Monami Kirjavainen,

Riku Toivanen,

Victoria Vavulina

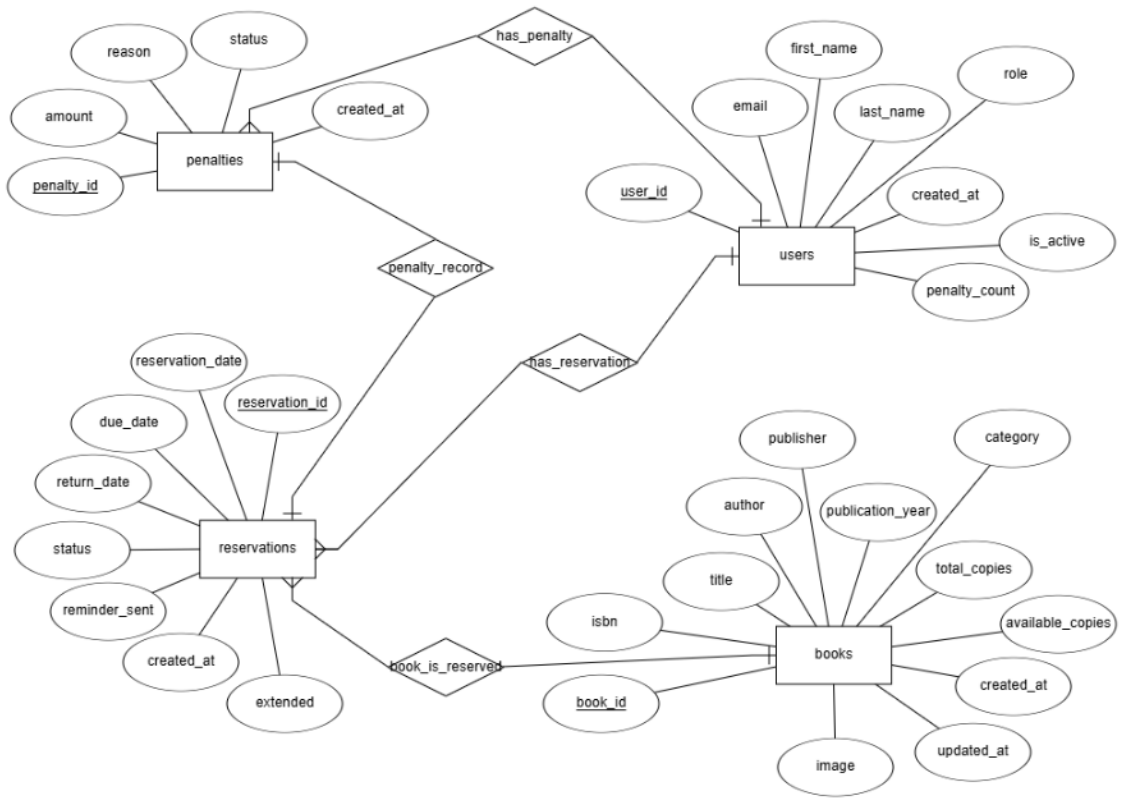
LibraryHub - Library application

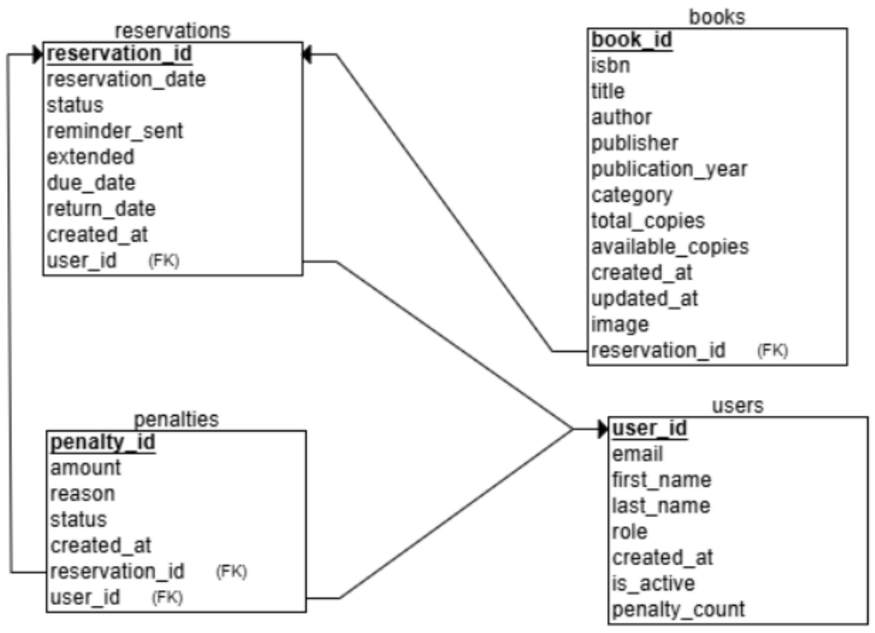
Metropolia University of Applied Sciences
Software Engineering Project – Product Documentation
12.12.2025

1. Introduction

Library Hub is an accessible, user-friendly, and efficient digital library catalogue system that simplifies the process of searching, borrowing, and returning books. The project aims to increase the efficiency of the library system by automating routine operations and providing real-time updates on book availability, which supports user engagement and positive outcomes.

2. Design





2.1 ER Diagrams

2.1.1 Database Schema overview

ER diagrams are used in this section to describe the database structure. Four linked tables make up LibraryHub's database, which uses Supabase to enable user management, book information, book reservations, and overdue penalty features.

2.1.2 Tables and Relationships

users

- **Purpose:** Describes a library user
- **Attributes:**
  - user_id (primary key)
  - email – user's email address
  - first_name – user's first name
  - last_name – user's last name
  - role – user's role (enum: customer, librarian)
  - created_at - timestamp
  - is_active - boolean (true, false), indicates whether the user is active

  - penalty_count - number of penalties accumulated
  - language - saved language for the user (enum: en, ja, ru)
- **Relationships:**
  - has_penalty - (users - penalties) links a user to their penalties (1:M)
  - - has_reservation - (users - reservations) links a user to their reservations (1:M)

penalties

- **Purpose:** Describes the user's penalties
- **Attributes:**
  - penalty_id (primary key)
  - amount - total penalty amount in currency
  - reason - name of the book that caused the penalty, explanation
  - status - (enum - pending, paid, waived)
  - created_at - timestamp
  - reservation_id (foreign key)
  - user_id (foreign key)
- **Relationships:**
  - has_penalty - (users - penalties) links a user to their penalties (1:M)
  - penalty_record - (penalties - reservations) links a penalty to a reservation (1:1)

reservations

- **Purpose:** Describes a book loan transaction
- **Attributes:**
  - reservation_id (primary key)
  - reservation_date - timestamp
  - status - (enum - active, returned, overdue, cancelled)
  - remainder_sent - boolean (true, false)
  - extended - boolean (true, false), indicates whether the loan was extended
  - due_date – timestamp, due date for returning the book
  - return_date - timestamp, actual return date of the book, NULL if not returned
  - created_at - timestamp
  - user_id (foreign key)

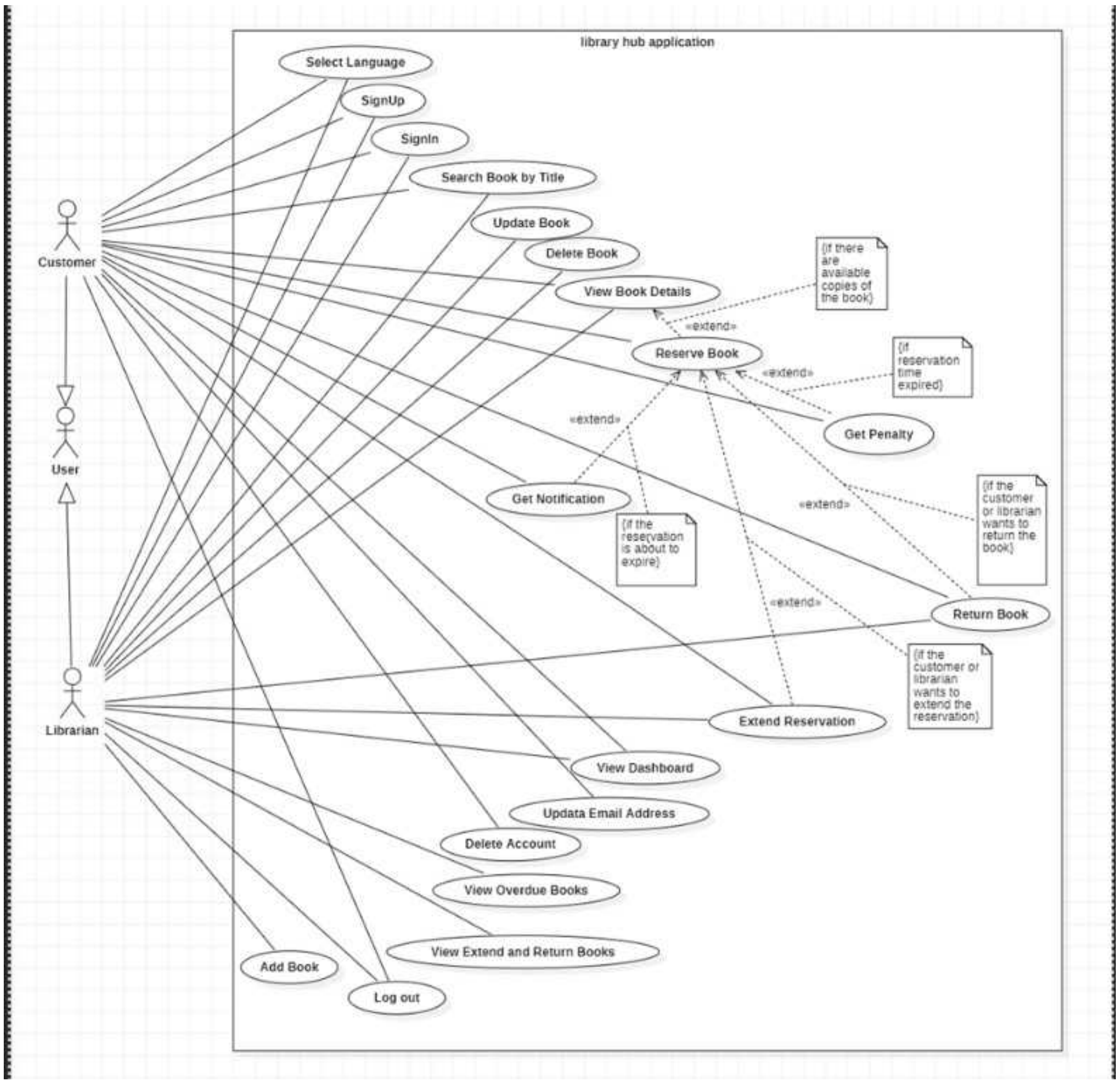
    - book_id (foreign key)
- **Relationships:**
    - penalty_record - (penalties - reservations) links a penalty to a reservation (1:1)
    - has_reservation - (users – reservations) links a user to their reservations (1:M)
    - book_is_reserved - (books - reservations) links books to reservations (1:M)

books

- **Purpose:** Describes books in the library
- **Attributes:**
    - book_id (primary key)
    - isbn - ISBN code
    - title - book title
    - author - book author
    - publisher - book publisher
    - publication_year - year of publication
    - category - category
    - total_copies - total number of copies
    - available_copies - number of available copies
    - created_at - timestamp
    - updated_at - timestamp, last update date of the book
    - image - URL of the book cover image
- **Relationships:**
    - book_is_reserved - (books - reservations) links

2.2 UML Diagrams

The interconnections and behaviors of the system are explained by these diagrams.



2.2.1 Use Case Diagram

2.2.1.1 Actors:

User: Represents a general user who can access and use the library application.

Customer: Represents a customer-type user who can perform actions such as borrowing, returning, and extending books, signing up and signing in, and viewing the dashboard.

Librarian: Represents a librarian-type user who manages the library system, including returning and extending users' books, adding, updating, and deleting books, and viewing the dashboard.

Use Cases & Actions

Account Management:

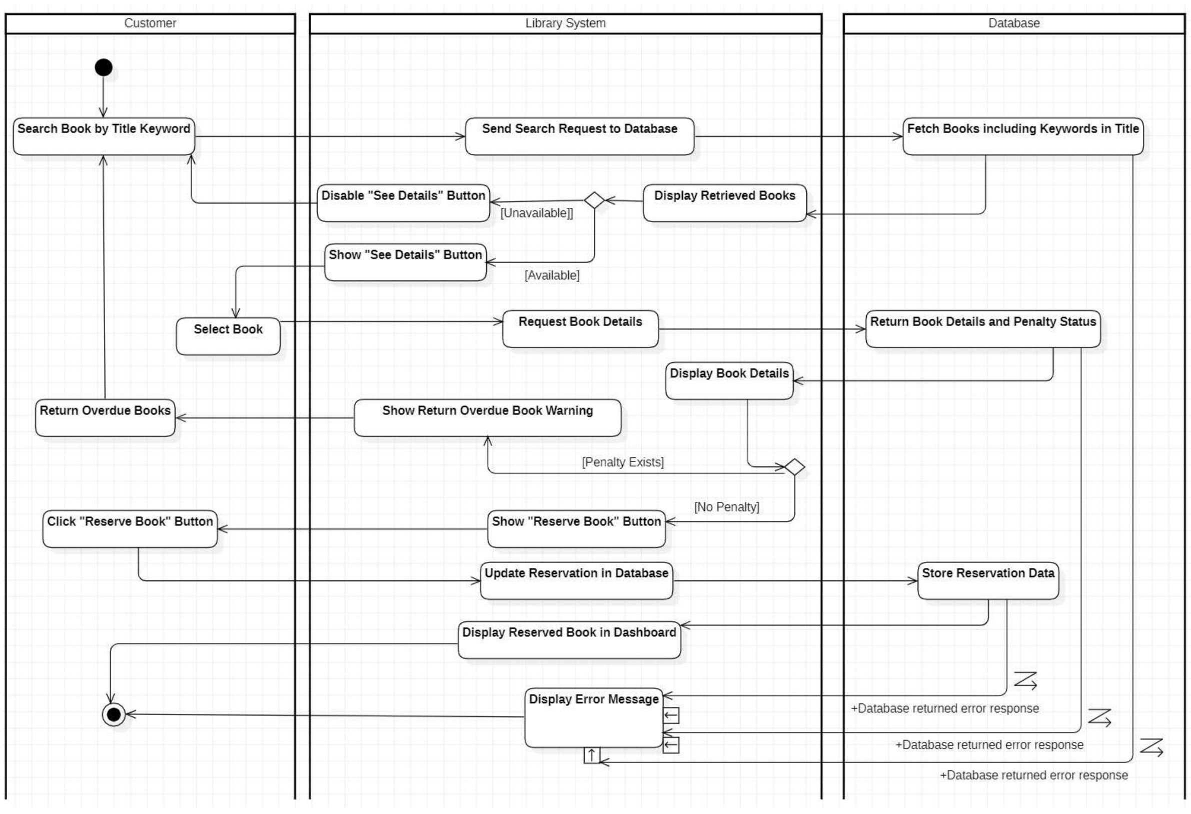
1. **Select Language:** User chooses the preferred language for the website
2. **Sign Up:** Create a new user account
3. **Sign In:** User can log on their account
4. **View Dashboard:** View user profile page
5. **Update Email Address:** Change user's email address
6. **Log out:** Logs user out of their account

Book Management (user):

1. **Search Book by Title:** Look up books with title keyword
2. **View Book Details:** Shows user more details about a book
3. **Reserve Book:** Lets user reserve a book
4. **Return Book:** Returns book after reservation
5. **Get Notification:** Notifies user when a reservation is near due
6. **Get Penalty:** Gives the user a penalty when not returning book in time
7. **Delete Account:** Deletes user account

Book Management (Librarian)

1. **View Overdue Books:** Views books past due for return
2. **Update Book:** Updates book information
3. **Delete Book:** Deletes book
4. **Extend Reservation:** Extends book reservation
5. **View Extend and Return Books:** Views extended books and returns them to the system
6. **Add Book:** Adds a new book to the system

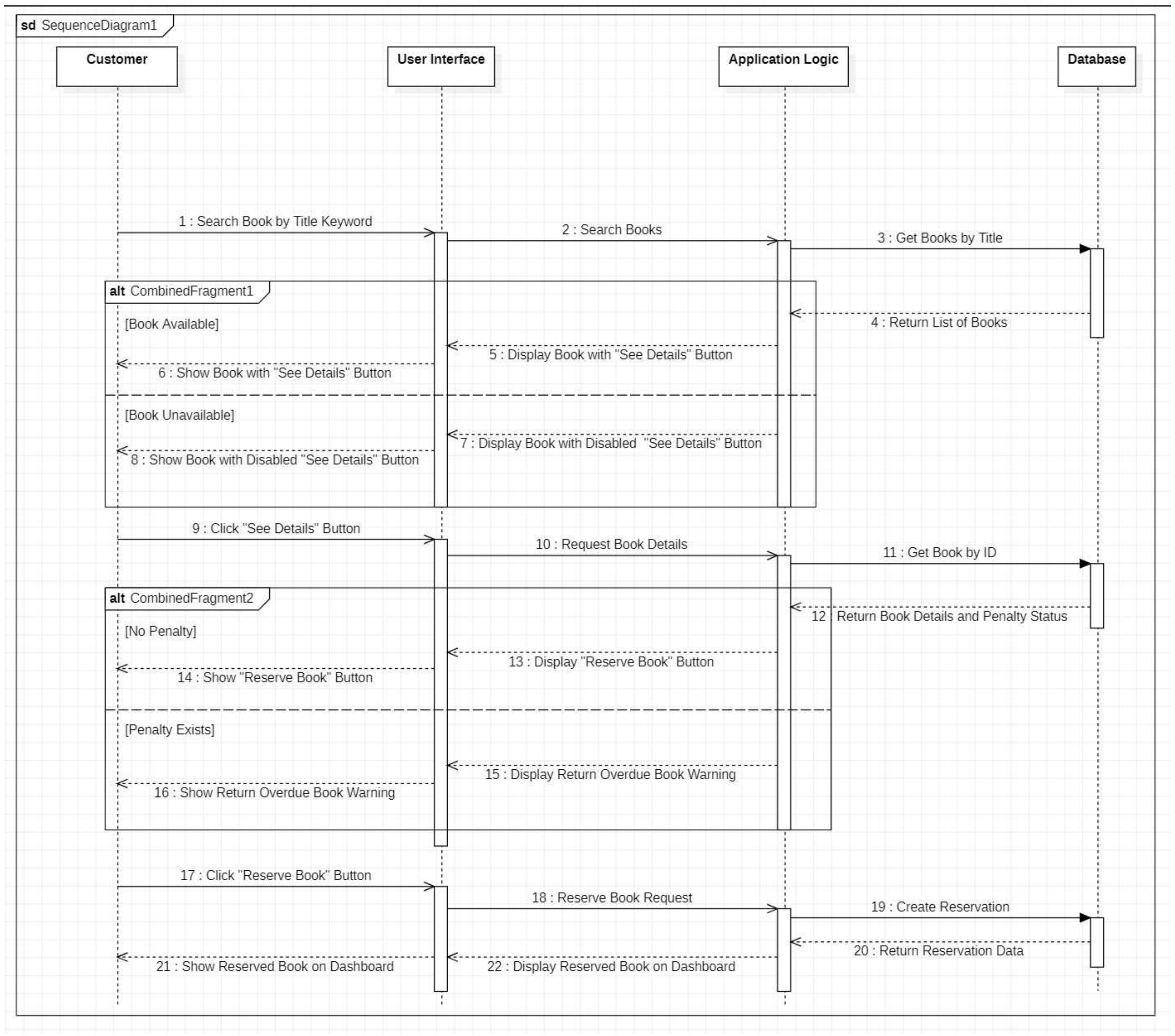


Customer
Library System
Database
Search Book by Title Keyword
Send Search Request to Database
Fetch Books including Keywords in Title
Disable "See Details" Button
[Unavailable]]
Display Retrieved Books
Show "See Details" Button
[Available]
Select Book
Request Book Details
Return Book Details and Penalty Status
Display Book Details
Return Overdue Books
Show Return Overdue Book Warning
[Penalty Exists]
[No Penalty]
Click "Reserve Book" Button
Show "Reserve Book" Button
Update Reservation in Database
Store Reservation Data
Display Reserved Book in Dashboard
Display Error Message
+Database returned error response
+Database returned error response
+Database returned error response

2.2.2 Activity Diagram, Reserving Book:

1. **Search Book by Title Keyword:** As a user, I can search for books by entering keywords in the title field.
2. **Send Search Request to Database:** If entering keywords, the system sends a request to the database to fetch books that match the keywords in their titles.
3. **Fetch Books including Keywords in Title:** The database retrieves a list of books whose titles contain the specified keywords.
4. **Display Retrieved Books:** The system displays the list of books fetched from the database to the user.
5. **Disable "See Details" Button:** The "See Details" button is disabled if no books are available in the library.
6. **Show "See Details" Button:** The "See Details" button is enabled where there are books available in the library.
7. **Select Book:** As a user, I can select a book from the displayed list to view more details about it.
8. **Request Book Details:** If selecting a book, the system requests detailed information about the chosen book from the database.

9. **Return Book Details and Penalty Status:** The database returns detailed information of the selected book along with the user’s penalty status.
10. **Display Book Details:** The system displays detailed information about the selected book to the user.
11. **Show Return Overdue Book Warning:** If the user has any overdue book, the system displays a warning message to the user.
12. **Return Overdue Books:** As a user, I can return overdue books to avoid penalties.
13. **Show the "Reserve Book" Button:** The system displays a "Reserve Book" button for books that are available for reservations.
14. **Click the "Reserve Book" Button:** As a user, I can click the "Reserve Book" button to initiate the reservation process.
15. **Update Reservation in Database:** If clicking the "Reserve Book" button, the system updates the reservation status of the selected book in the database.
16. **Store Reservation Data:** The database stores reservation details, including user information and reservation date.
17. **Display Reserved Book in Dashboard:** The system displays the reserved book in the user’s dashboard for easy access.
18. **Display Error Message:** If the fetching, returning, or reservation of books fails at any point, the system displays an appropriate error message to the user.


sd SequenceDiagram1
Customer
User Interface
Application Logic
Database
1 : Search Book by Title Keyword
2 : Search Books
3 : Get Books by Title
alt CombinedFragment1
[Book Available]
4 : Return List of Books
5 : Display Book with "See Details" Button
6 : Show Book with "See Details" Button
[Book Unavailable]
7 : Display Book with Disabled "See Details" Button
8 : Show Book with Disabled "See Details" Button
9 : Click "See Details" Button
10 : Request Book Details
11 : Get Book by ID
alt CombinedFragment2
[No Penalty]
12 : Return Book Details and Penalty Status
13 : Display "Reserve Book" Button
14 : Show "Reserve Book" Button
[Penalty Exists]
15 : Display Return Overdue Book Warning
16 : Show Return Overdue Book Warning
17 : Click "Reserve Book" Button
18 : Reserve Book Request
19 : Create Reservation
20 : Return Reservation Data
21 : Show Reserved Book on Dashboard
22 : Display Reserved Book on Dashboard

2.2.3 Sequence Diagram, Reserving Book

Customer: Logged in customer user

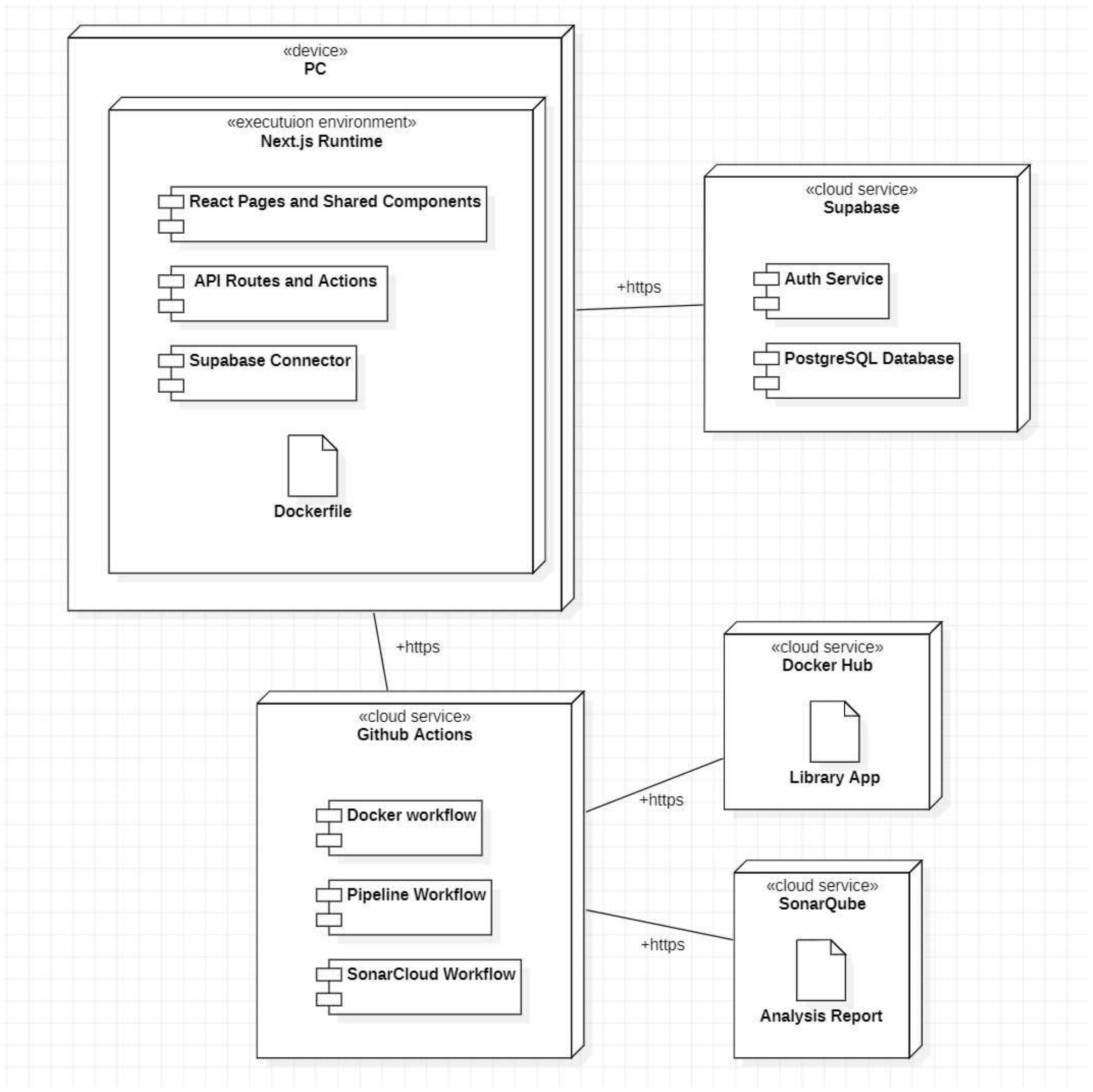
User Interface

Application Logic

Database

1. Search Book by Title Keyword: The Customer searches for a book by entering a title keyword.
2. Search Books: User sends a search request.
3. Get Books by Title: Searching database with the search perimeter.
4. Return List of Books: Returns list of books that match the search.
5. Display Book with "See Details" Button: Shows the “See Details” button.

6. Show Book with "See Details" Button: Shows a book with a button that opens a detailed page if pressed.
7. Display Book with Disabled "See Details" Button: Shows a book with text that says, “Currently unavailable”.
8. Show Book with Disabled "See Details" Button: User won’t be able to open the details page on disabled books.
9. Click "See Details" Button: Opens a detailed page for the selected book.
10. Request Book Details: User sends a request to get a book.
11. Get Book by ID: Book gets searched from the database with the book ID.
12. Return Book Details and Penalty Status: Database sends back book details and if the user has any penalties.
13. Display "Reserve Book" Button: Shows the “Reserve Book” button.
14. Show "Reserve Book" Button: Shows a reserve book button for the user.
15. Display Return Overdue Book Warning: Displays an overdue book warning.
16. Show Return Overdue Book Warning: Displays an overdue book warning to the user.
17. Click "Reserve Book" Button: User presses reserve book button.
18. Reserve Book Request: Sends a request to reserve a book.
19. Create Reservation: Sends reservation to the database.
20. Return Reservation Data: Database sends out successful reservation data.
21. Display Reserved Book on Dashboard: Shows reserved book data.
22. Show Reserved Book on Dashboard: Shows user reserved book data.



2.2.4 Deployment Diagram

Next.js Runtime (Node): Application code runs on a full Node server.

React Pages and Shared Components (Component): In Next.js, a page is a React component that lives in the app/ folder as a page file. Components are reusable building blocks that can be used in multiple pages.

API Routes and Actions (Component): In Next.js, API Routes are server-side endpoints (inside app/api/) that handle requests like fetching or saving data. Actions are server functions that React components can call directly to run server-side logic.

Supabase Connector (Component): In Next.js, a Supabase Connector is a component that connects to the Supabase database, enabling pages or API routes to read and write data.

Dockerfile (Artifact): The Dockerfile is a build recipe that the pipeline uses to create Docker images. The pipeline reads the Dockerfile, builds the image, optionally tests it, and pushes the final image to a container for registry.

GitHub Actions (Node): GitHub Actions is a CI/CD system where workflows run in environments that support Node.js.

Docker workflow (Component): This lets the pipeline build Docker images, test them, and push them to a registry automatically.

Pipeline Workflow (Component): This lets the pipeline run Node scripts to build the app, run tests, install packages, and create reports.

SonarCloud Workflow (Component): The SonarCloud Workflow is a pipeline component that analyzes the codebase for quality, security vulnerabilities, and maintainability issues. It runs automatically in CI, sending reports to SonarCloud where metrics like code smells, test coverage, and duplicated logic are tracked.

Supabase (Node): Supabase is a backend service node that provides authentication, database access, storage, and real-time features. It acts as the main backend platform that the application communicates with through client libraries, server components, API routes, or server actions.

Auth Service (Node): The Auth Service is a node responsible for handling user authentication, including sign-up, login, session management, and token validation. It integrates Supabase Auth.

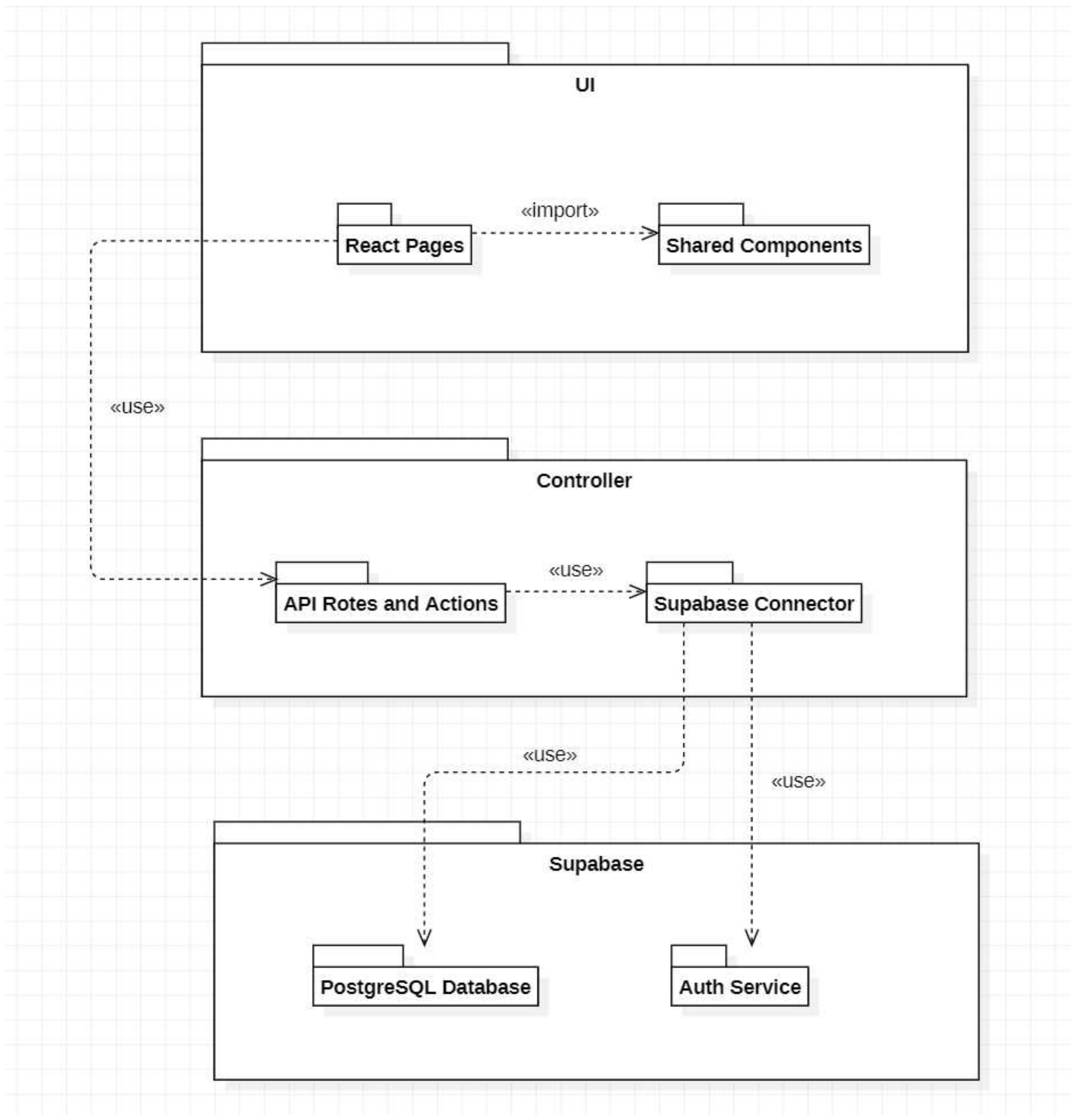
PostgreSQL Database (Node): The PostgreSQL Database is a data storage node that stores structured application data. Supabase provides and manages this database behind the scenes, allowing queries, inserts, updates, and real-time subscriptions through SQL or Supabase client calls.

Docker Hub (Node): Docker Hub is a container registry node where built Docker images are stored. The CI/CD pipeline can push images to Docker Hub, allowing deployments or other services to pull the latest version of the application container image.

Library App (Artifact): The Library App is the packaged output of the application—such as the compiled Next.js build or the final Docker image—that represents the deployable version of the software. This artifact is produced by the pipeline and used in staging or production environments.

SonarQube (Node): SonarQube is a code-quality analysis node that inspects the project for bugs, vulnerabilities, code smells, and maintainability issues. It processes scans from the CI workflow and provides dashboards, metrics, and quality gates that help maintain high code standards.

Analysis Report (Artifact): The Analysis Report is an artifact generated by the quality and testing of workflows. It includes test results, linting output, code coverage data, or SonarQube Cloud findings. This artifact is stored for review in the pipeline and can be used to assess code of health before deployment.



2.2.5 Packages Diagram

1. **UI Package:** Contains the React Pages that form the user-facing interface of the application. This is the presentation layer where users interact with the system.

2. **Shared Components Package:** Houses reusable React components (buttons, forms, navigation elements, etc.) that are imported by React Pages.

3. **Controller Package:** Serves as the intermediary layer between the UI and backend services. It contains the API Routes and Actions that handle business logic and uses the Supabase Connector to communicate with the database layer.

4. **API Routes and Actions Package:** Defines the API endpoints and server actions that the UI can call. This package handles HTTP requests, input validation, and delegates data operations to the Supabase Connector.

5. **Supabase Connector Package:** Provides an abstraction layer for database operations, connecting the Controller to Supabase services.

6. **Supabase Package:** The backend-as-a-service platform that hosts and manages the PostgreSQL Database and Auth Service.

7. **PostgreSQL Database Package:** The relational database that stores all application data. It's accessed through Supabase.

8. **Auth Service Package:** Manages user authentication and authorization within Supabase. It handles user registration, login, session management, and access control, ensuring secure access to application resources.

React Pages <<import>> Shared Components: Pages in the app (like page.tsx) bring in and reuse shared UI pieces. Shared components are buttons, cards, layouts, forms, etc.

React Page <<use>> API Routes and Actions: Pages call server functions to get or send data. In Next.js this is usually inside app/api/... or inside “Server Actions”.

API Routes and Actions <<use>> Supabase Connector: API routes do not talk to the database directly. They call a helper module (the “Supabase Connector”) that wraps Supabase calls.

Supabase Connector <<use>> PostgreSQL Database: The connector uses the Supabase client library to read and write data in the PostgreSQL database.

Supabase Connector <<use>> Auth Service: The connector also handles user login, logout, tokens, and session validation through Supabase Auth.

3. UI Design

The platform's user interface has been developed with Next.js, focusing on clarity and straightforward navigation. Since the current version is designed exclusively for desktop use, responsiveness across devices is not included. The system supports full localization in **English, Japanese, and Russian**, ensuring accessibility for a diverse user base.

3.1 Key UI Features:

- **Consistent Header Across All Pages**: A fixed header is present on every screen, displaying the platform logo, navigation options, and essential functions such as login and book search.
- **Homepage**: The landing page introduces the platform with a central banner and a prominently placed search bar, allowing users to immediately begin searching for books by title.
- **Authentication Screens**: Separate login and signup pages provide clear, easy-to-use forms, enabling both new and returning users to access the system quickly.
- **Role-Based Dashboards**:
    - **Customer Dashboard**: Displays customer information along with lists of borrowed, overdue, and returned books. Borrowed books can be managed directly through **extend** and **return** buttons. Customers may update their email address, delete their account, and view notifications for approaching due dates via the navbar icon. Additionally, book details can be opened after a search, with a **reserve** button available for borrowing. A penalty system prevents customers with overdue books from reserving new ones.
    - **Librarian Dashboard**: Provides staff information and includes a form to add new books. Added books are listed below the form for quick reference. Librarians can also access dedicated pages to manage all borrowed books and all overdue books collectively, with options to extend or return items on behalf of users. After searching by title, librarians can update or delete book records directly.

- **Multilingual Support**: A language toggle allows seamless switching between **English**, **Japanese**, and **Russian** for both the interface and book listings.
- **Notification System**: Once logged in, customers receive alerts through a notification icon in the navbar, reminding them of upcoming due dates and system messages.

3.2 Features Overview

The library platform is equipped with a set of essential and user-friendly features:

- **Book Search and Browsing**: Users can search for books by title, and after retrieving results, they can open detailed book information and reserve items directly.
- **Real-Time Availability & Reservation**: The system displays up-to-date availability, allowing users to reserve books instantly with a single click.
- **Return Management**: Borrowed books can be returned easily through the customer dashboard, with clear action buttons for extending or returning items.
- **Automated Notifications**: Users receive timely alerts about upcoming due dates and overdue items via the notification icon in the navbar.
- **Multilingual Support**: The platform supports seamless language switching between English, Japanese, and Russian, ensuring accessibility for diverse audiences.
- **User Roles & Access Control**: Two distinct roles are implemented — Customer and Librarian. Customers manage their own reservations, overdue books, and account settings, while Librarians oversee book inventory, add new titles, and manage all borrowed and overdue books across the system.
- **Security Features**: Authentication and role-based access are handled through Supabase, ensuring secure login and account management without relying on JWT-based custom implementations.
- **Accessibility-Focused UI**: The interface emphasizes a simple, clean layout and intuitive navigation, making it easy for users to operate the system without unnecessary complexity

Figma UI Prototyping

(https://www.figma.com/design/pzKWLdwl0qDaUedtmmVdAp/Library-System-Design?nodeid=0-1&p=f&m=draw)

4. Implementation Details

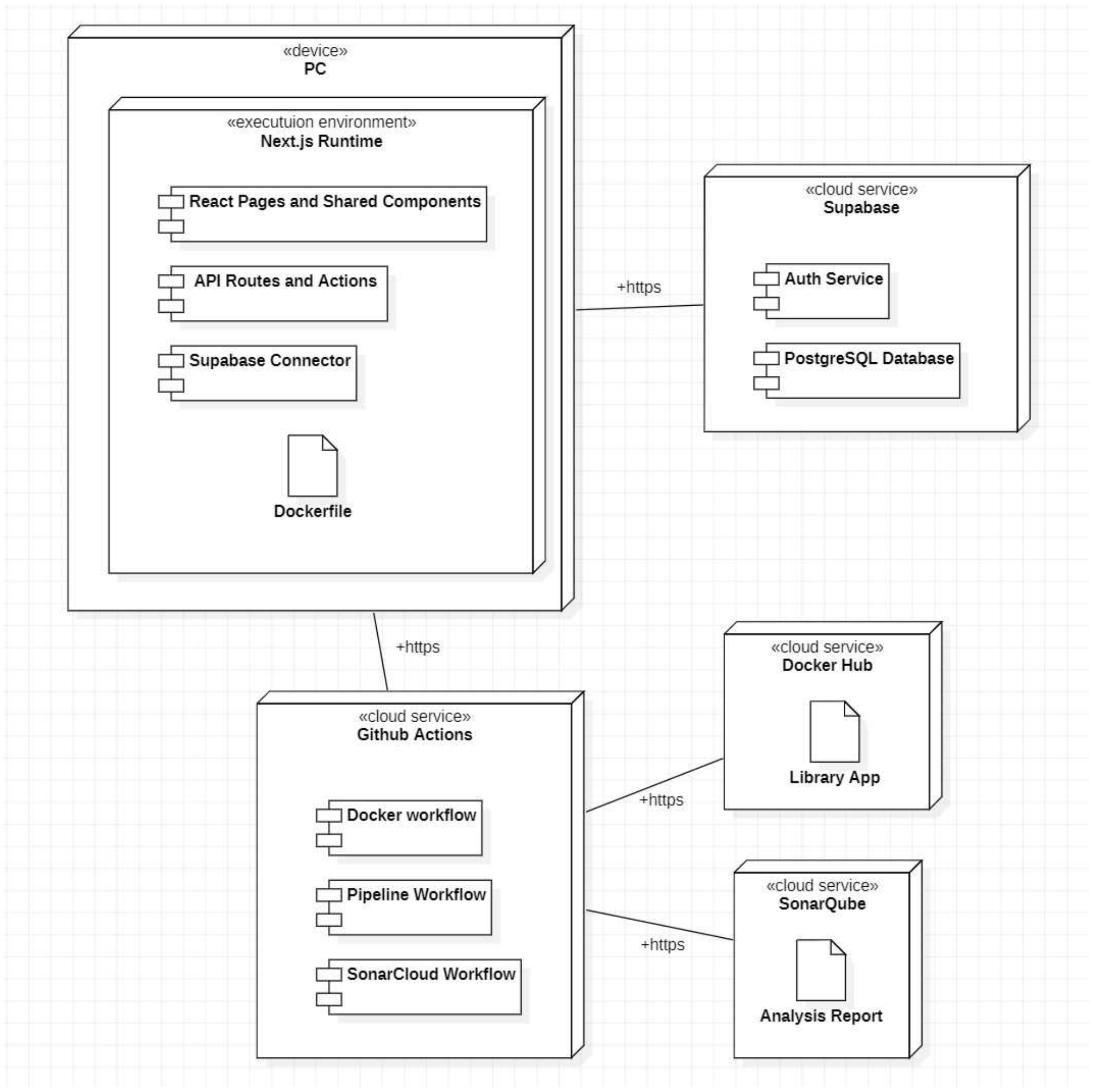
LibraryHub is built using TypeScript as the main programming language, and Node.js is used as the runtime and for handling the build process.

4.1 Key implementations:

- **React (Next.js 15)** for the user interface (frontend)
- **Supabase** as the database and backend service
- **Jest** for unit testing and generating test coverage reports
- **Docker** for containerization and consistent environments

4.2 Architecture overview:

- **Unified App** – Pages, API routes, and server logic in one project
- **Database** – Stores users, books, and borrowing history
- **CI/CD Pipelines** – Automated tests, SonarCloud checks, and Docker builds
- **Containerization** – Docker keeps environments consistent
- **Static Code Analysis** – SonarCloud and ESLint ensure quality and consistency



4.2.1 Core components:

Unified LibraryHub App: Manages the app’s lifecycle. It initializes necessary providers, waits for data synchronization, mounts the interface, and cleans up resources when navigating between views.

Database: Handles two-way communication between the app’s local state and Supabase, keeping the database and real-time channels synchronized.

4.2.2 Data flow:

1. **User interacts** with the LibraryHub App (e.g., searches for a book or borrows one).
2. **App sends a request** through API routes to the **Database**.
3. **Database processes** the request (e.g., updates borrowing history or fetches of book info) and returns the response.
4. **App updates the UI** with the returned data.
5. **CI/CD Pipelines** automatically run tests and checks whenever code is changed to ensure everything works.
6. **Containerization** (Docker) ensures the app behaves the same across development, testing, and production environments.
7. **Static code analysis** runs continuously to maintain code quality.

4.2.3 Key design principles

- **Modular architecture** – different parts of the system (catalog, user accounts borrowing logic) are separated for easier maintenance and scalability
- **Consistency and reliability** – ensure accurate tracking of borrowed and returned books across all users
- **Secure data handling** – user and book information is managed safely with proper access controls
- **User-friendly** – interface and operations are fast and intuitive for all users
- **Extensibility** – easy to add new features, like notifications, recommendations, or reporting, without breaking existing functionality

4.3 Localization:

- The application supports 3 languages: **English**, **Japanese** and **Russian**

4.4 Continuous Integration and Development:

4.4.1 GitHub Actions:

- **CI/CD** pipeline using pipeline.yml to create coverage reports as artifacts
- **CI/CD** pipeline using sonarcloud.yml to run automated tests. SonarCloud generates quality and analysis reports
- **CI/CD** pipeline using docker.yml to build the Docker image and publish it to Docker Hub

4.4.2 Static code analysis

- **SonarCloud** for detecting code issues
- **ESLint** for checking code style and consistency

4.5 Containerization

The application runs inside a Docker container locally. Environment variables are provided through Docker, which helps keep the application's behavior consistent across different environments. The docker.yml pipeline is responsible for building the Docker image.

5. Testing Strategy and Results

LibraryHub uses several types of tests focused mainly on the frontend, since the application is heavily frontend-based. Testing is done on different levels: unit tests for core functionality, UI tests for the main interface, static code analysis for code readability, and non-functional testing to check overall quality.

5.1 Functional testing

5.1.1 Unit Testing

Definition: Unit testing verifies the behavior of individual components of the application in isolation. In the context of a React-based platform, this means testing UI components, hooks, and utility functions to ensure they behave correctly under different scenarios.

Jest: Jest is a widely used JavaScript testing framework that provides a fast, reliable environment for running unit tests. It includes built-in assertions, mocking capabilities, and snapshot testing, making it well-suited for validating React applications.

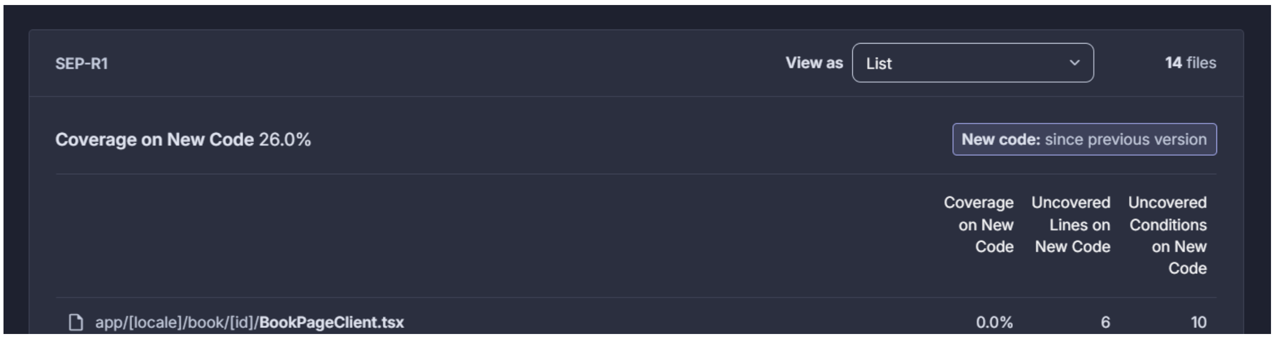
React Testing Library (RTL): React Testing Library focuses on testing components from the user's perspective. Instead of testing implementation details, RTL encourages writing tests that interact with the DOM as a user would (e.g., clicking buttons, filling forms, reading text). This ensures that components are tested in a way that reflects real-world usage.

- **Tools**: Jest + React Testing Library
- **Scope**: Component rendering, event handling, form validation, and integration of UI logic with state management.
- **Highlights**:
    - **Comprehensive testing** across key UI components and business logic.
    - **User-centric approach** that validates actual behavior rather than internal implementation details.

5.1.2 Code coverage report

Our project uses Jest as the primary testing framework, with coverage reporting enabled through its built-in instrumentation tools. The coverage results are visualized using the Jest Coverage Report, which provides detailed metrics on statements, branches, functions, and lines tested.

As of the latest test cycle, the SonarQube cloud integration reports an overall coverage rate of 26.0%, reflecting the current state of unit test implementation across the codebase.


SEP-R1 View as List 14 files

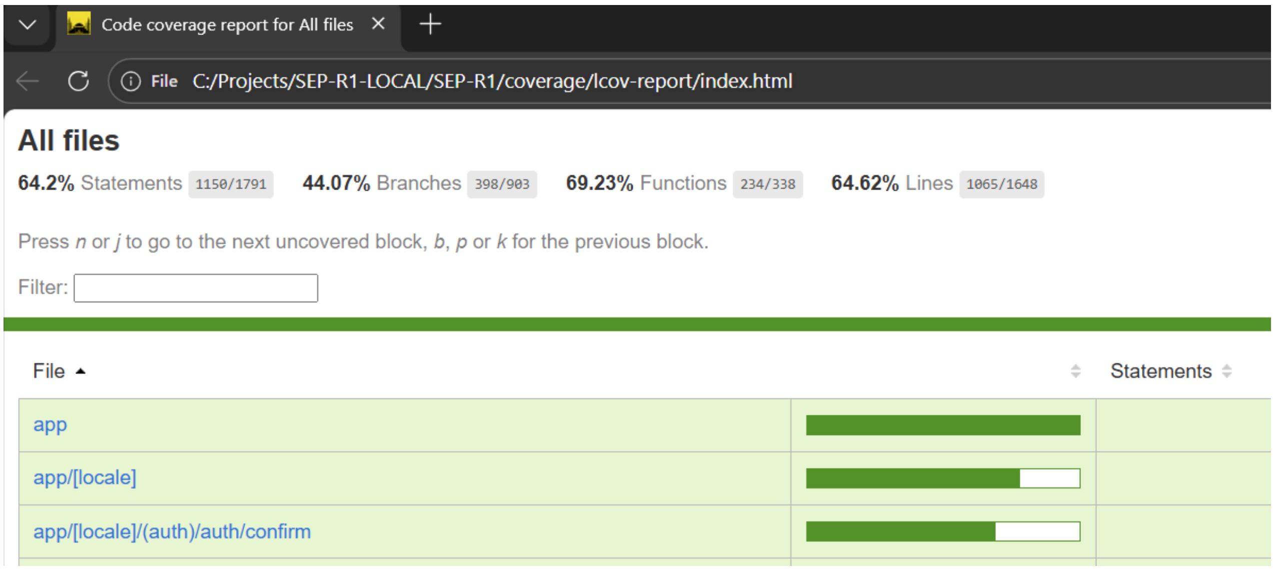
Coverage on New Code 26.0% New code: since previous version

	Coverage on New Code	Uncovered Lines on New Code	Uncovered Conditions on New Code
app/[locale]/book/[id]/BookPageClient.tsx	0.0%	6	10

The Jest Coverage Report offers a more granular view of coverage distribution. For example, the current report shows:

- Statements: 64.2%
- Branches: 44.07%
- Functions: 69.23%
- Lines: 64.62%

Coverage is broken down by file and folder, allowing the team to identify untested areas and prioritize improvements. The report interface includes visual indicators and filtering options to support targeted test development.


Code coverage report for All files

File C:/Projects/SEP-R1-LOCAL/SEP-R1/coverage/lcov-report/index.html

All files

64.2% Statements 1150/1791 44.07% Branches 398/903 69.23% Functions 234/338 64.62% Lines 1065/1648

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements
app		
app/[locale]		
app/[locale]/(auth)/auth/confirm		

5.2 Static Code Analysis

This report represents the results of a code cleaning process carried out on the library management system code using **SonarQube Cloud**. The goal was to identify problems in the code, understand their impact, and improve overall code quality. The analysis focused on issues such as bugs, code smells, security risks, and maintainability. Based on the findings, fixes were made to make the system more stable, easier to read, and easier to maintain.

5.2.1 SonarQube report

Objectives

- Added a static code analysis tool (SonarQube Cloud / SonarScanner) to detect code quality issues.
- Cleaned and improved overall code quality and maintainability based on analysis results.

Obstacle Identification

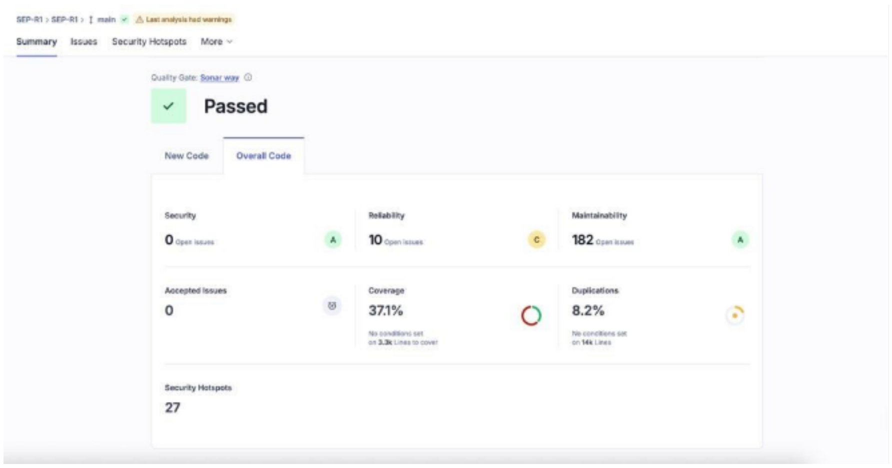
- SonarScanner started, but SonarQube Cloud did not accept or finish the scan.

SonarQube Cloud Results

- **Code Cleaning:** The team used **SonarQube Cloud** / **SonarScanner** and **ESLint** to review code quality issues. Several problems were fixed, including unused code, formatting errors, duplicated logic, and missing types.

SonarQube Cloud Starting Point:

Before the cleaning work began, the project was analyzed with SonarQube Cloud to see the initial code quality status. The following picture shows the starting point of the analysis.


SEP-R1 › SEP-R1 › main ⚠ Last analysis had warnings

Summary Issues Security Hotspots More

Quality Gate: Sonar way

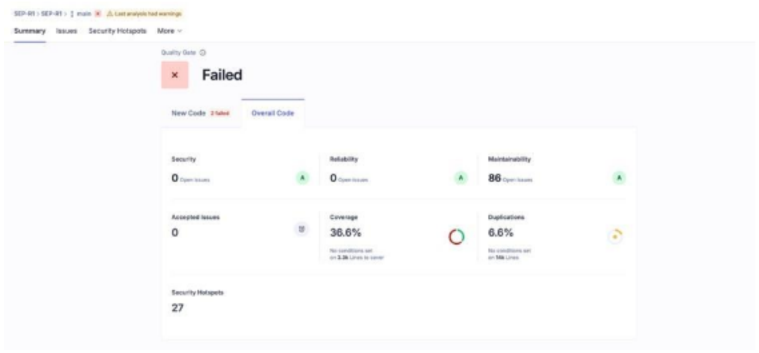
Passed

New Code | Overall Code

Security: 0 Open issues (A)
Reliability: 10 Open issues (C)
Maintainability: 182 Open issues (A)
Accepted Issues: 0
Coverage: 37.1% — No conditions set on 3.3k Lines to cover
Duplications: 8.2% — No conditions set on 14k Lines
Security Hotspots: 27

SonarQube Cloud Results After Code Cleaning:

After the improvements were made, the project was scanned again. The results show better code quality, fewer issues, and higher maintainability. The picture below presents an updated analysis.


SEP-R1 › SEP-R1 › main ✕ ⚠ Last analysis had warnings

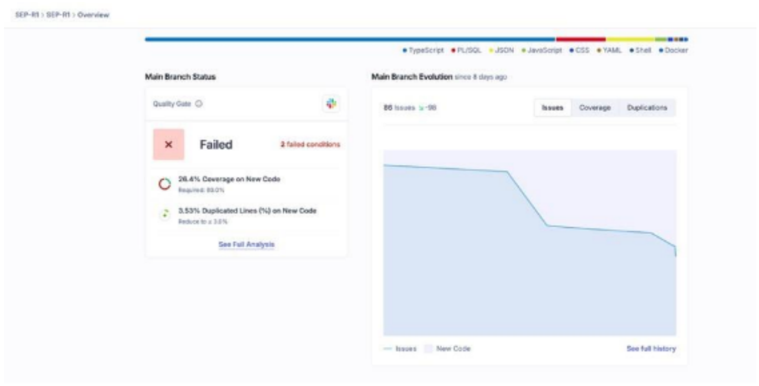
Summary Issues Security Hotspots More

Quality Gate

Failed

New Code 2 failed | Overall Code

Security: 0 Open issues (A)
Reliability: 0 Open issues (A)
Maintainability: 86 Open issues (A)
Accepted Issues: 0
Coverage: 36.6% — No conditions set on 3.3k Lines to cover
Duplications: 6.6% — No conditions set on 14k Lines
Security Hotspots: 27


SEP-R1 › SEP-R1 › Overview

● TypeScript ● PL/SQL ● JSON ● JavaScript ● CSS ● YAML ● Shell ● Docker

Main Branch Status

Quality Gate

Failed — 2 failed conditions

26.4% Coverage on New Code — Required: 80.0%

3.53% Duplicated Lines (%) on New Code — Reduce to ≤ 3.0%

See Full Analysis

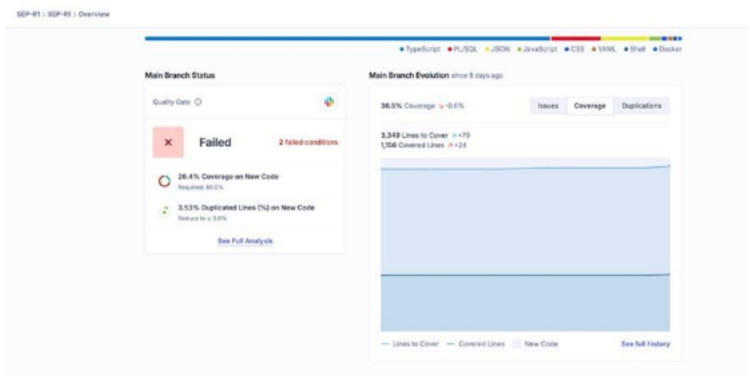
Main Branch Evolution since 8 days ago

86 Issues ↘ -96

Issues | Coverage | Duplications

— Issues ▢ New Code

See full history


SEP-R1 > SEP-R1 > Overview

● TypeScript ● PL/SQL ● JSON ● JavaScript ● CSS ● YAML ● Shell ● Docker

Main Branch Status

Quality Gate

Failed 2 failed conditions

26.4% Coverage on New Code
Required: 80.0%

3.53% Duplicated Lines (%) on New Code
Reduce to ≤ 3.0%

See Full Analysis

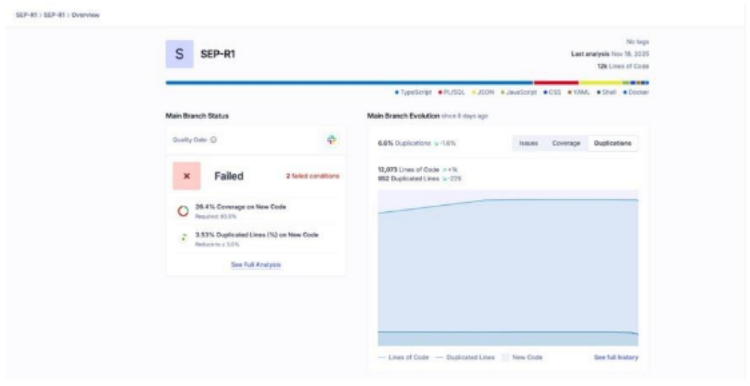
Main Branch Evolution since 8 days ago

36.5% Coverage ↘ -0.6%

Issues Coverage Duplications

3,349 Lines to Cover ↗ +79
1,156 Covered Lines ↗ +24

— Lines to Cover — Covered Lines New Code See full history


SEP-R1 > SEP-R1 > Overview

S SEP-R1

No tags
Last analysis Nov 18, 2025
12k Lines of Code

● TypeScript ● PL/SQL ● JSON ● JavaScript ● CSS ● YAML ● Shell ● Docker

Main Branch Status

Quality Gate

Failed 2 failed conditions

26.4% Coverage on New Code
Required: 80.0%

3.53% Duplicated Lines (%) on New Code
Reduce to ≤ 3.0%

See Full Analysis

Main Branch Evolution since 8 days ago

6.6% Duplications ↘ -1.6%

Issues Coverage Duplications

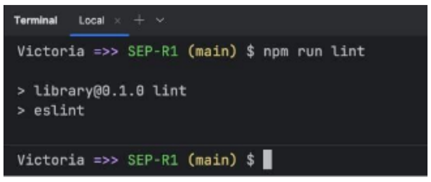
12,073 Lines of Code ↗ +1k
952 Duplicated Lines ↘ -228

— Lines of Code — Duplicated Lines New Code See full history

5.2.2 ESLint result

ESLint results (no errors, no warnings):

After the cleanup, ESLint reported **no errors** and **no warnings**, showing that the code follows the defined style and quality rules.


```
Terminal   Local
Victoria =>> SEP-R1 (main) $ npm run lint

> library@0.1.0 lint
> eslint

Victoria =>> SEP-R1 (main) $
```

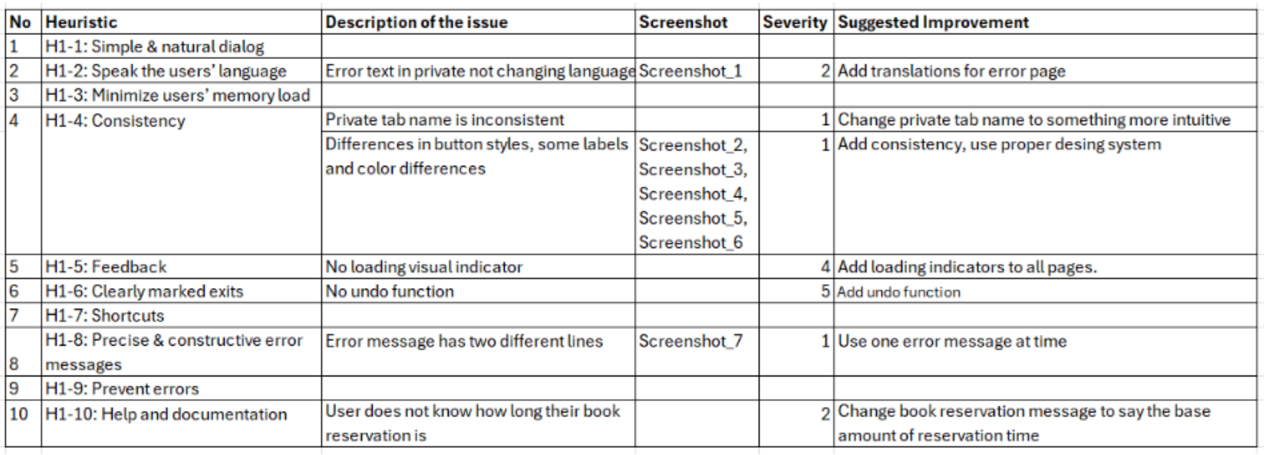
5.2.3 Conclusion of Static Code Analysis

The library management system improved clearly after the code cleaning process. Using SonarQube Cloud, SonarScanner, and ESLint helped the team find weak areas in the code and fix them in a structured way. As a result, the project now has fewer issues, better readability, and stronger maintainability. These improvements make the system more reliable and easier to develop further in the future.

5.3 Non-Functional Testing

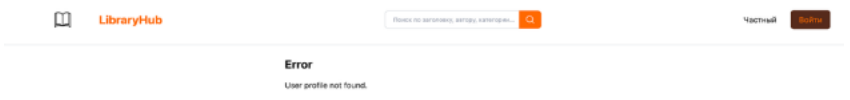
5.3.1 Heuristic Evaluation

This report represents the results of a heuristic evaluation conducted on the library management system prototype.



No	Heuristic	Description of the issue	Screenshot	Severity	Suggested Improvement
1	H1-1: Simple & natural dialog				
2	H1-2: Speak the users' language	Error text in private not changing language	Screenshot_1	2	Add translations for error page
3	H1-3: Minimize users' memory load				
4	H1-4: Consistency	Private tab name is inconsistent		1	Change private tab name to something more intuitive
		Differences in button styles, some labels and color differences	Screenshot_2, Screenshot_3, Screenshot_4, Screenshot_5, Screenshot_6	1	Add consistency, use proper desing system
5	H1-5: Feedback	No loading visual indicator		4	Add loading indicators to all pages.
6	H1-6: Clearly marked exits	No undo function		5	Add undo function
7	H1-7: Shortcuts				
8	H1-8: Precise & constructive error messages	Error message has two different lines	Screenshot_7	1	Use one error message at time
9	H1-9: Prevent errors				
10	H1-10: Help and documentation	User does not know how long their book reservation is		2	Change book reservation message to say the base amount of reservation time

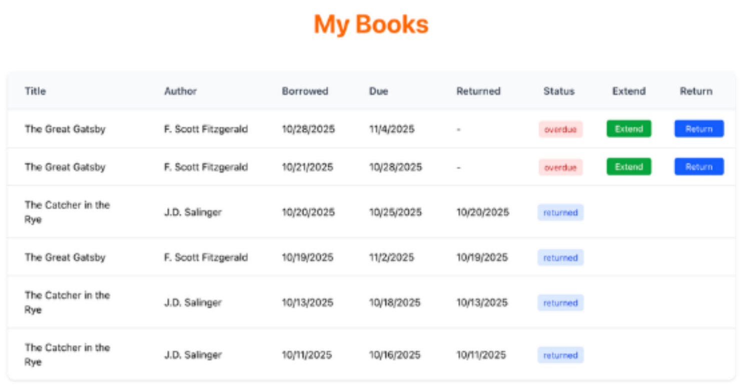
Screenshot_1



Error

User profile not found.

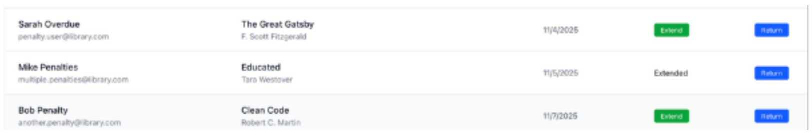
Screenshot_2



My Books

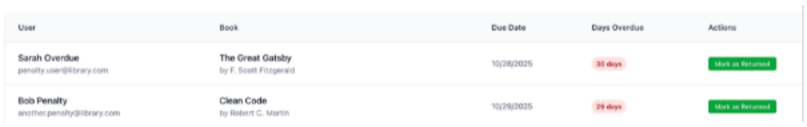
Title	Author	Borrowed	Due	Returned	Status	Extend	Return
The Great Gatsby	F. Scott Fitzgerald	10/28/2025	11/4/2025	-	overdue	Extend	Return
The Great Gatsby	F. Scott Fitzgerald	10/21/2025	10/28/2025	-	overdue	Extend	Return
The Catcher in the Rye	J.D. Salinger	10/20/2025	10/25/2025	10/20/2025	returned		
The Great Gatsby	F. Scott Fitzgerald	10/19/2025	11/2/2025	10/19/2025	returned		
The Catcher in the Rye	J.D. Salinger	10/13/2025	10/18/2025	10/13/2025	returned		
The Catcher in the Rye	J.D. Salinger	10/11/2025	10/16/2025	10/11/2025	returned		

Screenshot_3



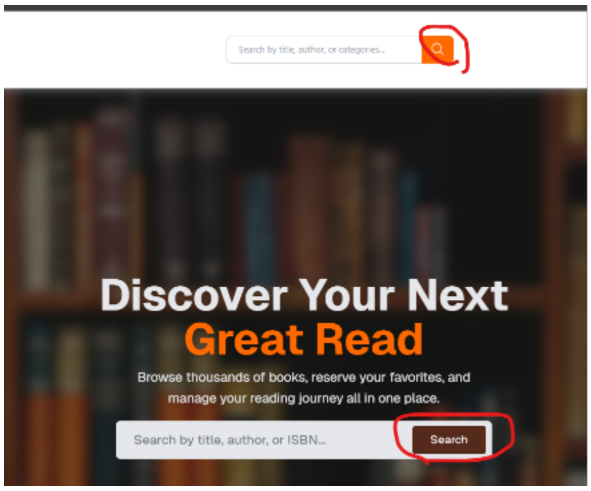
Sarah Overdue penalty.user@library.com	The Great Gatsby F. Scott Fitzgerald	11/4/2025	Extend	Return
Mike Penalties multiple.penalties@library.com	Educated Tara Westover	11/5/2025	Extended	Return
Bob Penalty another.penalty@library.com	Clean Code Robert C. Martin	11/7/2025	Extend	Return

Screenshot_4

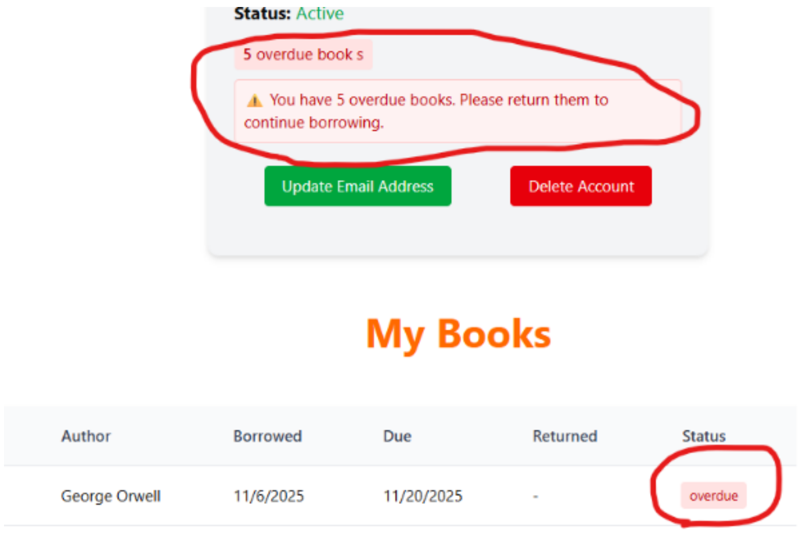


User	Book	Due Date	Days Overdue	Actions
Sarah Overdue penalty.user@library.com	The Great Gatsby by F. Scott Fitzgerald	10/28/2025	30 days	Mark as Returned
Bob Penalty another.penalty@library.com	Clean Code by Robert C. Martin	10/29/2025	29 days	Mark as Returned

Screenshot_5


Search by title, author, or categories...
Discover Your Next
Great Read
Browse thousands of books, reserve your favorites, and manage your reading journey all in one place.
Search by title, author, or ISBN...
Search

Screenshot_6



Status: Active

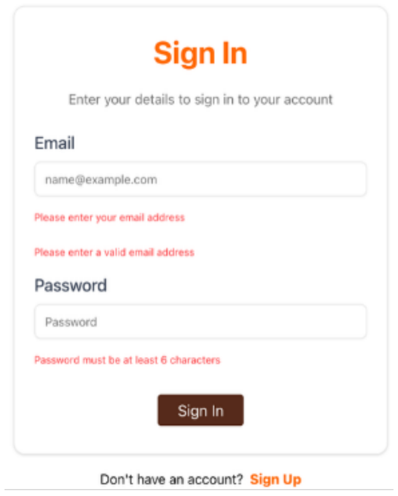
5 overdue book s

⚠ You have 5 overdue books. Please return them to continue borrowing.

My Books

Author	Borrowed	Due	Returned	Status
George Orwell	11/6/2025	11/20/2025	-	overdue

Screenshot_7


Sign In

Enter your details to sign in to your account

Email

name@example.com

Please enter your email address

Please enter a valid email address

Password

Password

Password must be at least 6 characters

Sign In

Don't have an account? Sign Up

Conclusion and Recommendations

The system demonstrates strong usability foundations, particularly in visual feedback, language clarity, and minimalist design. However, several issues must be addressed to ensure user trust, accessibility, and efficiency:

- Add **loading indicators** to all pages.
- Implement **undo or confirmation mechanisms** for critical actions.
- Standardize button styles, color usage, and labels; rename private tab to something intuitive.
- Provide translation for error pages to support multilingual users.
- Ensure error messages to display only one line at a time.
- Clarify reservation duration in user messages.

By resolving these issues, the system will achieve higher usability, reduce user frustration, and align more closely with established design heuristics.

5.3.2 User Acceptance Testing

Introduction

The Final UAT cycle for the **LibraryHub** project was conducted on 4 December 2025, prepared and overseen by Riku Toivanen. The purpose of this cycle was to validate the core user-facing functionalities of the system before release. The scope of testing included sign-up, sign-in, email confirmation, book searching, and book reservation features. Testing was carried out by three participants—Riku, Victoria, and Monami—ensuring coverage across multiple perspectives. This cycle aimed to confirm that the platform delivers a seamless and reliable experience for end users, with all critical workflows functioning as expected.

Metadata

Project Name: Library Hub

UAT Phase: Final UAT cycle

Date: 4.12.2025

Prepared by: Riku Toivanen

Summary

This UAT cycle covers 3 test cases across all the product features. These tests were successfully completed.

Scope of testing

- Features tested
  - Sign up
  - Sign in
  - Email confirmation
  - Searching for books
  - Reserving a book
- Features not tested
  - Admin features
- UAT participants
  - Riku, Victoria, Monami

Test Results Overview

Test Area	Total Cases	Passed	Failed	Blocked
Sign up	3	3	0	0
Sign in	3	3	0	0
Searching for book	3	3	0	0

Reserving book	3	3	0	0
Extending book	3	3	0	0

Detailed Test Case Results

Test Case ID	Description	Tester	Result	Notes
UAT-001	User can sign up	Riku	Passed	-
UAT-002	User can sign in with valid credentials	Riku	Passed	-
UAT-003	User can find a book to reserve and reserve it	Riku	Passed	-
UAT-004	User can sign up	Victoria	Passed	
UAT-005	User can sign in with valid credentials	Victoria	Passed	
UAT-006	User can find a book to reserve and reserve it	Victoria	Passed	
UAT-007	User can sign up	Monami	Passed	
UAT-008	User can sign in with valid credentials	Monami	Passed	
UAT-009	User can find a book to reserve and reserve it	Monami	Passed	

Overall Assessment

The system meets most business and user requirements. There were no issues identified during the testing of the system, and it is therefore ready for production.

5.3.3 System Performance Report

Introduction

This report represents the results of a system performance test conducted on the library management system prototype using **Google Lighthouse**. The goal was to measure performance, identify bottlenecks, and suggest improvements to enhance speed and user experience.

Objectives

- Evaluate page load speed and responsiveness.
- Identify performance, accessibility, and best practice issues.
- Provide actionable recommendations based on Lighthouse metrics.

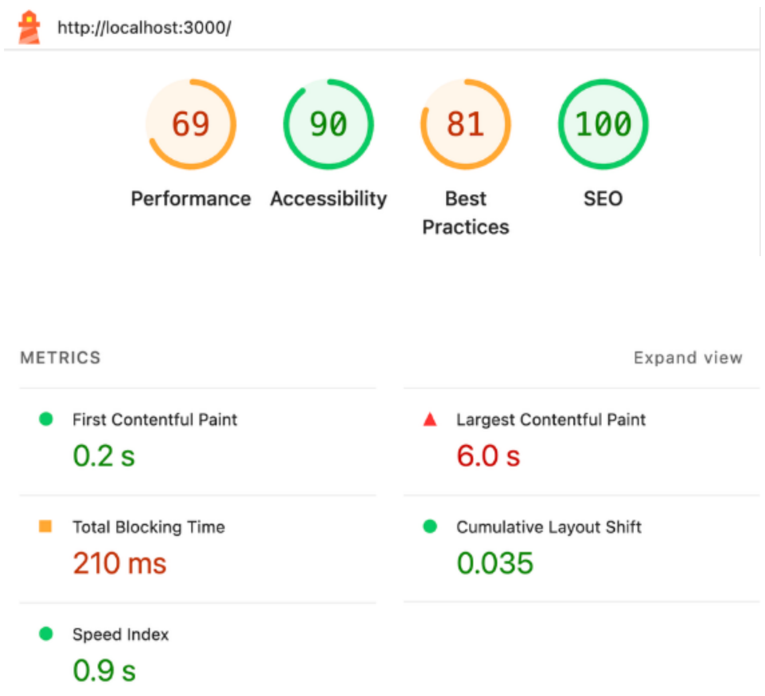
Methodology:

- **Test Environment:** MacBook Pro, Apple Silicon (M1), RAM 16GB, Google Chrome Version 142.0.7444.176 (Official Build) (arm64)
- **Tool Used:** Google Lighthouse
- **Metrics Measured:**
  - **Performance:** First Contentful Paint (FCP), Largest Contentful Paint (LCP), Speed Index, Total Blocking Time (TBT), Cumulative Layout Shift (CLS)
  - **Accessibility:** ARIA labels, color contrast, and focus management
  - **Best Practice & SEO:** HTTPS, image optimization, meta tags

Lighthouse Results

Metric	**Score / Value**	**Notes**
Performance	69 / 100	LCP was slightly high due to large images
First Contentful Paint (FCP)	0.2 s	Excellent
Largest Contentful Paint (LCP)	6.0 s	Needs optimization (image compression)
Speed Index	0.9 s	Excellent
Total Blocking Time (TBT)	210 ms	Within limits
Cumulative Layout Shift (CLS)	0.035	Excellent
Accessibility	90 / 100	Missing some ARIA labels for buttons

Best Practices	81 / 100	Minor HTTPS warnings
SEO	100 / 100	Excellent



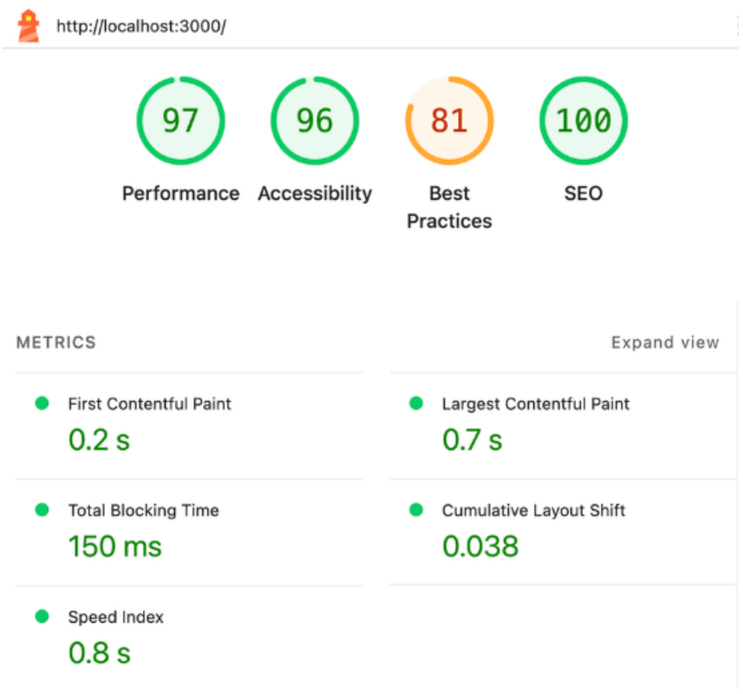
● First Contentful Paint 0.2 s	▲ Largest Contentful Paint 6.0 s
■ Total Blocking Time 210 ms	● Cumulative Layout Shift 0.035
● Speed Index 0.9 s	

Analysis

- Performance is mostly good, but LCP can be improved by optimizing images and scripts.
- Accessibility is high, but some buttons lack ARIA labels (e.g., icon-only buttons).
- Best practices score is 81/100; Best Practices score is 81/100, indicating minor issues such as the use of deprecated APIs and missing source maps for large JavaScript files. These issues affect maintainability and future compatibility.
- SEO is excellent at 100/100, so no immediate action is needed.

Recommendations

- **Optimize images:** compress or use modern formats like WebP to reduce LCP.
- **Add ARIA labels** for icon-only buttons to improve accessibility.
- **Fix minor Best Practices issues**: ensure HTTPS is fully enforced, check for outdated libraries, and follow security recommendations from Lighthouse.
- **Defer non-critical JS** or use code-splitting to reduce TTI.
- **Monitor Lighthouse scores regularly** to track improvements after changes.


http://localhost:3000/

97 Performance
96 Accessibility
81 Best Practices
100 SEO

METRICS Expand view

● First Contentful Paint 0.2 s
● Largest Contentful Paint 0.7 s
● Total Blocking Time 150 ms
● Cumulative Layout Shift 0.038
● Speed Index 0.8 s

Conclusion

The library management system prototype performs well according to Lighthouse metrics. Implementing the above recommendations will improve page speed, accessibility, and user experience, ensuring the system is fast and user-friendly.

5.4 Testing

Test Category	Tool	Result
Unit Testing	Jest	157 Test cases
Static analysis	SonarQube and ESLint	SonarQube issues 86 Grade: A ESLint: no errors and no warnings
Heuristic Evaluation	Manual	11 issues found
UAT	Manual	Succeeded
System Performance Testing	Google Lighthouse	Performance: 97 Accessibility: 96

		Best Practices: 81 SEO: 100

6. Installation & Setup Guide

LibraryHub can be launched either on your local machine or via Docker. The following steps outline the local setup process:

6.1 Local Setup

1. **Clone the repository**
   Retrieve the project source code from GitHub:
   Repository Link (https://github.com/vickneee/SEP-R1.git)

2. **Create a Supabase database**
   Set up a new database instance using Supabase Link (https://supabase.com/)

3. **Configure environment variables**
   Inside the frontend folder, create a file named .env.local and include the following keys:

```
NEXT_PUBLIC_SUPABASE_URL= {your_url}
NEXT_PUBLIC_SUPABASE_ANON_KEY= {your_key}
SUPABASE_SERVICE_ROLE_KEY= {your_service_role_key}
```

4. **Start the application**
   Run the development server with: ‘npm run dev’

5. **Execute tests**:
   Use the following commands to run tests:

   - ‘npm run test’ Executes all unit tests in the project. This command is used to verify that individual components and logic behave as expected.

- ‘npm run test:coverage’, ‘npm run test:coverage:open’ After running unit tests, this command generates a coverage report, and second command opens the coverage UI in your browser. It provides a visual overview of statements, branches, functions, and lines tested, helping the team identify untested areas of the codebase.

6.2 Docker Deployment

This document explains how to build and run the library management system using Docker.

Overview

The application uses a multi-stage Docker build process optimized for production deployment. The build system securely handles environment variables and creates a standalone Next.js application.

Build Script (docker-build.sh)

The docker-build.sh script automates the Docker build process with the following features:

- Secure environment handling: Reads variables from. env.docker without exposing them in git history
- Build argument passing: Safely passes only public environment variables to the Docker build
- Multi-stage optimization: Uses Docker's multi-stage builds for smaller production images
- Standalone output: Creates a self-contained application bundle

How it works:

1. Validates that .env.docker exists
2. Sources environment variables from .env.docker
3. Builds Docker image from repository root (required for proper file context)
4. Passes only public Supabase variables as build arguments
5. Creates a production-optimized image tagged as library-app

Quick Start

Prerequisites

- Docker Desktop installed and running
- .env.docker file configured with your production Supabase credentials

Build and Run

```
# Navigate to the project root
cd project-root/

# Build the Docker image
npm run docker:build

# Run the production container
npm run docker:run
```

Environment Configuration

Create .env.docker in the project-root/ directory:

```
# Supabase Configuration (Production)
NEXT_PUBLIC_SUPABASE_URL=https://your-project.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=your-anon-key

# Environment
NODE_ENV=production
```

Available Commands

Command	Description
npm run docker:build	Build Docker image using build script
npm run docker:run	Run production container on port 3000

Docker Image Details

- Base Image: Node.js 20 Alpine Linux
- Port: 3001

- Build Type: Standalone Next.js application
- Optimization: Multi-stage builds with production dependencies only

Troubleshooting

Build Fails

- Ensure .env.docker exists with valid Supabase credentials
- Verify Docker Desktop is running
- Run from the project-root/ directory

Container Won't Start

- Check that port 3001 is available
- Verify environment variables in .env.docker
- Check Docker logs: docker logs <container-id>

Security Notes

- Only public environment variables (NEXT_PUBLIC_*) are embedded in the client bundle
- Private keys should never be included in build arguments
- The build script prevents accidental exposure of sensitive data

Docker Hub Deployment

The Docker image can be pushed to Docker Hub for easy deployment.

Docker Image pull

You can pull the pre-built Docker image from Docker Hub:

```
docker pull username/library-app:latest # Replace 'username' with the actual Docker Hub username
```

Run the pre-built Docker image

```
docker run -d --name library-app -p 3001:3001 username/library-app:latest # Replace 'username' with the actual Docker Hub username
```

Be sure your desktop Docker is running.

The application Image will be available at http://localhost:3001

6.3 Project structure

```
SEP-R1/

├─ .github/          # GitHub workflows and actions
├─ .idea/            # IDE-specific project settings (JetBrains)
├─ .next/            # Next.js build output and cache
├─ .vscode/          # VSCode workspace settings
├─ __mocks__/        # Mock data for testing
├─ __tests__/        # Unit and integration tests
├─ app/[locale]              # Main application code (pages, API routes)
├─ components/       # Reusable UI components
├─ context/          # React context providers for global state
├─ documents/        # Document-related logic and assets
├─ hooks/            # Custom React hooks
├─ lib/              # Utility libraries and helper functions
├─ locales/          # Translation files and localization
├─ node_modules/     # Installed npm packages
├─ public/           # Public static files (images, fonts, etc.)
├─ supabase/         # Supabase client setup and queries
├─ test/             # Additional test setup/configuration
├─ types/            # TypeScript type definitions
├─ utils/            # Utility functions and helpers
├─ .dockerignore     # Files/folders to ignore in Docker build
├─ .env.docker       # Environment variables for Docker
├─ .env.local        # Local environment variables
├─ .gitignore        # Files/folders ignored by Git
├─ components.json   # Component metadata/configuration
├─ DOCKER.md         # Documentation for Docker setup
├─ docker-build.sh       # Script to build Docker images
├─ Dockerfile            # Dockerfile defining container setup
├─ eslint.config.mjs     # ESLint configuration for linting
├─ i18nConfig.ts         # Internationalization configuration
├─ jest.config.ts        # Jest test framework configuration
├─ jest.d.ts             # TypeScript definitions for Jest
├─ jest.setup.js         # Jest setup file for tests
├─ LOCALIZATION.md       # Documentation on localization setup
├─ middleware.ts         # Custom middleware for the app
├─ next.config.ts        # Main configuration for Next.js (routing, build options, etc.)
├─ package.json          # Lists dependencies, scripts, and project metadata
├─ package-lock.json       # Locks versions of dependencies to ensure consistency
├─ postcss.config.mjs      # PostCSS plugins configuration
├─ README.md             # Project documentation
├─ run_sonar.sh          # Shell script to run SonarCloud analysis
├─ sonar-project.properties       # SonarCloud project configuration
├─ SONARQUBE.md          # Project documentation
├─ tailwind.config.js      # Tailwind CSS configuration
├─ tsconfig.json         # TypeScript compiler options for the project
```

6.4 Database initialization

The included schema file can be used to establish a database on the Supabase website.

6.5 Code quality & static analysis

ESLint

- **Code Quality and Readability:** Use CheckStyle to ensure consistent formatting and maintain clean, readable code
- **ESLint Setup:** ESLint documentation for installation and configuration instructions. ESLint Link (https://eslint.org/docs/latest/)

SonarQube Cloud

- **SonarQube Cloud:** A tool for static code analysis
- The application includes a sonar.properties configuration file
- **Using SonarQube Cloud**: Generate a token and add it as GitHub secret: SONAR_TOKEN={your_token} SonarQube Cloud Link (https://docs.sonarsource.com/sonarqube-cloud)

7. Usage Instructions

To get LibraryHub running locally, follow these steps:

7.1 Launching the Application

1. **Open the project:** Navigate to the project folder and open it in VS Code (or your preferred editor).
2. **Install dependencies:** Run the following command to install all required packages:

```
npm install
```

3. **Set up environment variables:** Inside the `frontend` directory, create a file named `.env.local` and add your Supabase credentials:

```
NEXT_PUBLIC_SUPABASE_URL= {your_url}
NEXT_PUBLIC_SUPABASE_ANON_KEY= {your_key}
SUPABASE_SERVICE_ROLE_KEY= {your_service_role_key}
```

4. **Start the development server:** Launch the application with:

```
npm run dev
```

5. **Access the application:** Open the URL displayed in the terminal (commonly http://localhost:3000)
6. **Language selection:** Choose your preferred interface language: **English, Japanese, or Russian**.

7. **Authentication:**

- Existing users can log in with their credentials.
- New users can register by clicking on a **Register** to create an account.

7.2 Book Search and Reservation (Customer Role)

- Customers can search for books by entering the **title** in the search bar.
- Selecting **See Details** opens the book's information page.
- Click **Reserve Book** places, a hold on the chosen item.

7.3 Manage Reservation Role

- Customers can view a list of all their reserved books from the dashboard.
- If a book qualifies for renewal, the **Extend** option will be available.
- The **Return** button allows customers to manually return a book before its due date.

7.4 Update and Delete Book Records (Librarian Role)

- Librarians can search for books by title to locate specific entries.
- The **edit icon** enables updating book details, while the **trash icon** removes the book from the catalog.

7.5 Add New Books (Librarian Role)

- Librarians access their dashboard by clicking on the **private** section in the navbar.
- A form is provided to input book details, and submitting the form adds the book to the collection.
- Newly added books are displayed in the list below for quick reference.

7.6 Manage Reserved and Overdue Books (Librarian Role)

- From the librarian dashboard, links are available to navigate to **Overdue Books** or **Extend/Return Books** management pages.

- In the **Overdue Books Management** page, librarians can:
  - Mark books as returned.
  - Process overdue items.
  - Refresh the page to update the list.

- In the **Extend & Return Management** page, librarians can:
  - Use the **Extend** option if the book is eligible for renewal.
  - Use the **Return** option to process book returns on behalf of customers.

8. Troubleshooting Guide

Issue: Tests pass locally but not in pipeline Fix:

- The failure is usually caused by timing or environmental differences in the CI pipeline.
- Re-run the pipeline jobs, as the issue often fixes itself on retry.

Localization doesn’t work Fix:

- Check the translation files in /locales/.
- Make sure all language files include the same keys.
- Restart the app so the updated translations load correctly.

9. Frequently Asked Questions (FAQs)

Q1: How long do my reservations last?

A: 1 Week, but the reservation can be extended by another week.

Q2: What happens if I don’t return my book on time?

A: You will be unable to reserve more books until you return all overdue reservations.

10. Support and Contact Information

You can contact us for support or get in touch by submitting an issue on our GitHub repository page. Repository Link (https://github.com/vickneee/SEP-R1.git).