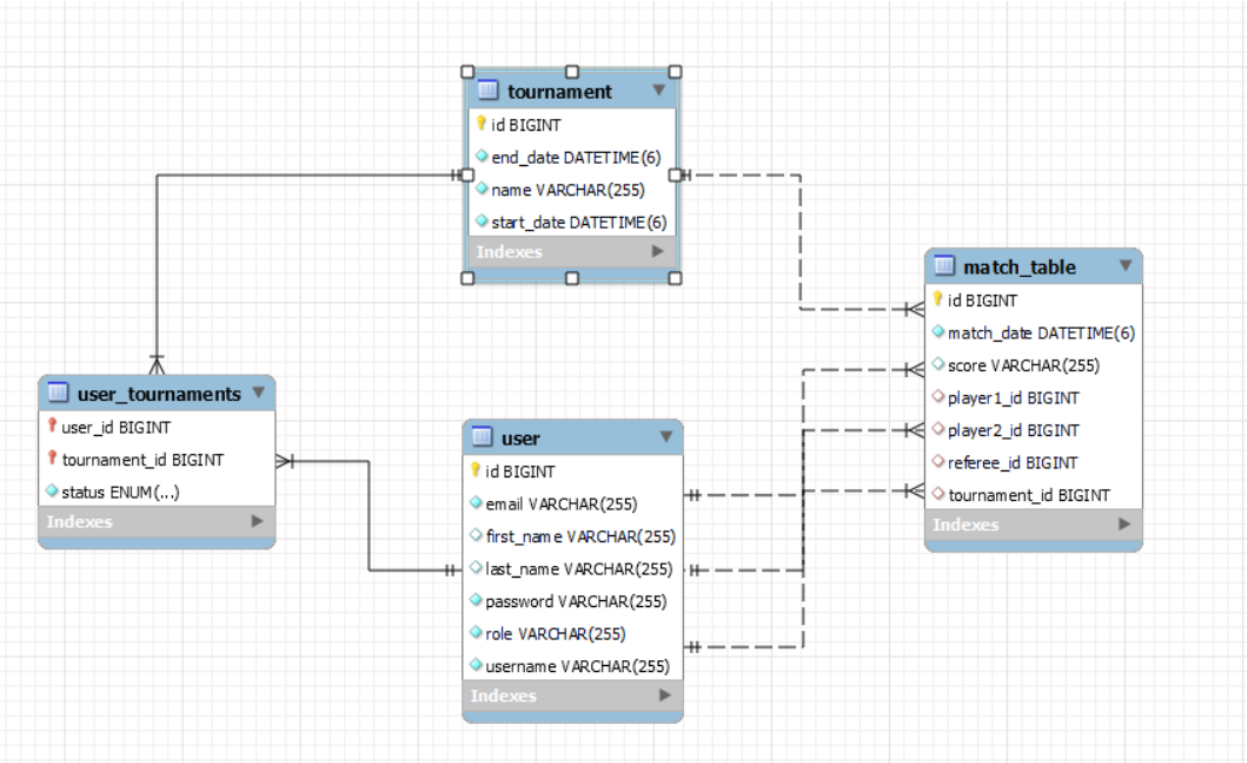
Software Design – Assignment 2

Tennis App

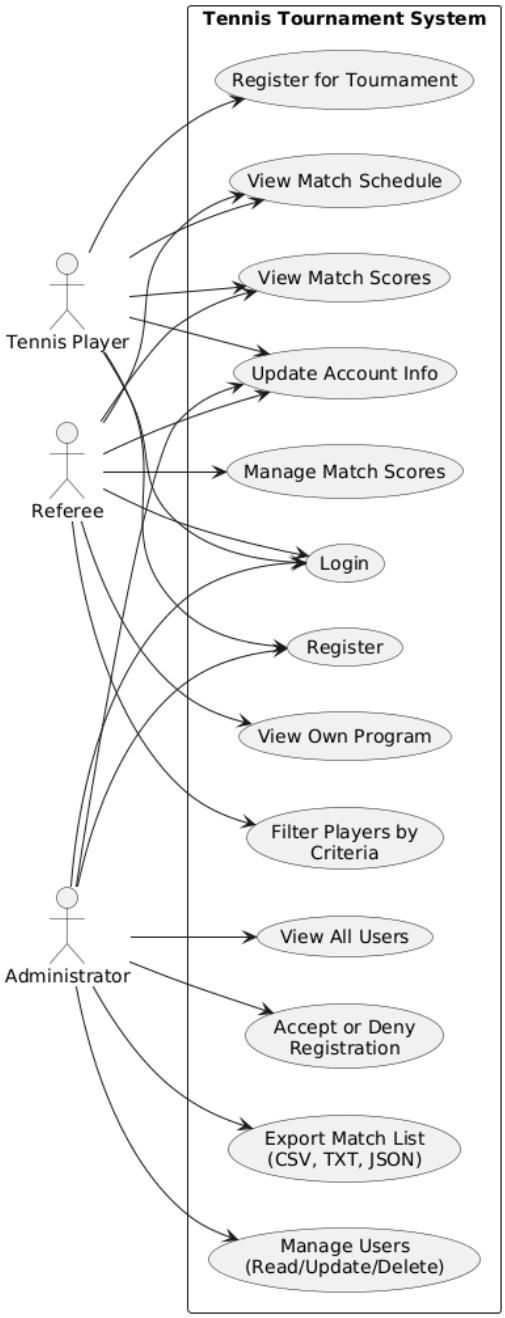
Database Schema



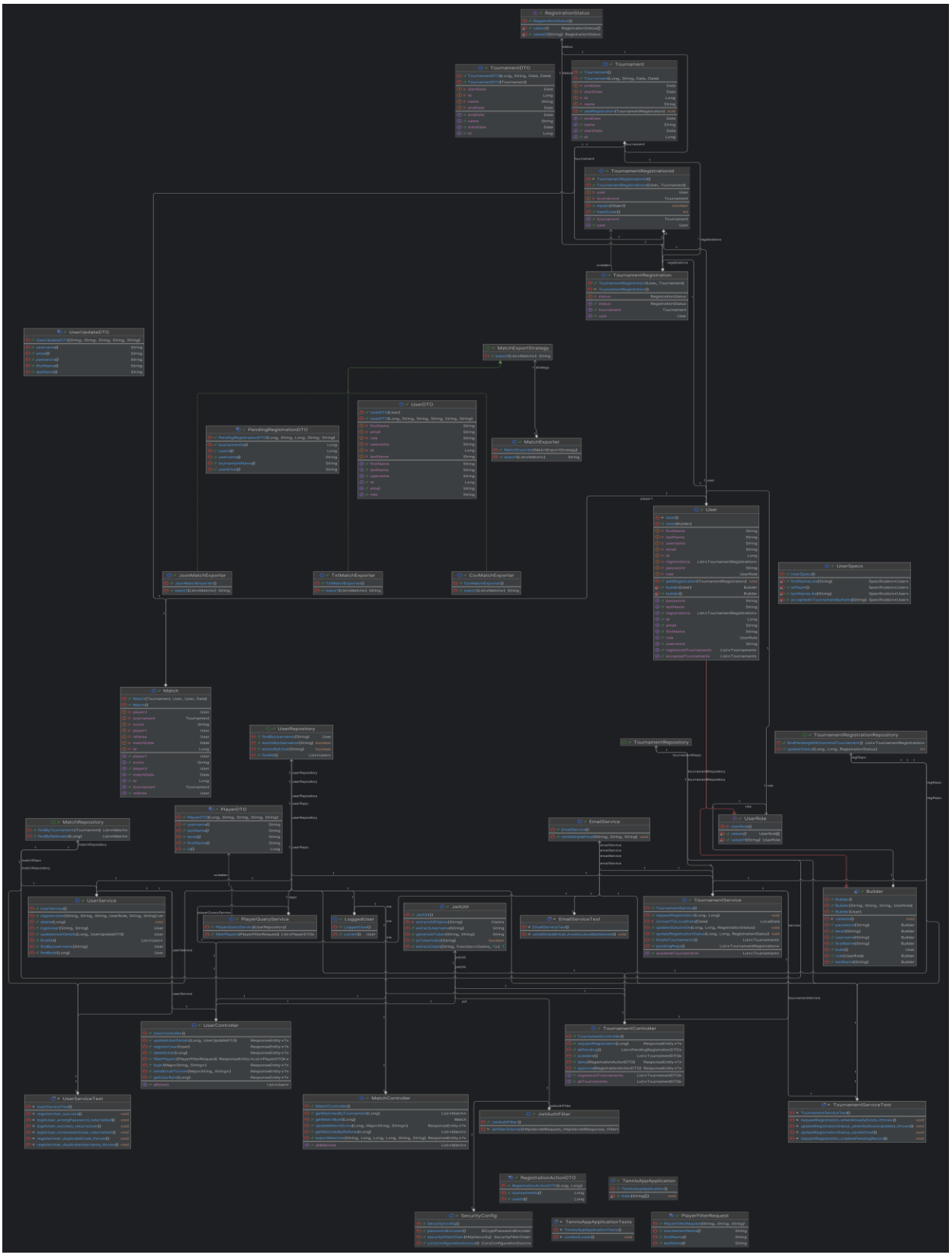
- The User table holds every account in the system—players, referees and administrators—and is used whenever someone logs in or is assigned to a match or tournament.
- The Tournament table records each competition's basic information and serves as the anchor for both registrations and match schedules.
- The User_Tournaments table sits between User and Tournament and tracks who has signed up for which event and whether their entry is pending approval, has been accepted or has been denied.

- The Match_Table links individual contests back to their tournament and to the two players and one referee involved, storing the date played and the outcome for each match.

Use Case Diagram



Class Diagram



1. **Entities**
   • **User**: a system user with fields like id, username, password, email, firstName, lastName and role. Users can register for tournaments, referee matches, or play in them.
   • **Tournament**: holds tournament details (id, name, startDate, endDate, status). It has a many-to-many link to User via TournamentRegistration.
   • **Match**: represents a single match with id, matchDate, score, two players (player1, player2), a referee, and the tournament it belongs to.
   • **TournamentRegistration** (and its composite key TournamentRegistrationId + enum RegistrationStatus): the join entity tracking which users have requested, been accepted or denied registration for a tournament.

2. **Repositories**
   • **UserRepository**: CRUD and query methods for User entities.
   • **TournamentRepository**: CRUD and custom queries for Tournament.
   • **MatchRepository**: CRUD and custom queries for Match.
   • **TournamentRegistrationRepository**: operations on the user_tournaments join table (e.g. updateStatus).

3. **Controllers**
   • **UserController**: exposes REST endpoints for user registration, login, profile updates, and listing/filtering players.
   • **TournamentController**: endpoints for listing tournaments, requesting registration, viewing pending registrations, and approving or denying them.
   • **MatchController**: endpoints to fetch all matches, update match scores, and trigger export of match lists in various formats.

4. **Services**
   • **UserService**: business logic around creating users, checking duplicates, encoding passwords, and authenticating via JWT.
   • **TournamentService**: handles tournament-registration workflows, status updates, pending-list retrieval and sends notification emails.
   • **PlayerQueryService**: builds dynamic JPA Specifications to filter players by first name, last name and tournament participation.
   • **MatchExportStrategy** (interface) and its implementers **CsvMatchExporter**, **JsonMatchExporter**, **TxtMatchExporter**: strategy-pattern components that know how to format a list of matches into the respective file format.
   • **MatchExporter**: picks a concrete strategy at runtime and coordinates the export.

5. **Utilities & Configuration**
   • **SecurityConfig**: sets up spring-security filters, JWT authentication and authorization rules.
   • **JwtUtil** / **JwtService**: helper for generating and validating JWT tokens.
   • **LoggedUser**: utility that resolves the currently authenticated User from the security context.
   • **TennisAppApplication**: the Spring Boot main class that boots the application.
   • **TennisAppApplicationTests**: starter test class for loading the Spring context in integration tests.

6. **Builder**
   • **User.Builder** (or similar Builder class): provides a fluent API for constructing User (or other entity) instances with chained setter methods.