Contents

[Overview 5](#_Toc208230175)

[Purpose and Scope 5](#_Toc208230176)

[Application Architecture 5](#_Toc208230177)

[Key Components 7](#_Toc208230178)

[Build System and Deployment 8](#_Toc208230179)

[External Integrations 9](#_Toc208230180)

[Technology Stack 10](#_Toc208230181)

[Getting Started 10](#_Toc208230182)

[Prerequisites 10](#_Toc208230183)

[Installation 11](#_Toc208230184)

[Environment Configuration 11](#_Toc208230185)

[Development Workflows 12](#_Toc208230186)

[Build Commands 14](#_Toc208230187)

[Development Tools 14](#_Toc208230188)

[Project Structure 15](#_Toc208230189)

[Next Steps 15](#_Toc208230190)

[Architecture 16](#_Toc208230191)

[System Architecture Overview 16](#_Toc208230192)

[Routing and Layout Selection System 17](#_Toc208230193)

[Context Aggregation Pattern 18](#_Toc208230194)

[Dual-Target Build Architecture 19](#_Toc208230195)

[Authentication System Integration 20](#_Toc208230196)

[Application Initialization Sequence 21](#_Toc208230197)

[Application Structure 21](#_Toc208230198)

[Entry Points and Application Initialization 22](#_Toc208230199)

[Provider Stack Architecture 24](#_Toc208230200)

[Routing System 25](#_Toc208230201)

[Layout Components 26](#_Toc208230202)

[Build Target Considerations 28](#_Toc208230203)

[Component Hierarchy Overview 28](#_Toc208230204)

[State Management 30](#_Toc208230205)

[Purpose and Scope 30](#_Toc208230206)

[Dual Authentication Context Architecture 30](#_Toc208230207)

[UserContext - AWS Amplify Authentication System 30](#_Toc208230208)

[ApiContext - Basketball API and Context Aggregation 31](#_Toc208230209)

[Individual Context Slice Implementations 33](#_Toc208230210)

[State Access Patterns 35](#_Toc208230211)

[Performance Optimizations 37](#_Toc208230212)

[Authentication System 38](#_Toc208230213)

[Purpose and Scope 38](#_Toc208230214)

[Authentication Architecture Overview 38](#_Toc208230215)

[AWS Cognito Authentication 39](#_Toc208230216)

[API Key Authentication 42](#_Toc208230217)

[User Context and State Management 43](#_Toc208230218)

[Dual Authentication Integration 44](#_Toc208230219)

[Key Components Summary 47](#_Toc208230220)

[Build System 48](#_Toc208230221)

[Overview 48](#_Toc208230222)

[Build Configuration 48](#_Toc208230223)

[Build Scripts 49](#_Toc208230224)

[Development Workflow 50](#_Toc208230225)

[Production Builds 50](#_Toc208230226)

[Output Artifacts 51](#_Toc208230227)

[Configuration Files 52](#_Toc208230228)

[User Interface 52](#_Toc208230229)

[UI Architecture Overview 53](#_Toc208230230)

[Theme and Styling System 54](#_Toc208230231)

[Core UI Components 55](#_Toc208230232)

[Advanced UI Components 60](#_Toc208230233)

[Component Integration Patterns 61](#_Toc208230234)

[Design System Conventions 62](#_Toc208230235)

[Core Components 63](#_Toc208230236)

[Component Architecture Overview 63](#_Toc208230237)

[Data Display Components 63](#_Toc208230238)

[Navigation Components 65](#_Toc208230239)

[Access Control Components 66](#_Toc208230240)

[Form and Input Components 67](#_Toc208230241)

[Component Usage Patterns 68](#_Toc208230242)

[Game Setup Interface 69](#_Toc208230243)

[Purpose and Scope 69](#_Toc208230244)

[Component Architecture Overview 70](#_Toc208230245)

[Data Flow and State Management 70](#_Toc208230246)

[API Integration Points 71](#_Toc208230247)

[TypeScript Interface System 73](#_Toc208230248)

[Team Logo Management System 74](#_Toc208230249)

[Error Handling and Loading States 74](#_Toc208230250)

[Game Simulation Interfaces 75](#_Toc208230251)

[Full Season Simulation Interface 75](#_Toc208230252)

[Single Game Simulation Interface 77](#_Toc208230253)

[Supporting Components 78](#_Toc208230254)

[Data Flow and State Management 79](#_Toc208230255)

[User Interaction Patterns 81](#_Toc208230256)

[Layout and Navigation 81](#_Toc208230257)

[Application Bootstrap Architecture 81](#_Toc208230258)

[Routing System 83](#_Toc208230259)

[Layout Components 84](#_Toc208230260)

[Context Provider Hierarchy 85](#_Toc208230261)

[Navigation Integration 86](#_Toc208230262)

[Navigation State Integration 87](#_Toc208230263)

[Authentication UI 88](#_Toc208230264)

[Purpose and Scope 88](#_Toc208230265)

[Authentication UI Component Architecture 88](#_Toc208230266)

[User Authentication Flows 88](#_Toc208230267)

[Component Implementation Details 89](#_Toc208230268)

[Form Validation and Error Handling 91](#_Toc208230269)

[Styling and Layout Patterns 92](#_Toc208230270)

[Game Features 92](#_Toc208230271)

[Simulation Architecture Overview 93](#_Toc208230272)

[Simulation Modes 93](#_Toc208230273)

[Player and Team Data Integration 94](#_Toc208230274)

[Advanced Simulation Features 96](#_Toc208230275)

[Statistical Output and Reporting 98](#_Toc208230276)

[Game Configuration API Integration 98](#_Toc208230277)

[Basketball Simulation 99](#_Toc208230278)

[System Architecture 99](#_Toc208230279)

[Game Simulation Modes 99](#_Toc208230280)

[Player Data Management 100](#_Toc208230281)

[Real-Time Game State Management 101](#_Toc208230282)

[Play-by-Play and Box Score Systems 102](#_Toc208230283)

[API Integration Architecture 103](#_Toc208230284)

# Overview

## Purpose and Scope

This document provides a comprehensive overview of the NBA Basketball Simulation application, a React+TypeScript web and desktop application that enables users to simulate basketball games and seasons using real NBA data. The application supports both single-game simulations and full 82-game season simulations, with authentication through AWS Cognito and data integration via external basketball APIs.

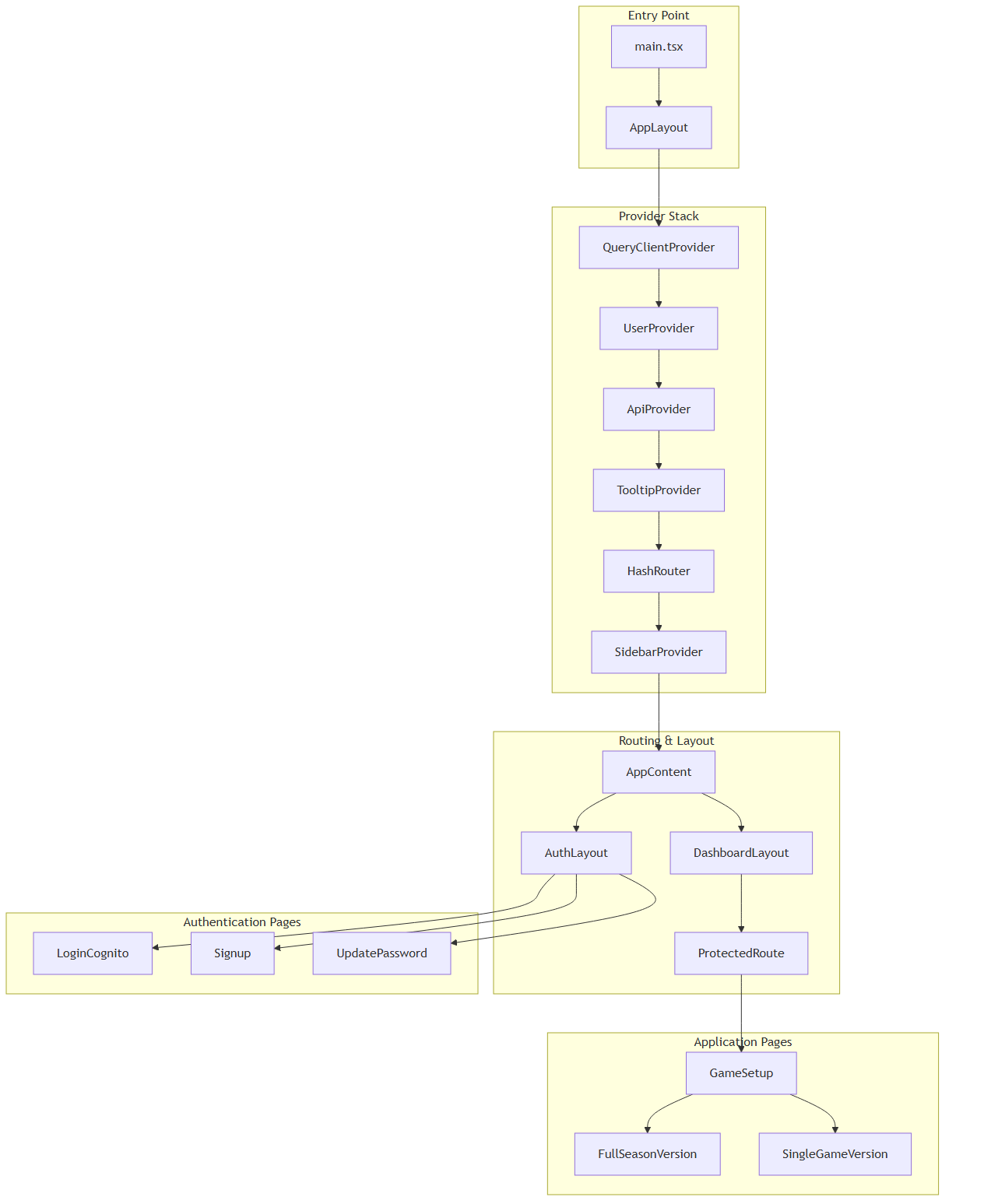
The application is designed with dual deployment capabilities, supporting both web browser deployment via Vite and desktop application deployment via Electron. This overview covers the core architecture, component relationships, and system integrations that enable the basketball simulation functionality.

For detailed information about specific subsystems, see [Getting Started](#_Getting_Started) for setup instructions, [Architecture](#_Architecture) for technical details, [User Interface](#_User_Interface) for component documentation, and [Game Features](#_Game_Features) for simulation mechanics.

## Application Architecture

The application follows a layered architecture pattern with React context providers managing state and external service integrations. The core structure centers around the GameSetup component as the primary orchestrator, coordinating between authentication services, external APIs, and specialized simulation components.

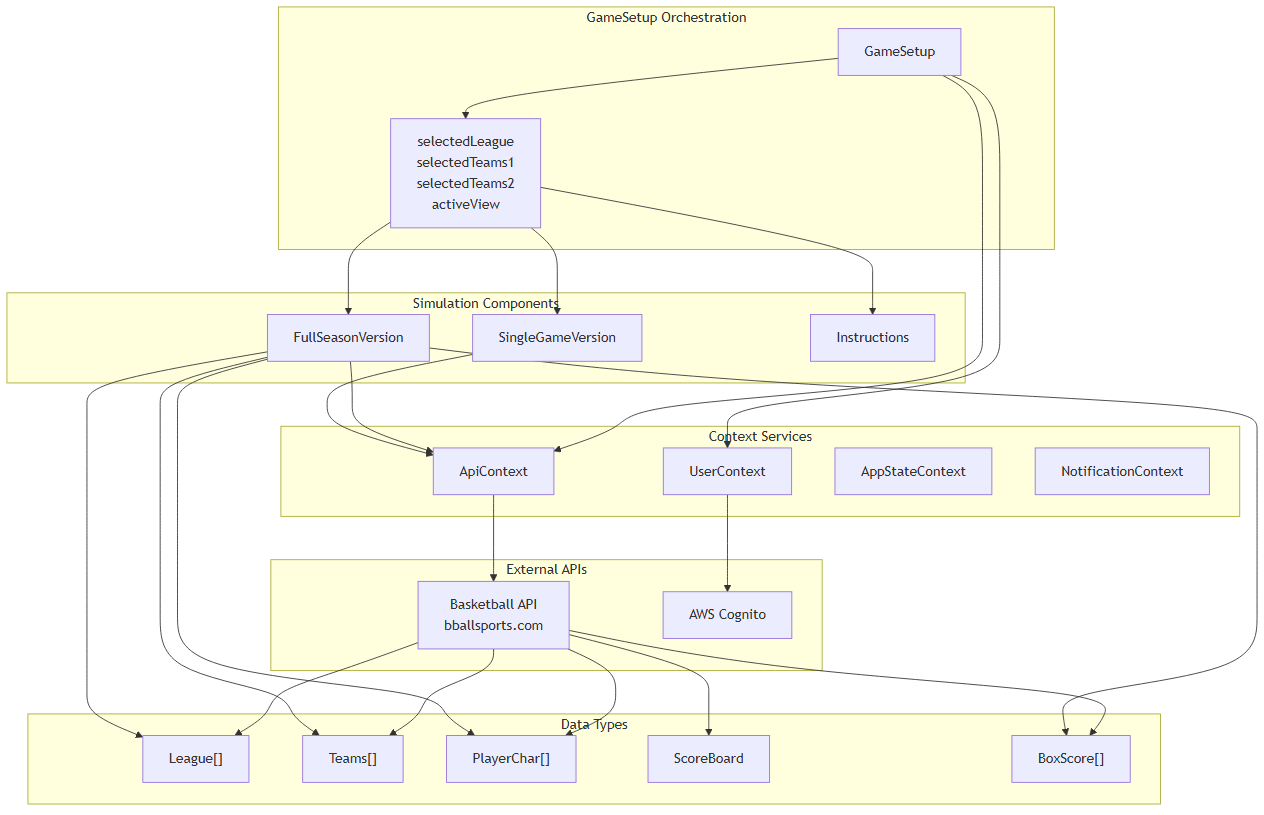
### Core Application Flow



**Application Entry and Provider Hierarchy**

Sources: [/src/layout.tsx:87-103](file:///C:\src\layout.tsx), [/src/main.tsx](file:///C:\src\main.tsx)

### Component Relationships



**Core Component Architecture and Data Flow**

Sources: [/src/pages/GameSetup.tsx:268-947](file:///C:\src\pages\GameSetup.tsx), [/src/pages/FullSeasonVersion.tsx:153-191](file:///C:\src\pages\FullSeasonVersion.tsx)

## Key Components

### GameSetup Component

The GameSetup component serves as the primary orchestrator, managing user interactions for league selection, team configuration, and simulation mode switching. It maintains state for leagues, teams, players, and game results while coordinating API calls to external basketball services.

Key state management includes:  
- leagues: Available basketball leagues from API  
- selectedLeague: Currently selected league for simulation  
- selectedTeams1/selectedTeams2: Away and home teams for games  
- activeView: Current mode ('full-season', 'single-game', 'instructions')  
- playersTeam1/playersTeam2: Player statistics for both teams  
- scoreBoard: Live game scoreboard data  
- boxScore and boxScoreFullSeason: Game statistics and results

Sources: [/src/pages/GameSetup.tsx:268-279](file:///C:\src\pages\GameSetup.tsx), [/src/pages/GameSetup.tsx:287-367](file:///C:\src\pages\GameSetup.tsx)

### Simulation Components

The application includes two specialized simulation components:

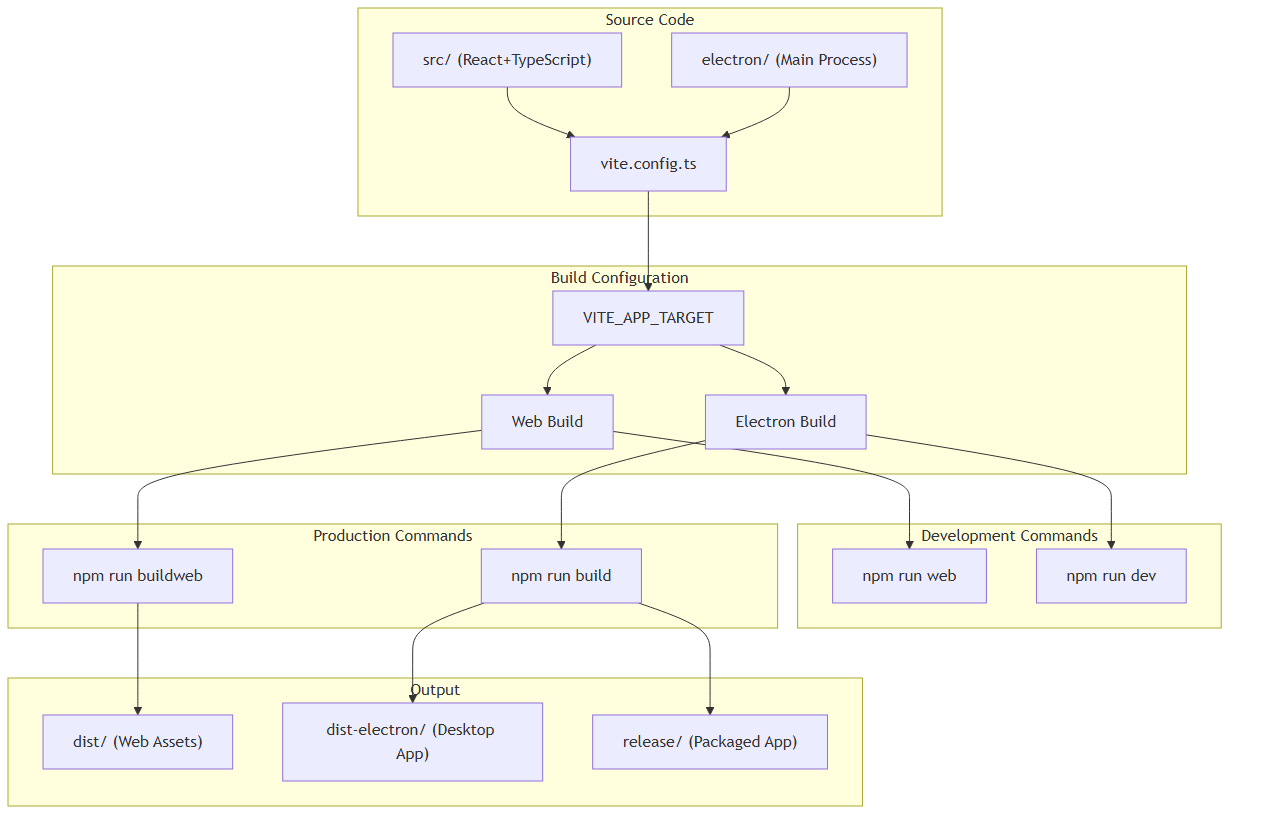
**FullSeasonVersion**: Manages complete 82-game season simulations with features including:  
- Season schedule generation via handleSchedule82()  
- Player substitution pattern management  
- Season-long statistics tracking  
- Multiple simulation modes (predict, 8200, fsv)

**SingleGameVersion**: Handles individual game simulations with real-time scoreboard updates and play-by-play tracking.

Sources: [/src/pages/FullSeasonVersion.tsx:153-191](file:///C:\src\pages\FullSeasonVersion.tsx), [/src/pages/GameSetup.tsx:4-5](file:///C:\src\pages\GameSetup.tsx)

## Build System and Deployment

### Dual Deployment Architecture



**Build System Configuration and Deployment Targets**

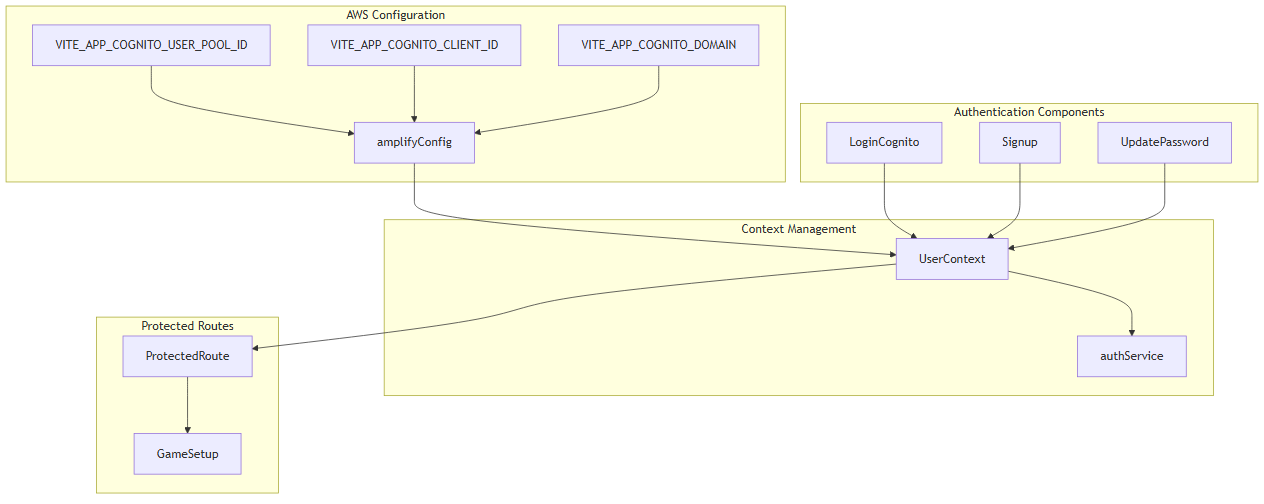
The application uses environment variable VITE\_APP\_TARGET to determine build configuration, enabling single-codebase deployment to both web browsers and Electron desktop applications.

Sources: [/package.json:13-20](file:///C:\package.json), [/package.json:158-173](file:///C:\package.json)

## External Integrations

### Authentication System

The application integrates with AWS Amplify and Cognito for user management:



**AWS Cognito Integration Architecture**

Sources: [/src/layout.tsx:27-49](file:///C:\src\layout.tsx), [/src/layout.tsx:73-77](file:///C:\src\layout.tsx)

### Basketball API Integration

The application communicates with external basketball APIs through the ApiContext using a dual-token authentication system:

• AWS ID tokens for API authentication headers

• Specialized NBA tokens for request body authentication

• API endpoints include league data, team rosters, player statistics, and game simulation services

Key API interactions include:  
- handleFetchLeagues(): Retrieve available basketball leagues  
- handleFetchTeams(): Get teams for selected league  
- handleFetchPlayersTeam1/2(): Load player statistics  
- handlePredictMode(): Execute season simulations  
- handleSingleGameInitial(): Initialize single game simulations

Sources: [/src/pages/GameSetup.tsx:371-447](file:///C:\src\pages\GameSetup.tsx), [/src/pages/GameSetup.tsx:467-494](file:///C:\src\pages\GameSetup.tsx)

## Technology Stack

The application leverages a comprehensive technology stack:

|  |  |
| --- | --- |
| **Category** | **Technologies** |
| Frontend Framework | React 18.3.1, TypeScript 5.8.3 |
| UI Libraries | Radix UI components, PrimeReact 10.9.6 |
| Styling | Tailwind CSS 3.4.17, CSS animations |
| State Management | React Context API, TanStack Query 5.56.2 |
| Routing | React Router DOM 6.26.2 |
| Authentication | AWS Amplify 6.15.3, AWS Cognito |
| Desktop Platform | Electron 30.0.1, Electron Builder 24.13.3 |
| Build Tools | Vite 5.1.6, TypeScript compiler |
| Development | ESLint, Electron DevTools |

Sources: [/package.json:22-79](file:///C:\package.json), [/package.json:114-156](file:///C:\package.json)

# Getting Started

This document provides installation, environment setup, and basic usage instructions for the NBA basketball simulation application. It covers setting up the development environment, configuring build targets, and running the application in both web browser and Electron desktop modes.

For architectural details and design patterns, see [Architecture](file:///C:\Repositorios\md2docx\output\3_Architecture.md). For information about the authentication system and AWS integration, see [Authentication System](file:///C:\Repositorios\md2docx\output\6_Authentication_System.md). For details about the user interface components and game features, see [User Interface](file:///C:\Repositorios\md2docx\output\8_User_Interface.md).

## Prerequisites

Before setting up the application, ensure you have the following installed on your development machine:

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Version** | **Purpose** |
| Node.js | 18+ | Runtime environment for React and build tools |
| npm | 8+ | Package dependency management |
| Git | Latest | Source code version control |

The application requires access to external basketball simulation APIs and AWS Amplify services for full functionality.

## Installation

Clone the repository and install dependencies:

git clone https://github.com/metantonio/react-test-ts-nb  
cd react-test-ts-nb  
npm install

The installation process downloads dependencies including React 18.3.1, TypeScript 5.8.3, Electron 30.0.1, and extensive UI component libraries from Radix UI and PrimeReact.

**Sources:** [package.json:1-174](file:///C:\package.json)

## Environment Configuration

The application uses environment variables to control build targets and external service configuration. Create a .env file based on the provided template:

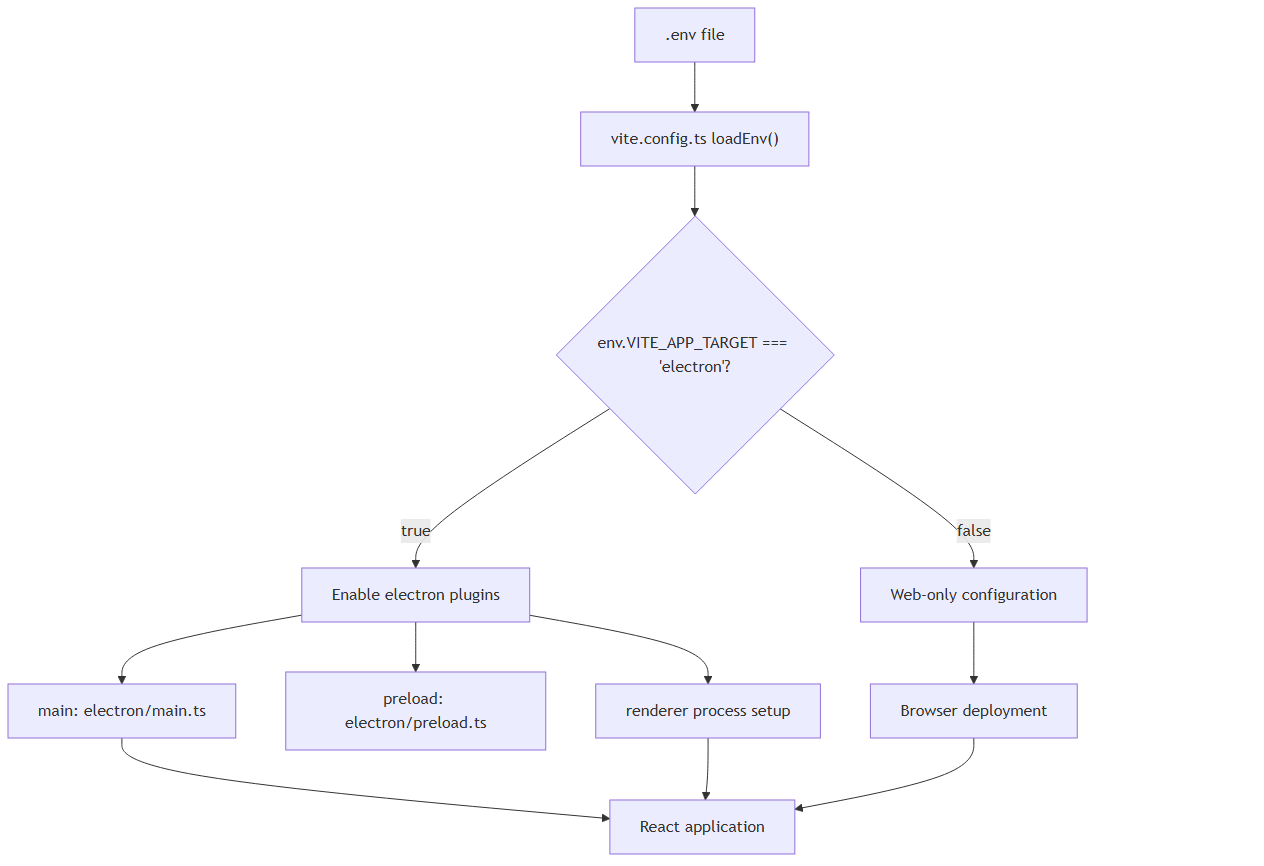
cp .env.example .env

### Build Target Configuration

The VITE\_APP\_TARGET environment variable determines whether the application builds for Electron desktop or web browser deployment:

|  |  |  |
| --- | --- | --- |
| **Variable** | **Values** | **Effect** |
| VITE\_APP\_TARGET | electron | Enables Electron plugins and desktop features |
| VITE\_APP\_TARGET | web or unset | Web-only build without Electron dependencies |

### Build Target Flow



**Sources:** [vite.config.ts:8-11](file:///C:\vite.config.ts), [.env.example:1](file:///C:\.env.example)

## Development Workflows

The application supports two development modes through npm scripts defined in package.json.

### Web Development Mode

Start the application in web browser mode:

npm run web

This command executes vite and starts a development server at http://localhost:5173. The application runs without Electron-specific features.

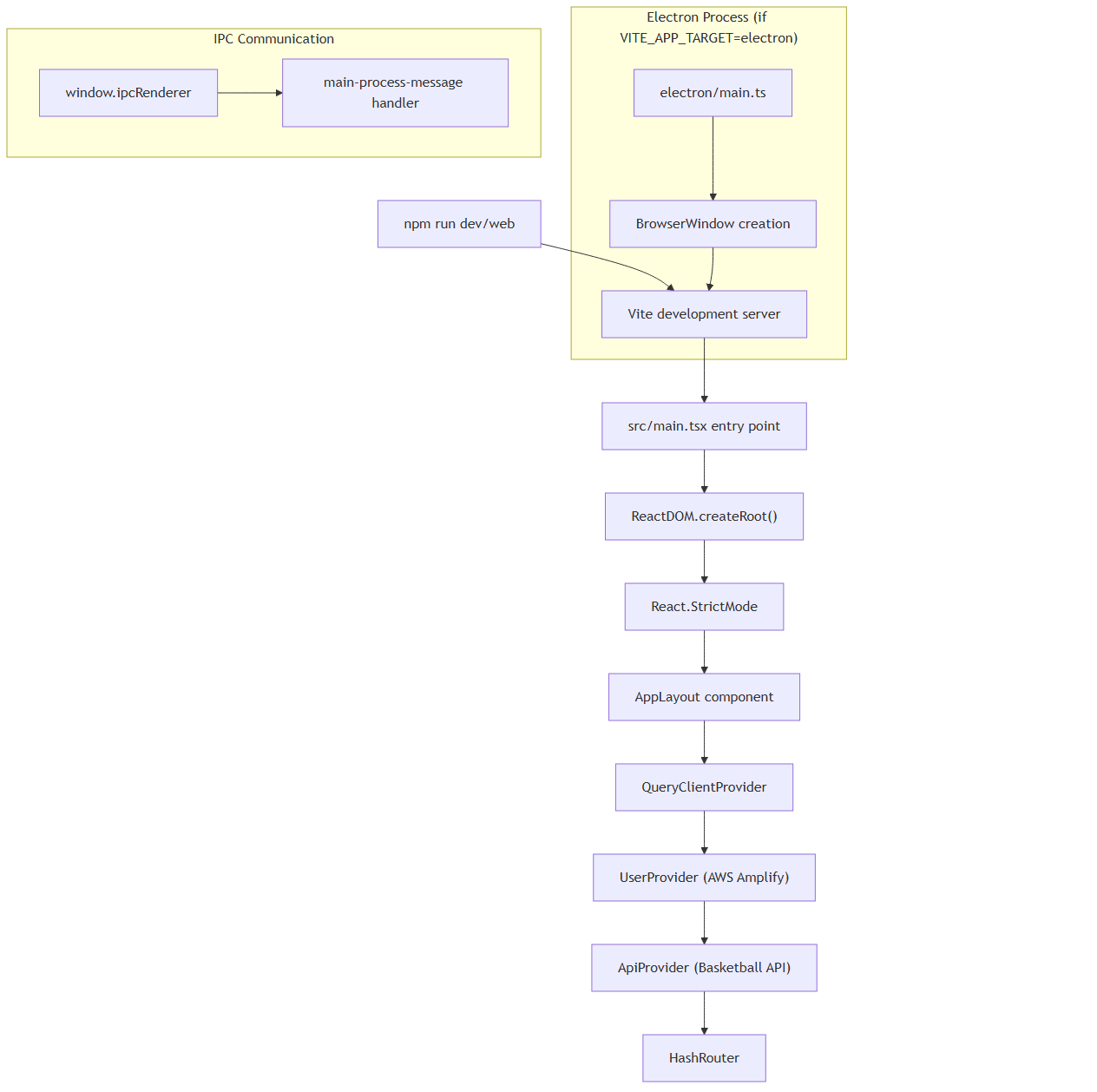
### Desktop Development Mode

Start the application as an Electron desktop application:

npm run dev

This command also executes vite but with Electron integration enabled, launching both the Vite development server and the Electron application window.

### Application Startup Sequence



**Sources:** [src/main.tsx:8-19](file:///C:\src\main.tsx), [package.json:14-15](file:///C:\package.json)

## Build Commands

The application provides separate build commands for different deployment targets.

### Web Production Build

Create an optimized web build:

npm run buildweb

This command executes tsc && vite build, compiling TypeScript and generating static files in the dist/ directory.

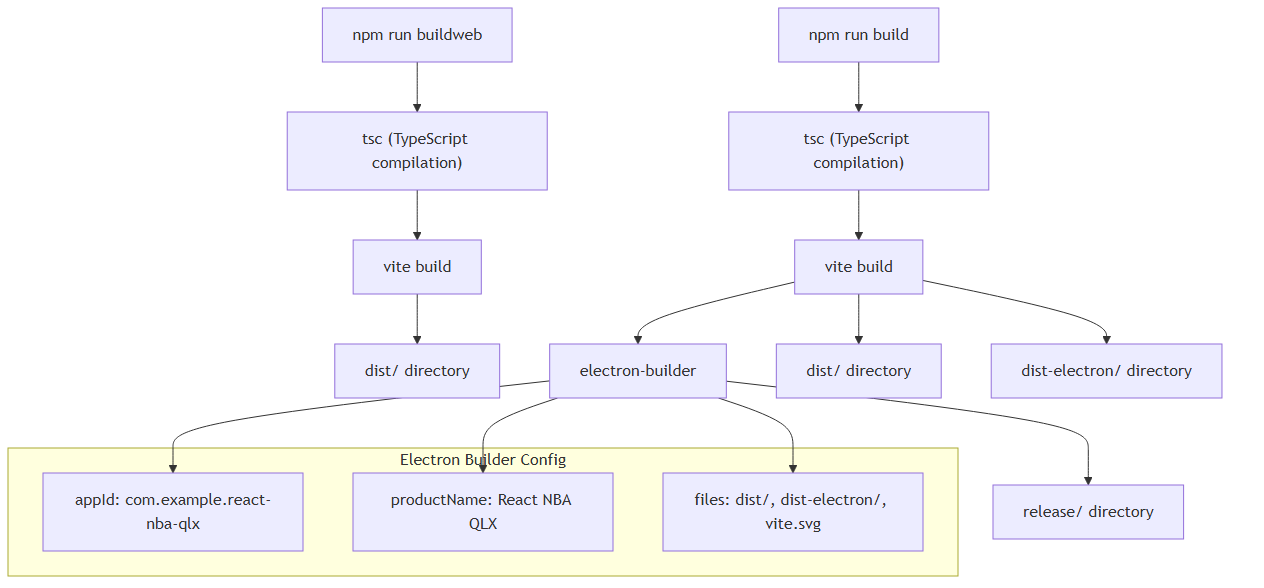
### Desktop Production Build

Create Electron desktop application packages:

npm run build

This command executes tsc && vite build && electron-builder, generating both web assets and platform-specific desktop installers.

### Build Process Flow



**Sources:** [package.json:16-17](file:///C:\package.json), [package.json:159-173](file:///C:\package.json)

## Development Tools

### Code Quality

Run ESLint to check TypeScript and TSX files:

npm run lint

The linting configuration uses @typescript-eslint/eslint-plugin and eslint-plugin-react-hooks for React-specific rules.

### Production Preview

Test the production web build locally:

npm run preview

This serves the built application from the dist/ directory using Vite's preview server.

**Sources:** [package.json:18-19](file:///C:\package.json)

## Project Structure

After installation and initial builds, the project contains these key directories:

|  |  |  |
| --- | --- | --- |
| **Directory** | **Purpose** | **Generated By** |
| src/ | React application source code | Manual development |
| electron/ | Electron main and preload processes | Manual development |
| dist/ | Web production build output | npm run buildweb |
| dist-electron/ | Electron main process build | npm run build |
| release/ | Platform-specific installers | npm run build |
| node\_modules/ | Package dependencies | npm install |

The build system generates different output directories based on the target platform, with dist/ containing web assets and dist-electron/ containing Electron-specific compiled code.

**Sources:** [vite.config.ts:17-25](file:///C:\vite.config.ts)

## Next Steps

After completing installation and setup:

1. Start development using npm run web for browser testing or npm run dev for desktop development

1. Configure API credentials in the .env file for basketball simulation services

1. Review authentication setup documentation in [Authentication System](file:///C:\Repositorios\md2docx\output\6_Authentication_System.md)

1. Explore the primary game interface documentation in [Game Setup Interface](file:///C:\Repositorios\md2docx\output\10_Game_Setup_Interface.md)

The application requires valid API credentials for the external basketball simulation service to access game data and run simulations. For information about state management and context providers, see [State Management](file:///C:\Repositorios\md2docx\output\5_State_Management.md).

**Sources:** [.env.example:2-5](file:///C:\.env.example)

# Architecture

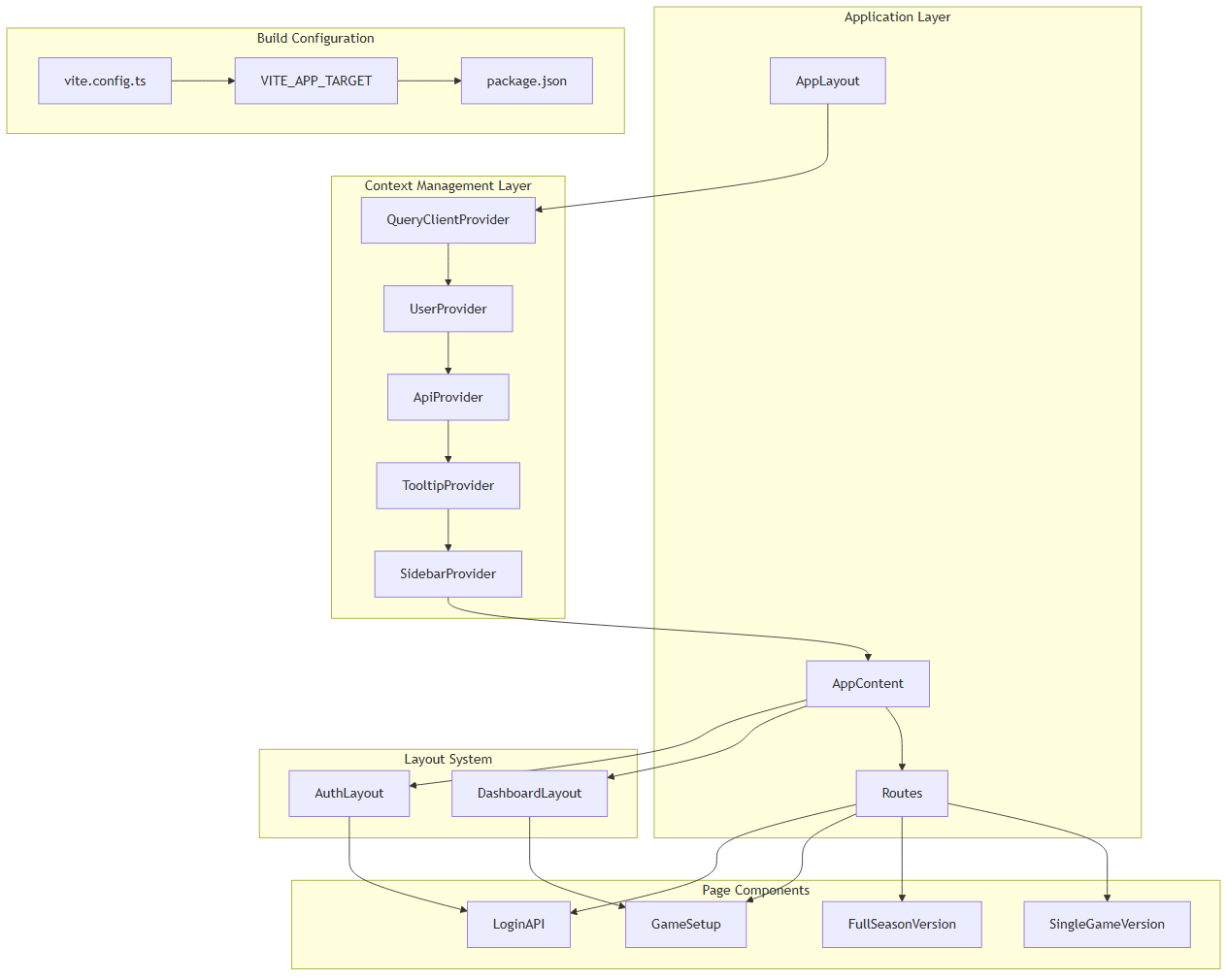
This document explains the core architectural patterns and design decisions of the NBA simulation application. It covers the React application structure, context-based state management system, conditional layout patterns, and dual-target build configuration that enables both web and desktop deployment from a single codebase.

For specific implementation details about individual contexts, see [State Management](#_State_Management). For authentication flow specifics, see [Authentication System](#_Authentication_System). For build configuration details, see [Build System](#_Build_System).

## System Architecture Overview

The application implements a layered React architecture with centralized state management through React Context. The system is designed around a provider composition pattern where multiple specialized contexts are aggregated into a unified interface.

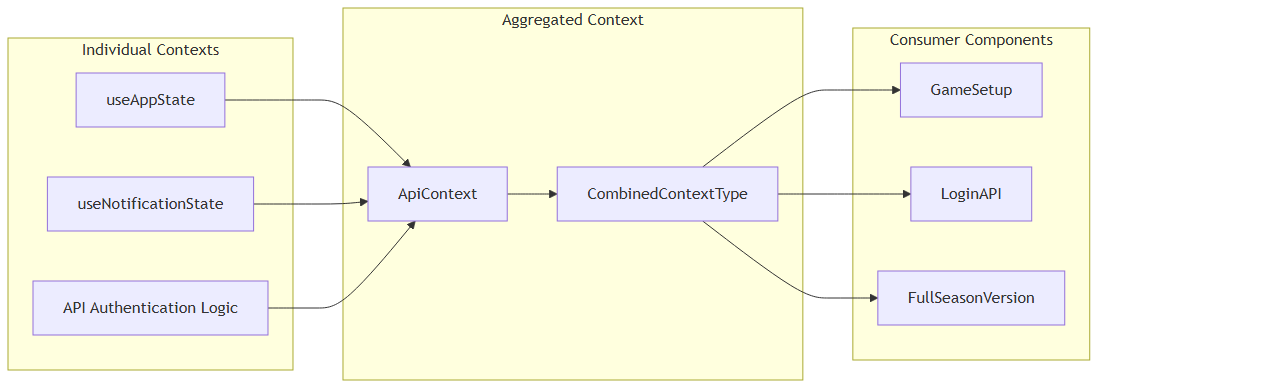
### Core System Layers



Sources: [src/layout.tsx:87-103](file:///C:\src\layout.tsx), [src/main.tsx:8-12](file:///C:\src\main.tsx), [vite.config.ts:8-11](file:///C:\vite.config.ts)

### Provider Composition Architecture

The application uses a sophisticated provider hierarchy where each context wraps the next, creating cascading availability of services. The ApiProvider serves as the primary aggregator, combining multiple context slices into a unified interface.

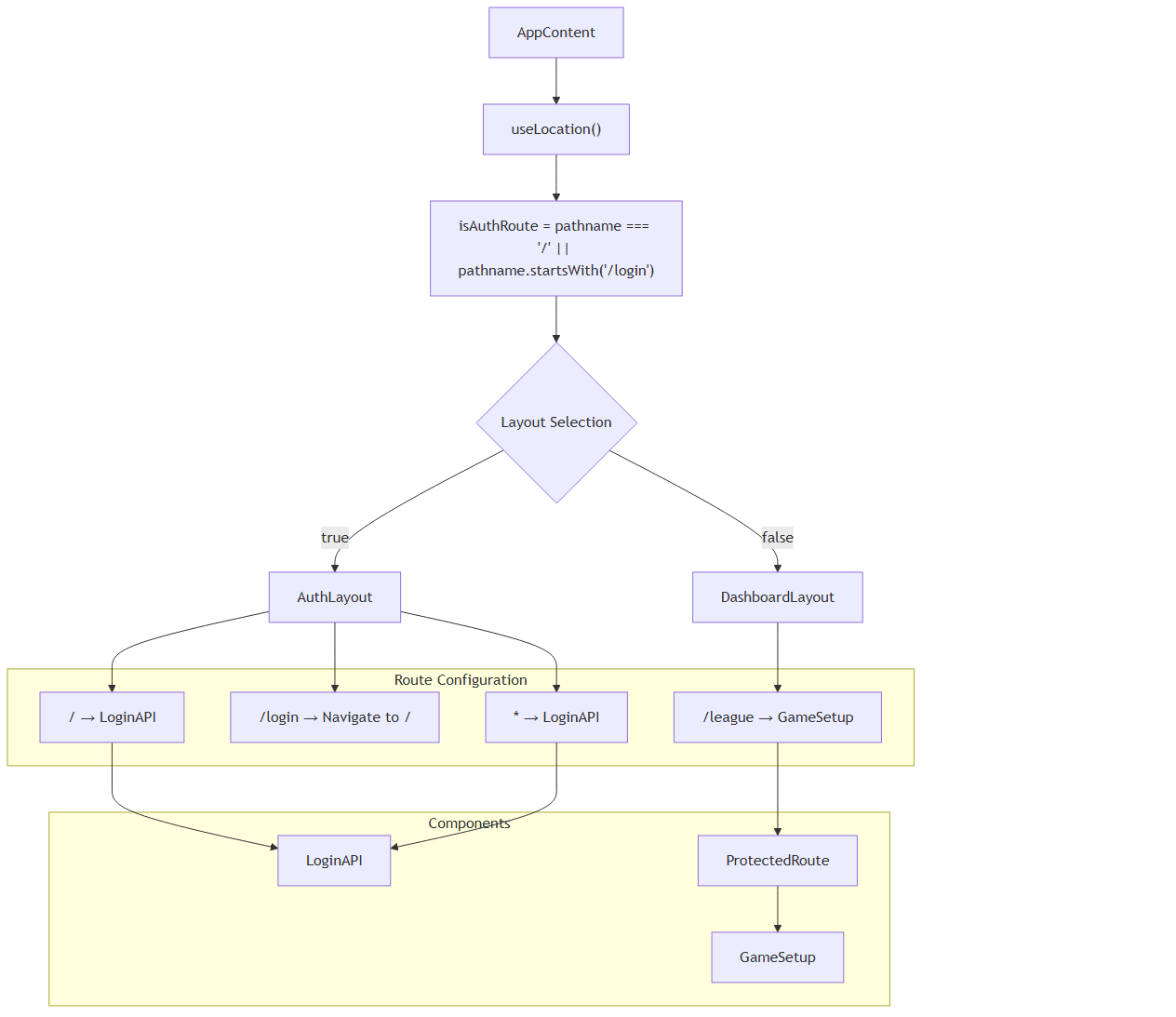


Sources: [src/contexts/ApiContext.tsx:13-21](file:///C:\src\contexts\ApiContext.tsx), [src/contexts/ApiContext.tsx:72-82](file:///C:\src\contexts\ApiContext.tsx)

## Routing and Layout Selection System

The application implements a conditional layout system that dynamically selects between AuthLayout and DashboardLayout based on the current route pattern. This enables different UI structures for authentication and main application flows.

### Route-Based Layout Logic

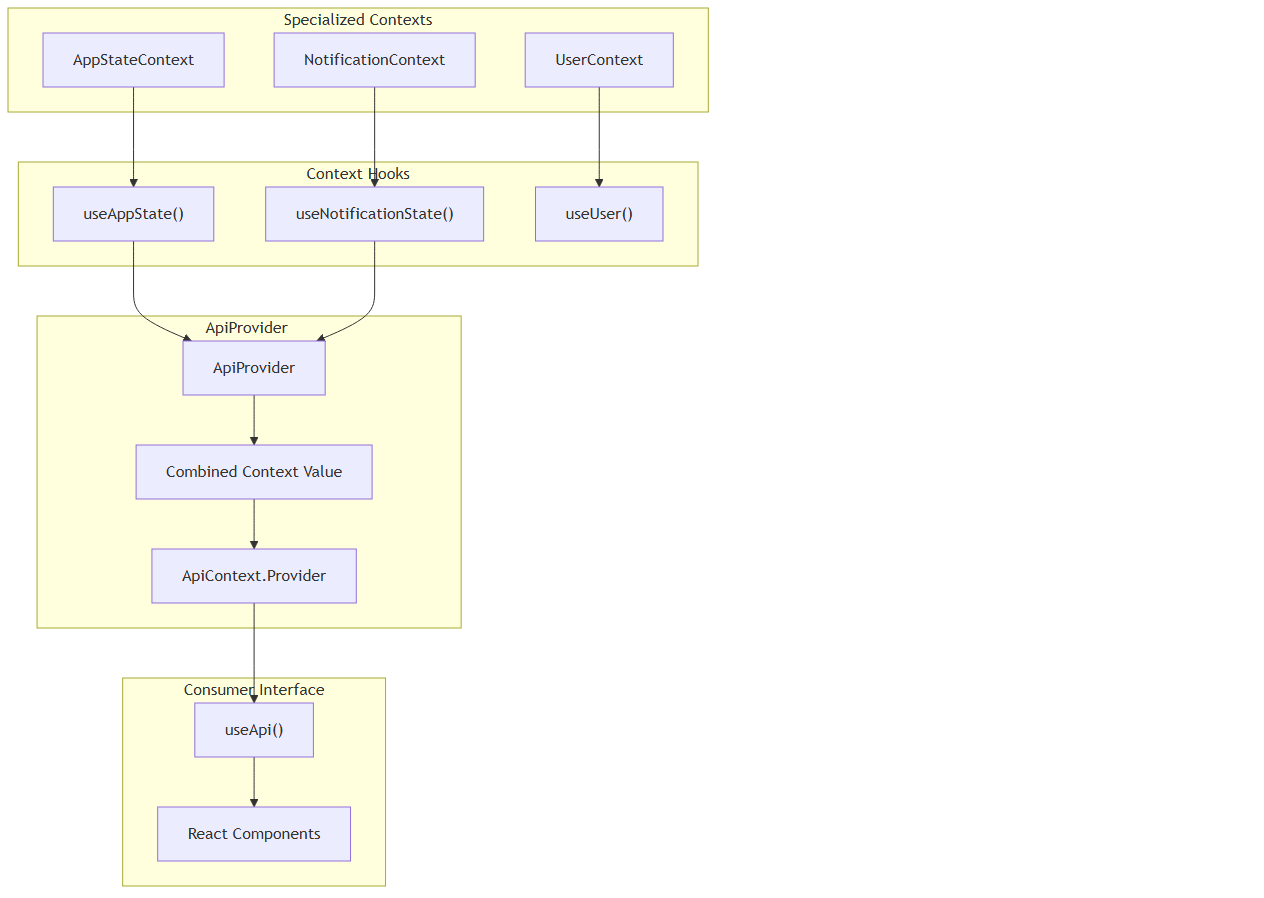


Sources: [src/layout.tsx:51-56](file:///C:\src\layout.tsx), [src/layout.tsx:68-82](file:///C:\src\layout.tsx)

## Context Aggregation Pattern

The application implements a unique context aggregation pattern where the ApiProvider combines multiple specialized contexts into a single unified interface. This reduces complexity for consuming components while maintaining separation of concerns.

### Context Integration Flow

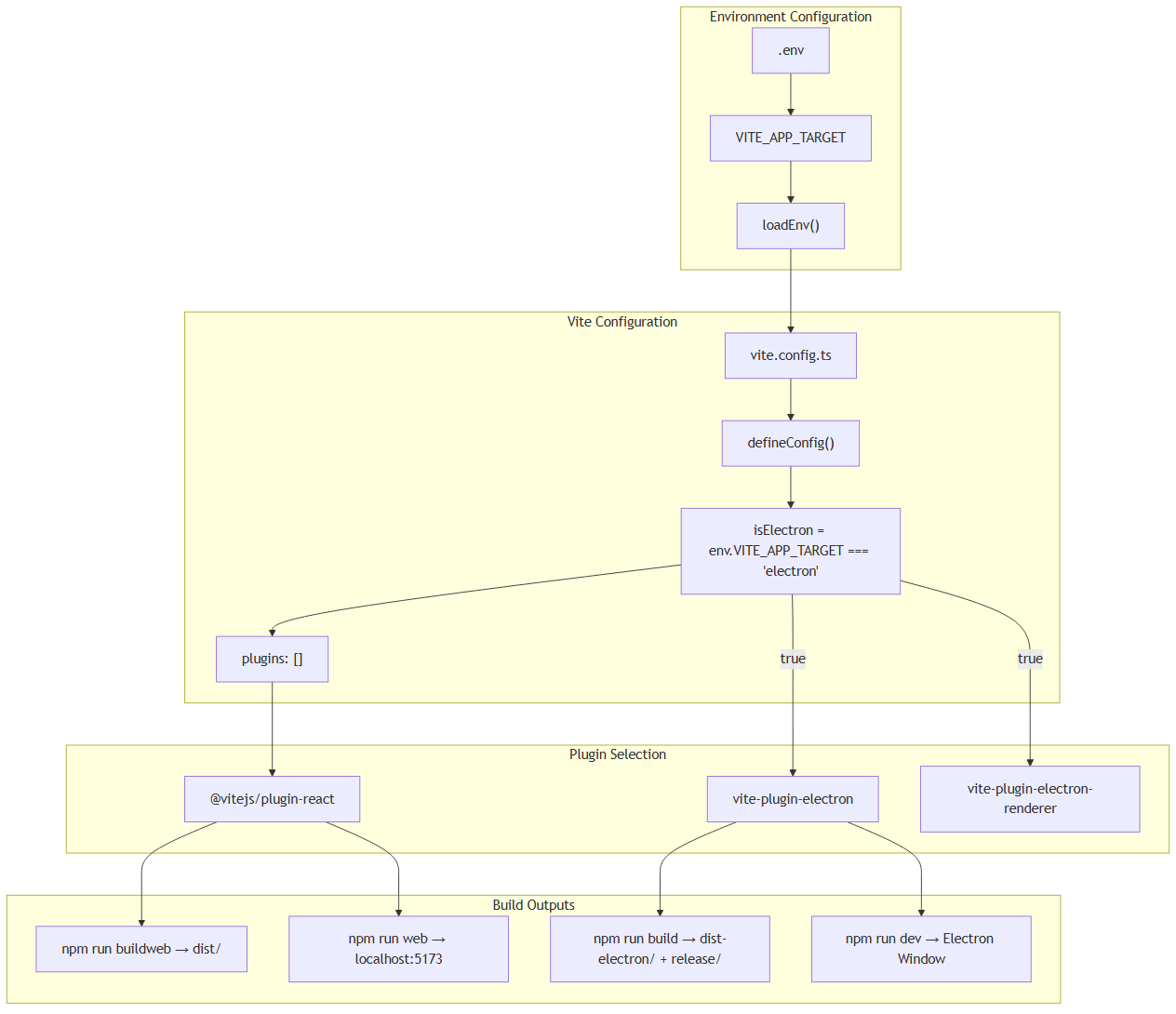


Sources: [src/contexts/ApiContext.tsx:24-88](file:///C:\src\contexts\ApiContext.tsx), [src/contexts/ApiContext.tsx:91-97](file:///C:\src\contexts\ApiContext.tsx)

## Dual-Target Build Architecture

The build system enables deployment to both web browsers and Electron desktop applications through environment-driven configuration. The VITE\_APP\_TARGET environment variable controls build behavior without requiring code changes.

### Build Configuration Flow

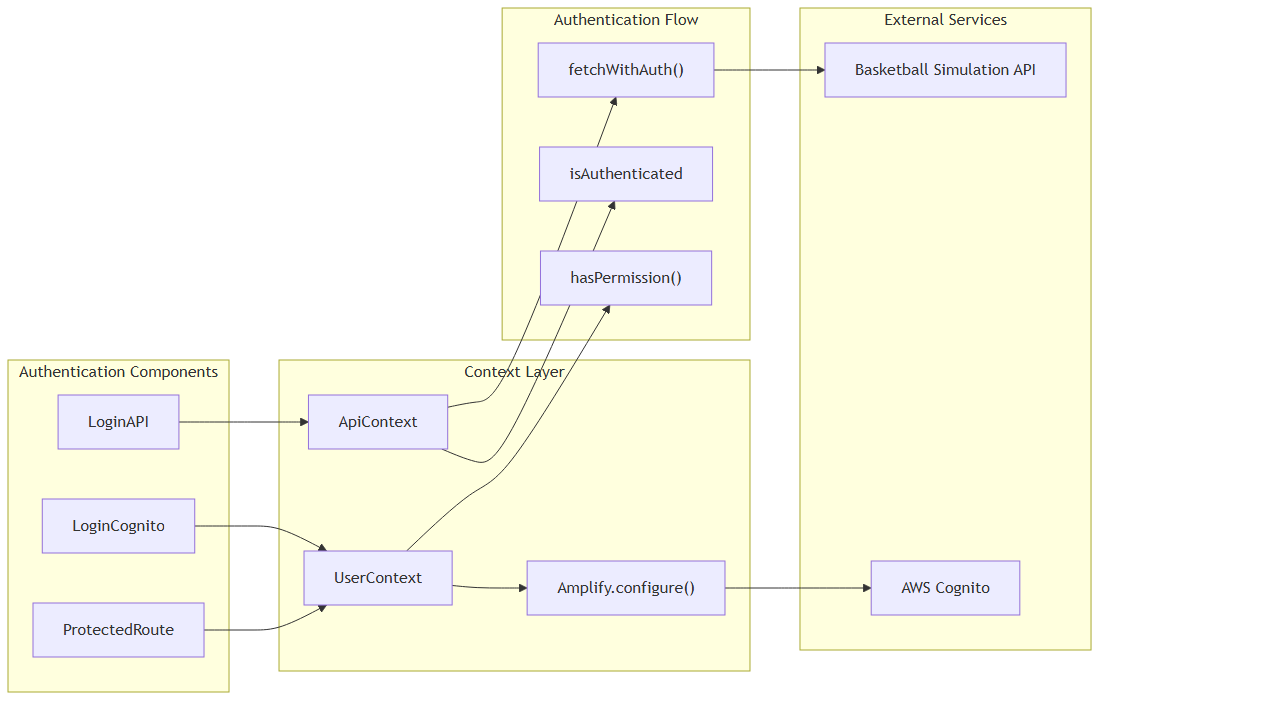


Sources: [vite.config.ts:8-11](file:///C:\vite.config.ts), [vite.config.ts:16-34](file:///C:\vite.config.ts), [package.json:13-20](file:///C:\package.json)

## Authentication System Integration

The application implements a dual authentication system: API key-based authentication for the basketball simulation service and AWS Amplify integration for user management. The ApiContext manages API authentication while UserContext handles user session management.

### Authentication Architecture

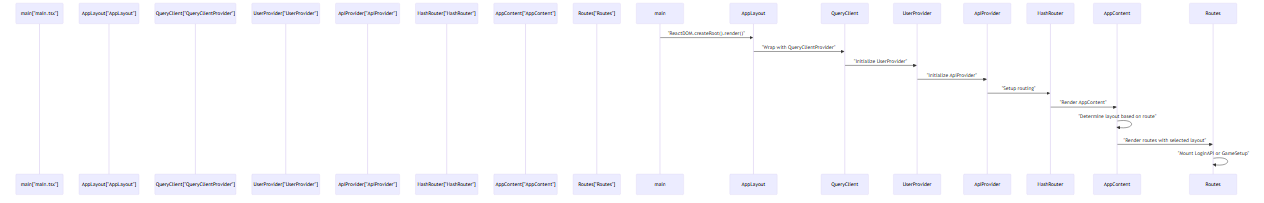


Sources: [src/layout.tsx:31-49](file:///C:\src\layout.tsx), [src/layout.tsx:73-77](file:///C:\src\layout.tsx), [src/contexts/ApiContext.tsx:45-70](file:///C:\src\contexts\ApiContext)

## Application Initialization Sequence

The application follows a specific initialization flow that establishes the provider hierarchy, configures routing, and sets up the authentication system before rendering any user interface components.

### Initialization Flow



Sources: [src/main.tsx:8-12](file:///C:\src\main.tsx), [src/layout.tsx:87-103](file:///C:\src\layout.tsx), [src/layout.tsx:51-84](file:///C:\src\layout.tsx)

# Application Structure

This document covers the foundational architecture of the NBA simulation application, including entry points, provider stack configuration, routing system, and layout components. For information about state management patterns within these components, see [State Management](#_State_Management). For details about the authentication flows that integrate with this structure, see [Authentication System](#_Authentication_System).

## Entry Points and Application Initialization

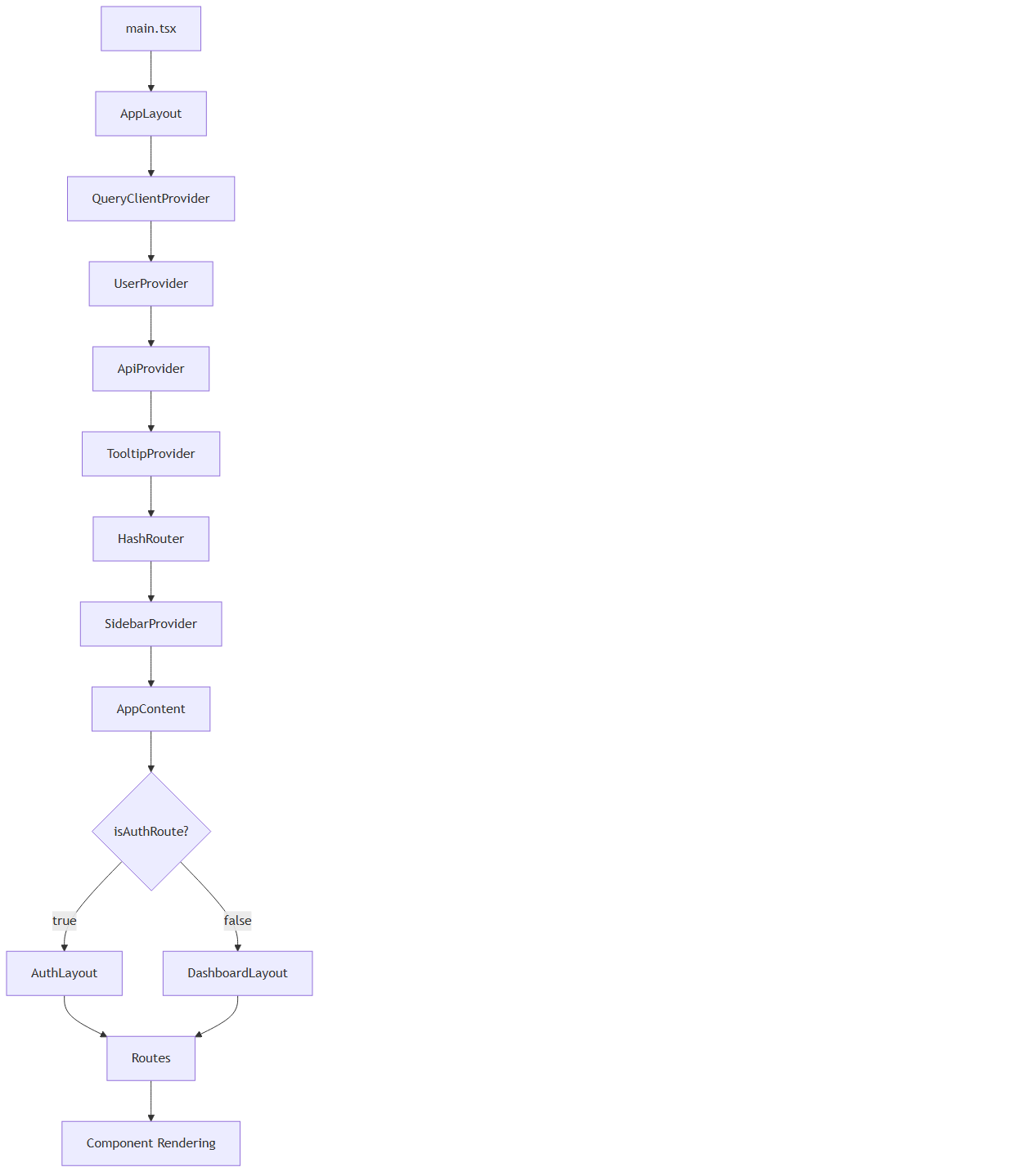
The application has a single entry point that initializes the React application and establishes the foundation for both web and Electron deployment targets.

### Main Entry Point

The application starts at main.tsx, which renders the root AppLayout component:

ReactDOM.createRoot(document.getElementById('root')!).render(  
 <React.StrictMode>  
 <AppLayout />  
 </React.StrictMode>,  
)

**Application Initialization Flow**



Sources: [src/main.tsx:8-12](file:///C:\src\main.tsx), [src/layout.tsx:87-103](file:///C:\src\layout.tsx)

## Provider Stack Architecture

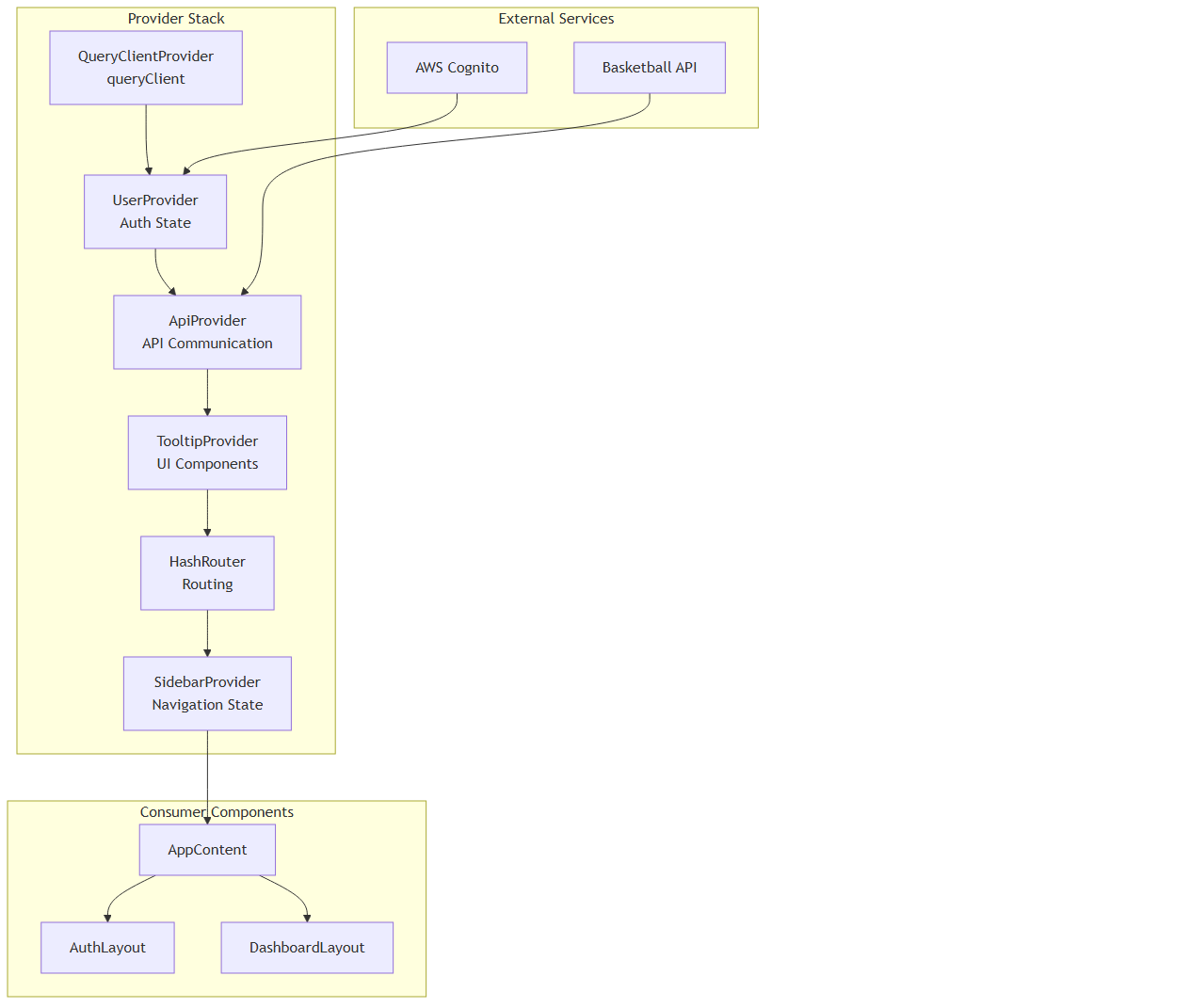
The AppLayout component establishes a comprehensive provider stack that makes essential services available throughout the application component tree.

### Provider Hierarchy

The providers are wrapped in a specific order to ensure proper dependency resolution:

|  |  |  |
| --- | --- | --- |
| **Provider** | **Purpose** | **Dependencies** |
| QueryClientProvider | React Query client for data fetching | None |
| UserProvider | AWS Cognito authentication state | QueryClient |
| ApiProvider | Basketball API communication | UserProvider |
| TooltipProvider | UI tooltip functionality | None |
| HashRouter | Client-side routing | None |
| SidebarProvider | Sidebar state management | None |

**Provider Dependencies and Services**



Sources: [src/layout.tsx:87-103](file:///C:\src\layout.tsx)

## Routing System

The routing system uses a conditional layout strategy where different layouts are applied based on the current route path.

### Route-Based Layout Selection

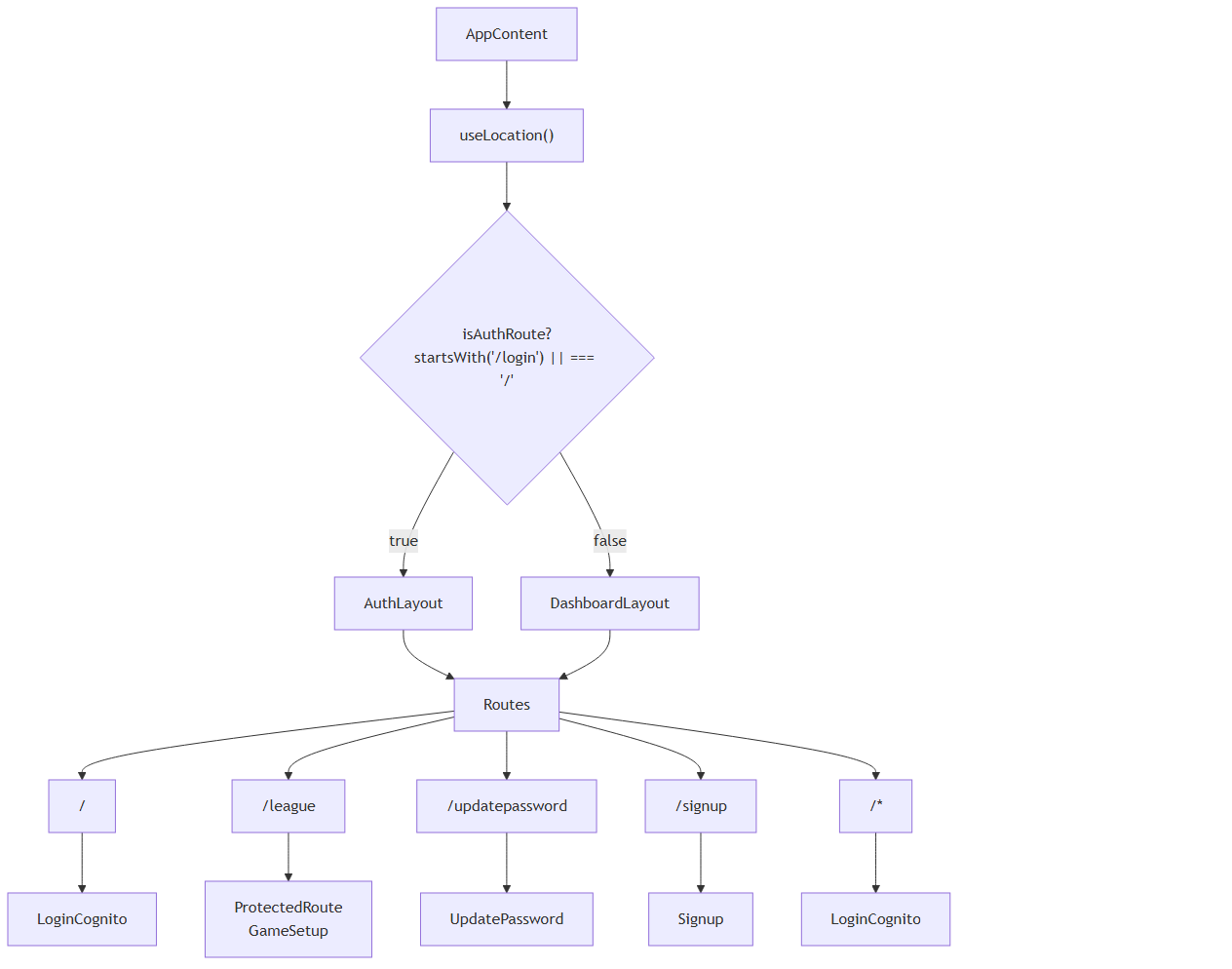
The AppContent component implements intelligent layout switching:

const isAuthRoute = location.pathname.startsWith("/login") || location.pathname === "/";  
const Layout = isAuthRoute ? AuthLayout : DashboardLayout;

### Route Configuration

|  |  |  |  |
| --- | --- | --- | --- |
| **Route Pattern** | **Component** | **Layout** | **Protection** |
| / | LoginCognito | AuthLayout | None |
| /login | Redirect to / | AuthLayout | None |
| /league | GameSetup | DashboardLayout | ProtectedRoute |
| /updatepassword | UpdatePassword | AuthLayout | None |
| /signup | Signup | AuthLayout | None |
| \* (catch-all) | LoginCognito | AuthLayout | None |

**Routing and Layout Decision Flow**



Sources: [src/layout.tsx:51-85](file:///C:\src\layout.tsx)

## Layout Components

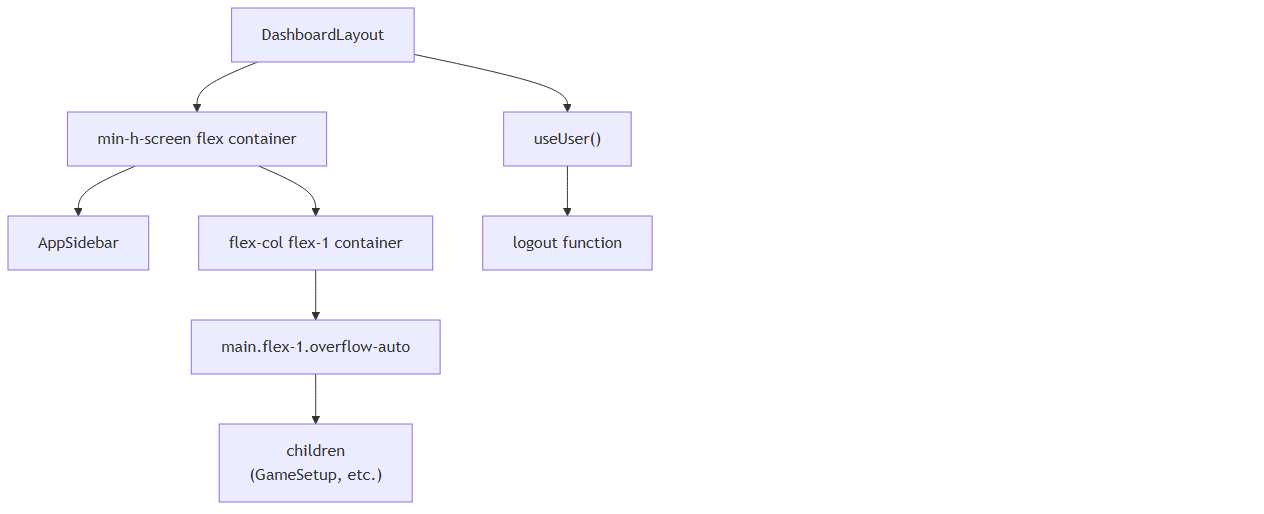
The application uses two primary layout components that provide different structural frameworks for authentication and main application flows.

### DashboardLayout Structure

The DashboardLayout provides the main application interface for authenticated users:

const DashboardLayout = ({ children }: { children: React.ReactNode }) => {  
 return (  
 <div className="min-h-screen flex w-full">  
 <AppSidebar />  
 <div className="flex flex-col flex-1">  
 <main className="flex-1 overflow-auto">{children}</main>  
 </div>  
 </div>  
 );  
};

**DashboardLayout Component Structure**



Sources: [src/components/MainLayout.tsx:6-18](file:///C:\src\components\MainLayout.tsx)

### Layout Integration with Authentication

The layout system integrates with the authentication state through the useUser hook, which provides loading states and user management functions.

|  |  |  |
| --- | --- | --- |
| **Layout** | **Authentication State** | **Available Components** |
| AuthLayout | Unauthenticated | Login, Signup, Password Reset |
| DashboardLayout | Authenticated | Sidebar, Game Setup, Logout |

## Build Target Considerations

The application structure supports dual deployment through the VITE\_APP\_TARGET environment variable, configured in vite.config.ts:

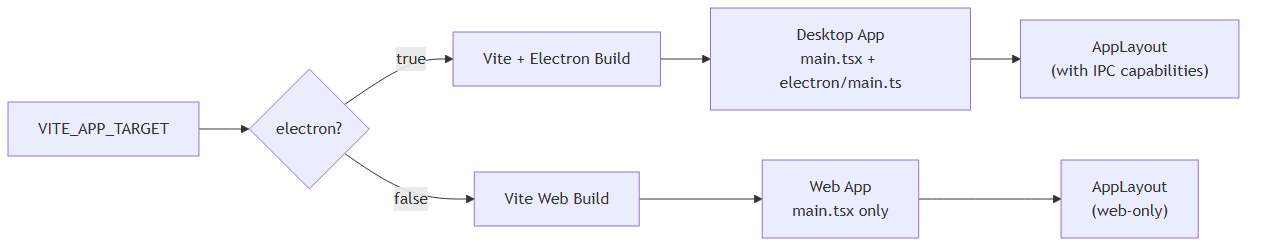
const isElectron = env.VITE\_APP\_TARGET === 'electron';

This affects how the application initializes and which features are available:

• **Web Target**: Standard React web application with hash routing

• **Electron Target**: Desktop application with additional native integrations

**Build Target Application Flow**

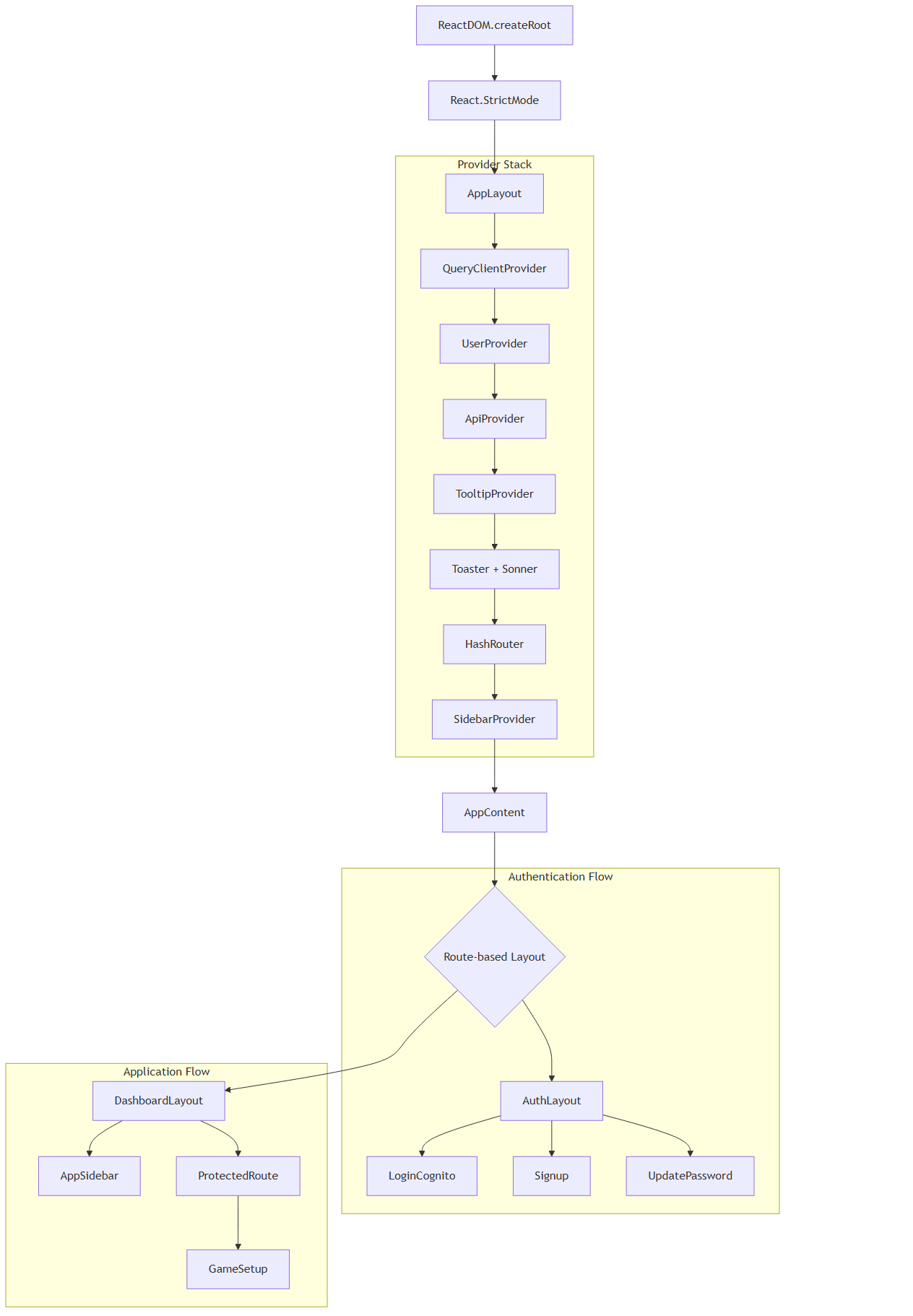


Sources: [vite.config.ts:8-11](file:///C:\vite.config.ts), [src/main.tsx:14-19](file:///C:\src\main.tsx)

## Component Hierarchy Overview

The complete component hierarchy flows from the entry point through the provider stack to the final rendered components:

**Complete Application Component Tree**



Sources: [src/main.tsx:8-12](file:///C:\src\main.tsx), [src/layout.tsx:87-103](file:///C:\src\layout.tsx), [src/layout.tsx:51-85](file:///C:\src\layout.tsx), [src/components/MainLayout.tsx:6-18](file:///C:\src\components\MainLayout.tsx)

# State Management

## Purpose and Scope

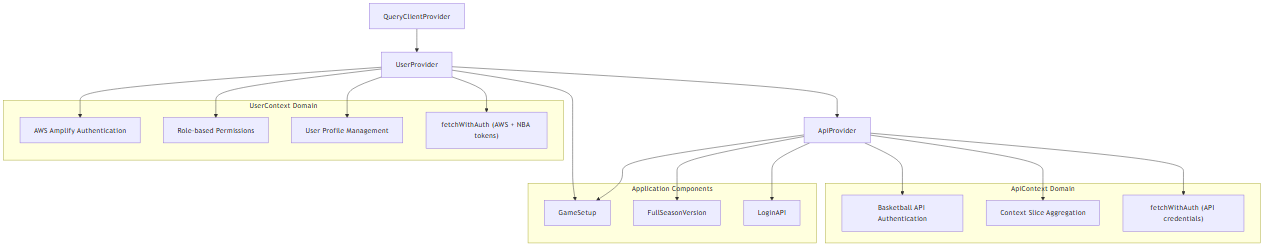
This document covers the React Context-based state management system used throughout the NBA simulation application. The system implements a dual authentication architecture with context aggregation patterns that combine multiple specialized contexts into unified interfaces. This eliminates prop drilling while maintaining separation of concerns across user authentication, API authentication, application state, and notifications.

For information about AWS Amplify authentication specifically, see [Authentication System](#_Authentication_System). For details about API integration patterns, see [API Integration](#_API_Integration_Architecture).

## Dual Authentication Context Architecture

The application implements two distinct authentication contexts that serve different purposes in the basketball simulation system:

### Overall Context System Architecture



**Sources:** [src/contexts/UserContext.tsx:32-47](file:///C:\src\contexts\UserContext.tsx), [src/contexts/ApiContext.tsx:12-21](file:///C:\src\contexts\ApiContext.tsx), [src/layout.tsx:44-60](file:///C:\src\layout.tsx)

### Authentication System Comparison

|  |  |  |
| --- | --- | --- |
| **Aspect** | **UserContext** | **ApiContext** |
| Purpose | AWS Amplify user authentication | Basketball API service authentication |
| Scope | User identity, permissions, profile | Game simulation API access |
| Token Management | AWS ID token + NBA custom token | API key + authorization header |
| Primary Hook | useUser() | useApi() |
| fetchWithAuth | Dual token system | API credentials system |

**Sources:** [src/contexts/UserContext.tsx:32-47](file:///C:\src\contexts\UserContext.tsx), [src/contexts/ApiContext.tsx:5-21](file:///C:\src\contexts\ApiContext.tsx)

## UserContext - AWS Amplify Authentication System

The UserContext provides comprehensive user authentication through AWS Amplify integration with a sophisticated dual token system for basketball API access.

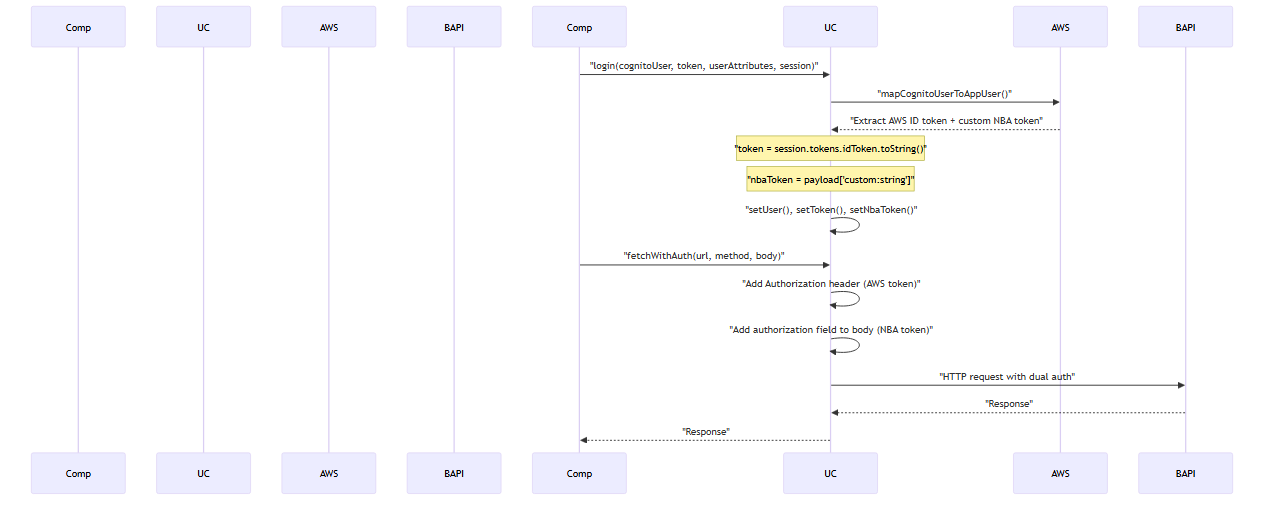
### UserContext State Management



**Sources:** [src/contexts/UserContext.tsx:17-30](file:///C:\src\contexts\UserContext.tsx), [src/contexts/UserContext.tsx:32-47](file:///C:\src\contexts\UserContext.tsx)

### Dual Token Authentication System

The UserContext implements a sophisticated dual token system for accessing both AWS services and basketball simulation APIs:



**Sources:** [src/contexts/UserContext.tsx:84-109](file:///C:\src\contexts\UserContext.tsx), [src/contexts/UserContext.tsx:166-175](file:///C:\src\contexts\UserContext.tsx), [src/contexts/UserContext.tsx:230-257](file:///C:\src\contexts\UserContext.tsx)

### Role-Based Permission System

The UserContext implements granular role-based permissions:

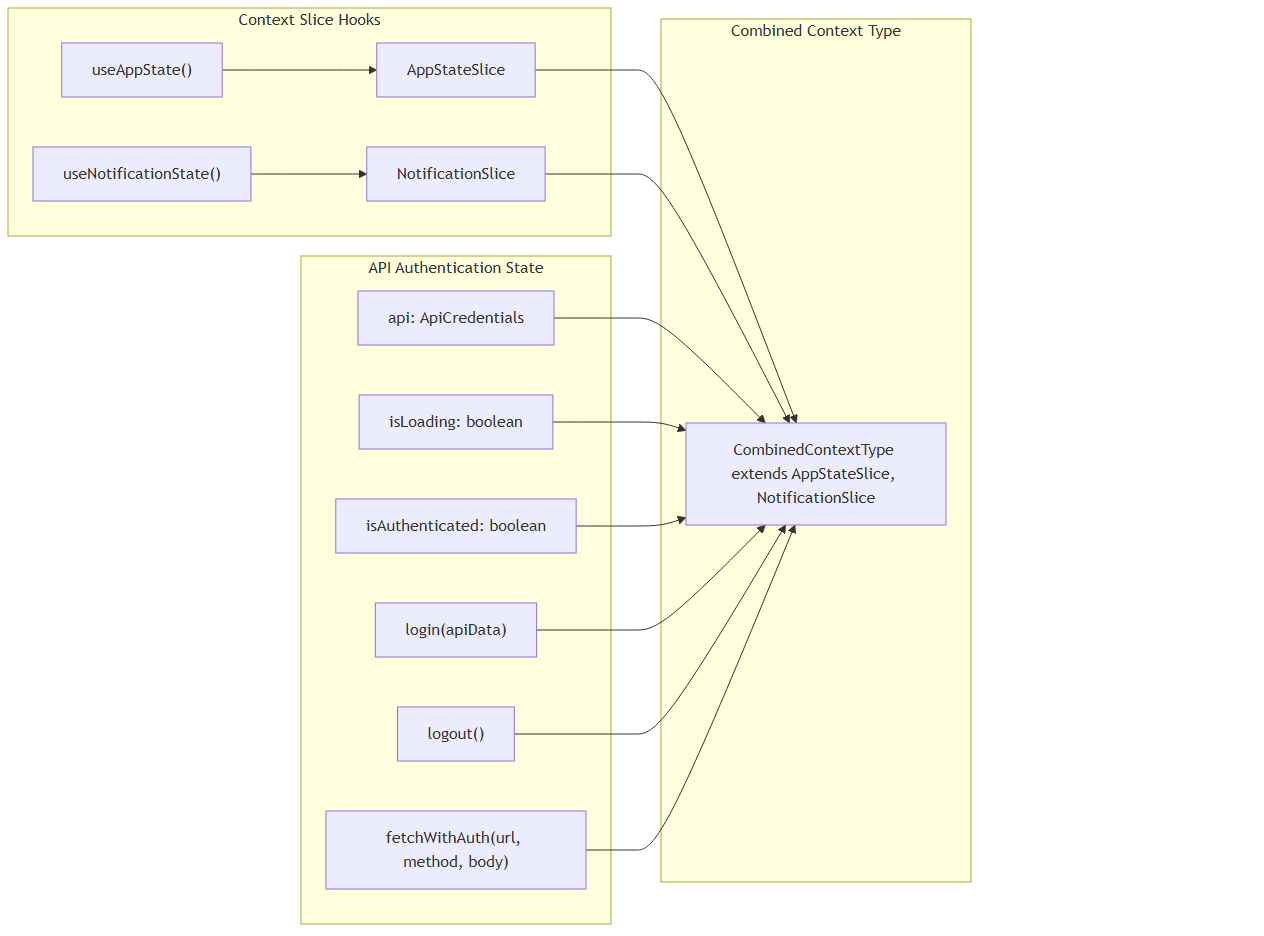
|  |  |
| --- | --- |
| **Role** | **Permissions** |
| admin | view\_all, add\_edit\_delete\_users, add\_edit\_records, delete\_records, edit\_profile |
| developer | view\_all, add\_edit\_records, delete\_records, edit\_profile |
| guest | view\_all |

**Sources:** src/contexts/UserContext.tsx:196-206

## ApiContext - Basketball API and Context Aggregation

The ApiContext serves dual purposes: basketball API authentication and aggregation of application state slices into a unified interface.

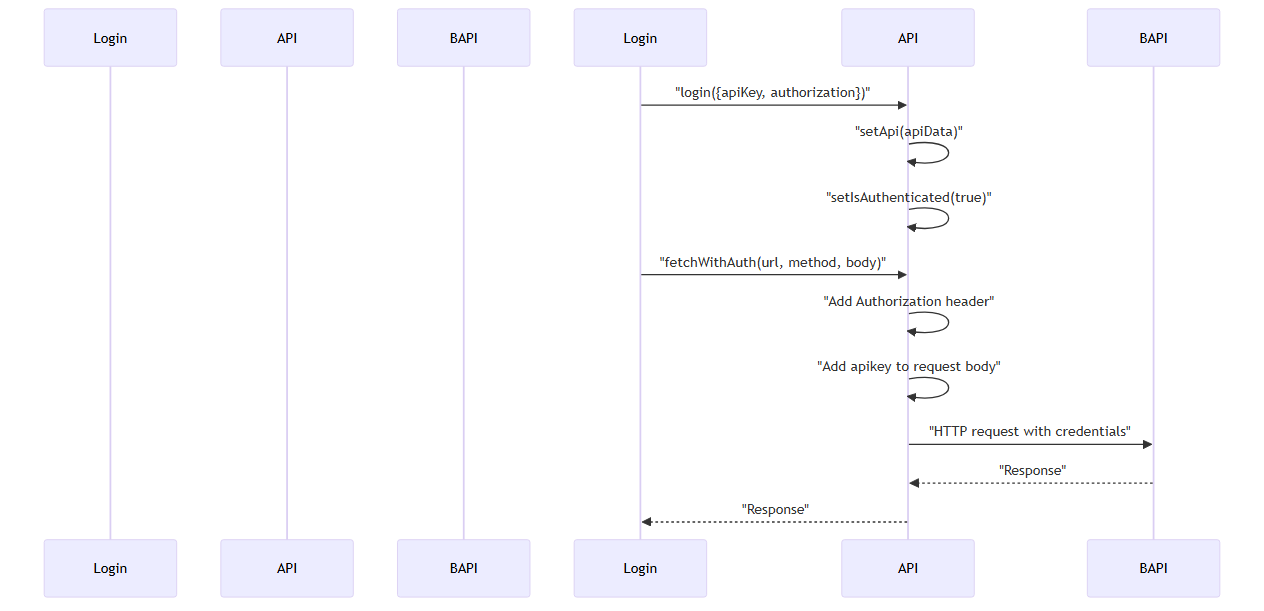
### ApiContext Composition Pattern



**Sources:** [src/contexts/ApiContext.tsx:12-21](file:///C:\src\contexts\ApiContext.tsx), [src/contexts/ApiContext.tsx:30-32](file:///C:\src\contexts\ApiContext.tsx), [src/contexts/ApiContext.tsx:81-91](file:///C:\src\contexts\ApiContext.tsx)

### Basketball API Authentication Flow

The ApiContext manages API key-based authentication for basketball simulation services:

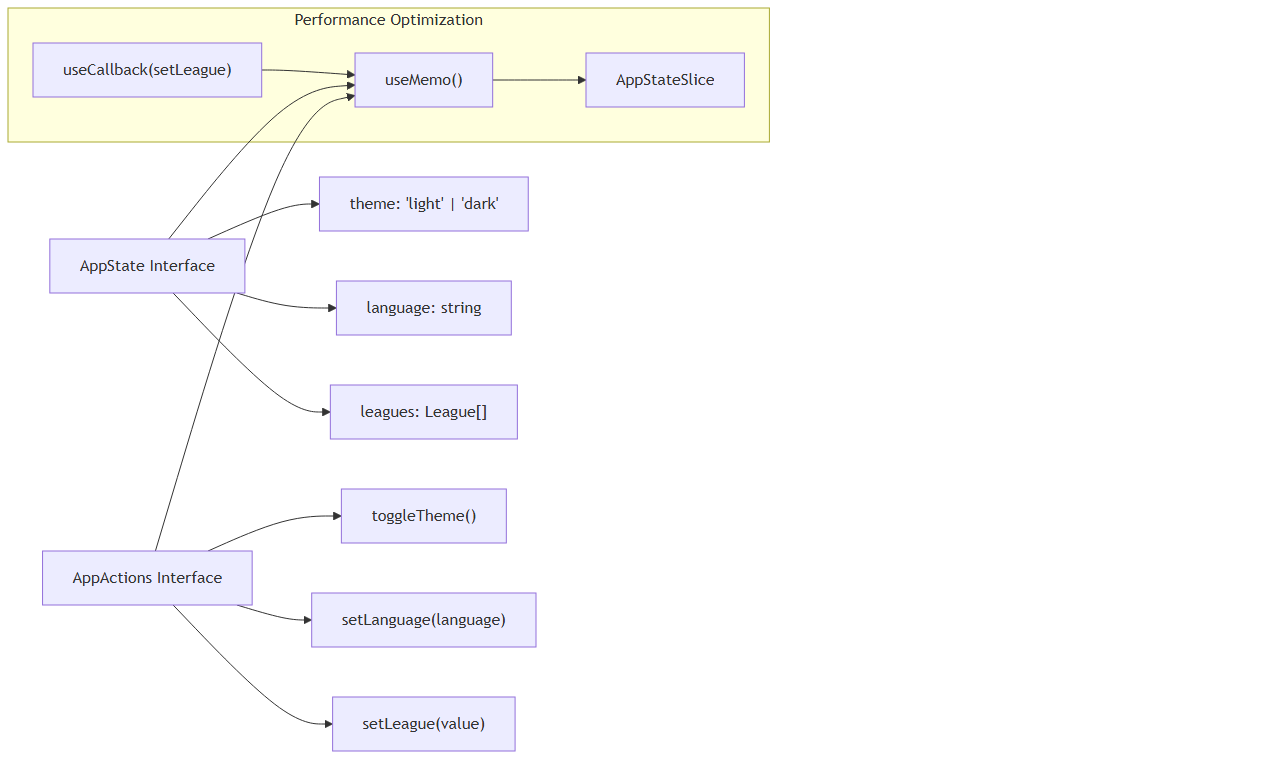


**Sources:** [src/contexts/ApiContext.tsx:34-39](file:///C:\src\contexts\ApiContext.tsx), [src/contexts/ApiContext.tsx:53-78](file:///C:\src\contexts\ApiContext.tsx)

## Individual Context Slice Implementations

### AppStateContext - Application Global State

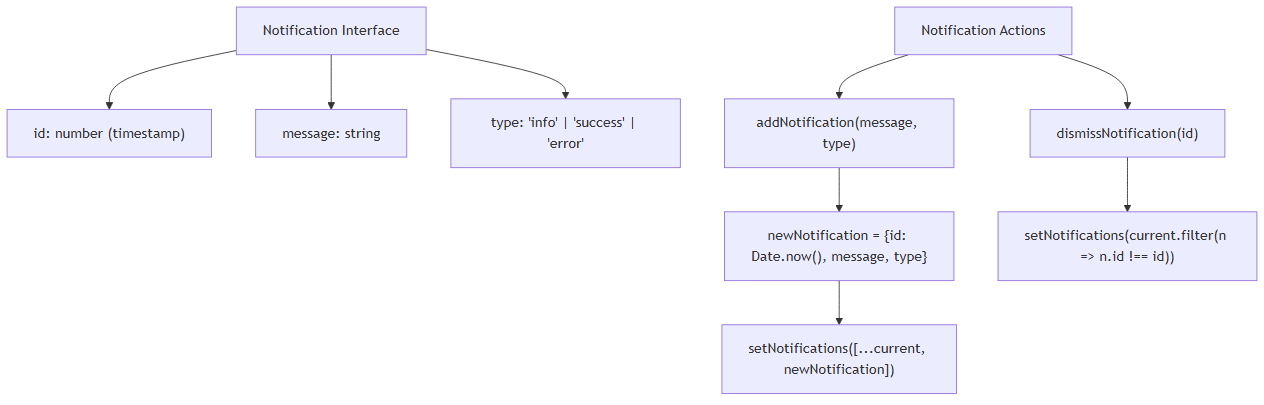
The AppStateContext manages theme, language, and cached league data using the slice pattern:



**Sources:** [src/contexts/AppStateContext.tsx:4-7](file:///C:\Repositorios\md2docx\output\src\contexts\AppStateContext.tsx), [src/contexts/AppStateContext.tsx:18-21](file:///C:\Repositorios\md2docx\output\src\contexts\AppStateContext.tsx), [src/contexts/AppStateContext.tsx:43-50](file:///C:\Repositorios\md2docx\output\src\contexts\AppStateContext.tsx)

### NotificationContext - Toast Notification Management

The NotificationContext implements a timestamp-based notification queue system:



**Sources:** [src/contexts/NotificationContext.tsx:4-8](file:///C:\Repositorios\md2docx\output\src\contexts\NotificationContext.tsx), [src/contexts/NotificationContext.tsx:31-36](file:///C:\Repositorios\md2docx\output\src\contexts\NotificationContext.tsx), [src/contexts/NotificationContext.tsx:38-41](file:///C:\Repositorios\md2docx\output\src\contexts\NotificationContext.tsx)

### UserContext Implementation Details

The UserContext integrates with AWS Amplify through the AuthService and implements sophisticated token management:

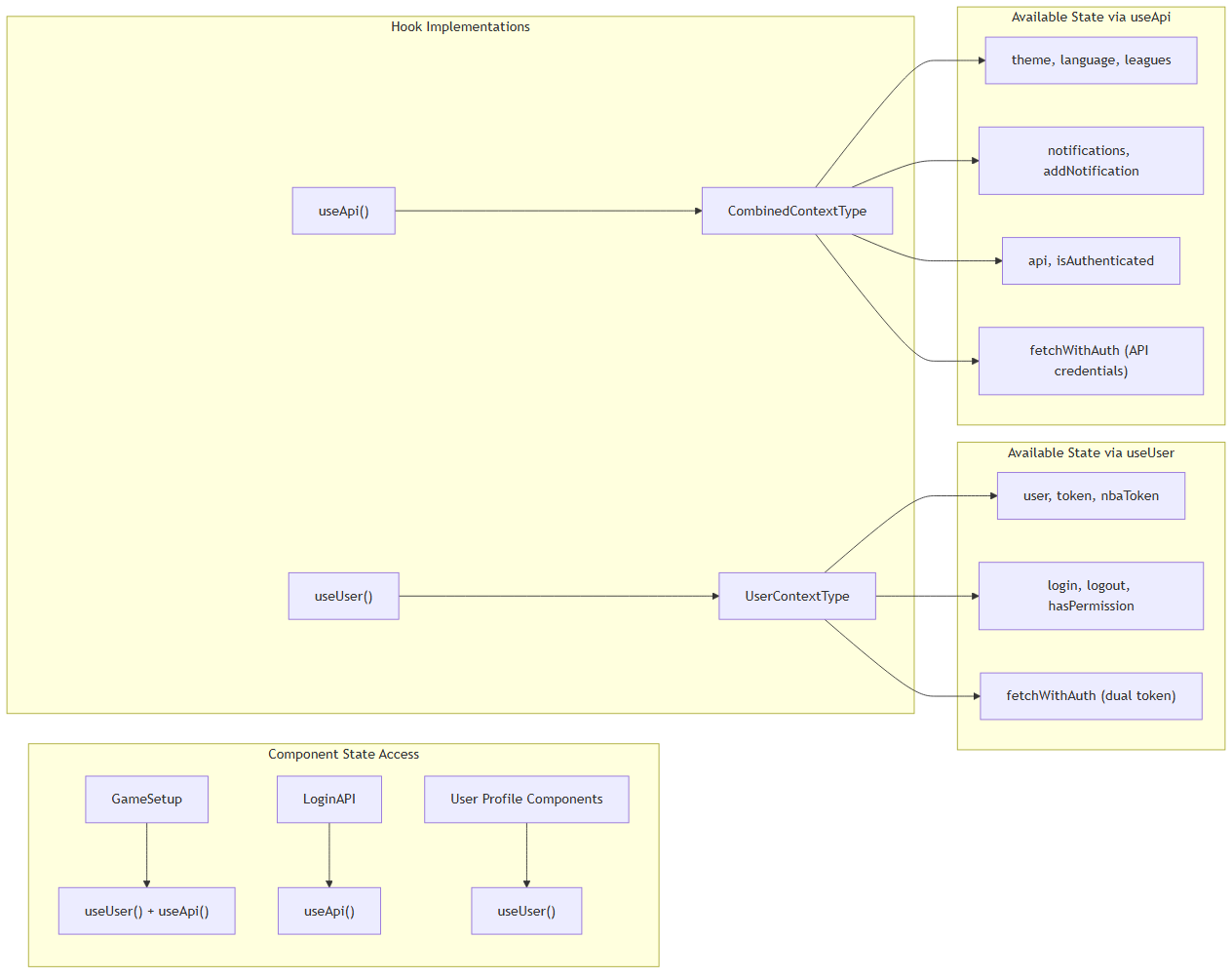
|  |  |  |
| --- | --- | --- |
| **State Field** | **Type** | **Purpose** |
| user | User \| null | Current user profile with AWS and custom fields |
| token | string \| null | AWS ID token for API authorization headers |
| nbaToken | string \| null | NBA custom token for API request bodies |
| isAuthenticated | boolean | Authentication status |
| isLoading | boolean | Authentication loading state |
| **Method** | **Purpose** | **Key Implementation** |
| loadUser | Initialize user from AWS session | src/contexts/UserContext.tsx:124-163 |
| login | Map Cognito user to app user | src/contexts/UserContext.tsx:166-175 |
| fetchWithAuth | Dual token HTTP requests | src/contexts/UserContext.tsx:230-257 |
| hasPermission | Role-based access control | src/contexts/UserContext.tsx:196-206 |

**Sources:** [src/contexts/UserContext.tsx:32-47](file:///C:\Repositorios\md2docx\output\src\contexts\UserContext.tsx), [src/contexts/UserContext.tsx:124-163](file:///C:\Repositorios\md2docx\output\src\contexts\UserContext.tsx), [src/contexts/UserContext.tsx:230-257](file:///C:\Repositorios\md2docx\output\src\contexts\UserContext.tsx)

## State Access Patterns

### Dual Hook System

Components access state through two primary hooks based on their requirements:



**Sources:** [src/contexts/UserContext.tsx:291-297](file:///C:\src\contexts\UserContext.tsx), [src/contexts/ApiContext.tsx:100-106](file:///C:\src\contexts\ApiContext.tsx)

### Context Provider Hierarchy in Application

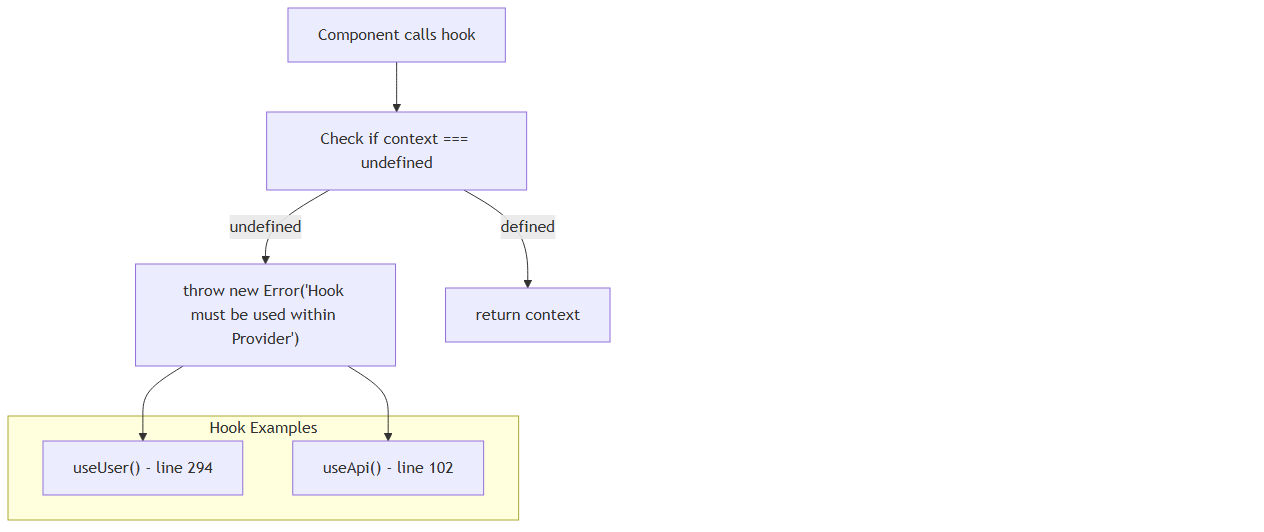
The application establishes a specific provider nesting order in the layout system:

|  |  |  |  |
| --- | --- | --- | --- |
| **Layer** | **Provider** | **Purpose** | **File Reference** |
| 1 | QueryClientProvider | TanStack Query client | src/layout.tsx:44 |
| 2 | UserProvider | AWS Amplify authentication | src/layout.tsx:45 |
| 3 | ApiProvider | Basketball API + context aggregation | src/layout.tsx:46 |
| 4 | TooltipProvider | UI tooltip system | src/layout.tsx:47 |
| 5 | HashRouter | Client-side routing | src/layout.tsx:50 |
| 6 | SidebarProvider | Sidebar state management | src/layout.tsx:51 |

**Sources:** [src/layout.tsx:44-60](file:///C:\src\layout.tsx)

### Error Handling and Runtime Validation

Both context hooks implement runtime validation to ensure proper provider hierarchy:



**Sources:** [src/contexts/UserContext.tsx:293-296](file:///C:\src\contexts\UserContext.tsx), [src/contexts/ApiContext.tsx:102-105](file:///C:\src\contexts\ApiContext.tsx)

## Performance Optimizations

### Memoization Strategy

Both AppStateContext and NotificationContext implement useMemo to stabilize context values:

• AppStateContext memoizes the complete state object at /src/contexts/AppStateContext.tsx:43-50

• NotificationContext memoizes the return value at /src/contexts/NotificationContext.tsx:45-49

• Functions are wrapped in useCallback to prevent recreation on every render

### Context Composition Benefits

The aggregation pattern provides several performance advantages:  
- Single context subscription reduces re-render frequency  
- Memoized slice implementations prevent cascading updates  
- Centralized state access eliminates multiple context subscriptions per component

**Sources:** [src/contexts/ApiContext.tsx:72-82](file:///C:\src\contexts\ApiContext), [src/contexts/AppStateContext.tsx:43-50](file:///C:\src\contexts\AppStateContext.tsx), [src/contexts/NotificationContext.tsx:45-49](file:///C:\src\contexts\NotificationContext.tsx)

# Authentication System

## Purpose and Scope

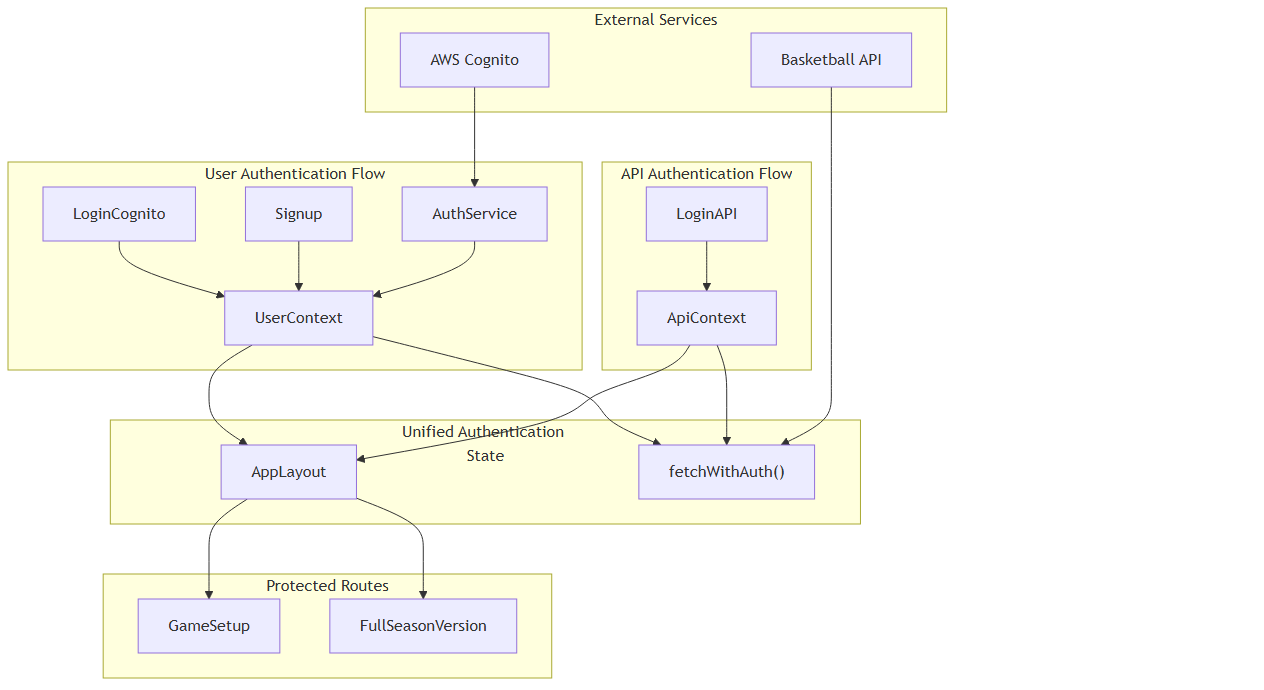
This document covers the dual authentication system implemented in the NBA simulation application. The system supports both AWS Cognito user account authentication and API key-based authentication for accessing basketball simulation services. For information about the broader state management architecture, see [State Management](#_State_Management). For details about the user interface components, see [Authentication UI](#_Authentication_UI).

The authentication system provides unified user session management, token handling, permission-based access control, and seamless integration between user accounts and API service credentials.

## Authentication Architecture Overview

The application implements a dual authentication strategy to handle two distinct authentication needs: user identity management through AWS Cognito and service access through API credentials.

### System Architecture

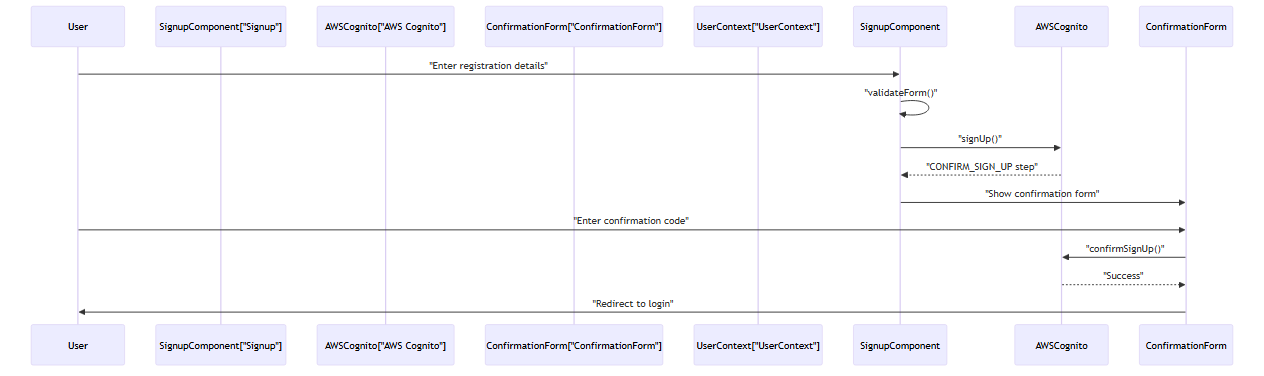


Sources: [src/contexts/UserContext.tsx:1-298](file:///C:\src\contexts\UserContext.tsx), [src/pages/auth/loginCognito.tsx:1-314](file:///C:\src\pages\auth\loginCognito.tsx), [src/pages/auth/signup.tsx:1-321](file:///C:\src\pages\auth\signup.tsx), [src/LoginAPI.tsx:1-172](file:///C:\src\LoginAPI.tsx), [src/contexts/AuthService.ts:1-62](file:///C:\%5bsrc\contexts\AuthService.ts)

## AWS Cognito Authentication

### User Registration and Confirmation

The signup process implements a two-step verification flow using AWS Cognito's built-in email confirmation system.



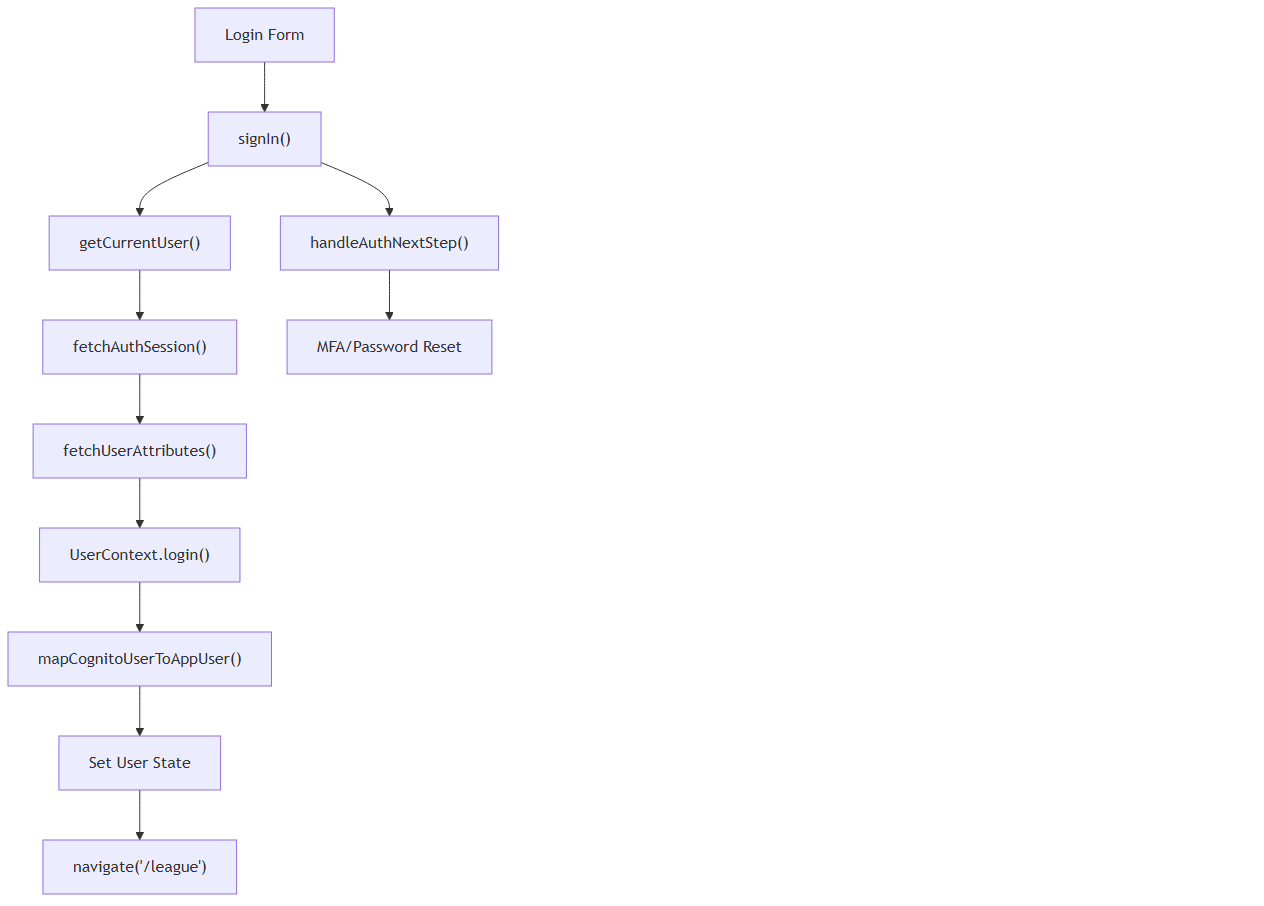
The Signup component handles user registration with the following key features:

|  |  |  |
| --- | --- | --- |
| **Feature** | **Implementation** | **Location** |
| Form validation | validateForm() function | src/pages/auth/signup.tsx:68-89 |
| User attributes | Custom fields including custom:string for NBA token | src/pages/auth/signup.tsx:110-116 |
| Confirmation flow | Two-step process with email verification | src/pages/auth/signup.tsx:124-130 |
| Code resending | handleResendCode() for retry functionality | src/pages/auth/signup.tsx:176-195 |

Sources: [src/pages/auth/signup.tsx:35-321](file:///C:\src\pages\auth\signup.tsx), [src/pages/auth/ConfirmationForm:1-210](file:///C:\src\pages\auth\ConfirmationForm.tsx)

### Login Flow and Session Management

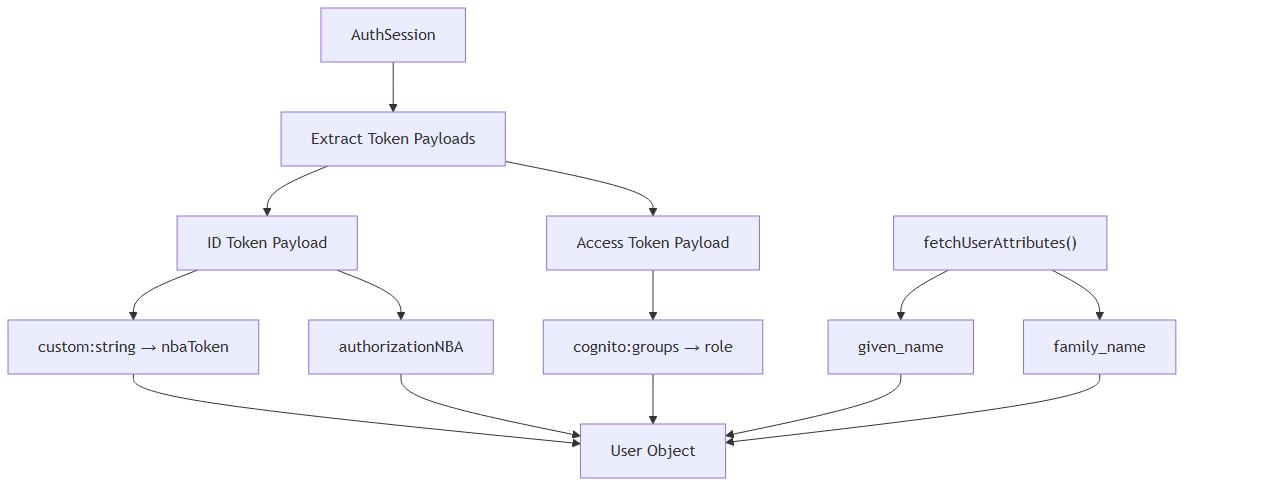
The LoginCognito component manages AWS Cognito authentication with comprehensive error handling and session establishment.



The authentication process extracts multiple token types and user attributes through the mapCognitoUserToAppUser function:

|  |  |  |  |
| --- | --- | --- | --- |
| **Token/Attribute** | **Purpose** | **Source** | **Code Location** |
| idToken | Primary authentication token | session.tokens.idToken | src/pages/auth/loginCognito.tsx:62 |
| accessToken | Role assignment via groups | session.tokens.accessToken | src/contexts/UserContext.tsx:87 |
| custom:string | NBA API authorization token | idToken.payload["custom:string"] | src/contexts/UserContext.tsx:102 |
| authorizationNBA | Additional NBA authorization | idToken.payload["authorizationNBA"] | src/contexts/UserContext.tsx:103 |
| cognito:groups | User role assignment | accessToken.payload["cognito:groups"] | src/contexts/UserContext.tsx:88 |

#### User Mapping Process

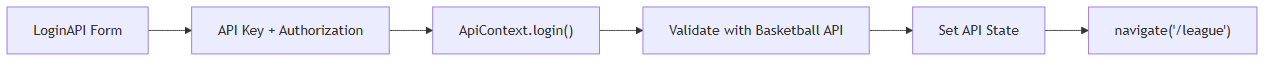


Sources: [src/pages/auth/loginCognito.tsx:59-82](file:///C:\src\pages\auth\loginCognito.tsx), [src/contexts/UserContext.tsx:84-109](file:///C:\src\contexts\UserContext.tsx)

## API Key Authentication

### Direct API Credentials

The LoginAPI component provides an alternative authentication method using direct API credentials for basketball simulation services.



The API authentication stores credentials for direct service access:

• **API Key**: Primary service identifier

• **Authorization**: Service authorization token

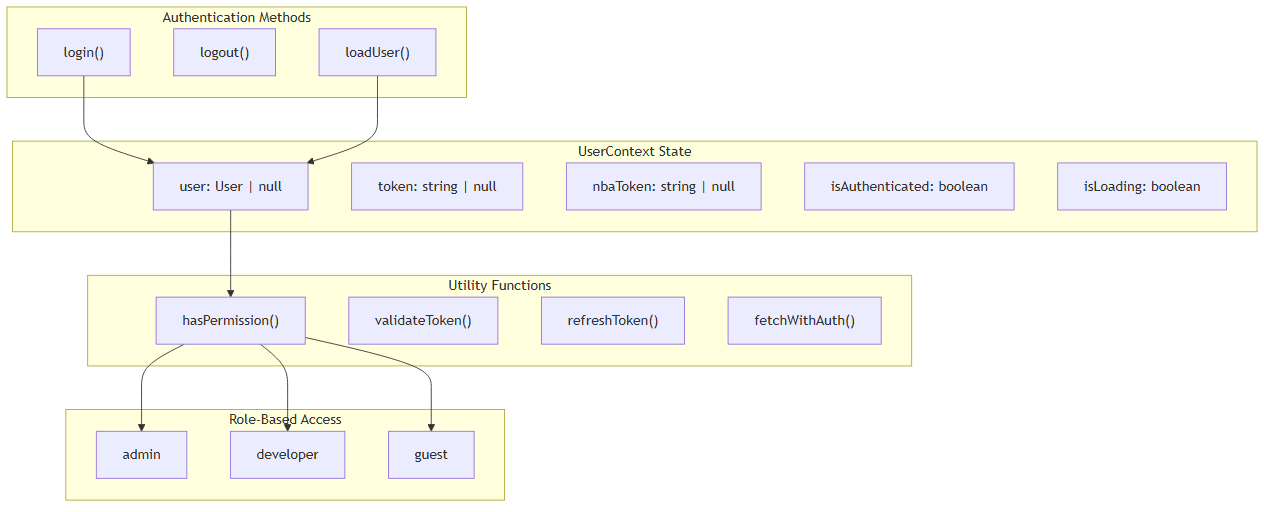
• **Direct Storage**: Credentials stored in ApiContext for immediate use

Sources: [src/LoginAPI.tsx:27-172](file:///C:\src\LoginAPI.tsx)

## User Context and State Management

### UserContext Provider

The UserContext serves as the central authentication state manager, providing a unified interface for both authentication methods.



### Permission System

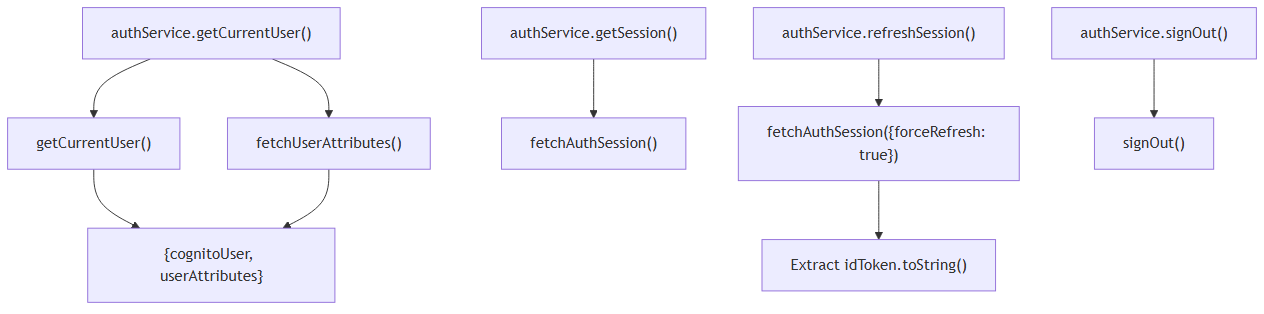
The authentication system implements role-based access control:

|  |  |  |
| --- | --- | --- |
| **Role** | **Permissions** | **Description** |
| admin | view\_all, add\_edit\_delete\_users, add\_edit\_records, delete\_records, edit\_profile | Full system access |
| developer | view\_all, add\_edit\_records, delete\_records, edit\_profile | Development access without user management |
| guest | view\_all | Read-only access |

Sources: [src/contexts/UserContext.tsx:196-206](file:///C:\src\contexts\UserContext.tsx)

### Authentication Service Layer

The AuthService provides low-level AWS Amplify integration with error handling and token management:



#### Service Methods Implementation

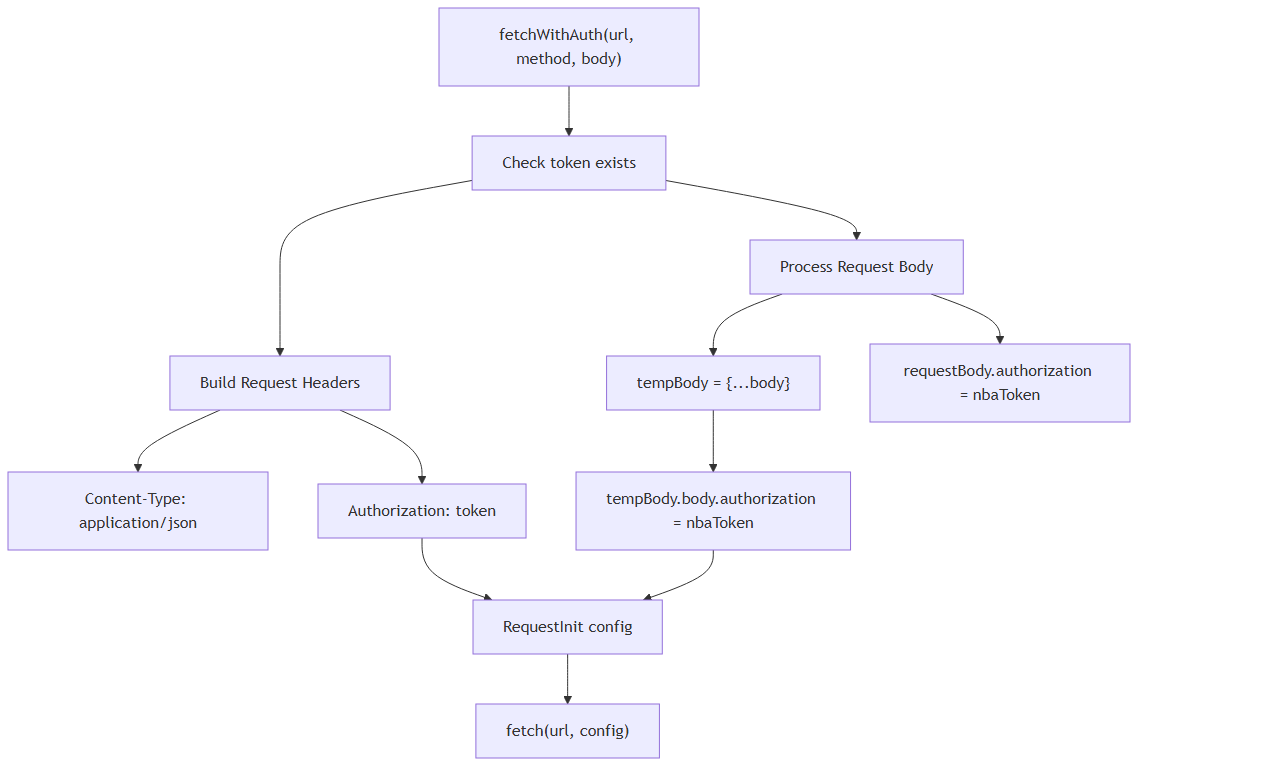
|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Implementation** | **Returns** | **Error Handling** |
| getCurrentUser() | await getCurrentUser() + await fetchUserAttributes() | {cognitoUser, userAttributes} \| null | Logs error, returns null |
| getSession() | await fetchAuthSession() | AuthSession \| null | Logs error, returns null |
| signOut() | await signOut() | boolean | Returns false on error |
| refreshSession() | await fetchAuthSession({forceRefresh: true}) | string \| null | Returns null on error |

Sources: [src/contexts/AuthService.ts:4-61](file:///C:\src\contexts\AuthService.ts)

## Dual Authentication Integration

### Unified Request Handling

The fetchWithAuth function integrates Cognito authentication with NBA API credentials through dual token injection:



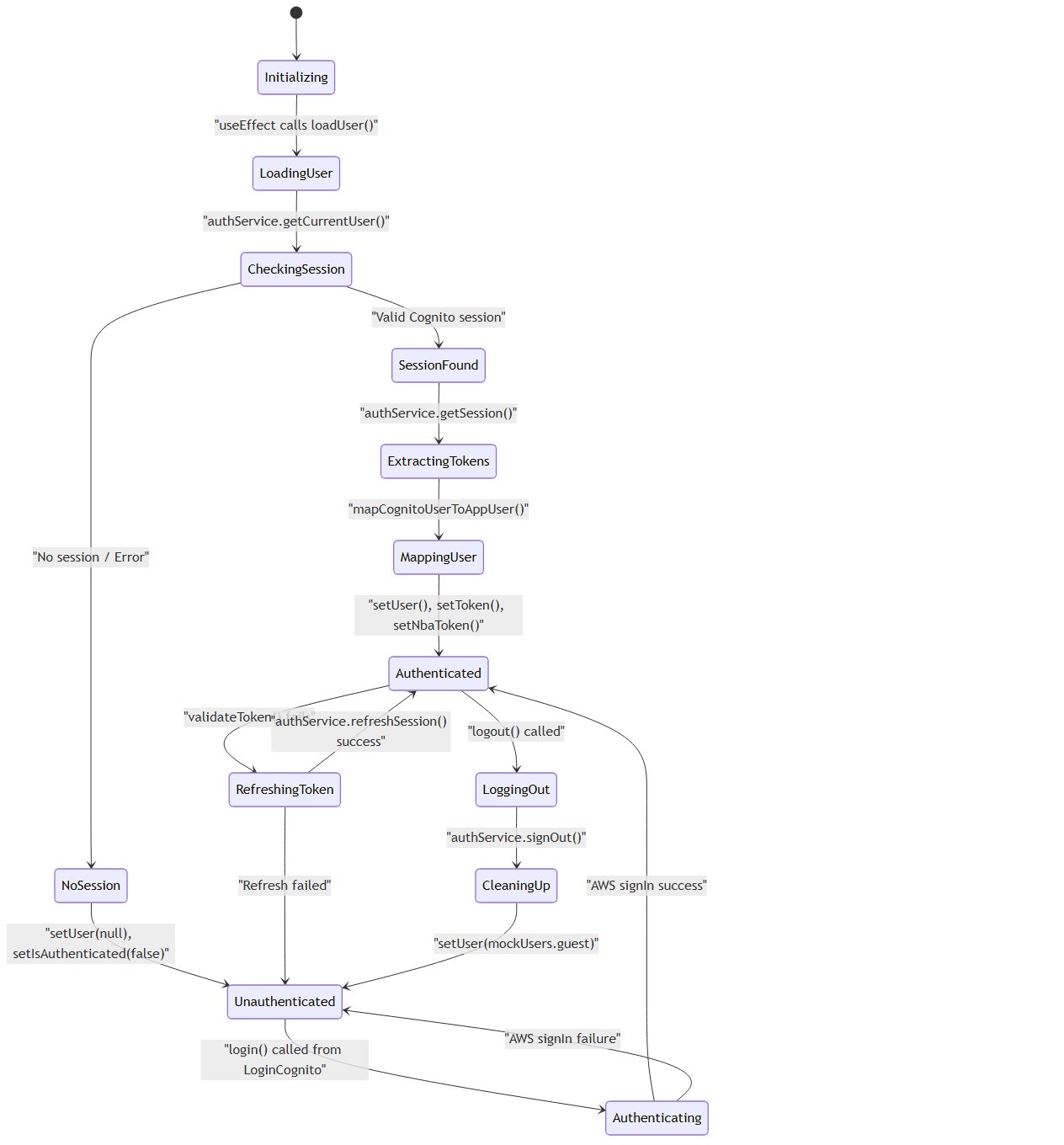
#### Implementation Details

|  |  |  |
| --- | --- | --- |
| **Step** | **Code Implementation** | **Purpose** |
| Token Validation | if (!token) throw new Error(...) | Ensures user is authenticated |
| Header Setup | Authorization: token \|\| "" | Adds Cognito JWT to headers |
| Body Processing | tempBody["body"]["authorization"] = nbaToken | Injects NBA token into request body |
| Method Handling | if (method !== 'GET' && method !== 'DELETE') | Only add body for data methods |
| Request Execution | return fetch(url, config) | Returns native Response object |

**Dual Token Strategy:**  
- **Cognito Token**: Used in Authorization header for user identity verification  
- **NBA Token**: Injected into request body as authorization field for service access  
- **Source**: NBA token extracted from Cognito idToken.payload["custom:string"]

Sources: [src/contexts/UserContext.tsx:230-257](file:///C:\src\contexts\UserContext.tsx)

### Authentication State Lifecycle



#### Key Lifecycle Functions

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Trigger** | **State Changes** | **Code Location** |
| loadUser() | useEffect on mount | Loading → Authenticated/Unauthenticated | src/contexts/UserContext.tsx:124-163 |
| login() | LoginCognito success | Unauthenticated → Authenticated | src/contexts/UserContext.tsx:166-175 |
| logout() | User action | Authenticated → Unauthenticated | src/contexts/UserContext.tsx:177-188 |
| validateToken() | Token expiration check | Validation without state change | src/contexts/UserContext.tsx:217-228 |

Sources: [src/contexts/UserContext.tsx:124-188](file:///C:\src\contexts\UserContext.tsx), [/src/contexts/UserContext.tsx:217-228](file:///C:\src\contexts\UserContext.tsx)

## Key Components Summary

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Purpose** | **Key Methods** | **State Management** |
| UserContext | Central authentication state and token management | login(), logout(), fetchWithAuth(), loadUser(), hasPermission() | User, tokens, authentication status |
| LoginCognito | AWS Cognito sign-in interface | handleSubmit(), handleAuthNextStep() | Form state, loading, error handling |
| Signup | User registration with email confirmation | handleSubmit(), handleConfirmSignUp(), handleResendCode() | Form data, confirmation flow |
| AuthService | AWS Amplify service wrapper | getCurrentUser(), getSession(), signOut(), refreshSession() | Service-level error handling |
| mapCognitoUserToAppUser | Token extraction and user mapping | N/A (pure function) | Transforms Cognito data to app User object |

The authentication system provides a robust foundation for securing the NBA simulation application while maintaining flexibility for both user account management and service integration.

Sources: [src/contexts/UserContext.tsx:1-298](file:///C:\src\contexts\UserContext.tsx), [src/pages/auth/loginCognito.tsx:1-314](file:///C:\src\pages\auth\loginCognito.tsx), [src/LoginAPI.tsx:1-172](file:///C:\src\LoginAPI.tsx), [src/pages/auth/signup.tsx:1-321](file:///C:\src\pages\auth\signup.tsx), [src/contexts/AuthService.ts:1-62](file:///C:\src\contexts\AuthService.ts)

# Build System

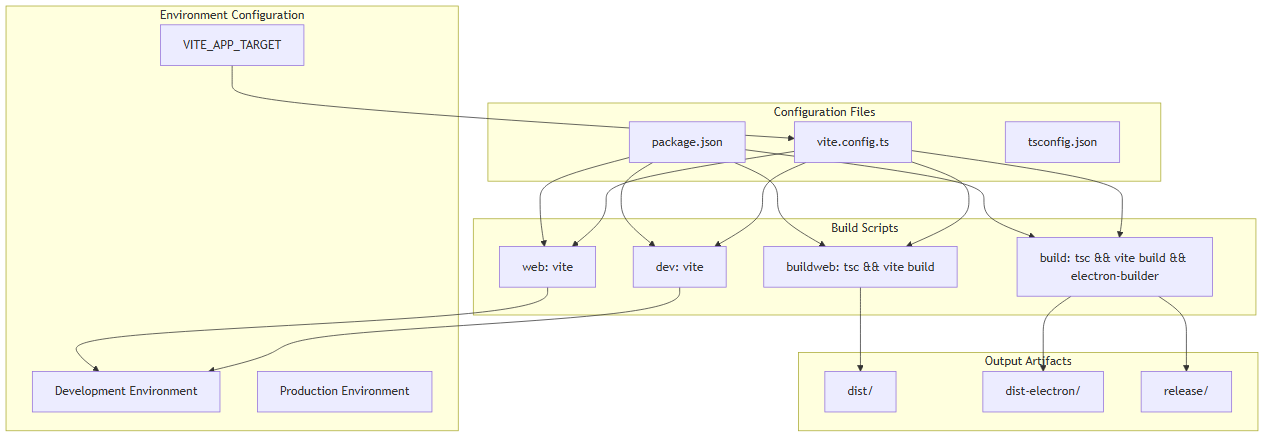
This document covers the Vite-based build system that enables dual-target deployment to both web browsers and Electron desktop applications from a single codebase. The build system uses environment variables and conditional configuration to support multiple development and production workflows.

For information about the overall application architecture, see [Architecture](file:///C:\Repositorios\md2docx\output\3_Architecture.md). For development workflows and setup instructions, see [Getting Started](file:///C:\Repositorios\md2docx\output\2_Getting_Started.md).

## Overview

The build system is built on Vite and supports four primary build targets through npm scripts defined in package.json. The system uses the VITE\_APP\_TARGET environment variable to conditionally enable Electron-specific plugins and configurations.

### Build Target Architecture

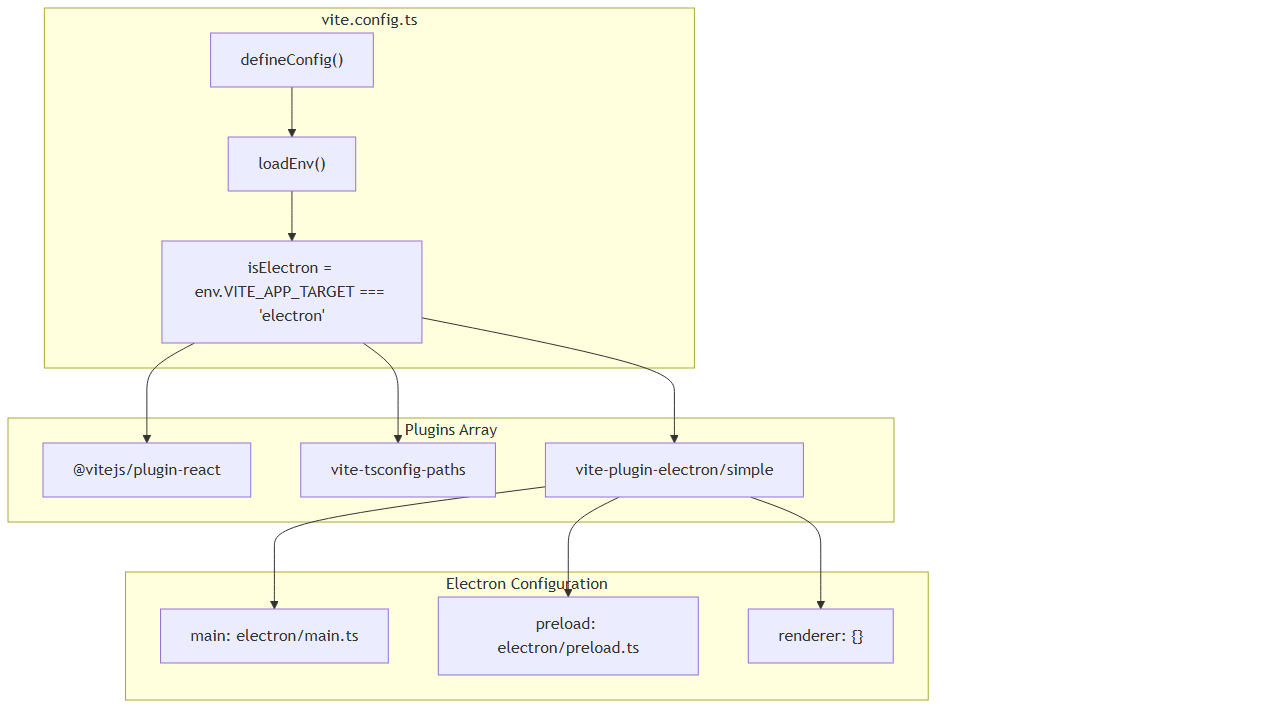


Sources: [package.json:13-21](file:///C:\package.json), [vite.config.ts:8-37](file:///C:\vite.config.ts), [README.md:38-86](file:///C:\README.md)

## Build Configuration

The core build configuration is defined in vite.config.ts, which uses conditional logic to enable Electron-specific features based on the VITE\_APP\_TARGET environment variable.

### Vite Configuration Structure



Sources: [vite.config.ts:1-37](file:///C:\vite.config.ts)

The configuration file loads environment variables and conditionally includes the Electron plugin only when VITE\_APP\_TARGET=electron. The isElectron boolean controls plugin inclusion through array filtering.

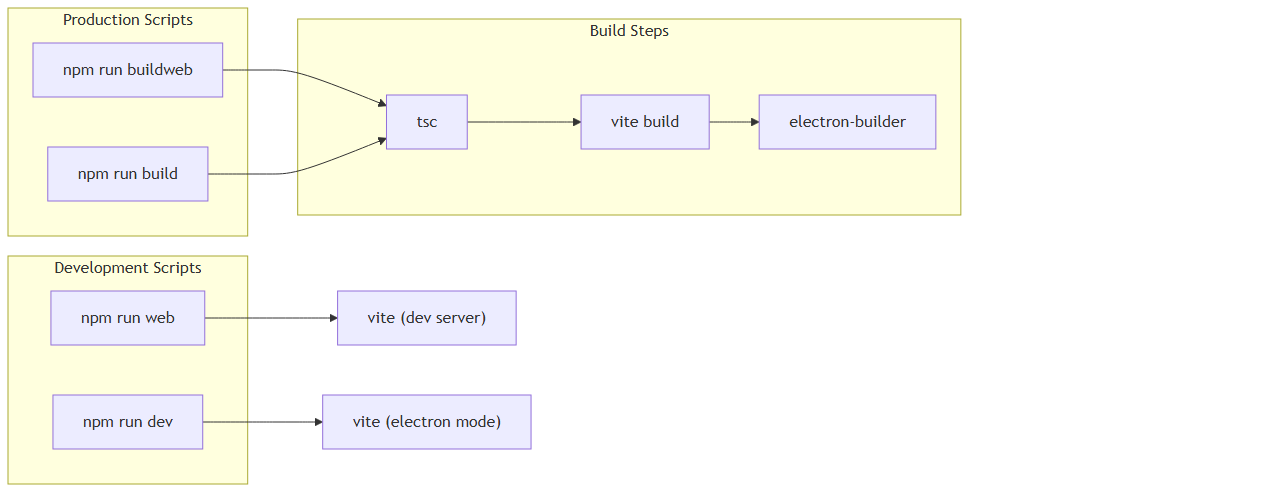
## Build Scripts

The package.json defines four primary build scripts that handle different development and production scenarios:

|  |  |  |  |
| --- | --- | --- | --- |
| **Script** | **Command** | **Purpose** | **Output** |
| web | vite | Web development server | localhost:5173 |
| dev | vite | Electron development mode | Electron app window |
| buildweb | tsc && vite build | Web production build | dist/ directory |
| build | tsc && vite build && electron-builder | Electron production build | dist-electron/ and release/ |

Sources: [package.json:13-21](file:///C:\package.json)

### Build Script Flow



Sources: [package.json:13-21](file:///C:\package.json)

## Development Workflow

The development workflow varies based on the target platform:

### Web Development Mode

• Runs vite development server on port 5173

• Hot module replacement (HMR) enabled

• Uses standard browser environment

• Requires VITE\_APP\_TARGET to be unset or not equal to "electron"

### Electron Development Mode

• Runs Vite with Electron plugin enabled

• Spawns Electron application window

• Requires VITE\_APP\_TARGET=electron environment variable

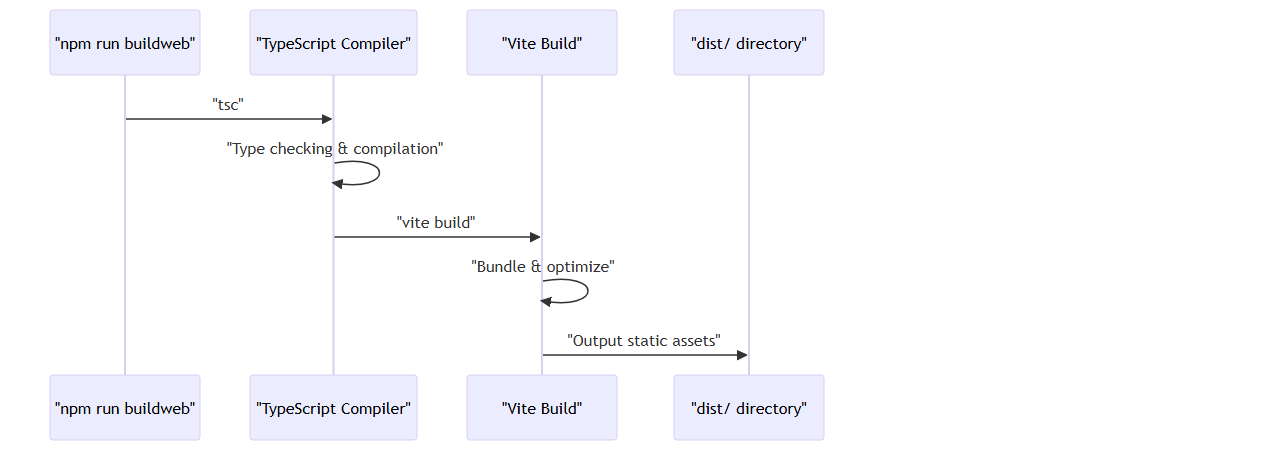
• Supports main process and renderer process development

Sources: [README.md:40-60](file:///C:\README.md), [vite.config.ts:10-16](file:///C:\vite.config.ts)

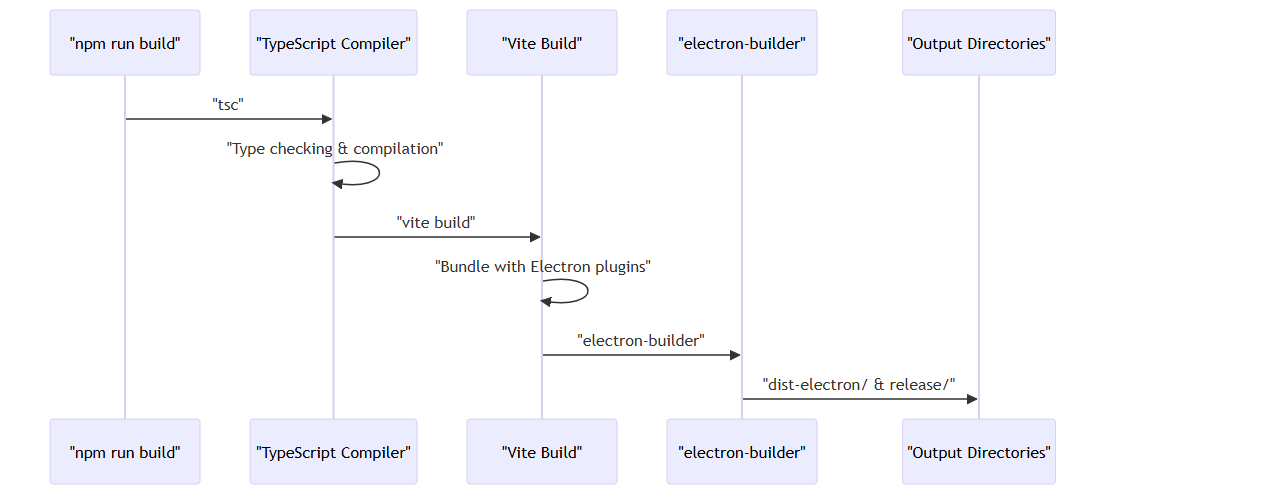
## Production Builds

Production builds involve TypeScript compilation followed by Vite bundling, with optional Electron packaging.

### Web Production Build Process



### Electron Production Build Process

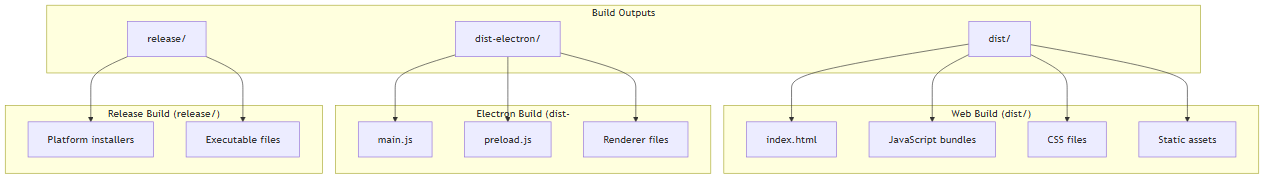


Sources: [package.json:16-17](file:///C:\package.json), [README.md:62-86](file:///C:\README.md)

## Output Artifacts

The build system generates different artifacts based on the build target:

### File Structure Output



Sources: [package.json:158](file:///C:\package.json), [README.md:74-86](file:///C:\README.md)

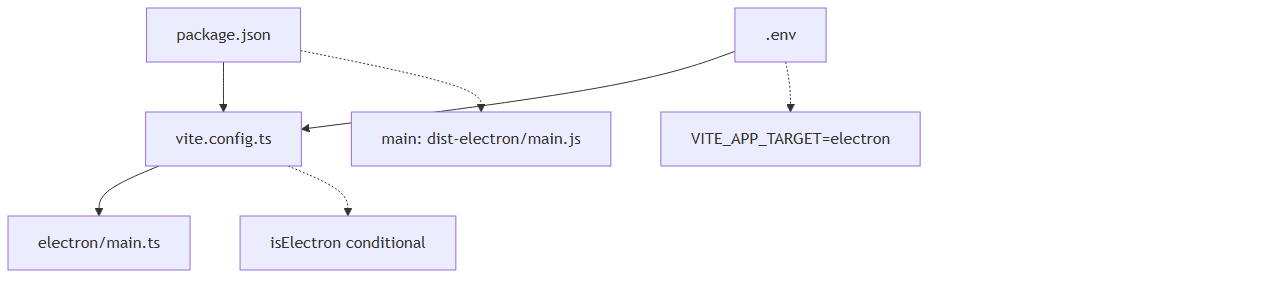
## Configuration Files

Several configuration files support the build system:

### Supporting Configuration Files

|  |  |  |
| --- | --- | --- |
| **File** | **Purpose** | **Key Settings** |
| vite.config.ts | Main build configuration | Plugin loading, entry points, base path |
| package.json | Dependencies and scripts | Build scripts, main entry, electron-builder config |
| electron/main.ts | Electron main process | Window creation, app lifecycle |
| .env | Environment variables | VITE\_APP\_TARGET, API URLs |

### Configuration Dependencies



Sources: [vite.config.ts:9-10](file:///C:\vite.config.ts), [package.json:158](file:///C:\package.json), [electron/main.ts:1](file:///C:\electron\main.ts), [.env.example:1](file:///C:\.env.example)

The build system provides a flexible, environment-driven approach to supporting both web and desktop deployment targets through conditional configuration and unified tooling.

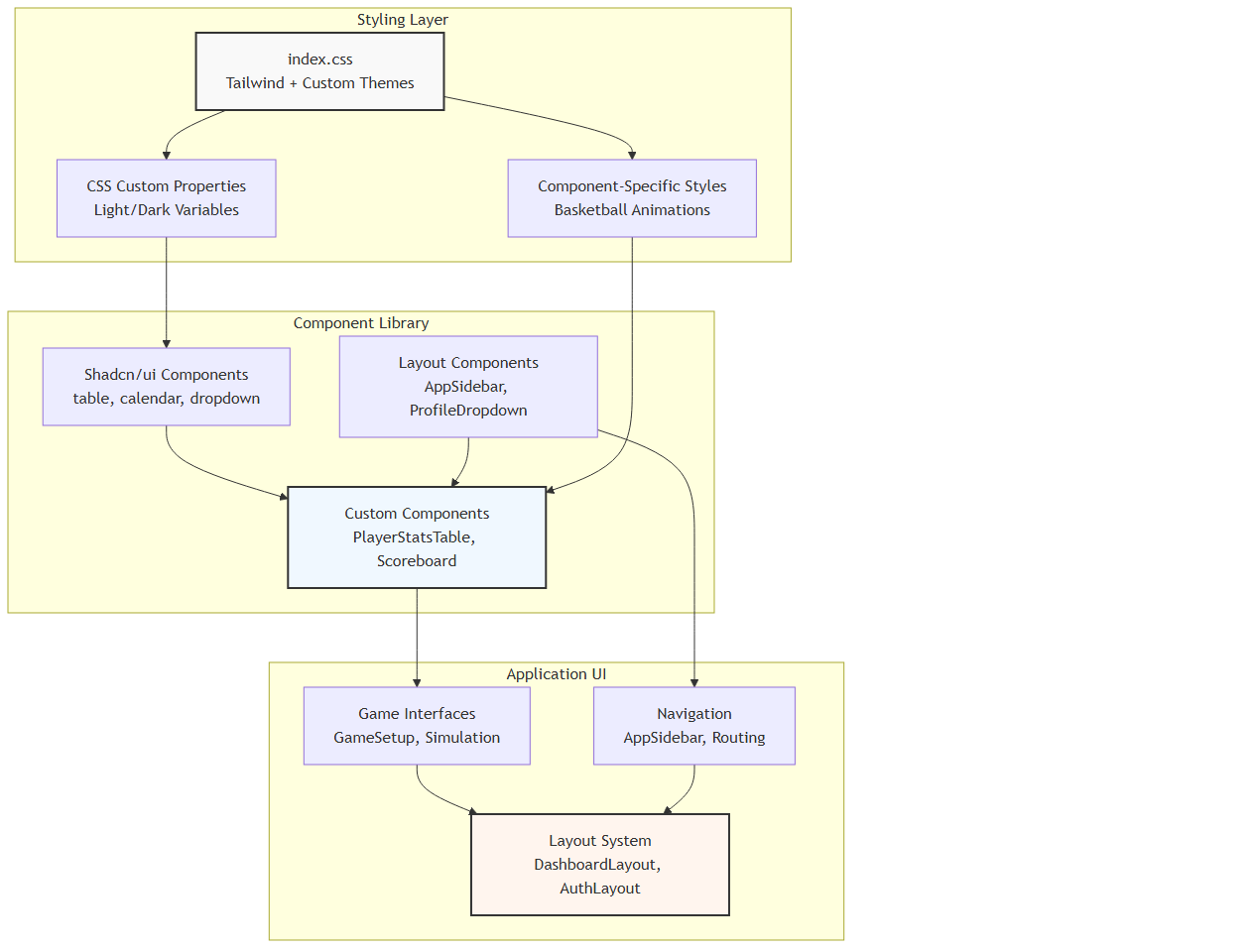
# User Interface

This document covers the user interface architecture, components, and styling systems of the NBA basketball simulation application. It includes the core UI component library, theming system, layout structures, and interactive game elements.

For authentication-specific UI components, see [Authentication UI](file:///C:\Repositorios\md2docx\output\13_Authentication_UI.md). For game simulation interface details, see [Game Features](file:///C:\Repositorios\md2docx\output\14_Game_Features.md). For build system and styling configurations, see [Build System](file:///C:\Repositorios\md2docx\output\7_Build_System.md) and [Styling and Theming](file:///C:\Repositorios\md2docx\output\19_Styling_and_Theming.md).

## UI Architecture Overview

The application employs a component-based UI architecture built on React with TypeScript, utilizing Tailwind CSS for styling and a custom theme system supporting light/dark modes. The UI system integrates multiple component libraries including Shadcn/ui and Radix UI primitives.

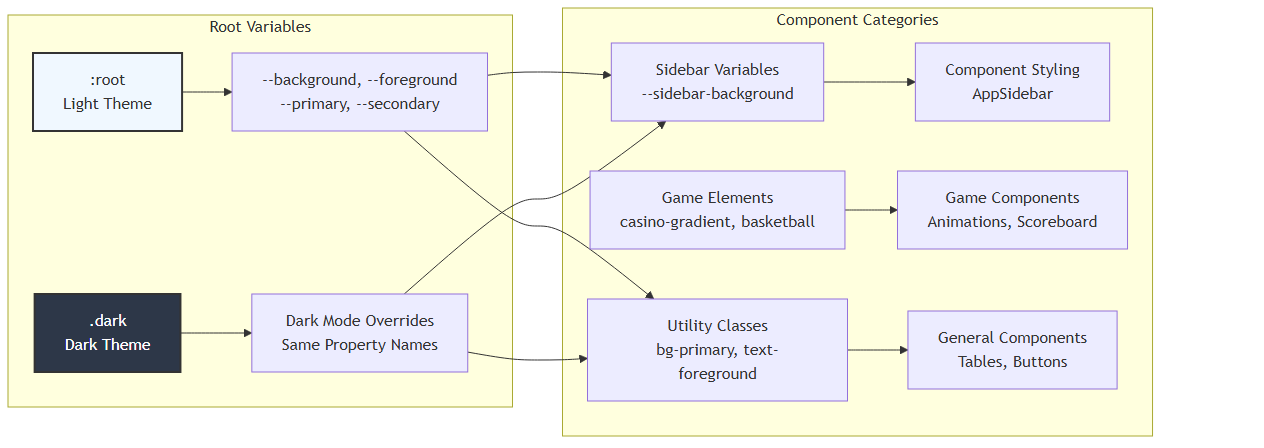


**Sources:** [src/index.css:1-439](file:///C:\src\index.css), [src/styles/index.css:1-203](file:///C:\src\styles\index.css), [src/components/ui/calendar.tsx:1-64](file:///C:\src\components\ui\calendar.tsx)

## Theme and Styling System

The application implements a comprehensive theming system using CSS custom properties with automatic light/dark mode switching. The styling architecture combines Tailwind utility classes with custom CSS for specialized components.

### CSS Custom Properties Architecture



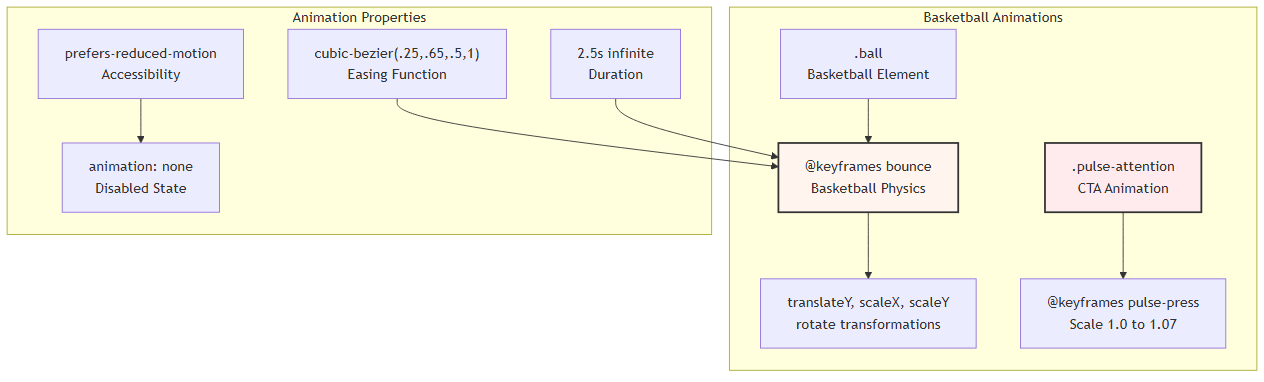
The theme system defines comprehensive color schemes in src/index.css:6-83 with CSS custom properties for:

|  |  |  |  |
| --- | --- | --- | --- |
| **Property Category** | **Light Theme** | **Dark Theme** | **Usage** |
| Background | --background: 0 0% 100% | --background: 222.2 84% 4.9% | Main app background |
| Primary | --primary: 210 72% 22% | --primary: 210 40% 98% | Primary UI elements |
| Sidebar | --sidebar-background: 210 72% 22% | --sidebar-background: 240 5.9% 10% | Navigation sidebar |
| Destructive | --destructive: 0 84.2% 60.2% | --destructive: 0 62.8% 30.6% | Error states |

**Sources:** [src/index.css:5-84](file:///C:\src\index.css), [src/styles/index.css:8-11](file:///C:\src\styles\index.css)

### Custom Animation System

The application includes specialized animations for basketball-themed elements:



Key animation implementations in [src/index.css:367-439](file:///C:\src\index.css):  
- Basketball bounce animation with physics-based scaling and rotation  
- Pulse attention animation for call-to-action buttons  
- Accessibility support with prefers-reduced-motion media query

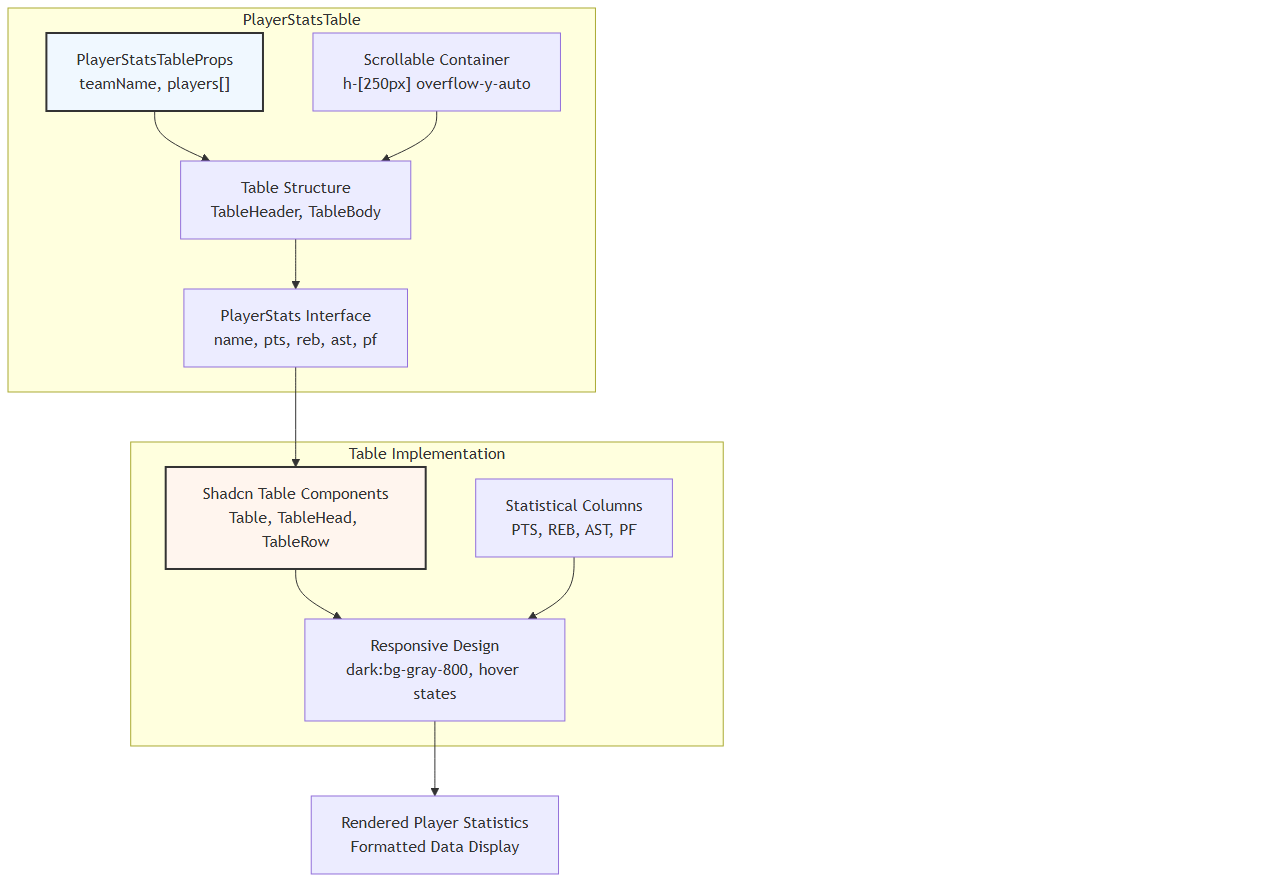
**Sources:** [src/index.css:367-439](file:///C:\src\index.css), [src/index.css:216-282](file:///C:\src\index.css)

## Core UI Components

### Data Display Components

The application provides specialized components for displaying basketball statistics and game information.

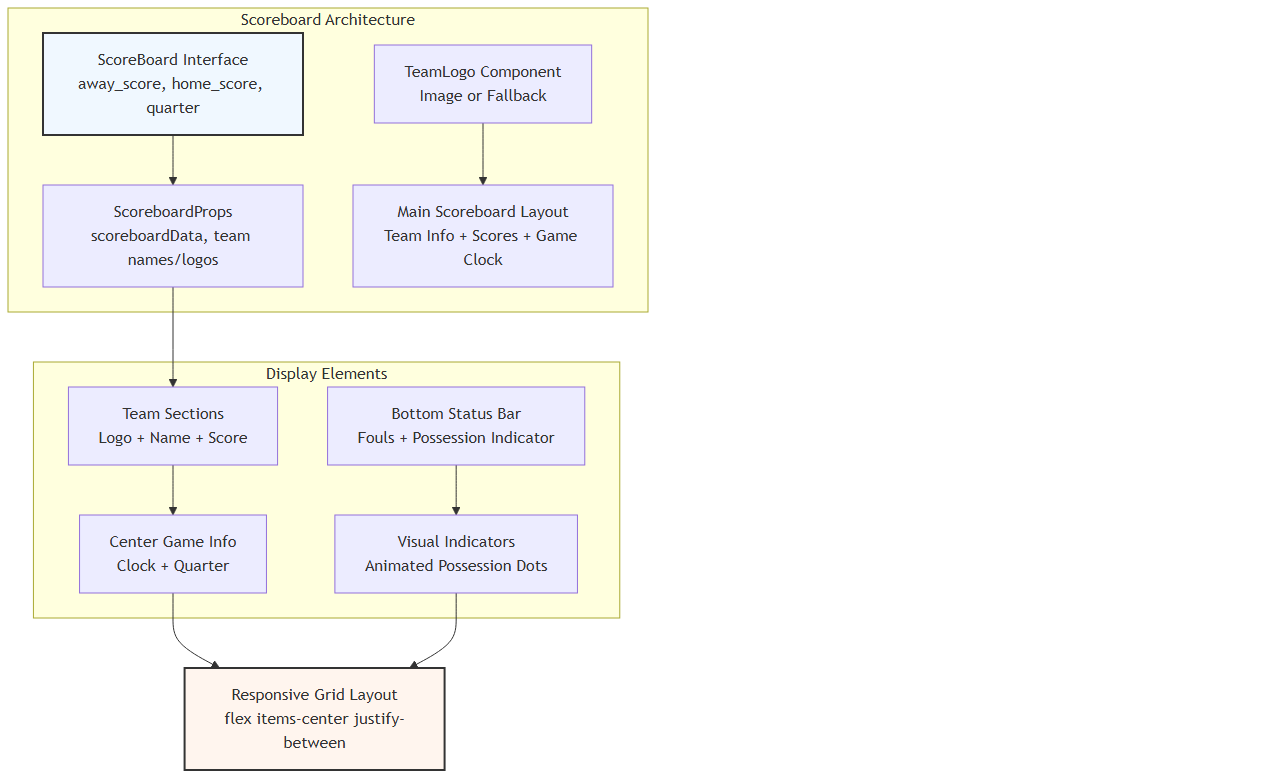
#### PlayerStatsTable Component



The PlayerStatsTable component in [src/components/PlayerStatsTable.tsx:24-56](file:///C:\src\components\PlayerStatsTable.tsx) implements:  
- Fixed-height scrollable container for large datasets  
- Responsive design with dark theme support  
- Structured data display for basketball statistics  
- TypeScript interfaces for type safety

**Sources:** [src/components/PlayerStatsTable.tsx:1-57](file:///C:\src\components\PlayerStatsTable.tsx)

#### Scoreboard Component

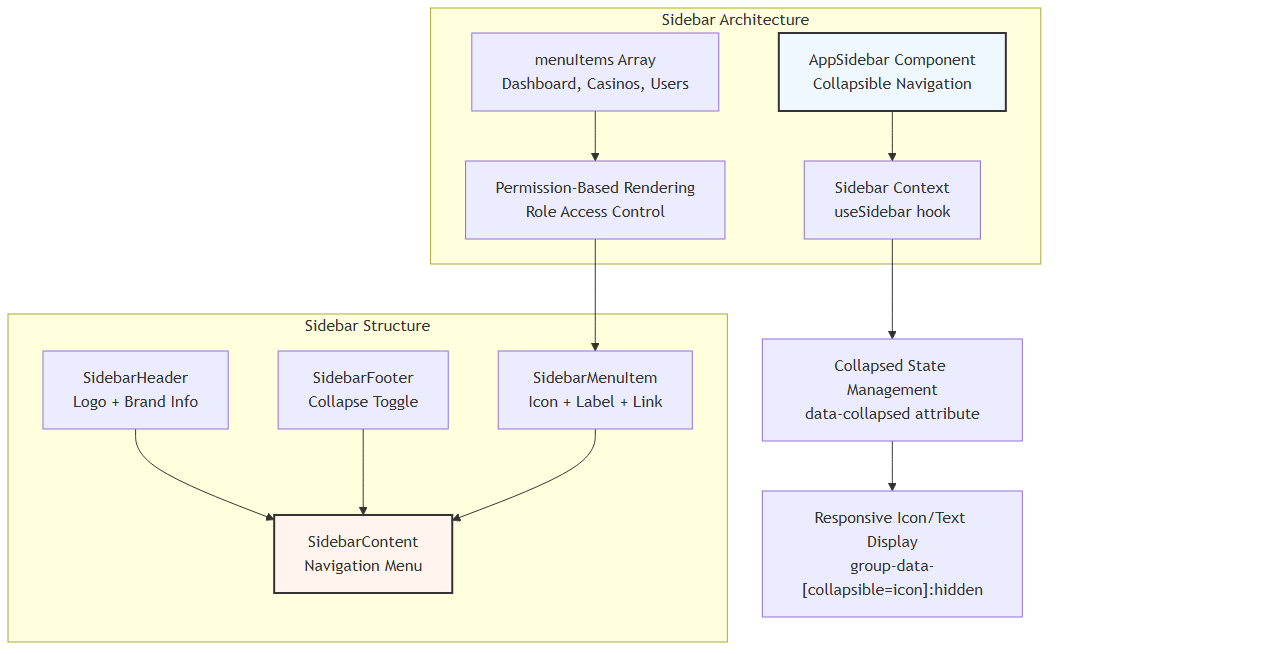


The Scoreboard component in [src/components/Scoreboard.tsx:24-99](file:///C:\src\components\Scoreboard.tsx) features:  
- Real-time game state display with animated possession indicators  
- Conditional logo rendering with fallback team initials  
- Responsive layout with basketball-themed styling  
- Team offense/defense state visualization

**Sources:** [src/components/Scoreboard.tsx:1-101](file:///C:\src\components\Scoreboard.tsx)

### Navigation Components

#### AppSidebar Implementation



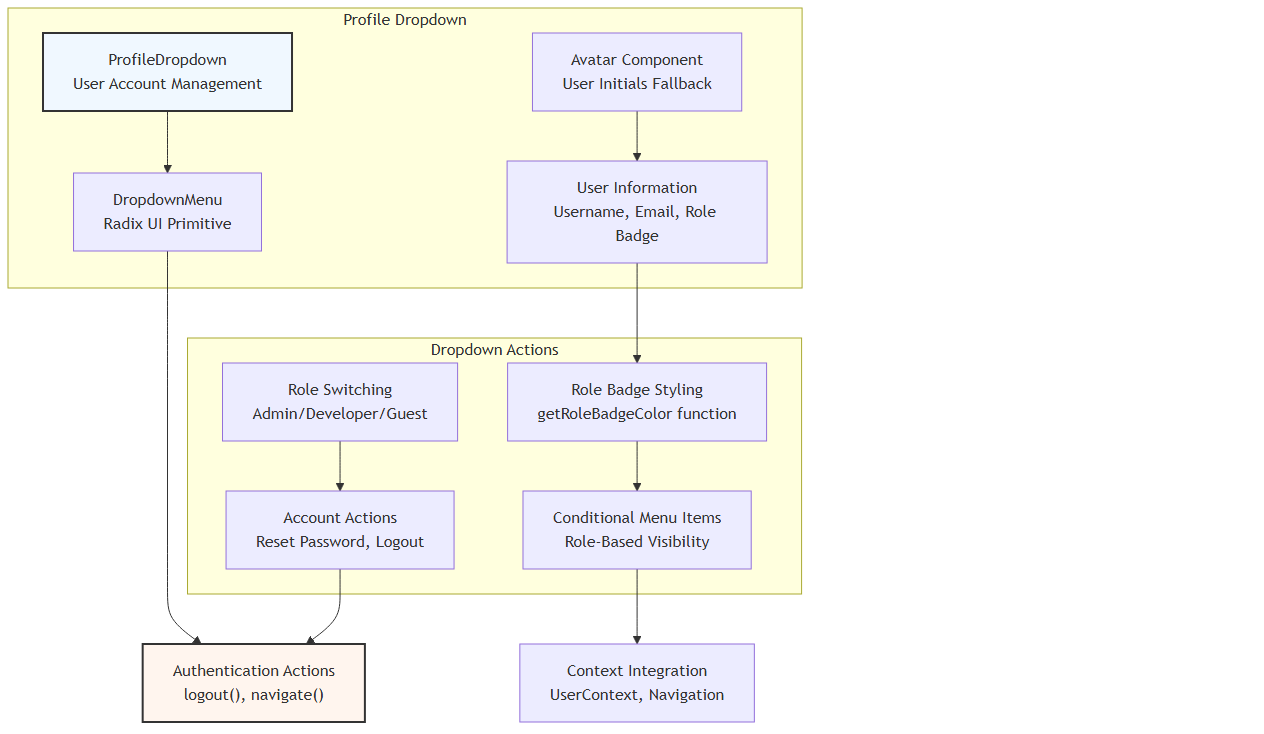
The sidebar system in [src/components/AppSidebar.tsx:35-116](file:///C:\src\components\AppSidebar.tsx) implements:  
- Collapsible navigation with icon-only and expanded states  
- Role-based menu item visibility using UserContext  
- Responsive design with conditional content display  
- Integration with React Router for navigation state

Navigation menu structure defined in [src/components/AppSidebar.tsx:27-33](file:///C:\src\components\AppSidebar.tsx):

|  |  |  |  |
| --- | --- | --- | --- |
| **Menu Item** | **Route** | **Icon** | **Required Permission** |
| Dashboard | /adminpanel | Home | view\_all |
| Casinos | /adminpanel/casinos | Building2 | view\_all |
| Users | /adminpanel/users | Users | add\_edit\_delete\_users |
| Analytics | /adminpanel/analytics | BarChart3 | view\_all |
| Settings | /adminpanel/settings | Settings | view\_all |

**Sources:** [src/components/AppSidebar.tsx:1-117](file:///C:\src\components\AppSidebar.tsx)

#### ProfileDropdown Component



The profile dropdown in [src/components/ProfileDropdown.tsx:17-120](file:///C:\src\components\ProfileDropdown.tsx) provides:  
- User role visualization with color-coded badges  
- Administrative role switching functionality  
- Account management actions (password reset, logout)  
- Integration with authentication context and routing

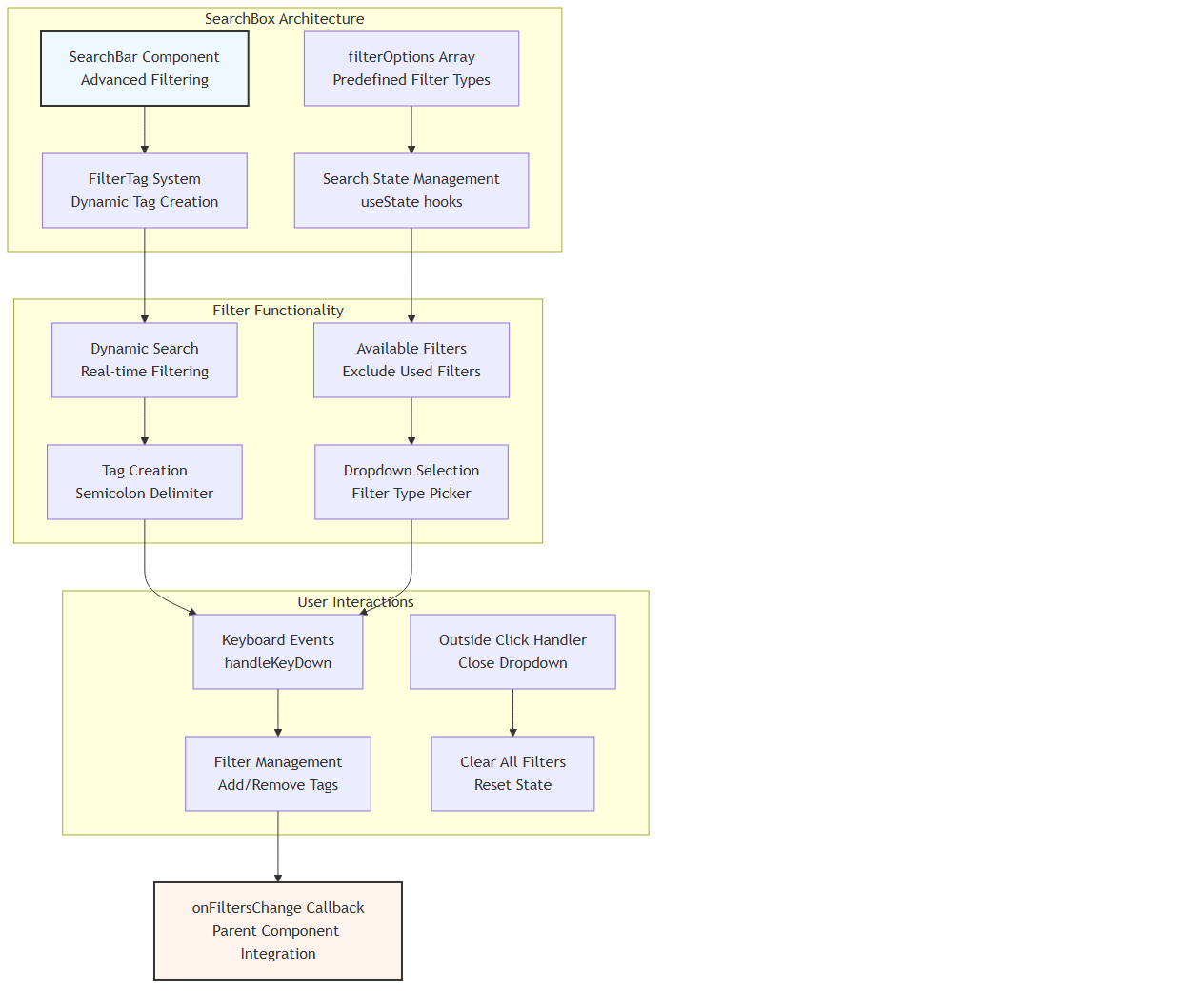
Role badge color mapping in [src/components/ProfileDropdown.tsx:35-42](file:///C:\src\components\ProfileDropdown.tsx):

getRoleBadgeColor = (role: UserRole) => {  
 admin: 'bg-red-100 text-red-800'  
 developer: 'bg-blue-100 text-blue-800'   
 guest: 'bg-gray-100 text-gray-800'  
}

**Sources:** [src/components/ProfileDropdown.tsx:1-121](file:///C:\src\components\ProfileDropdown.tsx)

## Advanced UI Components

### SearchBox Component



The advanced search component in [src/components/searchbox.tsx:27-218](file:///C:\src\components\searchbox.tsx) implements:  
- Dynamic filter tag system with semicolon-triggered creation  
- Real-time search with immediate parent callback updates  
- Available filter management to prevent duplicate filters  
- Comprehensive keyboard and mouse interaction handling

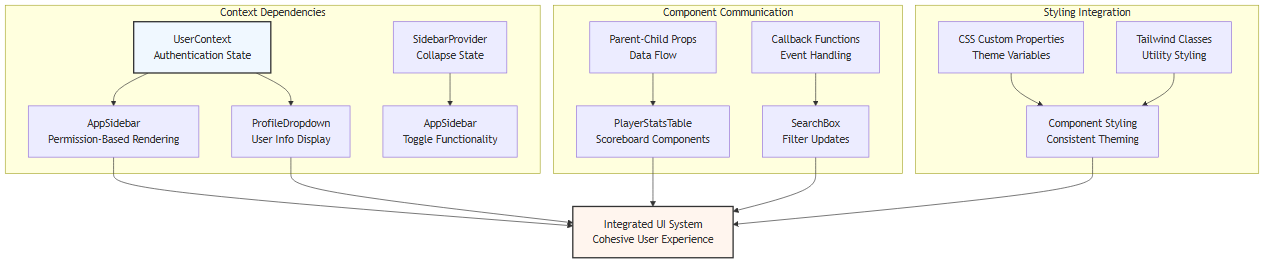
Filter configuration in [src/components/searchbox.tsx:34-41](file:///C:\src\components\searchbox.tsx):

|  |  |  |
| --- | --- | --- |
| **Filter Type** | **Available Values** | **Source** |
| casinoName | Unique casino names | data.map(item => item.casinoName) |
| location | Unique locations | data.map(item => item.location) |
| category | Unique categories | data.map(item => item.category) |
| status | ['active', 'inactive'] | Static values |

**Sources:** [src/components/searchbox.tsx:1-218](file:///C:\src\components\searchbox.tsx)

## Component Integration Patterns

### UI Context Integration



The UI system demonstrates several key integration patterns:

1. **Context-Driven Components**: Navigation and user interface elements integrate with authentication and application state contexts

1. **Prop-Based Data Flow**: Game-specific components receive data through typed props interfaces

1. **Callback Communication**: Advanced components like SearchBox communicate state changes to parent components

1. **Consistent Theming**: All components utilize the centralized CSS custom property system

**Sources:** [src/components/AppSidebar.tsx:24-38](file:///C:\src\components\AppSidebar.tsx), [src/components/ProfileDropdown.tsx:13-19](file:///C:\src\components\ProfileDropdown.tsx), [src/components/searchbox.tsx:22-75](file:///C:\src\components\searchbox.tsx)

## Design System Conventions

### Component Structure Standards

The application follows consistent patterns for component architecture:

// Interface-driven props  
interface ComponentProps {  
 data: TypedData[];  
 onCallback: (result: CallbackType) => void;  
}  
  
// Functional component with TypeScript  
const Component: React.FC<ComponentProps> = ({ data, onCallback }) => {  
 // Hooks for state management  
 const [localState, setLocalState] = useState();  
 const { contextData } = useContext();  
  
 // Event handlers  
 const handleInteraction = () => { /\* ... \*/ };  
  
 // Render with conditional logic  
 return (  
 <div className="responsive-container">  
 {/\* Conditional rendering \*/}  
 {/\* Event handlers \*/}  
 {/\* Data mapping \*/}  
 </div>  
 );  
};

### Styling Conventions

The application employs a hybrid styling approach:

1. **Tailwind Utilities**: Primary styling method for layout, spacing, and common properties

1. **CSS Custom Properties**: Theme-aware color and sizing values

1. **Component-Specific CSS**: Complex animations and specialized styling

1. **Conditional Classes**: Dynamic styling based on state and props

Example from [src/components/Scoreboard.tsx:52-97](file:///C:\src\components\Scoreboard.tsx):

className={`w-4 h-4 rounded-full bg-red-500 transition-opacity duration-300 ${  
 isHomeTeamOffense ? 'opacity-100 animate-pulse' : 'opacity-0'  
}`}

**Sources:** [src/components/PlayerStatsTable.tsx:24-56](file:///C:\src\components\PlayerStatsTable.tsx), [src/components/Scoreboard.tsx:24-99](file:///C:\src\components\Scoreboard.tsx), [src/components/AppSidebar.tsx:35-116](file:///C:\src\components\AppSidebar.tsx), [src/index.css:1-439](file:///C:\src\index.css)

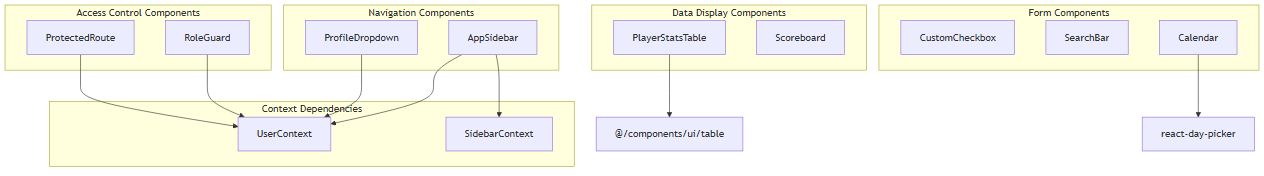
# Core Components

This document covers the reusable UI components that form the foundation of the application's user interface. These components include data display tables, navigation elements, access control utilities, and custom form controls that are used throughout the application.

For authentication-specific UI components, see [Authentication UI](file:///C:\Repositorios\md2docx\output\13_Authentication_UI.md). For game-specific interfaces, see [Game Setup Interface](file:///C:\Repositorios\md2docx\output\10_Game_Setup_Interface.md) and [Game Simulation Interfaces](file:///C:\Repositorios\md2docx\output\11_Game_Simulation_Interfaces.md).

## Component Architecture Overview

The core components follow a hierarchical structure with clear separation between data display, navigation, access control, and form components.



Sources: [src/components/PlayerStatsTable.tsx:1-56](file:///C:\src\components\PlayerStatsTable.tsx), [src/components/Scoreboard.tsx:1-101](file:///C:\src\components\Scoreboard.tsx), [src/components/AppSidebar.tsx:1-116](file:///C:\src\components\AppSidebar.tsx), [src/components/ProfileDropdown.tsx:1-120](file:///C:\src\components\ProfileDropdown.tsx), [src/components/RoleGuard.tsx:1-21](file:///C:\src\components\RoleGuard.tsx), [src/components/ProtectedRoute.tsx:1-22](file:///C:\src\components\ProtectedRoute.tsx)

## Data Display Components

### PlayerStatsTable Component

The PlayerStatsTable component renders basketball player statistics in a tabular format with scrollable content.

**Key Features:**  
- Displays player statistics (points, rebounds, assists, personal fouls)  
- Fixed height with vertical scrolling  
- Dark mode support  
- Responsive design

**Interface:**

interface PlayerStats {  
 name: string;  
 pts: string;  
 reb: string;   
 ast: string;  
 pf: string;  
}  
  
interface PlayerStatsTableProps {  
 teamName: string;  
 players: PlayerStats[];  
}

The component utilizes the shadcn/ui Table components for consistent styling and implements a 250px fixed height container with overflow scrolling [src/components/PlayerStatsTable.tsx:28-29](file:///C:\src\components\PlayerStatsTable.tsx).

### Scoreboard Component

The Scoreboard component displays real-time game information with a professional sports broadcast appearance.

**Key Features:**  
- Team scores, quarter, and game clock  
- Team logos with fallback initials  
- Possession indicators with animated pulsing  
- Foul tracking for both teams  
- Responsive layout with backdrop blur effects

**Interface:**

interface ScoreBoard {  
 away\_score: string;  
 home\_score: string;  
 quarter: string;  
 clock: string;  
 away\_possessions: string;  
 home\_possessions: string;  
 away\_fouls: string;  
 home\_fouls: string;  
 home\_team\_offense: string;  
 player\_with\_ball: string;  
}

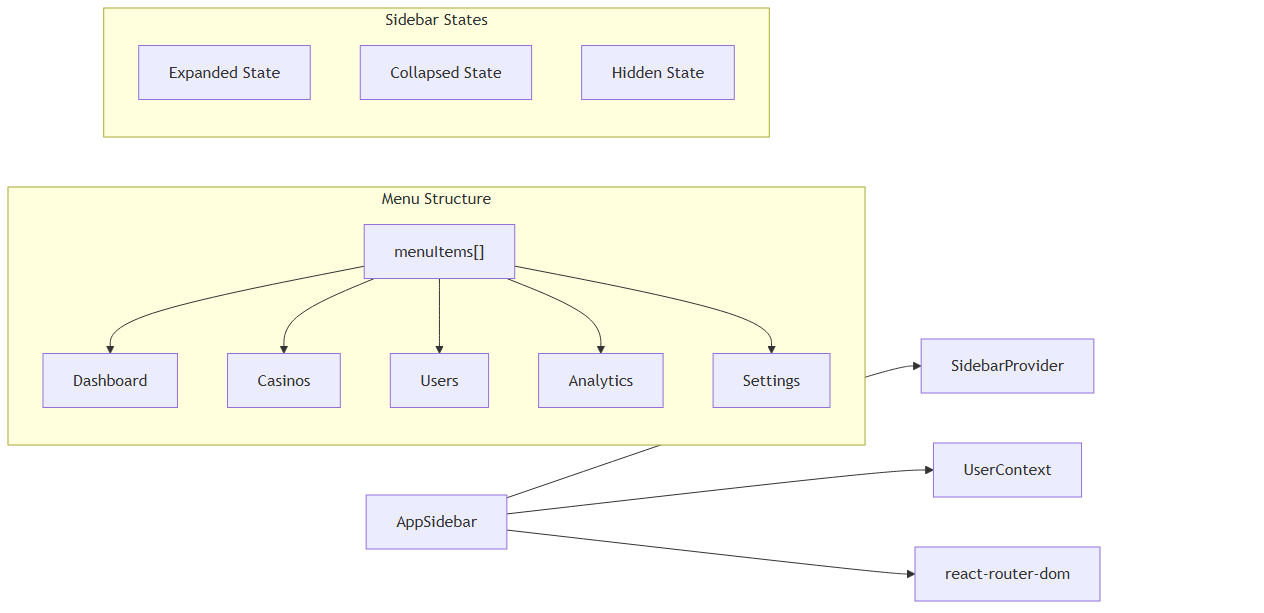
The component implements a TeamLogo subcomponent that handles both image URLs and text fallbacks [src/components/Scoreboard.tsx:47-49](file:///C:\src\components\Scoreboard.tsx).

Sources: [src/components/PlayerStatsTable.tsx:11-22](file:///C:\src\components\PlayerStatsTable.tsx), [src/components/Scoreboard.tsx:3-22](file:///C:\src\components\Scoreboard.tsx)

## Navigation Components

### AppSidebar Component

The AppSidebar component provides application navigation with role-based menu items and collapsible functionality.



**Key Features:**  
- Role-based menu visibility using permission checks  
- Collapsible/expandable with icon-only mode  
- Active route highlighting  
- Company branding with logo  
- Toggle controls with chevron icons

The sidebar uses a menuItems array with permission-based filtering [src/components/AppSidebar.tsx:27-33](file:///C:\src\components\AppSidebar.tsx) and implements a hidden state controlled by the hide boolean [src/components/AppSidebar.tsx:39](file:///C:\src\components\AppSidebar.tsx).

### ProfileDropdown Component

The ProfileDropdown provides user account management with role switching capabilities.

**Key Features:**  
- User avatar with initials fallback  
- Role badge with color coding  
- Role switching for demo purposes  
- Password reset navigation  
- Logout functionality

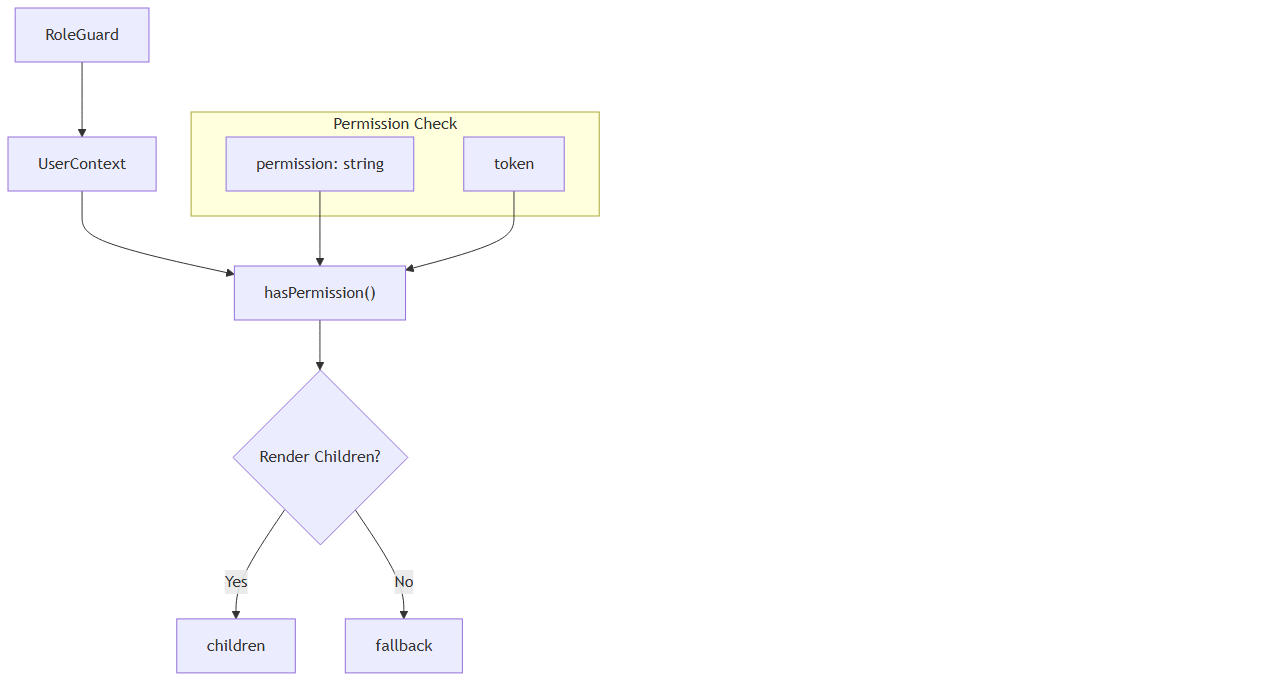
The component implements role-specific badge colors using the getRoleBadgeColor function [src/components/ProfileDropdown.tsx:35-42](file:///C:\src\components\ProfileDropdown.tsx) and handles demo role switching [src/components/ProfileDropdown.tsx:23-33](file:///C:\src\components\ProfileDropdown.tsx).

Sources: [src/components/AppSidebar.tsx:35-116](file:///C:\src\components\AppSidebar.tsx), [src/components/ProfileDropdown.tsx:17-120](file:///C:\src\components\ProfileDropdown.tsx)

## Access Control Components

### RoleGuard Component

The RoleGuard component provides declarative access control for UI elements based on user permissions.



**Interface:**

interface RoleGuardProps {  
 children: ReactNode;  
 permission: string;  
 fallback?: ReactNode;  
}

The component performs permission checking using hasPermission(permission) from the UserContext [src/components/RoleGuard.tsx:14](file:///C:\src\components\RoleGuard.tsx).

### ProtectedRoute Component

The ProtectedRoute component provides route-level access control with automatic redirection.

**Key Features:**  
- Permission-based route protection  
- Automatic redirection for unauthorized access  
- Token validation  
- Configurable redirect destination

The component uses React Router's Navigate component for redirection when access is denied [src/components/ProtectedRoute.tsx:16](file:///C:\src\components\ProtectedRoute.tsx).

Sources: src/components/RoleGuard.tsx:5-21, [src/components/ProtectedRoute.tsx:6-22](file:///C:\src\components\ProtectedRoute.tsx)

## Form and Input Components

### CustomCheckbox Component

The CustomCheckbox component provides a styled checkbox input with custom visual design.

**Features:**  
- Hidden native checkbox input  
- Custom styled checkbox with SVG checkmark  
- Label association for accessibility  
- Controlled component pattern

**Interface:**

interface CustomCheckboxProps {  
 checked: boolean;  
 onChange: (checked: boolean) => void;  
 label: string;  
 id: string;  
}

The component implements a custom visual design by hiding the native input [src/components/ui/CustomCheckbox.tsx:18](file:///C:\src\components\ui\CustomCheckbox.tsx) and using a styled div with conditional SVG rendering [src/components/ui/CustomCheckbox.tsx:21-23](file:///C:\src\components\ui\CustomCheckbox.tsx).

### SearchBar Component

The SearchBar component provides advanced filtering capabilities with tag-based filter management.

**Key Features:**  
- Dynamic filter options based on data  
- Real-time search with semicolon delimiter  
- Filter tag creation and management  
- Multiple filter types support  
- Clickout detection for dropdown closure

**Interface:**

interface SearchBarProps {  
 onFiltersChange: (filters: FilterTag[]) => void;  
 data: Casino[];  
}  
  
type FilterTag = {  
 id: string;  
 type: string;  
 label: string;  
 value: string;  
}

The component manages filter options dynamically [src/components/searchbox.tsx:34-41](file:///C:\src\components\searchbox.tsx) and implements semicolon-triggered filter creation [src/components/searchbox.tsx:83-98](file:///C:\src\components\searchbox.tsx).

### Calendar Component

The Calendar component is a wrapper around the react-day-picker library with custom styling.

**Features:**  
- Custom button variants for navigation  
- Tailwind CSS integration  
- Icon components for navigation arrows  
- Comprehensive day state styling

The component extends DayPicker props [src/components/ui/calendar.tsx:8](file:///C:\src\components\ui\calendar.tsx) and implements custom class mappings for consistent styling [src/components/ui/calendar.tsx:20-53](file:///C:\src\components\ui\calendar.tsx).

Sources: [src/components/ui/CustomCheckbox.tsx:3-30](file:///C:\src\components\ui\CustomCheckbox.tsx), [src/components/searchbox.tsx:22-218](file:///C:\src\components\searchbox.tsx), [src/components/ui/calendar.tsx:10-64](file:///C:\src\components\ui\calendar.tsx)

## Component Usage Patterns

### Context Integration

Most navigation and access control components integrate with the application's context system:

|  |  |  |
| --- | --- | --- |
| **Component** | **Context Dependencies** | **Key Methods** |
| AppSidebar | UserContext, SidebarContext | user.role, toggleSidebar() |
| ProfileDropdown | UserContext | user, logout() |
| RoleGuard | UserContext | hasPermission(), token |
| ProtectedRoute | UserContext | hasPermission(), token |

### Styling Conventions

Components follow consistent styling patterns:  
- Tailwind CSS for utility-first styling  
- Dark mode support with dark: prefixes  
- Responsive design with breakpoint modifiers  
- Custom CSS variables for theming  
- shadcn/ui component integration where applicable

### State Management

Components use various state management approaches:  
- **Local State**: useState for component-specific state  
- **Context State**: Integration with UserContext and other providers  
- **URL State**: React Router integration for navigation state  
- **Controlled Components**: Props-based state management for form inputs

Sources: [src/components/AppSidebar.tsx:24-38](file:///C:\src\components\AppSidebar.tsx), [src/components/ProfileDropdown.tsx:13-18](file:///C:\src\components\ProfileDropdown.tsx), [src/components/RoleGuard.tsx:12-14](file:///C:\src\components\RoleGuard.tsx), [src/components/ProtectedRoute.tsx:12-15](file:///C:\src\components\ProtectedRoute.tsx)

# Game Setup Interface

## Purpose and Scope

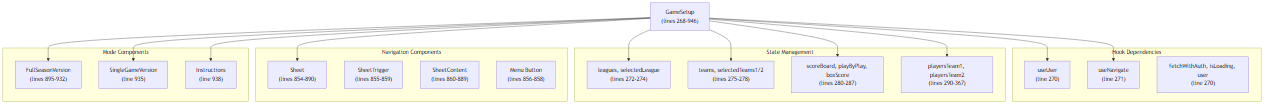
The GameSetup component serves as the central orchestration point for basketball game simulations in the NBA simulation application. Located at [src/pages/GameSetup.tsx](file:///C:\src\pages\GameSetup.tsx), this component manages league selection, team configuration, and game initialization through a comprehensive state management system that coordinates with external basketball simulation APIs.

The component implements a Sheet-based navigation interface supporting full season simulation, single game modes, and user instructions. It acts as the primary data orchestrator that bridges user selections with the underlying simulation engine through the useUser context system, providing authenticated API access and user state management.

For simulation execution details, see [Basketball Simulation](file:///C:\Repositorios\md2docx\output\15_Basketball_Simulation.md). For authentication prerequisites, see [Authentication System](file:///C:\Repositorios\md2docx\output\6_Authentication_System.md).

## Component Architecture Overview

The GameSetup component implements a Sheet-based navigation interface that orchestrates different simulation modes while maintaining shared state and API integration.

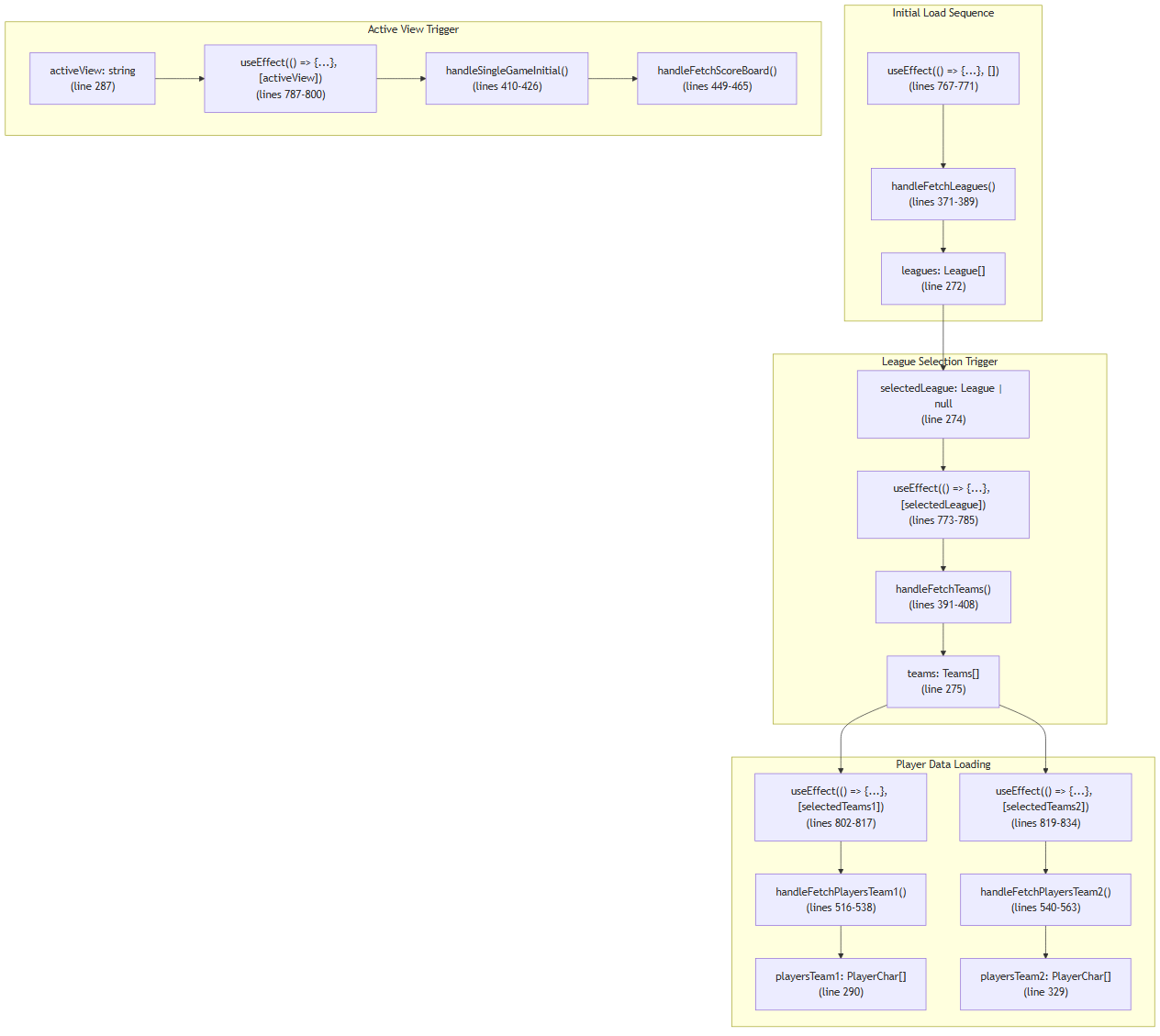


**Component Responsibilities:**  
- **FullSeasonVersion**: Complete league/team setup with 82-game schedules and live API data  
- **SingleGameVersion**: Hardcoded game simulation for testing  
- **Instructions**: User documentation and guidance  
- **Sheet Navigation**: Side panel menu for mode switching and user controls

Sources: [src/pages/GameSetup.tsx:268-946](file:///C:\src\pages\GameSetup.tsx), [src/pages/GameSetup.tsx:854-890](file:///C:\src\pages\GameSetup.tsx), [src/pages/GameSetup.tsx:895-938](file:///C:\src\pages\GameSetup.tsx)

## Data Flow and State Management

The component implements a cascading state management system where user selections trigger sequential API calls and state updates through useEffect hooks.

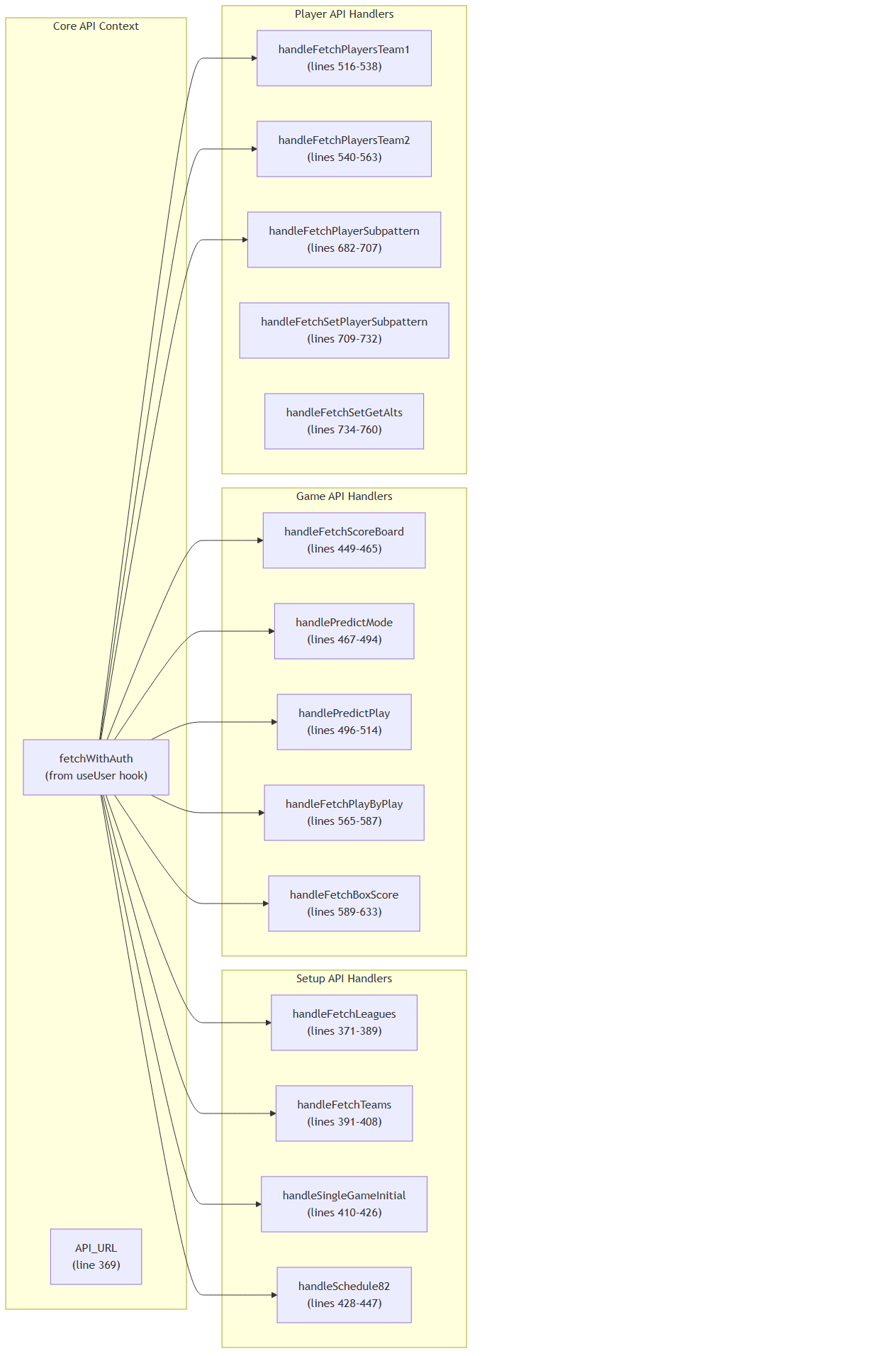


**State Initialization Pattern:**  
Each state variable follows a defensive initialization pattern with default empty or null values to prevent runtime errors during the loading sequence. The component includes detailed default player objects with all required statistical fields.

Sources: [src/pages/GameSetup.tsx:767-834](file:///C:\src\pages\GameSetup.tsx), [src/pages/GameSetup.tsx:272-367](file:///C:\src\pages\GameSetup.tsx)

## API Integration Points

The component uses the fetchWithAuth method from useApi context to communicate with basketball simulation endpoints, implementing consistent error handling and authentication across all API operations.



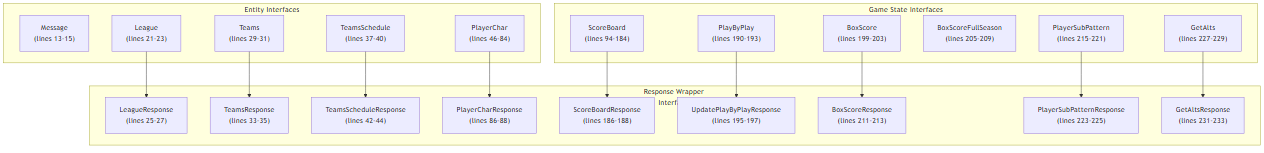
**API Endpoint Configuration**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Handler Function** | **Endpoint** | **Request Method** | **Payload** | **State Update** |
| handleFetchLeagues | get\_leagues.php | POST | None | setLeagues |
| handleFetchTeams | get\_teams.php | POST | selectedLeague | setTeams |
| handleSingleGameInitial | playsinglegame\_initial.php | POST | Team configuration | handleFetchScoreBoard |
| handleFetchScoreBoard | get\_singlegame\_stats.php | POST | None | setScoreBoard |
| handleFetchPlayersTeam1 | get\_actual\_player\_stats.php | POST | League + team1 | setPlayersTeam1 |
| handleFetchPlayersTeam2 | get\_actual\_player\_stats.php | POST | League + team2 | setPlayersTeam2 |
| handlePredictMode | play\_predict.php / play\_82 / play\_fsv | POST | Game array config | Conditional box score fetch |
| handleSchedule82 | get\_82\_game\_schedule.php | POST | League + team | setTeamsSchedule |
| handleFetchBoxScoreFullSeason | get\_raw\_box\_scores.php | POST | Game number filter | setBoxScoreFullSeason |

Sources: [src/pages/GameSetup.tsx:369-760](file:///C:\src\pages\GameSetup.tsx)

## TypeScript Interface System

The component defines comprehensive TypeScript interfaces that structure API responses and game state data, ensuring type safety across the simulation system.



**Critical Interface Details:**

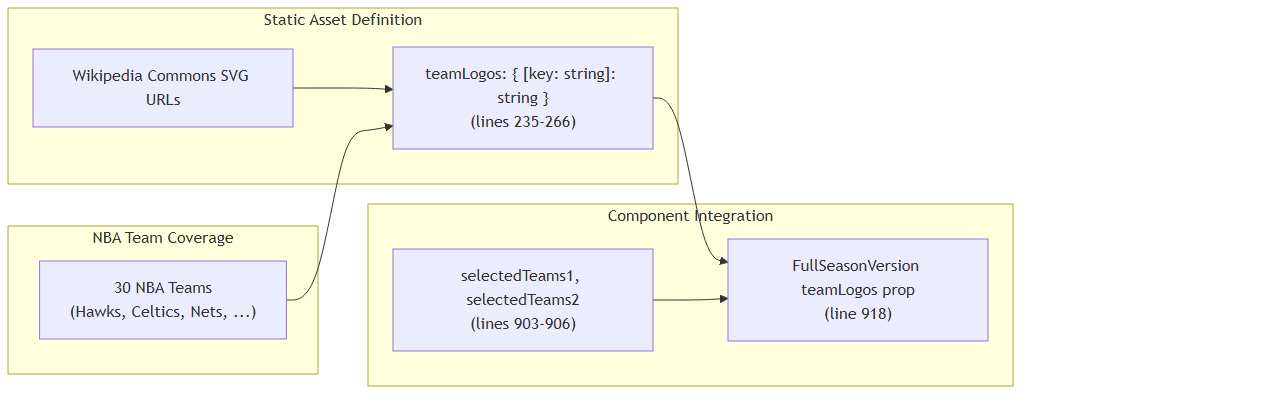
|  |  |  |
| --- | --- | --- |
| **Interface** | **Properties** | **Purpose** |
| PlayerChar | 45+ basketball statistics | Complete player statistical profile including both editable and actual stats |
| ScoreBoard | 90+ game state fields | Real-time game tracking including scores, player positions, possession data |
| TeamsSchedule | teams, games | 82-game schedule management for full season simulation |
| PlayerSubPattern | 5 position fields | Player substitution patterns for lineup management |
| BoxScoreFullSeason | text, game\_number, line\_number | Full season box score data format |
| GetAlts | alt\_sub | Alternative player substitution configurations |

**API Response Pattern:**  
All API responses follow a consistent wrapper pattern where data arrays are contained within a data property, except ScoreBoardResponse which uses scoreboard and UpdatePlayByPlayResponse which uses playbyplay.

Sources: [src/pages/GameSetup.tsx:13-233](file:///C:\src\pages\GameSetup.tsx)

## Team Logo Management System

The component includes a static asset mapping system that provides visual team identification throughout the simulation interface.



**Logo Mapping Implementation:**  
The teamLogos constant provides a complete mapping of NBA team names to their official SVG logo URLs hosted on Wikipedia Commons. This ensures consistent visual branding across the simulation interface.

**Team Coverage:**  
- 30 complete NBA team mappings  
- SVG format for scalable rendering  
- External CDN hosting via Wikipedia Commons  
- Direct team name key matching with API response data

Sources: [src/pages/GameSetup.tsx:235-266](file:///C:\src\pages\GameSetup.tsx), [src/pages/GameSetup.tsx:918](file:///C:\src\pages\GameSetup.tsx)

## Error Handling and Loading States

The component implements consistent error handling and loading state management across all API operations.

|  |  |  |
| --- | --- | --- |
| **State Variable** | **Purpose** | **Usage** |
| error | Store error messages | Set in catch blocks, displayed in UI |
| isLoading | Track API request status | From useApi() hook, disables buttons |
| isGameInitial | Track game initialization | Local loading state for game setup |

**Error Handling Pattern:**  
1. Clear existing errors with setError(null)  
2. Wrap API calls in try-catch blocks  
3. Parse error responses and set descriptive error messages  
4. Display errors in UI components

Sources: [src/pages/GameSetup.tsx:230](file:///C:\src\pages\GameSetup.tsx), [src/pages/GameSetup.tsx:235](file:///C:\src\pages\GameSetup.tsx), [src/pages/GameSetup.tsx:319-331](file:///C:\src\pages\GameSetup.tsx)

# Game Simulation Interfaces

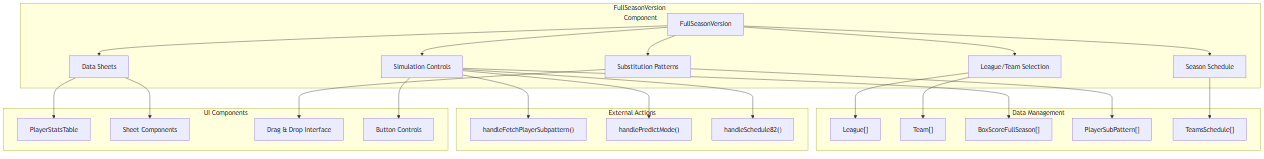
This document covers the user interface components responsible for basketball game simulation, including both full season and single game simulation modes. These interfaces provide controls for configuring simulations, displaying real-time results, and managing complex basketball-specific features like player substitution patterns.

For information about the underlying game simulation logic and mechanics, see [Basketball Simulation](#_Basketball_Simulation). For the main orchestration interface that routes users to these simulation modes, see [Game Setup Interface](#_Game_Setup_Interface).

## Full Season Simulation Interface

The FullSeasonVersion component provides a comprehensive interface for simulating entire basketball seasons. This interface handles complex season-level operations including multi-team scheduling, player rotation management, and season-long statistics tracking.

### Component Architecture



Sources: [src/pages/FullSeasonVersion.tsx:1-1413](file:///C:\src\pages\FullSeasonVersion.tsx)

### Key Interfaces and Data Structures

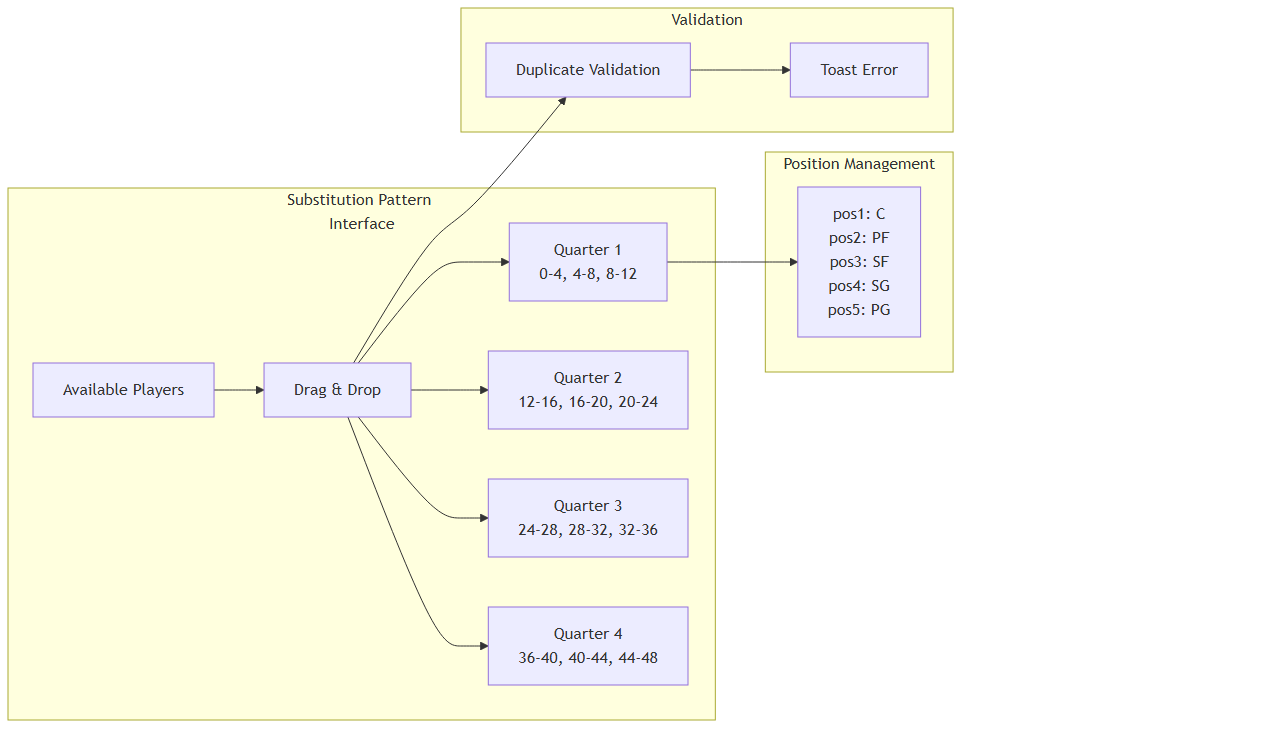
The full season interface relies on several TypeScript interfaces that define the data structures for different aspects of season simulation:

|  |  |  |
| --- | --- | --- |
| **Interface** | **Purpose** | **Key Fields** |
| League | League selection | league\_name |
| Team | Team selection | teams |
| TeamsSchedule | Season scheduling | teams, games |
| PlayerChar | Player characteristics | name, position, poss\_fact, two\_pt\_fg\_pct, etc. |
| PlayerSubPattern | 4-minute substitution intervals | pos1, pos2, pos3, pos4, pos5 |
| BoxScoreFullSeason | Season game results | text, game\_number, line\_number |

Sources: [src/pages/FullSeasonVersion.tsx:37-151](file:///C:\src\pages\FullSeasonVersion.tsx)

### Player Substitution Management

The full season interface includes a sophisticated drag-and-drop system for managing player substitutions across 4-minute intervals throughout a game:



The system prevents duplicate player assignments within the same 4-minute interval and provides visual feedback through the handleDrop function.

Sources: [src/pages/FullSeasonVersion.tsx:216-242](file:///C:\src\pages\FullSeasonVersion.tsx), [src/pages/FullSeasonVersion.tsx:245-279](file:///C:\src\pages\FullSeasonVersion.tsx)

### Sheet-Based Data Display

The interface uses sheet components to display various types of simulation data:

• **Box Score Sheet**: Displays full season box scores with attention indicators when new data is available

• **Player Statistics Sheet**: Shows detailed player stats for both teams with CSV export functionality

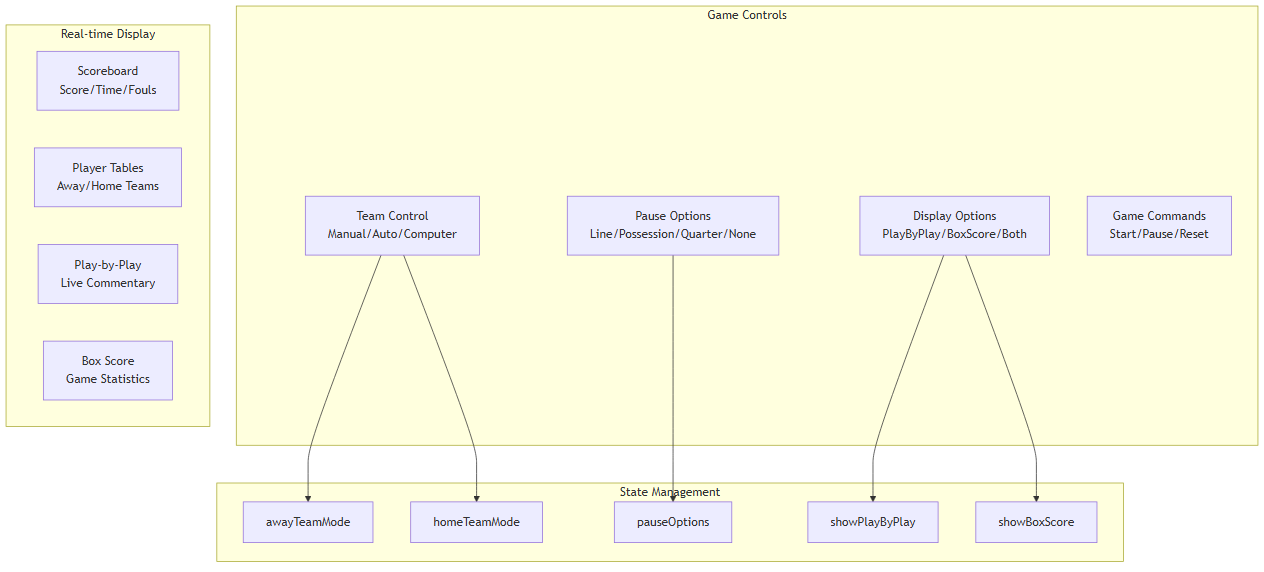
• **Substitution Pattern Sheet**: Full-screen drag-and-drop interface for managing player rotations

Sources: [src/pages/FullSeasonVersion.tsx:314-333](file:///C:\src\pages\FullSeasonVersion.tsx), [src/pages/FullSeasonVersion.tsx:569-708](file:///C:\src\pages\FullSeasonVersion.tsx)

## Single Game Simulation Interface

The SingleGameVersion component provides a real-time interface for simulating individual basketball games with granular control over game progression and display options.

### Control Systems Architecture



Sources: [src/pages/SingleGameVersion.tsx:46-56](file:///C:\src\pages\SingleGameVersion.tsx)

### Real-time Game Display

The single game interface provides multiple synchronized views of the ongoing simulation:

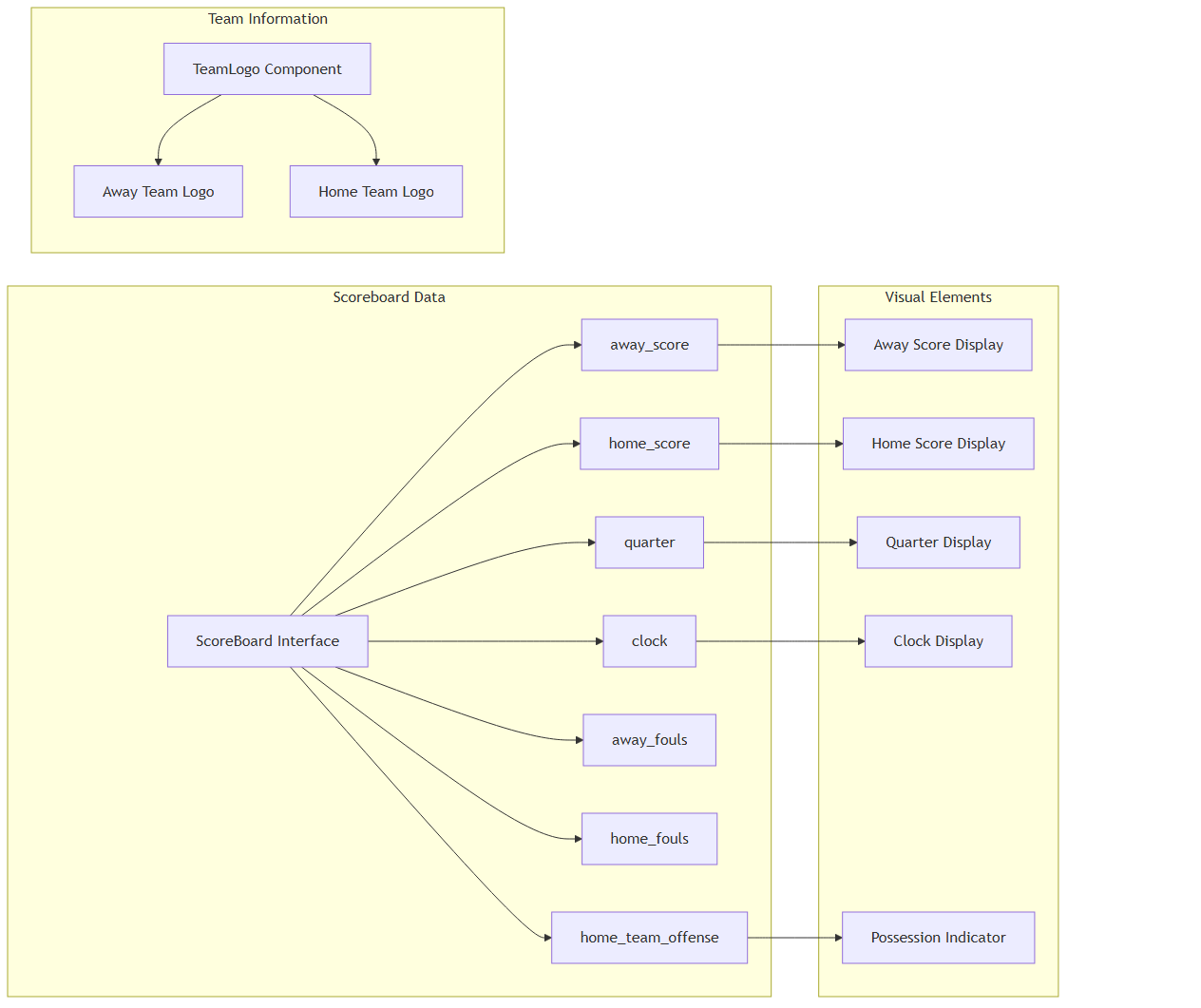
|  |  |  |
| --- | --- | --- |
| **Component** | **Purpose** | **Data Source** |
| Live Scoreboard | Current score, time, fouls | scoreboardData |
| Player Statistics Tables | Real-time player stats | awayTeamStats, homeTeamStats |
| Play-by-Play Feed | Game commentary | playByPlayData |
| Box Score Display | Detailed game statistics | boxScoreData |

Sources: [src/pages/SingleGameVersion.tsx:14-44](file:///C:\src\pages\SingleGameVersion.tsx), [src/pages/SingleGameVersion.tsx:125-175](file:///C:\src\pages\SingleGameVersion.tsx)

## Supporting Components

### Scoreboard Component

The Scoreboard component provides a professional basketball scoreboard display with real-time game information:

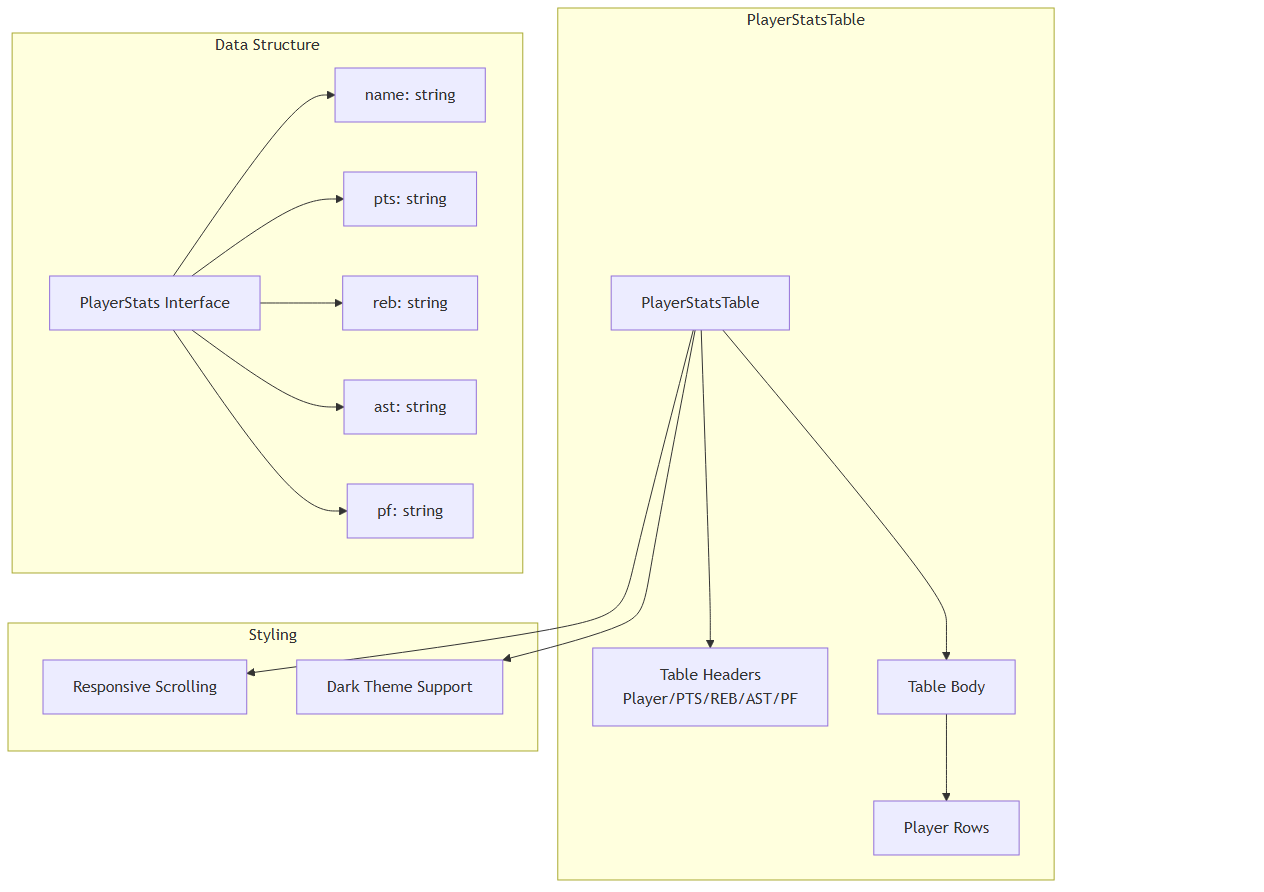


The scoreboard uses conditional styling to indicate which team has possession and includes fallback team abbreviations when logos are unavailable.

Sources: [src/components/Scoreboard.tsx:3-49](file:///C:\src\components\Scoreboard.tsx), [src/components/Scoreboard.tsx:52-98](file:///C:\src\components\Scoreboard.tsx)

### Player Statistics Table

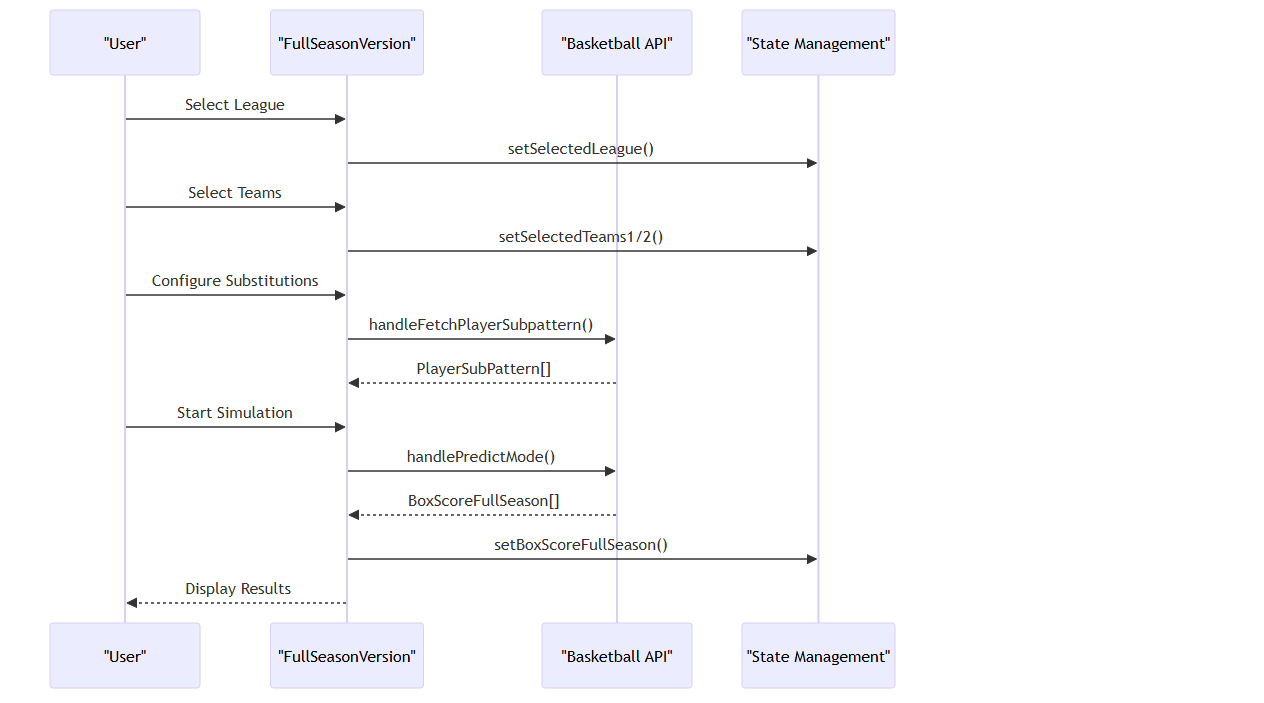
The PlayerStatsTable component provides a reusable interface for displaying player performance data:



Sources: [src/components/PlayerStatsTable.tsx:11-54](file:///C:\src\components\PlayerStatsTable.tsx)

## Data Flow and State Management

### Full Season Simulation Flow



Sources: [src/pages/FullSeasonVersion.tsx:115-151](file:///C:\src\pages\FullSeasonVersion.tsx), [src/pages/FullSeasonVersion.tsx:291-305](file:///C:\src\pages\FullSeasonVersion.tsx)

### State Coordination

Both simulation interfaces manage complex state through props passed from the parent GameSetup component. The full season interface handles over 20 state variables while the single game interface manages real-time game state:

|  |  |  |
| --- | --- | --- |
| **State Category** | **Full Season** | **Single Game** |
| Team Selection | selectedTeams1/2, selectedLeague | Team mode controls |
| Simulation Data | boxScoreFullSeason, teamsSchedule | Live game stats |
| UI Controls | schedule, location, getAltsSelected | pauseOptions, displayOptions |
| Player Management | playersTeam1/2, playerSubPattern | Real-time player stats |

Sources: [src/pages/FullSeasonVersion.tsx:115-151](file:///C:\src\pages\FullSeasonVersion.tsx), [src/pages/SingleGameVersion.tsx:46-56](file:///C:\src\pages\SingleGameVersion.tsx)

## User Interaction Patterns

### Export and Persistence

Both interfaces provide data export capabilities using the exportToCSV utility function. The full season interface supports exporting player statistics, substitution patterns, and season data, while maintaining persistent state across simulation runs.

### Visual Feedback Systems

The interfaces implement several visual feedback mechanisms:  
- **Attention Indicators**: Pulsing buttons (pulse-attention class) when user action is required  
- **Loading States**: Spinner components during API operations  
- **Toast Notifications**: Error and success messages for user actions  
- **Drag Feedback**: Visual cues during player substitution management

Sources: [src/pages/FullSeasonVersion.tsx:354-365](file:///C:\src\pages\FullSeasonVersion.tsx), [src/pages/FullSeasonVersion.tsx:227-232](file:///C:\src\pages\FullSeasonVersion.tsx), [src/lib/utils.ts:8-42](file:///C:\src\lib\utils.ts)

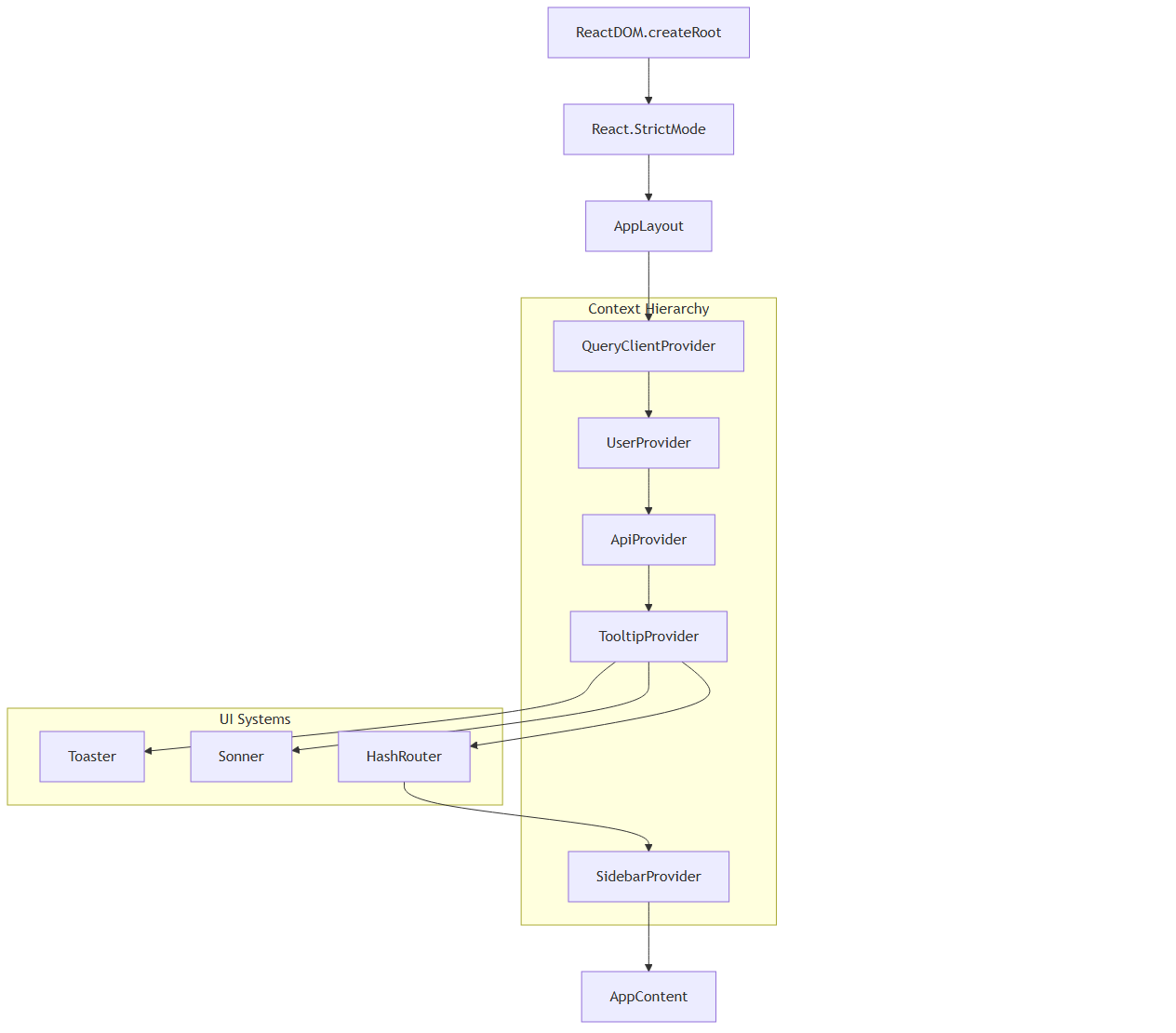
# Layout and Navigation

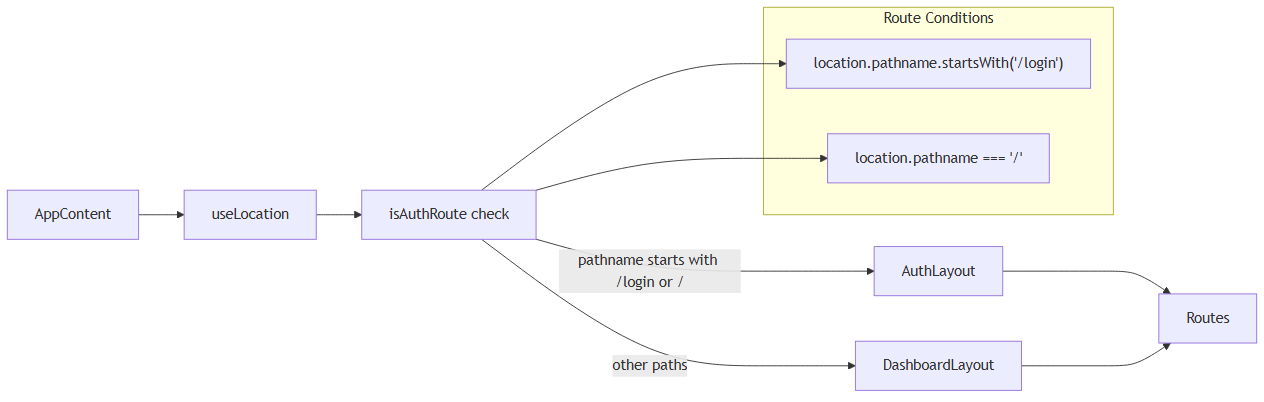
This document covers the NBA simulation application's layout system, routing architecture, and navigation components. The layout system provides a hierarchical provider structure with distinct layouts for authentication and game simulation interfaces.

For information about the core UI components used within these layouts, see [Core Components](#_Core_Components). For details about the game-specific interfaces that render within these layouts, see [Game Setup Interface](#_Game_Setup_Interface) and [Game Simulation Interfaces](#_Game_Simulation_Interfaces).

## Application Bootstrap Architecture

The application uses a hierarchical provider system established in AppLayout with distinct layouts for authentication and game simulation states. The architecture follows a layered approach with context providers wrapping the routing system.

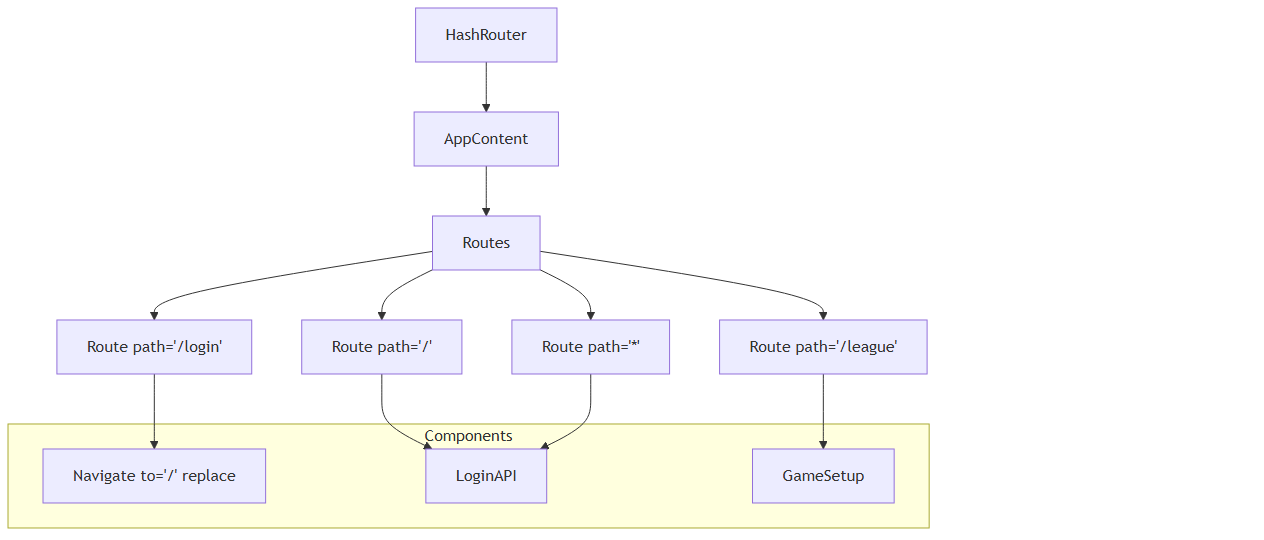




Sources: [src/layout.tsx:44-62](file:///C:\src\layout.tsx), [src/layout.tsx:22-42](file:///C:\src\layout.tsx)

## Routing System

The application uses HashRouter for navigation with a simple route configuration that separates authentication and game simulation flows.



### Route Mapping

|  |  |  |
| --- | --- | --- |
| **Route** | **Component** | **Purpose** |
| / | LoginAPI | Authentication entry point |
| /login | Redirect to / | Login redirect handler |
| /league | GameSetup | Main game setup interface |
| \* | LoginAPI | Catch-all fallback |

### Layout Selection

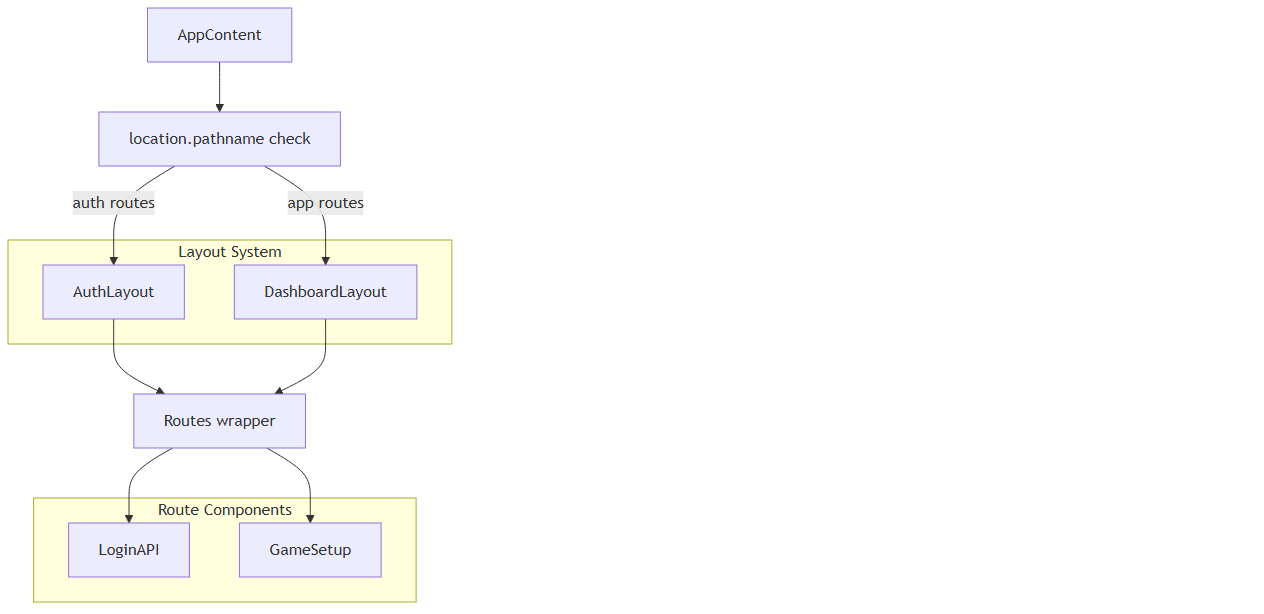
The AppContent component determines layout based on route path:

const isAuthRoute = location.pathname.startsWith("/login") || location.pathname === "/";  
const Layout = isAuthRoute ? AuthLayout : DashboardLayout;

Sources: [src/layout.tsx:22-42](file:///C:\src\layout.tsx), [src/layout.tsx:34-40](file:///C:\src\layout.tsx)

## Layout Components

The application uses two primary layout components imported from the components directory to handle different application states.



### Layout Selection Logic

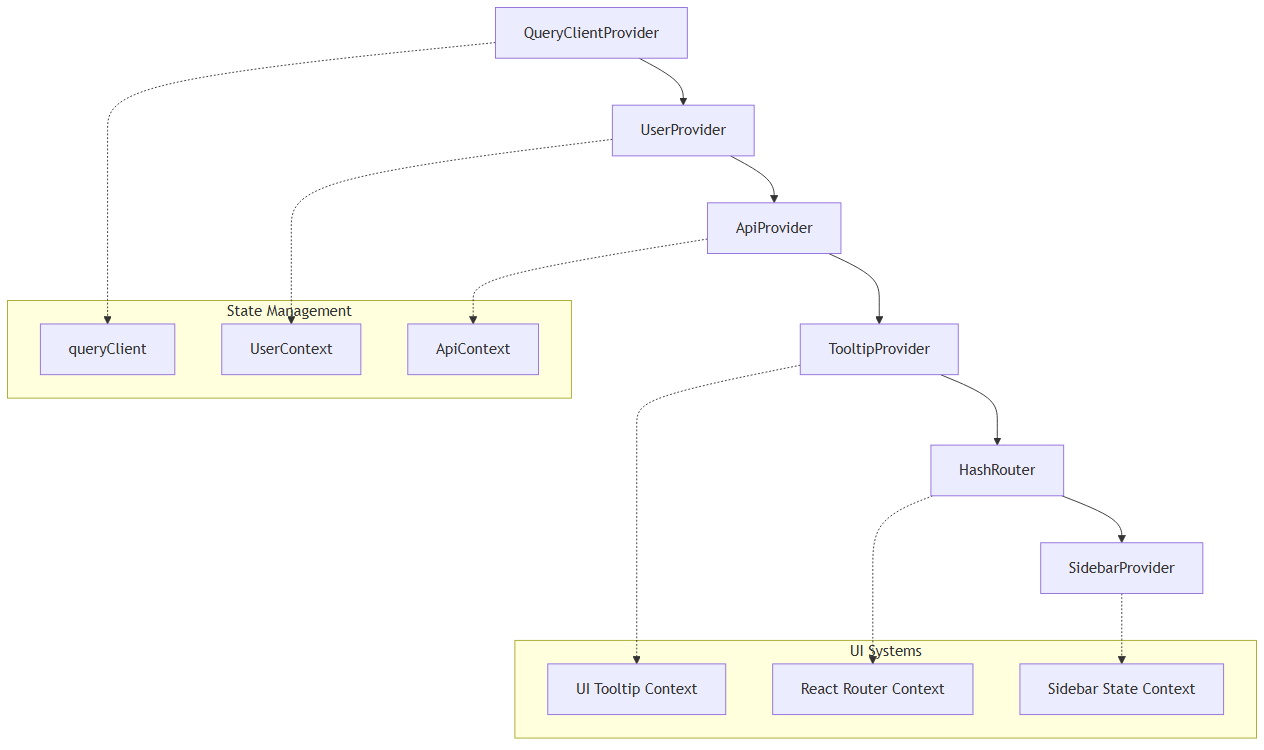
The layout selection follows this pattern:

|  |  |  |
| --- | --- | --- |
| **Condition** | **Layout** | **Used For** |
| pathname.startsWith("/login") | AuthLayout | Login flows |
| pathname === "/" | AuthLayout | Root authentication |
| All other paths | DashboardLayout | Game simulation |

Sources: [src/layout.tsx:16-18](file:///C:\src\layout.tsx), [src/layout.tsx:25-27](file:///C:\src\layout.tsx)

## Context Provider Hierarchy

The application establishes a sophisticated context provider hierarchy in AppLayout that manages global state, API communication, and UI systems.



### Provider Responsibilities

|  |  |  |
| --- | --- | --- |
| **Provider** | **Purpose** | **Scope** |
| QueryClientProvider | React Query client management | API caching and synchronization |
| UserProvider | User authentication state | User session and profile data |
| ApiProvider | API context aggregation | Unified API interface |
| TooltipProvider | UI tooltip system | Component tooltip management |
| SidebarProvider | Sidebar state management | Sidebar collapse/expand state |

### Notification Systems

The application includes dual notification systems:

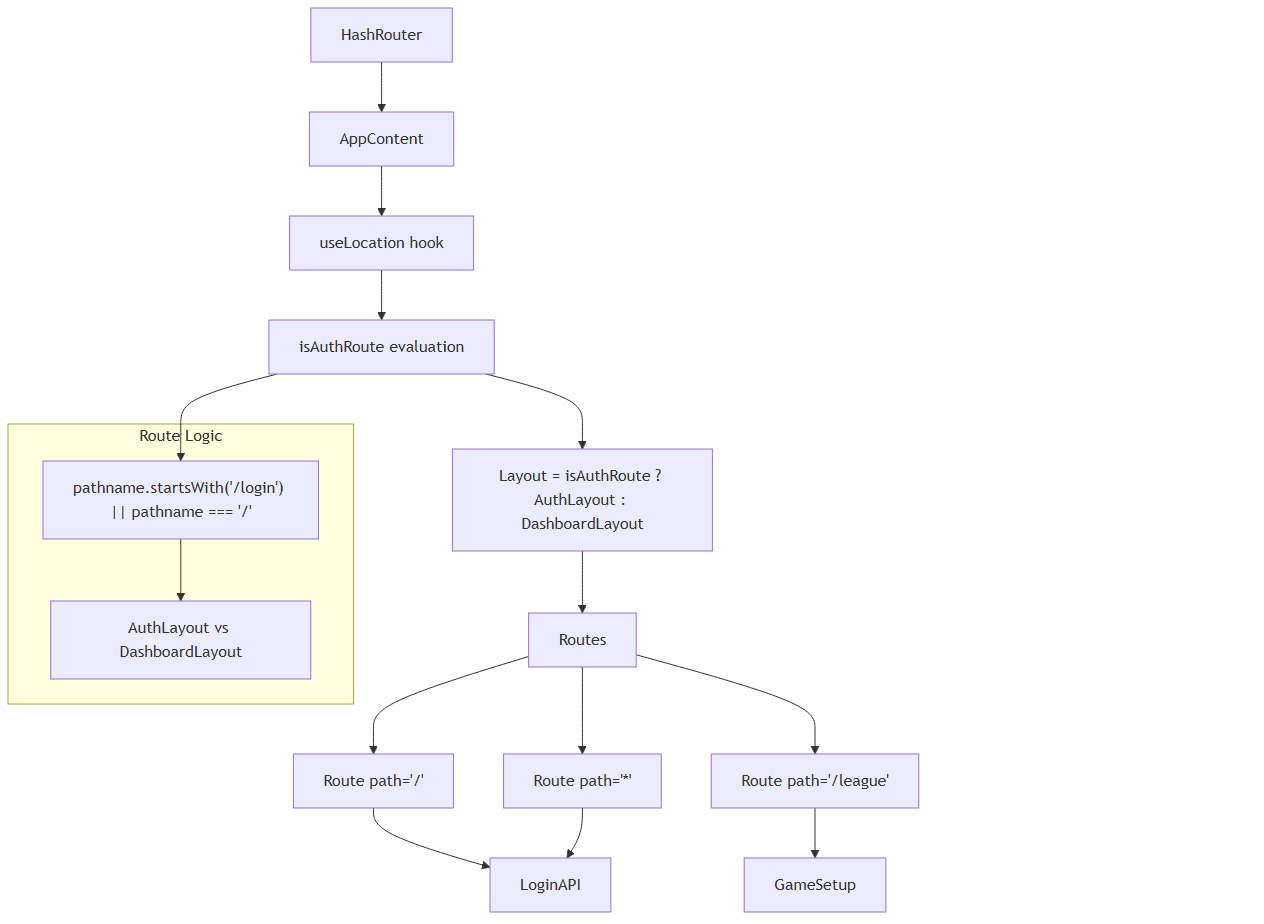
<Toaster />  
<Sonner />

These provide toast notifications and sound notifications for user feedback.

Sources: [src/layout.tsx:44-60](file:///C:\src\layout.tsx), [src/layout.tsx:6-9](file:///C:\src\layout.tsx)

## Navigation Integration

The application integrates navigation through React Router with location-aware layout selection and component rendering.



### Component Navigation Patterns

The application uses several navigation patterns:

|  |  |  |
| --- | --- | --- |
| **Pattern** | **Implementation** | **Usage** |
| Route-based Layout | isAuthRoute conditional | Layout switching |
| Component Routing | Routes with Route elements | Page rendering |
| Programmatic Navigation | useNavigate hook | Component-triggered navigation |
| Location Awareness | useLocation hook | Active state management |

### Route Guard Implementation

The route configuration implements implicit authentication guards:

• **Auth Routes** (/, /login): Use AuthLayout, render LoginAPI

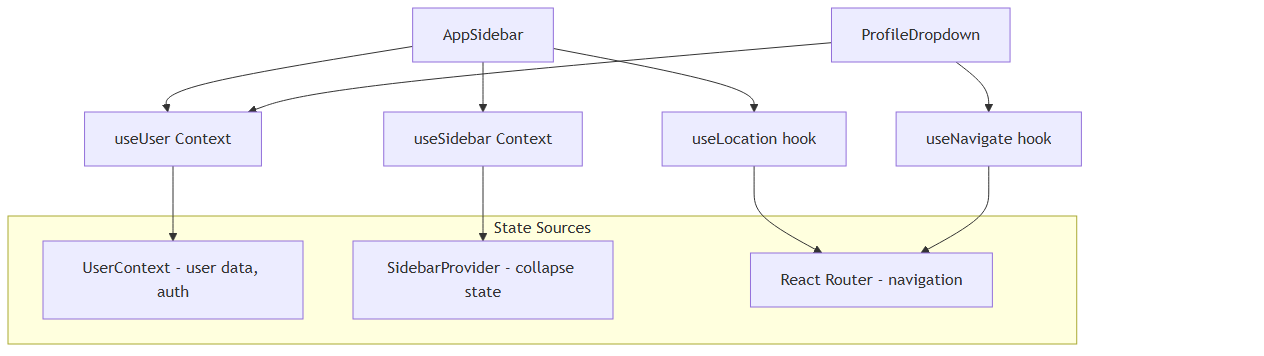
• **App Routes** (/league): Use DashboardLayout, render GameSetup

• **Fallback**: All unknown routes redirect to authentication

Sources: [src/layout.tsx:22-30](file:///C:\src\layout.tsx), [src/layout.tsx:34-40](file:///C:\src\layout.tsx)

## Navigation State Integration

The layout and navigation components integrate with several React contexts for state management:



The navigation system maintains active route highlighting through location.pathname comparison and provides seamless navigation through React Router integration.

Sources: [src/components/AppSidebar.tsx:24-38](file:///C:\src\components\AppSidebar.tsx), [src/components/ProfileDropdown.tsx:13-19](file:///C:\src\components\ProfileDropdown.tsx)

# Authentication UI

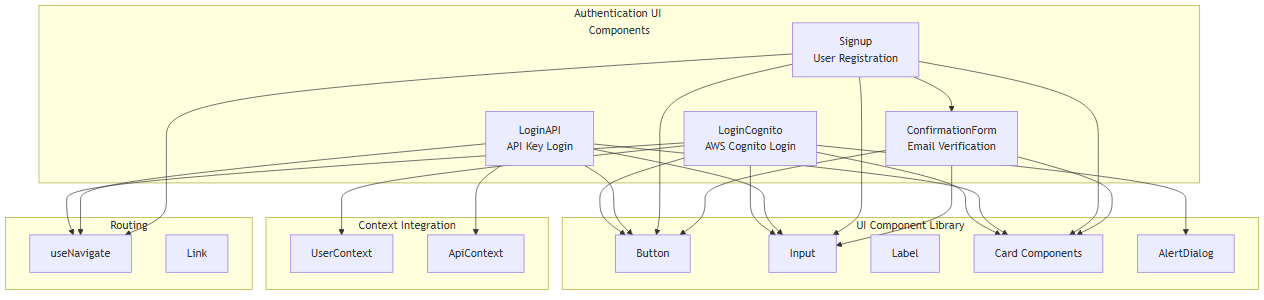
## Purpose and Scope

This document covers the user interface components responsible for authentication workflows in the react-nba-qlx application. It focuses on the presentation layer and user interaction patterns for login, signup, and confirmation processes. For the underlying authentication system architecture and context management, see [Authentication System](file:///C:\Repositorios\md2docx\output\6_Authentication_System.md). For overall application state management patterns, see [State Management](file:///C:\Repositorios\md2docx\output\5_State_Management.md).

The authentication UI system supports two distinct authentication methods:  
- AWS Cognito user account authentication with email confirmation  
- Direct API key authentication for basketball simulation services

## Authentication UI Component Architecture

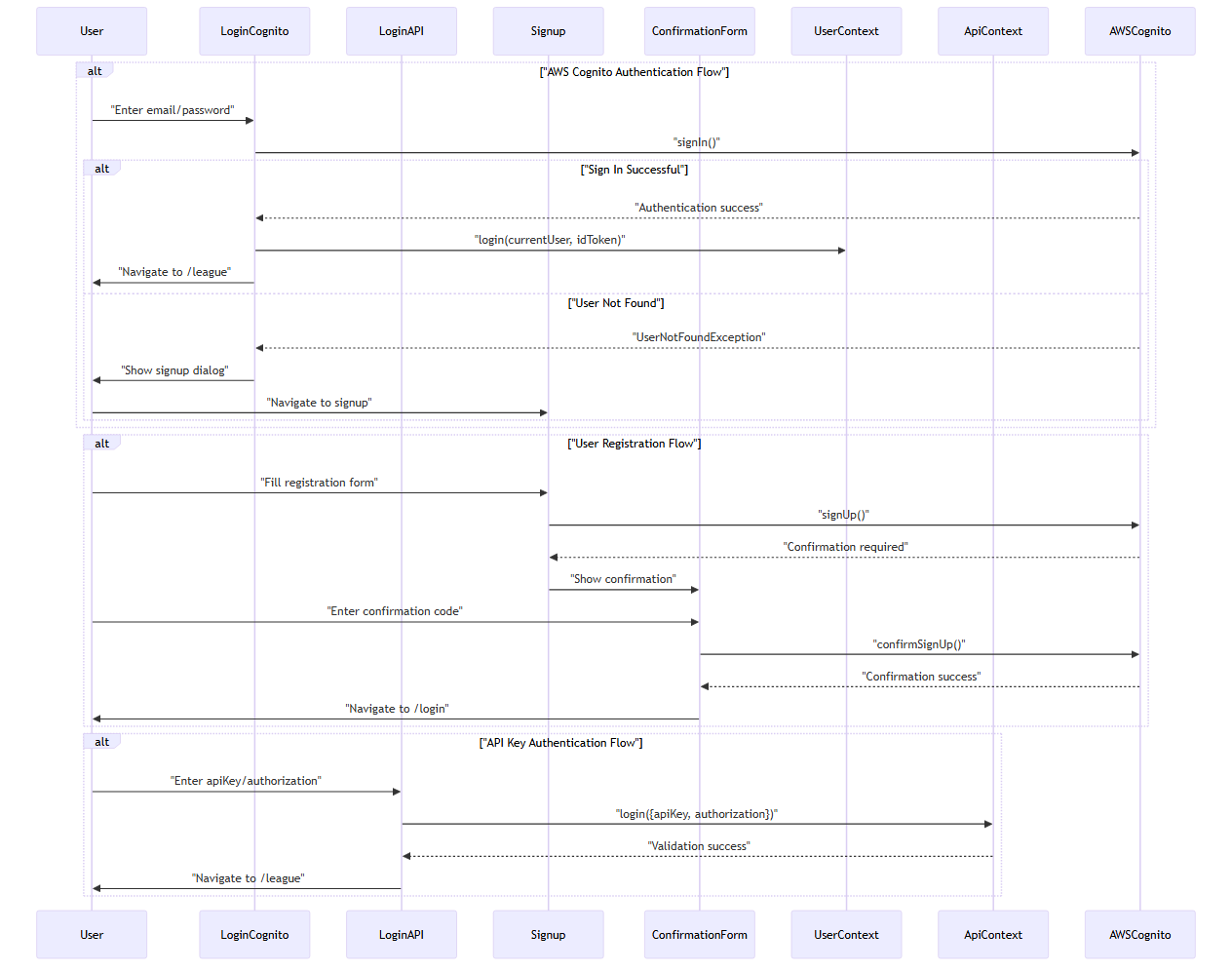
The authentication UI is organized into specialized components that handle different aspects of the user authentication flow:



**Sources:** [src/pages/auth/loginCognito.tsx:1-314](file:///C:\src\pages\auth\loginCognito.tsx), [src/LoginAPI.tsx:1-172](file:///C:\src\LoginAPI.tsx), [src/pages/auth/signup.tsx:1-321](file:///C:\src\pages\auth\signup.tsx), [src/pages/auth/ConfirmationForm.tsx:1-76](file:///C:\src\pages\auth\ConfirmationForm.tsx)

## User Authentication Flows

The authentication UI supports multiple user flows depending on the authentication method and user state:



**Sources:** [src/pages/auth/loginCognito.tsx:44-97](file:///C:\src\pages\auth\loginCognito.tsx), [src/pages/auth/signup.tsx:91-141](file:///C:\src\pages\auth\signup.tsx), [src/pages/auth/signup.tsx:143-174](file:///C:\src\pages\auth\signup.tsx), [src/LoginAPI.tsx:42-52](file:///C:\src\LoginAPI.tsx)

## Component Implementation Details

### LoginCognito Component

The LoginCognito component provides AWS Cognito authentication with a sophisticated two-panel layout:

|  |  |  |
| --- | --- | --- |
| **Feature** | **Implementation** | **Location** |
| Form State | FormState interface with username/password | src/pages/auth/loginCognito.tsx:22-25 |
| Password Visibility | Toggle between text/password input types | src/pages/auth/loginCognito.tsx:29 |
| Error Handling | Displays authentication errors and user guidance | src/pages/auth/loginCognito.tsx:83-96 |
| Split Layout | Left panel for form, right panel for branding | src/pages/auth/loginCognito.tsx:119-309 |
| User Context Integration | Calls login() from useUser() hook | src/pages/auth/loginCognito.tsx:69 |

The component handles multiple authentication scenarios including new password requirements and user not found errors.

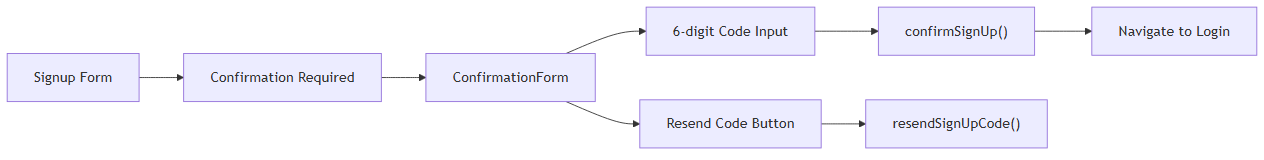
### Signup Component

The Signup component manages user registration with comprehensive form validation:

|  |  |  |
| --- | --- | --- |
| **Feature** | **Implementation** | **Location** |
| Form Interface | IForm with user attributes | src/pages/auth/signup.tsx:21-33 |
| Email Validation | Custom regex validation | src/pages/auth/signup.tsx:74-76 |
| Password Confirmation | Client-side password matching | src/pages/auth/signup.tsx:97-101 |
| AWS Integration | Uses signUp() from aws-amplify/auth | src/pages/auth/signup.tsx:106-118 |
| State Management | Tracks loading, error, success states | src/pages/auth/signup.tsx:50-53 |

### ConfirmationForm Component

The ConfirmationForm component handles email verification during registration:



**Sources:** [src/pages/auth/ConfirmationForm.tsx:7-16](file:///C:\src\pages\auth\ConfirmationForm.tsx), [src/pages/auth/ConfirmationForm.tsx:41-58](file:///C:\src\pages\auth\ConfirmationForm.tsx)

### LoginAPI Component

The LoginAPI component provides API key-based authentication for direct service access:

|  |  |  |
| --- | --- | --- |
| **Feature** | **Implementation** | **Location** |
| Form State | FormState with apiKey/authorization | src/LoginAPI.tsx:22-25 |
| Background Image | NBA-themed background styling | src/LoginAPI.tsx:55-58 |
| API Context Integration | Uses useApi() hook for authentication | src/LoginAPI.tsx:32 |
| Credential Masking | Shows/hides authorization field | src/LoginAPI.tsx:118-137 |

## Form Validation and Error Handling

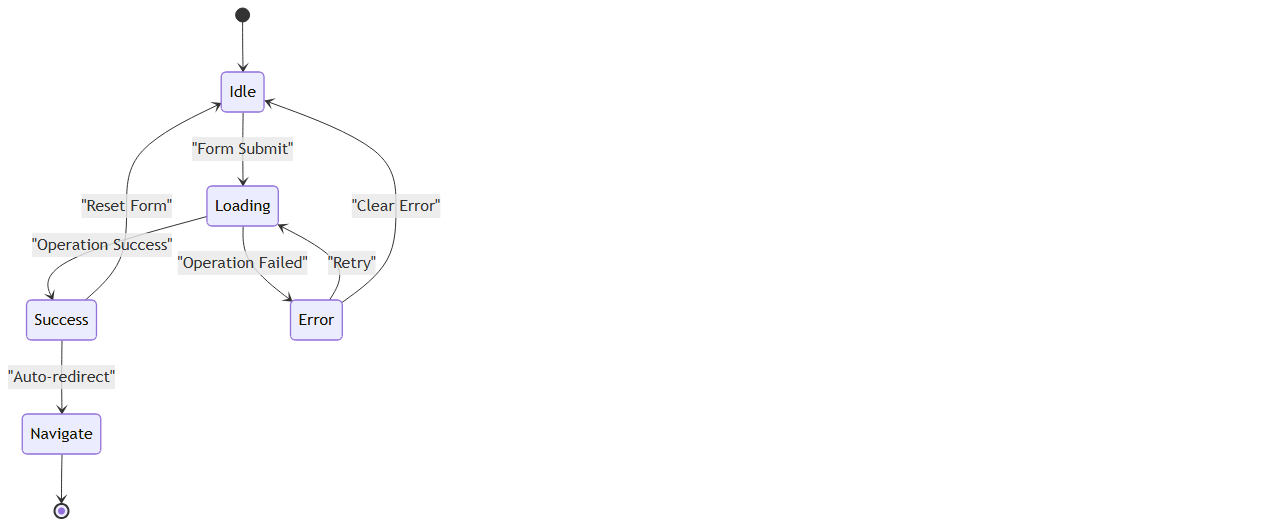
The authentication UI implements comprehensive validation and error handling patterns:

### Validation Rules

|  |  |  |
| --- | --- | --- |
| **Component** | **Validation Rules** | **Implementation** |
| Signup | Email format validation | src/pages/auth/signup.tsx:74-76 |
| Signup | Password minimum length (8 chars) | src/pages/auth/signup.tsx:86-88 |
| Signup | Password confirmation matching | src/pages/auth/signup.tsx:97-101 |
| LoginCognito | Required field validation | src/pages/auth/loginCognito.tsx:50-52 |
| ConfirmationForm | 6-digit code validation | src/pages/auth/ConfirmationForm.tsx:49 |

### Error State Management

All authentication components follow a consistent error handling pattern:



**Sources:** [src/pages/auth/loginCognito.tsx:34](file:///C:\src\pages\auth\loginCognito.tsx), [src/pages/auth/signup.tsx:51](file:///C:\src\pages\auth\signup.tsx), [src/LoginAPI.tsx:31](file:///C:\src\LoginAPI.tsx)

## Styling and Layout Patterns

### Component Layout Structure

The authentication UI uses consistent layout patterns across components:

|  |  |  |
| --- | --- | --- |
| **Layout Element** | **Implementation** | **Usage** |
| Card Container | Card, CardHeader, CardContent, CardFooter | All auth forms |
| Input Groups | Label + Input with spacing | Form fields |
| Icon Integration | Lucide icons (Mail, Lock, Eye) | Visual enhancement |
| Error/Success Messages | Colored background containers | User feedback |
| Button States | Loading and disabled states | Form submission |

### Responsive Design

• **LoginCognito**: Split-screen layout with responsive breakpoints [src/pages/auth/loginCognito.tsx:119-309](file:///C:\src\pages\auth\loginCognito.tsx)

• **Signup**: Single column layout with grid-based field organization [src/pages/auth/signup.tsx:233-258](file:///C:\src\pages\auth\signup.tsx)

• **LoginAPI**: Full-screen background with centered form overlay [src/LoginAPI.tsx:55-67](file:///C:\src\LoginAPI.tsx)

### Visual Themes

• **LoginCognito**: Professional split-panel with animated basketball graphics

• **LoginAPI**: NBA-themed background with semi-transparent form overlay

• **Signup/Confirmation**: Clean, minimal card-based design

**Sources:** [src/pages/auth/loginCognito.tsx:229-308](file:///C:\src\pages\auth\loginCognito.tsx), [src/LoginAPI.tsx:4](file:///C:\src\LoginAPI.tsx), [src/pages/auth/signup.tsx:213-316](file:///C:\src\pages\auth\signup.tsx), [src/pages/auth/ConfirmationForm.tsx:20-72](file:///C:\src\pages\auth\ConfirmationForm.tsx)

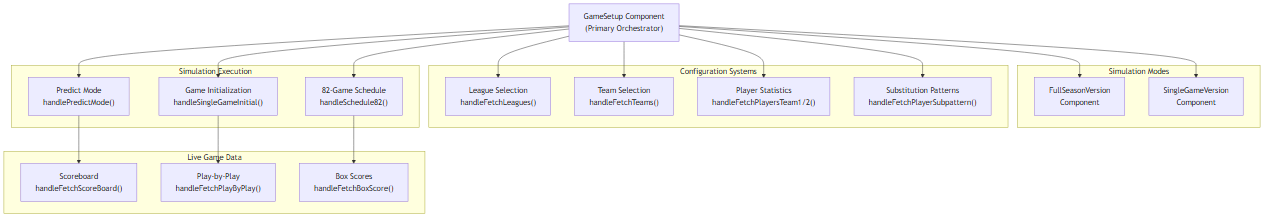
# Game Features

This document covers the basketball simulation functionality and game mechanics implemented in the NBA simulation application. It focuses on the core gameplay features, simulation modes, and the underlying systems that enable realistic basketball game simulation.

For information about the user interface components that display these features, see [Game Simulation Interfaces](#_Game_Simulation_Interfaces). For details about the external API integration that powers the simulations, see [API Integration](#_API_Integration_Architecture).

## Simulation Architecture Overview

The game features are built around a dual-mode simulation system that supports both single games and full season simulations. The architecture centers on the GameSetup component as the primary orchestrator.



Sources: src/pages/GameSetup.tsx:1-947, src/pages/FullSeasonVersion.tsx:1-1023, src/pages/SingleGameVersion.tsx:1-273

## Simulation Modes

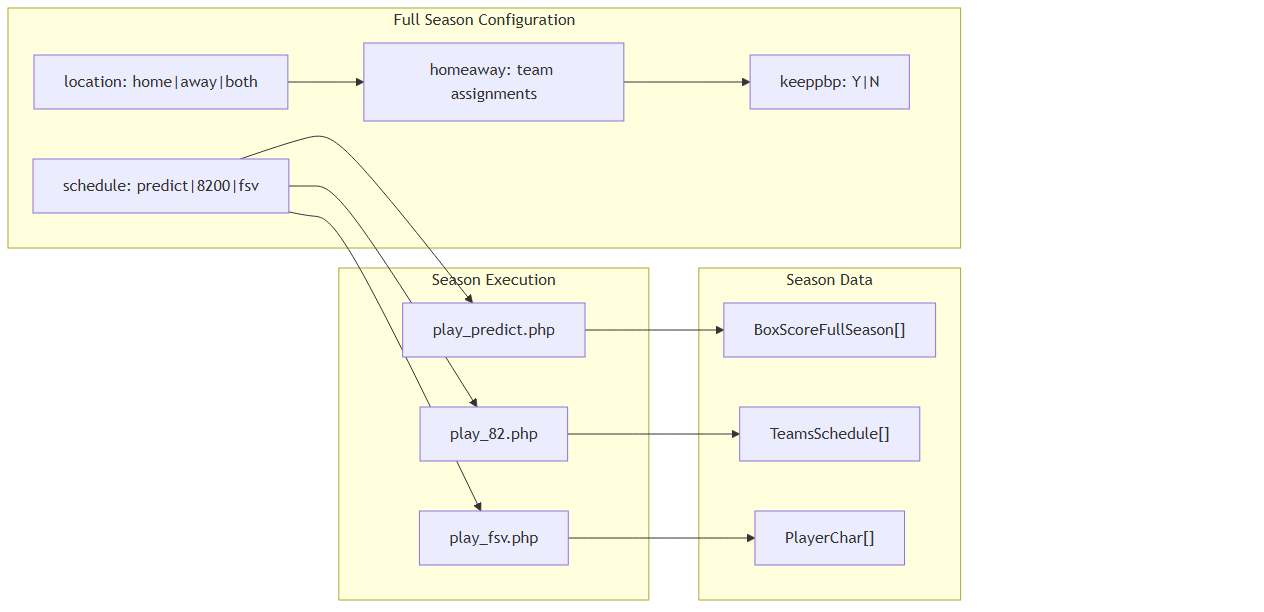
The application supports two primary simulation modes, each designed for different user scenarios and gameplay experiences.

### Full Season Simulation

The full season mode (FullSeasonVersion) provides comprehensive season-long basketball simulation with multiple configuration options:

**Game Mode Options**:  
- predict: Prediction-based simulation for limited games  
- 8200: Full 82-game NBA season simulation  
- fsv: Full season version with enhanced features

**Configuration Parameters**:  
- schedule: Determines the type of season simulation  
- location: Controls home/away game distribution (home, away, both)  
- homeaway: Specific home/away team assignments  
- keeppbp: Whether to maintain play-by-play data (Y/N)



Sources: src/pages/FullSeasonVersion.tsx:147-151, src/pages/GameSetup.tsx:467-494

### Single Game Simulation

The single game mode focuses on detailed, real-time simulation of individual basketball games with comprehensive play-by-play tracking and live statistics.

**Key Features**:  
- Real-time scoreboard updates via ScoreBoard interface  
- Live play-by-play commentary through PlayByPlay[] data  
- Detailed box score tracking with BoxScore[] arrays  
- Interactive game controls and pause options

Sources: src/pages/SingleGameVersion.tsx:12-273, src/pages/GameSetup.tsx:410-426

## Player and Team Data Integration

The simulation system integrates real NBA player statistics and team data to ensure realistic game outcomes.

### Player Statistics System

Player data is managed through the PlayerChar interface, which contains comprehensive statistical profiles:

**Core Player Attributes**:

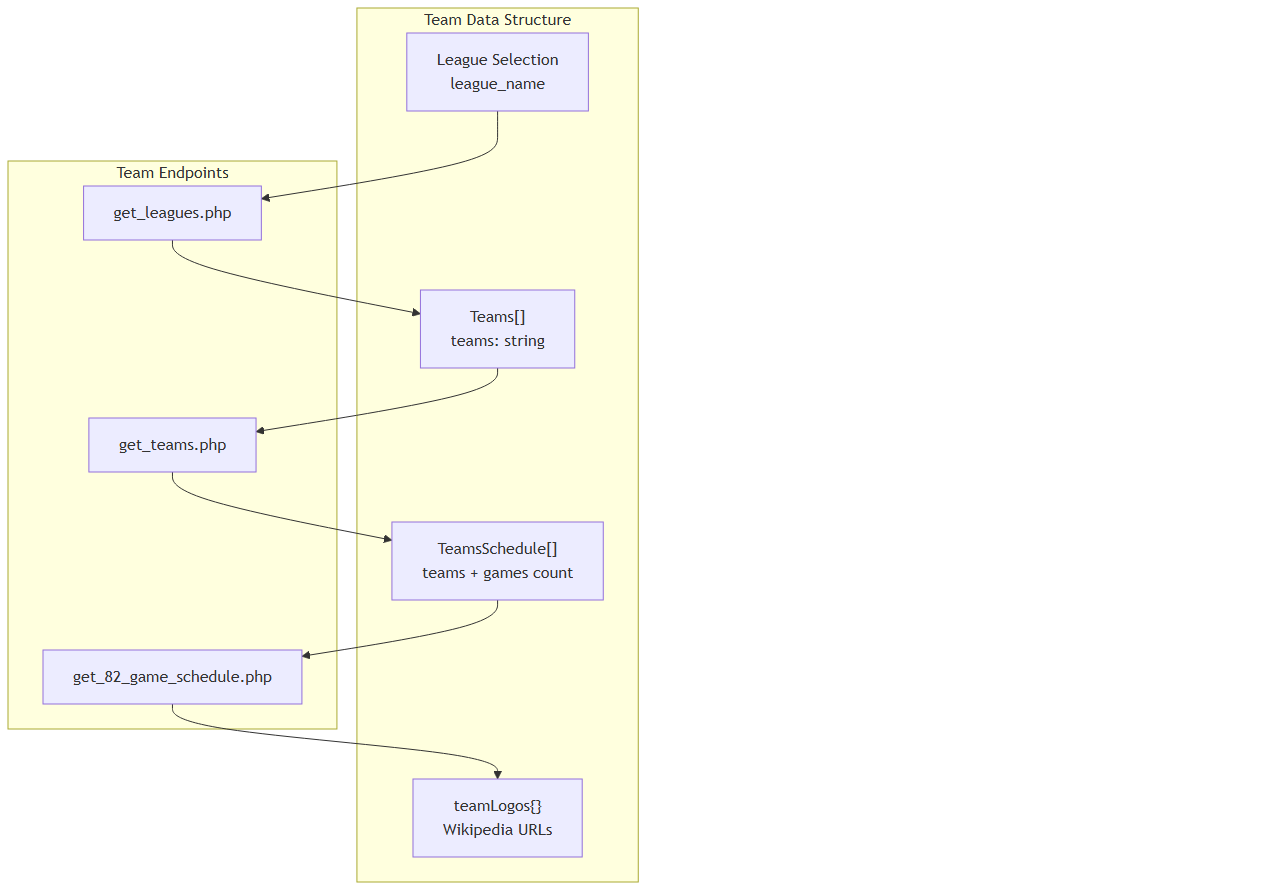
interface PlayerChar {  
 name: string;  
 position: string;   
 poss\_fact: string; // Possession factor  
 two\_pt\_fg\_pct: string; // 2-point field goal percentage  
 ft\_pct: string; // Free throw percentage  
 three\_pt\_pct\_shot: string; // 3-point shot percentage  
 pct\_fouled: string; // Fouling percentage  
 pct\_to: string; // Turnover percentage  
 off\_reb: string; // Offensive rebounds  
 def\_reb: string; // Defensive rebounds  
 // ... additional statistical fields  
}

**Player Data Endpoints**:  
- get\_actual\_player\_stats.php: Retrieves real player statistics  
- get\_players\_chars.php: Fetches editable player characteristics  
- get\_players\_subs.php: Manages substitution patterns

Sources: src/pages/GameSetup.tsx:46-84, src/pages/GameSetup.tsx:516-538

### Team Selection and Management

Teams are organized by league with comprehensive logo and scheduling support:



Sources: src/pages/GameSetup.tsx:235-266, src/pages/GameSetup.tsx:371-408, src/pages/GameSetup.tsx:428-447

## Advanced Simulation Features

### Substitution Pattern Management

The application includes a sophisticated substitution management system that allows users to define player rotations across 4-minute intervals throughout a 48-minute game.

**Substitution Pattern Structure**:

interface PlayerSubPattern {  
 pos1: string; // Center (C)  
 pos2: string; // Power Forward (PF)   
 pos3: string; // Small Forward (SF)  
 pos4: string; // Shooting Guard (SG)  
 pos5: string; // Point Guard (PG)  
}

The system manages 12 intervals (4-minute segments) across 4 quarters, allowing detailed control over player rotation strategies.

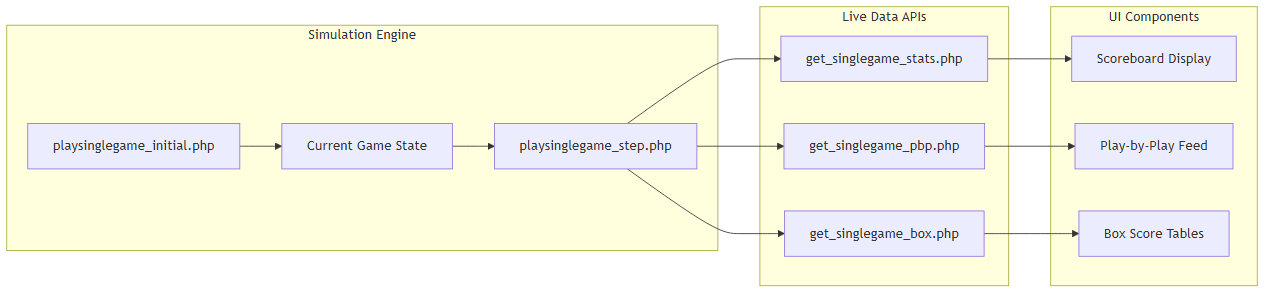
**Substitution Features**:  
- Drag-and-drop player assignment interface  
- Duplicate player validation within intervals  
- Quarter-by-quarter rotation visualization  
- Export functionality for substitution patterns

Sources: src/pages/FullSeasonVersion.tsx:215-242, src/pages/FullSeasonVersion.tsx:682-732

### Live Game Simulation Engine

The simulation engine provides real-time game progression with multiple data streams:

**Live Data Streams**:  
1. **Scoreboard Updates**: Real-time scores, quarters, fouls, possessions  
2. **Play-by-Play**: Detailed action commentary with color coding  
3. **Box Score Tracking**: Cumulative player and team statistics  
4. **Player Performance**: Individual player metrics during simulation



Sources: src/pages/GameSetup.tsx:449-465, src/pages/GameSetup.tsx:565-587, src/pages/GameSetup.tsx:589-633

### Alternative Team Configurations

For advanced simulations, the system supports alternative team configurations through the GetAlts system:

interface GetAlts {  
 alt\_sub: string; // Alternative substitution pattern identifier  
}

This feature enables users to experiment with different team compositions and strategic approaches within the same league framework.

Sources: src/pages/GameSetup.tsx:734-760, src/pages/FullSeasonVersion.tsx:401-419

## Statistical Output and Reporting

The simulation system generates comprehensive statistical reports across multiple formats:

### Box Score Generation

Two distinct box score systems serve different simulation modes:

• **Single Game Box Scores** (BoxScore[]): Line-by-line game statistics

• **Full Season Box Scores** (BoxScoreFullSeason[]): Aggregated season data with game numbering

### Data Export Capabilities

The system includes built-in export functionality through the exportToCSV utility:

exportToCSV<T extends Record<string, any>>(  
 data: T[],  
 filename: string  
)

**Export Options**:  
- Player statistics to CSV format  
- Substitution patterns for analysis  
- Box score data for external processing  
- Team performance metrics

Sources: src/lib/utils.ts:8-42, src/pages/FullSeasonVersion.tsx:441-442

## Game Configuration API Integration

All simulation features integrate with external basketball simulation APIs through a centralized request system:

**Core API Endpoints**:  
- conversionjs: Primary simulation gateway  
- Basketball simulation endpoints for game execution  
- Player and team data retrieval services  
- Statistical reporting and box score generation

The system uses a dual-token authentication approach, combining AWS authentication tokens with specialized NBA API tokens for secure access to simulation services.

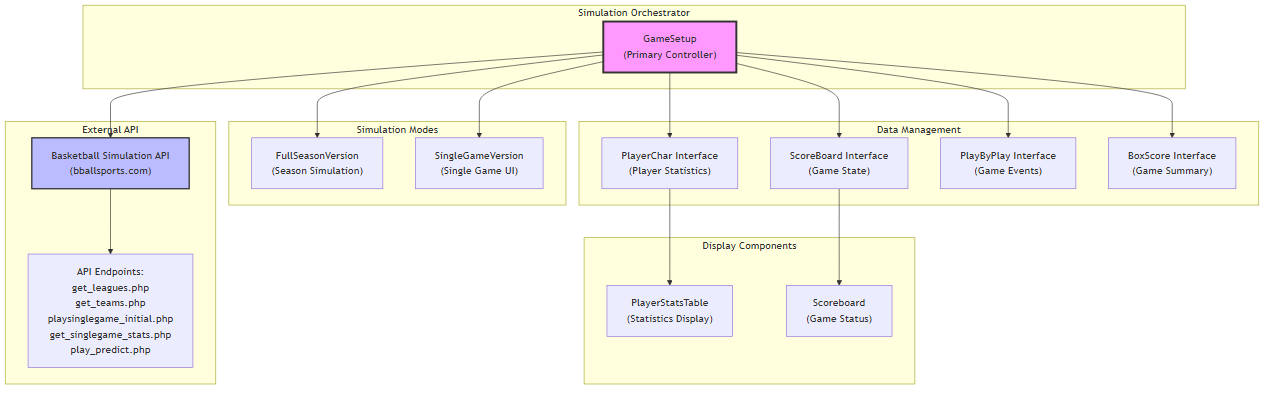
Sources: src/pages/GameSetup.tsx:369-389, src/pages/GameSetup.tsx:467-494

# Basketball Simulation

This document covers the core basketball simulation functionality that powers the NBA game simulation features. The basketball simulation system handles game mechanics, player statistics, real-time game state management, and integration with external basketball APIs. For information about the user interface components that display simulation results, see [Game Simulation Interfaces](#4.3). For details about season-long management features, see [Season Management](#5.2).

## System Architecture

The basketball simulation system is built around a central orchestrator pattern where the GameSetup component coordinates all simulation activities through external API calls and state management.

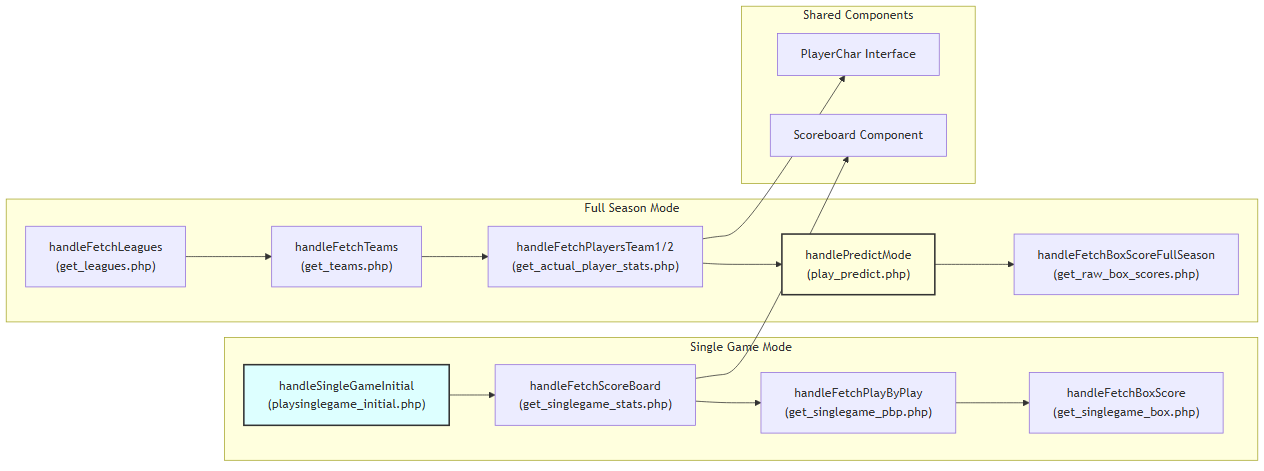


The simulation system operates through a series of coordinated API calls that manage game state, player data, and simulation execution. The GameSetup component serves as the central coordinator, managing data flow between the external basketball simulation API and the application's user interface components.

Sources: src/pages/GameSetup.tsx:1-947, src/components/PlayerStatsTable.tsx:1-57, src/components/Scoreboard.tsx:1-101, src/pages/SingleGameVersion.tsx:1-273

## Game Simulation Modes

The basketball simulation supports two primary modes of operation, each with distinct data flows and API interaction patterns.



### Full Season Simulation

Full season simulation operates through the handlePredictMode function which supports multiple simulation types controlled by the schedule state variable. The simulation can run in three modes: "predict" for predictive analysis, "8200" for full season play, and "fsv" for full season version.

Sources: src/pages/GameSetup.tsx:467-494, src/pages/GameSetup.tsx:287-288

### Single Game Simulation

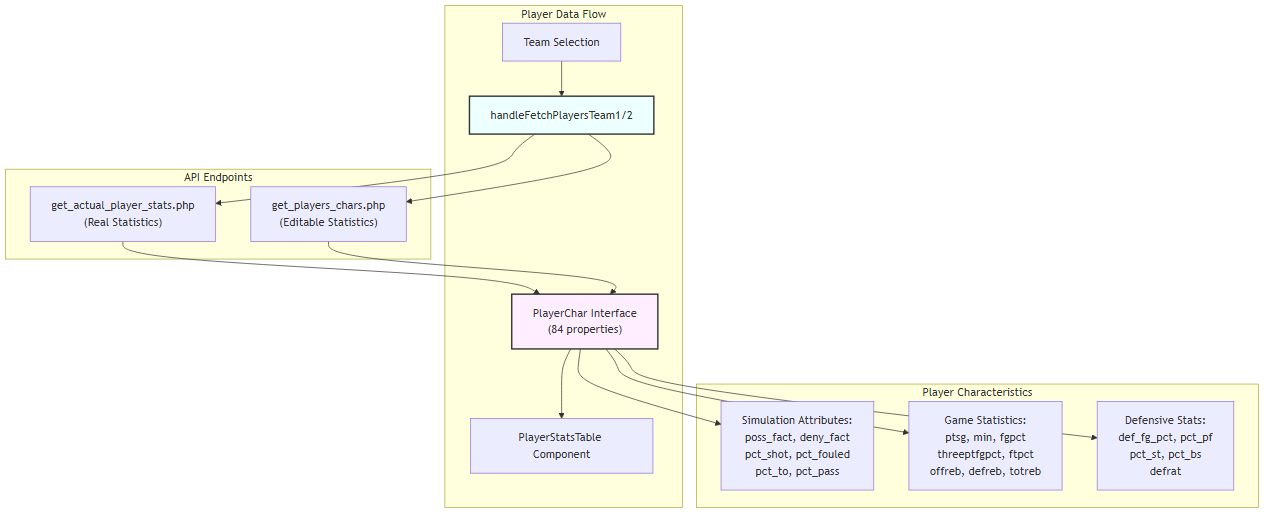
Single game simulation begins with handleSingleGameInitial which sets up the initial game state, followed by real-time updates through handleFetchScoreBoard to retrieve current game status. The system tracks detailed play-by-play events and maintains comprehensive box score statistics.

Sources: src/pages/GameSetup.tsx:410-426, src/pages/GameSetup.tsx:449-465

## Player Data Management

The basketball simulation manages comprehensive player statistics through the PlayerChar interface, which contains both current season statistics and simulation-specific attributes.

|  |  |  |
| --- | --- | --- |
| **Category** | **Properties** | **Purpose** |
| Basic Info | name, position, height, year, team\_code | Player identification |
| Simulation Stats | poss\_fact, two\_pt\_fg\_pct, ft\_pct, pct\_shot | Simulation mechanics |
| Game Stats | ptsg, min, fgpct, threeptfgpct, totreb | Performance metrics |
| Advanced Stats | def\_fg\_pct, pct\_pf, pct\_st, pct\_bs | Defensive abilities |

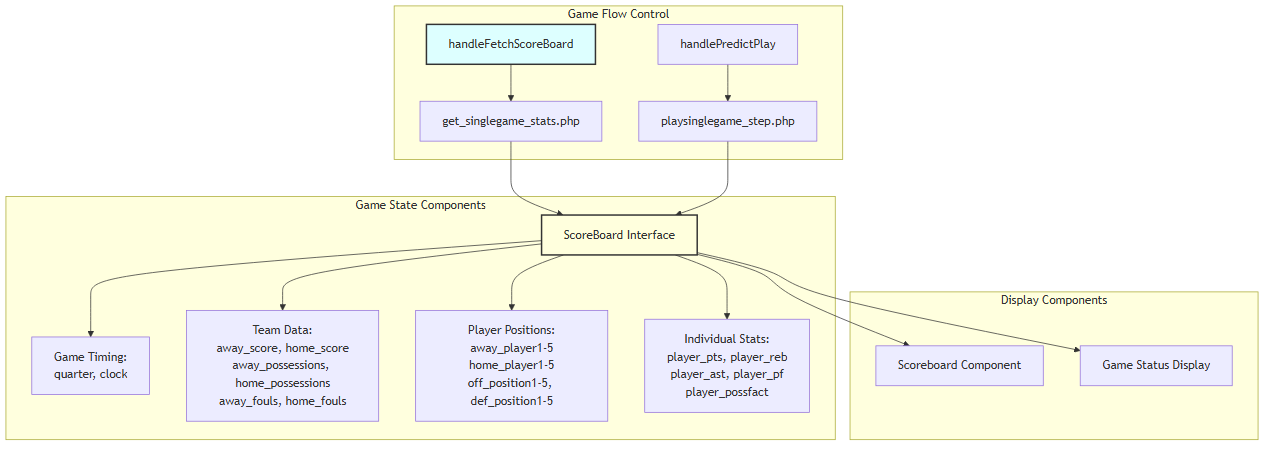


The PlayerChar interface serves dual purposes, containing both actual player statistics from real NBA data and editable simulation parameters. The system maintains separate endpoints for retrieving actual statistics versus simulation-specific characteristics.

Sources: src/pages/GameSetup.tsx:46-84, src/pages/GameSetup.tsx:516-538, src/pages/GameSetup.tsx:540-563

## Real-Time Game State Management

The simulation system maintains real-time game state through the ScoreBoard interface, which tracks comprehensive game information including scores, game clock, player positions, and individual player statistics.

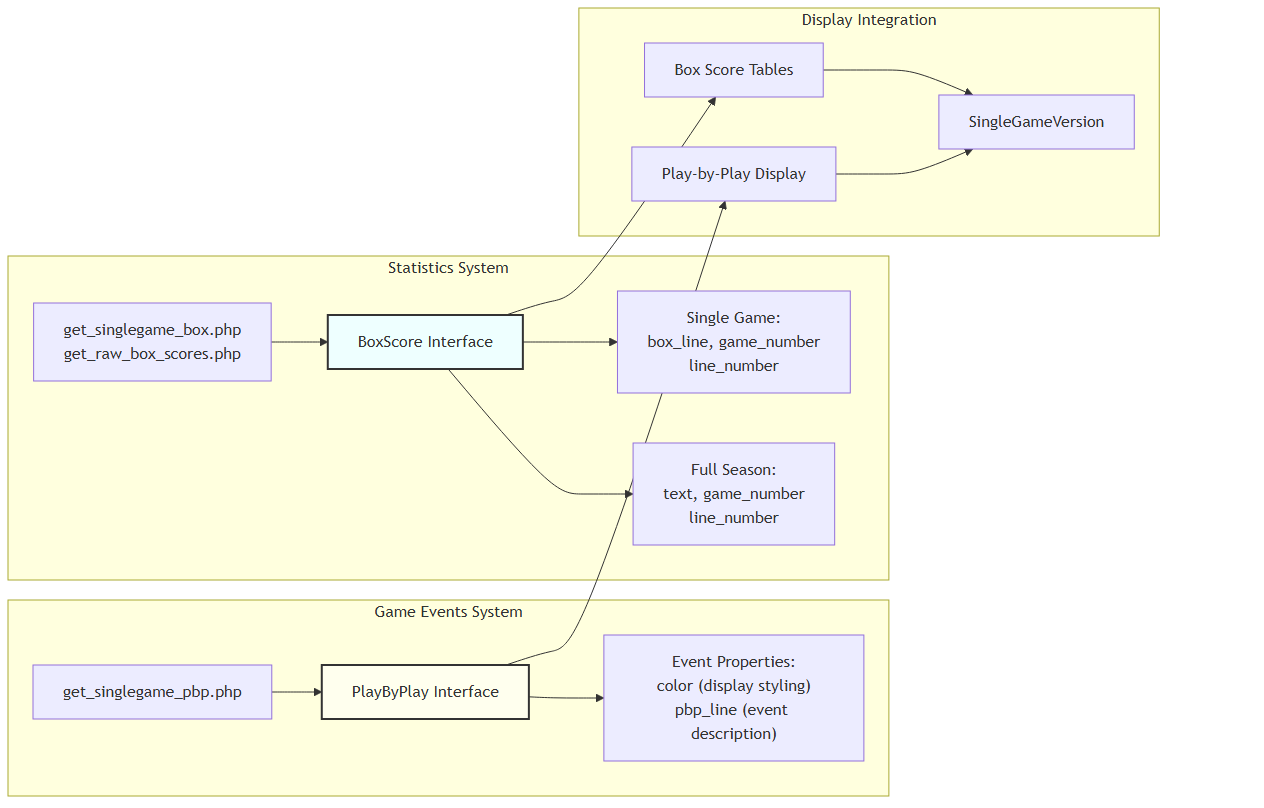


The ScoreBoard interface contains 184 individual properties tracking every aspect of the game state, from basic scores to detailed player positioning and individual performance metrics. The system updates this state through periodic API calls during game simulation.

Sources: src/pages/GameSetup.tsx:94-184, src/pages/GameSetup.tsx:449-465, src/components/Scoreboard.tsx:3-14

## Play-by-Play and Box Score Systems

The simulation generates detailed game narratives through the play-by-play system and comprehensive statistical summaries through box scores.

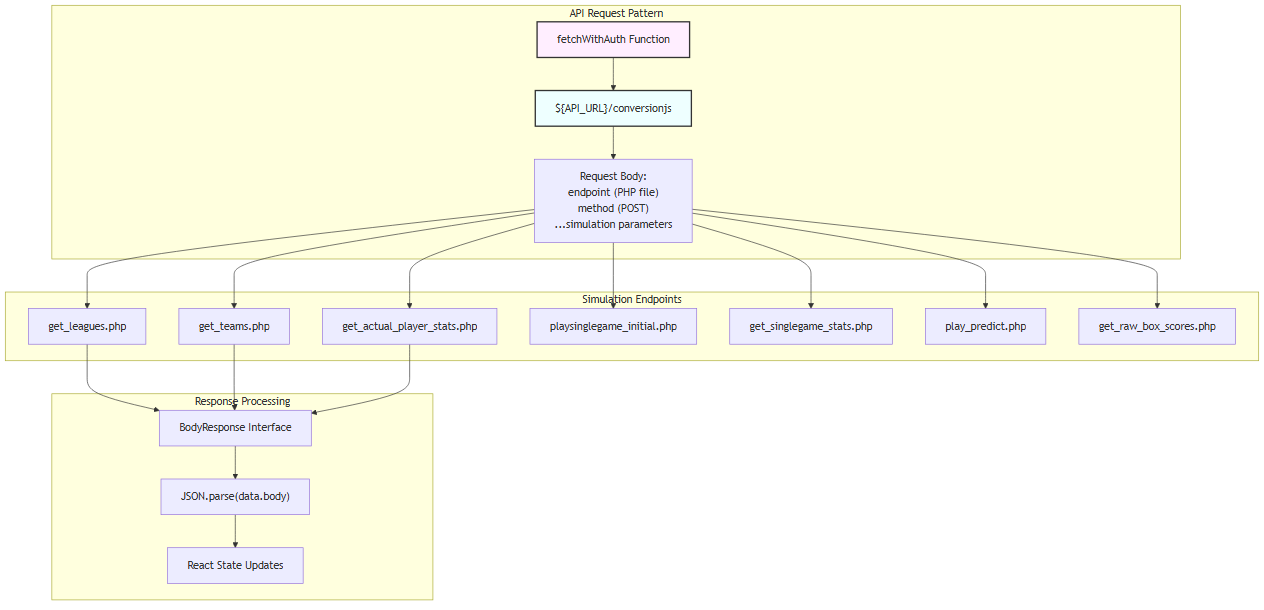


The play-by-play system generates color-coded event descriptions that provide narrative context for game actions. The box score system maintains separate interfaces for single games versus full season statistics, with different data structures optimized for each use case.

Sources: src/pages/GameSetup.tsx:190-197, src/pages/GameSetup.tsx:199-213, src/pages/GameSetup.tsx:565-587, src/pages/GameSetup.tsx:589-633

## API Integration Architecture

The basketball simulation integrates with external APIs through a standardized request pattern that supports multiple simulation endpoints and authentication methods.



All simulation API calls follow a consistent pattern where requests are sent to /conversionjs with a body containing the target PHP endpoint and simulation parameters. Responses are wrapped in a BodyResponse interface that requires JSON parsing to extract the actual data.

Sources: src/pages/GameSetup.tsx:369-389, src/pages/GameSetup.tsx:17-19, src/pages/GameSetup.tsx:414-426, src/pages/GameSetup.tsx:467-494