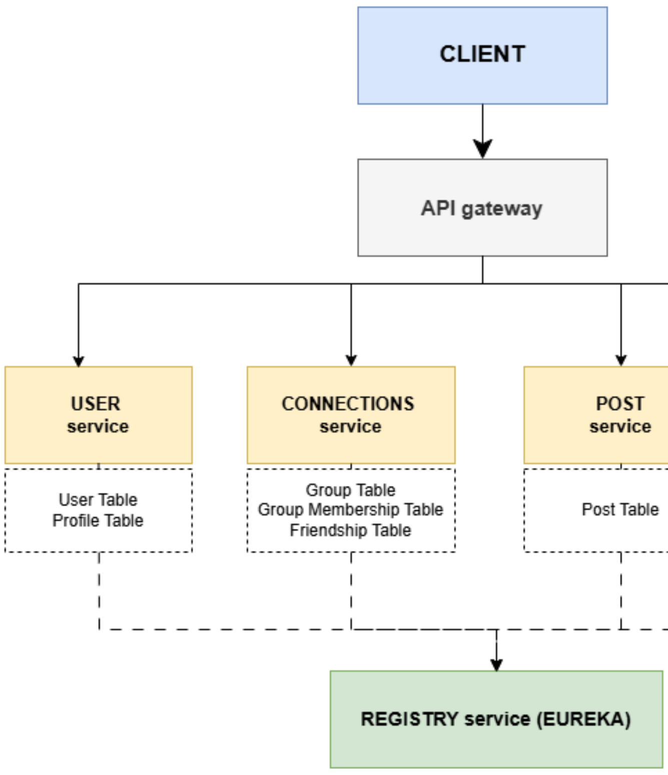
Project Assignment Report

Internet Applications Design and Implementation 2024/25

Magdalena Augustyniak 71787
Sonia Becz 72167

1. System Architecture



While designing our system architecture for the purpose of the project we defined the structure and responsibilities of the four microservices (User, Post, Pipeline, and Connections). In each of them, we designed and implemented database tables shown on the block diagram and developed APIs for the appropriate management. While dividing the task into microservices, we were determined to ensure proper separation of concerns to follow the principles of microservices. To efficiently route the client requests to the backend services, we configured the API Gateway. To enable service discovery, we set up and configured the Eureka Service Registry for dynamic service registration and discovery and so we implemented the aforementioned service registration and service discovery in each microservice and the API Gateway.

2. User stories

The user stories that were completed:

1. As an unregistered user, I want to access the picastlo social network homepage to see the first page of the public timeline of the system.
3. As an unregistered user, I want to access the picastlo social network homepage to sign in to the system and see the homepage of the system.
4. As a registered user, I want to access the picastlo social network homepage to see the homepage and see the first page of my own timeline with my posts, the public posts, my friends post, and the posts of the groups I am registered in.
6. As a registered user, I want to access the picastlo social network homepage and select my profile to see my own posts and my own pipelines.
8. As a user, I want to access the picastlo social network homepage to see the first page of the list of users of the system.
10. As a user, I want to search the list of users by username and name.
11. As a user, I want to select a user from the list of users and see their profile.

3. Description of the Application and Microservice

USER SERVICE

a) Resources

- <u>User:</u> Stores user information such as username and hashed passwords
- <u>Profile:</u> Contains additional user details like their profile avatar and bio connecting it to the proper user by userId

b) Main Operations (API)

Retrieving user and profile information based on provided details.
Examples:

● *GET /users*	- Retrieves all users
● *GET /users/{username}*	- Retrieves user based on provided username
● *GET /users/{id}*	- Retrieves user based on provided userId
● *GET /users/current*	- Retrieves current user
● *GET /users/profiles*	- Retrieves all user profiles
● *GET /users/profiles/{id}*	- Retrieves profile based on provided userId

c) **Security Rules**
- /users/current - requires authentication
- all the other endpoints are open to all.

CONNECTIONS SERVICE

a) **Resources**
- <u>Group:</u> Represents user groups for organizing shared content and posts. Contains information like group name and description.
- <u>GroupMemberships</u>: Manages the relationships between users and groups, storing groupId and userId.
- <u>Friendships:</u> Tracks friendships between users, storing userIds indicating a friendship.

b) **Main Operations (API)**
Retrieving friends and group members based on provided details.
Examples:
- *GET /connections/friendships/{username}*
  - Retrieves all friends of user with provided username
- *GET /connections/groups/{name}*
  - Retrieves group based on provided name
- *GET /connections/groups/{id}*
  - Retrieves group based on provided id
- *GET /connections/groups/{groupName}/members*
  - Retrieves all members of a given group (identified by name)
- *GET /connections/memberships/{username}*
  - Retrieves all groups that a user with provided username is a member of

c) **Security Rules**
It is only accessed from the user profile, where the user is already authenticated so no additional rules are enforced

POST SERVICE

a) **Resources**
- Post: Stores user-generated posts, including text content, image, and associated pipeline.

b) **Main Operations (API)**
Managing user posts.
Examples:

- *GET /posts/public_feed*
  - Retrieves all public posts
- *GET /posts/{username}*
  - Retrieves posts of a user with provided username
- *POST /posts/new*
  - Creates new post
- *DELETE /posts/{id}*
  - Deletes a post based on its id

c) **Security Rules**

- /posts/public_feed is open to all (public access).
- All other endpoints require authentication.

PIPELINE SERVICE

a) **Resources**

- Pipeline: Represents transformation pipelines created or used in the Picastlo GUI

b) **Main Operations (API)**
Retrieving user and profile information based on provided details.
Examples:

- *POST /pipelines*
  - Creates new pipeline
- *GET /pipelines/owner/{username}*
  - Retrieves pipeline owned by a user with provided username
- *GET /pipelines/{id}*
  - Retrieves pipeline based on provided id
- *PUT /pipelines/{id}*
  - Updates pipeline based on provided id
- *DELETE /pipelines/{id}*
  - Deletes pipeline based on provided id

c) **Security Rules**
It is only accessed from the user profile, where the user is already authenticated so no additional rules are enforced

SECURITY

The security implementation in the microservices architecture leverages Spring Security to provide token-based authentication and capability-driven access control. This approach ensures fine-grained authorization and robust protection for the system's resources. The security framework is designed to manage user authentication, session management, and resource access control effectively. Key elements include:

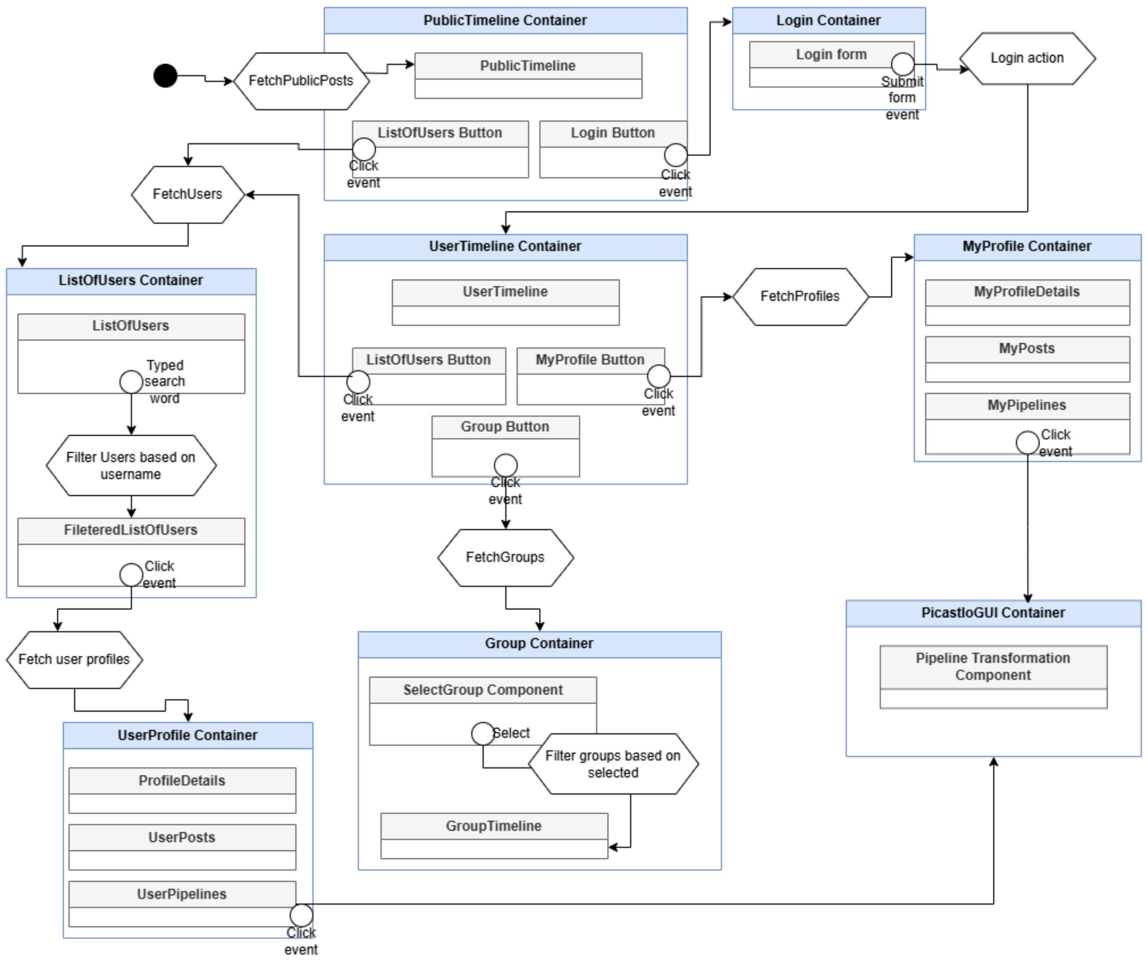
- **Authentication Provider**: The DaoAuthenticationProvider integrates with the MyUserDetailsService to authenticate users based on their credentials. Passwords are securely hashed using the BCrypt algorithm.
- **JWT Token Authentication**: A JWTAuthenticationFilter validates incoming JWT tokens, extracts user details and capabilities, and populates the SecurityContext. The tokens encode user details and their associated capabilities, enabling stateless session management. For unauthenticated users, a default set of capabilities is assigned, allowing limited access to public resources.

Authorization is enforced at multiple levels. Endpoint-level access control is defined in the SecurityConfig class, where rules dictate public and protected endpoints. Beyond endpoint-level security, the system employs capability-based access control, where user permissions are represented by capabilities encoded in JWT tokens. Capabilities link resources to specific operations, such as READ, CREATE, or ALL, ensuring granular and resource-specific authorization.

The capability-based model is central to the security design. Each capability consists of a resource identifier and an associated operation. Upon authentication, capabilities are generated and embedded in the JWT token. Guests are assigned default capabilities, such as READ_PUBLIC for global resources, while authenticated users typically receive broader access (ALL). During resource access, the system validates user capabilities through the capabilitiesService. This validation process ensures that a user’s assigned capabilities meet or exceed the required operations for the requested resource, adhering to a predefined hierarchy where operations are ranked (e.g., NONE < READ_PUBLIC < READ < WRITE < ALL).

The security workflow begins with user authentication via the /login endpoint, where credentials are validated, and a JWT token is issued. Each subsequent request is processed through a filter that validates the token and extracts user capabilities, populating the SecurityContext with this information. When accessing a protected resource, Spring Security annotations trigger methods in the capabilitiesService to confirm that the user has the necessary permissions.

IFML DIAGRAM



The above IFML diagram illustrates user stories to be implemented as a part of the project. The containers are implemented as pages in React and the hexagons show actions, which are mapped into Redux actions and used in the client-side application.
The data flow goes from the public timeline, where a user can see public posts and be redirected to either lookup a list of users and search users based on their username (and possibly go to their profile) or to login. After logging in, they can see their own posts, their friends' posts and posts in the groups they are members of. From there, they can go to their profile page, look up the list of users (what was already described on the public homepage) or go to the page where they can select a group that they are a part of and see all the posts inside. After selecting a pipeline from a profile, a user can use picastlo to transform images. We did not manage to implement all of it.

<u>Examples of detailed mapping into React and Redux:</u>

- Public Timeline Container -> Homepage page
    - React Components:
        - Timeline: Displays the public posts timeline
        - LoginButton: Redirects to the sign-in form
    - Redux actions:
        - FETCH_PUBLIC_POSTS: Fetches public posts
        - LOGIN: Initiates login flow

- Login Container -> Login page
- UserProfile Container -> UserProfile page

4. Assessment

TOPIC	ASSESSMENT
System Architecture (Micro-services)	85%
Internal Architecture of Components (Layered Architecture, DAO, DTO, internal Services)	60%
Architecture of Client Application (React/Redux/Async/OpenAPI)	50%
REST API of Application	90%
REST API of Services	90%
Use of OpenAPI	80%
Seed Data Used	100%
Secure Connection between Application and Services	70%
Completeness of Tests	10%
Tests of security policies	15%

5. Use of AI tools

AI tools were used strategically throughout the development process of the project to enhance efficiency, generate high-quality output, and troubleshoot technical challenges. We used AI tools to debug some issues, if no other solution was working, in both client-side and server-side code. Examples include interpreting error messages, identifying misconfigurations in API endpoints and clarifying build issues in the Spring framework or React. Another way we employed AI tools was for seeding data into the datatables. It provided an easy and effective solution for generating realistic and human-like seed data for users, profiles, posts, groups and pipelines. In some cases, we consulted AI to explain complex concepts, such as distributed access control, JWT token generation or microservice architecture. AI was also used to assist in styling the React application, ensuring a professional and consistent UI design. When we decided on a specific look and style of the website, AI helped to maintain such look throughout the process.
However, AI was just a help, not the main work agent. We designed and implemented the components of the project, only consulting AI in some matters or limiting time-consuming, repetitive tasks. If any, we couldn't claim ownership of seeded data, as it’s solely generated by AI tools, making it synthetic, artificial data.

Positive Aspects of Using AI Tools:

- Increased efficiency - on creating seed data and problem-solving
- Higher quality outputs - generated human-like data

Negative Aspects of Using AI Tools:

- Over-reliance risk
- Lack of context - sometimes AI lacked context, requiring some more manual work

6. Conclusions

The whole process of creating the project equipped us with an experience in designing and implementing microservice-based architecture using Spring Boot. We learned how to manage inter-service communication through APIs, a service registry (Eureka) and an API gateway. While designing tables, we were able to understand the idea behind the domain-specific databases for microservices. The security part was something very new for both of us, so by working on it, we gained the knowledge of how to implement distributed access control using JWT tokens and capabilities to ensure safe data flow between the client and server.

The client-side application forced us to enhance our skills in React and to learn Redux, maintaining a professional look of the page (which we think we achieved).
Integrating the client-side application with server-side microservices using OpenAPI also provided some practical knowledge. The whole process also taught us how to incorporate different diagrams into the design of the application, and how they can positively influence the work of the team.

Challenges
Overall the project was a challenge in terms of complexity and time consumption. Coordinating multiple microservices with dependent functionalities was confusing, especially while debugging inter-service communication. Maintaining clear boundaries and keeping the functionalities working correctly was also not an easy task to achieve. However, in terms of the server-side, the biggest challenge we faced was implementing security. We had to put a significant effort into that and it took a huge amount of time to achieve it.
Another thing is, we had to carefully manage the time, to balance the work on all aspects of the project with other outside commitments, which at the end turned out to be a difficulty.

Positive Aspects
Nevertheless, the project provided valuable hands-on experience in full-stack development and microservices systems. We learned about advanced tools, frameworks and best practices in both front-end and back-end development. The amount of work encouraged productive teamwork and division of tasks to meet deadlines.

Negative Aspects
The ambitious scope of the project led to challenges in managing deliverables within the allocated time.

Lastly, we are proud of what we were able to achieve, as it cost us a lot of time, effort and stress, and the things we didn't manage to complete are a learning experience for the future of how we could distribute the work and manage the big, complex project better.