Dokumentation 233

Lukas, Yara, Cyrill

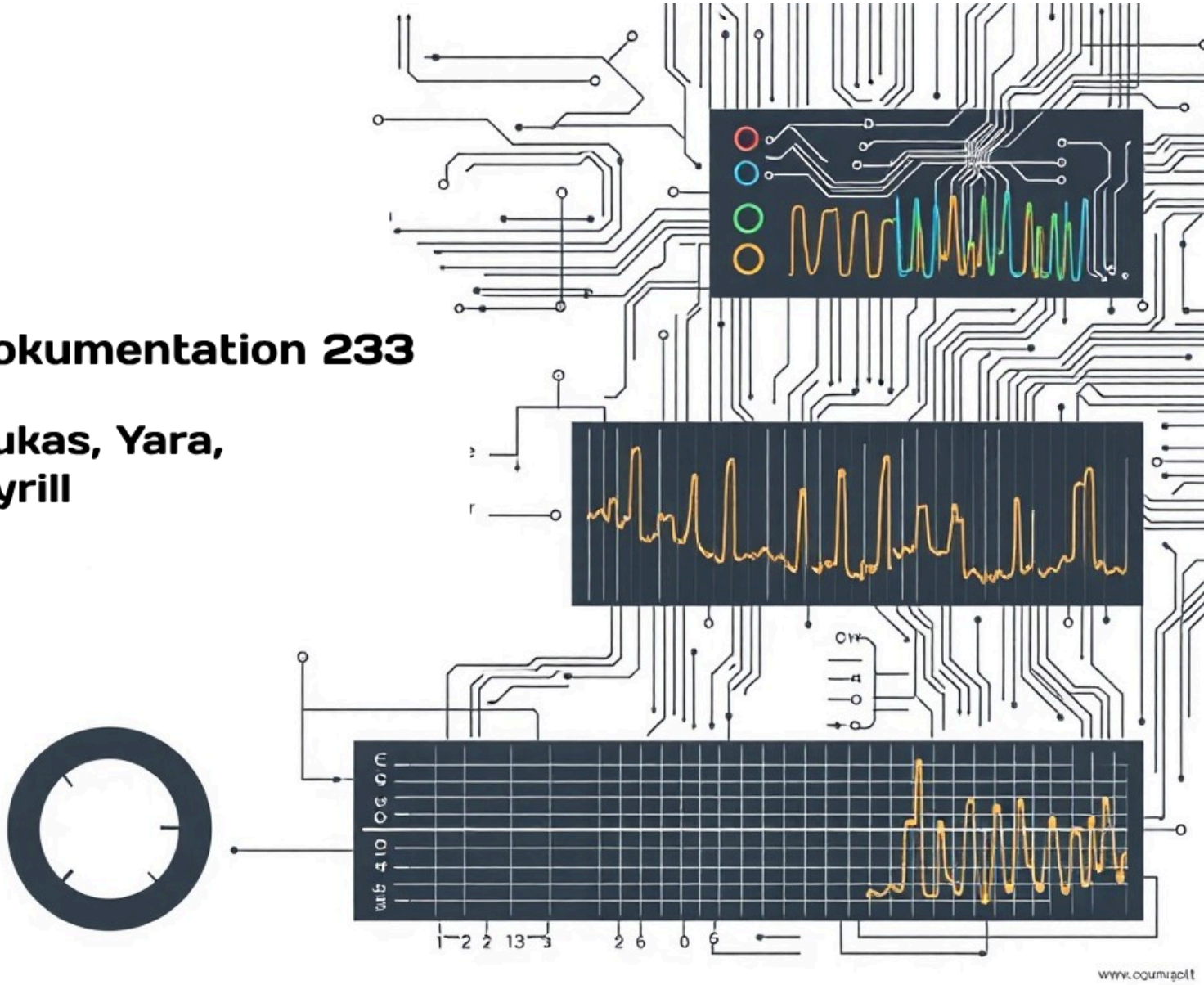


Table of contents

1. Project Assignment

1.1. Functional Requirements

User Roles & Privileges:

- Existing roles and authorities are edited or extended in order to fulfill and test the requirements below.
- The personal information of a user is only accessible to administrators or the user himself.
- Admins can also edit, create and delete other users.

Frontend:

- Login-Page, which is publicly available (provided)
- Publicly available HomePage (provided)
- Homepage for logged in Users
- Admin Page (only available for admins)
- At least one component to enable group-specific functionalities in the frontend to enable group-specific functionalities in the frontend.

Security:

- Each REST-Endpoint should only be accessible with meaningful authorities. This is checked with automated tests.
- There are areas of the front end that are only accessible to logged-in users.
- There are areas of the front end that are only accessible to admins
- The authentication mechanism is implemented with JSON web tokens.(provided)

Group specific assignment

1. **User Profile:**
    - Create a UserProfile model that contains additional information about a user (address, birthdate, profile picture URL, age)
    - Each entry in UserProfile can be clearly assigned to a user.
    - Create endpoints in your application to perform typical CRUD operations to UserProfile.
    - A user should (only) be able to see their own UserProfile.
    - Administrators should be able to access all UserProfiles. This list of user profiles should use pagination and be sorted by any parameters of the user profile in ascending and descending order.

1.2. Non-Functional Requirements

Implementation:

- Data is persisted in a PostgreSQL database, the OR mapping is realized with JPA.
- A Frontend with React (Typescript) is used.
- A Springboot (Java) backend is used.
- The source code is committed daily in a GIT repository.

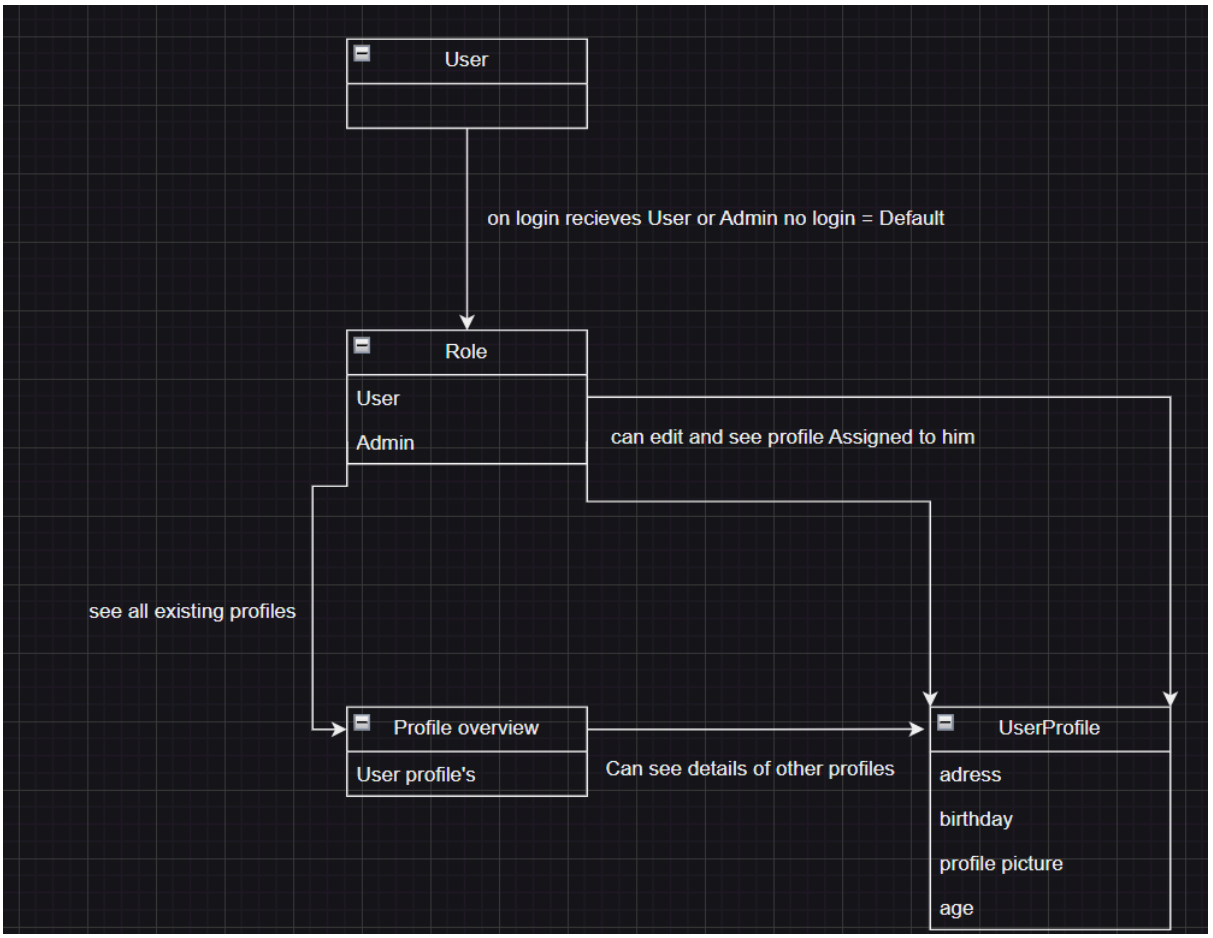
Testing:

- General functionality of all self-implemented endpoints is tested with Cypress (mandatory), Postman and/or JUnit.
- Special emphasis is placed on testing access authorizations.
- At least one use case is tested in detail with Cypress. This
- includes at least
    - The endpoint is tested with multiple users & roles
    - At least one success case and one error case is tested.
    - Use cases are described according to the UML standard for these cases.

Multi-user capability:

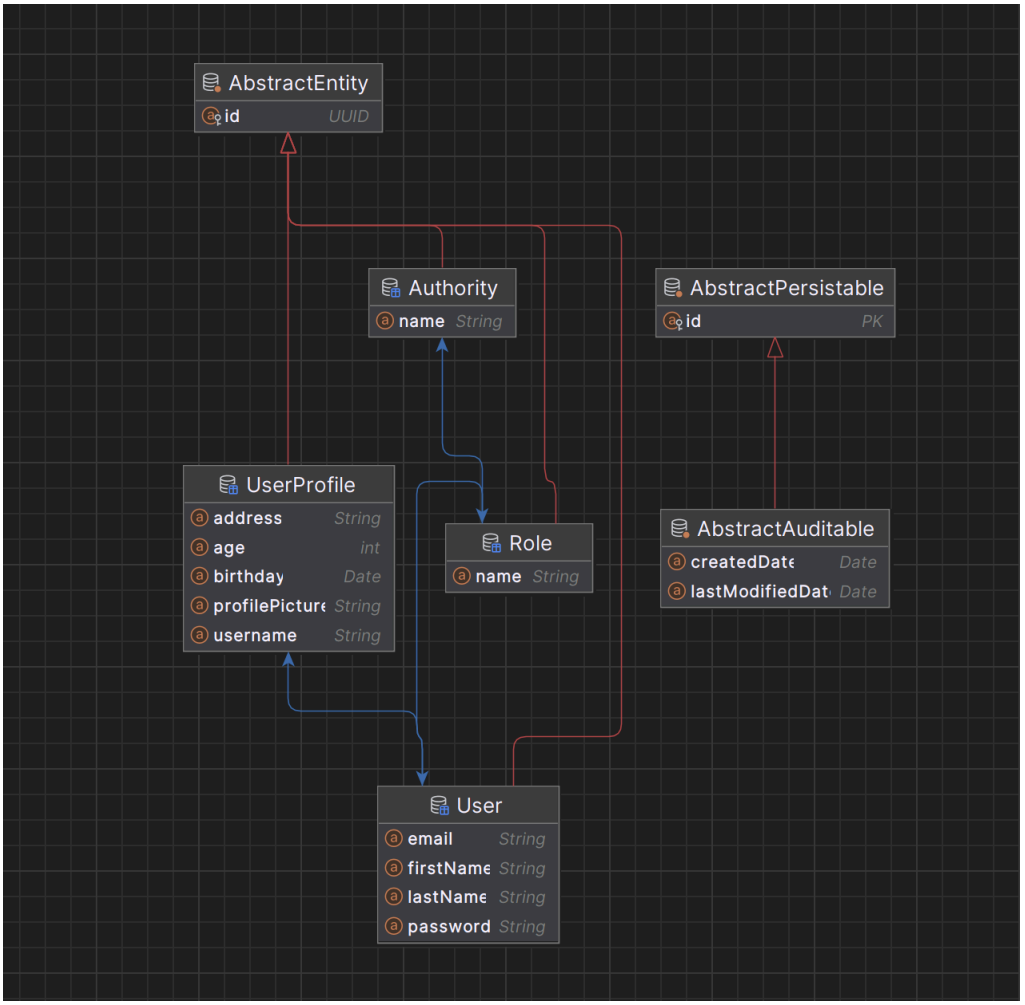
- Aspects of multi-user capability, such as compliance with the ACID principles are taken into account

2. Domain Model



The Domain Diagram is built to visualize the different Pages and what functions they contain. Looking at the Diagram you can see that there is a UserProfile which contains information about the user such as username, birthdate and so on. There is also ProfileOverview where all Userprofiles are displayed, with their given fields. Most important of all are the roles which an user has. Users with a user role are able to edit and see their own UserProfile, but aren’t able to see other Profiles which aren't their own. A user with an admin role is able to see the ProfileOverview, but they can’t edit another user's Profile.

3. ERD



An Entity Relationship Diagram (ERD) is used to visualize the relationships between different data entities within a system. Looking at the diagram, you can see that each entity, such as "User" and "UserProfile," contains specific attributes like email, address, or birthday. Relationships between these entities, such as how a user might have a profile or be assigned specific roles, are represented by connecting lines.

4. Testing-Strategy

Cypress
We will use Cypress to extensively test one endpoint according to our Use Case Description.

Postman
We will make a Postman Collection that tests

- Get Userprofile:
  - If a User has access to his Userprofile
  - If a User can **only** access his Userprofile
  - If the Admin has access to a Userprofile other than his own
- Get All Userprofiles (paginated)
  - If the Admin gets all Usrprofiles
  - If a User doesn't have access
  - If the sort parameter changes the sorting
  - If limit and offset return the expected Userprofiles
  - If a negative offset returns a 400
- Post Userprofile
  - If a User has access to create a Userprofile
  - If a User can **only** create his Userprofile
  - If the Admin has access to create a Userprofile other than his own
  - If it returns a error when the User already has a Post
- Delete Userprofile
  - If a User has access to delete a Userprofile
  - If a User can **only** delete his Userprofile
  - If the Admin has access to delete a Userprofile other than his own
  - If it returns a error when the Userprofile doesnt exitsts
- Put Userprofile
  - If a User has access to update a Userprofile
  - If a User can **only** update his Userprofile
  - If the Admin has access to update a Userprofile other than his own
  - If it returns a error when the Userprofile doesnt exitsts

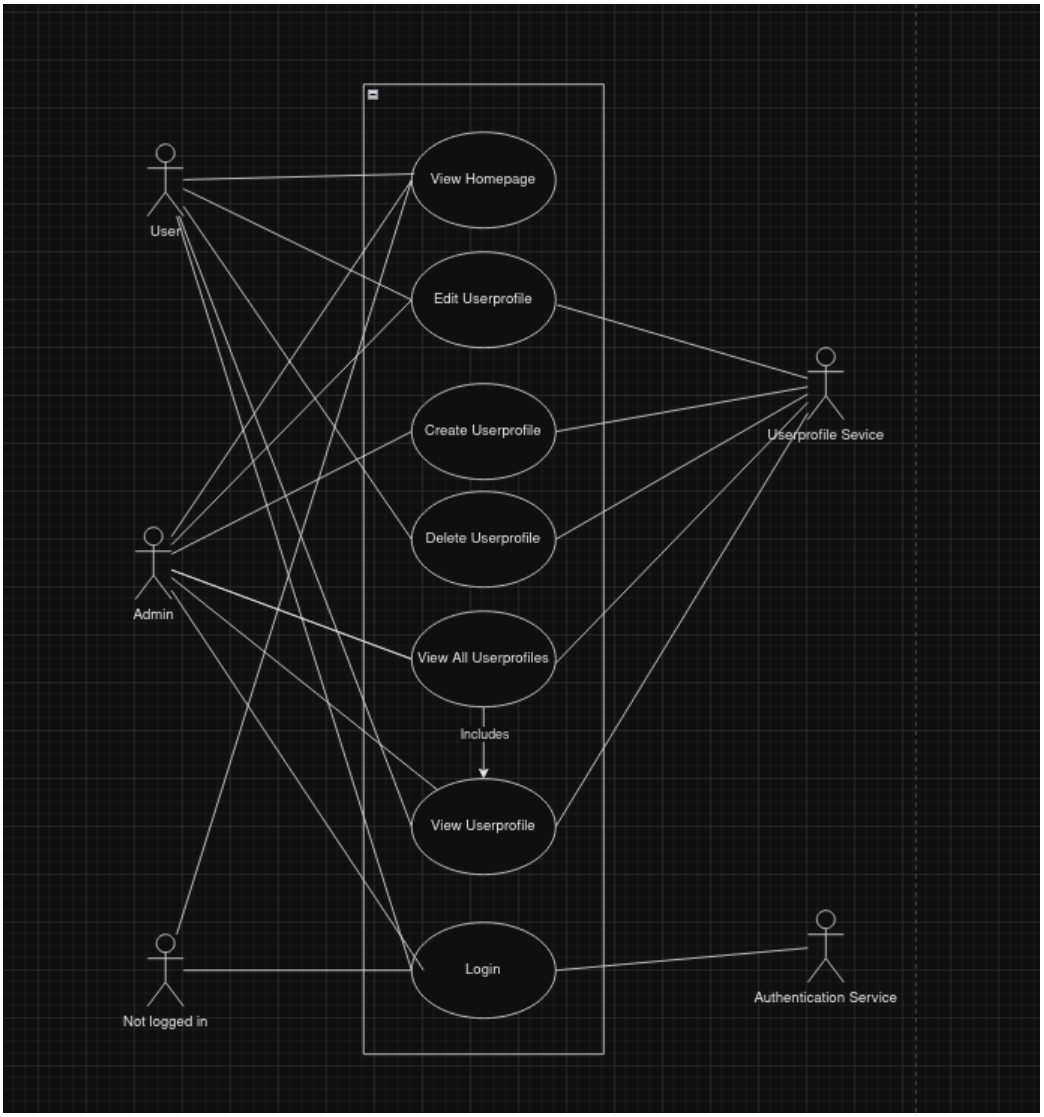
Expected Behavior:
We expect that the endpoints return Status Code 200 when successful, 401 when Unauthorized, 404 when the resource is not found and 201 when something is successfully created.

JUnit
We won't use JUnit to test because we decided that we want to test all Endpoints with Postman. We will test the Endpoints to see if the data that we receive is correct, and correlates to the authorization the user has.

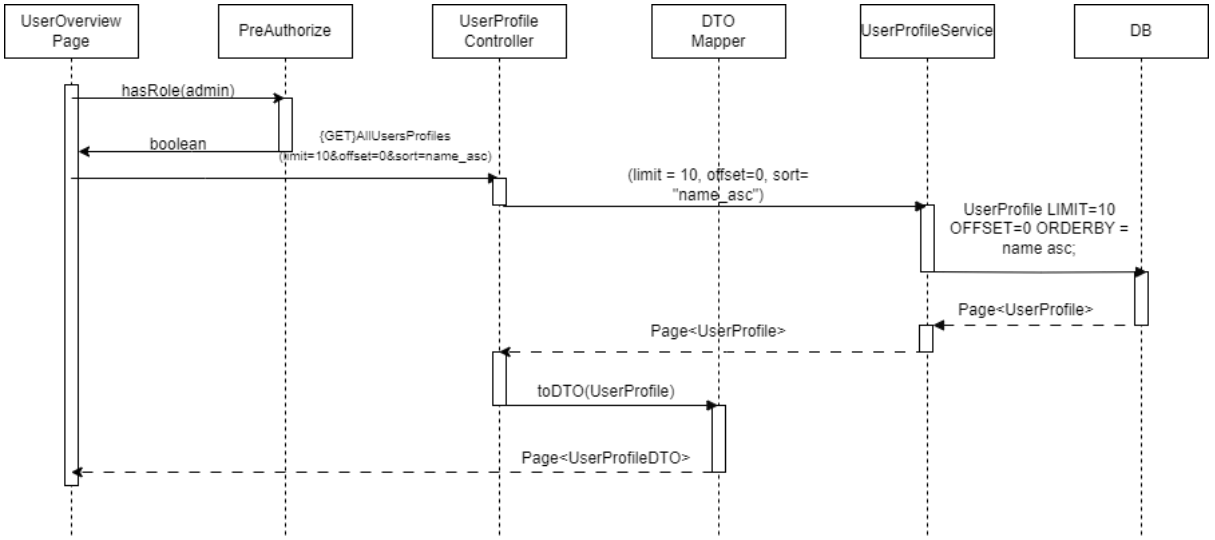
4.1. Use-Case Description

Actor(s):	**Admin**
Description:	The Admin wants to see all the Userprofiles.
Preconditions:	- Admin Account with Correct access rights exists
	- Admin is Authorized (has Admin rights)
Postconditions:	- Admin navigates away
Normal Course:	1. Admin logs in
	2. Navigates to Userprofile Overview
	3 Get request for Userprofile list with params limit: 10 and offset: 0
	4 Userprofile list gets Displayed
	5. If Admin sorts displayname ascending, a new get request for Userprofile list with limit: 10, offset: 0 and displayname ascending
	6. Sorted Userprofile list gets Displayed
	7. Admin goes to the next page
	8 Get request for Userprofile list with params limit: 10 and offset: 1
	9 Userprofile list gets Displayed
Alternative Courses:	- If less than 11 Userprofile exist Admin can’t go to the next page (button doesn’t exist)
Exceptions:	None

5. Use-Case Diagram



6. Sequence Diagram



The sequence diagram depicts the process of handling a GET request for retrieving all user profiles. The sequence begins with a role check to verify whether the user making the request has the necessary **authorization**. If the authorization check returns true, the request is allowed to proceed to the Controller/Endpoint.

Once the request reaches the **UserProfile Controller**, the parameters are passed on to the **UserProfile Service**. The service processes these parameters and generates an SQL query to retrieve the paginated user profiles from the database.

After the data is fetched from the database, the controller uses a **DTO Mapper** to map the returned data into a Data Transfer Object (DTO) format. The mapped data is then sent back as a response to the User Overview Page, where the user profiles are displayed.