Internet programming project

Theme: Local football championship

Team:28

Submitted by : Lydia Georganta(A.M.:1095484)

Ana-Maria Băncilă (A.M.: 1119887)

Date:28/5/2025

Introduction:

The “**Fantasyland League Web Application”** is a comprehensive, full-stack web application designed to deliver an engaging and interactive experience for football enthusiasts. The platform features real-time updates on league standings, match results, player and coach profiles, and statistical visualizations. Additionally, it includes a secure administrative interface for managing data dynamically.

The primary objective of this project is to simulate a real-world sports league management system by integrating **frontend**, **backend**, and **database** technologies. This report outlines the development process, technologies used, key features, and overall evaluation of the project.

Project objectives:

The objectives of the “Fantasyland League” project are the following:

1. Develop a dynamic and user-friendly platform for viewing league information.
2. Implement robust backend logic to serve dynamic content to users.
3. Ensure data security and integrity through proper authentication and authorization mechanisms.
4. Provide an administrative interface for data management
5. Utilize modern web development technologies to create a responsive and maintainable application.

Creation of the project:

ERD

At the beginning of the project, before starting the implementation, we created an '**Entity Relationship Diagram' (ERD)** to model the system's data structures and relationships accurately. This process was crucial in identifying the entities involved in the league, their attributes, and how they connect with one another. Our ERD includes the following key entities, the 'Team', the 'Player', the 'Coach', the 'Match', the 'Goal' and the 'Admin'. Each team has a key attribute named 'team_id' and the other attributes include the 'team_name', the 'number_of_players' and the 'home_field'. This entity is connected to 'Player' with an 1-to-many relationship (has) as well as to 'Coach'(has) and 'Match'(plays). The 'Team' is involved in matches either as home or guest team. The entity 'Player' has as key attribute the 'player_id' and the other attributes consist of 'name', 'lastname', 'age', 'active_years', and 'biography'. The player is connected to 'Match' and 'Coach' with the relationship 'participate' (many-to-many), because players and coaches can participate in multiple matches. The entity 'Coach' has as key attribute the 'c_id' and its other attributes are the 'biography', the 'c_lastname', 'c_name', 'age', 'nationality'. The entity match contains match-specific data like the date , the time, the location,the result and the potential delay, its key attribute is 'm_id'. The entity 'Goal' records in-match events like the type of the goal, the minute it was scored and has its own id. The 'Goal' is connected to the 'Team' and the 'Match' via the relationship 'plays' and with the Player via 'scored'. The entity 'Admin stores admin details for system access and ensures data updates are controlled and secure.

SCHEMA:

This ERD was used as a prototype in order to create a relational schema. The relationships 'plays' and 'participate' were translated as individual tables named 'playing_match' and 'participation' respectively. The first table models a specific instance of a match between two teams and the second table tracks the involment of players and coaches in each match.

The erd and the schema ensure the data integrity, the efficient queries , the scalability and the security of our project. They minimize data redundancy and ensure accurate relationships as well as they support advanced queries like league standings through well-defined joins. Furthermore the admin data is isolated from general user data for controlled data management.

DATABASE:

Based on the erd model and the schema we created the database for the local championship using the SQLite. Our database -project2.db- serves as the backbone for our entire web application because it stores data for teams, players, matches, coaches and admins. The 'playing_match' table is used as an associative entity, linking home and guest teams in a match and recording scores. The 'participation' table is a junction table that manages many-to-many relationships, tracks player and coach involvement in matches. All tables have primary keys for unique identification and the foreign keys link related entities. The 'team_rankings' SQL view dynamically calculates standings—using SQL's GROUP BY and JOIN features—ensuring live data for the league table.

Fronted Implementation:

The frontend includes multiple HTML pages such as index.html, table.html, resultsFixtures.html, teamSearch.html, teams.html, statistics.html, and profile pages for

players and coaches. Each page is designed for a clear and intuitive user experience, with **CSS** ensuring consistent and responsive styling.

Dynamic content is handled through ‘JavaScript’ files (e.g., teams_views.js, players.js, matches.js), which use the ‘Fetch API’ to retrieve data asynchronously from the backend APIs.

The index.html creates the welcoming landing page that introduces the platform. Some key features include the Navbar with links to major sections such as Table,Result/Fixtures,Teams and Statistics. We also have implemented a Hero section with a welcoming message and we have dynamic list of **top** scorers using an AJAX fetch to /api/statistics.

The table.html displays the current league table standings. It ranks teams by points, with color coding for top 4 and bottom 3, its rows are dynamically generated and updated based on the latest standings /api/rankings. The teams displayed are linkes to their respective teams.html profile.

The resultFixtures.html display upcoming and past match fixtures. The Date picker allows users to view matches for specific dates, and if they select the first day of each month they can see all the scheduled matches of the month. It fetches matches data based on the selected date (/api/matches?date).

The teamSearch.html allows users to search for and view teams dynamically. The search input filters team logos dynamically as users type and each team logo is linked to a detailed page-teams.html-using the team name as a query parameter. On the dynamic aspects the page loads all teams from the /api/teams endpoint and the live search via input events hides or shows logos accordingly .When the user presses the enter is directly navigated to the team’s page.

The teams.html displays detailed information for a specific team. It uses the URL query parameters to identify the team and them shows the team name, logo, home field, and number of players. It fetches team data, players, and coaches from respective API endpoints lincing to each player and coaching staff profile.

The player.html displays detailed information about a specific player. It fetches player data based on URL parameters (player ID) and provides fallback text if data is missing or there’s an error. This is also the way the coach.html works about the coaching staff.

The statistics.html visualizes the league’s statistics with interactive charts. The buttons allow users to view different statistics like top scorers , best defense and average goals. It uses Chart.js

The style.css unifies the stylesheet for the entire application. It provides a consistent styling for navbar, tables,cards and sections, responsive adjustments for smaller screens and clear visual distinctions.

Overall each page uses the Fetch API to retrieve data from backend API endpoints (/api/teams, /api/players, /api/rankings, etc.) and the asynchronous requests allow data to be loaded and updated in real-time without page reloads. As far as the DOM Manipulation is

concerned JavaScript dynamically updates the HTML content based on fetched data, ensuring a fluid and interactive experience. For example, in table.html, the league table rows are dynamically generated and updated based on the latest standings.

Backend Routing Overview

In our Node.js/Express.js backend, the routes folder contains the core JavaScript files that handle API endpoints and data retrieval. These files work together to fetch data from the database (db.sqlite) and send JSON responses to the frontend for dynamic rendering. Using the express.Router() each file exports a router instance to keep routes modular and organized. All data endpoints are prefixed with /api/, making it easy to distinguish between API routes and static file routes. The routes include try-catch blocks or error handling middleware to manage unexpected database errors or invalid queries. The database operations use async/await or Promises for non-blocking behavior ensuring quick an responsive APIs.

The app.js is the main server file for Node,js and Express, it acts as the central hub for routing. It uses app.use('/api/players', require('./routes/players')) to mount each route module under its appropriate prefix.It also serves the public folder for static files (index.html, CSS, images, etc.) and configures session handling and argon2-based hashing for admin security.

The backend routing structure demonstrates a well-organized and modular approach to serving data in the Fantasyland League app. Each route file focuses on a specific area of the system—players, matches, teams—allowing for a robust and maintainable backend architecture.

This project follows a clear MVC-style architecture that was implemented to keep our codebase organized and maintainable:

- Models / Database Layer – SQLite tables (Team, Player, Coach, Match, Playing_Match) were used to support all our features. SQL queries were written directly using the sqlite wrapper since it gave us more control over the database operations.
- Controllers – teamController.mjs and matchController.mjs were created to handle all the business logic through REST endpoints that perform CRUD operations and JOINs. This way, our front-end never has to deal with raw SQL queries.
- Routes – Our teams.js file acts as a lightweight Express router that connects the controller logic under the /api/teams namespace. A detail sub-route was also added to handle specific team information.
- Views – Two different rendering approaches were implemented to meet our project requirements:
    - Server-side Handlebars (.hbs) for our public-facing pages and authentication screens, which was configured in app.mjs.
    - Static admin dashboard (plain HTML/JS in public/) for our data management interface, including features like Edit Teams and Edit Matches.
- Authentication – Admin registration and login pages were built.

  Our application workflow operates as follows:

    - HTTP requests come into app.mjs and get routed to the appropriate /api endpoints.
    - Controllers handle the database operations and return JSON responses.
    - Our admin single-page application fetches this JSON data and updates the DOM using vanilla JavaScript.
    - Meanwhile, Handlebars takes care of rendering our public pages through server-side rendering.

This modular approach was chosen because it makes our codebase much easier to understand, test, and expand. If new features like referee management or stadium information needed to be added, this architecture could easily be extended without breaking existing functionality.

Admin's side - File-by-file Walkthrough

app.mjs – Application Entry Point

```
import express from 'express';

import { engine } from 'express-handlebars';

…

app.engine('hbs', engine({ extname: '.hbs', defaultLayout: 'main' }));

app.use('/api', matchController); // mounts /matches endpoints
```

- **Bootstraps Express** and configures Handlebars with main.hbs as the site-wide shell.
- **Static middleware** exposes /public so HTML admin tools load assets directly.
- **Auth routes**: GET /admin/login | register render the corresponding .hbs templates. POST variants delegate to adminController.mjs.
- **Team routes**: wires CRUD handlers from teamController.mjs and the separate teams.js router.
- Starts server on PORT 3001.

teamController.mjs – Team CRUD

```
export async function addTeam(req,res){

  const { team_name, number_of_players, home_field } = req.body;

  await db.run(`INSERT INTO Team (team_name,number_of_players,home_field)

            VALUES (?,?,?)`,[team_name,number_of_players,home_field]);

}
```

- Opens **SQLite** once (top-level await open…).
- Exposes four handlers: getTeams, addTeam, removeTeam, updateTeamName – each returns JSON or status text so the UI can alert success/failure.
- Performs **input validation** (checks required fields, uniqueness implied by DB constraint) and returns proper *404* when records are missing.

matchController.mjs – Match CRUD

```
router.get('/matches', async (_,res)=>{

  const rows = await db.all(`SELECT M.match_id,…,T2.team_name AS team2_name

                        FROM Match M

                        JOIN Playing_Match PM ON M.match_id = PM.playing_match_id

                        … ORDER BY M.date DESC`);

  res.json(rows);

});
```

- Self-contained Express **sub-router**. Exported default and mounted by app.mjs under /api.
- Handles:
  - **GET /matches** → list (JOIN adds human-readable team names).
  - **POST /matches** → create *Match* row **and** corresponding *Playing_Match* linking home/guest teams.
  - **PUT /matches/********:id** → update delay/result.
- Demonstrates parameterised SQL to avoid injections.

admin-dashboard.html

- Pure UI: two-pane layout (sidebar + main). Links to the three CRUD pages and a logout.
- Responsive CSS via media query (max-width: 768px).

edit_teams.html

- Fetches **/api/teams** on load and renders *cards* per team.
- **Add team** form POSTs JSON to /api/teams.
- **Edit** overlay pre-fills current values, updates via /api/teams/update.
- **Delete** triggers /api/teams/remove with confirmation.

edit_matches.html

- Two async loaders:
  - loadTeams() to populate <select>s for new-match creation.
  - loadMatches() to populate a table with existing matches.
- Emits POST /api/matches for create and PUT /api/matches/:id for edits.

- Simple modal (#editModal) avoids page reloads for inline editing.

admin-login.hbs, admin-register.hbs, main.hbs

- Not reproduced here, but they supply **Handlebars** markup for auth forms and the site layout (nav bar referencing *Table*, *Results/Fixtures*, *Teams*, *Statistics*).

Timeline

Mid-March:
The initial requirements analysis was conducted and discussions were held about the application's features.

End of March:
The initial Entity Relationship Diagram (ERD) was created, along with the first user interface mockups using Figma.

Early April:
The first stage of the application was presented during an intermediate presentation. After this presentation, the ERD was revised and the database schema was created.

Easter Holidays:
The database was implemented, referential integrity was ensured, and constraints were created. Additionally, some test data was added for testing purposes.

End of April - May:
The frontend was developed using Handlebars.
The routes and basic functionalities were implemented.
Dynamic elements were integrated, and the backend was set up.
The backend was connected to the database.
Authentication functionality was implemented, and password hashing was applied.

Who did what:

Lydia Georganta: Together with Ana-Maria we created the erd model and the figma pages for our project. I additionally created the schema , the database based on the schema with the referential integrities and the constraints, collected the data and the pictures used at our site. In the frontend I created the admin-dashboard.html, coach.html,index.html,player.html,resultsFixture.html,statistics.html,my parts at style.css, the table.html,the teams.html, the teamSearch.html .I actively participated in the creation of sessions and the authentication process .In the backend I created the routes : coach.js, matches.js, players.js, rankings.js, statistics.js, team_views.js, teams.js as well as the app.js .

Ana-Maria: In a joint effort with Lydia, I have designed and implemented the overall flow of the website, focusing on both the aesthetic and functional components. The key features developed include admin authentication: login and registration logic (handled in app.mjs and

adminController.mjs), cosmetic changes in the views (admin-login.hbs and admin-register.hbs - the logic for these files has been created by Lydia) and administrative functionalities: managing teams (adding, editing, and removing) through teamsController.mjs and edit_teams.html and additional admin operations for players and specific match details.

This structured approach ensured a seamless design and functionality across admin and management interfaces.

Appendix : Project Installation Guide

This section provides step-by-step instructions for setting up and running the Fantasyland League Web Application on a local development environment.

1. Prerequisites

   a. Ensure you have Node.js (v16+) and npm installed on your machine. You can download them from: https://nodejs.org/

   b. This project uses SQLite as the database engine.Download it from: https://www.sqlite.org/download.html .Alternatively, use a GUI tool like DB Browser for SQLite for easy database management.

2. Clone the project

   Open your terminal and run the following command to clone the repository: git clone [REPOSITORY_URL]. Replace [REPOSITORY_URL] with the actual URL GitHub repository.

3. Navigate to the Project Directory:

Write the command: cd [project-folder-name]

4. Install Dependencies:

Install the required Node.js packages using npm: npm install

This will install dependencies defined in the package.json file, including:

express (for the web server),express-session (for admin authentication),argon2 (for password hashing),sqlite3 (for database operations),dotenv (for environment variable management).

5. Configure Environment Variables:

If you want to use environment variables, create a .env file in the root directory:
PORT=3000

SESSION_SECRET=your_secret_key

6. Set Up the Database

7. Run the Application :

In the terminal, run : node app.js or use nodemon for hot reloading (if installed globally): nodemon app.js. The server will start on: http://localhost:3000

Github link:

https://github.com/lydiageo7/fantasy_league