# **BallByBall**

Design Document

**Overview:** BallByBall is a web application designed to simplify the process of tracking cricket scores online. It provides users with a user-friendly interface to create, update, and manage cricket scorecards. This design document outlines the key features, architecture, and user interface design of BallByBall.

**Target Audience:** The target audience for the "BallByBall" website includes cricket enthusiasts, players, scorers, umpires, and spectators involved in local cricket matches.

## **1. Features**

### Core Features:

* Users can create a new cricket scorecard by providing basic information such as match details, teams, and players.
* Users can update an existing scorecard by adding ball-by-ball information, including runs scored, wickets taken, and overs bowled.
* Users can delete a scorecard if it's no longer needed.
* Users can manage and update tournaments in this web app.

### Additional Features:

* User Authentication: This will allow users to create accounts and log in to access their scorecards.
* Audience can vote for their favorite player.
* Real-time Updates: Implement real-time updates to display live scorecards as the match progresses.
* Statistics: Generate statistics and insights from scorecards, including player performance, team trends, and match summaries.
* Live Chat: Users can communicate and give opinions about the ongoing match.

## **2. Architecture**

BallByBall follows a client-server architecture, with the frontend and backend components interacting via RESTful APIs.

### Frontend:

* Framework: React.js
* State Management: Redux
* Routing: React Router
* Styling: CSS and TailWind
* HTTP Requests: Axios

### Backend:

* Framework: Express.js
* Database: MongoDB (via Mongoose)
* Authentication: Passport.js
* API Documentation: Postman

### Deployment:

* Frontend: Hosted on a platform like Vercel
* Backend: Hosted on a platform like Heroku

## **3. User Interface Design**

### Home Page:

* Clean and simple interface which will allow users to go to different live matches, live tournaments or manage/create their own match scorecard.

### Match Page:

* Form-based interface for users to input match details, teams, players, and other relevant information.
* Table-based layout to display scorecard data, including runs scored, wickets taken, overs bowled, and other match details.

Tournament Page:

* This page will have the information about the specific tournament and user can see the different matches ongoing in the tournament.

User Page:

* In this page the User can update their own profile.

## **4. Security Considerations**

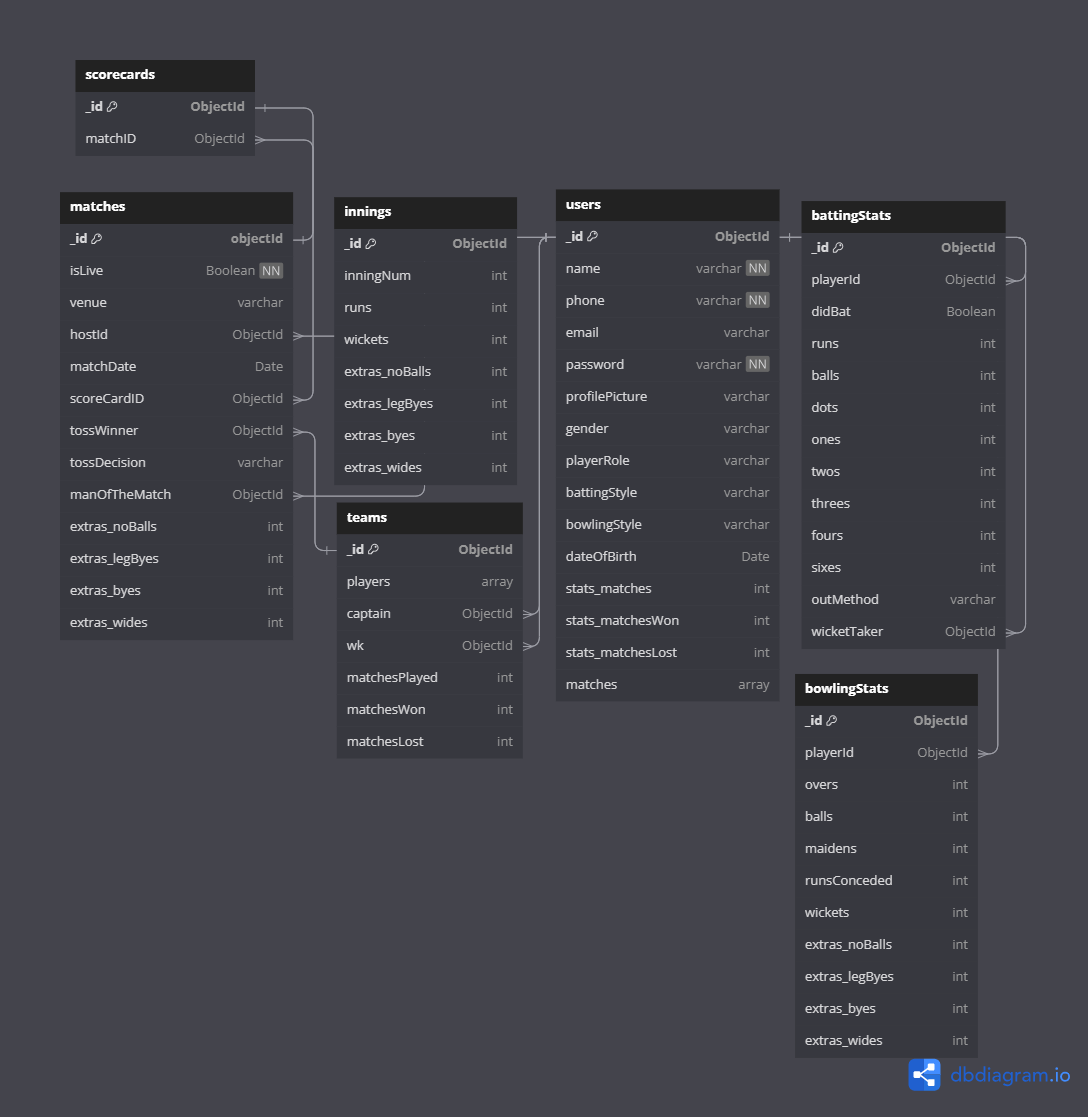
* Implement proper input validation and sanitization to prevent injection attacks.
* Use secure authentication mechanisms (if implemented) such as JWT tokens or OAuth.
* Ensure data privacy and protection by encrypting sensitive information and passwords stored in the database.

## **5. Testing Strategy**

* Conduct integration testing to ensure seamless interaction between frontend and backend.
* Test user flows and edge cases to identify and resolve any bugs or issues.

**6. Database Schema:**

* **Users:** 
  + \_id: objectId [primary key]
  + isLive: Boolean [not null, default: false]
  + venue: varchar
  + hostId: ObjectId [ref: > users.\_id]
  + matchDate: Date
  + scoreCardID: ObjectId [ref: > scorecards.\_id]
  + tossWinner: ObjectId [ref: > teams.\_id]
  + tossDecision: varchar
  + manOfTheMatch: ObjectId [ref: > users.\_id]
  + extras: array
* **Match:**
  + \_id: objectId [primary key]
  + isLive: Boolean [not null, default: false]
  + venue: varchar
  + hostId: ObjectId [ref: > users.\_id]
  + matchDate: Date
  + scoreCardID: ObjectId [ref: > scorecards.\_id]
  + tossWinner: ObjectId [ref: > teams.\_id]
  + tossDecision: varchar
  + manOfTheMatch: ObjectId [ref: > users.\_id]
  + extras: array
* **Scorecard:**
  + \_id: ObjectId [pk]
  + matchID: ObjectId [ref: > matches.\_id]
* **battingStats: Array of Objects**
  + \_id: ObjectId [pk]
  + playerId: ObjectId [ref: > users.\_id]
  + didBat: Boolean
  + Runs: int
  + Balls: int
  + Dots: int
  + Ones: int
  + Twos: int
  + Threes: int
  + Fours: int
  + Sixes: int
  + outMethod: varchar
  + wicketTaker: ObjectId [ref: > users.\_id]
* **bowlingStats: Array of Objects** 
  + \_id: ObjectId [pk]
  + playerId: ObjectId [ref: > users.\_id]
  + Overs: int
  + Balls: int
  + Maidens: int
  + runsConceded: int
  + Wickets: int
  + extras\_noBalls: int
  + extras\_legByes: int
  + extras\_byes: int
  + extras\_wides: int
* **Innings**:
  + \_id: ObjectId [pk]
  + inningNum: int
  + runs: int
  + wickets: int
  + extras: array
* **Teams:**
  + \_id: ObjectId [pk]
  + Players: array
  + Captain: ObjectId [ref: > users.\_id]
  + Wk: ObjectId [ref: > users.\_id]
  + matchesPlayed: int
  + matchesWon: int
  + matchesLost: int

****

## 

## **7. API EndPoints:**

* **Users:**
  + POST /api/users/register: Register a new user.
  + POST /api/users/login: Log in an existing user.
  + GET /api/users/:userId: Get user details.
  + PUT /api/users/:userId: Update user details.
  + DELETE /api/users/:userId: Delete user account
* **Matches**:
  + POST /api/matches: Create a new match.
  + GET /api/matches/:matchId: Get match details.
  + PUT /api/matches/:matchId: Update match details.
  + DELETE /api/matches/:matchId: Delete a match.
* **Scoresheets:**
  + POST /api/matches/:matchId/scoresheets: Create a new scoresheet for a match.
  + GET /api/matches/:matchId/scoresheets/:scoresheetId: Get scoresheet details.
  + PUT /api/matches/:matchId/scoresheets/:scoresheetId: Update scoresheet details.
  + DELETE /api/matches/:matchId/scoresheets/:scoresheetId: Delete a scoresheet.

## 

## **8. Technology Stack:**

## **9. Conclusion**

BallByBall aims to provide cricket enthusiasts with a seamless and enjoyable experience for tracking cricket scores online. By following the outlined design and implementation strategies, the application will deliver on its promise of simplicity, reliability, and efficiency.