**Full Stack Sport and Fixture Management Application**

**Final Project Report**

**1. Introduction**

This report outlines the development and implementation of the Sport and Fixture Management Application, created as a final project for the course BLM19417E Web Design and Programming. The primary goal of the project was to design a web-based application using React for the frontend, ASP.NET Core for the backend Web API, and SQL Server as the database. The application enables users to manage sports teams, players, and fixtures, providing full CRUD (Create, Read, Update, Delete) functionality.

Technologies Used

* Frontend: React (with React Hooks, Node.js, Next.js)
* Backend: ASP.NET Core Web API
* Database: SQL Server
* State Management: Redux
* Routing: Next.js built-in routing system

**2. Data Model**

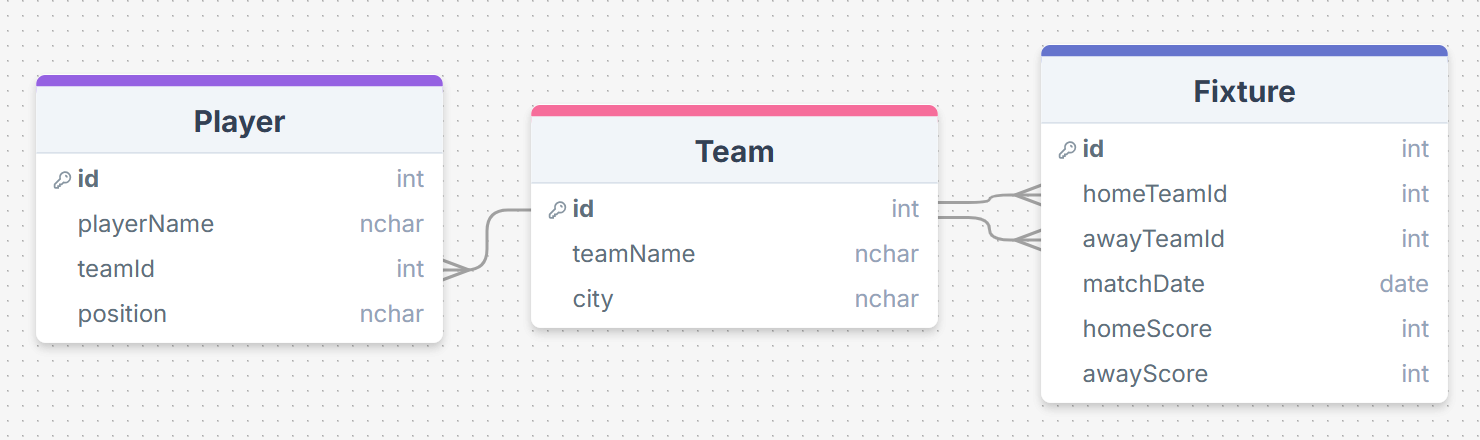
The database model consists of three primary tables:

1. Teams: Stores information about teams, including their names and cities.
   * Fields: Id, TeamName, City
2. Players: Stores player details associated with specific teams.
   * Fields: Id, FullName, Position, TeamId
   * Foreign Key: TeamId references Teams(Id)
3. Fixtures: Stores fixture details including the participating teams and scores.
   * Fields: Id, HomeTeamId, AwayTeamId, HomeTeamScore, AwayTeamScore, MatchDate
   * Foreign Keys: HomeTeamId and AwayTeamId both reference Teams(Id)

The relationships are as follows:

* A Team can have multiple Players.
* A Fixture involves two Teams (Home and Away).

Database Model Diagram



**3. Application Features**

**CRUD Operation Support**

The application supports full CRUD operations for Teams, Players, and Fixtures through a Web API.

* Create: New teams, players, and fixtures can be added using dedicated form pages.
* Read: Data is displayed in HTML tables on respective pages for teams, players, and fixtures.
* Update: Existing records can be edited through "Edit" pages.
* Delete: Records can be deleted after confirmation, with error handling for dependent data.

**Data Entry Form (Local State and Props)**

The "Add Team" and "Add Player" pages utilize local state (useState) and props to manage form data and handle user input.

**Lookup Field (Combo Box)**

On the "Add Fixture" page, a combo box is used to select teams by displaying their names while returning their IDs.

**Sum Calculation and Display**

On the Team Details page, the application calculates and displays the total number of matches played, wins, losses, draws, and not-played fixtures for the selected team.

**Status Message Using Redux**

Status messages (e.g., "Saved", "Error", "Not Changed") are displayed using Redux. Messages automatically disappear after a few seconds.

**Simple Validations**

Basic validations are implemented to ensure that required fields are filled before form submission.

**Menu and Routing**

A navigation menu is provided at the top of the application, enabling users to switch between the Teams, Players, and Fixtures pages. Routing is handled using Next.js's built-in routing mechanism.

**Web Service Call and Context Storage**

The application makes web service calls to fetch data from the backend and stores the data in Redux, which acts as a centralized context.

**React Hooks Used**

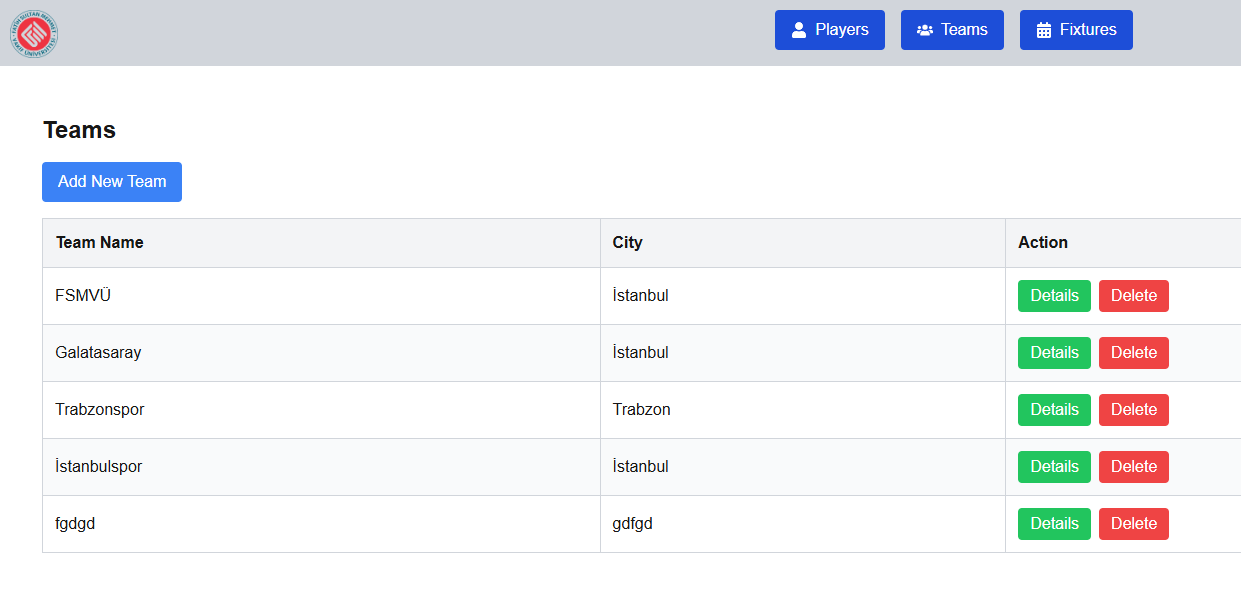
The following React Hooks are used throughout the application:

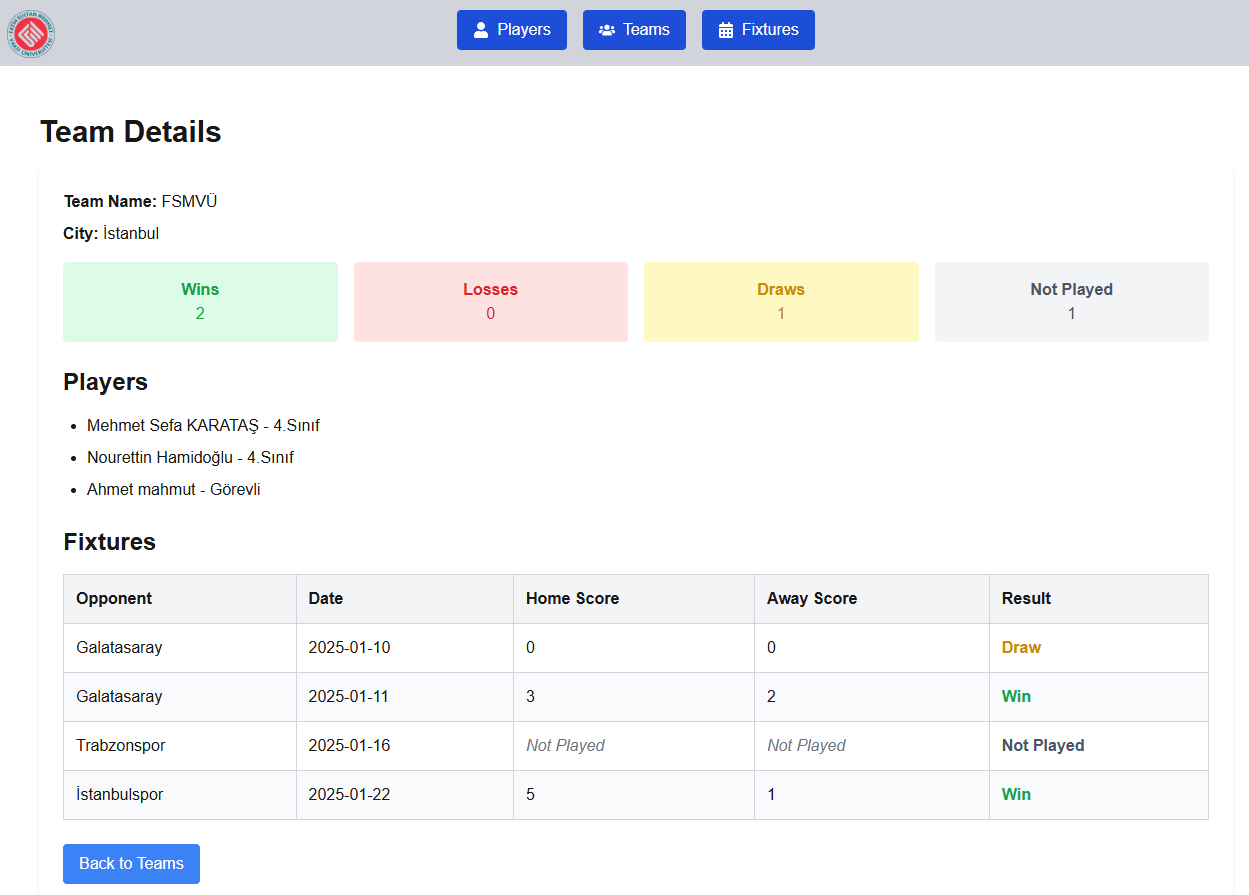
* useState: To manage local state in form components.
* useEffect: To handle side effects, such as fetching data from the API.
* useMemo: To optimize performance by memoizing derived data, such as position colors.
* useSelector and useDispatch: For interacting with the Redux store.

**4. Frontend Development**

**Teams Page**

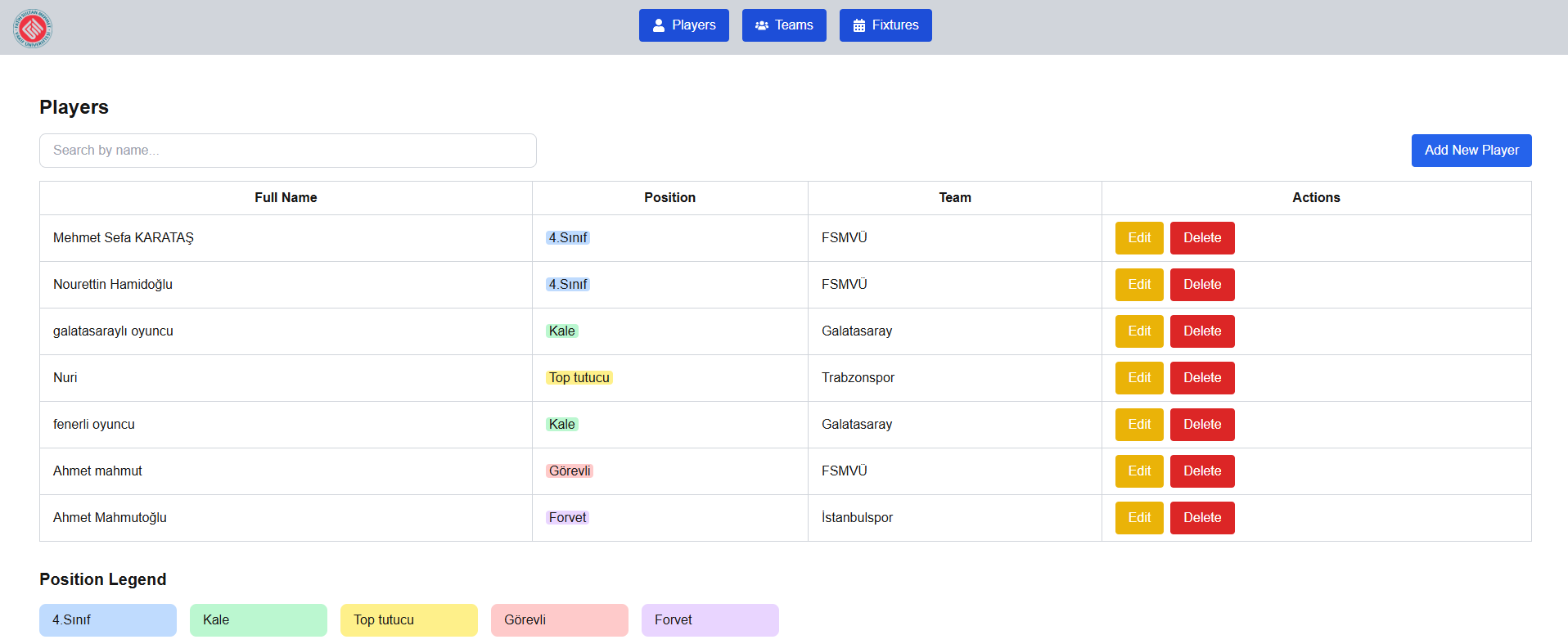
* Displays a list of teams in an HTML table.
* Provides buttons to view details, edit, or delete a team.
* Implements status messages using Redux.
* Example screenshot:





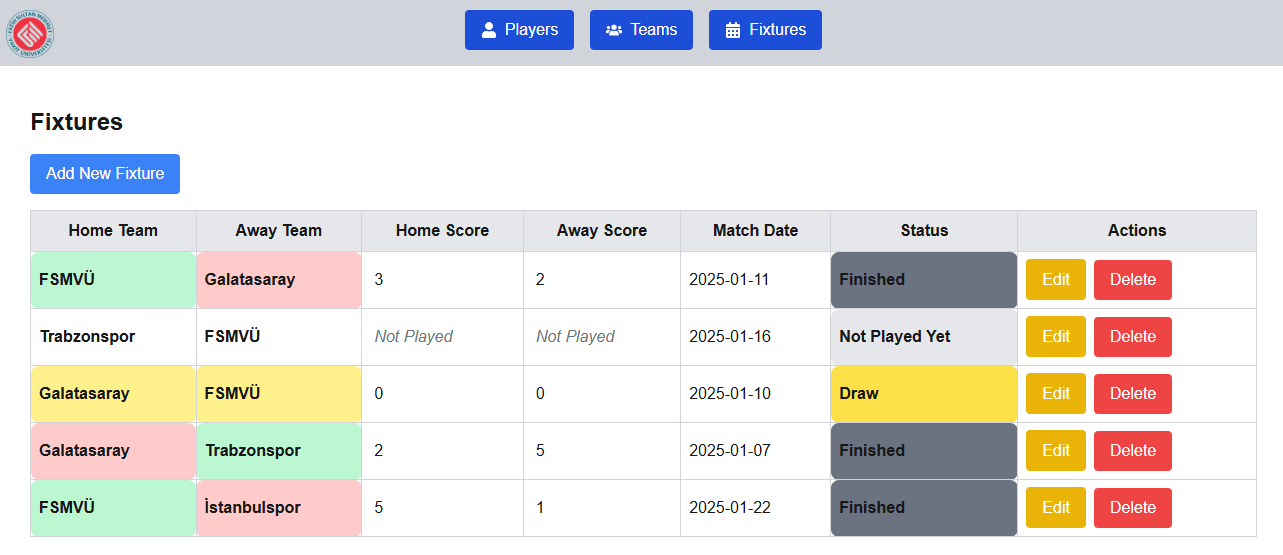
**Players Page**

* Displays a list of players with their positions and associated teams.
* Provides a search feature to filter players by name.
* Example screenshot:



**Fixtures Page**

* Displays a list of fixtures, including the participating teams and scores.
* Implements color-coded status for fixtures (e.g., "Not Played", "Draw").
* Example screenshot:



**5. Backend Development**

The backend was developed using ASP.NET Core Web API. The following endpoints were implemented:

* GET /api/Teams: Retrieves the list of teams.
* GET /api/Teams/{id}: Retrieves details of a specific team, including associated players and fixtures.
* POST /api/Teams: Adds a new team.
* PUT /api/Teams/{id}: Updates an existing team.
* DELETE /api/Teams/{id}: Deletes a team if it has no associated players or fixtures.

Similar endpoints were created for Players and Fixtures.

**6. Testing and Results**

The application was tested extensively to ensure proper functionality. Screenshots of different pages and operations are provided below:

* Teams Page *(Teams.png)*
* Team Details Page *(TeamDetails.png)*
* Players Page *(Players.png)*
* Edit Fixture Page *(EditFixture.png)*
* Delete Player Confirmation *(PlayerDelete.png)*
* Fixtures Page *(Fixtures.png)*

All CRUD operations were tested, and appropriate status messages were displayed during each operation.

**7. Conclusion and Evaluation**

The Sport and Fixture Management Application successfully meets all the requirements outlined in the project description. The use of React, ASP.NET Core Web API, and SQL Server ensured a robust and scalable solution. Key features such as CRUD operations, status messages using Redux, and dynamic form handling were implemented effectively.

Through this project, the following learning outcomes were achieved:

* Hands-on experience with full-stack web development.
* Understanding of React Hooks, Redux, and ASP.NET Core.
* Implementation of a relational database model and its interaction with a Web API.

The project is ready for deployment and can be further extended with additional features, such as user authentication and role-based access control.

Mehmet Sefa KARATAŞ **2021221008**