High-Level Design Document for Cricket API

Table of Contents

<h3>Detailed Sections</h3>

Introduction

1.1 Purpose

This document provides a high-level overview of the Cricket API system. It details the architecture, design, and interactions between different components, serving as a guide for developers and stakeholders.

1.2 Scope

The document covers the architectural design, including frontend, backend, and database components. It explains how these components interact and integrates third-party APIs for live scores. Security and deployment processes are also discussed.

System Overview

The Cricket API system is designed to provide real-time cricket match information. It allows users to register, login, and view match schedules. Administrators can manage matches, and the system integrates with third-party APIs to fetch live scores.

Architecture

3.1 High-Level Architecture Diagram

The high-level architecture diagram illustrates the interaction between the frontend, backend, and

database. It shows how the frontend communicates with the backend via API endpoints and how the backend interacts with the database.

3.2 Components

Frontend: Built with React, handles user interface and interactions.

Backend: Developed using Node.js, manages API endpoints and business logic.

Database: Stores user and match data, uses MongoDB for flexibility and scalability.

Third-Party API: Provides live scores, integrated into the backend.

Frontend Design

4.1 Structure

The frontend application is structured as follows:

components/: Reusable UI components like LiveScore and MatchCard.

pages/: Main pages including Home, Login, Register, MatchSchedule, and AdminDashboard.

assets/: Static assets such as images and styles.

4.2 Routing

Routing is managed using React Router. Below is an example from <code>App.jsx</code>:

```jsx
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Home from './pages/Home.jsx';
```

```
import MatchSchedule from './pages/MatchSchedule.jsx';
import Login from './pages/Login.jsx';
import Register from './pages/Register.jsx';
import AdminDashboard from './pages/AdminDashboard.jsx';

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/login" element={<Login />} />
        <Route path="/match-schedule" element={<MatchSchedule />} />
        <Route path="/register" element={<Register />} />
        <Route path="/admindashboard" element={<AdminDashboard />} />
      </Routes>
    </Router>
  );
}

export default App;
```

4.3 Key Components

Home: Main landing page.

Login: User authentication.

Register: User registration.

MatchSchedule: Displays upcoming matches.

AdminDashboard: Admin functionalities for managing matches.

Backend Design

5.1 Structure

The backend application is organized as follows:

controllers/: Handles incoming requests and business logic.

models/: Defines database schemas.

routes/: Maps endpoints to controllers.

middlewares/: Custom middleware functions.

5.2 API Endpoints

/api/login: Authenticates user.

/api/register: Registers new user.

/api/admin/addmatch: Adds a new match (Admin only).

/api/admin/getmatch: Retrieves match information.

/api/userauth: Authenticates users for protected routes.

Database Design

6.1 Schema

The MongoDB database includes the following collections:

users

<code>json

{

  "name": "string",

  "email": "string",

  "password": "string",

  "role": "string"

}

</code>

matches

<code>json

{

  "teams": ["string"],

  "date": "Date",

  "venue": "string",

  "status": "string"

}

</code>

6.2 Relationships

There are no direct foreign key relationships due to the NoSQL nature of MongoDB, but logical relationships exist:

Users and matches are linked by user roles (admin, user).

Third-Party API Integration

The system integrates with third-party APIs to fetch live scores. The backend periodically calls these APIs and updates the match status in the database.

Security

Security measures include:

Authentication: JWT for securing endpoints.

Authorization: Role-based access control.

Data Protection: Encryption of sensitive data like passwords.

Deployment

The application is deployed using the following processes:

Frontend: Deployed on a web server (e.g., Vercel, Netlify).

Backend: Deployed on a server (e.g., Heroku, AWS).

Database: Hosted on a cloud database service (e.g., MongoDB Atlas).

Conclusion

This document provides a comprehensive overview of the Cricket API system's high-level design. Future iterations will include more detailed designs and implementation specifics.

