Football Player Spotlight (Java and Spring) Application Architecture Overview

1. Spring Boot Application

- Acts as the core backend service, exposing RESTful APIs for authentication, user management, posts, comments, and notifications.
- Organized into **controllers**, **services**, and **repositories** for clean separation of concerns.

2. Authentication & Security

- **AuthController** handles `/auth/register` and `/auth/login`.
- **JWT Token Provider** generates and validates tokens, embedding user ID and role.
- **JwtAuthenticationFilter** intercepts requests, validates tokens, and injects `userId` into the request context.
- **Spring Security** ensures protected endpoints require valid tokens.

3. Controllers

- **UserController**: manages profiles (`/users/{id}`, `/users/me`), follow/unfollow, profile updates, and fetching followers/following.
- **PostController**: handles creating posts, fetching feeds, likes, and comments.
- **AuthController**: signup/login endpoints.

4. Services

- **UserService**: business logic for user profiles, updates, following/unfollowing, and notifications.
- **AuthService**: registration and login, password encoding, token generation.
- **NotificationService**: sends alerts (e.g., when someone follows you).
- **PostService**: manages post creation, retrieval, and interactions.

5. Repositories (Data Access Layer)

- **UserRepository**: JPA repository for user entities, with queries for email, username, and search.

- **PostRepository**: fetches posts by user, ordered by creation date.
- **Other repositories**: handle comments and relationships.

6. Database

- **PostgreSQL** stores all persistent data:
  - `User` table: id, username, email, password, teamSupported, profilePicture, followers/following relationships.
  - `Post` table: id, userId, title, image, createdAt.
  - `Comment` table: id, postId, userId, content.
- JPA/Hibernate maps entities to tables.

🔄 Flow

1. **User registers** → AuthService saves user in DB.
2. **User logs in** → AuthService validates credentials, returns JWT.
3. **Frontend calls** `/users/me` → JwtAuthenticationFilter validates token, injects `userId`, UserController fetches profile.
4. **Authenticated requests** (posts, follow, comments) → token validated, services update DB, notifications triggered.