ER DIAGRAMM: Изображение выглядит как текст, снимок экрана, число, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**users** ↔ **accounts**: связь «один ко многим» (один пользователь — много аккаунтов).

**accounts** ↔ **transactions**: каждая транзакция ссылается на два счёта (отправитель и получатель

**List of API Endpoints (CRUD + Authentication)**

**1. Authentication**

* **GET /login**  
  Returns the login page (Thymeleaf).
* **POST /login**  
  Processes the login form (handled by Spring Security).
* **GET /register**  
  Registration page (Thymeleaf).
* **POST /register**  
  Creates a new user (via JSON or form submission).

**2. Users (User)**

* **GET /users**  
  Returns a list of all users (JSON).
* **GET /users/{id}**  
  Returns a user by ID.
* **POST /users**  
  Creates a user (JSON).
* **PUT /users/{id}**  
  Updates an existing user (JSON).
* **DELETE /users/{id}**  
  Deletes a user by ID.

**3. Accounts (Account)**

* **GET /accounts**  
  Returns a list of all accounts.
* **GET /accounts/{id}**  
  Returns details for a specific account.
* **POST /accounts**  
  Creates a new account.
* **PUT /accounts/{id}**  
  Updates an account (if necessary).
* **DELETE /accounts/{id}**  
  Deletes an account (optional).

**4. Transactions (Transaction)**

* **GET /transactions**  
  Returns a list of all transactions (optionally filtered by account\_id).
* **POST /transactions**  
  Creates a new transaction (JSON containing from\_account\_id, to\_account\_id, amount).
* **(Optional)** **GET /transactions/{id}**  
  Returns detailed information about a specific transaction.

**Brief Overview of the Architecture (MVC)**

**Layers**

1. **Controller**

Receives HTTP requests (REST endpoints or Thymeleaf pages).

Calls the corresponding services.

Returns either JSON responses or rendered HTML templates.

1. **Service**

Contains the business logic (e.g., checking the account balance before a transaction).

Calls repositories for data storage and retrieval.

May invoke other services (e.g., an EmailService) if needed.

1. **Repository (DAO)**

Interfaces that extend Spring Data JPA (e.g., UserRepository, AccountRepository, TransactionRepository).

Responsible for CRUD operations on the database (create, read, update, delete).

1. **Model (Entity)**

Classes such as User, Account, and Transaction annotated with @Entity.

Fields correspond to columns in the database (e.g., id, username, balance, etc.).

**Example of Interaction**

1. A client (browser or Postman) sends GET /users/1.
2. **UserController** calls userService.getUserById(1).
3. **UserService** calls userRepository.findById(1).
4. **UserRepository** (via Spring Data) executes SELECT \* FROM users WHERE id = 1.
5. The User entity is returned; the service applies any needed logic, and the controller constructs a JSON response.
6. The client receives a JSON representation of the user data.