.10 Usage

17.10.1 Import all components

```
import '@myorg/larc-components';
```
```

17.10.2 Import specific components

```
import '@myorg/larc-components/dialog';
import '@myorg/larc-components/button';
```

17.10.3 Use in HTML

```
<pan-dialog open title="Example">
  <p>Dialog content</p>
</pan-dialog>
```

17.11 Theming

```
import { themeManager } from '@myorg/larc-components/themes/theme-
         manager.js';

// Apply dark theme
themeManager.applyTheme('dark');

// Register custom theme
themeManager.registerTheme('custom', {
  '--color-primary': '#ff0000',
  // ...more tokens
});
```

17.12 Components

- `<pan-dialog>` - Modal dialog

- `<pan-button>` - Button component
- `<pan-card>` - Card container
- More coming soon...

17.13 License

MIT

```
**3. Publish:**

```bash
# Login to npm
npm login

# Publish (first time)
npm publish --access public

# Publish update
npm version patch  # or minor, major
npm publish
```

17.14 Version Management

Follow semantic versioning and maintain a changelog:
```

```
# Changelog

All notable changes to this project will be documented in this file.

## [1.2.0] - 2024-01-15

### Added
- New `<pan-dialog>` component with full accessibility support
- Dark theme support
- CSS parts for style customization

### Changed
- Improved button component focus styles
- Updated default theme colors

### Fixed
- Fixed focus trap in dialog component
- Fixed memory leak in theme manager

## [1.1.0] - 2024-01-01

### Added
- New `<pan-card>` component
- Theme switcher component

### Changed
- Breaking: Renamed `variant` to `type` in button component

### Migration Guide

Update button variant attribute:
```html
<!-- Before -->
<pan-button variant="primary">Click</pan-button>
```
```

```
<!-- After -->
<pan-button type="primary">Click</pan-button>
```

17.15 [1.0.0] - 2023-12-01

- Initial release

```
## Component API Design Patterns

Design consistent, predictable component APIs:

**1. Boolean attributes:**

```javascript
// ✅ Good: Use presence/absence
<pan-dialog open></pan-dialog>
<pan-button disabled></pan-button>

// ❌ Bad: Use string values
<pan-dialog open="true"></pan-dialog>
```

2. Enum attributes:

```
// ✅ Good: Use lowercase, hyphenated
<pan-button variant="primary"></pan-button>
<pan-input type="email"></pan-input>

// ❌ Bad: Use camelCase or weird casing
<pan-button variant="Primary"></pan-button>
```
```

3. Event naming:

```
// ✅ Good: Use present tense for state changes
element.addEventListener('open', () => {});
element.addEventListener('close', () => {});

// ✅ Good: Use past tense for completed actions
element.addEventListener('loaded', () => {});
element.addEventListener('changed', () => {});
```

4. CSS custom properties:

```
// ✅ Good: Namespace and descriptive
--dialog-width
--button-primary-bg
--card-border-radius

// ❌ Bad: Generic or unclear
--width
--bg
--radius
```

5. Slots:

```
// ✅ Good: Named slots for specific content
<slot name="header"></slot>
<slot name="footer"></slot>
<slot></slot>  // default slot

// ❌ Bad: Too many unnamed slots
```

17.16 Troubleshooting Component Libraries

17.16.1 Problem 1: Styles Bleeding Between Components

Symptoms: Components inherit unwanted styles from global CSS or other components.

Cause: Not using Shadow DOM or improperly scoped styles.

Solution:

```
// ✅ Always use Shadow DOM for encapsulation
class MyComponent extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });  // ← Critical!
  }

  connectedCallback() {
    this.shadowRoot.innerHTML = `
      <style>
        /* These styles won't leak out */
        :host {
          display: block;
        }
      </style>
      <div class="content">
        <slot></slot>
      </div>
    `;
  }
}
```

17.16.2 Problem 2: Components Not Updating When Attributes Change

Symptoms: Changing attributes doesn't update the component.

Cause: Not implementing `observedAttributes` or `attributeChangedCallback`.

Solution:

```
class MyComponent extends HTMLElement {
  // ✅ Declare which attributes to watch
  static observedAttributes = ['value', 'disabled'];

  attributeChangedCallback(name, oldValue, newValue) {
    if (oldValue === newValue) return;

    // Handle attribute change
    if (name === 'value') {
      this.updateValue(newValue);
    } else if (name === 'disabled') {
      this.updateDisabledState(newValue !== null);
    }
  }
}
```

17.16.3 Problem 3: Memory Leaks in Components

Symptoms: Memory usage grows over time, especially when components are added/removed.

Cause: Event listeners not cleaned up.

Solution:

```
class MyComponent extends HTMLElement {
  connectedCallback() {
    // Store handler reference
    this._resizeHandler = () => this.handleResize();
    window.addEventListener('resize', this._resizeHandler);

    this._panSubscriptionId = pan.subscribe('data.updated', () => {
      this.refresh();
    });
  }

  disconnectedCallback() {
    // ✅ Clean up everything
    window.removeEventListener('resize', this._resizeHandler);

    if (this._panSubscriptionId) {
      pan.unsubscribe(this._panSubscriptionId);
    }
  }
}
```

17.16.4 Problem 4: Slow Component Registration

Symptoms: Initial page load is slow when many components are imported.

Cause: Importing all components upfront.

Solution:

```
// ❌ Bad: Import everything upfront
import './components/dialog.js';
import './components/button.js';
import './components/card.js';
// ... 50 more imports

// ✅ Good: Lazy load on demand
const componentRegistry = new Map([
  ['pan-dialog', () => import('./components/dialog.js')],
  ['pan-button', () => import('./components/button.js')],
  ['pan-card', () => import('./components/card.js')]
]);

// Auto-load when component is used
const observer = new MutationObserver((mutations) => {
  mutations.forEach(mutation => {
    mutation.addedNodes.forEach(node => {
      if (node.nodeType === 1) {
        const tagName = node.tagName.toLowerCase();
        if (componentRegistry.has(tagName)) {
          componentRegistry.get(tagName)();
          componentRegistry.delete(tagName);  // Load once
        }
      }
    });
  });
});

observer.observe(document.body, {
  childList: true,
  subtree: true
});
```

17.17 Component Library Best Practices

1. **Always Use Shadow DOM**: Encapsulate styles and prevent leakage.
2. **Document Everything**: Use JSDoc comments for all public APIs.
3. **Test Thoroughly**: Unit tests, accessibility tests, and visual regression tests.
4. **Follow Web Standards**: Use standard HTML attributes and patterns when possible.
5. **Make Components Composable**: Small, focused components that work well together.
6. **Provide Customization**: CSS parts, custom properties, and slots for flexibility.
7. **Accessibility First**: Always include ARIA attributes, keyboard navigation, and focus management.
8. **Version Carefully**: Follow semantic versioning and provide migration guides for breaking changes.
9. **Optimize Performance**: Lazy load components, clean up resources, avoid unnecessary re-renders.
10. **Maintain Consistency**: Use consistent naming, patterns, and behaviors across all components.

17.18 Hands-On Exercises

17.18.1 Exercise 1: Build a Toast Notification Component

Create a `<pan-toast>` component that:

- Shows temporary notifications
- Supports different types (success, error, warning, info)

- Auto-dismisses after a timeout
- Stacks multiple toasts
- Has accessible announcements (aria-live)

Bonus: Add slide-in animations and a queue system.

17.18.2 Exercise 2: Create a Component Documentation Site

Build an interactive documentation site for your components that:

- Shows live examples with editable code
- Displays component API (props, events, slots)
- Includes accessibility information
- Has a theme switcher
- Provides copy-paste code snippets

Bonus: Generate documentation from JSDoc comments automatically.

17.18.3 Exercise 3: Implement a Component Testing Suite

Set up comprehensive testing for a component:

- Unit tests for all functionality
- Accessibility tests (keyboard nav, screen readers)
- Visual regression tests with screenshots
- Performance tests (render time)
- Cross-browser testing

Bonus: Integrate tests into CI/CD pipeline.

17.18.4 Exercise 4: Publish a Component Package

Package and publish a component library:

- Set up proper package.json with exports
- Create comprehensive README
- Write CHANGELOG.md
- Publish to npm
- Set up automated versioning and releases

Bonus: Create a landing page with examples and documentation.

17.19 Summary

Building a component library is about more than just writing components—it's about creating a sustainable system:

- **Organize logically**: Group related components, use consistent naming
- **Document thoroughly**: Every component, prop, event, and slot
- **Test comprehensively**: Unit, integration, accessibility, visual
- **Theme consistently**: Design tokens and CSS custom properties
- **Version carefully**: Semantic versioning with clear changelogs
- **Distribute effectively**: npm, CDN, or monorepo

A well-maintained component library accelerates development and ensures consistency across your applications.

17.20 Further Reading

- **Building with LARC - Chapter 15 (Component Patterns)**: Advanced component architecture
- **Building with LARC - Chapter 5 (Shadow DOM)**: Deep dive into encapsulation
- **Building with LARC - Chapter 10 (Accessibility)**: Making components accessible

18 Tooling

While LARC doesn't require a build step, the right tools make development faster and more enjoyable.

18.1 Development Server

A simple development server with live reload:

```
// dev-server.js
const http = require('http');
const fs = require('fs');
const path = require('path');
const WebSocket = require('ws');

const PORT = 3000;
const PUBLIC_DIR = './public';

// HTTP Server
const server = http.createServer((req, res) => {
  let filePath = path.join(PUBLIC_DIR, req.url === '/' ? 'index.html' :
        req.url);

  const ext = path.extname(filePath);
  const contentTypes = {
    '.html': 'text/html',
    '.js': 'application/javascript',
    '.mjs': 'application/javascript',
    '.css': 'text/css',
    '.json': 'application/json'
  };

  fs.readFile(filePath, (err, content) => {
    if (err) {
      res.writeHead(404);
      res.end('Not found');
      return;
    }

    res.writeHead(200, { 'Content-Type': contentTypes[ext] || 'text/
        plain' });
    res.end(content);
  });
});
```

```
// WebSocket for live reload
const wss = new WebSocket.Server({ server });

fs.watch(PUBLIC_DIR, { recursive: true }, () => {
  wss.clients.forEach(client => {
    if (client.readyState === WebSocket.OPEN) {
      client.send('reload');
    }
  });
});

server.listen(PORT, () => console.log(`Dev server at
        http://localhost:${PORT}`));
```

Add live reload to your HTML:

```
<script>
  const ws = new WebSocket('ws://localhost:3000');
  ws.onmessage = () => location.reload();
</script>
```

18.2 VS Code Configuration

Enhance your editor experience:

```
// .vscode/settings.json
{
  "editor.formatOnSave": true,
  "editor.defaultFormatter": "esbenp.prettier-vscode",
  "emmet.includeLanguages": {
    "javascript": "html"
  },
  "files.associations": {
    "*.mjs": "javascript"
  }
}
```

Useful snippets:

```
// .vscode/snippets/larc.code-snippets
{
  "LARC Component": {
    "prefix": "larc",
    "body": [
      "class ${1:ComponentName} extends HTMLElement {",
      "  constructor() {",
      "    super();",
      "    this.attachShadow({ mode: 'open' });",
      "  }",
      "",
      "  connectedCallback() {",
      "    this.render();",
      "  }",
      "",
      "  render() {",
      "    this.shadowRoot.innerHTML = `",
      "      <style>",
      "        :host { display: block; }",
      "      </style>",
      "      <div>$2</div>",
      "    `;",
      "  }",
      "}",
      "",
      "customElements.define('${3:component-name}', ${1:ComponentName});"
    ]
  }
}
```

18.3 ESLint Configuration

Lint your code for consistency:

```
// eslint.config.js
export default [
  {
    files: ['**/*.js', '**/*.mjs'],
    rules: {
      'no-unused-vars': 'warn',
      'no-console': ['warn', { allow: ['warn', 'error'] }],
      'prefer-const': 'error',
      'no-var': 'error'
    }
  }
];
```

18.4 Browser DevTools

Chrome DevTools has excellent Web Component support:

- **Elements panel**: Inspect shadow DOM by clicking the `#shadow-root` toggle
- **Console**: Access element's shadow root with `$0.shadowRoot`
- **Network panel**: Monitor fetch requests and WebSocket connections
- **Performance panel**: Profile render performance
- **Application panel**: Inspect localStorage, sessionStorage, IndexedDB

18.5 Debugging PAN Bus

Add a debug utility:

```
// pan-debug.js
pan.subscribe('*', (data, topic) => {
  console.log(`[PAN] ${topic}`, data);
});
```

Or use the LARC DevTools extension for a visual message inspector.

18.6 Complete Development Environment Setup

Let's set up a full-featured development environment:

1. Project structure:

```
my-larc-project/
├── public/
|   ├── index.html
|   ├── app.js
|   ├── components/
|   └── styles/
├── tests/
|   ├── unit/
|   └── e2e/
├── .vscode/
|   ├── settings.json
|   ├── extensions.json
|   └── snippets/
├── .eslintrc.js
├── .prettierrc
├── package.json
└── dev-server.js
```

2. Package.json with dev scripts:

```
{
  "name": "my-larc-app",
  "version": "1.0.0",
  "type": "module",
  "scripts": {
    "dev": "node dev-server.js",
    "test": "web-test-runner \"tests/**/*.test.js\"",
    "test:watch": "npm test -- --watch",
    "lint": "eslint '**/*.{js,mjs}'",
    "lint:fix": "eslint '**/*.{js,mjs}' --fix",
    "format": "prettier --write '**/*.{js,mjs,html,css,json,md}'",
    "format:check": "prettier --check '**/*.{js,mjs,html,css,json,md}'",
    "analyze": "wca analyze 'public/components/**/*.js' --outFile custom-
        elements.json"
  },
  "devDependencies": {
    "@open-wc/testing": "^4.0.0",
    "@web/dev-server": "^0.4.0",
    "@web/test-runner": "^0.18.0",
    "eslint": "^8.0.0",
    "prettier": "^3.0.0",
    "web-component-analyzer": "^2.0.0",
    "ws": "^8.0.0"
  }
}
```

3. Complete dev server with HMR:

```
// dev-server.js
import { createServer } from 'http';
import { readFile } from 'fs/promises';
import { watch } from 'fs';
import { join, extname } from 'path';
import { WebSocketServer } from 'ws';

const PORT = 3000;
const PUBLIC_DIR = './public';

const contentTypes = {
  '.html': 'text/html; charset=utf-8',
  '.js': 'application/javascript; charset=utf-8',
  '.mjs': 'application/javascript; charset=utf-8',
  '.css': 'text/css; charset=utf-8',
  '.json': 'application/json; charset=utf-8',
  '.png': 'image/png',
  '.jpg': 'image/jpeg',
  '.svg': 'image/svg+xml',
  '.ico': 'image/x-icon'
};

// HTTP Server
const server = createServer(async (req, res) => {
  try {
    // Handle SPA routing - serve index.html for non-file routes
    let filePath = join(PUBLIC_DIR, req.url === '/' ? 'index.html' :
        req.url);

    // Try to read the file
    let content;
    try {
      content = await readFile(filePath);
    } catch (err) {
```

```
    // If file not found and URL doesn't have extension, serve
       index.html
    if (!extname(req.url)) {
      filePath = join(PUBLIC_DIR, 'index.html');
      content = await readFile(filePath);
    } else {
      throw err;
    }
  }

  const ext = extname(filePath);
  const contentType = contentTypes[ext] || 'application/octet-stream';

  // Inject live reload script for HTML files
  if (ext === '.html') {
    const liveReloadScript = `
      <script>
        (function() {
          const ws = new WebSocket('ws://localhost:${PORT}');
          ws.onmessage = (event) => {
            if (event.data === 'reload') {
              console.log('[Dev Server] Reloading...');
              location.reload();
            } else if (event.data.startsWith('hmr:')) {
              const module = event.data.substring(4);
              console.log('[Dev Server] Hot reload:', module);
              // Implement HMR logic here
            }
          };
          ws.onerror = () => console.error('[Dev Server] WebSocket
      error');
          ws.onclose = () => console.log('[Dev Server] Disconnected');
        })();
      </script>
    `;
```

```
      content = Buffer.from(
        content.toString().replace('</body>',
        `${liveReloadScript}</body>`)
      );
    }

    res.writeHead(200, {
      'Content-Type': contentType,
      'Cache-Control': 'no-cache',
      'Access-Control-Allow-Origin': '*'
    });
    res.end(content);
  } catch (err) {
    console.error('Server error:', err);
    res.writeHead(404, { 'Content-Type': 'text/html' });
    res.end(`
      <!DOCTYPE html>
      <html>
        <head><title>404 Not Found</title></head>
        <body>
          <h1>404 - Not Found</h1>
          <p>${req.url}</p>
        </body>
      </html>
    `);
  }
});

// WebSocket for live reload
const wss = new WebSocketServer({ server });

wss.on('connection', (ws) => {
  console.log('Client connected');
  ws.on('close', () => console.log('Client disconnected'));
});
```

```
// File watcher
watch(PUBLIC_DIR, { recursive: true }, (eventType, filename) => {
  if (filename) {
    console.log(`File changed: ${filename}`);

    // Notify all connected clients
    wss.clients.forEach(client => {
      if (client.readyState === 1) { // WebSocket.OPEN
        if (filename.endsWith('.js') || filename.endsWith('.mjs')) {
          client.send(`hmr:${filename}`);
        } else {
          client.send('reload');
        }
      }
    });
  }
});

server.listen(PORT, () => {
  console.log(`
🚀 Dev server running at http://localhost:${PORT}
📁 Serving: ${PUBLIC_DIR}
🔥 Live reload enabled
  `);
});
```

18.7 Advanced VS Code Configuration

Complete settings.json:

```
{
  "editor.formatOnSave": true,
  "editor.defaultFormatter": "esbenp.prettier-vscode",
  "editor.codeActionsOnSave": {
    "source.fixAll.eslint": "explicit"
  },
  "editor.quickSuggestions": {
    "strings": true
  },
  "emmet.includeLanguages": {
    "javascript": "html"
  },
  "files.associations": {
    "*.mjs": "javascript"
  },
  "files.exclude": {
    "**/.git": true,
    "**/node_modules": true,
    "**/.DS_Store": true
  },
  "search.exclude": {
    "**/node_modules": true,
    "**/dist": true,
    "**/.git": true
  },
  "javascript.preferences.quoteStyle": "single",
  "javascript.suggest.autoImports": true,
  "javascript.updateImportsOnFileMove.enabled": "always",
  "css.lint.unknownAtRules": "ignore"
}
```

Recommended extensions:

```
// .vscode/extensions.json
{
  "recommendations": [
    "dbaeumer.vscode-eslint",
    "esbenp.prettier-vscode",
    "bradlc.vscode-tailwindcss",
    "ritwickdey.liveserver",
    "christian-kohler.path-intellisense",
    "zignd.html-css-class-completion",
    "formulahendry.auto-rename-tag"
  ]
}
```

Custom snippets:

```
// .vscode/snippets/larc.code-snippets
{
  "LARC Component with Shadow DOM": {
    "prefix": "larc-component",
    "body": [
      "/**",
      " * ${1:ComponentName} - ${2:Component description}",
      " * @element ${3:component-name}",
      " */",
      "class ${1:ComponentName} extends HTMLElement {",
      "  constructor() {",
      "    super();",
      "    this.attachShadow({ mode: 'open' });",
      "  }",
      "",
      "  connectedCallback() {",
      "    this.render();",
      "  }",
      "",
      "  disconnectedCallback() {",
      "    // Clean up",
      "  }",
      "",
      "  render() {",
      "    this.shadowRoot.innerHTML = `",
      "      <style>",
      "        :host {",
      "          display: block;",
      "        }",
      "      </style>",
      "      <div class=\"${3:component-name}\">",
      "        $4",
      "      </div>",
      "    `;",
```

```
      "  }",
      "}",
      "",
      "customElements.define('${3:component-name}', ${1:ComponentName});"
    ],
    "description": "Create a LARC component with Shadow DOM"
  },
  "PAN Subscribe": {
    "prefix": "pan-sub",
    "body": [
      "this.subscription = pan.subscribe('${1:topic}', (data) => {",
      "  $2",
      "});"
    ],
    "description": "Subscribe to PAN topic"
  },
  "PAN Publish": {
    "prefix": "pan-pub",
    "body": [
      "pan.publish('${1:topic}', {",
      "  $2",
      "});"
    ],
    "description": "Publish to PAN topic"
  }
}
```

18.8 ESLint and Prettier Configuration

Complete ESLint config:

```
// eslint.config.js
export default [
  {
    files: ['**/*.{js,mjs}'],
    languageOptions: {
      ecmaVersion: 2022,
      sourceType: 'module',
      globals: {
        window: 'readonly',
        document: 'readonly',
        customElements: 'readonly',
        HTMLElement: 'readonly',
        console: 'readonly',
        fetch: 'readonly',
        localStorage: 'readonly',
        sessionStorage: 'readonly',
        navigator: 'readonly',
        URL: 'readonly',
        URLSearchParams: 'readonly'
      }
    },
    rules: {
      // Best practices
      'no-unused-vars': ['warn', { argsIgnorePattern: '^_' }],
      'no-console': ['warn', { allow: ['warn', 'error', 'info'] }],
      'prefer-const': 'error',
      'no-var': 'error',
      'eqeqeq': ['error', 'always'],
      'no-eval': 'error',
      'no-implied-eval': 'error',

      // Formatting (let Prettier handle most)
      'semi': ['error', 'always'],
      'quotes': ['error', 'single', { avoidEscape: true }],
```

```
      'comma-dangle': ['error', 'never'],

      // ES6+
      'arrow-spacing': 'error',
      'prefer-arrow-callback': 'warn',
      'prefer-template': 'warn',
      'template-curly-spacing': 'error',
      'object-shorthand': 'warn',

      // Async
      'no-async-promise-executor': 'error',
      'require-await': 'warn',

      // Custom Elements
      'no-constructor-return': 'error'
    }
  },
  {
    files: ['**/*.test.{js,mjs}'],
    languageOptions: {
      globals: {
        describe: 'readonly',
        it: 'readonly',
        expect: 'readonly',
        beforeEach: 'readonly',
        afterEach: 'readonly'
      }
    }
  }
];
```

Prettier configuration:

```
// .prettierrc
{
  "semi": true,
  "singleQuote": true,
  "trailingComma": "none",
  "printWidth": 100,
  "tabWidth": 2,
  "useTabs": false,
  "arrowParens": "avoid",
  "endOfLine": "lf"
}
```

Ignore files:

```
# .prettierignore
node_modules
dist
build
coverage
*.min.js
package-lock.json
```

18.9 Advanced Debugging Techniques

1. Source maps for debugging:

Although LARC doesn't need transpilation, you can still use source maps for development:

```
// Add inline source maps during development
if (import.meta.env?.MODE === 'development') {
  Error.stackTraceLimit = Infinity;
}
```

2. Component debugger:

```
// debug-component.js
export function debugComponent(ComponentClass) {
  const originalConnected = ComponentClass.prototype.connectedCallback;
  const originalDisconnected =
        ComponentClass.prototype.disconnectedCallback;
  const originalAttributeChanged =
        ComponentClass.prototype.attributeChangedCallback;

  ComponentClass.prototype.connectedCallback = function(...args) {
    console.log(`[Connected] <${this.tagName.toLowerCase()}>`, this);
    return originalConnected?.apply(this, args);
  };

  ComponentClass.prototype.disconnectedCallback = function(...args) {
    console.log(`[Disconnected] <${this.tagName.toLowerCase()}>`, this);
    return originalDisconnected?.apply(this, args);
  };

  ComponentClass.prototype.attributeChangedCallback = function(name,
        old, val) {
    console.log(`[Attribute] <${this.tagName.toLowerCase()}> ${name}:
        ${old} → ${val}`);
    return originalAttributeChanged?.call(this, name, old, val);
  };

  return ComponentClass;
}

// Usage
@debugComponent
class MyComponent extends HTMLElement {
  // ...
}
```

3. Performance profiling:

```
// performance-tracker.js
class PerformanceTracker {
  constructor(name) {
    this.name = name;
    this.marks = new Map();
  }

  start(label) {
    const markName = `${this.name}:${label}:start`;
    performance.mark(markName);
    this.marks.set(label, markName);
  }

  end(label) {
    const startMark = this.marks.get(label);
    if (!startMark) {
      console.warn(`No start mark for ${label}`);
      return;
    }

    const endMark = `${this.name}:${label}:end`;
    performance.mark(endMark);

    const measureName = `${this.name}:${label}`;
    performance.measure(measureName, startMark, endMark);

    const measure = performance.getEntriesByName(measureName)[0];
    console.log(`   ${measureName}: ${measure.duration.toFixed(2)}ms`);

    // Clean up
    performance.clearMarks(startMark);
    performance.clearMarks(endMark);
    performance.clearMeasures(measureName);
    this.marks.delete(label);
```

```
  }
}

// Usage in component
class MyComponent extends HTMLElement {
  constructor() {
    super();
    this.perf = new PerformanceTracker('MyComponent');
  }

  connectedCallback() {
    this.perf.start('render');
    this.render();
    this.perf.end('render');
  }
}
```

4. Network debugging:

```
// network-debugger.js
const originalFetch = window.fetch;

window.fetch = function(...args) {
  const startTime = performance.now();
  const [url, options] = args;

  console.log(`[Fetch] →`, {
    url,
    method: options?.method || 'GET',
    headers: options?.headers,
    body: options?.body
  });

  return originalFetch.apply(this, args)
    .then(response => {
      const duration = performance.now() - startTime;
      console.log(`[Fetch] ← ${response.status}
        (${duration.toFixed(0)}ms)`, url);
      return response;
    })
    .catch(error => {
      const duration = performance.now() - startTime;
      console.error(`[Fetch] ✗ (${duration.toFixed(0)}ms)`, url, error);
      throw error;
    });
};
```

18.10 PAN Bus DevTools

Build a visual inspector for PAN messages:

```
// pan-devtools.js
class PanDevTools extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.messages = [];
    this.maxMessages = 100;
    this.filter = '';
  }

  connectedCallback() {
    this.render();
    this.setupPanMonitoring();
  }

  setupPanMonitoring() {
    // Subscribe to all topics
    pan.subscribe('*', (data, topic) => {
      this.logMessage({
        timestamp: new Date(),
        topic,
        data,
        type: 'received'
      });
    });

    // Intercept publishes (if PAN client supports it)
    const originalPublish = pan.publish;
    pan.publish = (topic, data) => {
      this.logMessage({
        timestamp: new Date(),
        topic,
        data,
        type: 'sent'
```

```
    });
    return originalPublish.call(pan, topic, data);
  };
}

logMessage(message) {
  this.messages.unshift(message);
  if (this.messages.length > this.maxMessages) {
    this.messages.pop();
  }
  this.renderMessages();
}

render() {
  this.shadowRoot.innerHTML = `
    <style>
      :host {
        position: fixed;
        bottom: 0;
        right: 0;
        width: 400px;
        max-height: 600px;
        background: #1e1e1e;
        color: #d4d4d4;
        font-family: 'Monaco', 'Courier New', monospace;
        font-size: 12px;
        box-shadow: 0 -2px 10px rgba(0,0,0,0.3);
        display: flex;
        flex-direction: column;
        z-index: 10000;
      }

      .header {
        padding: 8px;
```

```
  background: #252526;
  border-bottom: 1px solid #3e3e42;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.title {
  font-weight: bold;
  color: #4ec9b0;
}

.controls {
  display: flex;
  gap: 8px;
}

button {
  background: #3e3e42;
  border: none;
  color: #d4d4d4;
  padding: 4px 8px;
  border-radius: 3px;
  cursor: pointer;
  font-size: 11px;
}

button:hover {
  background: #4e4e52;
}

.filter {
  padding: 8px;
  border-bottom: 1px solid #3e3e42;
```

```
}

input {
  width: 100%;
  padding: 4px 8px;
  background: #3c3c3c;
  border: 1px solid #3e3e42;
  color: #d4d4d4;
  border-radius: 3px;
  font-size: 11px;
}

.messages {
  flex: 1;
  overflow-y: auto;
  padding: 8px;
}

.message {
  margin-bottom: 8px;
  padding: 8px;
  background: #252526;
  border-left: 3px solid #4ec9b0;
  border-radius: 3px;
}

.message.sent {
  border-left-color: #569cd6;
}

.message-header {
  display: flex;
  justify-content: space-between;
  margin-bottom: 4px;
```

```
    }

    .topic {
      color: #4ec9b0;
      font-weight: bold;
    }

    .timestamp {
      color: #858585;
      font-size: 10px;
    }

    .data {
      color: #ce9178;
      white-space: pre-wrap;
      word-break: break-all;
    }

    .type {
      display: inline-block;
      padding: 2px 6px;
      border-radius: 3px;
      font-size: 10px;
      margin-left: 8px;
    }

    .type.sent {
      background: #569cd6;
    }

    .type.received {
      background: #4ec9b0;
    }
  </style>
```

```
      <div class="header">
        <span class="title">PAN DevTools</span>
        <div class="controls">
          <button id="clear-btn">Clear</button>
          <button id="close-btn">Close</button>
        </div>
      </div>

      <div class="filter">
        <input type="text" id="filter-input" placeholder="Filter by
        topic...">
      </div>

      <div class="messages" id="messages"></div>
    `;

    // Setup event listeners
    this.shadowRoot.getElementById('clear-
        btn').addEventListener('click', () => {
      this.messages = [];
      this.renderMessages();
    });

    this.shadowRoot.getElementById('close-
        btn').addEventListener('click', () => {
      this.remove();
    });

    this.shadowRoot.getElementById('filter-
        input').addEventListener('input', (e) => {
      this.filter = e.target.value.toLowerCase();
      this.renderMessages();
    });
  }
```

```
  renderMessages() {
    const messagesEl = this.shadowRoot.getElementById('messages');
    if (!messagesEl) return;

    const filtered = this.filter
      ? this.messages.filter(m =>
        m.topic.toLowerCase().includes(this.filter))
      : this.messages;

    messagesEl.innerHTML = filtered.map(msg => `
      <div class="message ${msg.type}">
        <div class="message-header">
          <span class="topic">${msg.topic}</span>
          <span
        class="timestamp">${msg.timestamp.toLocaleTimeString()}.${msg.timestamp.getM


        </div>
        <div>
          <span class="type ${msg.type}">${msg.type}</span>
        </div>
        <div class="data">${JSON.stringify(msg.data, null, 2)}</div>
      </div>
    `).join('');
  }
}

customElements.define('pan-devtools', PanDevTools);

// Add to page with keyboard shortcut
document.addEventListener('keydown', (e) => {
  if (e.ctrlKey && e.shiftKey && e.key === 'P') {
    if (!document.querySelector('pan-devtools')) {
      document.body.appendChild(document.createElement('pan-devtools'));
    }
  }
```

```
});
```

18.11 Optional Build Tools

While LARC doesn't need a build step, sometimes you want to bundle for production:

Using esbuild for optimization:

```
// build.js
import * as esbuild from 'esbuild';

await esbuild.build({
  entryPoints: ['public/app.js'],
  bundle: true,
  minify: true,
  sourcemap: true,
  target: ['es2020'],
  outfile: 'dist/app.min.js',
  format: 'esm',
  splitting: false,
  treeShaking: true
});

console.log('✅ Build complete');
```

Using Rollup:

```
// rollup.config.js
import { terser } from 'rollup-plugin-terser';
import resolve from '@rollup/plugin-node-resolve';

export default {
  input: 'public/app.js',
  output: {
    file: 'dist/app.min.js',
    format: 'esm',
    sourcemap: true
  },
  plugins: [
    resolve(),
    terser()
  ]
};
```

18.12 Troubleshooting Tooling Issues

18.12.1 Problem 1: Live Reload Not Working

Symptoms: Changes aren't reflected after saving files.

Diagnosis:

```
# Check if dev server is running
ps aux | grep node

# Check WebSocket connection in browser console
# Should see WebSocket connected message
```

Solution:

```
// Ensure WebSocket script is injected
// Check browser console for WebSocket errors
// Verify port 3000 is not blocked by firewall

// Alternative: Use browser extension like LiveReload
```

18.12.2 Problem 2: ESLint Errors in VS Code

Symptoms: Red squiggly lines everywhere, but code works fine.

Cause: ESLint configuration mismatch.

Solution:

```
# Reinstall ESLint
npm install --save-dev eslint

# Restart VS Code ESLint server
# Cmd+Shift+P → "ESLint: Restart ESLint Server"

# Check ESLint output panel
# View → Output → Select "ESLint" from dropdown
```

18.12.3 Problem 3: Import Paths Not Resolving

Symptoms: VS Code shows import errors, but browser loads fine.

Cause: VS Code doesn't understand import maps.

Solution:

```
// jsconfig.json - Help VS Code understand paths
{
  "compilerOptions": {
    "module": "esnext",
    "moduleResolution": "bundler",
    "baseUrl": "./public",
    "paths": {
      "@components/*": ["components/*"],
      "@utils/*": ["utils/*"]
    }
  },
  "include": ["public/**/*"],
  "exclude": ["node_modules", "dist"]
}
```

18.12.4 Problem 4: Shadow DOM Not Visible in DevTools

Symptoms: Can't inspect Shadow DOM elements.

Cause: Shadow DOM panel not enabled.

Solution:

1. Open Chrome DevTools
2. Go to Settings (gear icon)
3. Preferences → Elements → Enable "Show user agent shadow DOM"
4. Now you can see `#shadow-root` in Elements panel

18.13 Tooling Best Practices

1. **Use a Dev Server**: Never open HTML files directly—use a local server for proper CORS and module loading.

2. **Enable Live Reload**: Instant feedback speeds up development significantly.
3. **Configure Your Editor**: Proper linting, formatting, and snippets save hours of work.
4. **Debug with DevTools**: Master the Elements, Console, Network, and Performance panels.
5. **Monitor PAN Messages**: Use a visual inspector to understand message flow.
6. **Profile Performance**: Use Performance API and DevTools to find bottlenecks.
7. **Automate Formatting**: Let Prettier handle code style—focus on logic.
8. **Test Continuously**: Run tests in watch mode during development.
9. **Document As You Go**: JSDoc comments help VS Code provide better autocomplete.
10. **Keep It Simple**: Don't over-tool. Start minimal and add tools as needed.

18.14 Hands-On Exercises

18.14.1 Exercise 1: Set Up Complete Dev Environment

Configure a full development environment with:

- Dev server with live reload
- VS Code with all recommended extensions
- ESLint and Prettier configured
- Custom snippets for LARC components
- Git hooks for pre-commit linting

Bonus: Add automated testing on file save.

18.14.2 Exercise 2: Build a PAN Message Inspector

Create a browser extension or dev panel that:

- Shows all PAN messages in real-time
- Filters messages by topic pattern
- Records and exports message history
- Shows message timing and frequency
- Highlights retained vs. transient messages

Bonus: Add ability to publish test messages.

18.14.3 Exercise 3: Create Custom VS Code Extension

Build a VS Code extension that:

- Generates LARC component boilerplate
- Validates component structure
- Provides autocomplete for PAN topics
- Shows component documentation on hover
- Refactors component names across files

Bonus: Publish to VS Code marketplace.

18.14.4 Exercise 4: Implement Advanced Debugging

Set up comprehensive debugging tools:

- Component lifecycle logger
- Performance profiler for renders
- Network request interceptor
- Error boundary with stack traces
- State inspector for component properties

Bonus: Create a dashboard showing all metrics.

18.15 Summary

Good tooling makes LARC development faster and more enjoyable:

- **Dev server**: Live reload and hot module replacement
- **Editor setup**: VS Code configuration, extensions, snippets
- **Linting**: ESLint for code quality, Prettier for formatting
- **Debugging**: DevTools mastery, PAN inspector, performance tracking
- **Automation**: npm scripts, git hooks, CI integration

Start simple and add tools as your project grows. The beauty of LARC is you can be productive with just a text editor and a browser—everything else is optional.

18.16 Further Reading

- **Building with LARC - Chapter 20 (DevTools)**: Advanced debugging techniques
- **Building with LARC - Chapter 11 (Best Practices)**: Development workflow patterns
- **Building with LARC - Chapter 14 (Testing)**: Testing infrastructure setup

19 Real-World Applications

Theory only takes you so far. Let's examine how LARC principles apply to real applications.

19.1 Case Study: E-Commerce Platform

An online store built with LARC demonstrates the architecture at scale.

19.1.1 Architecture Overview

```
store/
├── index.html
├── components/
|   ├── product-card.js
|   ├── product-grid.js
|   ├── shopping-cart.js
|   ├── cart-item.js
|   ├── checkout-form.js
|   └── order-confirmation.js
├── services/
|   ├── cart-service.js
|   ├── product-service.js
|   └── order-service.js
└── styles/
    └── main.css
```

19.1.2 Product Catalog

The product grid loads data and renders cards:

```
// product-grid.js
class ProductGrid extends HTMLElement {
  async connectedCallback() {
    this.innerHTML = '<p>Loading products...</p>';

    try {
      const products = await productService.getAll();
      this.render(products);
    } catch (error) {
      this.innerHTML = `<p class="error">Failed to load products</p>`;
    }
  }

  render(products) {
    this.innerHTML = `
      <div class="grid">
        ${products.map(p => `
          <product-card
            product-id="${p.id}"
            name="${p.name}"
            price="${p.price}"
            image="${p.image}">
          </product-card>
        `).join('')}
      </div>
    `;
  }
}
```

19.1.3 Shopping Cart

The cart subscribes to add-to-cart events and persists state:

```
// shopping-cart.js
class ShoppingCart extends HTMLElement {
  constructor() {
    super();
    this.items = JSON.parse(localStorage.getItem('cart')) || [];
  }

  connectedCallback() {
    pan.subscribe('cart.add', ({ product }) => {
      this.addItem(product);
    });

    pan.subscribe('cart.remove', ({ productId }) => {
      this.removeItem(productId);
    });

    this.render();
  }

  addItem(product) {
    const existing = this.items.find(i => i.id === product.id);
    if (existing) {
      existing.quantity++;
    } else {
      this.items.push({ ...product, quantity: 1 });
    }
    this.save();
    this.render();
  }

  removeItem(productId) {
    this.items = this.items.filter(i => i.id !== productId);
    this.save();
    this.render();
```

```
  }

  save() {
    localStorage.setItem('cart', JSON.stringify(this.items));
    pan.publish('cart.updated', { items: this.items, total: this.total
        });
  }

  get total() {
    return this.items.reduce((sum, i) => sum + i.price * i.quantity, 0);
  }

  render() {
    this.innerHTML = `
      <h2>Cart (${this.items.length} items)</h2>
      ${this.items.map(item => `
        <cart-item
          product-id="${item.id}"
          name="${item.name}"
          price="${item.price}"
          quantity="${item.quantity}">
        </cart-item>
      `).join('')}
      <p class="total">Total: $${this.total.toFixed(2)}</p>
      <button class="checkout-btn">Checkout</button>
    `;
  }
}
```

19.2 Case Study: Dashboard Application

A data dashboard shows real-time metrics with role-based access.

19.2.1 Real-Time Updates

WebSocket messages update charts automatically:

```
// metrics-chart.js
class MetricsChart extends HTMLElement {
  connectedCallback() {
    this.data = [];

    pan.subscribe('ws.message.metrics', ({ value, timestamp }) => {
      this.data.push({ value, timestamp });
      if (this.data.length > 100) this.data.shift();
      this.render();
    });

    this.render();
  }

  render() {
    // Render chart using canvas or SVG
    const canvas = this.querySelector('canvas') ||
        document.createElement('canvas');
    if (!this.contains(canvas)) this.appendChild(canvas);

    const ctx = canvas.getContext('2d');
    // ... draw chart
  }
}
```

19.2.2 Role-Based Views

Different users see different widgets:

```
// dashboard-page.js
class DashboardPage extends HTMLElement {
  connectedCallback() {
    const user = auth.getCurrentUser();

    this.innerHTML = `
      <h1>Dashboard</h1>

      <div class="widgets">
        <metrics-chart></metrics-chart>
        <recent-activity></recent-activity>

        ${rbac.can(user, 'view-analytics') ? `
          <analytics-panel></analytics-panel>
        ` : ''}

        ${rbac.can(user, 'manage-users') ? `
          <user-management></user-management>
        ` : ''}
      </div>
    `;
  }
}
```

19.3 Lessons Learned

Building real applications with LARC teaches valuable lessons:

1. **Start simple**: Begin with basic components and add complexity as needed.
2. **Use the PAN bus liberally**: It's cheap and powerful. When in doubt, publish an event.
3. **Embrace the platform**: Native APIs are well-optimized. Use fetch, not axios. Use

template literals, not a template library.

4. **Think in components**: Small, focused components are easier to test, reuse, and understand.
5. **Test early**: Writing tests as you build prevents painful debugging later.
6. **Profile before optimizing**: Measure performance before assuming where bottlenecks are.
7. **Document as you go**: Future you will thank present you.
8. **Progressive enhancement**: Build core functionality first, then enhance for modern browsers.

19.3.1 Complete Checkout Flow

Let's implement the full checkout process:

```
// checkout-form.js
class CheckoutForm extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
  }

  connectedCallback() {
    this.render();
    this.setupEventListeners();
  }

  render() {
    this.shadowRoot.innerHTML = `
      <style>
        form {
          max-width: 600px;
          margin: 0 auto;
        }

        .section {
          margin-bottom: 2rem;
          padding: 1.5rem;
          border: 1px solid #e0e0e0;
          border-radius: 8px;
        }

        .field {
          margin-bottom: 1rem;
        }

        label {
          display: block;
          margin-bottom: 0.5rem;
```

```
      font-weight: 600;
    }

    input, select {
      width: 100%;
      padding: 0.5rem;
      border: 1px solid #ccc;
      border-radius: 4px;
    }

    .error {
      color: #dc3545;
      font-size: 0.875rem;
      margin-top: 0.25rem;
    }

    button {
      width: 100%;
      padding: 1rem;
      background: #0066cc;
      color: white;
      border: none;
      border-radius: 4px;
      font-size: 1rem;
      cursor: pointer;
    }

    button:disabled {
      opacity: 0.5;
      cursor: not-allowed;
    }
  </style>

  <form id="checkout-form">
```

```
        <div class="section">
          <h2>Shipping Information</h2>
          <div class="field">
            <label for="email">Email</label>
            <input type="email" id="email" required>
            <div class="error" id="email-error"></div>
          </div>
          <div class="field">
            <label for="name">Full Name</label>
            <input type="text" id="name" required>
          </div>
          <div class="field">
            <label for="address">Address</label>
            <input type="text" id="address" required>
          </div>
          <div class="field">
            <label for="city">City</label>
            <input type="text" id="city" required>
          </div>
          <div class="field">
            <label for="zip">ZIP Code</label>
            <input type="text" id="zip" pattern="[0-9]{5}" required>
          </div>
        </div>

        <div class="section">
          <h2>Payment Information</h2>
          <div class="field">
            <label for="card-number">Card Number</label>
            <input type="text" id="card-number" pattern="[0-9]{16}"
        required>
          </div>
          <div class="field">
            <label for="expiry">Expiry Date</label>
            <input type="text" id="expiry" placeholder="MM/YY" required>
```

```
          </div>
          <div class="field">
            <label for="cvv">CVV</label>
            <input type="text" id="cvv" pattern="[0-9]{3}" required>
          </div>
        </div>

        <button type="submit" id="submit-btn">Place Order</button>
      </form>
    `;
  }

  setupEventListeners() {
    const form = this.shadowRoot.getElementById('checkout-form');
    const submitBtn = this.shadowRoot.getElementById('submit-btn');

    form.addEventListener('submit', async (e) => {
      e.preventDefault();

      if (!this.validate()) {
        return;
      }

      submitBtn.disabled = true;
      submitBtn.textContent = 'Processing...';

      try {
        const order = this.getFormData();
        const result = await orderService.submit(order);

        pan.publish('order.completed', { orderId: result.id });
        pan.publish('router.navigate', { path: `/order-
        confirmation/${result.id}` });
      } catch (error) {
        alert('Order failed: ' + error.message);
```

```
      } finally {
        submitBtn.disabled = false;
        submitBtn.textContent = 'Place Order';
      }
    });
  }

  validate() {
    // Implement validation logic
    return true;
  }

  getFormData() {
    const form = this.shadowRoot.getElementById('checkout-form');
    return {
      email: form.email.value,
      name: form.name.value,
      address: form.address.value,
      city: form.city.value,
      zip: form.zip.value,
      payment: {
        cardNumber: form['card-number'].value,
        expiry: form.expiry.value,
        cvv: form.cvv.value
      }
    };
  }
}

customElements.define('checkout-form', CheckoutForm);
```

19.3.2 Inventory Management

Admin panel for managing products:

```
// admin-inventory.js
class AdminInventory extends HTMLElement {
  async connectedCallback() {
    this.products = await productService.getAll();
    this.render();
  }

  render() {
    this.innerHTML = `
      <div class="admin-panel">
        <h1>Inventory Management</h1>
        <button class="add-product-btn">Add New Product</button>

        <table class="inventory-table">
          <thead>
            <tr>
              <th>ID</th>
              <th>Name</th>
              <th>Price</th>
              <th>Stock</th>
              <th>Status</th>
              <th>Actions</th>
            </tr>
          </thead>
          <tbody>
            ${this.products.map(p => `
              <tr>
                <td>${p.id}</td>
                <td>${p.name}</td>
                <td>$${p.price}</td>
                <td>${p.stock}</td>
                <td>
                  <span class="badge ${p.stock > 0 ? 'in-stock' : 'out-
        of-stock'}">
```

```
                  ${p.stock > 0 ? 'In Stock' : 'Out of Stock'}
                </span>
              </td>
              <td>
                <button
      onclick="this.getRootNode().host.editProduct(${p.id})">
                  Edit
                </button>
                <button
      onclick="this.getRootNode().host.deleteProduct(${p.id})">
                  Delete
                </button>
              </td>
            </tr>
          `).join('')}
        </tbody>
      </table>
    </div>
  `;
}

async editProduct(id) {
  const product = this.products.find(p => p.id === id);
  // Show edit dialog
  pan.publish('dialog.open', {
    component: 'product-edit-form',
    data: product
  });
}

async deleteProduct(id) {
  if (!confirm('Are you sure?')) return;

  await productService.delete(id);
  this.products = this.products.filter(p => p.id !== id);
```

```
    this.render();
  }
}

customElements.define('admin-inventory', AdminInventory);
```

19.4 Case Study: Blog/CMS Application

A content management system demonstrates LARC's flexibility for content-heavy applications.

19.4.1 Architecture

```
blog/
├── index.html
├── components/
|   ├── article-editor.js
|   ├── article-list.js
|   ├── article-card.js
|   ├── markdown-renderer.js
|   └── tag-selector.js
├── services/
|   ├── content-service.js
|   └── media-service.js
└── admin/
    ├── dashboard.js
    ├── editor.js
    └── settings.js
```

19.4.2 Rich Text Editor

```
// article-editor.js
class ArticleEditor extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    this.article = { title: '', content: '', tags: [] };
  }

  connectedCallback() {
    pan.subscribe('article.load', ({ article }) => {
      this.article = article;
      this.render();
    });

    this.render();
    this.setupAutoSave();
  }

  render() {
    this.shadowRoot.innerHTML = `
      <style>
        .editor {
          max-width: 900px;
          margin: 0 auto;
          padding: 2rem;
        }

        .title-input {
          width: 100%;
          font-size: 2rem;
          font-weight: bold;
          border: none;
          border-bottom: 2px solid #e0e0e0;
          padding: 0.5rem 0;
```

```
  margin-bottom: 2rem;
}

.content-editor {
  min-height: 400px;
  border: 1px solid #e0e0e0;
  border-radius: 4px;
  padding: 1rem;
  font-family: monospace;
}

.toolbar {
  display: flex;
  gap: 0.5rem;
  margin-bottom: 1rem;
  padding: 0.5rem;
  background: #f5f5f5;
  border-radius: 4px;
}

.toolbar button {
  padding: 0.5rem 1rem;
  background: white;
  border: 1px solid #ccc;
  border-radius: 4px;
  cursor: pointer;
}

.save-status {
  text-align: right;
  color: #666;
  font-size: 0.875rem;
  margin-top: 1rem;
}
```

```
      </style>

      <div class="editor">
        <input
          type="text"
          class="title-input"
          placeholder="Article Title"
          value="${this.article.title}"
        >

        <div class="toolbar">
          <button data-action="bold">Bold</button>
          <button data-action="italic">Italic</button>
          <button data-action="link">Link</button>
          <button data-action="image">Image</button>
          <button data-action="code">Code Block</button>
        </div>

        <textarea
          class="content-editor"
          placeholder="Write your article in Markdown..."
        >${this.article.content}</textarea>

        <div class="save-status">
          <span class="status-text">All changes saved</span>
        </div>

        <div class="actions">
          <button class="preview-btn">Preview</button>
          <button class="publish-btn">Publish</button>
          <button class="save-draft-btn">Save Draft</button>
        </div>
      </div>
    `;
```

```
    this.setupEditorEvents();
  }

  setupEditorEvents() {
    const titleInput = this.shadowRoot.querySelector('.title-input');
    const contentEditor = this.shadowRoot.querySelector('.content-
        editor');

    titleInput.addEventListener('input', (e) => {
      this.article.title = e.target.value;
      this.markDirty();
    });

    contentEditor.addEventListener('input', (e) => {
      this.article.content = e.target.value;
      this.markDirty();
    });

    // Toolbar actions
    this.shadowRoot.querySelectorAll('[data-action]').forEach(btn => {
      btn.addEventListener('click', () => {
        this.applyFormatting(btn.dataset.action);
      });
    });
  }

  setupAutoSave() {
    setInterval(() => {
      if (this.dirty) {
        this.save();
      }
    }, 10000); // Auto-save every 10 seconds
  }
```

```
  markDirty() {
    this.dirty = true;
    this.shadowRoot.querySelector('.status-text').textContent = 'Unsaved
        changes';
  }

  async save() {
    try {
      await contentService.save(this.article);
      this.dirty = false;
      this.shadowRoot.querySelector('.status-text').textContent = 'All
        changes saved';
    } catch (error) {
      this.shadowRoot.querySelector('.status-text').textContent = 'Save
        failed';
    }
  }

  applyFormatting(action) {
    const textarea = this.shadowRoot.querySelector('.content-editor');
    const start = textarea.selectionStart;
    const end = textarea.selectionEnd;
    const selectedText = textarea.value.substring(start, end);

    let formattedText;
    switch (action) {
      case 'bold':
        formattedText = `**${selectedText}**`;
        break;
      case 'italic':
        formattedText = `*${selectedText}*`;
        break;
      case 'code':
        formattedText = `\`\`\`\n${selectedText}\n\`\`\``;
        break;
      case 'link':
```

```
        formattedText = `[${selectedText}](url)`;
        break;
      case 'image':
        formattedText = `![${selectedText}](image-url)`;
        break;
    }

    textarea.value =
      textarea.value.substring(0, start) +
      formattedText +
      textarea.value.substring(end);

    this.article.content = textarea.value;
    this.markDirty();
  }
}

customElements.define('article-editor', ArticleEditor);
```

19.4.3 Content Preview

```
// markdown-renderer.js
class MarkdownRenderer extends HTMLElement {
  static observedAttributes = ['content'];

  attributeChangedCallback(name, oldValue, newValue) {
    if (name === 'content' && oldValue !== newValue) {
      this.render();
    }
  }

  async render() {
    const markdown = this.getAttribute('content') || '';

    // Use marked.js for markdown parsing
    const { marked } = await import('https://cdn.jsdelivr.net/npm/
        marked@12/+esm');

    this.innerHTML = `
      <div class="markdown-content">
        ${marked.parse(markdown)}
      </div>
    `;
  }
}

customElements.define('markdown-renderer', MarkdownRenderer);
```

19.5 Architecture Decisions and Tradeoffs

19.5.1 When to Use LARC

✅ **Great for:** - Progressive web apps - Content-heavy sites (blogs, documentation) - Dashboards and admin panels - Internal tools - Prototypes and MVPs - Projects with long maintenance horizons

❌ **Consider alternatives for:** - Apps requiring server-side rendering for SEO - Highly interactive games (use Canvas/WebGL directly) - Apps requiring React Native for mobile - Teams deeply invested in React/Vue ecosystem

19.5.2 State Management Patterns

Local State (Component Properties):

```
class Counter extends HTMLElement {
  constructor() {
    super();
    this.count = 0; // Local state
  }

  increment() {
    this.count++;
    this.render();
  }
}
```

Shared State (PAN Bus):

```
// One component publishes
pan.publish('user.login', { userId, name });

// Many components subscribe
pan.subscribe('user.login', ({ name }) => {
  this.userName = name;
  this.render();
});
```

Persistent State (LocalStorage + PAN):

```
class AppState {
  constructor() {
    this.state = JSON.parse(localStorage.getItem('app-state')) || {};

    pan.subscribe('state.*', (data, topic) => {
      const key = topic.split('.')[1];
      this.state[key] = data;
      localStorage.setItem('app-state', JSON.stringify(this.state));
    });
  }

  get(key) {
    return this.state[key];
  }

  set(key, value) {
    pan.publish(`state.${key}`, value);
  }
}

export const appState = new AppState();
```

19.5.3 Scaling Considerations

Code Organization:

```
large-app/
├── index.html
├── app.js
├── features/
|   ├── auth/
|   |   ├── components/
|   |   ├── services/
|   |   └── index.js
|   ├── products/
|   |   ├── components/
|   |   ├── services/
|   |   └── index.js
|   └── checkout/
|       ├── components/
|       ├── services/
|       └── index.js
├── shared/
|   ├── components/
|   ├── services/
|   └── utils/
└── config/
    ├── router.js
    └── pan.js
```

Lazy Loading Features:

```
// app.js - Load features on demand
const features = {
  'auth': () => import('./features/auth/index.js'),
  'products': () => import('./features/products/index.js'),
  'checkout': () => import('./features/checkout/index.js')
};

pan.subscribe('feature.load', async ({ name }) => {
  if (features[name]) {
    await features[name]();
    pan.publish('feature.loaded', { name });
  }
});

// Auto-load on route change
pan.subscribe('router.navigate', ({ path }) => {
  const feature = path.split('/')[1];
  pan.publish('feature.load', { name: feature });
});
```

19.5.4 Performance at Scale

Virtual Scrolling for Large Lists:

```
class VirtualList extends HTMLElement {
  constructor() {
    super();
    this.items = [];
    this.itemHeight = 50;
    this.visibleCount = 20;
    this.scrollTop = 0;
  }

  set data(items) {
    this.items = items;
    this.render();
  }

  connectedCallback() {
    this.addEventListener('scroll', () => {
      this.scrollTop = this.scrollTop;
      requestAnimationFrame(() => this.render());
    });
  }

  render() {
    const startIndex = Math.floor(this.scrollTop / this.itemHeight);
    const endIndex = Math.min(startIndex + this.visibleCount,
        this.items.length);
    const visibleItems = this.items.slice(startIndex, endIndex);

    this.innerHTML = `
      <div style="height: ${this.items.length * this.itemHeight}px;
        position: relative;">
        ${visibleItems.map((item, i) => `
          <div style="
            position: absolute;
            top: ${(startIndex + i) * this.itemHeight}px;
            height: ${this.itemHeight}px;
```

```
            width: 100%;
          ">
            ${item.name}
          </div>
        `).join('')}
      </div>
    `;
  }
}
```

Memoization and Caching:

```
class ProductCatalog extends HTMLElement {
  constructor() {
    super();
    this.cache = new Map();
    this.cacheTimeout = 5 * 60 * 1000; // 5 minutes
  }

  async getProducts(category) {
    const cacheKey = `products-${category}`;
    const cached = this.cache.get(cacheKey);

    if (cached && Date.now() - cached.timestamp < this.cacheTimeout) {
      return cached.data;
    }

    const data = await productService.getByCategory(category);

    this.cache.set(cacheKey, {
      data,
      timestamp: Date.now()
    });

    return data;
  }
}
```

19.6 Maintenance and Evolution

19.6.1 Versioning Components

```
// v1/button.js
class ButtonV1 extends HTMLElement {
  // Original implementation
}
customElements.define('app-button-v1', ButtonV1);

// v2/button.js
class ButtonV2 extends HTMLElement {
  // New implementation with breaking changes
}
customElements.define('app-button', ButtonV2);

// Migration path: Both versions coexist
// Old code uses app-button-v1
// New code uses app-button
// Gradual migration over time
```

19.6.2 Feature Flags

```
// feature-flags.js
class FeatureFlags {
  constructor() {
    this.flags = {
      'new-checkout': false,
      'beta-dashboard': true,
      'experimental-editor': false
    };
  }

  isEnabled(feature) {
    return this.flags[feature] ?? false;
  }

  enable(feature) {
    this.flags[feature] = true;
    pan.publish('feature-flag.changed', { feature, enabled: true });
  }
}

export const featureFlags = new FeatureFlags();

// Usage in component
class Checkout extends HTMLElement {
  connectedCallback() {
    if (featureFlags.isEnabled('new-checkout')) {
      this.innerHTML = '<new-checkout-flow></new-checkout-flow>';
    } else {
      this.innerHTML = '<legacy-checkout-flow></legacy-checkout-flow>';
    }
  }
}
```

19.6.3 Migration from React

Here's how to migrate a React app to LARC:

React Component:

```
function TodoItem({ todo, onComplete }) {
  return (
    <div className="todo-item">
      <input
        type="checkbox"
        checked={todo.completed}
        onChange={() => onComplete(todo.id)}
      />
      <span>{todo.text}</span>
    </div>
  );
}
```

LARC Equivalent:

```
class TodoItem extends HTMLElement {
  static observedAttributes = ['completed'];

  connectedCallback() {
    this.render();
  }

  attributeChangedCallback() {
    this.render();
  }

  render() {
    const completed = this.hasAttribute('completed');
    const text = this.getAttribute('text') || '';

    this.innerHTML = `
      <div class="todo-item">
        <input
          type="checkbox"
          ${completed ? 'checked' : ''}
        >
        <span>${text}</span>
      </div>
    `;

    this.querySelector('input').addEventListener('change', () => {
      pan.publish('todo.complete', { id: this.getAttribute('todo-id') });
    });
  }
}

customElements.define('todo-item', TodoItem);
```

19.6.4 Migration from Vue

Vue Component:

```
<template>
  <div class="user-card">
    <img :src="user.avatar" :alt="user.name">
    <h3>{{ user.name }}</h3>
    <p>{{ user.email }}</p>
    <button @click="viewProfile">View Profile</button>
  </div>
</template>

<script>
export default {
  props: ['user'],
  methods: {
    viewProfile() {
      this.$router.push(`/user/${this.user.id}`);
    }
  }
}
</script>
```

LARC Equivalent:

```
class UserCard extends HTMLElement {
  static observedAttributes = ['user-id'];

  async connectedCallback() {
    const userId = this.getAttribute('user-id');
    this.user = await userService.get(userId);
    this.render();
  }

  render() {
    this.innerHTML = `
      <div class="user-card">
        <img src="${this.user.avatar}" alt="${this.user.name}">
        <h3>${this.user.name}</h3>
        <p>${this.user.email}</p>
        <button class="view-btn">View Profile</button>
      </div>
    `;

    this.querySelector('.view-btn').addEventListener('click', () => {
      pan.publish('router.navigate', { path: `/user/${this.user.id}` });
    });
  }
}

customElements.define('user-card', UserCard);
```

19.7 Real-World Challenges and Solutions

19.7.1 Challenge 1: SEO for Content Sites

Problem: Client-side rendering isn't indexed by search engines.

Solution: Pre-render static content

```
// build-static.js - Pre-render pages at build time
import { JSDOM } from 'jsdom';
import { writeFileSync } from 'fs';

async function prerender(url, outputPath) {
  const dom = new JSDOM(html);
  global.window = dom.window;
  global.document = dom.window.document;

  // Load and execute components
  await import('./components/article-page.js');

  // Wait for async content to load
  await new Promise(resolve => setTimeout(resolve, 1000));

  // Write rendered HTML
  writeFileSync(outputPath, dom.serialize());
}

// Pre-render all blog posts
const posts = await contentService.getAllPosts();
for (const post of posts) {
  await prerender(`/blog/${post.slug}`, `dist/blog/${post.slug}.html`);
}
```

19.7.2 Challenge 2: Complex State Synchronization

Problem: Multiple components need to stay in sync with complex state.

Solution: Centralized state manager

```
// state-manager.js
class StateManager {
  constructor() {
    this.state = {};
    this.subscribers = new Map();
  }

  get(path) {
    return path.split('.').reduce((obj, key) => obj?.[key], this.state);
  }

  set(path, value) {
    const keys = path.split('.');
    const lastKey = keys.pop();
    const target = keys.reduce((obj, key) => {
      if (!obj[key]) obj[key] = {};
      return obj[key];
    }, this.state);

    target[lastKey] = value;

    // Notify subscribers
    pan.publish(`state.${path}`, value);
  }

  subscribe(path, callback) {
    return pan.subscribe(`state.${path}`, callback);
  }
}

export const state = new StateManager();
```

19.7.3 Challenge 3: Testing Async Component Behavior

Problem: Components with async data loading are hard to test.

Solution: Dependency injection and mocking

```
// product-list.test.js
import { fixture, html } from '@open-wc/testing';
import './product-list.js';

// Mock service
const mockProductService = {
  getAll: () => Promise.resolve([
    { id: 1, name: 'Product 1', price: 10 },
    { id: 2, name: 'Product 2', price: 20 }
  ])
};

describe('ProductList', () => {
  it('renders products after loading', async () => {
    // Inject mock
    window.productService = mockProductService;

    const el = await fixture(html`<product-list></product-list>`);

    // Wait for async render
    await new Promise(resolve => setTimeout(resolve, 100));

    const products = el.querySelectorAll('product-card');
    expect(products.length).to.equal(2);
  });
});
```

19.8 Team Practices and Workflows

19.8.1 Code Review Checklist

- [ ] Component follows single responsibility principle
- [ ] Shadow DOM used for encapsulation
- [ ] Event listeners cleaned up in `disconnectedCallback`
- [ ] Attributes declared in `observedAttributes`
- [ ] JSDoc comments for public API
- [ ] Tests cover happy path and error cases
- [ ] Accessible (keyboard navigation, ARIA)
- [ ] Performance profiled (if rendering > 100 items)

19.8.2 Component Design Guidelines

1. **Keep components small**: If a component is > 200 lines, split it
2. **Prefer composition**: Use slots and nested components
3. **Clear naming**: `<user-profile-card>` not `<component-5>`
4. **Consistent patterns**: All similar components follow same structure
5. **Document props**: JSDoc with type information
6. **Handle errors gracefully**: Show user-friendly error messages

19.8.3 Development Workflow

```
# 1. Create feature branch
git checkout -b feature/user-authentication

# 2. Write component with tests
npm run test:watch

# 3. Run linter
npm run lint

# 4. Create PR
git push origin feature/user-authentication

# 5. CI runs tests and linter
# 6. Code review
# 7. Merge to main
# 8. Deploy to production
npm run deploy
```

19.9 Troubleshooting Real-World Applications

19.9.1 Problem 1: Component State Gets Out of Sync

Symptoms: UI shows stale data, actions don't reflect in other parts of the app

Cause: Multiple sources of truth, missed PAN bus updates

Solution:

```
// Bad: Duplicated state
class ProductCard extends HTMLElement {
  constructor() {
    super();
    this.product = null;  // Local copy
  }
}

// Good: Single source of truth
class ProductCard extends HTMLElement {
  connectedCallback() {
    // Subscribe to state changes
    this.unsub = pan.subscribe('products.*.updated', (data, topic) => {
      const productId = topic.split('.')[1];
      if (productId === this.productId) {
        this.render();
      }
    });
  }
}
```

19.9.2 Problem 2: Memory Leaks in Long-Running Apps

Symptoms: App slows down over time, browser tab uses increasing memory

Cause: Event listeners not cleaned up, retained references

Solution:

```
class Dashboard extends HTMLElement {
  connectedCallback() {
    // Track subscriptions for cleanup
    this.subscriptions = [
      pan.subscribe('metrics.updated', this.handleMetrics),
      pan.subscribe('alerts.new', this.handleAlert)
    ];

    // Track intervals
    this.updateInterval = setInterval(() => this.fetchUpdates(), 5000);
  }

  disconnectedCallback() {
    // Clean up all subscriptions
    this.subscriptions.forEach(unsub => unsub());
    this.subscriptions = [];

    // Clear intervals
    clearInterval(this.updateInterval);
  }
}
```

19.9.3 Problem 3: Performance Degrades with Large Datasets

Symptoms: Slow rendering, laggy interactions when displaying many items

Cause: Rendering all items at once, no virtualization

Solution:

```
// Use virtual scrolling for large lists
class VirtualProductList extends HTMLElement {
  render() {
    const viewportHeight = this.clientHeight;
    const scrollTop = this.scrollTop;
    const itemHeight = 100;

    // Calculate visible range
    const startIndex = Math.floor(scrollTop / itemHeight);
    const endIndex = Math.ceil((scrollTop + viewportHeight) /
        itemHeight);

    // Only render visible items
    const visibleItems = this.products.slice(startIndex, endIndex + 1);

    this.innerHTML = `
      <div style="height: ${this.products.length * itemHeight}px">
        <div style="transform: translateY(${startIndex * itemHeight}px)">
          ${visibleItems.map(product =>
         this.renderItem(product)).join('')}
        </div>
      </div>
    `;
  }
}
```

19.9.4 Problem 4: Race Conditions with Async Operations

Symptoms: Wrong data displayed, operations complete out of order

Cause: Multiple async requests, no cancellation or ordering

Solution:

```
class SearchBox extends HTMLElement {
  constructor() {
    super();
    this.abortController = null;
    this.requestId = 0;
  }

  async search(query) {
    // Cancel previous request
    if (this.abortController) {
      this.abortController.abort();
    }

    this.abortController = new AbortController();
    const currentRequestId = ++this.requestId;

    try {
      const results = await fetch(`/api/search?q=${query}`, {
        signal: this.abortController.signal
      });

      // Only update if this is still the latest request
      if (currentRequestId === this.requestId) {
        this.displayResults(await results.json());
      }
    } catch (err) {
      if (err.name !== 'AbortError') throw err;
    }
  }
}
```

19.10 Real-World Application Best Practices

1. **Design for Scale from Day One**
    - Plan component boundaries before coding
    - Use lazy loading for routes and heavy components
    - Profile performance early and often
2. **Establish Clear State Management Patterns**
    - Single source of truth for shared state
    - Local state for component-specific data
    - Document state flow in architecture diagrams
3. **Implement Comprehensive Error Handling**
    - Catch errors at component boundaries
    - Display user-friendly error messages
    - Log errors for debugging and monitoring
4. **Write Tests for Critical Paths**
    - Test user journeys end-to-end
    - Cover edge cases and error scenarios
    - Maintain test coverage above 80%
5. **Optimize for Production**
    - Minify and compress assets
    - Enable HTTP/2 and compression
    - Use CDN for static assets
    - Implement service worker caching
6. **Monitor and Measure**
    - Track Core Web Vitals
    - Set up error tracking (Sentry, etc.)
    - Monitor real user metrics
    - Set up alerts for performance regressions
7. **Plan for Evolution**
    - Version your components
    - Use feature flags for gradual rollouts
    - Keep dependencies up to date
    - Document breaking changes
8. **Prioritize Developer Experience**

    - Set up linting and formatting
    - Create component templates/generators
    - Document common patterns
    - Maintain example implementations
9. **Focus on Accessibility**
    - Test with screen readers
    - Support keyboard navigation
    - Follow ARIA best practices
    - Include accessibility in code review checklist
10. **Build for the Long Term**
    - Prefer web standards over frameworks
    - Keep dependencies minimal
    - Write clear, self-documenting code
    - Invest in comprehensive documentation

19.11 Hands-On Exercises

19.11.1 Exercise 1: Build a Multi-Page E-Commerce App

Build a complete e-commerce application with:

- Product listing with filters and sorting
- Individual product pages
- Shopping cart with persistence
- Checkout flow with form validation
- Admin panel for inventory management

Requirements: - Use PAN bus for state management - Implement lazy loading for routes - Add comprehensive error handling - Include unit and E2E tests - Deploy to production hosting

Bonus Challenge: Add user authentication with JWT tokens and implement role-based access control for admin features.

19.11.2 Exercise 2: Create a Real-Time Dashboard

Build a dashboard application that displays:

- Real-time metrics (using WebSocket)
- Interactive charts and graphs
- Data filtering and time range selection
- Alert notifications
- User preferences and saved views

Requirements: - Virtual scrolling for large datasets - Optimistic UI updates - Service worker for offline functionality - Performance profiled (60fps interactions) - Responsive design for mobile/desktop

Bonus Challenge: Implement data export (CSV, PDF), scheduled reports, and email notifications for alerts.

19.11.3 Exercise 3: Build a Blog CMS

Create a complete content management system with:

- Rich text editor for articles
- Draft/publish workflow
- Tag and category management
- SEO optimization (meta tags, sitemaps)
- Media library for images
- Comment moderation

Requirements: - Server integration (Node.js or your choice) - Client-side routing with SSR for SEO - Form validation and error handling - Auto-save functionality - Search functionality - User roles (admin, editor, author)

Bonus Challenge: Add markdown support with live preview, scheduled publishing, and content analytics (views, engagement).

19.11.4 Exercise 4: Migrate an Existing Application

Take an existing React or Vue application and migrate it to LARC:

- Audit current architecture and dependencies
- Create migration plan with phases
- Implement hybrid approach (gradual migration)
- Maintain feature parity during migration
- Compare bundle sizes and performance

Requirements: - Document migration process - Create mapping guide (React/Vue → LARC) - Identify and solve migration challenges - Set up automated tests to prevent regressions - Deploy both versions and compare metrics

Bonus Challenge: Create a migration tool/CLI that automates conversion of common component patterns.

19.12 Summary

Building real applications with LARC teaches you to:

- **Think in components**: Break UI into small, reusable pieces
- **Leverage web standards**: Use what browsers provide
- **Embrace simplicity**: No build step means faster development
- **Scale thoughtfully**: Lazy load, virtualize, cache strategically
- **Test continuously**: Catch bugs early with comprehensive tests
- **Profile performance**: Measure before optimizing
- **Plan for evolution**: Version components, use feature flags

LARC's no-build philosophy and standards-based approach make it ideal for long-lived projects that need maintainability and performance at scale.

19.13 Further Reading

- **Building with LARC - All Chapters**: Complete API reference and advanced patterns
- **MDN Web Components**: Deep dive into the platform
- **Open-wc**: Testing and tooling best practices for web components

19.14 About the Author

Christopher Robison is a veteran software engineer and architect with nearly three decades of experience building systems that range from biotech and online trading platforms to complex web applications and AI-driven tools. A lifelong maker with a deep appreciation for open standards, he has spent his career exploring the boundaries of what the web can do when you stop fighting the platform and start embracing it.

He is the creator of LARC.js and the PAN message bus, a browser-native architecture inspired by the elegant simplicity of the automotive CAN bus. His work blends engineering pragmatism with a playful curiosity that has led him to design everything from 3D printers and robotics to interactive music systems and decentralized applications.

Christopher currently lives in San Francisco, where he continues to build things that bridge the digital and physical worlds — and occasionally sneaks off to play punk rock shows with his band.

19.15 The Web Has Grown Up. It’s Time Our Apps Did Too.

Modern browsers aren’t the brittle playgrounds they once were. They’re fast, secure, richly capable application platforms — yet most of today’s development stacks still treat them like dumb terminals that need layers of tooling, bundling, and framework magic just to function.

Learning LARC shows another path.

LARC embraces the browser as a mature runtime, using nothing but open standards — Web Components, modules, events, and message buses — to build complex, deeply interactive applications without build systems, without monoliths, and without ceremony. Through clear narrative examples and real architectural stories, this book teaches you how to design apps as ecosystems: small parts, clearly defined, communicating through a shared bus.

You’ll learn how to structure large systems out of tiny cooperating modules, expose capabilities through message patterns instead of global state, keep your interfaces clean, and let the platform do the heavy lifting it was built for.

No bundlers. No scaffolding. No twenty-layer dependency stacks. Just the browser, finally treated like the grown-up it is.

Whether you’re maintaining a legacy system or starting fresh, **Learning LARC** will help you rethink how modern web apps can — and should — be built.