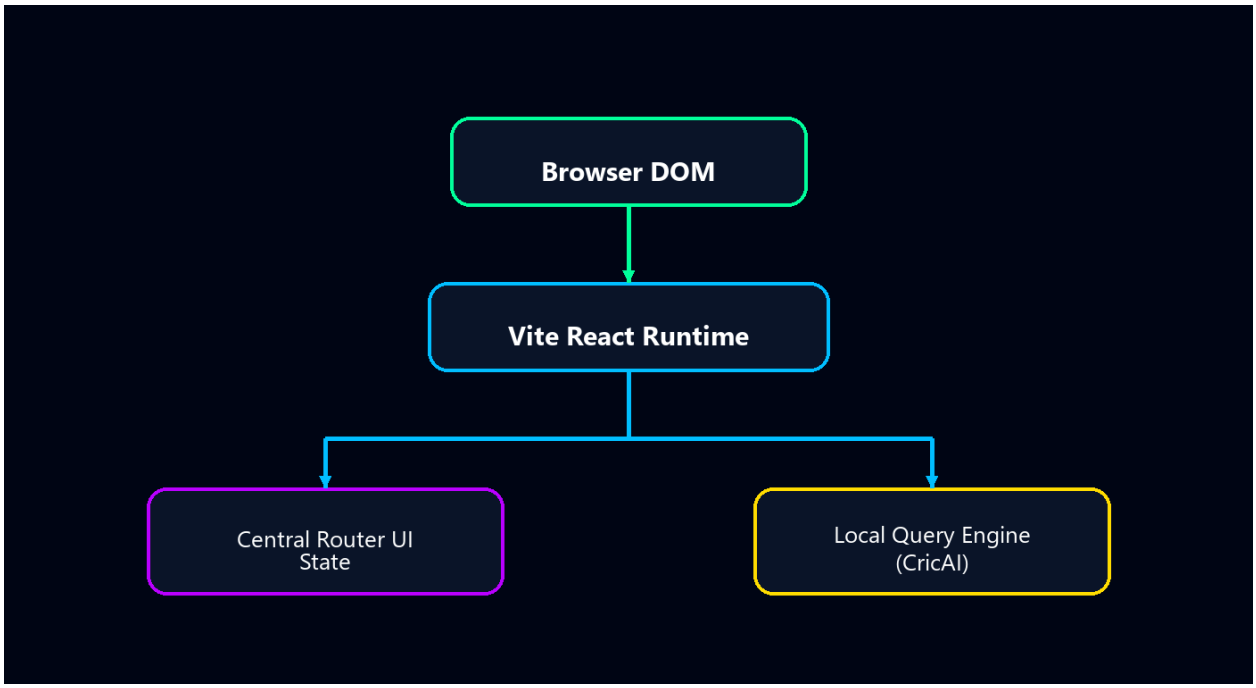
🏏 CricMind — Supporting Docs

This document serves as the comprehensive technical specification and design systems manual for **CricMind**, a premium, IPL Cricket Intelligence & Match Simulator platform.

1. Architecture Explanation

CricMind3D is engineered as a modern, high-performance **Single Page Application (SPA)** that operates under an **offline-first paradigm**.

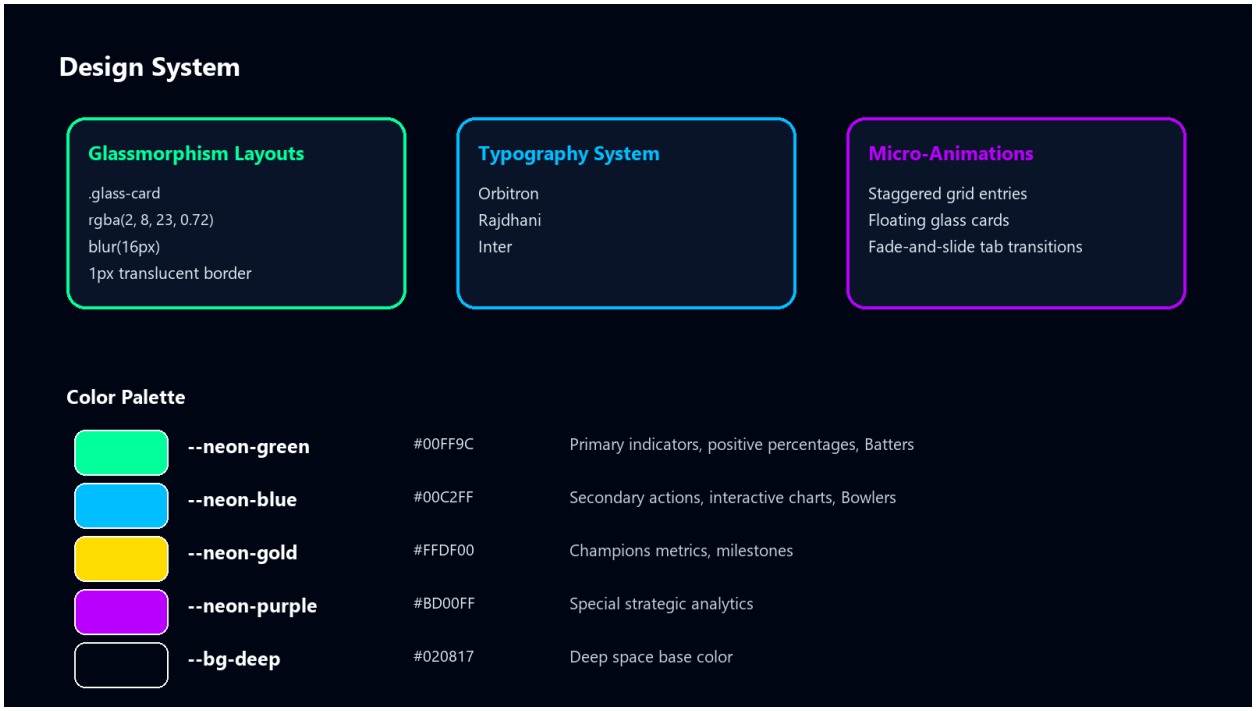


Key Architectural Pillars:

- **Fully Offline-First Data Model:** The application does not rely on external cloud databases, microservices, or APIs. It stores a pre-processed dataset containing **1,226 matches across 19 seasons (2008–2026)** inside a optimized static JSON file (`src/data/ipl_analytics.json`). This ensures instantaneous data retrieval, zero latency, and absolute privacy.
- **Lightweight Routing & Navigation:** To maintain high performance and simplicity, the application bypasses complex routing packages like `react-router-dom` in favor of a centralized React state router managed directly within `App.tsx` (`const [page, setPage] = useState<Page>('home')`).

- **Decoupled Rendering Contexts:** Page displays are isolated into distinct, modular functional components (e.g., `Dashboard`, `Simulator`, `Players`, `CricAI`) that ingest slices of pre-structured data via React props.
- **Asset Ingestion Engine:** Media elements (such as the futuristic looping background video) are managed locally and loaded as seamless HTML5 loop modules, eliminating external network reliance and buffering.

2. Design System



The platform features a **futuristic, cyberpunk-inspired visual hierarchy** that translates raw statistics into an immersive, game-like experience.

Core Visual Elements:

1. **Glassmorphism Layouts:** Components are containerized in semi-transparent, blurred cards designed to float over an active canvas:

`css

.glass-card {

background: rgba(2, 8, 23, 0.72);

backdrop-filter: blur(16px);

-webkit-backdrop-filter: blur(16px);

border: 1px solid rgba(255, 255, 255, 0.08);

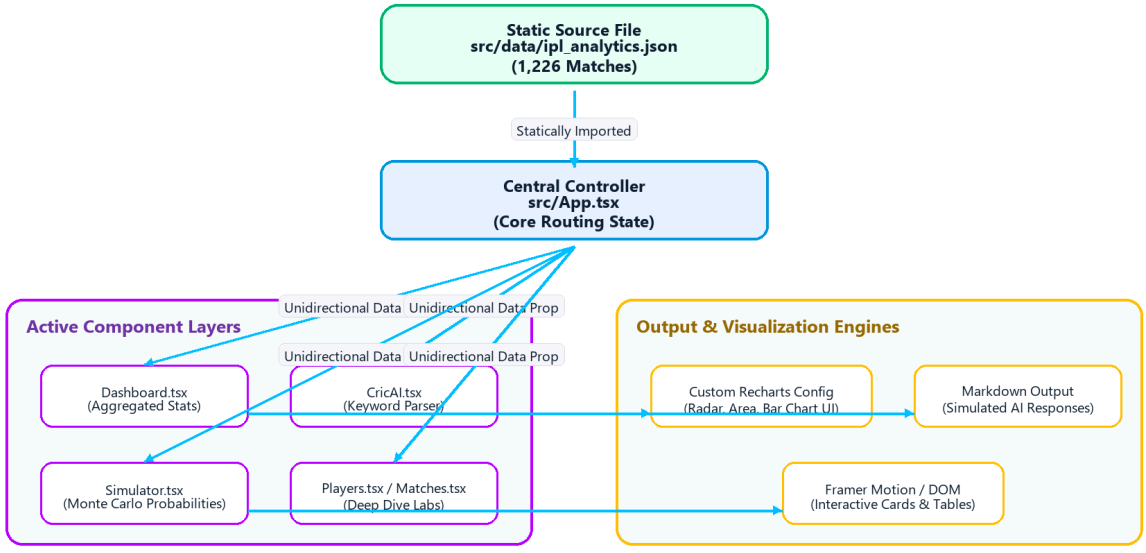
box-shadow: 0 8px 32px 0 rgba(0, 0, 0, 0.37);

}
`

2. **Color Palette:** Standardized in CSS custom properties inside `index.css`:
    - `--neon-green`: `#00FF9C` (Primary indicators, positive percentages, Batters)
    - `--neon-blue`: `#00C2FF` (Secondary actions, interactive charts, Bowlers)
    - `--neon-gold`: `#FFDF00` (Champions metrics, milestones)
    - `--neon-purple`: `#BD00FF` (Special strategic analytics)
    - `--bg-deep`: `#020817` (Deep space base color)
3. **Typography System:**
    - **Display Font:** `Orbitron` (Used for futuristic headers, large metrics, and timers).
    - **Subheadings & Actions:** `Rajdhani` (Condensed, high-tech sans-serif).
    - **Body & Descriptions:** `Inter` (Optimized for text readability in detailed statistics tables).
4. **Micro-Animations (Framer Motion):**
    - Staggered grid entries using `motion.div` with dynamic delays.
    - Glass card floating oscillations to provide spatial depth.
    - Smooth transitions between application tabs (`initial={{ opacity: 0, y: 15 }}`).

3. Data Flow Diagram

The application leverages a unidirectional data flow starting from the static JSON dataset, moving through state distribution, and terminating at specialized visual and text engines.



Step-by-Step Data Lifecycle:

5. **Statically Loaded:** The bundle pre-imports `ipl_analytics.json` during the Vite build pipeline.
6. **Cast to TypeScript Interfaces:** The imported object is cast to strongly-typed models to enforce compile-time safety across batting averages, bowling economy, and head-to-head records.
7. **Unidirectional Propagation:** The central state controller (`App.tsx`) passes the complete dataset reference down to all view modules via props.
8. **Local Derivation & Mapping:** View components filter and process the local dataset on-the-fly rather than duplicating state, avoiding memory leaks and keeping operations instantaneous.

4. Technical Workflow

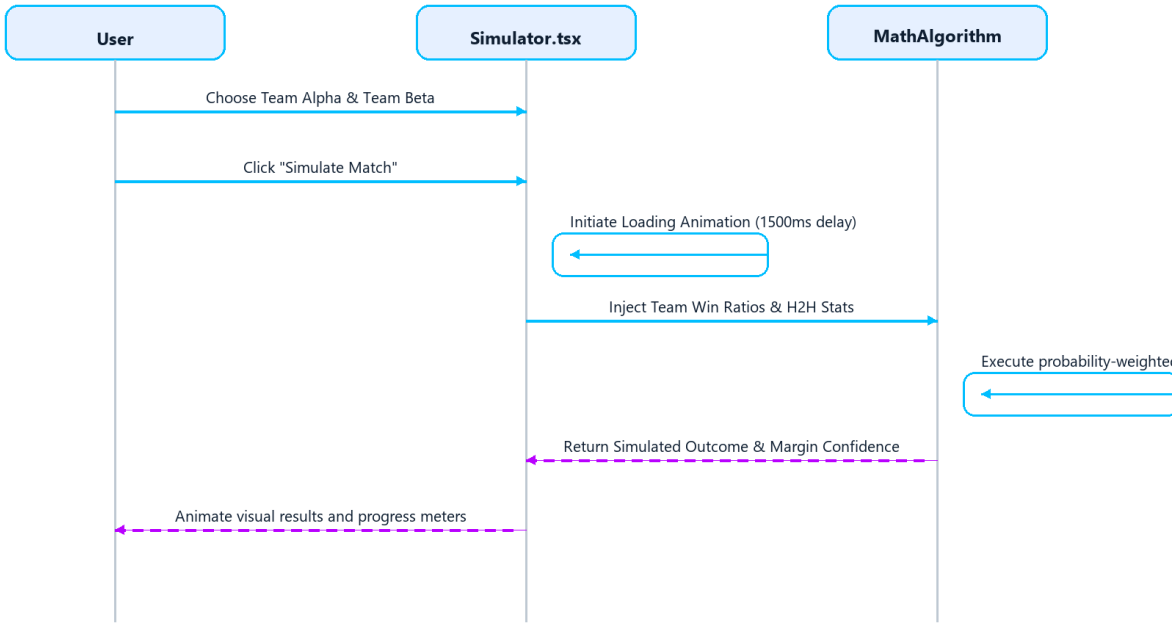
The technical workflow details how specific interactive operations are parsed and executed purely client-side:

A. Navigation & Routing Workflow

When a user clicks an interactive dashboard link:

9. An `onClick` trigger fires a state update: `setPage('targetPage')`.
10. Framer Motion's `<AnimatePresence>` intercepts the current page, performing a smooth fade-and-slide exit animation.
11. The central router conditionally mounts the target component, passing the global analytics object.
12. The new component mounts and triggers entry animations on its child glass panels sequentially.

B. Match Simulation Engine Workflow



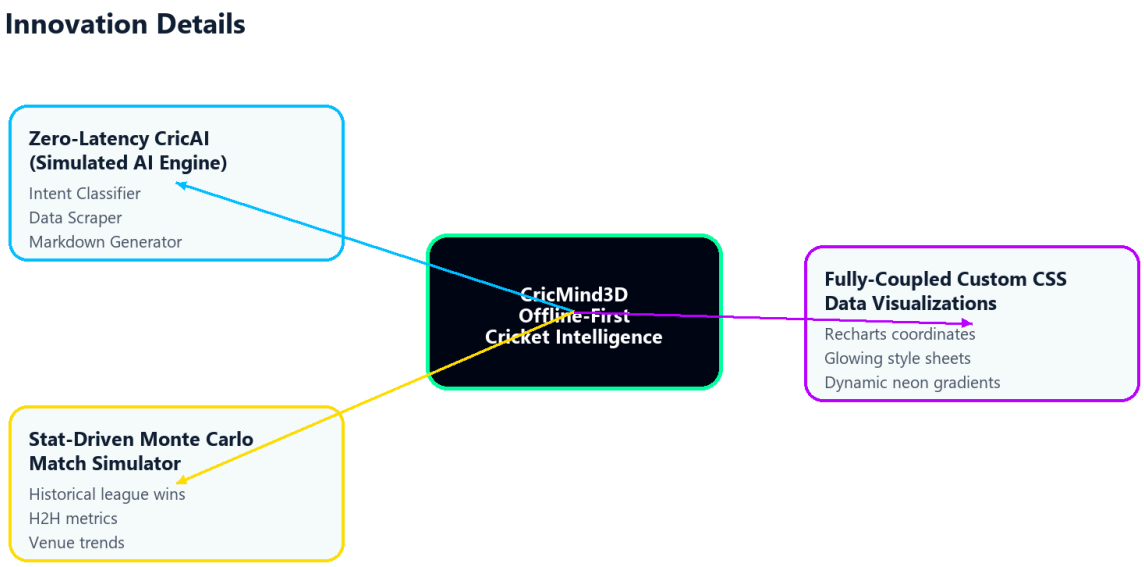
5. Feature Explanation / Specification

Feature Explanation / Specification

Immersive Hub (Home) An	tics Command Center (Dashb	CricAI Bot	Match Simulator	Titan Labs
Tech Stack Framer Motion tickers <SeamlessVideo /> HTML5 loops	**Tech Stack** Recharts Radar, Area, Bar, Polar Angle Charts	**Tech Stack** Regular Expression pattern-matching engine Markdown renderer	**Tech Stack** Probability-based prediction formula weighted factors	**Tech Stack** Tabbed modular state custom search query matching
Visual Element Custom glass cards large glowing number tickers	**Visual Element** Custom neon tooltip boxes custom gradients	**Visual Element** Sleek chatbot console active typing state	**Visual Element** Dynamic confidence progress bars team badge transitions	**Visual Element** Detailed tables interactive statistics radar grids

Feature Module	Primary Utility	Tech Stack Implementation	Visual Element
Immersive Hub (Home)	Welcomes users and displays lifetime statistics summary stats.	Framer Motion tickers, `<SeamlessVideo />` HTML5 loops	Custom glass cards, large glowing number tickers
Analytics Command Center (Dashboard)	Displays historical trends, tossing impact, and inning breakdown.	`Recharts` (Radar, Area, Bar, and Polar Angle Charts)	Custom neon tooltip boxes, custom gradients
CricAI Bot	Explores database queries via natural language chat patterns.	Regular Expression pattern-matching engine, Markdown renderer	Sleek chatbot console with active typing state
Match Simulator	Simulates matches between selected franchises.	Probability-based prediction formula with weighted factors	Dynamic confidence progress bars, team badge transitions
Titan Labs	Explores complete seasons, schedules, teams, and player leaderboards.	Tabbed modular state, custom search query matching	Detailed tables, interactive statistics radar grids

6. Innovation Details



The primary engineering breakthroughs inside CricMind3D center around translating classic data analysis into a frictionless, interactive digital experience:

- **Zero-Latency CricAI (Simulated AI Engine):** Real-time AI chat is simulated client-side. Rather than making high-latency, expensive calls to a backend LLM (which would compromise the offline architecture), the system uses a custom natural language processor:
  - **Intent Classifier:** Analyzes query phrases using optimized regular expression rules.
  - **Data Scraper:** Extracts matching H2H statistics, season titles, or player profiles from the local database.

    - **Markdown Generator:** Synthesizes custom-styled Markdown text blocks instantly, providing rapid insights.
- **Stat-Driven Monte Carlo Match Simulator:** Instead of displaying random probabilities, the match simulator executes a math-focused, weighted-probability distribution based on historical performance ratios. It factors in overall historical league wins, team-specific H2H metrics, and venue trends to output realistic victory scenarios.
- **Fully-Coupled Custom CSS Data Visualizations:** Recharts coordinates are visually integrated with the overall glowing style sheets. Dynamic neon gradients are injected into the SVG definitions (`<linearGradient>`), allowing the charts to visually pop alongside standard dashboard panels.

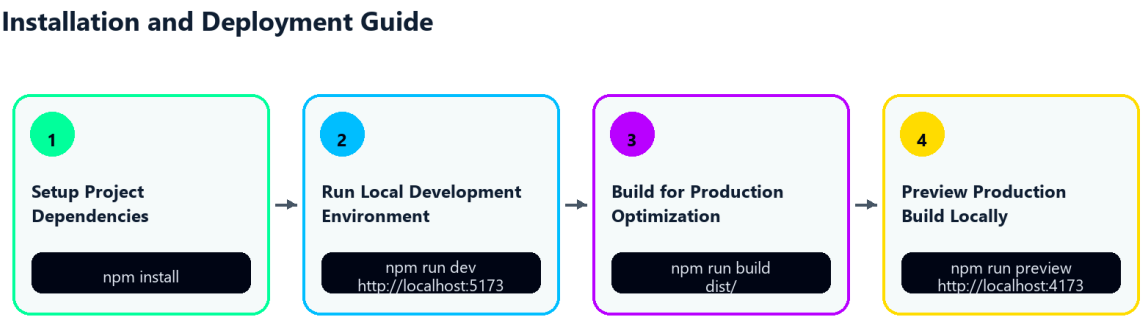
7. Setup & Implementation Details

CricMind3D is configured as a standard Node.js development and build workspace.

Core Package Dependencies

```
"dependencies": {
  "react": "^19.2.6",
  "react-dom": "^19.2.6",
  "recharts": "^3.8.1",
  "framer-motion": "^12.38.0",
  "lucide-react": "^1.16.0"
},
"devDependencies": {
  "vite": "^8.0.12",
  "typescript": "~6.0.2",
  "eslint": "^10.3.0"
}
```

Installation and Deployment Guide



> [!IMPORTANT]
>
> Make sure Node.js (v20 or higher) is installed on your local environment before proceeding.

1. Setup Project Dependencies

Clone the repository and install the project workspace packages:

```
# Install dependencies
npm install
```

2. Run the Local Development Environment

Launch Vite's instantaneous hot-reloading development server:

```
npm run dev
```

Navigate to `http://localhost:5173` in your web browser.

3. Build for Production Optimization

To build the application into optimized, compressed static web assets:

```
npm run build
```

This processes TypeScript compilation and outputs fully self-contained static HTML, JS, and CSS files inside the `dist/` directory.

4. Preview the Production Build Locally

Verify the production-ready assets locally:

```
npm run preview
```

This starts a lightweight server running the compiled assets at `http://localhost:4173`.