Projektrapport

Development of a Manga Tracking and Community Platform

Author: William Vesterberg & Alexander Kommel

Course: LerniaSJPI23

Date: 28/01-25

<u>Sammanfattning</u>

This report documents the development of a web-based platform designed to help users organize, track, and discuss manga and other serialized media. The platform addresses the challenge of managing reading progress and discovering new titles in an increasingly accessible digital landscape. Key features include review threads, reading progress tracking and community-driven discussions.
The frontend leverages React with TypeScript, React Query for efficient data fetching, and Axios for HTTP requests. The backend is built with Spring Boot, utilizing Spring Security for OAuth2 authentication and MySQL for relational data management. The report highlights the technical decisions, challenges faced, and lessons learned during the project.
The platform successfully demonstrates how modern web technologies can be combined to create a user-friendly and scalable application. Future improvements include real-time features and comprehensive automated testing.

<u>Abstract</u>

Denna rapport beskriver utvecklingen av en webbaserad plattform för att organisera, spåra och diskutera manga och andra former av serier. Plattformen är byggd med React (TypeScript) och Spring Boot och erbjuder funktioner såsom att recensera och kommentera serier, spårning av läsprogression och personliga insikter. Viktiga tekniska beslut, såsom användningen av React Query för datahämtning och Spring Security för OAuth2-autentisering, lyfts fram. Projektet demonstrerar effektiv användning av moderna webbteknologier för att skapa en skalbar och användarvänlig applikation, med framtida förbättringar som inkluderar realtidsfunktioner, historik och automatiserad testning.

Innehållsförteckning

1. Introduction

1.1. What is our project?

In an age where manga and other series are so easily accessible, it can be challenging to keep track of what you've read, what you thought about it, and whether it's worth reading in the first place. To address this challenge, our project aims to develop a web-based platform dedicated to helping users organize, evaluate, and discover manga and other serialized works.

The platform will allow users to **search for specific series**, **track their reading progress**, and **save favorite titles** to a personal library. By **rating** and **reviewing** each series, readers can share their insights and foster discussions with the community. On each series' dedicated page, users can post reviews and initiate threaded comments, creating an interactive environment where opinions and recommendations can flourish.

Additionally, the platform allows the users to create custom lists to organize titles. Ultimately, the goal is to build a user-friendly site that helps users track their reading progress while fostering a vibrant community.

The main features include:

- **Review and Comment**: Post reviews and have threaded discussions about each series.
- **Track and Update**: Save manga, rate them, and update reading status or chapter progress.
- **Custom Lists**: Create and manage specialized lists to organize manga however you like.
- **Search and Discover**: Browse a vast library of titles or find specific ones via targeted search.

1.2. Why is this project relevant?

While many websites offer basic reading lists or a single review score, few provide an integrated system for both tracking progress and engaging in detailed discussions. A dedicated platform that allows users to log each chapter read, rate and review titles, and then exchange feedback with fellow readers fills this gap by making it easier to decide what to start next. With in-depth tracking, personalized insights (future functionality), and a community-driven review process, users can form more informed opinions about their reading choices and stay motivated to explore new series.

1.3. What will the report cover?

This report focuses primarily on the **techniques** that made development more efficient, highlighting **React** but maybe more specific about **React Query** and its data caching capabilities (useQuery, useMutation, and useInfiniteQuery), how **Axios** facilitates HTTP requests, and the integration of infinite querying with an **infinite scroll** component. we will also discuss **React Router DOM** for client-side navigation.

On the **back-end**, the report explores **Spring Security** (including an OAuth2 flow for GitHub login), the use of **pagination** with Spring's Page and Pageable classes, and the rationale behind choosing an **SQL database (MySQL)** to manage the multiple relationships among users, series, reviews, and other entities.

Together, these decisions illustrate how various tools and architectural choices combine to create a secure, scalable, and user-friendly platform.

1.4. Limitations

While the platform delivers core functionalities, certain features and improvements were deprioritized due to time constraints. Below is a summary of the limitations:

1. **History Features**
   Users will not have access to a history or activity log to track their past interactions on the platform. This includes the inability to view previously written reviews, comments, or other engagement activities.
2. **Notifications**
   The platform does not include a notification system. Users will not be alerted about new comments, replies, reviews, or other relevant activities.
3. **Automated Testing**
   Automated testing for both the frontend and backend has not been implemented. This decision was made to prioritize the development of core features within the project timeline.
4. **Static Pages**
   The platform will not include additional static pages such as a "Contact Us" page, "About Us" page, or similar informational pages. The only static page available is the landing page, which serves as the entry point for logging in.
5. **Concrete Statistic Feedback**
   Initially, we planned to include a statistics section on the user profile page. This feature would have provided visual feedback (e.g., diagrams or charts) on user preferences, such as the types of series they typically engage with or complete. However, this feature was deprioritized in favor of more critical functionalities.

Why These Limitations?
These limitations were a conscious trade-off to ensure the timely delivery of the platform's core functionalities. While these features could enhance the user experience, they were deemed non-essential for the initial release and could be considered for future updates.

---

2. Background and Goals

2.1. Project Goals:

Project Objectives

- **Create a user-friendly platform** where readers can track, rate, and discuss manga or other serialized media.
- **Implement core features**, including review threads, reading progress tracking, and personalized insights.
- **Ensure a secure, scalable back end**, using Spring Security for authentication and SQL for relational data management.

Personal Development Goals

- **Expand front-end expertise** by leveraging React Query, Axios, and React Router DOM to manage data fetching, caching, and navigation.
- **Strengthen back-end skills** with Spring Boot, focusing on RESTful API design, OAuth2 authentication, and database relationships.
- **Enhance overall full-stack proficiency** by integrating front-end and back-end concepts into a cohesive, real-world application.

2.2. Technological Stack:

Frontend

- **React (with TypeScript)**: Chosen for its component-based architecture, large ecosystem, and improved type safety compared to plain JavaScript.
- **React Query**: Simplifies data fetching, caching, and synchronization with the server, reducing boilerplate and improving performance.
- **Axios**: A lightweight HTTP client for making GET, POST, and other requests to the back-end API.
- **Tailwind CSS**: A utility-first CSS framework that speeds up styling with minimal custom CSS.
- **React Router DOM**: Provides client-side routing for a more seamless single-page application experience.

Backend

- **Spring Boot**: Offers a robust, convention-over-configuration framework for building scalable REST APIs quickly.
- **Spring Security**: Manages authentication and authorization; supports OAuth2 logins (e.g., GitHub, Google). User sessions are maintained via the standard **JSESSIONID** mechanism, ensuring authenticated requests as users navigate the site.

- **Hibernate/JPA**: Simplifies database interactions via object-relational mapping.
- **MySQL**: A reliable SQL database well-suited for handling relational data (e.g., users, series, reviews).

2.3. Project Planning and Preparation:

To ensure an efficient development process and to stay aligned with our project goals, we began by creating wireframes and a Work Breakdown Structure (WBS). These preparatory steps helped us identify the most critical pages and functionalities, forming the backbone of our project.

- **Wireframes:**
  We designed wireframes in figma to visualize the core layout and user flows for the platform. These served as a blueprint for the front-end development and ensured consistency across the user interface.
- **Work Breakdown Structure (WBS) in FigJam:**
  Using FigJam, we broke down the project into manageable components by identifying the most important pages and functionalities. This process allowed us to:
  - Define clear milestones.
  - Break out the core functionalities, such as tracking progress, reviewing manga, and managing user lists.
  - Prioritize tasks based on project dependencies and complexity.
- **GitHub Integration:**
  From the WBS, we created Product Backlog Items (PBIs) directly in GitHub to streamline our workflow.

  This structured planning process ensured that the project was well-organized from the start, enabling us to deliver a cohesive, feature-rich application within the given timeframe.

3. Front End Analysis

3.1. Why React?

React’s component-based architecture allows for the creation of reusable UI elements, making it easier to build and maintain dynamic single-page applications (SPAs). Its virtual DOM ensures efficient updates, and the extensive React ecosystem (libraries, tools, and community support) provides a solid foundation for rapid development.

Key Advantages:

- Modular Components: Reusable building blocks for UI.
- Virtual DOM: Optimizes rendering performance.
- Large Community: Wealth of documentation, libraries, and troubleshooting resources.

We chose React for our project due to its combination of familiarity and the opportunity to deepen our understanding of the library. Having worked with React before, we are comfortable with its component-based architecture, which promotes modularity and reusability, making it easier to build and maintain our dynamic application.

3.2. React Query

React Query simplifies server state management by handling data fetching, caching, and automatic re-fetching without needing to manually write useState and useEffect hooks. It also supports features like stale-while-revalidate and polling, helping our UI stay in sync with our back end effortlessly.

Why React Query?

- Caching: Reduces redundant network calls by storing fetched data.
- Automatic Refetching: Keeps data fresh without manual triggers.
- Asynchronous Logic: Centralizes fetch-related state (loading, error) more cleanly than manual hooks.

Key Uses in This Project:

Manga Data Retrieval:

- Fetching paginated manga data using useInfiniteQuery to enable infinite scrolling in search and discovery pages.
- Displaying detailed information for individual manga, such as synopsis, authors, genres, and user reviews.

Reading Progress Tracking:

- Syncing the user's reading progress with the server, including current chapter, reading status, and personal ratings.

User Library Management:

- Loading saved manga lists and user-created custom lists.
- Synchronizing library data when items are added, updated, or deleted using invalidateQueries to refresh relevant queries.

Reviews and Comments:

- Fetching reviews and threaded comments for individual manga titles.
- Supporting upvotes, downvotes, and replying to reviews or comments while keeping the data fresh with automatic refetching.

Search and Filtering:

- Powering dynamic search queries and filtering options across various pages.
- Keeping the search results cached while enabling real-time updates for new inputs.

3.3. React Query: Mutations

React Query's useMutation simplifies server-side state changes like creating, updating, or deleting data. Unlike useQuery, which focuses on fetching and caching, useMutation handles asynchronous write operations efficiently while keeping the UI responsive and synchronized with the server.

Why useMutation?

- Optimistic Updates: Improves user experience by temporarily updating the UI before receiving the server's confirmation.
- Error Handling: Provides structured error management to handle failed requests gracefully.
- Flexibility: Offers onSuccess, onError, and onSettled callbacks for custom behavior after operations.

Key Uses in This Project:

- Submitting Reviews: Use useMutation to send POST requests for new reviews. On success, trigger a refetch of the relevant series' reviews to display the new data instantly.
- Updating Reading Progress: Apply useMutation for PATCH requests to update chapter progress or reading status. The UI is optimistically updated while the server processes the request.
- Managing Custom Lists: Enable users to create or delete lists using POST and DELETE mutations. Use callbacks to ensure related data, such as the user's library, remains synchronized.

3.4. Axios

Axios is a popular HTTP client that automatically parses JSON, offers straightforward request/response interceptors, and robust error handling compared to the native Fetch API.

Key Uses in This Project:

- We created a centralized Axios instance where:
    - credentials were set to true to ensure cookies (JSESSIONID) are automatically included in every request.
    - A baseURL was defined to standardize all API calls and eliminate repetitive setup.
- This instance was used for all API calls across the project, simplifying implementation and improving maintainability.
- Axios error handling was implemented using interceptors, allowing us to catch and handle errors consistently, such as showing appropriate error messages.

3.5. Other Tools

- **Tailwind CSS:** A utility-first CSS framework that speeds up styling and promotes a consistent design language.
- **React Router DOM:** Provides client-side routing for SPAs, allowing seamless navigation without full page reloads.
- **TypeScript:** Enhances type safety and reduces bugs by catching errors at compile time.

---

4. Backend Analysis

4.1. Why Spring Boot?

Spring Boot offers a convention-over-configuration approach, enabling rapid setup of RESTful web services. It also benefits from the Spring ecosystem, which includes libraries for security, data access, and more.

Key Advantages:

- **Rapid Development:** Built-in starters simplify configuration.
- **Robust Community:** Access to extensive documentation and support.
- **Easily Scalable:** Suited for both small projects and enterprise-level applications.

We chose to use Spring Boot because we had the opportunity to work with it in a previous course, which gave us a strong foundation to build upon. Its convention-over-configuration approach and robust ecosystem made it seem like a better choice compared to alternatives like Node.js or Django. Spring Boot's seamless integration with tools like Spring Security and Hibernate/JPA, as well as its scalability and extensive community support, reinforced our decision. We're happy with this choice as it allowed us to develop a secure, reliable, and maintainable back end efficiently.

4.2. Spring Security (JSESSIONID)

Spring Security is the cornerstone of our authentication and authorization system, enabling us to build a secure, reliable back end. In this project, we rely on the JSESSIONID mechanism to manage sessions server-side and have implemented OAuth2 login for streamlined user authentication.

Implementation Details:

- **Session Management:**
  We utilize Spring Security's standard isAuthenticated condition on almost all routes, which ensures that only authenticated users can access protected resources. This mechanism relies on the JSESSIONID stored in HTTP-only cookies, which are automatically included in subsequent requests after a user logs in. This approach eliminates the need for manual token management on the front end while ensuring a secure session lifecycle.
- **OAuth2 Login:**
  We have implemented OAuth2 login with GitHub as the authentication provider.
  - **User Details Retrieval:** When a user logs in via GitHub, we retrieve their details from the OAuth2 principle provided by Spring Security. This includes information such as the OAuth ID, email, and username.
  - **Database Integration:** If the user has a public email and username, we save these details to our database to associate them with their account.

    - **Handling Missing Details:** If the user's GitHub account lacks a public email or username, they are redirected to a custom form where they must provide this information manually. Once submitted, the details are stored in the database, completing the registration process.
- **Securing Our Routes:**
    - To ensure only authorized users can access or modify data, we validate each request against the current user's context.
    - Using Spring Security's SecurityContextHolder, we extract the authenticated user's OAuth ID and compare it to the OAuth ID of the data owner in the database. This step ensures that users can only access or manipulate data they own, preventing unauthorized access to sensitive information.

4.3. Pagination with Page/Pageable

To handle large data sets (e.g., 75,000 manga records), Spring Data JPA provides Page and Pageable, enabling efficient, paginated queries.

How It Works:

- **Pageable Interface:** Accepts parameters (page number, size, sorting) that the repository can use for paginated queries.
- **Page Object:** Returns relevant metadata (total pages, total elements) alongside the current batch of data.

Benefits:

- **Scalability:** Large result sets are broken into manageable chunks.
- **Performance:** Reduces memory usage and load times.

4.4. Database Design

We chose MySQL for its reliability, performance, and strong support for handling relational data, making it an ideal choice for managing the complex relationships between users, manga, reviews, and other entities in our project. Its compatibility with Hibernate/JPA ensured a smooth development process by leveraging features like schema generation, relationship mapping, and query optimization.

Our database is structured to represent these relationships effectively:

- **Users Table:** Stores user-specific data such as OAuth provider details, email, and profile information. It is linked to several tables, including saved_manga, lists, reviews, and comments, to manage user interactions across the platform.

- **Manga Table:** Central to the schema, it contains detailed information about manga, including titles, publication dates, scores, and synopsis. It connects to related entities such as images, manga_authors, manga_genres, and reviews.

- **Images Table:** Stores image URLs for manga, linking back to the manga table via a foreign key relationship.

- **Authors and Manga_Authors Tables:** authors stores details about authors, while manga_authors bridges the many-to-many relationship between authors and manga.

- **Genres and Manga_Genres Tables**: genres stores genre names, and manga_genres maps the many-to-many relationship between genres and manga.

- **Saved_Manga Table:** Tracks manga saved by users, capturing additional details like personal ratings, current chapter, and reading status. A unique constraint ensures no duplicate entries for the same user and manga.

- **Lists and List_Manga Tables:** lists stores user-created custom lists, while list_manga bridges the many-to-many relationship between lists and saved manga.

- **Reviews and Comments Tables:** reviews allow users to share their thoughts on manga, including ratings and spoilers, while comments enable threaded discussions on reviews.

---

5. Challenges and Solutions

5.1. Frontend Challenges:

1. **Handling Large Datasets Efficiently**:
   Managing large datasets like manga libraries required effective caching and pagination to prevent performance bottlenecks. Using React Query allowed us to simplify caching and manage paginated data seamlessly with useInfiniteQuery.
   Solution: React Query's features, such as automatic refetching and background updates, enabled us to keep the UI responsive and synced with the back end, even for large-scale data.

2. **Revamping the Application's Design for a Modern and User-Friendly Experience**:
   The initial styling of the application looked unappealing to us and failed to provide a modern and user-friendly experience. The light colors, semi-inconsistent design elements, and mismatched fonts made the interface less engaging and visually distracting for users.
   Solution: We revamped the styling by switching to a **dark color scheme**, creating a sleek and visually appealing look. This change not only improved aesthetics but also enhanced readability and reduced eye strain, particularly for users who prefer dark mode. Additionally, we standardized the typography by using the **Montserrat font**, replacing the previously inconsistent fonts. This brought uniformity to the design and elevated the professional appearance of the application.

3. **Challenge: Managing the Logged-In User State Across the Application**
   One of the initial challenges we faced was effectively managing and distributing the logged-in user's information across the application. After a user logged in via GitHub OAuth, we needed to fetch and store their details in a way that allowed us to access this information globally. This was particularly important because certain API calls required the user's ID to perform actions such as saving manga, updating reading progress, or posting reviews. However, relying on the user's ID in the frontend posed security risks, as it exposed internal database identifiers.

   Solution: Implementing a Custom Auth Provider with React Context
   To address this, we created a custom authentication provider using React's createContext and useContext hooks. This allowed us to establish a global user context that could be accessed from any component in the application. Here's how we implemented it:

   Fetching User Data: After a user logs in via GitHub OAuth, we retrieve the user's details from the OAuth provider (e.g., GitHub). Using these details, we then fetch the corresponding user record from our database. While this approach initially felt like it might involve redundant work, it ensured that we had a consistent and reliable way to access the user's information across the application.

   Global User Context: We wrapped our application's pages in the custom AuthProvider, which handles user authentication state. This provider checks whether a user is logged in, manages the logout functionality, and stores the user's details in the useAuthContext. This context could then be accessed from anywhere in the app, eliminating the need to repeatedly fetch user data.

Enhancing Security: Initially, we relied on the user's database ID in the frontend for certain API calls. However, we later realized this was not secure, as it exposed internal database identifiers. To improve security, we transitioned to using the oauthId (retrieved from the authenticated principal) in the backend. This allowed us to validate user requests by comparing the oauthId from the context with the corresponding value in the database. As a result, the frontend no longer needed to handle sensitive user IDs, and all critical validation was performed securely on the server side.

Outcome:
This solution not only streamlined user management across the application but also significantly improved security by minimizing the exposure of sensitive data in the frontend. While the initial implementation involved some redundancy, it provided a robust foundation for handling user authentication and state management.

5.2. Backend Challenges:

1. **Finding the Right API**:
   One of the biggest challenges was finding an API that provided a comprehensive manga library with minimal restrictions. Many free APIs lacked the depth we required or imposed limitations on the number of requests or accessible features.
   Solution: We chose **Jikan API**, an unofficial API for MyAnimeList, because it offered a vast library of manga with detailed metadata and sufficient flexibility for our project. However, Jikan API is community-maintained and not intended for commercial use. If we decide to scale our product commercially, we would need to transition to a paid API like **MangaDex**, which offers greater reliability and commercial rights but comes at a cost.

2. **Paginating and Serving Large Datasets**:
   With thousands of manga records in the database, serving data efficiently was critical to maintaining performance. Querying and transferring large datasets without pagination would lead to slower load times and excessive memory usage.
   Solution: Using Spring Data JPA's Page and Pageable classes, we implemented efficient pagination at the database level. This approach limited the amount of data sent to the front end, reducing server load and improving response times.

3. **Getting the security to work with OAuth2.0 and github with spring security**
   We faced challenges adding security to our API, particularly with integrating OAuth2.0 authentication via GitHub. Initially, securing routes and validating user access were complex.
   Solution: By leveraging Spring Securitys standard isAuthenticated method, we secured all routes requiring authentication. To enhance the login process, we implemented a custom success handler that executed specific logic upon successful login. Initially, user access validation was done using database IDs, but we transitioned to a more secure approach. This involved storing the oauthId

from the authenticated principal (retrieved from Spring Security's context) and comparing it with the oauthId of users in the database.
This method improved security by eliminating the need to expose or rely on internal database IDs. The session management was handled seamlessly by Spring Security's use of JSESSIONID, allowing us to maintain secure and efficient authentication flows.

---

6. Reflection and Alternatives

6.1. What Worked Well?

Many aspects of the project worked exceptionally well, thanks to the tools and techniques we chose. Having prior experience with React made it a comfortable choice for the frontend, and pairing it with TypeScript added an extra layer of reliability. While there were occasional hurdles, the familiarity and flexibility of React allowed us to overcome them efficiently.

One of the standout tools was **React Query**, which we had discovered through a video highlighting its caching capabilities. After suggesting it to Alexander, we decided to give it a try, and it turned out to be a game-changer. React Query simplified both data fetching and posting with its useQuery and useMutation hooks. However, the real win was its useInfiniteQuery hook, which seamlessly integrated with the react-infinite-scroll-component library. This combination allowed us to implement infinite scrolling on the browse page, where users can effortlessly scroll through an almost endless list of manga series without overwhelming the system with a single large data fetch. That said, we did encounter one issue with React Query—a classic case of user error. We accidentally reused the same query key for two different fetches, causing cached data to overlap. It took some time to diagnose, but once we identified the problem, it was an easy fix and a valuable lesson in careful key management.
Another tool that proved its worth was **Axios**. While some developers prefer using the built-in fetch API, we found Axios to be more intuitive and feature-rich. Its built-in functionality, such as request/response interceptors and streamlined error handling, made our HTTP requests smoother and more reliable. We experienced zero issues with it throughout the project, reinforcing our decision to use it.

On the backend, **JPA Buddy** was a lifesaver for generating our database entities. It sped up the development process significantly, allowing us to focus on higher-level logic rather than boilerplate code. Although we had to revise our entities a few times as we refined the database schema, the process was seamless. The way JPA Buddy handles table relationships almost feels like magic, making complex joins and mappings effortless.

Finally, the **Figma design process** was both enjoyable and productive. Creating the wireframes and mockups went smoother than expected, and while we ultimately switched to a dark theme for the app, the initial designs provided a solid foundation. The wireframes guided much of our frontend work, ensuring consistency and saving us time during development.

6.2. What Could Be Improved?

While the project was largely successful, there are areas where we could have improved our processes and tools. One notable area was our use of the **Kanban board** for task management. Although we started with good intentions, our workflow didn't always align with the structured approach that a

Kanban board is designed to facilitate.

A significant factor was our tendency to work in intense, co-coding sessions. Instead of breaking tasks into smaller, manageable chunks and moving cards incrementally across the board, we often coded relentlessly until we achieved a major milestone. This meant that multiple tasks were completed in one go, and we frequently dragged cards directly to the "Done" column without reflecting on the intermediate steps. While this approach kept us productive in the short term, it hindered our ability to maintain a clear overview of progress and prioritize tasks effectively.
In hindsight, we could have benefited from more structured planning sessions, such as daily or weekly scrums, to review the board, discuss priorities, and assign specific tasks. This would have allowed us to focus on smaller, well-defined goals and ensure that each task was properly tracked and reviewed before merging into the development branch. Additionally, it would have provided a better sense of direction and helped us allocate time more efficiently.

Another improvement could have been to integrate more frequent code reviews and testing into our workflow. While we did test our features, a more systematic approach—such as writing unit tests or conducting peer reviews—could have caught issues earlier and improved the overall quality of the codebase.

Another area for improvement is the lack of **automated testing** in both the frontend and backend. While we didn't neglect testing entirely, it wasn't a priority during development due to time constraints and the focus on delivering core features before the deadline. In an ideal scenario, we would have adopted a **Test-Driven Development (TDD)** approach, ensuring that each feature was accompanied by corresponding tests.

For the frontend, **end-to-end (E2E) tests** using a tool like Cypress would have been invaluable for verifying user flows and ensuring the application behaves as expected. On the backend, **integration tests** for services and controllers would have helped catch issues early and ensured the reliability of our API endpoints.

We did attempt to write tests for the backend, but we encountered challenges with GitHub OAuth authentication, which complicated the process. As a workaround, we considered testing with security temporarily disabled, but this is something we haven't had the chance to implement yet. While this decision allowed us to focus on delivering functionality, it's an area we plan to address in the future to improve the robustness and maintainability of the application.

6.3. Testing Improvements

One area we did not prioritize sufficiently during the project was testing. Currently, we lack automated tests for both the frontend and backend, which is a key area for improvement. Early in the project, we considered various testing approaches, such as unit tests for individual components and functions, integration tests to ensure different parts of the system work together, and end-to-end (E2E) tests to validate user flows.

However, due to time constraints, we focused on delivering functional code, and testing was deprioritized. For example, while attempting to write backend integration tests, we encountered challenges with GitHub OAuth authentication. To address this, we planned to disable security temporarily during testing but did not implement this solution in time.

Future Testing Improvements
To improve reliability and maintainability, we plan to implement a robust testing strategy:

- **Unit Testing with JUnit and Mockito:**
  Use JUnit to test individual classes and methods and Mockito to mock external dependencies, such as databases or APIs, ensuring isolated and efficient testing.
- **Integration Testing:**
  Leverage Spring Boot Test to validate how system components interact, including API endpoints, database operations, and authentication flows.
- **End-to-End Testing (Frontend):**
  Use **Cypress** to create automated E2E tests for critical user flows, such as logging in, tracking manga, and managing user lists. Cypress will help ensure that the frontend behaves as expected when integrated with the backend.
- **CI/CD Pipeline for Automated Testing:**
  Integrate automated tests into the CI/CD pipeline using GitHub Actions. This ensures tests are run automatically on every push, helping catch bugs early in the development process.

  Example that shows a GitHub Actions-config:

```
name: Java CI
on: [push]  # Kör när kod pushas till repositoryt
jobs:
 build:
   runs-on: ubuntu-latest  # Kör på en virtuell Ubuntu-maskin
   steps:
    - uses: actions/checkout@v2  # Checka ut koden
    - name: Set up JDK
      uses: actions/setup-java@v2
      with:
        java-version: '17'  # Ange Java-version
    - name: Run tests
      run: mvn test  # Kör JUnit-tester med Maven
```

6.4. Lessons Learned:

This project was an invaluable learning experience, both technically and personally. On the technical side, it deepened my understanding of **full-stack development**, particularly the integration of frontend and backend systems. Working with tools like **React Query**, **Axios**, and **Spring Boot** allowed us to explore advanced features and best practices, such as efficient data fetching, caching, and secure authentication. We also gained a better appreciation for the importance of **database design** and how tools like **JPA Buddy** can streamline entity management and relationships.

One of the most significant lessons was the importance of **planning and process**. While our intense co-coding sessions were productive, they sometimes led to a lack of structure in task management. This highlighted the value of **agile practices**, such as regular stand-ups and incremental task tracking, to maintain a clear overview of progress and priorities. Additionally, the challenges we faced with testing reinforced the need to prioritize **automated testing** early in the development process, even if it means sacrificing some speed in the short term.

On a personal level, the project taught us the value of **adaptability** and **problem-solving**. Whether it was debugging unexpected issues, refactoring code for better security, or rethinking our approach to user authentication, we learned to approach challenges with patience and creativity. Collaborating closely together also improved our **teamwork** and **communication skills**, as we constantly shared ideas, divided tasks, and supported each other through obstacles.
Finally, the project underscored the importance of **balancing ambition with practicality**. While we had many ideas for features and improvements, we had to make tough decisions about what to prioritize to meet deadlines. This experience taught us how to focus on delivering a **minimum viable product (MVP)** while leaving room for future enhancements.

---

7. Conclusion

Project Summary

This project aimed to address the challenge of organizing, tracking, and discovering manga and other serialized media in an increasingly accessible digital landscape. By developing a web-based platform, we created a space where users can search for series, track their reading progress, rate and review titles, and engage in community discussions. Key achievements include the implementation of core features such as review threads, reading progress tracking, and personalized insights, all supported by a secure and scalable backend.

The use of modern tools like **React Query** for efficient data fetching, **Spring Boot** for robust backend services, and **OAuth2** for secure authentication allowed us to deliver a user-friendly and performant application. Despite the challenges we faced, the project successfully demonstrated how thoughtful design and the right technologies can come together to solve real-world problems.

Future Work

While the platform is functional and meets its core objectives, there are several areas for future improvement and expansion:

Enhanced Authentication: Adding support for more OAuth providers, such as Google or Facebook, would make the platform more accessible to a broader audience.

Real-Time Features: Implementing real-time updates, such as live notifications for new reviews or comments, could enhance user engagement and create a more dynamic experience.

Testing and Reliability: Introducing comprehensive automated testing, including end-to-end tests with **Cypress** and backend integration tests with JUnit and Mockito, would improve the application's robustness and maintainability. Even better would be if we made the test automated on for example every push.

Performance Optimization: Further optimizing database queries and exploring tools like **GraphQL** for more efficient data fetching could enhance performance, especially as the platform scales.

Additional Media Support: Expanding the platform to support other forms of serialized media, such as web novels or comics, would increase its versatility and appeal.

User Analytics: Providing users with more detailed reading statistics and personalized recommendations could help them discover new series tailored to their preferences.
By addressing these areas, the platform can continue to evolve and provide even greater value to its users.

8. References

- **Youtube tutorial for OAuth2:** https://www.youtube.com/watch?v=JUOSHhbbBOY
- Spring Boot. (n.d.). **Spring Boot Documentation**. [online] Available at: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/ [Accessed 28 January 2025].
- Spring Security. (n.d.). **Spring Security Documentation**. [online] Available at: https://docs.spring.io/spring-security/reference/ [Accessed 28 January 2025].
- Tailwind CSS. (n.d.). **Tailwind CSS Documentation**. [online] Available at: https://tailwindcss.com/docs [Accessed 28 January 2025].
- React Query. (n.d.). **React Query Documentation**. [online] Available at: https://tanstack.com/query/latest/docs/react/overview [Accessed 28 January 2025].
- Axios. (n.d.). **Axios Documentation**. [online] Available at: https://axios-http.com/docs/intro [Accessed 28 January 2025].
- React Router DOM. (n.d.). **React Router DOM Documentation**. [online] Available at: https://reactrouter.com/en/main [Accessed 28 January 2025].
- React. (n.d.). **React Documentation**. [online] Available at: https://react.dev/ [Accessed 28 January 2025].