## Metalworks Premier League (MPL) Project Documentation

**Version:** Based on code provided on July 31, 2024  
**Project Goal:** To create a web application for managing a Box Cricket league, including player registration, team management, scheduling, live scoring, and viewing results.

1. Architecture Overview

The project follows a standard client-server architecture:

* **Backend (mpl-backend):** A Node.js application built with Express.js. It serves as the API server, handling business logic, database interactions, authentication, and real-time updates via WebSockets.
* **Frontend (mpl-frontend):** A React application built with Vite. It provides the user interface for both public users (viewing schedules, players, live matches) and administrators (managing the league, scoring matches).
* **Database:** MySQL is used to store all persistent data (players, teams, matches, scores, etc.). The backend interacts with it using the mysql2 library.
* **Real-time Communication:** Socket.IO is used for pushing live score updates from the backend to connected frontend clients during a match.

**Data Flow:**

1. **Frontend UI** -> **API Request (Axios)** -> **Backend API Route** -> **Backend Controller** -> **Database** (for CRUD operations, fetching data) -> **Backend Controller** -> **API Response** -> **Frontend UI**
2. **Admin Scorer UI (Frontend)** -> **API Request (Score Ball)** -> **Backend Scoring Controller** -> **Database Updates (BallByBall, PlayerMatchStats, Matches)** -> **Backend Scoring Controller** -> **Socket.IO Emit (updateScore)** -> **Socket Server** -> **Connected Frontend Clients (Admin & Viewers)** -> **Frontend Socket Listener (handleUpdateScore)** -> **Frontend UI Update**

2. Backend (mpl-backend) Details

2.1. Setup Instructions

1. Navigate to the mpl-backend directory: cd mpl-backend
2. Install dependencies: npm install
3. Create a .env file in the mpl-backend root directory.
4. Copy the contents of mpl-backend/.env (provided in the code) into your new .env file.
5. **Crucially:** Update the MySQL connection details (DB\_HOST, DB\_USER, DB\_PASSWORD, DB\_NAME) in your .env file to match your local MySQL setup.
6. **Crucially:** Generate a strong, unique JWT\_SECRET and replace the placeholder in .env. You can use the command: node -e "console.log(require('crypto').randomBytes(64).toString('hex'))" in your terminal.
7. Ensure the FRONTEND\_URL in .env matches where your React frontend will run (default is http://localhost:5173).
8. Make sure your MySQL server is running and the database specified in DB\_NAME exists. You may need to create the database and the necessary tables manually (schema not provided, but can be inferred from controllers).
9. Start the backend server: npm run dev (for development with nodemon) or npm start (for production).

2.2. Core Technologies

* **Node.js:** JavaScript runtime environment.
* **Express.js:** Web application framework for creating the API.
* **mysql2:** MySQL database driver for Node.js (with Promise support).
* **Socket.IO:** Real-time bidirectional event-based communication library.
* **bcrypt:** Library for hashing passwords securely.
* **jsonwebtoken (JWT):** For creating and verifying access tokens for admin authentication.
* **cors:** Middleware to enable Cross-Origin Resource Sharing.
* **dotenv:** Loads environment variables from a .env file.
* **nodemon:** Utility to automatically restart the server during development.

2.3. Folder Structure

* config/: Contains database configuration (db.js).
* controllers/: Business logic for handling API requests. Separated into public and admin/ subfolders.
* middleware/: Custom middleware functions (e.g., authMiddleware.js for protecting routes).
* node\_modules/: Project dependencies (managed by npm).
* routes/: Defines API endpoints and maps them to controller functions. Separated into public and admin/ subfolders.
* socket/: Contains Socket.IO event handling logic (socketHandler.js).
* .env: Stores environment variables (DB credentials, JWT secret, etc.). **(Not version controlled)**
* package.json / package-lock.json: Define project metadata and dependencies.
* server.js: The main entry point for the backend application. Initializes Express, Socket.IO, middleware, routes, and starts the server.

2.4. Key Files & Modules

* **server.js**:
  + Initializes Express app and HTTP server.
  + Initializes Socket.IO server and attaches it to the HTTP server, configuring CORS.
  + Applies global middleware (CORS, express.json, express.urlencoded).
  + Mounts all public and admin API routes, applying the protect middleware to admin routes.
  + Calls initializeSocket to set up Socket.IO event listeners.
  + Sets up global 404 and error handling middleware.
  + Starts the HTTP server.
  + Includes graceful shutdown logic.
* **config/db.js**:
  + Creates and exports the MySQL connection pool using credentials from .env.
  + Includes basic connection testing on startup.
* **middleware/authMiddleware.js**:
  + Exports the protect function.
  + Verifies the JWT token provided in the Authorization: Bearer header.
  + Fetches the admin user associated with the token from the database.
  + Attaches the admin user object (req.admin) to the request if valid.
  + Returns 401 Unauthorized errors if the token is missing, invalid, or expired.
* **controllers/**: Contain functions handling specific API requests (e.g., playerController.js, matchController.js, admin/scoringController.js). They interact with the database pool and format responses.
* **routes/**: Define the API structure. Each file uses express.Router() and maps HTTP methods/paths to controller functions.
* **socket/socketHandler.js**:
  + Initialized by server.js, receives the io instance.
  + Handles new socket connections (io.on('connection', ...)).
  + Manages clients joining/leaving specific match rooms (socket.on('joinMatchRoom', ...)).
  + **Important:** It *does not* handle the core scoring logic itself anymore. It primarily listens for connections and manages rooms. The actual updateScore events are emitted *from the backend controllers* (scoringController.js) after database operations.
  + Maintains a basic in-memory cache (liveMatchesState) to send current state to users joining a room mid-match. (Note: This is not production-ready for scalability).

2.5. API Endpoints

**(Access: Public / Admin Protected)**

* **Health Check:**
  + GET /api: Public - Checks if the API is running.
* **Players:**
  + GET /api/players: Public - Get list of players (ID, name, role).
  + POST /api/players: Public (as configured) - Register a new player.
  + GET /api/players/:id: Public - Get details for a single player.
  + GET /api/players/:id/stats: Public - Get aggregated stats for a player (career or by season\_id).
  + PUT /api/players/:id: Admin Protected - Update player details.
  + DELETE /api/players/:id: Admin Protected - Delete a player.
* **Matches (Public View):**
  + GET /api/matches: Public - Get match fixtures (filterable by season, status, team).
  + GET /api/matches/:id: Public - Get full details for a single match (including final player stats if completed).
  + GET /api/matches/:id/commentary: Public - Get ball-by-ball commentary for a match.
* **Ratings:**
  + POST /api/ratings: Protected (Auth TBD) - Submit a player rating. *(Currently returns 501 Not Implemented)*
  + GET /api/ratings/player/:playerId: Public - Get all ratings received by a player (filterable by season).
  + GET /api/ratings/player/:playerId/average: Public - Get average rating for a player (filterable by season).
* **Admin Authentication:**
  + POST /api/admin/auth/login: Public - Authenticate admin and get JWT token.
  + POST /api/admin/auth/register: Public (as configured, **SECURITY RISK**) - Register a new admin.
* **Admin Seasons:**
  + GET /api/admin/seasons: Admin Protected - Get list of all seasons.
  + POST /api/admin/seasons: Admin Protected - Create a new season.
  + GET /api/admin/seasons/:id: Admin Protected - Get details of a single season.
  + PUT /api/admin/seasons/:id: Admin Protected - Update season details.
  + DELETE /api/admin/seasons/:id: Admin Protected - Delete a season (Not implemented in controller).
* **Admin Teams:**
  + GET /api/admin/teams: Admin Protected - Get teams (requires ?season\_id=X query).
  + POST /api/admin/teams: Admin Protected - Add a new team to a season.
  + GET /api/admin/teams/:id: Admin Protected - Get team details including players (requires ?season\_id=X query).
  + PUT /api/admin/teams/:id: Admin Protected - Update team details (name, budget, captain).
  + DELETE /api/admin/teams/:id: Admin Protected - Delete a team (Not implemented in controller).
  + POST /api/admin/teams/players: Admin Protected - Assign a player to a team for a season.
  + DELETE /api/admin/teams/players/:teamPlayerId: Admin Protected - Remove a player assignment from a team.
* **Admin Matches (Schedule):**
  + GET /api/admin/matches: Admin Protected - Get match list (filterable).
  + POST /api/admin/matches: Admin Protected - Create a new match fixture.
  + GET /api/admin/matches/:id: Admin Protected - Get details of a single match for admin view/editing.
  + PUT /api/admin/matches/:id: Admin Protected - Update match details (only certain fields/statuses allowed).
  + DELETE /api/admin/matches/:id: Admin Protected - Delete a 'Scheduled' match.
* **Admin Scoring:**
  + GET /api/admin/scoring/setup-list: Admin Protected - Get 'Scheduled' matches ready for setup.
  + POST /api/admin/scoring/matches/:matchId/setup: Admin Protected - Submit toss/decision/super over, updates status to 'Setup'.
  + GET /api/admin/scoring/matches/:matchId/state: Admin Protected - Get the current detailed state of a match (for resuming).
  + POST /api/admin/scoring/matches/:matchId/ball: Admin Protected - Record a single ball event, updates DB and emits socket update.
  + DELETE /api/admin/scoring/matches/:matchId/ball/last: Admin Protected - Undo the last recorded ball event.
  + POST /api/admin/scoring/matches/:matchId/finalize: Admin Protected - Manually finalize match score and stats (optional).

2.6. Socket.IO Events

* **Events Emitted by Backend (to Clients in match\_X room):**
  + updateScore (object): Sent after every successful scoreSingleBall or undoLastBall API call. Contains the complete, updated state of the match (fullLiveState).
  + inningsBreak (object): Sent when inning 1 ends (contains the state).
  + matchEnded (object): Sent when the match completes (contains the final state/result).
  + scoringError (object): Emitted if an error occurs during scoreSingleBall or undoLastBall processing on the backend.
  + matchLive (object): [Can potentially be emitted from submitMatchSetup or first scoreBall to signal start]. *(Current code emits from frontend startMatchScoringAction which might be redundant)*
* **Events Handled by Backend (from Clients):**
  + connection: Built-in event when a client connects.
  + disconnect: Built-in event when a client disconnects.
  + connect\_error: Built-in event on connection failure.
  + joinMatchRoom (matchId): Client requests to join a specific match room.
  + leaveMatchRoom (matchId): Client requests to leave a specific match room.
  + startMatchScoring (matchId, initialState): [Currently handled in frontend context, might not be needed if setup API response triggers navigation]

2.7. Environment Variables (.env)

* DB\_HOST: MySQL host address.
* DB\_USER: MySQL username.
* DB\_PASSWORD: MySQL password.
* DB\_NAME: MySQL database name.
* PORT: Port number for the backend server (default 5000).
* JWT\_SECRET: **Crucial** secret key for signing/verifying admin tokens.
* FRONTEND\_URL: URL of the React frontend (for CORS and Socket.IO).
* NODE\_ENV: Environment mode ('development' or 'production').

3. Frontend (mpl-frontend) Details

3.1. Setup Instructions

1. Navigate to the mpl-frontend directory: cd mpl-frontend
2. Install dependencies: npm install
3. *(Optional)* Create a .env file in the mpl-frontend root if you need to override default API/Socket URLs.
   * VITE\_API\_URL=http://localhost:5000/api (Base URL for backend API)
   * VITE\_SOCKET\_URL=http://localhost:5000 (URL for backend Socket.IO server)
4. Start the frontend development server: npm run dev. It will typically run on http://localhost:5173.

3.2. Core Technologies

* **React:** JavaScript library for building user interfaces.
* **Vite:** Fast frontend build tool and development server.
* **react-router-dom:** Library for handling routing within the React application.
* **axios:** Promise-based HTTP client for making API requests to the backend.
* **socket.io-client:** Client library for connecting to the Socket.IO backend.

3.3. Folder Structure

* public/: Static assets.
* src/: Contains the React application source code.
  + assets/: Image/SVG files.
  + components/: Reusable UI components (Navbar, PrivateRoute, LoadingFallback).
  + context/: React Context providers (SocketContext.jsx).
  + pages/: Top-level components representing application pages/routes (public and admin/).
  + services/: API interaction logic (api.js).
  + App.jsx: Main application component defining routes.
  + main.jsx: Entry point, renders the App component.
  + index.css: Global CSS styles.
* dist/: Build output directory (generated by npm run build).
* eslint.config.js: ESLint configuration.
* index.html: Main HTML file template used by Vite.
* package.json / package-lock.json: Project metadata and dependencies.
* vite.config.js: Vite configuration file.

3.4. Key Components & Files

* **main.jsx**: Sets up React StrictMode and renders the <App /> component wrapped in <SocketProvider>.
* **App.jsx**: Defines the application's routing structure using react-router-dom. Uses Suspense and lazy for code-splitting pages. Wraps admin routes with the <PrivateRoute /> component. Includes the <Navbar />.
* **services/api.js**: Configures an Axios instance with the backend base URL. Includes interceptors to:
  + Automatically attach the JWT token (from localStorage) to outgoing requests.
  + Handle common response errors globally (e.g., 401 Unauthorized redirects to login, extracts error messages).
* **context/SocketContext.jsx**:
  + Creates and manages the Socket.IO client connection.
  + Provides the socket instance (socket) and connection status (isConnected) via the useSocket hook.
  + Includes memoized helper functions (joinMatchRoom, leaveMatchRoom, etc.) for common socket actions.
  + Handles basic socket events (connect, disconnect, connect\_error).
* **components/PrivateRoute.jsx**:
  + A route wrapper component.
  + Checks for the presence and basic validity of the admin token in localStorage.
  + If authenticated, renders the nested child routes (<Outlet />).
  + If not authenticated, redirects the user to /admin/login, saving the intended destination.
* **components/Navbar.jsx**:
  + Displays navigation links.
  + Uses NavLink for active route styling.
  + Dynamically shows "Admin Dashboard" / "Logout" or "Admin Login" based on the presence of the admin token in localStorage.
  + Handles logout by removing the token and navigating to login.
* **pages/**: Each file represents a distinct view/page of the application, fetching data via the api service and managing component-level state.
  + **MatchDetailPage.jsx**: Fetches initial match details via API, then uses the SocketContext to join the match room and listen for updateScore events to display live data.
  + **AdminLiveScoringPage.jsx**: Fetches the current match state via API on load/refresh. Uses SocketContext to join the room and receive updateScore events. Contains the UI for selecting players and triggering scoring actions (which call backend API endpoints), and an Undo button.

3.5. Routing

* Managed by react-router-dom within App.jsx.
* Public routes include /, /players, /players/:id, /schedule, /matches/:matchId.
* Admin login is at /admin/login.
* All other routes under /admin/\* are protected by <PrivateRoute />.

3.6. State Management

* Primarily uses React's built-in hooks (useState, useEffect, useCallback, useMemo, useRef).
* SocketContext manages the global socket connection state.
* Admin authentication state is persisted in localStorage.

3.7. Socket.IO Usage

* The SocketProvider establishes and manages the connection.
* The useSocket hook provides access to the socket instance and helper functions.
* MatchDetailPage and AdminLiveScoringPage use joinMatchRoom and leaveMatchRoom.
* These pages listen for backend events (updateScore, inningsBreak, matchEnded, scoringError) to update their local state and re-render the UI.
* **Crucially:** Scoring actions on AdminLiveScoringPage *do not* emit socket events directly. They call backend API endpoints, and the backend emits the updates after processing.

4. Database Schema Overview (Inferred)

Based on the controllers and API calls, the database likely contains tables similar to these (actual schema definition not provided):

* **Admins**: Stores admin login credentials (admin\_id, username, email, password\_hash).
* **Seasons**: Stores league season information (season\_id, year, name, start\_date, end\_date, status).
* **Players**: Stores player details (player\_id, name, email, phone, role, base\_price).
* **Teams**: Stores team information for each season (team\_id, season\_id, name, budget, captain\_player\_id FK to Players).
* **TeamPlayers**: Junction table linking players to teams for a specific season (team\_player\_id, team\_id, player\_id, season\_id, purchase\_price, is\_captain).
* **Matches**: Stores match schedule, setup, and result information (match\_id, season\_id, team1\_id, team2\_id, match\_datetime, venue, status, toss\_winner\_team\_id, decision, super\_over\_number, winner\_team\_id, result\_summary, man\_of\_the\_match\_player\_id).
* **PlayerMatchStats**: Stores detailed statistics for each player in each match (stat\_id, match\_id, player\_id, team\_id, runs\_scored, balls\_faced, fours, sixes, is\_out, how\_out, wickets\_taken, runs\_conceded, overs\_bowled, maidens, wides, no\_balls, catches, stumps, run\_outs).
* **BallByBall**: Stores details for every ball bowled in a match (ball\_id, match\_id, inning\_number, over\_number, ball\_number\_in\_over, bowler\_player\_id, batsman\_on\_strike\_player\_id, runs\_scored, is\_bye, is\_extra, extra\_type, extra\_runs, is\_wicket, wicket\_type, fielder\_player\_id, commentary\_text).
* **PlayerRatings**: Stores player ratings given by others (rating\_id, season\_id, rated\_player\_id, rater\_player\_id, rating\_value, comment, rated\_at).

5. Key Features & Workflows

* **Admin Login:** Admins log in via /admin/login, receiving a JWT token stored in localStorage.
* **Admin Management:** Admins can create/update seasons, add/update teams within seasons, assign/remove players from teams.
* **Scheduling:** Admins can schedule new matches between teams within a season.
* **Match Setup:** Before scoring, admins select a 'Scheduled' match, input toss results, decision, and super over number via the /admin/scoring/setup page. This updates the match status to 'Setup'.
* **Live Scoring:** Admins navigate to the /admin/scoring/live/:matchId page.
  + The page fetches the current state via API.
  + Admins select the opening/current bowler and batsman.
  + Admins click buttons corresponding to ball events (runs, extras, wickets).
  + Each click triggers an API call to the backend /ball endpoint.
  + The backend processes the ball, updates the database, calculates the new state, and emits an updateScore event via Socket.IO.
  + The admin page (and any public viewers on MatchDetailPage) receive the updateScore event and update their UI.
  + Admins can undo the last ball via an API call.
* **Public Viewing:** Users can view player lists, schedules/results, and detailed match pages (including live scores received via Socket.IO).

6. Potential Issues & Areas for Improvement (TODOs)

* **Socket State Persistence:** The backend liveMatchesState cache is in-memory and will be lost on server restart. For production, use a persistent store like Redis if you need to maintain live state independently of the database for performance or immediate joins. (Currently, the frontend relies on API fetch for initial state, which is more resilient).
* **Error Handling:** While the API interceptor handles some errors, more granular error handling and user feedback can be added on both frontend and backend.
* **Security:**
  + Admin registration endpoint (/api/admin/auth/register) should likely be removed or heavily protected, not left public.
  + Implement more input validation on backend controllers (e.g., checking data types, ranges, string lengths).
  + Consider rate limiting API endpoints.
* **Testing:** No automated tests (unit, integration, e2e) are present. Adding tests is crucial for maintainability.
* **Database Schema:** The exact schema needs definition and creation scripts. Ensure appropriate indexing for performance. Consider ON DELETE constraints carefully (e.g., CASCADE vs. SET NULL vs. RESTRICT).
* **UI/UX:**
  + Loading states can be more granular.
  + User feedback on actions (e.g., success/error toasts).
  + Replace Player ID inputs with searchable dropdowns/autocompletes.
  + Improve ScoreDisplay team name resolution.
* **Scalability:**
  + Database queries (especially aggregations in getPlayerStats) might need optimization for large datasets.
  + Socket.IO server might need horizontal scaling considerations for many concurrent matches/viewers (requires adapter like Redis adapter).
* **Missing Features:** Player self-registration flow, payments, detailed auction logic, user roles beyond admin/public.
* **Frontend Auth Check:** The PrivateRoute currently only checks for token existence. It could decode the token to check expiry for better security.

7. Getting Started Summary

1. Clone the repository.
2. Set up your MySQL database and create the required schema/tables.
3. **Backend:**
   * cd mpl-backend
   * npm install
   * Create and configure .env (DB details, **new JWT\_SECRET**, Frontend URL).
   * npm run dev
4. **Frontend:**
   * cd mpl-frontend
   * npm install
   * *(Optional)* Create .env to set VITE\_API\_URL and VITE\_SOCKET\_URL if different from defaults.
   * npm run dev
5. Access the frontend (usually http://localhost:5173).
6. *(If needed)* Register an admin user (consider using a script or temporarily enabling the /register route securely).
7. Log in as admin and start managing the league.