Tennis Tournament Management System

Crihălmeanu Tudor - 30434

Introduction

This application is a web-based tennis tournament manager developed using Java and Spring Boot on the backend, along with a simple HTML/CSS/JavaScript frontend. The system allows users to register as players, referees, or administrators, participate in tournaments, track match results, and manage tournament data.

The project follows best practices in software architecture, design, and documentation, and includes key features such as role-based functionality, score management, file exports, and more.

Application Architecture

The application is structured according to the **Layered Architecture Pattern**, a well-known design that separates concerns across different parts of the system. This makes the system easier to maintain, test, and scale.

1. Presentation Layer (Frontend)
The frontend is built using plain HTML, CSS, and JavaScript. It provides a clean and intuitive interface where users can log in, join tournaments, view matches, submit scores (for referees), and perform administrative actions. It interacts with the backend solely via HTTP REST calls using the `fetch()` API.

2. Controller Layer (Web/API Layer)
Controllers expose RESTful endpoints annotated with `@RestController`. They handle incoming HTTP requests, delegate the actual processing to services, and return responses. Each controller is mapped to a domain-specific route (`/users`, `/matches`, `/tournaments`, etc.).

3. Service Layer (Business Logic)
Services implement the business logic and mediate between controllers and repositories. They contain validations and operations such as match creation, score submission, and tournament registration.

4. Repository Layer (Persistence)
Using Spring Data JPA, the repositories define how entities interact with the database. By extending `JpaRepository`, CRUD operations are handled automatically. Custom queries can be defined when necessary.

5. Security Configuration
Security is configured via Spring Security's `SecurityFilterChain`, and passwords are hashed using `BCryptPasswordEncoder`. Login is not restricted by endpoint, but password storage remains secure.

Design Patterns Used

1. Layered Architecture Pattern
The system separates responsibilities into layers: controller, service, and repository. This ensures low coupling and high cohesion, improving maintainability and scalability.

2. Data Transfer Object (DTO) Pattern
DTOs such as `LoginDTO` and `MatchScoreUpdateDTO` are used to transfer specific data structures between layers and with the frontend. This enhances security and minimizes exposed data.

Use Case Diagram

The system supports three types of users:

- **Player:** Registers, logs in, joins tournaments, views their matches, and updates their profile.
- **Referee:** Views assigned matches and submits scores.
- **Admin:** Creates tournaments, updates/deletes users, views/export users and match data.

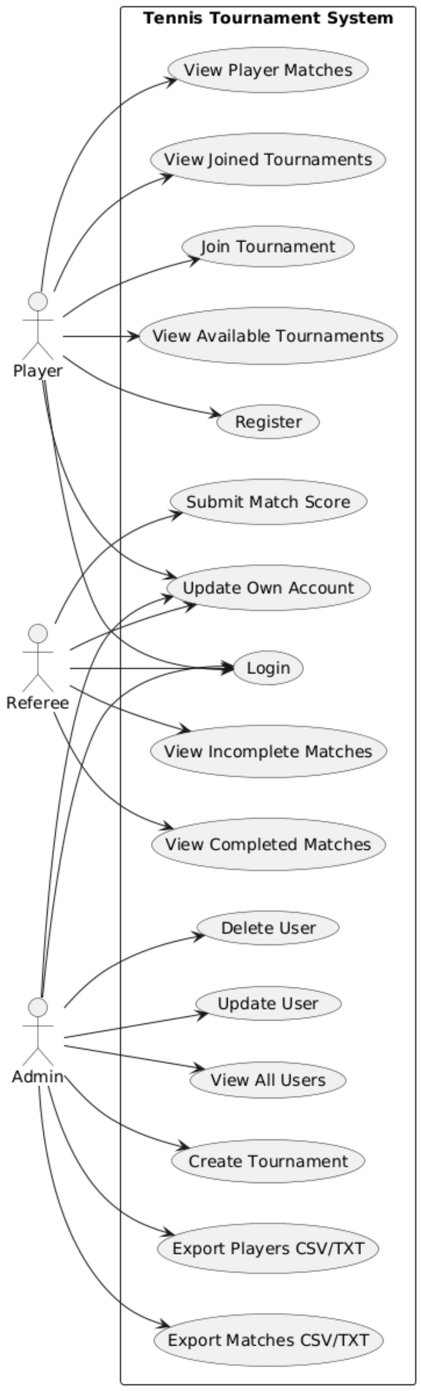

Figure 1: Use Case Diagram

Class Diagram

The class diagram shows the relationship between main entities such as User, Tournament, and Match. It highlights how these classes are organized across layers and how they relate to each other.

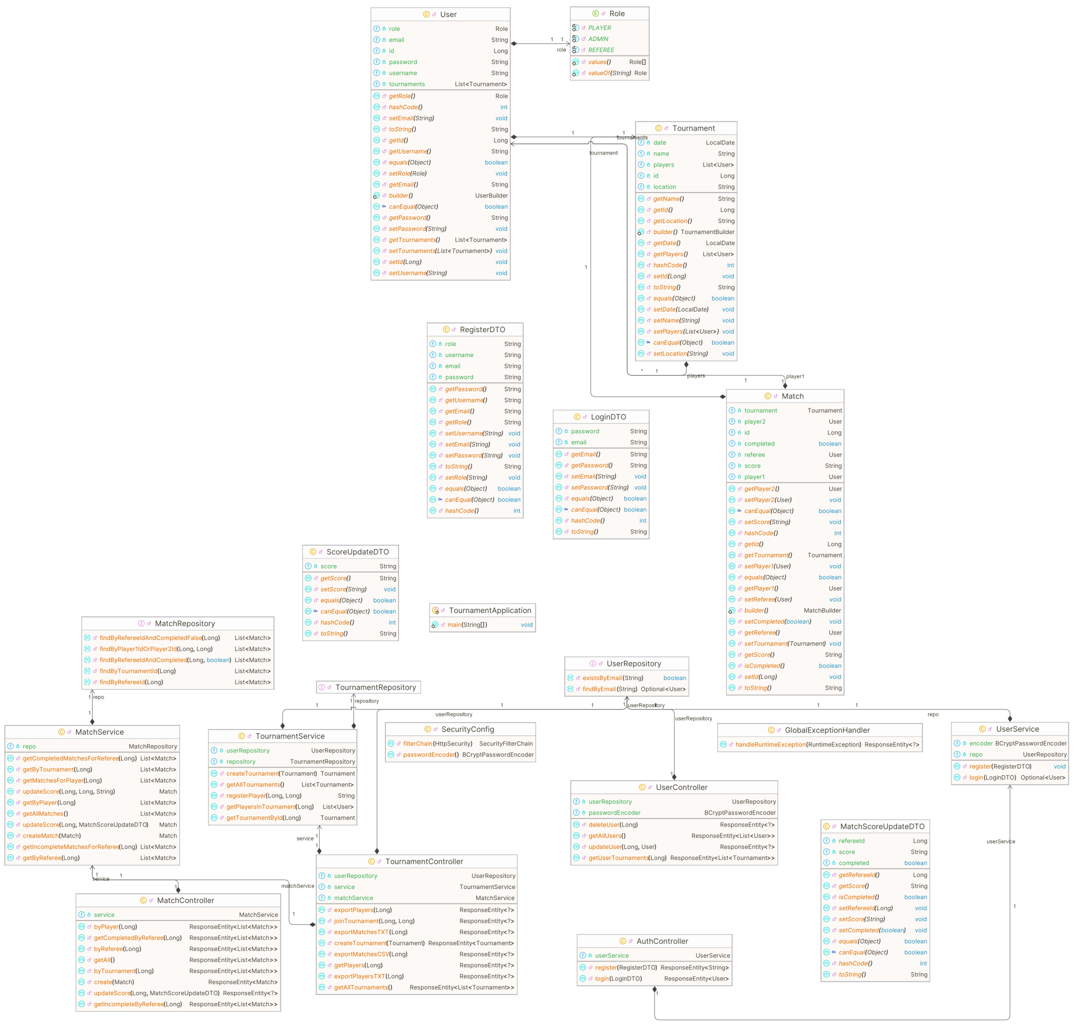

Figure 2: Class Diagram

Database Diagram

The database structure includes the following tables:

- `user` - stores user data
- `tournament` - stores tournament data
- `matches` - stores match data, including player and referee relationships
- `user_tournaments` - a many-to-many join table

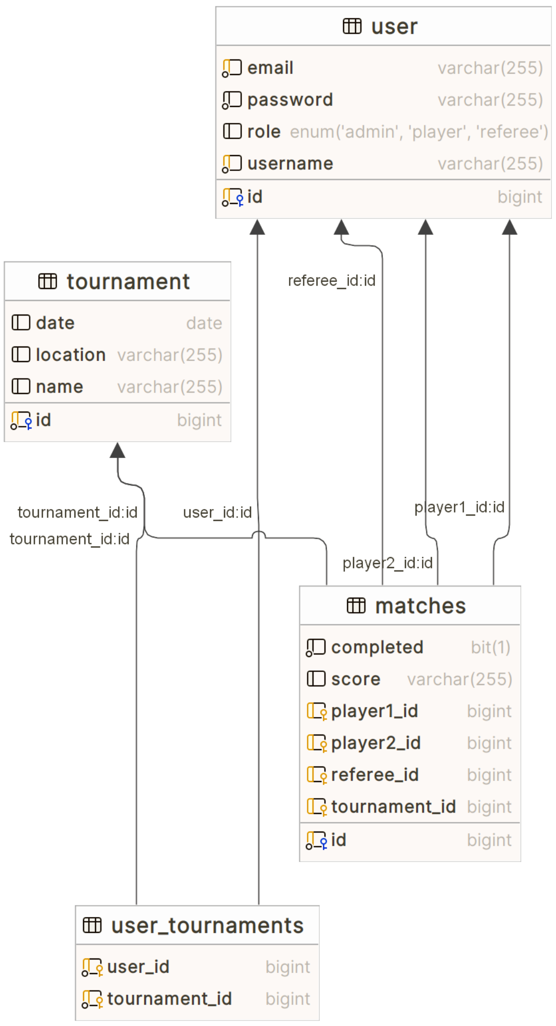

Figure 3: Database Schema

Export Functionality

Admins can export both match and player data in CSV or TXT formats for any tournament. The backend generates these files on the fly using Spring Boot, and the frontend fetches and downloads them using dynamic JavaScript logic. Invalid IDs trigger an alert popup, improving user experience.

Account Management

Users can update their account information, including their password (which is automatically re-hashed). Admins have additional privileges to update any user's data and role.

Conclusion

This system demonstrates a full-stack, role-based web application using Spring Boot and JavaScript. The implementation adheres to modern software development principles, is well-organized through the use of layered architecture and DTOs, and satisfies all functional and non-functional requirements.