Project documentation for the portfolio exam

Tim Knüttel (5123101) Adrian Stelter (5123031)
Laurin Neubert (6824016) Jakob Hämmelmann (5123113)

February 15, 2025

1 Introduction of the Project Topic

1.1 Domain

The domain of our project is a social media platform like X (formerly Twitter), Threads or Bluesky. Our goal for the project is to create a platform that enables users to interact with one another through means of text posts. The main objective here is to give users the opportunity to interact not only by posting their own posts, but also to give feedback through likes and comments. Tags help the users to find posts based on interests or topics.

1.2 Use Cases

1.2.1 Posts

The central and most important use case for our project is the ability for users to post text posts. When posting, users should be able to fill their post with content, also it should be visible which user posted at what exact time.

1.2.2 Replies & Likes

Users should be able to interact with posts directly by replying to and "liking" them. This gives users the ability to converse on the platform and discuss their favorite topics. Additionally, replying keeps all the different posts together, so there can be an easy understanding of which post is a new topic and which post was written in response to an already existing topic.

1.2.3 Tags

Tags can be used to link a topic to the created post. By searching for posts tagged with the desired topic, users can easily find posts that are relevant to what they wish to converse about. This is designed to increase the usability and also user retention.

2 Description of the Software Architecture

The hexagonal architecture is generally divided into two main components: adapters and the application. In our case, the adapter component consists of an API and a persistence adapter, as our application does not rely on any external APIs.

The API adapter contains the web controllers responsible for handling API calls and requests, several helper classes in the utils directory that provide functionalities such as caching and AuthorizationBinding, as well as the models used within the API. The persistence adapter also has its own models, as each adapter in a hexagonal architecture should depend only on its own defined models. Additionally, each model has a dedicated repository responsible for storing and managing entities.

The application component consists of three main elements. First, the domain component, where the business logic resides. This includes mappers that handle conversions between the models of both adapters and the corresponding domain models. Additionally, there are predefined classes with methods for generating specific models, which are useful for testing the application. Second, the ports, which contain the interfaces for communication between the API and persistence adapters. In a well-structured hexagonal architecture, the application should only interact with the adapters through these interfaces to ensure a clear separation of concerns. Lastly, the service directory, another key part of the application component, where the core business logic is implemented. Here, parsed requests from the web controllers are processed and forwarded to the appropriate repositories in the persistence component.

One of the most challenging aspects of implementing a hexagonal architecture is designing and maintaining each component as truly independent. It is easy to unintentionally couple components, for example, by directly using repository methods within the corresponding services. Adhering strictly to the architectural principles ensures better modularity, maintainability, and testability of the application.

3 Explanation of the API technology

For this project, a WebAPI architecture was chosen due to its efficiency, scalability, and seamless integration capabilities. The choice of an API technology is a critical decision when designing a social media platform. The API must facilitate efficient data retrieval, support large-scale user interactions, and integrate seamlessly with various front-end clients. This report evaluates the suitability of WebAPI compared to gRPC and GraphQL, providing justification for choosing WebAPI for this project.

3.1 Justification for Choosing WebAPI

WebAPI is chosen as the primary API technology for the social media platform due to its simplicity, broad compatibility, and ease of maintenance. Unlike gRPC, which is optimized for internal microservices, or GraphQL, which introduces complexity in caching and performance, WebAPI provides a balance of usability, scalability, and security for a public-facing application.

3.2 Advantages of WebAPI for a Social Media Platform

The following advantages make WebAPI the preferred choice for this project:

3.2.1 Wide Adoption and Compatibility

WebAPI is universally supported across web and mobile platforms, ensuring seamless integration with front-end technologies such as React, Angular, and native mobile applications.

3.2.2 Scalability and Caching

The stateless nature of WebAPI allows for easy horizontal scaling, which is crucial for handling large numbers of concurrent users. HTTP caching mechanisms can be leveraged to optimize performance, reducing server load for frequently requested data such as user profiles and public posts.

3.2.3 Ease of Development and Maintenance

WebAPI follows a straightforward request-response model using HTTP methods (GET, POST, PUT, DELETE), making it easy to implement and understand.

3.2.4 Security and Authentication

WebAPI works well with authentication protocols such as OAuth 2.0 and JWT (JSON Web Tokens), ensuring secure user access and data protection. API keys and token-based authentication mechanisms allow fine-grained access control, which is essential for managing user permissions and privacy settings.

3.3 Limitations and Trade-offs of WebAPI

While WebAPI is well-suited for this project, it does come with certain limitations:
Over-fetching/Under-fetching: Unlike GraphQL, WebAPI may return more data than needed or require multiple requests to gather related data.
Less Flexible Queries: WebAPI endpoints provide fixed response structures, which may not be as flexible as GraphQL's dynamic querying capabilities.

Strict API Versioning: As the platform evolves, explicit versioning will be necessary to maintain backward compatibility with older clients.

3.4 Conclusion

WebAPI is the most appropriate choice for implementing the social media platform due to its simplicity, broad adoption, and efficient handling of CRUD operations. While gRPC excels in microservices communication and GraphQL provides flexible queries, WebAPI offers a balanced approach that ensures scalability, ease of integration, and security for a public-facing application. The limitations of WebAPI can be managed with best practices such as structured endpoint design and efficient caching strategies, making it the optimal choice for this project.

4 Implementation Details

As a framework we chose Quarkus and therefore RESTEasy. Previous experience with Quarkus is a benefit for the members of our project. Furthermore, from a future-proofing perspective, Quarkus is ready for the use as a base for micro-services, meaning the application being split into multiple small parts.
Quarkus is a lightweight, high-performance framework specifically optimized for cloud-native environments, ensuring efficient resource utilization. Its compatibility with various frontend technologies and mobile applications makes it particularly flexible. Additionally, Quarkus enhances developer productivity with built-in support for API development, authentication, and persistence, optimizing the entire development process.
To implement mapping we went with MapStruct. We choose it because it is a modern approach to mapping as well as being generated instead of manually written, this reduces the possibilities of user error in the implementation.
For authentication, we chose to implement a custom middleware that interacts with the THWS Authentication System and takes the Bearer token it generates. This decision relies on the ever increasing popularity of using bigger providers like Google or GitHub as the sole authentication source.
For our caching strategy. we found that a validity period of five minutes is sufficient for our use cases. Five minutes are enough to reduce the load of requests to the server by a considerable amount (especially when requesting likes for a post), if the application needs to be scaled up. On the other hand, a caching time of five minutes allows users to always see a relatively current state of all responses. This is especially critical for editing posts after they have been published, as a simple edit could drastically change the meaning of the content. This caching time offers the best compromise between reducing network load and offering the latest data.

5 Testing Strategy

Our testing strategy includes different levels to ensure the functionality, security, and scalability of the application. The tests are divided into two main categories: unit tests and integration tests.
Unit tests ensure that individual modules, particularly services and mappers, function correctly in isolation. These tests run in an in-memory environment without a real server to provide quick feedback on code changes.Integration tests verify the interaction of multiple modules, particularly between the domain layer and ports.
The implementation of tests covers all key components of the application. Unit tests validate individual methods, while integration tests check API endpoints and their interactions. Overall, this strategy guarantees a stable, high-performance, and error-free application. By combining different testing methods, it ensures that both individual modules and the entire system function reliably.

6 Learning Outcomes and Reflection

6.1 Improving Planning Process

The planning process is the most important part of the early development stages. By underestimating it's value, we had to recreate the entire repository, because we had already created a project before making a final decision on the API technology we wanted to use. This not only cost time but also a significant amount of planning, as we needed to keep the integrity of commits, so `git blame` works correctly (this is important for evaluating the process later).

6.2 Utilising Git To It's Full Potential

As most of the group had never worked with a team through Git, there was a steep learning process. This especially showed in the beginning, when a couple of mistakes were made because of lacking experience with version control systems and Git in particular. Additionally, utilising milestones in general and issues more broadly could have improved communication and efficiency. We were able to see this towards the end of the project, when we started to create more issues to document and communicate bugs we found. Through those issues, a direct line of communication (like meetings) were needed less, because all the information was found in the appropriate section.

6.3 Internal communication

A positive learning from the project was the weekly meetings we held, as this kept everyone in the loop of what exactly the state of each task is. Addition-

ally, this routine made helping each other solve technical problems easier, as presenting our issues to different sets of eyes often brought a new perspective to the problem. This helped improving our code and efficiently solving problems.

6.4 Standards

Starting from the beginnning, we implemented some standards for this project. These included conventional commit messages, a common code style and protecting the `main` -branch (thus requiring multiple approvals for pull requests). This helped finding errors, documenting changes in a concise manner and reducing the number of `style` commits (changing only the formatting of the code).

6.5 Conclusion

All in all, the project was a success in our view, as we not only learned the ins and outs of a WebAPI, but also coding in a team, using Git and solving complex problems.

Usage of Generative AI

This article was drafted and refined using GPT-4o based on an outline containing related information. The GPT-4o output was reviewed, revised, and enhanced with additional content. It was then edited for improved readability and active tense.