Web Application Programming 2025/26

Exam Project

Introduction

We want to develop a web application that allows the **booking of sports fields** (football, volleyball, basketball) and the **management of amateur tournaments**. The application will allow registered users to:

- book sports fields for specific time slots;
- search for fields, tournaments, teams and players;
- create tournaments associated with a specific sport;
- add teams and players to tournaments;
- automatically generate the tournament match schedule;
- enter match results;
- view automatically updated standings.

Users register with a unique *username*, *name*, and *surname*, and authenticate using a *password*.
The project consists of a **server-side** component that stores the data and handles authentication/authorization, and a **client-side** component that displays the application and data to the user.

Rules

The project must be developed independently. You may use any frameworks or libraries you wish, as long as the provided API is fully respected. You may use any LLM to assist in developing the project, but you are responsible for every single line of code. You must be able to justify every choice you make (library, framework, design decision). **Suggestion:** rely on an LLM if you know what you need to do and how to do it, but want to do it better. If you don't know how to do it, use an LLM at your own risk.

Main Features

- **Field list:** anyone can view all available sports fields, filtering by sport or via a text search.
- **Field details:** each field includes name, sport type, address, and bookable slots.
- **Availability view:** users can view available time slots for a given date.
- **Field booking:** authenticated users can book any free time slot.

    - **Constraints:** a slot cannot be booked more than once, and users cannot book past slots.
- **Booking cancellation:** users may cancel their upcoming bookings.
- **Tournament list:** view all tournaments, either active or completed, with support for search queries.
- **Tournament creation:** authenticated users may create a tournament specifying:
    - name
    - sport (football, volleyball, basketball)
    - maximum number of teams
    - start date
- **Tournament editing:** the creator may edit certain fields (e.g., name, max teams).
- **Tournament deletion:** only the creator may delete a tournament.
- **Tournament details:** includes general information, teams, matches, and standings.
- **Team creation:** the tournament's creator can add teams by specifying a team name.
- **Player management:** teams can contain multiple players, each with:
    - name
    - surname
    - optional jersey number
- **Visualization:** list of teams and players available for each tournament.
- **Automatic schedule generation:** once all teams are registered, the match schedule can be generated (single round-robin).
- **Match details:** participating teams, date, optional field, status (upcoming/played), and result.
- **Result entry:** the tournament creator may enter the final score once the match date has passed.
- **Search**: partial, case-insensitive search should be available for fields, tournaments, teams, players (eg, searching "cal" should match "Calcio", "Calisthenics Arena", "California Team", etc.);
- **User list**: it is possible to view a list of users, optionally filtered by a search parameter. For each user, all tournaments created by them will be shown.
- **User registration**: A new user can register by providing:
    - Username
    - Password
    - Name
    - Surname

Standings are automatically computed based on match results (**football**: 3 points for a win, 1 for a draw, 0 for a loss, **volleyball/basketball**: 2 points for a win, 0 for a loss).
Standings are publicly available and include:
- points
- matches played
- goals/points scored and conceded
- goal/point difference

Authentication

Users can sign up by providing:

- username (unique)
- password
- name
- surname

Any operation that modifies data (booking a field, creating a tournament, adding teams, entering results, etc.) must require authentication. After login, ensure that the user has the necessary permissions (for example, only the tournament creator may modify or delete it)

REST Interface

The project requires the implementation of a REST interface. The API to be implemented is listed below.

Metodo	**API**	**Descrizione**
POST	/api/auth/signup	Register a new user
POST	/api/auth/signin	User login
GET	/api/fields?q=*query*	List of sports fields (searchable)
GET	/api/fields/*:id*	Field details
GET	/api/fields/*:id*/slots?date=*YYYY-MM-DD*	Availability for a specific date
POST	/api/fields/*:id*/bookings	Book a slot (authenticated)
DELETE	/api/fields/*:id*/bookings/*:bookingId*	Cancel a booking (authenticated)
GET	/api/tournaments?q=*query*	List of tournaments
POST	/api/tournaments	Create a new tournament (authenticated)
GET	/api/tournaments/*:id*	Tournament details
PUT	/api/tournaments/*:id*	Edit tournament data
DELETE	/api/tournaments/*:id*	Delete the tournament (creator only)
POST	/api/tournaments/*:id*/matches/generate	Generate match schedule
GET	/api/tournaments/*:id*/matches	List matches
GET	/api/matches/*:id*	Match details
PUT	/api/matches/*:id*/result	Enter match result
GET	/api/tournaments/*:id*/standings	Tournament standings
GET	/api/whoami	If authenticated, returns information about the current user

Project Delivery

To deliver a working demo of the project, Docker is strongly recommended so that everything required to run the application is included.
Use container names that are unique, preferably including your surname.
An example of a configuration with two containers using Node.js and MongoDB is shown below:

```
version: "3"
services:
 app:
   container_name: app
   build: .
   command: nodemon --watch /usr/src/app -e js app.js
   ports:
     - "3000:3000"
   volumes:
     - ./app:/usr/src/app
   links:
     - "mongo:mongosrv"
 mongo:
   container_name: mongo
   image: mongo
   volumes:
     - ./data:/data/db
   ports:
     - '27017:27017'
```

Associato ad un Dockerfile per il container “app”:

```
FROM node:latest
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
RUN npm install -g nodemon
COPY ./app/package.json /usr/src/app
RUN npm install
COPY ./app /usr/src/app
EXPOSE 3000
```

You may use any database (MongoDB, MySQL, PostgreSQL). If you use a database, you must provide a script to initialize the schema/collections. **Do not** submit: gigabytes of database data, node modules or similar dependency folders (these will be installed via package managers).