LibManager Project Report

Group Number 5

GROUP MEMBERS

Alvaro, Matt, Carlos, Julius

I. INTRODUCTION

A. Product Ideas and Vision

We wanted to create a project with real-world impact. Because we love books, we chose to develop a digital library management system in Software Engineering (INF 2900). Our goal was not only to deliver a technical solution but also to practice agile methods. We aimed to strengthen our understanding of software engineering practices.

This is our product statement:
FOR librarians and library users **WHO** need a convenient way to manage and access books online, **THE** LibManager system provides a simple, intuitive platform **THAT** allows librarians to easily manage book inventory and users to check availability, borrow, and return books online. **UNLIKE** a traditional paper-based system or overly complex commercial solutions, LibManager focuses on essential features and user-friendliness. This focus enables even non-technical users, such as children or older adults, to use the system effectively. **OUR PRODUCT** promotes digital access to knowledge. It improves operational efficiency for libraries. It also encourages a reading culture by lowering access barriers and modernizing library workflows.

Regarding our learning vision, this project helped us deepen our knowledge of agile methodologies and full-stack development. We also improved our collaboration skills. We gained a better understanding of real-world software engineering principles.

B. Product Goals and Description

Initially, we created user stories to define our primary goals for the LibManager system. We established the following key objectives:

- Provide an intuitive user experience for both users and librarians.
- Enable easy management of book inventory (Create, Read, Update, Delete - CRUD operations).
- Implement user authentication and role-based access control (User, Librarian, Admin).
- Include book borrowing and returning functionality.
- Allow online checking of book availability.
- Offer user profile management, including avatars.

We successfully developed a working library management system. This system met all the initial goals we set. Key features include user authentication, a carousel-style book catalog, and functions for borrowing and returning books. Users can manage their profiles, including changing email addresses and passwords. Administrators have specific management capabilities.

The main objective of this library system is to offer a simple and intuitive user experience. This improvement might seem small, but its impact is significant. It facilitates access to reading for people with little technological experience, such as children and older adults. It also simplifies the learning process for library staff, allowing efficient system use from the start. A notable feature is the ability to check book availability online, which saves users unnecessary trips. Overall, this system improves accessibility, streamlines library operations, and encourages reading among a wider audience.

II. PROJECT CONTENT

A. User Stories, Scenarios, Personas, and Feature List

1) User-Centric Design Approach: Our user research process involved creating user stories and personas based on potential users like librarians and library patrons. We held weekly meetings over three sprints, each lasting 2-3 hours. These activities helped us identify the most critical functionalities for LibManager. We focused on needs derived directly from these user profiles and scenarios. The user stories and personas document is available in the Appendix.

The personas and user stories directly shaped the final feature list. For example, the user Ragnar ”The Viking” needed a simple way to see book availability, which led us to implement a straightforward availability feature. Similarly, Astrid Vinterstad, the librarian, required efficient book management tools, prompting us to include features like book addition and editing.

- **Add New Book:** Allows librarians to add new books to the system catalog.
- **Search for Book:** Enables users, especially librarians, to quickly find books.
- **View Book Availability:** Provides a clear way for users to see if a book is available.
- **View Book Details:** Allows users to see comprehensive information about a book.
- **Browse by Category:** Enables users to explore the book collection by categories.
- **List All Books:** Provides an option to display all books in the catalog.
- **Edit Book Information:** Allows librarians to modify existing book details.
- **Remove Book:** Enables librarians and admins to remove books from the catalog.

- **Implement Simple User Interface:** Focuses on creating a clean and easy-to-navigate interface.

B. *Design, Architecture, and Technologies*

1) System Architecture Overview: The LibManager system uses a multi-tier architecture, illustrated in Figure 1. The frontend, built with React, communicates with the backend API. The backend, developed using Django and Django REST Framework, handles business logic and data processing. The MySQL database stores application data like user information and book details. Requests from the user's browser go through the frontend to the backend API, which then interacts with the database to retrieve or modify data before sending a response back.

2) Architectural Patterns: We adopted a Client-Server architectural pattern, separating the user interface (client) from the business logic and data storage (server). The backend implements a variation of the Model-View-Template (MVT) pattern, common in Django applications. This pattern separates data models, presentation logic (views/templates), and request handling. The frontend uses a component-based architecture with React, promoting modularity and reusability of UI elements. Communication between the client and server relies on a RESTful API design, using standard HTTP methods for resource manipulation. These patterns together provide a structured and maintainable system design.

3) Technology Stack and Rationale: Our technology stack consists of several components chosen for specific reasons.
For the backend, we selected Python with the Django framework and Django REST Framework (DRF). We chose Django because it facilitates rapid development and includes helpful built-in features like an Object-Relational Mapper (ORM) and user authentication. DRF simplified the creation of our RESTful API endpoints.
For the frontend, we used React with TypeScript and CSS. We chose React because it excels at building dynamic and interactive user interfaces. TypeScript added static typing, improving code reliability and developer collaboration. Vite served as our build tool.
We selected MySQL as our database. We chose MySQL because it is a widely used relational database management system with good performance and various visualization tools that aided development.
We used Git for version control to manage our codebase collaboratively. Our API design followed REST principles, ensuring statelessness and using standard HTTP methods. The database schema, defined in `models.py`, includes key models like `User`, `UserProfile`, and `Book`, with defined relationships between them.

4) System Design and Interaction: The overall system design focuses on separating concerns between the frontend, backend, and database. User interactions, like searching for a book or attempting to borrow one, start in the React frontend. The frontend sends API requests to the Django backend. The backend processes these requests, applying business logic (e.g., checking borrow limits) and interacting with the MySQL database via Django's ORM.

For instance, when a user borrows a book, the frontend sends a request to the '/borrow/' endpoint. The backend view verifies the user's authentication and checks if the book is available and if the user has exceeded their borrow limit (we set it to 3 books). If successful, the backend updates the book's status and the user's borrowed books list in the database. The backend then sends a confirmation response to the frontend, which updates the UI accordingly. This interaction pattern ensures data consistency and applies necessary rules server-side.

5) Database Schema Design: The database schema forms the foundation for storing application data. Figure 2 illustrates the main models and their relationships. The key models include `User` (using Django's built-in user model), `UserProfile` (extending the User model with additional details like avatar), and `Book` (containing information like title, author, ISBN, availability). Relationships, such as the one-to-one link between `User` and `UserProfile`, and the many-to-many relationship implicitly representing borrowed books, are defined in `models.py`.

C. *Implementation (Key Features)*

This section describes the implementation of key features at a high level. We focus on their functionality and the rationale behind important decisions. The component interactions for a book borrowing request are detailed in Figure 3. Throughout development, we made several design decisions to ensure LibManager delivered a simple, robust, and scalable experience. We prioritized a clean interface and essential features. Role-based access control was implemented early to secure workflows.

1) Feature: Authentication & Authorization: We implemented user authentication allowing users to register, log in, and log out. The backend uses Django's session-based authentication and includes CSRF protection for security. We defined specific permissions (e.g., `IsAdminUser`, `IsAdminOrLibrarian`) to enforce role-based access control. Only librarians and administrators can access certain management features. The frontend integrates with the backend authentication system using a React Context (`AuthContext.tsx`) to manage user sessions and control UI visibility based on user roles. This ensures secure access to different parts of the application.

2) Feature: Book Catalog and Management: Users can view a list of books, search for specific titles or authors, and filter results. Librarians and administrators have privileges to add new books, edit existing book details, and remove books from the catalog. The frontend provides forms for these actions. These forms communicate with corresponding

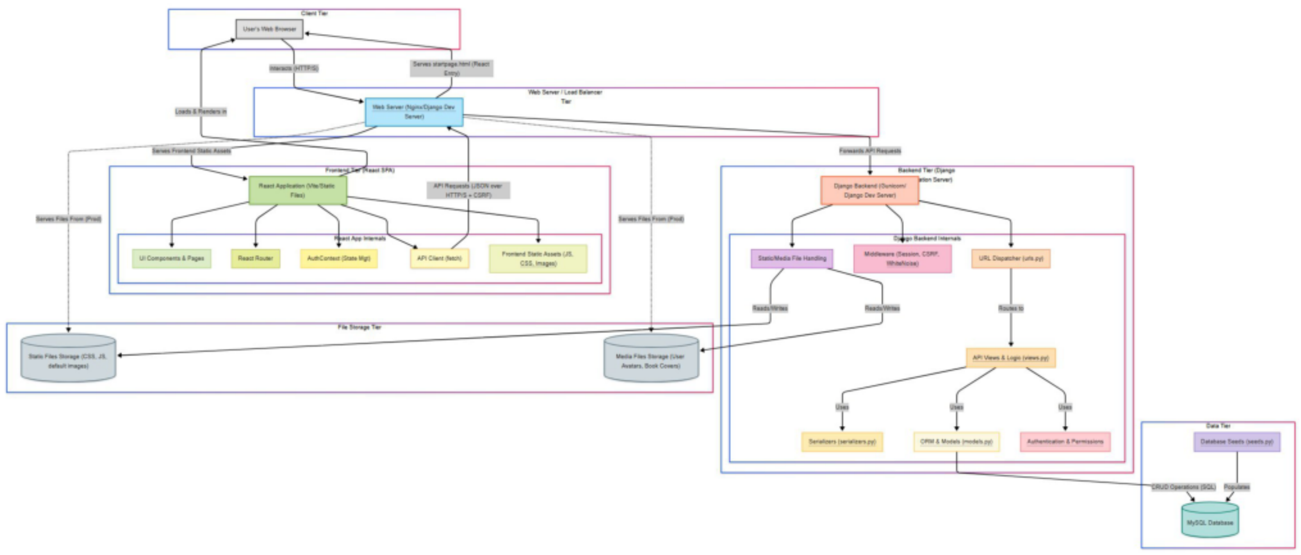

Client Tier
User's Web Browser
Interacts (HTTP/S)
Serves startpage.html (React Entry)
Loads & Renders in
Web Server / Load Balancer Tier
Web Server (Nginx/Django Dev Server)
Serves Frontend Static Assets
Forwards API Requests
Frontend Tier (React SPA)
React Application (Vite/Static Files)
API Requests (JSON over HTTP/S + CSRF)
Serves Files From (Prod)
Serves Files From (Prod)
React App Internals
UI Components & Pages
React Router
AuthContext (State Mgt)
API Client (fetch)
Frontend Static Assets (JS, CSS, Images)
Backend Tier (Django Application Server)
Django Backend (Gunicorn/ Django Dev Server)
Django Backend Internals
Static/Media File Handling
Middleware (Session, CSRF, WhiteNoise)
URL Dispatcher (urls.py)
Reads/Writes
Reads/Writes
Routes to
File Storage Tier
Static Files Storage (CSS, JS, default images)
Media Files Storage (User Avatars, Book Covers)
API Views & Logic (views.py)
Uses
Uses
Uses
Serializers (serializers.py)
ORM & Models (models.py)
Authentication & Permissions
Data Tier
Database Seeds (seeds.py)
CRUD Operations (SQL)
Populates
MySQL Database

Fig. 1. Overall System Architecture of LibManager.

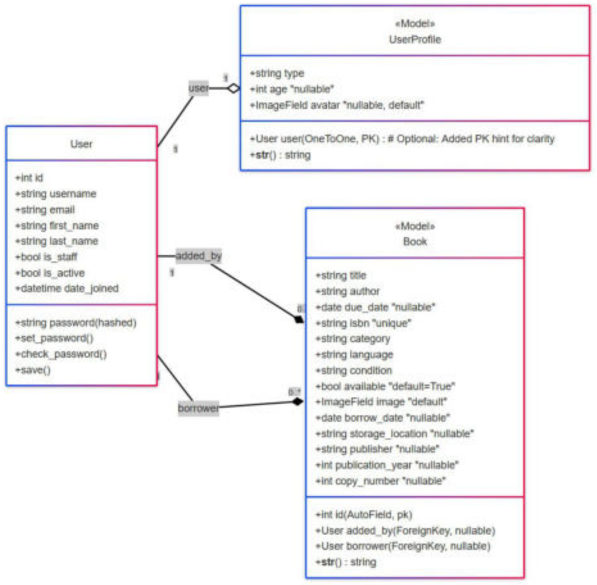


Fig. 2. Class Diagram illustrating the relationships between User, UserProfile, and Book models.

backend API endpoints. We chose a carousel view for the book catalog to make browsing engaging and visually appealing. Color-coded availability indicators provide quick visual feedback to users. We decided to include a default image for books without covers and managed image URLs through the `BookSerializer`. This feature set provides comprehensive book inventory management capabilities. A challenge was efficiently handling image uploads and storage, which required careful backend configuration.

3) Feature: Borrowing & Returning: Registered users can borrow available books up to a defined limit (e.g., three books). The backend logic enforces this limit and updates the book's status to 'borrowed'. Users can view their currently borrowed books and their due dates on their profile page. Returning a book updates its status back to 'available' and removes it from the user's borrowed list. The workflow for borrowing a book is illustrated in Figure 4. This feature simulates a core library function within the digital system.

4) Feature: User Profile Management: Users can view and update their profile information, including email address and password. We added an avatar selection feature. This allows users to choose a profile picture from a predefined set, aiming to enhance user engagement and personalization. The frontend provides the interface for viewing and editing profile details. The backend handles data validation and updates the `UserProfile` model. Ensuring secure password updates and validating email changes were important considerations during implementation.

5) Feature: Admin User Management: Administrators have access to a user management panel. They can view a list of all users, promote regular users to the librarian role, and delete user accounts if necessary. These actions are restricted to users with administrative privileges. This feature ensures that system administrators can manage the user base effectively. One challenge was to design a simple yet effective interface for these administrative actions.

D. Non-Functional Attributes

1) Performance and Responsiveness: Performance refers to how quickly the system responds to user actions. We aimed for low response times for both the user interface and API calls. During development, we observed that database queries and page loads were generally fast for typical operations like searching or viewing book details. For example, API

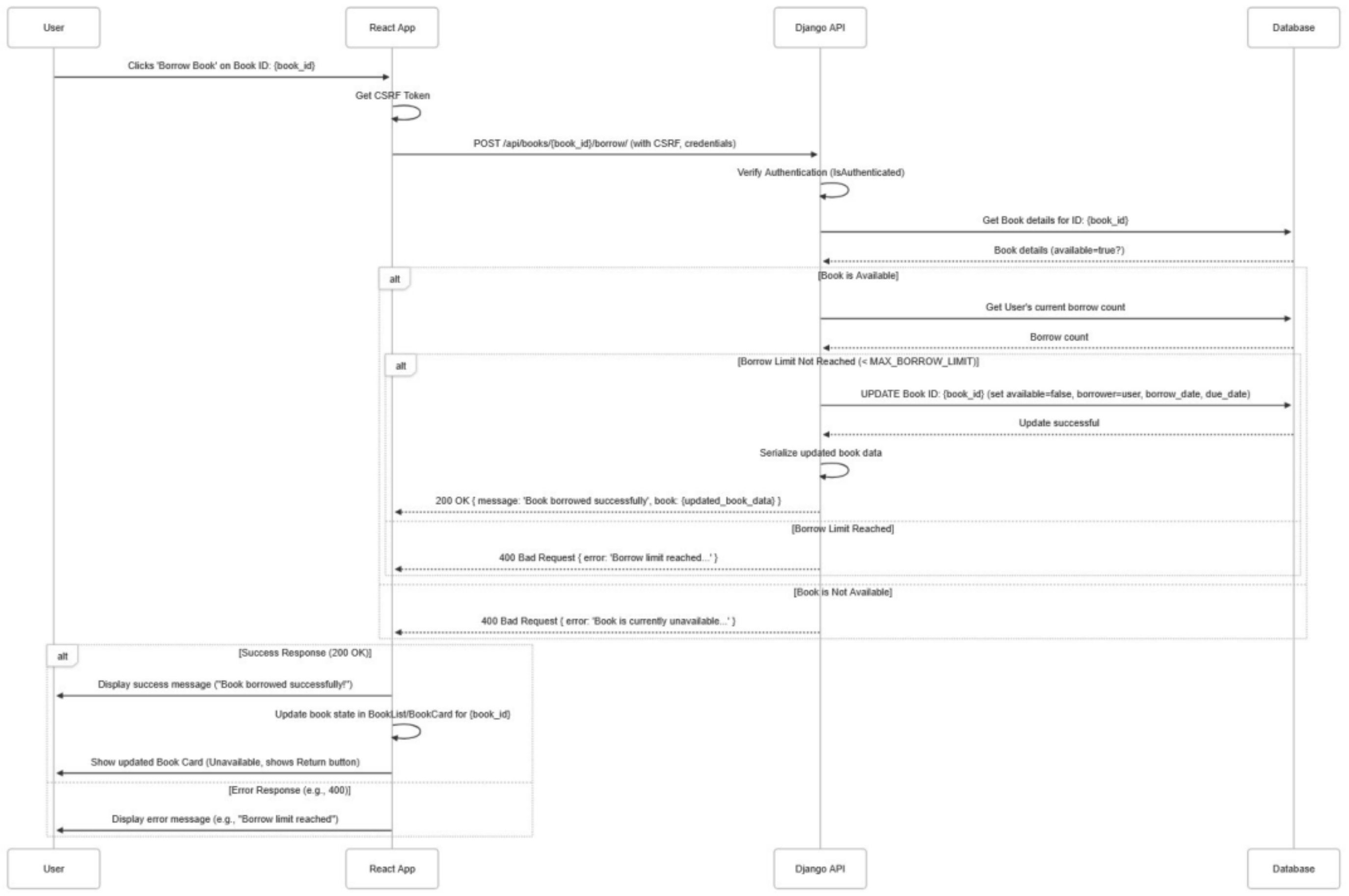

User
React App
Django API
Database
Clicks 'Borrow Book' on Book ID: {book_id}
Get CSRF Token
POST /api/books/{book_id}/borrow/ (with CSRF, credentials)
Verify Authentication (IsAuthenticated)
Get Book details for ID: {book_id}
Book details (available=true?)
alt
[Book is Available]
Get User's current borrow count
Borrow count
alt
[Borrow Limit Not Reached (< MAX_BORROW_LIMIT)]
UPDATE Book ID: {book_id} (set available=false, borrower=user, borrow_date, due_date)
Update successful
Serialize updated book data
200 OK { message: 'Book borrowed successfully', book: {updated_book_data} }
[Borrow Limit Reached]
400 Bad Request { error: 'Borrow limit reached...' }
[Book is Not Available]
400 Bad Request { error: 'Book is currently unavailable...' }
alt
[Success Response (200 OK)]
Display success message ("Book borrowed successfully!")
Update book state in BookList/BookCard for {book_id}
Show updated Book Card (Unavailable, shows Return button)
[Error Response (e.g., 400)]
Display error message (e.g., "Borrow limit reached")
User
React App
Django API
Database

Fig. 3. Sequence Diagram showing the interaction between Frontend, Backend, and Database during a book borrowing request.

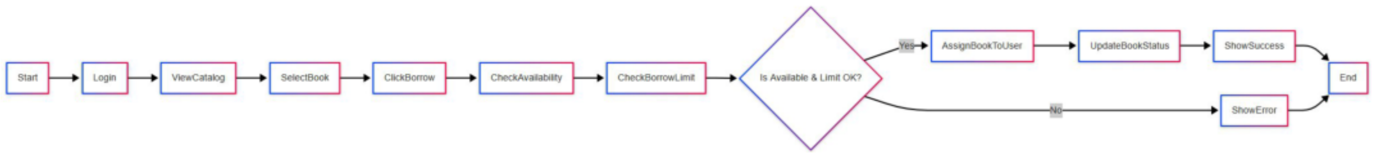


Fig. 4. Flowchart depicting the user process for borrowing a book.

responses for fetching book lists typically completed in under 200 milliseconds in our local testing environment with a moderate dataset. While we did not conduct formal load testing, the responsiveness felt adequate for the intended use case.

2) Security and Privacy: Security involves protecting the system from threats and unauthorized access. We implemented several security measures. Passwords stored in the database are hashed using Django's built-in password hashing functions. This prevents plain-text storage. Our database seeding script (`seeds.py`) also hashes passwords before insertion and tests the database connection. We enforced role-based access control to restrict access to sensitive functions based on user roles (User, Librarian, Admin). CSRF protection is enabled by default in Django, mitigating cross-site request forgery attacks.

3) Reliability: Reliability refers to the system's ability to perform its intended functions correctly and consistently over time. We addressed reliability through several measures. Comprehensive error handling mechanisms were implemented in both the frontend and backend. These mechanisms manage unexpected situations gracefully and provide informative feedback to users. Database integrity is maintained through constraints, such as ensuring the uniqueness of ISBNs using `unique=True` in our Django models. This prevents data corruption and ensures consistency. Furthermore, our development process included writing unit tests for critical components. These tests helped identify and fix defects early. We also focused on achieving low coupling between system modules to minimize the ripple effect of changes and improve overall system stability. These efforts resulted in a reliable system.

4) Usability: Usability measures how easy, efficient, and pleasant a system is for users. This was a primary focus during the development of LibManager. We aimed for an intuitive design with clear navigation and prompt feedback to the user. For example, the book browsing interface uses a carousel for visual appeal and easy interaction. Action buttons are consistently styled and labeled to make their purpose clear. We strived to maintain a simple design with straightforward actions. This enables users of varying technical experience to use the system effectively without significant learning overhead.

5) Maintainability: Maintainability refers to the ease with which a system can be modified, corrected, and enhanced. We addressed maintainability by adopting a clear code structure with logical separation of concerns. For instance, Django's MVT pattern in the backend and React's component-based architecture in the frontend promote modularity. We used descriptive names for variables and functions and included comments where necessary to explain complex logic. The use of serializers in Django REST Framework and reusable components in React also contributes to a more maintainable codebase. While we aimed for clean code, we recognize that some areas could be further improved with more consistent coding standards or additional refactoring. Overall, the modular design is crucial for supporting long-term maintenance.

6) Scalability: Scalability is the system's ability to handle increasing amounts of work or its potential to be enlarged to accommodate that growth. Our choice of technologies like Django and MySQL provides a solid foundation for scalability. These technologies are widely used and support various scaling strategies. The RESTful API design, being stateless, also aids scalability by allowing distribution of backend instances. Potential bottlenecks could arise from complex database queries under heavy load or rendering very large datasets in the frontend. Future work to enhance scalability might involve optimizing database queries, implementing pagination for large lists, and considering horizontal scaling for the backend server.

E. Testing

Our testing strategy involved multiple layers to ensure the quality and correctness of LibManager. We focused on verifying both individual components and complete workflows.

- **Testing Methodologies:**

  1) Backend Testing:

  - *Functional and Integration Testing:* We used Django's ‘APITestCase‘ for end-to-end tests on API views. These tests simulated HTTP requests (GET, POST, PUT, DELETE). They validated complete workflows such as authentication, user management, book catalog operations (CRUD, search, filter), and book borrowing/returning logic. We tested response status codes, data integrity, and session state.
  - *Unit Testing:* We focused on isolating and verifying individual components.
    - **Models:** Ensuring data integrity, default values, method correctness
    - **Serializers:** Validating data serialization/deserialization, field-level and object-level validation rules, and correct handling of read-only/write-only fields.
    - **View Logic:** Testing specific logic within views in isolation, often employing mocks to simulate dependencies like database interactions or external services.
    - **Utility Scripts:** Verifying the correctness of helper scripts, such as the database seeding script (‘seeds.py‘), by mocking file operations and database calls.
  - *Security Testing:* Specific tests were written to verify security mechanisms, including role-based access control (permissions like ‘IsAdminUser‘, ‘IsAdminOrLibrarian‘), authentication requirements for protected endpoints, and CSRF token generation and validation.
  - *Test Coverage*: We used the `coverage` Python module to measure test coverage. The overall backend test coverage is 82%. The `seeds.py` file has 76% coverage. The `views.py` file is at 86% coverage, and `serializers.py` is at 81% coverage, indicating good coverage for core logic.

2) Frontend Testing: For the frontend, we implemented several types of tests to ensure functionality, user experience, and performance. While frontend test coverage was not as extensive as backend coverage, these tests provided valuable checks for key user interface elements and interactions.

- **Component Testing:** We used Vitest along with React Testing Library to test individual React components in isolation. These tests verified that components render correctly, respond to user interactions as expected, and manage their state appropriately. Examples include `Button.test.tsx` to check button rendering and click handlers, and `UserCard.test.tsx` to validate correct display of user information.
- **End-to-End (E2E) Testing:** Cypress was employed for E2E testing. These tests simulated real user scenarios by interacting with the application through the browser. We covered critical user flows like login, registration, and navigation. For instance, `login.cy.ts` in `frontend/tests/endtoend/` tests the complete authentication flow, including successful login, error handling for invalid credentials, and navigation to the sign-up page.

- **Performance Testing:** We created basic performance tests using Vitest to measure execution times of specific JavaScript functions and simulated component rendering. These tests, such as `dataProcessing.test.js` and `rendering.test.js`, helped us establish benchmarks for critical operations and identify potential performance bottlenecks.

3) Test Automation: We developed automation scripts to streamline development and testing processes. These scripts handled tasks like starting backend and frontend servers and running test suites automatically. We created shell scripts, such as `Linux_RUN_ALL.sh` and `Windows_RUN_ALL.bat`, to manage these tasks. Test files are located within the respective `tests` subdirectories of backend applications and frontend components. The automation scripts are in the project's root directory and the `Linux_Start` directory.

4) Testing Reflections and Lessons Learned: Testing was critical for ensuring LibManager's quality and stability. Backend testing with Django's `APITestCase` allowed us to verify core workflows effectively. On the frontend, Vitest and Cypress helped us catch UI and user flow issues early. One challenge we faced was writing comprehensive frontend tests. These often required setting up complex states and mocking backend interactions, which was time-consuming. We also realized the importance of writing tests early in the development cycle. When we delayed testing until after feature development, covering all edge cases became more difficult. Despite time constraints, our testing efforts significantly improved system reliability. Testing reduced the number of bugs and gave us greater confidence during development. In the future, expanding automated test coverage, especially on the frontend, would further improve system quality and reduce manual testing effort.

III. Process Content

A. Agile Development Cycle

1) Scrum Framework Adaptation:

- **Practices Implemented:** We decided to have a weekly meeting with our teacher assistant. In some cases, we held another meeting to coordinate tasks and monitor progress. In this project, we initially planned two sprints of two weeks each. However, we ended up adding a third sprint to complete all work. Regarding functionalities, we created our user stories initially to establish a clear vision for the product's features.

- **Iterations:** In this project we were able to complete it over 3 sprints. The first one lasted two weeks (04/2/25 - 18/2/25), then another one week (27/02/25 - 06/03/25) and a last sprint where we had to extend it longer because we had to coincide with several deliveries of other courses, this sprint lasted two weeks from (13/03/25 - 27/03/25) with this we were able to complete the software part of the project.

- **Collaboration:** Collaboration was managed primarily through Discord for daily communication. This platform enabled quick resolution of minor issues and clarifications. Weekly meetings, especially during the first sprint, helped ensure a balanced workload. Our TA, Ine, played a key role in these meetings by clarifying doubts and providing feedback. During meetings, we discussed progress, and our TA offered suggestions for refining ideas. We evaluated all proposed changes and implemented them when feasible. This iterative feedback and adaptation loop was central to our process.

2) Detailed Development Process: Throughout the LibManager project, we followed an agile development process using the Scrum framework. We divided the work into three sprints. Each sprint had a clear set of goals, backlog items, and tasks.

We maintained a prioritized product backlog in Jira. In Jira, we tracked user stories, tasks, and bugs. Each sprint started with a sprint planning session. In these sessions, we selected the highest-priority tasks, estimated effort, and defined sprint goals.

Sprints and Timeline:

- Sprint 1 (04/02/2025 – 18/02/2025): Focused on setting up the development environment, backend API, and database models.
- Sprint 2 (27/02/2025 – 06/03/2025): Focused on frontend core features, connecting the backend and frontend, and implementing basic CRUD operations.
- Sprint 3 (13/03/2025 – 27/03/25): Focused on advanced features, UI/UX improvements, testing, and bug fixing.

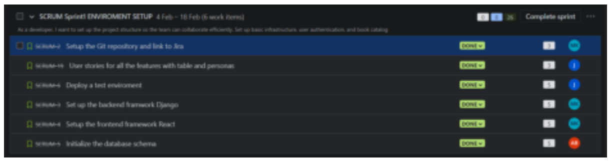
Collaboration Tools: We used Jira to manage issues, tasks, and progress across sprints. Our team communicated through Discord for daily coordination and Git for version control. We followed a simple Git branching strategy, using feature branches when needed and carefully managing merges to avoid conflicts.

Retrospectives and Adaptation: After each sprint, we conducted retrospectives. We reflected on what went well, what could be improved, and how to adjust our approach. For example, after Sprint 2, we identified a need for more frequent commits and better branch management. We implemented these changes in Sprint 3. This structured process helped the team stay organized, adapt to changes, and deliver a functional product within the timeline.

Impact of Scrum Process: This structured process helped the team stay organized, adapt to changes, and deliver a functional product within the timeline.

Sprint Backlog Overview: Screenshots of the sprint backlogs from Jira are included in this section. These provide a clear view of the completed and ongoing tasks across the sprints. The number at the end indicates the difficulty associated with the task.

Progress Overview: Figure 9 shows the cumulative progress made over time, with approximately three main jumps corresponding to our three sprints.

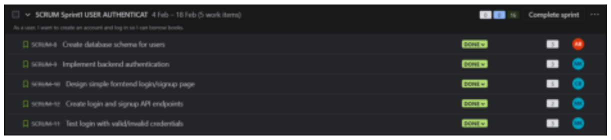


SCRUM Sprint1 ENVIROMENT SETUP 4 Feb – 18 Feb (6 work items) 0 0 26 Complete sprint

As a developer, I want to set up the project structure so the team can collaborate efficiently. Set up basic infrastructure, user authentication, and book catalog

SCRUM-2 Setup the Git repository and link to Jira DONE 3
SCRUM-19 User stories for all the features with table and personas DONE 3
SCRUM-6 Deploy a test enviroment DONE 5
SCRUM-3 Set up the backend framework Django DONE 5
SCRUM-4 Setup the frontend framework React DONE 5
SCRUM-5 Initialize the database schema DONE 5

Fig. 5. Sprint 1: Environment Setup Backlog in Jira

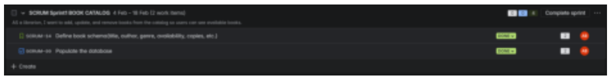


SCRUM Sprint1 USER AUTHENTICAT 4 Feb – 18 Feb (5 work items) 0 0 16 Complete sprint

As a user, I want to create an account and log in so I can borrow books.

SCRUM-8 Create database schema for users DONE 3
SCRUM-9 Implement backend authentication DONE 3
SCRUM-10 Design simple forntend login/signup page DONE 5
SCRUM-12 Create login and signup API endpoints DONE 2
SCRUM-11 Test login with valid/invalid credentials DONE 3

Fig. 6. Sprint 1: User Authentication Backlog in Jira



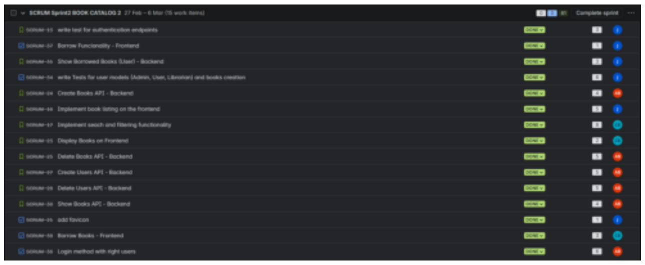
SCRUM Sprint1 BOOK CATALOG 4 Feb – 18 Feb (2 work items) 0 0 4 Complete sprint

As a librarian, I want to add, update, and remove books from the catalog so users can see available books.

SCRUM-14 Define book schema(title, author, genre, availability, copies, etc.) DONE 2
[illegible] Populate the database DONE 2

+ Create

Fig. 7. Sprint 1: Book Catalog Backlog in Jira



SCRUM Sprint2 BOOK CATALOG 2 27 Feb – 6 Mar (15 work items) Complete sprint

[illegible] write test for authentication endpoints DONE
[illegible] Borrow Funcionality - Frontend DONE
[illegible] Show Borrowed Books (User) - Backend DONE
[illegible] write Tests for user models (Admin, User, Librarian) and books creation DONE
[illegible] Create Books API - Backend DONE
[illegible] Implement book listing on the frontend DONE
[illegible] Implement seach and filtering functionality DONE
[illegible] Display Books on Frontend DONE
[illegible] Delete Books API - Backend DONE
[illegible] Create Users API - Backend DONE
[illegible] Delete Users API - Backend DONE
[illegible] Show Books API - Backend DONE
[illegible] add favicon DONE
[illegible] Borrow Books - Frontend DONE
[illegible] Login method with right users DONE

Fig. 8. Sprint 2: Book Catalog Features and Testing Backlog in Jira

Sprint Progress Reflection: As the project advanced, we became more structured in how we organized work. Sprint 1 focused on broader tasks like environment setup. Later sprints, especially Sprint 3, show that we began splitting tasks into smaller, well-defined subtasks. This allowed for more effective progress tracking, clearer ownership, and faster feature integration. This evolution in sprint planning reflects our improved understanding of the development process, Jira usage, and project scope. Each sprint became more focused on concrete deliverables. We increased both granularity and accountability over time.

B. Test-Driven Development (TDD)

Best practice dictates writing tests before implementing features. This approach helps clarify requirements and ensures that the code meets its intended purpose. However, due to time constraints, we did not implement Test-Driven Development (TDD) in this project; we wrote many tests after implementation. Because our project was large with an extensive code base, writing all tests beforehand would have significantly increased development time. Nevertheless, we acknowledge that TDD could have improved code quality and reduced bugs.

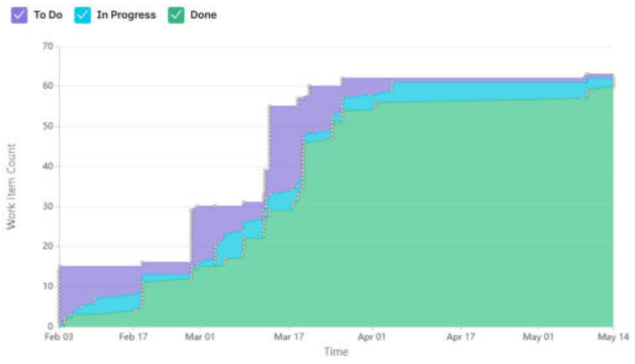


Fig. 9. Cumulative flow diagram of the progress

C. DevOps

1) DevOps Principles and Practices: Our DevOps approach focused on collaboration, automation, and efficient code management. We assigned tasks clearly within the team. For development, we used Git for version control. We primarily worked on feature branches for new functionalities or significant changes to isolate work and allow for parallel development. These branches were then merged into the main development branch after review. In some instances, for minor fixes or when a developer worked solitarily on a well-defined part, we made updates directly to the main branch after careful coordination.

2) Deployment Strategy: LibManager was primarily developed and run in a local environment for this project. A production deployment would involve several key steps. We would containerize the Django backend and React frontend applications using Docker. This process would ensure consistency across different environments. These containers would then be orchestrated using a platform like Kubernetes or deployed to a Platform-as-a-Service (PaaS) such as Heroku or AWS Elastic Beanstalk. For the database, we would use a managed MySQL service (e.g., Amazon RDS, Google Cloud SQL) to ensure reliability, backups, and scalability. A web server like Nginx would be configured to serve the frontend static files and act as a reverse proxy for the Django backend. Nginx would also handle SSL termination and load balancing if required. Continuous Integration/Continuous Deployment (CI/CD) pipelines, potentially using GitLab CI/CD or GitHub Actions, would automate the build, test, and deployment process upon code pushes to the main branch.

3) Code Management and Version Control (Git): We used Git for version control, hosted on a shared repository platform. Our workflow involved creating feature branches for new functionalities or significant changes. Developers worked on these isolated branches. This approach allowed for parallel development without disrupting the main codebase. Once a feature was complete and locally tested, the developer pushed the branch to the remote repository. We then used merge requests (or pull requests) to review changes before merging them into the main development branch (e.g., 'main' or 'develop'). This practice helped maintain code quality and provided an opportunity for peer review. Although we aimed for regular merges, managing merge conflicts occasionally required careful coordination. This branching strategy is a foundational practice for enabling CI/CD pipelines and collaborative software development.

4) Issue and Project Management (Jira): We used Jira for project management. This included organizing our sprints and tracking tasks. Jira helped us maintain a product backlog, plan sprint content, and monitor progress. Issues, bugs, and new features were logged in Jira, providing a central overview of the project's status. During sprint planning, we assigned tasks and estimated effort using Jira's features. When issues arose during development or testing, we used Jira to track them. We also used Discord for immediate communication to notify team members about critical issues or blockers. This process facilitated quick fixes. This combination of Jira for structured tracking and Discord for rapid communication supported our collaborative workflow.

D. Time and Effort Metrics

We logged individual contributions, and team members focused on different project aspects. Alvaro primarily handled backend tasks. These included database creation, project connection, initial seeding script, and CRUD operations for books and users; he also assisted with frontend visibility. Matt managed Jira processes and initialized the Django system. Carlos designed the initial web layout and developed the frontend structure, refining visual details throughout the project. Julius contributed to both frontend and backend, created numerous tests, developed automation scripts, and performed a major refactoring of both codebases near the project's end; most team members also shared Scrum Master duties.

We informally tracked time and effort through Jira task points. We spent a significant portion of the effort on coding, followed by testing and planning. Documentation was an ongoing activity.

IV. Conclusion

Our main achievements in this project include creating a useful and broadly accessible library management system. Key features include role-based access control, which assigns appropriate responsibilities. We also implemented book management for librarians, allowing easy CRUD operations. Our borrowing logic prevents users from borrowing more than three books simultaneously. We successfully developed a product that fully meets all initial objectives. Regarding learning, we gained knowledge of the Django framework and MySQL. We also learned to manage a large project using agile methodologies. Teamwork was a significant learning aspect, teaching us effective task distribution and the importance of individual contributions. Overall, we are proud of our final product. We met our initial goals, experienced valuable learning, and achieved smooth teamwork.

V. Discussion

A. Project Scope and Achievements

From the project's outset, we assessed our abilities and dedicated significant time to planning the application's scope. As a result, we successfully completed and integrated all planned functions. We also added new functions that emerged during development. In conclusion, we developed and implemented all intended functions successfully. Overall, we believe we developed a good product that satisfies all initial requirements.

B. Technology Stack Evaluation

1) Frontend (Why React/TypeScript was a Good Choice): React was a well-suited choice for the LibManager frontend because it facilitated the development of highly interactive and modular user interfaces. Its component-based architecture enabled us to encapsulate UI elements and logic in isolated, reusable units; this improved code maintainability and scalability. This was especially beneficial when implementing complex components such as the book carousel and animated transitions, which we handled using external libraries like `react-slick` and `Framer Motion`.

However, the use of React also introduced some architectural challenges, particularly around state management. As the number of components and interdependencies grew, we needed to adopt a more structured state management approach. We used context providers and, in some cases, considered external libraries. Despite this, React's flexibility and ecosystem allowed us to address these issues effectively. Overall, React proved to be a powerful framework for building the frontend of our application.

2) Backend (Django, Django Rest Framework): Django and Django REST Framework (DRF) offer a robust solution for backend development. Key advantages include rapid development capabilities because of integrated tools like Django's authentication system and Object-Relational Mapper (ORM). The ORM simplifies database interactions significantly. DRF makes the creation of RESTful APIs straightforward and efficient.

On the other hand, both Django and DRF can present a steep learning curve for developers unfamiliar with these technologies, although we received a good introduction to Django by Professor Yu. Nevertheless, integrating the API with the frontend can sometimes pose challenges, particularly in managing request/response formats and ensuring seamless data flow. Despite these potential difficulties, the benefits in terms of development speed and feature richness often outweigh the initial learning investment for many projects.

3) Database (MySQL): For our database, we required a relational system to manage structured data like user profiles and book inventories. The complexity was moderate, involving several interconnected tables. We considered both MySQL and MariaDB. MariaDB, a fork of MySQL, was familiar to some team members from past projects. However, we ultimately chose MySQL. We based this decision on its broader availability of learning resources, extensive documentation, and robust visualization tools like MySQL Workbench, which aided development and debugging. MySQL's widespread adoption also ensures good community support.

4) Integration Challenges: Integrating the frontend and backend components presented several challenges. Ensuring consistent data formats between the React frontend and the Django REST Framework API required careful definition of serializers and API contracts. Managing authentication state across both systems, including token handling and session management, also required meticulous implementation to ensure security and a smooth user experience. Debugging issues that spanned both frontend and backend often took more time, because it required tracing requests and data transformations through multiple layers of the application.

C. Performance and Optimization

We observed generally good performance for LibManager during development and testing. Page load times were typically fast, and API responses for common operations, like fetching book lists or user details, were usually processed quickly. For instance, in our local environment, API calls for book data often completed in under 200 milliseconds. We made some implicit optimizations. For example, Django's ORM can be efficient, and we aimed for simple queries where possible. However, we did not implement advanced optimizations like widespread use of `select_related` or `prefetch_related` for Django queries, which could further improve database interaction speed for complex data retrieval. Potential bottlenecks could arise if the database grows very large, particularly with features like the book carousel rendering many items or complex search queries without proper indexing. Loading multiple high-resolution images for book covers could also impact frontend rendering performance if not managed with techniques like lazy loading or image optimization. While current performance is adequate for the project's scale, these are areas for future optimization if the application were to handle significantly more data or users.

D. Design and User Experience (UX)

Evaluation of the UI design choices (layout, components like `BookCard`, `AvatarSelector`).

The interface selected for the entire application is an intuitive and easy to read interface so that the user can navigate and understand how it works without having to think too much. And thanks to the menu you can easily guide you where you want to go without getting lost. And now what really matters is how to book and view the available books, the carousel format was chosen as it provides a unique and dynamic style, you can see the books in a clear and orderly manner, all this added to the search bar to be able to do if one wishes a more in-depth search. We chose adding flipping cards to show the details as it will make everyithing looks more simple and easy. The books are shown with their cover and buttons to be able to interact with it. The *color code* was mantained simple and clear being blue: modify, red: exit or delete, green: create or update and yellow: return. Another importatn part was our effort to make the website dynamic which allow all type of users to see the changes without the need of reloading Also the diversity we give the user to personalise their profile by changing their avatar.

E. Challenges Faced

1) Technical Challenges: We encountered several technical challenges during development. Designing the API involved creating specific endpoints and establishing a uniform framework for routes, which required careful planning. In the backend, handling the CSRF token for authentication was problematic, initially leading to frequent 403 errors because of missing tokens in API requests from the frontend. Synchronizing API calls with the borrow/return logic, especially managing asynchronous operations and ensuring data consistency, also posed difficulties.

Regarding the database, we faced issues because of the use of reserved SQL keywords in model names, forcing us to rename certain tables. The initial complexity of the `seeds.py` file for populating the database also required significant effort to debug and resolve. In frontend development, managing the `AuthContext` for user authentication state throughout the application was a persistent challenge. Implementing CSS-based flip animations for book cards without disrupting the layout also took considerable trial and error. Furthermore, loading images, such as avatars and book covers, required careful configuration on both frontend and backend to ensure efficient storage and display. Dynamically adapting frontend styling based on backend data also added complexity and development time.

2) Team and Process Challenges: Throughout the development of the project using the Agile framework, we encountered several challenges related to coordination and task allocation. These sometimes impacted the overall efficiency of the team.

Communication was sometimes ineffective, leading to delays in decision-making and occasional misunderstandings about task requirements. We also faced issues with version control. Merge conflicts were frequent, especially when multiple team members worked on overlapping code sections, and resolving them consumed considerable time. Additionally, our use of Git branches was inconsistent initially. In some cases, branches were not created for each distinct issue or feature, which made it difficult to track changes systematically and integrate features smoothly.

Finally, time management was also an issue. Balancing tasks, individual responsibilities, and deadlines proved to be challenging throughout the project.

F. Real World Usability & Future Work

We have not configured a borrow function using a hand scanner, but this could be easily implemented. Instead, we have configured a new and modern system: a user can borrow a book directly in their profile. If the user leaves the library, all books would be scanned automatically via RFID chips inside the books. The scanner would be positioned before the exit that anyone has to pass through (Figure 10). If a book has not been borrowed, an alarm will sound. Regarding limitations, the main limitation is that RFID chips would need to be on every book for this system to work. Regarding limitations, the main thing is that RFID chips should be on every book in order to make it work.

Fig. 10. Book scanner at the entrance of UiT library (Illustration created by the authors using Google Imagen 3).

1) Potential Enhancements: We recognize that our product is not perfect and can be improved. First, we could add a scanner functionality. This would allow scanning a book's QR code to display its details within LibManager for borrowing. Additionally, we could implement automatic invoicing via email for overdue books. Other potential future improvements include advanced search, a reservation system, and automatic bill sending. These were not implemented because of time limits.

G. Testing and Quality Assurance

Our testing strategy effectively identified issues, particularly through backend API tests and frontend component tests. We discovered and fixed several key bugs during testing. These included authentication flaws and data inconsistency in borrowing logic. However, our testing had limitations. We lacked comprehensive automated UI tests. Unit test coverage, while good for the backend, could have been more extensive, especially for edge cases. A key lesson learned regarding Quality Assurance (QA) is the importance of integrating testing throughout the development lifecycle. We primarily tested the system manually with each server start. We created shell scripts to automate starting servers and running existing tests. A CI/CD pipeline would have significantly improved this process. We found that writing tests after the program already exists sometimes felt less effective than TDD. Achieving 100% test coverage was unreasonable for this project because of the significant time investment required. We focused on testing critical functionalities.

H. Code Quality and Maintainability

1) Code Organization and Readability: During development, we tried to maintain a code organization with a clear separation between frontend components and backend method types (e.g., separate files for views and URLs). Moreover, we thoroughly commented the code to enhance understanding. Finally, as mentioned previously, we aimed for modular code to promote reusability and agile development.

2) Complexity Management and Refactoring: Towards the project's end, Julius performed a major refactoring. This addressed several problems. For example, the backend had multiple UserProfile model definitions with different names and poorly named variables. Additionally, multiple copy-pasted definitions for local book, user profile, and user interfaces in the frontend existed; Julius replaced these with imports from central type definitions for backend models. The refactoring improved system cleanliness, and modules can now be reused more easily. This enhances reusability and potential scalability. However, some aspects would not scale well; for instance, the carousel view is unsuitable for displaying millions of books. We initially planned a paginated book view but have not yet implemented this feature because of time constratraints. Implementing a paginated view would not be difficult.

3) Application of Design Principles and Patterns: Although we tried to create clean, reusable code, some areas clearly need improvement. For example, the front-end CSS does not follow a clear structure.

I. Reflection on Agile/DevOps Process

Reflecting on the agile process, we found that dividing work into small tasks and setting short delivery deadlines was beneficial. This approach helped us maintain a constant workflow. Using Git proved very useful in nearly all situations with minimal drawbacks. Finally, regular meetings were highly effective. These allowed us to receive feedback on our work and improve organization.

J. Ethical Considerations & Personal Issues (If Applicable)

1) Data Privacy and Security Measures: Regarding data privacy, we store user information in the database. We use Django's password hashing to ensure data safety by avoiding plain text password storage.

K. Team Reflection and Lessons Learned

The LibManager project gave us valuable experience working as a team under real-world software engineering conditions. We improved our collaboration skills, learned how to divide tasks effectively, and gained insight into managing time and expectations across multiple sprints.

One key lesson was the importance of clear communication, especially when resolving merge conflicts and coordinating work across frontend and backend. We also learned that maintaining consistent code quality across team members required regular code reviews and agreed-upon naming conventions.

From a technical perspective, we deepened our understanding of Django, React, and MySQL, and we became more comfortable working with tools like Git and Jira. We also realized that testing and documentation need to be planned from the beginning to avoid last-minute rushes.

Finally, the project showed us the value of adaptability: we frequently had to adjust our plans, split tasks differently, or revisit implementation decisions based on feedback and time constraints. These lessons will help us in future projects, both academic and professional.

Appendix

A. *Generative AI Declaration*

- **Disclosure of AI Tools:** We leveraged advanced AI technologies such as **GPT-4o**, **o3 mini High**, **Gemini Thinking 01-29**, and **Chocolate**.
- **Compliance:** All work is in strict adherence to UiT guidelines.
- **Usage:** We used AI tools for
  - Programming support
  - error debugging
  - Mermaid.js diagram generation
  - book title generation
  - book cover generation
  - image generation
  - persona story generation
  - grammar check
- **Author Responsibility:** The authors fully assume responsibility for the content and its outcomes.

B. *Group Meetings*

Figure 1 provides a high-level overview of the project's progression through the weeks.

Week 5: Initialization and Framework Familiarization:

- **Idea Generation:** Team members reflected on the optimal application design to develop.
- **Framework Exploration:** The Django framework was selected and time was dedicated to familiarizing the team with its environment.

Week 6: Project Kick-Off and Methodological Framework:

1) **Introduction:**
   - Each participant introduced themselves and shared relevant project experience.
2) **Work Methodology:**
   - **User Stories:** Every feature was documented as a user story accompanied by a corresponding test; detailed documentation is included in the final report.
   - **Sprint Planning:**
     - **Prioritization:** Features were prioritized based on dependency requirements (e.g., user authentication as a prerequisite for subsequent functionalities).
     - **Sprint Structure:** Each sprint was designed to address three specific features along with one broader, overarching feature.
     - **Sprint Duration:** Each sprint was scheduled for a duration of two weeks.
3) **Definition of User Roles and Features:**
   - **User:**
     - *Availability Verification:* Provision for users to verify the availability of books.
     - *Borrowing Limit:* Enforcement of a limitation that restricts users to borrowing no more than three books concurrently.
   - **Librarian:**
     - *Book Management:*
       * *Adding New Books:* The system facilitates the creation of new book records.
       * *Editing Book Details:* Existing book information can be modified as necessary.
       * *Managing Copies:* The system supports the management of multiple copies of a single title.
   - **Administrator:**
     - *User Management:* Administrators are empowered to create librarian accounts and to impose bans or restrictions on borrowing privileges.
     - *Book Categorization:* The system facilitates the organization of books into meaningful categories.
4) **Operational Requirements:**
   - *Borrowing Period:* Books must be returned within two weeks, with users receiving notifications detailing the remaining days of their loan period.

Week 7: Progress Review, Discussion, and Action Planning:

1) **Progress Updates:**
   - *Front Page Design:* An engaging layout for the front page was developed.
   - *Database Schema:* A robust and scalable database structure was planned.
   - *Sprint Setup:* Tasks were organized for the sprint using the Jira platform.
   - *Scope Evaluation:* The team evaluated whether the planned features were manageable for the first sprint.
2) **Discussion Points:**
   - *User Story Presentation:* The optimal format for presenting user stories was debated, with a recommendation to include detailed feature lists and user personas in the appendix.
   - *Database Model Review:* The proposed table-based database model was deemed appropriate for reporting needs, though potential future enhancements were noted.
   - *Report Structure Guidance:* Example reports and sprint plans were reviewed to clarify the structure of the final report, with the appendix serving as a comprehensive record of work.
3) **Action Items:**
   - Refine the task list for Sprint 1 based on the latest feedback.
   - Organize and incorporate user stories, user personas, and feature lists into the report appendix.
   - Finalize the database schema for implementation.
   - Review exemplary report formats and sprint plans to ensure consistency in documentation.

Week 8: Sprint Update and Enhanced Coordination:

1) **Sprint 1 Progress:**

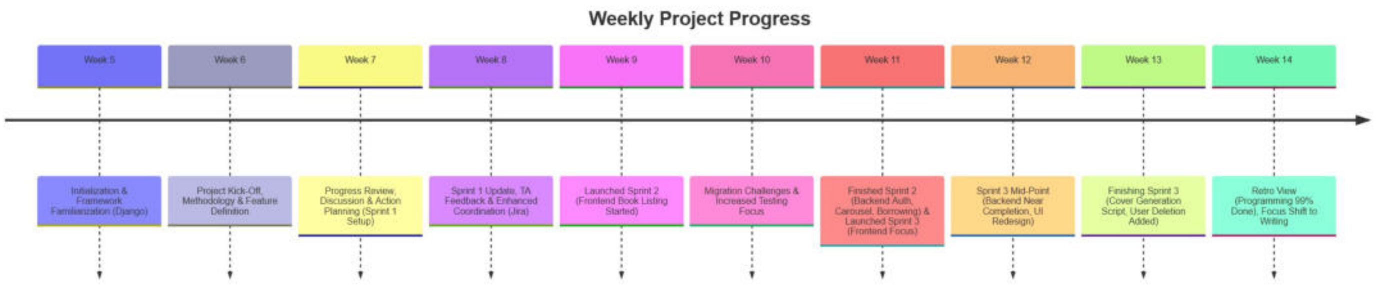

Weekly Project Progress
Week 5
Week 6
Week 7
Week 8
Week 9
Week 10
Week 11
Week 12
Week 13
Week 14
Initialization & Framework Familiarization (Django)
Project Kick-Off, Methodology & Feature Definition
Progress Review, Discussion & Action Planning (Sprint 1 Setup)
Sprint 1 Update, TA Feedback & Enhanced Coordination (Jira)
Launched Sprint 2 (Frontend Book Listing Started)
Migration Challenges & Increased Testing Focus
Finished Sprint 2 (Backend Auth, Carousel, Borrowing) & Launched Sprint 3 (Frontend Focus)
Sprint 3 Mid-Point (Backend Near Completion, UI Redesign)
Finishing Sprint 3 (Cover Generation Script, User Deletion Added)
Retro View (Programming 99% Done), Focus Shift to Writing

Fig. 1. Timeline illustrating the key milestones and activities across the project weeks.

- *Work Summary:* Completed user stories were presented alongside progress on backend authentication and the ongoing development of the frontend login page.
- *Database Initialization:* The MySQL database was initialized, and challenges regarding ensuring consistency across the team were discussed.

2) **Teaching Assistant Feedback:**
   - *User Story Documentation:* A recommendation was made to add a dedicated column for enhanced documentation of user stories.
   - *Time Tracking:* The importance of meticulous documentation of work hours was emphasized.
   - *Task Management:* When features change, new tasks should be added rather than deleting existing ones, ensuring a comprehensive record of progress.

3) **Team Coordination:**
   - *Enhanced Use of Jira:* Increased utilization of the Jira platform was agreed upon to better coordinate tasks.
   - *Sprint Planning Focus:* Emphasis was placed on upcoming tasks, such as refining search and filtering functionalities.

Week 9: Launch of 2. Sprint:

- **Leadership Update:** Alvaro assumed the role of Sprint Master.
- **Frontend Development:** The implementation of the book listing feature on the frontend commenced.

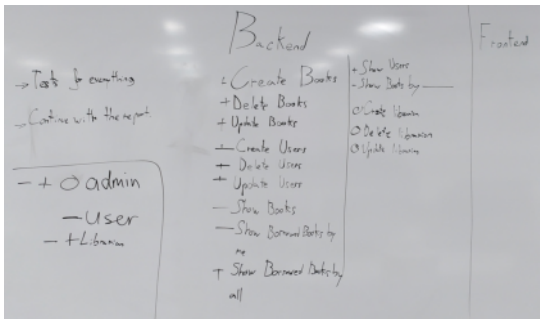


Fig. 2. Overview of the action points developed by the team leader.

Week 10: Migration Challenges and Testing Emphasis:

- **Migration Issue:** Build failures occurred due to an inadvertent double commit.
- **Automated Testing:** The team increased its commitment to integrating automated testing procedures into the development process.
- **Frontend Test Coverage:** The urgent need for enhanced test coverage on the frontend was recognized.

Week 11: End of the second Sprint :

- **Authentication Implementation:** Backend authentication implementation was completed.
- **Carousel View:** The carousel view of the books is functional.
- **Borrowing Functionality:** The borrowing function has been implemented, both in the frontend and backend.

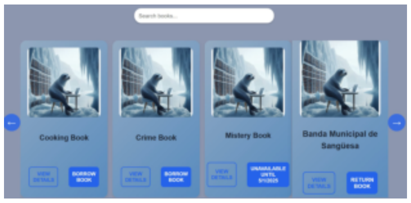


Fig. 3. Possible carousel view of books with navigation arrows, showcasing the newly implemented borrowing functionality.

Launching Sprint 3: Frontend Focus and User Views

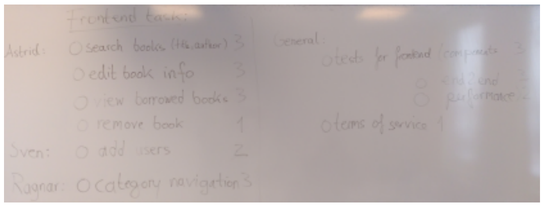
- **Frontend Testing:** More testing is needed for the frontend to ensure stability and reliability.
- **Third Sprint Goals:** Julius is scrummaster. Our aim is to finish the frontend and create tests for all cases. Also we want to implement the user view and the admin view.

Week 12: Middle of the third Sprint :

- **Fullfilling the remaining tasks for the personas:** The backend tasks have been almost been finished and many frontend tasks are remaining, see Figure 4
- **Frontend Testing** Component testing, end2end and performance tests have to be developed
- **Complete UI redesign by Matt:** To make the userinterface more appealing Matt redesigned it, see Figure 5

Week 13: Finishing the third Sprint :

- **Adding Covers:** Julius added a script to automatically generate images to the books using Gemini API


Frontend tasks:

Astrid: ○ search books (title, author) 3
○ edit book info 3
○ view borrowed books 3
○ remove book 1

Sven: ○ add users 2

Ragnar: ○ category navigation 3

General:
○ tests for frontend (components 3
○ end 2 end 3
○ performance 2
○ terms of service 1

Fig. 4. Overview of the remaining tasks on the frontend and in general.

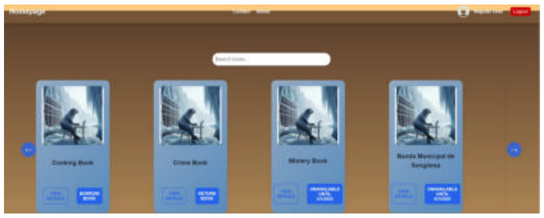


Fig. 5. New UI design.

- **Manage Users:** Matt added the functionality for administrators and librarians to delete users
- **Frontend Testing:** Component testing, end2end and performance tests are still not finished but will be done later.

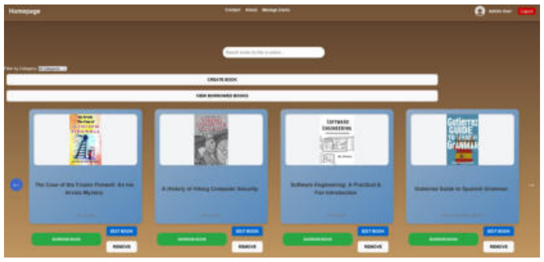


Fig. 6. Added covers using Gemini API.

Week 14: Retro View and Radar View :

- **Retro View:** The programming tasks are 99% done.
- **Writing Tasks:** In the following weeks we'll mainly focus on writing tasks.

C. Personas

PERSONAS:

Librarian: Astrid Vinterstad: A meticulous and efficient librarian, Astrid, in her late 40s, embodies the organized and pragmatic spirit of Tromsø. Obsessed with cross-country skiing and everything winter-related, she ensures the LibManager system is as streamlined and effective as a perfectly groomed ski track. Her goal is peak efficiency and an up-to-date, impeccably maintained digital library catalog.

Administrator: Sven Fjord: A sturdy and practical librarian in his 50s, Sven embodies the strong and silent type, with a deep connection to the Norwegian outdoors. When he's not hiking in the fjords, he's ensuring the LibManager system is robust and reliable. Sven values functionality, accuracy and a no-nonsense approach to library management. He needs the system to be as dependable as a good pair of hiking boots on a mountain trail.

User: Ragnar "The Viking" Haraldsson: Roughly 1200 years young (but a surprisingly spry 40 in modern years). Imagine him, bewildered by modern technology but strangely captivated by the concept of libraries as digital knowledge hordes. LibManager needs to be Viking-proof. Think of features as translating library actions into a language Ragnar understands – simple visual cues. His user stories remind us to prioritize absolute simplicity and clarity in design.

(Illustrations created by the authors using Google Imagen 3).

LibManager Feature Table

LibManager Team

Function	USER STORY	ACCEPTANCE CRITERIA	PRIORITY	ESTIMATION	DESCRIPTION
Add New Book	As Astrid Vinterstad, I want to easily add new books to the system, so that our digital catalog reflects our newest acquisitions immediately.	1. Librarian can access ”Add Book” form. 2. Form includes fields: Title, Author, ISBN, Category, Storage Location. 3. System successfully saves new book entry.	HIGH	Medium	Essential for catalog maintenance. Astrid, ever efficient, needs this to keep the collection up-to-date with minimal fuss.
Search for Book	As Astrid Vinterstad, I want to quickly search for books by title or author, so I can help patrons find what they need, even on a busy, rainy day.	1. Search bar is prominent and functional. 2. Search works by title (partial and full) and author name. 3. Search results are displayed clearly with book titles and authors.	HIGH	Medium	Core functionality.
View Book Availability	As Ragnar ”The Viking” Haraldsson, I need a very simple way to see if a ”book” is ”present”.	1. Book status (Available/Checked Out) is clearly visible on book details and list views. 2. Visual cues (e.g., color icons) represent availability. 3. System uses simple terms understandable by Ragnar.	HIGH	Small	Viking-proof availability status. Simple visuals for Ragnar (and everyone else!), making sure even a 9th-century warrior understands book status.

View Book Details	As Sven Fjord, I want to view detailed information for each book, such as publisher, publication year, and ISBN, to ensure accurate cataloging.	1. Book details page displays all relevant book information: Title, Author, ISBN, Category, Publisher, Year, Location. 2. Information is presented clearly.	MEDIUM	Small	Important for detailed catalog management. Sven, with his practical approach, values accuracy and completeness in the book information.
Browse by Category	As Astrid Vinterstad, I want to browse books by category, so patrons can easily discover books in genres that interest them, like local history.	1. Books are categorized logically (e.g., Fiction, Non-Fiction, Local History). 2. Users can navigate categories easily. 3. Category browsing displays books within the selected category.	MEDIUM	Medium	Enhances book discoverability. Astrid uses categories to organize and promote specific collections like Norwegian culture and history, which she knows are popular.
List All Books	As Astrid Vinterstad, I want to see a list of all books, to get a general overview of the collection and check for gaps or imbalances.	1. "View All Books" option is available. 2. System displays all books in a paginated list. 3. List includes basic book information: Title, Author, Availability.	MEDIUM	Small	Provides a comprehensive overview.
Edit Book Information	As Sven Fjord, I need to be able to edit book information, to correct mistakes or update details as needed, maintaining data accuracy.	1. "Edit Book" function is accessible for librarians. 2. Librarians can modify book details (Title, Author, etc.). 3. Changes are saved and reflected in the system.	MEDIUM	Medium	Crucial for data integrity. Sven insists on precision, and editing allows him to fix errors and keep the catalog impeccable.
Remove Book	As Astrid Vinterstad, I want to remove books from the system, so our digital catalog matches the physical collection, discarding outdated items.	1. "Remove Book" function is available for librarians. 2. System prompts for confirmation before permanent removal. 3. Book is removed from the digital catalog.	MEDIUM	Small	Necessary for catalog hygiene. Astrid proactively manages the collection, removing less relevant books to keep the catalog trim and current.

Implement Simple User Interface	As Ragnar ”The Viking” Haraldsson, the system needs to be very simple to use, even for those unfamiliar with ”glowing rectangles”.	1. User interface is clean and uncluttered. 2. Navigation is straightforward and logical. 3. Minimal technical jargon is used in labels and instructions.	LOW	Small	Viking-approved usability. Although crucial, it’s a foundational principle – like ensuring the library door isn’t a dragon-guarded portal. Needs to be accessible to all users, including Ragnar!