

Bookshop And Sales Management Software : Report

CSD325 - Software Engineering

Aditya Raghuram, Arnav Aditya, Sanskar Sugandhi, Rachit Anand,
Arnav Jalan

2210110126, 2210110189, 2210110898, 2210110487, 2210110898

May 5, 2025

Contents

1 Problem Statement (as given)	3
2 Software Requirements Specification (SRS)	3
3 Functional Requirements	3
3.1 Customer Interface Requirements	3
3.2 Sales Management Requirements	3
3.3 Inventory Management Requirements	4
3.4 Reporting Requirements	4
4 Non-Functional Requirements	4
4.1 Usability Requirements	4
4.2 Security Requirements	4
4.3 Interface Requirements	4
5 DFD (Data Flow Diagram) Model	5
5.1 Level 0 DFD (Context Diagram)	5
5.2 Level 1 DFD	6
6 Use Case Documentation	6
6.1 Use Case Documentation	7
7 Domain Model	10
8 General Sequence Diagram	11
9 System Architecture and Implementation	11
9.1 Technologies Used	11
10 Testing and Results	13
10.1 Testing Strategy	13
10.2 BlacklistedJWTToken Repository Tests	13
10.3 Book Repository Tests	14
10.4 Order Repository Tests	14
11 References	15
A Application Screenshots	16
A.1 Commit Tree	16
A.2 Demonstration	16
A.2.1 Backend Screenshots	16
A.2.2 Frontend Screenshots	23
A.2.3 Logging Screenshot after successful book search	29
A.2.4 Testing Screenshots	29

List of Figures

1	Level 0 Data Flow Diagram (Context Diagram)	5
2	Level 1 Data Flow Diagram	6
3	Use Case Diagram	7
4	Domain Model Diagram	10
5	Sequence Diagram	11
7	Account verification email	16
8	All users (enabled and not enabled)	16
6	GitHub commits	17
9	Stripe dashboard JSON output 1	18
10	Stripe dashboard JSON output 2	18
11	Stripe dashboard JSON output 3	18
12	Stripe dashboard JSON output 4	19
13	Stripe dashboard JSON output 5	19
14	Stripe dashboard JSON output 6	19
15	Postman login JWT token response	20
16	Postman pagination, filtering and sorting response	20
17	project structure	21
18	All DB relations	22
19	All available books in DB	22
20	Landing page interface	23
21	Register page interface	23
22	Email verification success page	24
23	Creating an account with invitation code for admin	24
24	Login page	24
25	User dashboard	25
26	Book search and filter page	25
27	Add to cart dialog	25
28	Shopping cart with quantity 1	26
29	Shopping cart with quantity 4	26
30	Payment success page	26
31	Stripe Payment redirect with fake card info	27
32	Email verification failure page	27
33	User order history page	27
34	Receipt for order in PDF format	28
35	Request book page	28
36	Low stock books and owner dashboard	28
37	System logging configuration	29
38	Unit test for repository layer (Blacklisted JWT Token repository success)	29
39	Unit test for repository layer (Blacklisted JWT Token repository fail)	30
40	All repository layer tests	31

1 Problem Statement (as given)

Bookshop Inventory and Sales Management System (BAS): It should help the customers query whether a book is in stock. The users can query the availability of a book either by using the book title or by using the author's name. If the book is not currently sold by the bookshop, then the customer is asked to enter full details of the book for procurement of the book in future. The customer can also provide his e-mail address, so that he can be intimated automatically by the software as and when the book copies are received. If a book is in stock, the exact number of copies available and the rack number in which the book is located should be displayed. If a book is not in stock, the query for the book is used to increment a request field for the book. The manager can periodically view the request field of the books to arrive at a rough estimate regarding the current demand for different books. BAS should maintain the price of various books. As soon as a customer selects his books for purchase, the salesclerk will enter the ISBN numbers of the books. BAS should update the stock and generate the sales receipt for the book. BAS should allow employees to update the inventory whenever a new supply arrives. Also, upon request by the owner of the book shop, BAS should generate sales statistics (viz., book name, publisher, ISBN number, number of copies sold, and the sales revenue) for any period. The sales statistics will help the owner to know the exact business done over any period and also to determine inventory level required for various books. The inventory level required for a book is equal to the number of copies of the book sold over a period of one week multiplied by the average number of weeks it takes to procure the book from its publisher. Every day the book shop owner would give a command for the BAS to print the books which have fallen below the threshold and the number of copies to be procured along with the full address of the publisher.

2 Software Requirements Specification (SRS)

Attached as SRS.pdf. Kindly refer to the file in the archive.

3 Functional Requirements

3.1 Customer Interface Requirements

- Book availability query system with real-time status display
- Search functionality by title, author, genre, and publication date
- Request system for books not in stock
- Automated notification system for book availability updates
- Form for capturing detailed book information for procurement

3.2 Sales Management Requirements

- Sales transaction processing with receipt generation
- Support for both printed and digital receipts
- Integration with inventory for automatic stock updates after sales

- Batch processing for new stock entries

3.3 Inventory Management Requirements

- Real-time inventory tracking and updates
- Automated low-stock alerts and reports
- Book procurement request management
- Stock level verification against purchase orders

3.4 Reporting Requirements

- Export functionality in multiple formats (PDF, CSV)
- Daily reports for low-stock books

4 Non-Functional Requirements

4.1 Usability Requirements

- Consistent UI design across all web pages
- Product images in catalog

4.2 Security Requirements

- Secure socket implementation for confidential transactions
- Automatic session timeout after inactivity
- Password masking and encryption
- Role-based access control
- Encrypted database storage
- Limited display of sensitive information (e.g., tokens)

4.3 Interface Requirements

- RESTful API architecture
- Integration with payment gateways such as Stripe

5 DFD (Data Flow Diagram) Model

5.1 Level 0 DFD (Context Diagram)

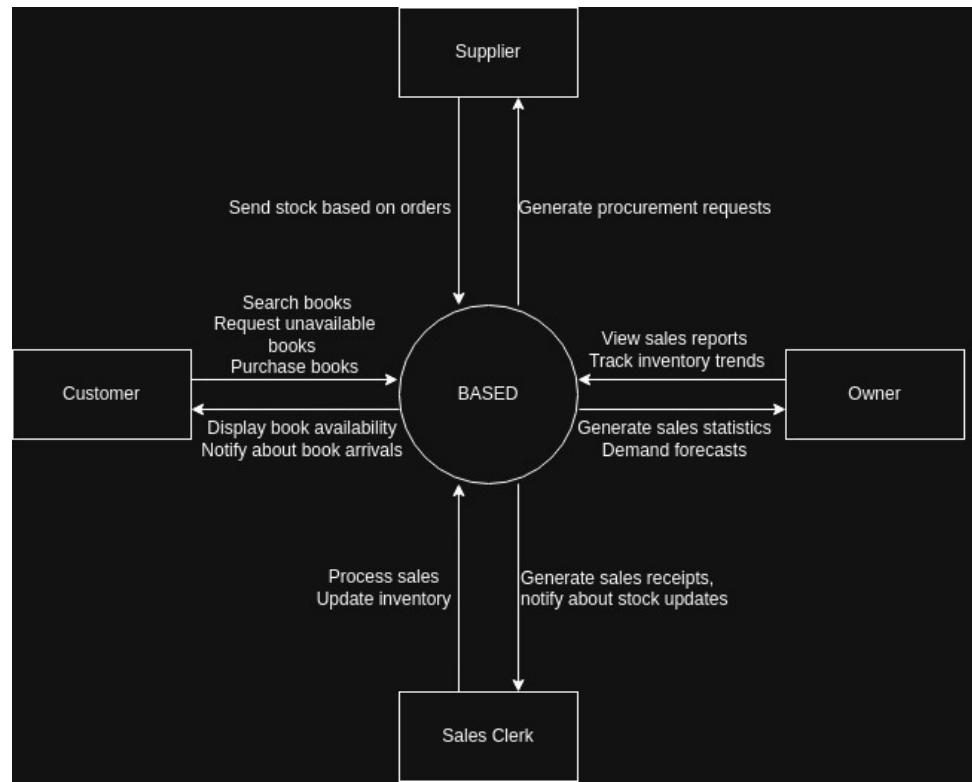


Figure 1: Level 0 Data Flow Diagram (Context Diagram).

5.2 Level 1 DFD

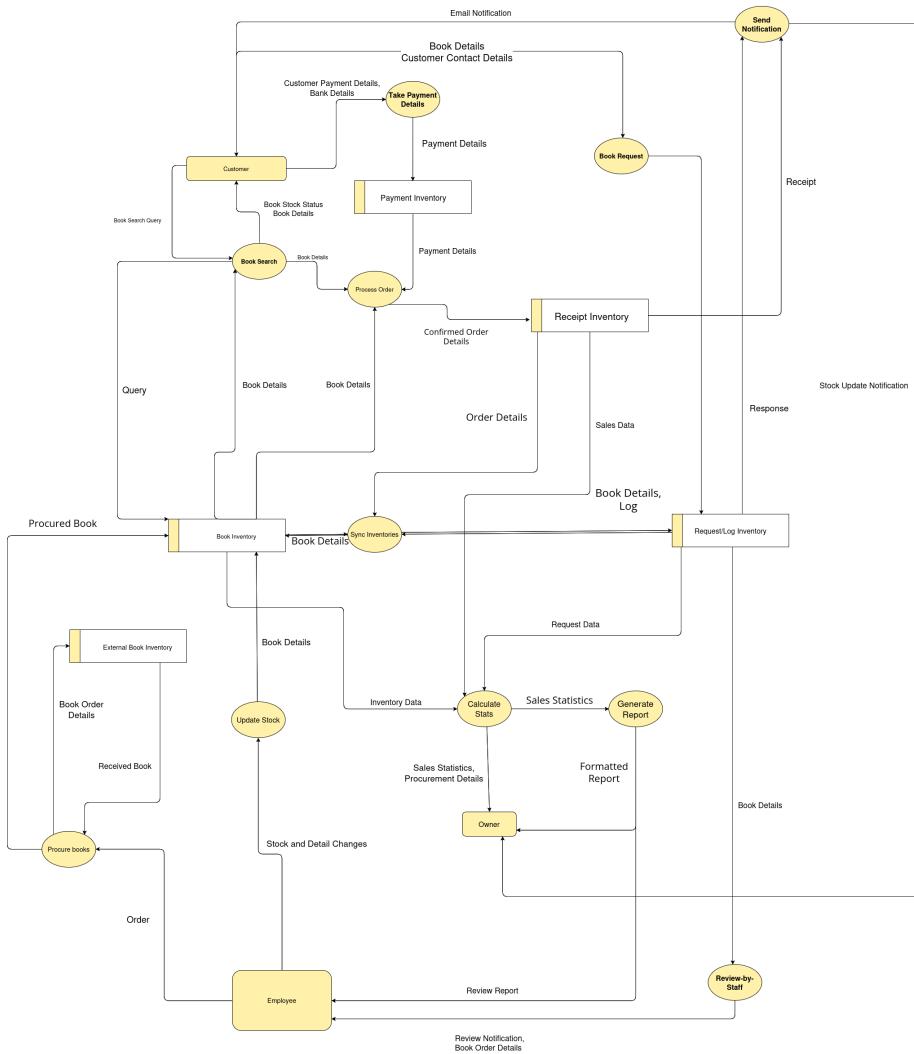


Figure 2: Level 1 Data Flow Diagram.

6 Use Case Documentation

Kindly refer to the Use Case document attached in the archive.

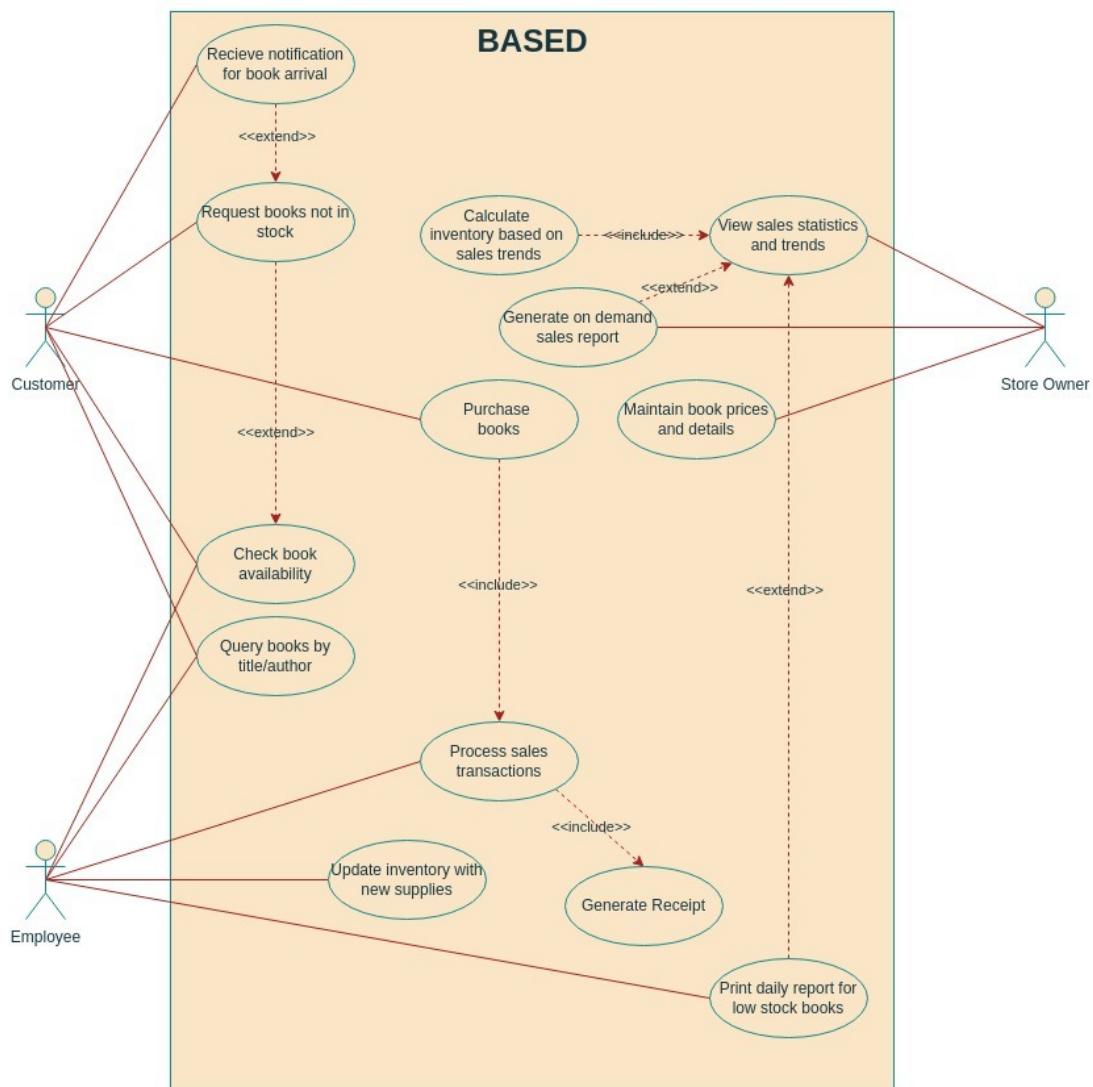


Figure 3: Use Case Diagram

6.1 Use Case Documentation

Use Case: Check Book Availability

ID: UC-001

Use Case Name: Check Book Availability

Actor(s): Customer, Employee

Description: Allows a customer to search for a book in the system to check its availability. The system provides real-time status updates on whether a book is in stock, low stock, or out of stock.

Use Case: Request Books Not in Stock

ID: UC-002

Use Case Name: Request Books Not in Stock

Actor(s): Customer, Inventory Manager

Description: This use case describes the process by which a customer requests a book that is currently out of stock. The system captures the customer's contact details, logs the request for future procurement, and notifies the customer upon availability.

Use Case: Query Books by Title and/or Author

ID: UC-003

Use Case Name: Query Books by Title and/or Author

Actor(s): Customer, Employee

Description: This use case allows a customer or employee to search for books in the system by entering a title, author, or both. The system will return a list of books that match the search criteria, displaying relevant details such as publication year, price, and availability status.

Use Case: Receive Notifications for Book Arrival

ID: UC-004

Use Case Name: Receive Notifications for Book Arrival

Actor(s): Customer, Book Inventory System

Description: This use case allows a customer to receive notifications when a previously unavailable book arrives in stock. The system monitors book inventory and sends an automated alert (via email) to customers who requested the book.

Use Case: Purchase Books

ID: UC-005

Use Case Name: Purchase Books

Actor(s): Customer

Description: This use case describes the process by which a customer purchases books from the system. The system processes the transaction, updates the inventory, and generates a receipt.

Use Case: Process Sales Transaction

ID: UC-006

Use Case Name: Process Sales Transaction

Actor(s): Employee, Inventory Management System

Description: Allows an employee to complete a book sale transaction in the system. The system verifies the selected books, calculates the total cost (including taxes and discounts), and processes the payment. Upon successful completion, a receipt is generated for the customer, and the inventory is updated accordingly.

Use Case: Generate Receipt

ID: UC-007

Use Case Name: Generate Receipt

Actor(s): Employee, System

Description: This use case allows bookstore staff to generate a receipt after a successful purchase. The receipt includes details such as book titles, quantities, prices, taxes, total amount, and payment method. It can be printed or sent digitally to the customer.

Use Case: Update Inventory with New Supplies

ID: UC-008

Use Case Name: Update Inventory with New Supplies

Actor(s): Employee, System

Description: This use case allows employees to update the book inventory when new stock arrives. The system will adjust stock levels, record batch numbers, and update availability status to reflect the latest inventory changes.

Use Case: Print Daily Reports for Low-Stock Books

ID: UC-010

Use Case Name: Print Daily Reports for Low-Stock Books

Actor(s): Employee

Description: This use case describes the process by which an employee generates and prints a daily report of books that have fallen below a predefined stock threshold.

7 Domain Model

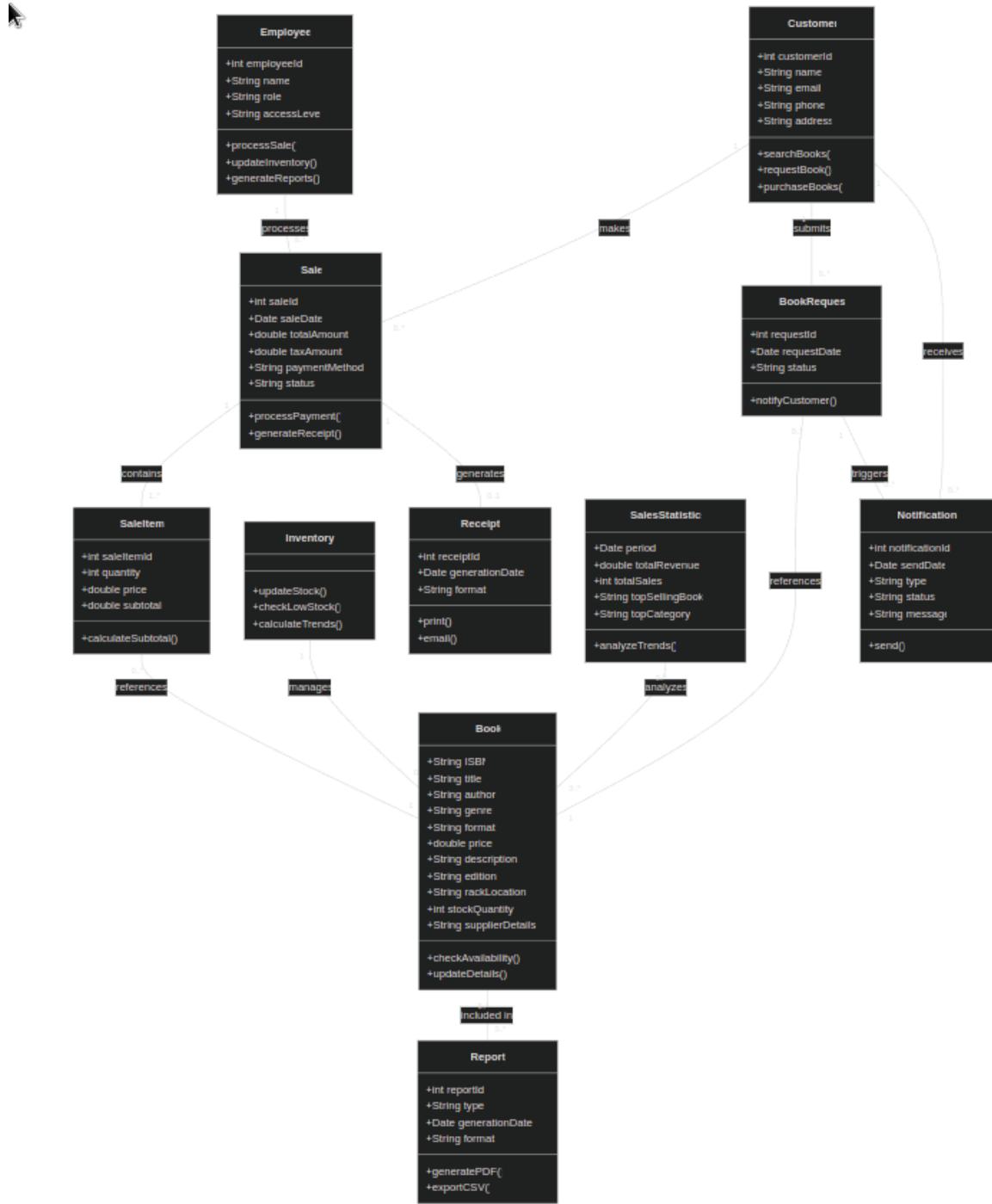


Figure 4: Domain Model Diagram.

8 General Sequence Diagram

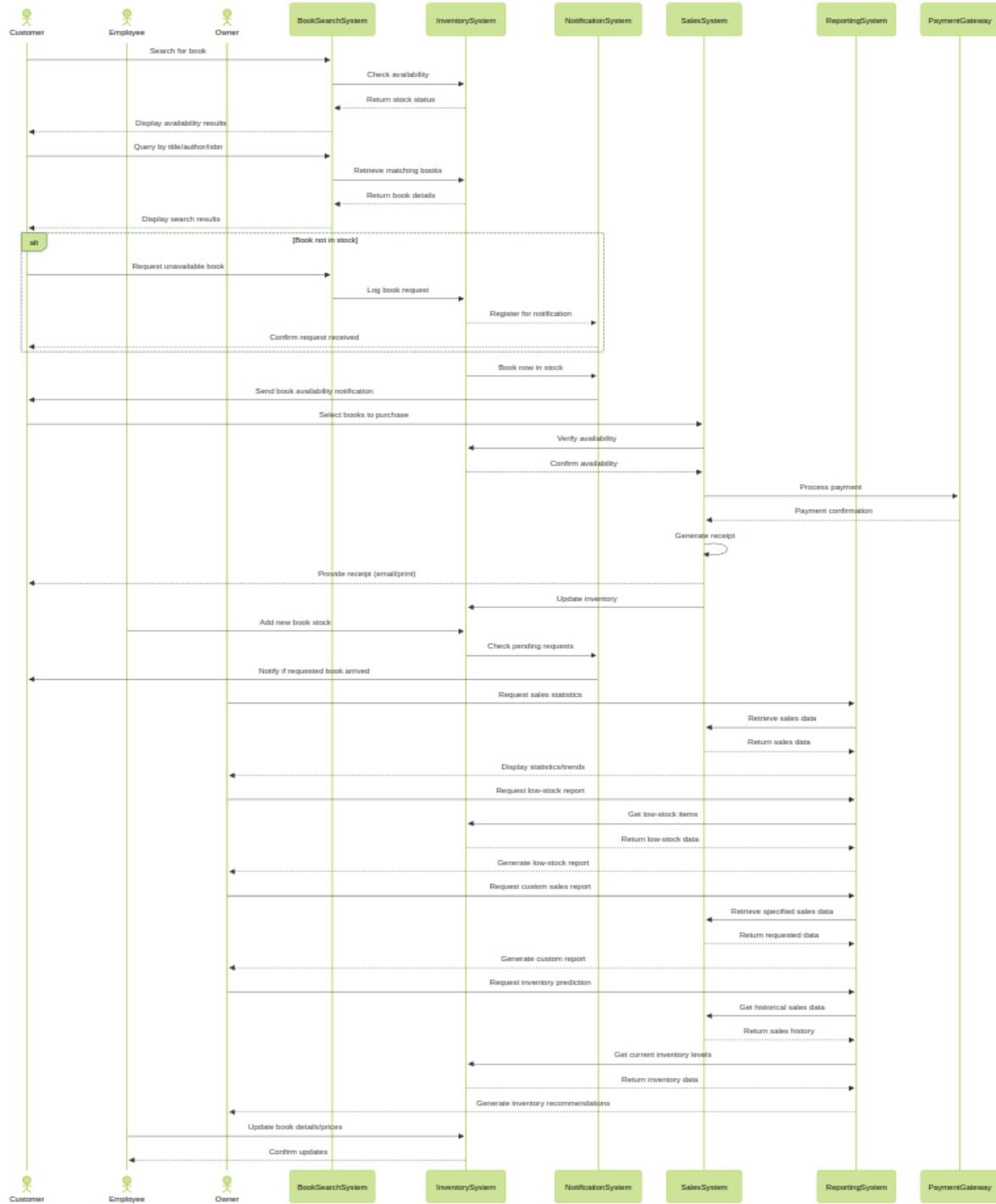


Figure 5: Sequence Diagram.

9 System Architecture and Implementation

9.1 Technologies Used

The backend is built using Java and the Spring ecosystem, while the frontend utilises React.

Backend Technologies

- **Java 21:** The core programming language used for backend development, specified by the ‘`java.version`’ property.
- **Spring Boot 3.4.3:** The core framework providing rapid application development features for building RESTful APIs and web services. Version specified in the ‘`<parent>`’ tag.
- **Spring Boot Starters:** Leveraging modules for specific abilities:
 - `spring-boot-starter-web`: For building web applications.
 - `spring-boot-starter-data-jpa`: For data persistence using the Java Persistence API (JPA), simplifying database interactions.
 - `spring-boot-starter-security`: For handling authentication, authorisation, and other security aspects.
 - `spring-boot-starter-validation`: For applying bean validation constraints.
 - `spring-boot-starter-websocket`: For enabling real-time, bidirectional communication channels.
 - `spring-boot-starter-mail`: For integrating email sending capabilities.
- **MySQL Connector/J:** The JDBC driver enabling the application to connect and interact with a MySQL database at runtime.
- **JSON Web Token (JWT) 0.11.5:** Libraries used for creating and validating JWTs (API security).
- **Project Lombok 1.18.38:** Utility annotation library.
- **OpenPDF 1.3.26:** For creating and manipulating PDF documents, used for generating sales receipts & reports.
- **OpenCSV 5.9:** Parsing and writing CSV (Comma Separated Values) files.
- **Stripe Java Client 29.0.0:** Interacting with the Stripe API, for payment processing.
- **Apache Maven:** Build automation and dependency management tool.

Frontend Technologies

- **React 19:** Core JavaScript library for building user interfaces.
- **Vite 6.2:** Frontend build tool and development server.
- **Tailwind CSS 4.0:** CSS framework used for rapid styling the application components.
- **ShadCN UI Components:** Utilises a collection of composable UI components built on top of Radix UI primitives.
- **Lucide React 0.487:** SVG icons used throughout.

- **Axios 1.8:** Promise-based HTTP client used for API requesting to the backend services.
- **Framer Motion 12.6:** Animation library.
- **Stripe.js 7.1:** Integrating Stripe payment elements in the frontend.

Frontend Development Tools

- **Vite React Plugin ('@vitejs/plugin-react':)**: Enabling React optimisations within the Vite environment.

Testing Technologies

- **Spring Boot Starter Test:** Core testing utilities:
 - **JUnit:** Primary framework for writing and running unit & integration tests.
 - **Spring Test & Spring Boot Test:** Support for loading application context and testing Spring components.
- **Spring Security Test:** Provides utilities specifically for testing Spring Security configurations and secured endpoints.
- **H2 Database:** An in-memory database used specifically during the testing phase to run tests without needing a full external MySQL instance.

10 Testing and Results

10.1 Testing Strategy

The testing strategy adopted in this project primarily revolves around **unit testing** of the repository layer using the `@DataJpaTest` annotation provided by the Spring Boot Test framework. This approach ensures that the JPA-based repositories interact correctly with the in-memory database (H2 in this case) and accurately reflect the state and behavior of the actual database layer.

JUnit 5 (via `org.junit.jupiter`) was used as the core testing framework, while **AssertJ** assertions were employed for fluent and readable test validations. The testing also leverages Spring Boot's auto-configuration for JPA tests, allowing fast and isolated tests of the repository logic.

Each repository was tested with a focus on verifying the correctness of custom query methods, entity relationships, and data persistence/retrieval logic. The tests simulate realistic scenarios, including both positive (data exists) and negative (data does not exist) cases, to ensure robustness.

10.2 BlacklistedJWTToken Repository Tests

This repository manages the blacklisted JWT tokens used for security purposes. The following scenarios were tested:

- **Token Existence Check:** Verified that the repository method `existsByToken(String token)` returns `true` when a token exists and `false` otherwise.

- **Expiry Date Persistence:** Ensured that the expiry date is correctly saved and retrieved without time drift, using millisecond comparison for accuracy.

10.3 Book Repository Tests

The Book repository supports complex search functionality. The tests covered:

- **Search by ISBN:** Checked that the method `findByIsbn(String isbn)` returns the expected book or empty result.
- **Case-Insensitive Search:** Confirmed that both `findByAuthorNameContainingIgnoreCase` and `findByBookNameContainingIgnoreCase` return results regardless of input case.
- **Combined Filtering:** Verified that `findByAuthorNameContainingIgnoreCaseAndBookNameContainingIgnoreCase` accurately filters books by both author name and book name.

10.4 Order Repository Tests

The Order repository involves more complex relationships, such as users and order items. The tests included:

- **Session ID Check:** Confirmed that `existsBySessionId(String sessionId)` identifies existing sessions correctly.
- **Order Sorting:** Ensured that `findAllByUserOrderByCreatedDateDesc(User user)` returns orders in descending order of creation time.
- **User-Order Association:** Validated that orders are properly linked to their respective users.
- **Order Items Retrieval:** Tested that all items related to a specific order are retrieved and linked to the correct books and order.

Overall, these tests validate the correctness of repository logic, data persistence, and relationships across entities in a controlled test environment, increasing confidence in data access layer integrity.

11 References

References

- [1] Spring Team / VMware, *Spring Boot Reference Documentation*. [Online]. Available: <https://docs.spring.io/spring-boot/docs/current/reference/html/>
- [2] Spring Team / VMware, *Spring Data JPA Reference Documentation*. [Online]. Available: <https://docs.spring.io/spring-data/jpa/reference/html/>
- [3] Spring Team / VMware, *Spring Security Reference Documentation*. [Online]. Available: <https://docs.spring.io/spring-security/reference/index.html>
- [4] Jakarta EE Maintainers, *Jakarta Bean Validation Specification*. [Online]. Available: <https://beanvalidation.org/specification/>
- [5] Mozilla Developer Network, *MDN Web Docs - CSS: Cascading Style Sheets*. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/CSS>
- [6] Tailwind Labs, *Tailwind CSS Documentation*. [Online]. Available: <https://tailwindcss.com/docs>
- [7] Oracle Corporation, *Java SE 21 Documentation*. [Online]. Available: <https://docs.oracle.com/en/java/javase/21/>
- [8] shadcn, *shadcn/ui Documentation*. [Online]. Available: <https://ui.shadcn.com/docs>
- [9] React Team / Meta Platforms, Inc., *React Documentation*. [Online]. Available: <https://react.dev/>
- [10] Stack Exchange Inc., *Stack Overflow*. Community Q&A Website. [Online]. Available: <https://stackoverflow.com/> (Note: Citing specific influential Q&A threads is preferable if applicable.)
- [11] Medium, *Medium.com*. Online Publishing Platform. [Online]. Available: <https://medium.com/>

A Application Screenshots

A.1 Commit Tree

A.2 Demonstration

A.2.1 Backend Screenshots

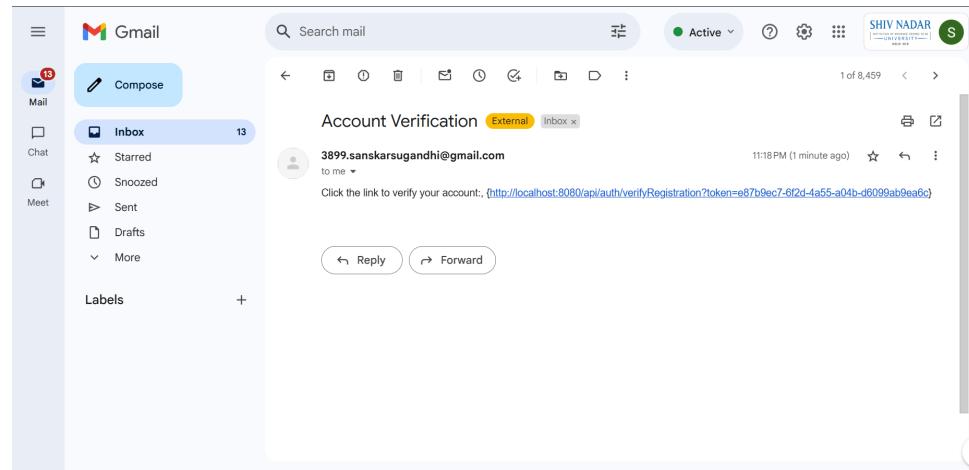
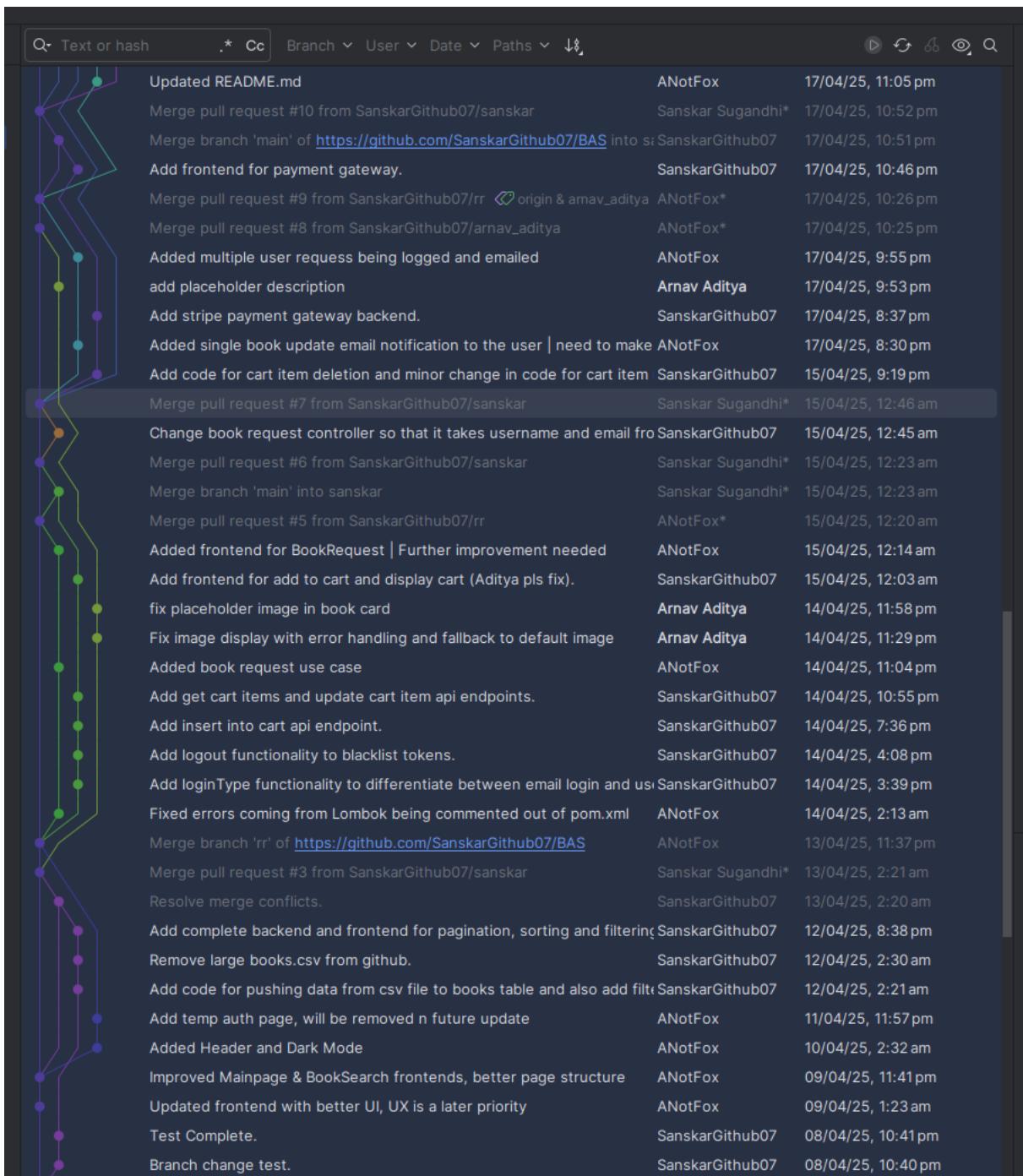


Figure 7: Account verification email

```
mysql> select * from users;
+----+-----+-----+-----+-----+
| id | email           | status | username | enabled | password | profile_pic |
+----+-----+-----+-----+-----+
| 1  | 3899.sanskar.sugandhi@gmail.com | NULL  | Sanny    | 0x00    | $2a$11$VmWF/abLssrVoEql0EQg2uHIWkoYd8P/K5uHK8yayS908tZtrBz4e | NULL
+----+-----+-----+-----+-----+
1 row in set (0.03 sec)

mysql> select * from users;
+----+-----+-----+-----+-----+
| id | email           | status | username | enabled | password | profile_pic |
+----+-----+-----+-----+-----+
| 1  | 3899.sanskar.sugandhi@gmail.com | 0x00  | Sanny    | $2a$11$VmWF/abLssrVoEql0EQg2uHIWkoYd8P/K5uHK8yayS908tZtrBz4e | NULL
| 2  | ss736@snu.edu.in   | 0x01  | Sanskar | $2a$11$760tR7r9m2ZsQz0IuDijJOORo4cXHs3iKKex5jvEFIF2CAK8nZzt6 | NULL
+----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Figure 8: All users (enabled and not enabled)



The screenshot shows a GitHub commit history with a detailed commit graph on the left and a list of commits on the right. The commit graph illustrates a complex merge history with multiple branches and pull requests. The commits listed are:

Commit Message	Author	Date
Updated README.md	ANotFox	17/04/25, 11:05 pm
Merge pull request #10 from SanskarGithub07/sanskar	Sanskarsugandhi*	17/04/25, 10:52 pm
Merge branch 'main' of https://github.com/Sanskargithub07/BAS into sanskar	Sanskarsugandhi*	17/04/25, 10:51 pm
Add frontend for payment gateway.	Sanskargithub07	17/04/25, 10:46 pm
Merge pull request #9 from SanskarGithub07/rr origin & arnav_aditya	ANotFox*	17/04/25, 10:26 pm
Merge pull request #8 from SanskarGithub07/arnav_aditya	ANotFox*	17/04/25, 10:25 pm
Added multiple user requests being logged and emailed	ANotFox	17/04/25, 9:55 pm
add placeholder description	Arnav Aditya	17/04/25, 9:53 pm
Add stripe payment gateway backend.	Sanskargithub07	17/04/25, 8:37 pm
Added single book update email notification to the user need to make ANotFox		17/04/25, 8:30 pm
Add code for cart item deletion and minor change in code for cart item	Sanskargithub07	15/04/25, 9:19 pm
Merge pull request #7 from SanskarGithub07/sanskar	Sanskarsugandhi*	15/04/25, 12:46 am
Change book request controller so that it takes username and email from SanskarGithub07		15/04/25, 12:45 am
Merge pull request #6 from SanskarGithub07/sanskar	Sanskarsugandhi*	15/04/25, 12:23 am
Merge branch 'main' into sanskar	Sanskarsugandhi*	15/04/25, 12:23 am
Merge pull request #5 from SanskarGithub07/rr	ANotFox*	15/04/25, 12:20 am
Added frontend for BookRequest Further improvement needed	ANotFox	15/04/25, 12:14 am
Add frontend for add to cart and display cart (Aditya pls fix).	Sanskargithub07	15/04/25, 12:03 am
fix placeholder image in book card	Arnav Aditya	14/04/25, 11:58 pm
Fix image display with error handling and fallback to default image	Arnav Aditya	14/04/25, 11:29 pm
Added book request use case	ANotFox	14/04/25, 11:04 pm
Add get cart items and update cart item api endpoints.	Sanskargithub07	14/04/25, 10:55 pm
Add insert into cart api endpoint.	Sanskargithub07	14/04/25, 7:36 pm
Add logout functionality to blacklist tokens.	Sanskargithub07	14/04/25, 4:08 pm
Add loginType functionality to differentiate between email login and user login	Sanskargithub07	14/04/25, 3:39 pm
Fixed errors coming from Lombok being commented out of pom.xml	ANotFox	14/04/25, 2:13 am
Merge branch 'rr' of https://github.com/Sanskargithub07/BAS	ANotFox	13/04/25, 11:37 pm
Merge pull request #3 from SanskarGithub07/sanskar	Sanskarsugandhi*	13/04/25, 2:21 am
Resolve merge conflicts.	Sanskargithub07	13/04/25, 2:20 am
Add complete backend and frontend for pagination, sorting and filtering	Sanskargithub07	12/04/25, 8:38 pm
Remove large books.csv from github.	Sanskargithub07	12/04/25, 2:30 am
Add code for pushing data from csv file to books table and also add filters	Sanskargithub07	12/04/25, 2:21 am
Add temp auth page, will be removed in future update	ANotFox	11/04/25, 11:57 pm
Added Header and Dark Mode	ANotFox	10/04/25, 2:32 am
Improved Mainpage & BookSearch frontends, better page structure	ANotFox	09/04/25, 11:41 pm
Updated frontend with better UI, UX is a later priority	ANotFox	09/04/25, 1:23 am
Test Complete.	Sanskargithub07	08/04/25, 10:41 pm
Branch change test.	Sanskargithub07	08/04/25, 10:40 pm

Figure 6: GitHub commits

The screenshot shows the Stripe dashboard in a browser window. The URL is `dashboard.stripe.com/test/settings/developers`. The page title is "Sandbox" and the sub-header says "You're testing in a sandbox — your place to experiment with Stripe functionality." A purple button on the right says "Get your live account". The main navigation bar includes "Overview", "Webhooks", "Events", "Logs" (which is selected), "Errors", "Inspector", "Blueprints", and "Shell". Below the navigation is a search bar and filter options: "Filter by resource ID...", "Date", "Status", "HTTP method", "POST, DELETE", "API endpoint", "IP address", "More filters", "Send feedback", "Copy link", "Clear filters", and "Reset filters". The main content area is titled "Logs > req_n4RhZaQE6a94m8". It lists several log entries for "POST /v1/checkout/sessions" requests. Each entry shows a green checkmark, the status code (200), the request method, the endpoint, and the timestamp (e.g., 5:56:05 PM). The last entry is timestamped "Apr 25, 2025". To the right of the log list is a detailed view of the last log entry, showing Request ID (req_n4RhZaQE6a94m8), HTTP status code (200), Time (May 5, 2025, 5:56:05 PM), IP address (103.27.167.91), API version (2025-03-31.basil), Idempotency key (27a12822-8caa-4ea8-8eb8-3a65e566e6e1), User agent (Stripe/v1 JavaBindings/29.0), and API key (sk_test_...64pXYw). Below this is a "Response body" section with some JSON output.

Figure 9: Stripe dashboard JSON output 1

This screenshot is similar to Figure 9, showing the Stripe dashboard logs for POST /v1/checkout/sessions requests. The expanded view of the last log entry shows the full JSON response body. The JSON object includes fields like id, object (checkout.session), adaptive_pricing, automatic_tax, and line items. The line items array contains objects with details such as amount, currency, description, and product.

```

{
  "id": "cs_test_aIKJS20neP2hNjHdiJdVbrg7FCPMP9uEXLpbZg6QkkSyFcCnVnAELR",
  "object": "checkout.session",
  "adaptive_pricing": {
    "enabled": true,
  },
  "automatic_tax": {
    "enabled": false,
    "liability": null,
    "provider": null,
    "rate": null
  },
  "allow_promotion_codes": null,
  "amount_subtotal": 490031,
  "amount_total": 490031,
  "amount_total_cents": 490031
}

```

Figure 10: Stripe dashboard JSON output 2

This screenshot shows the Stripe dashboard logs for POST /v1/checkout/sessions requests. The expanded view of the last log entry shows a different set of JSON fields, including billing_address_collection, cancel_url, client_reference_id, client_secret, collected_information, consent, consent_collection, created, currency, currency_conversion, custom_fields, and custom_text.

```

{
  "billing_address_collection": null,
  "cancel_url": "http://localhost:5173/payment/failed",
  "client_reference_id": null,
  "client_secret": null,
  "collected_information": null,
  "consent": null,
  "consent_collection": null,
  "created": 1746467765,
  "currency": "usd",
  "currency_conversion": null,
  "custom_fields": [],
  "custom_text": {
    "after_submit": null,
    "shipping_address": null,
    "submit": null
  }
}

```

Figure 11: Stripe dashboard JSON output 3

The screenshot shows the Stripe dashboard in a browser window. The URL is `dashboard.stripe.com/test/settings/developers`. The page title is "Sandbox" and the sub-header says "You're testing in a sandbox — your place to experiment with Stripe functionality." A blue button in the top right corner says "Get your live account". The main navigation bar includes "Overview", "Webhooks", "Events", "Logs", "Errors", "Inspector", "Blueprints", and "Shell". The "Logs" tab is selected. A search bar and filter options are at the top of the log table. The log table shows requests from today, with one entry expanded to show JSON details. The expanded log entry is:

```

Logs > req_UGMNRNuxSMhJ
{
  "cancel_url": "http://localhost:5173/payment/failed",
  "line_items": [
    {
      "0": {
        "price_data": {
          "currency": "usd",
          "product_data": {
            "name": "A Soldier of the Great War",
          },
          "unit_amount": "259552",
        },
        "quantity": "4",
      }
    }
  ],
  "mode": "payment",
  "nauman_merchant_taxes": []
}

```

Figure 12: Stripe dashboard JSON output 4

This screenshot is identical to Figure 12, showing the Stripe dashboard with the same URL and interface. The log table shows the same sequence of requests, with the last request expanded to show JSON details. The expanded log entry is:

```

Logs > req_bkhPm5tDL1E9K
{
  "1": {
    "billing_details": {
      "address": {
        "country": "IN",
      },
      "email": "ss736@snu.edu.in",
      "name": "ABC ABC",
    },
    "card": {
      "cvc": "****",
      "exp_month": "06",
      "exp_year": "28",
      "number": "*****4242",
    },
    "guid": "7ea2f221-aa38-453a-aecd-060d35fd50063aa689",
  }
}

```

Figure 13: Stripe dashboard JSON output 5

This screenshot is identical to Figures 12 and 13, showing the Stripe dashboard with the same URL and interface. The log table shows the same sequence of requests, with the last request expanded to show JSON details. The expanded log entry is:

```

Logs > req_bkhPm5tDL1E9K
{
  "cvc": "****",
  "exp_month": "06",
  "exp_year": "28",
  "number": "*****4242",
},
"guid": "7ea2f221-aa38-453a-aecd-060d35fd50063aa689",
"key": "pk_test *****",
"mid": "7e2dd9a7-600e-49f6-a359-f876c53aca787bf3d2",
"payment_user_agent": "stripe.js/ca98f11090; stripe-js-v3/ca98f11090; checkout",
"sid": "6aebac34-fe50-46c6-a0ff-846af467901c25b847",
"type": "card",
}

```

Figure 14: Stripe dashboard JSON output 6

The screenshot shows the Postman application interface. On the left, there's a sidebar with a 'History' section listing various API requests made throughout the day and on April 25. In the main area, a specific POST request to `http://localhost:8080/api/auth/login` is selected. The 'Body' tab is active, showing a JSON payload:

```

1 {
2   "usernameOrEmail": "Sanskar",
3   "password": "qwerty"
4 }

```

Below the body, the response status is 200 OK with a response time of 459 ms and a size of 626 B. The response body contains a JWT token:

```

1 {
2   "accessToken": "eyJhbGciOiJIUzI1NiJ9.eyJsb2dpblR5cGU1OjJ1c2VybmtZSIsInN1YiI6IjNhbnNrYXJiIiLCjpxQj0je3NDYi
3   "tokenType": "Bearer"
4 }

```

Figure 15: Postman login JWT token response

This screenshot shows another Postman session. The history on the left includes various GET requests for filtering, pagination, and sorting. A specific GET request to `http://localhost:8080/api/book/filtering&pagination&sorting?page=0&size=10&sort=%5B%7B%22field%...` is selected. The 'Body' tab displays the response content, which is a JSON array of books:

```

1 "content": [
2   {
3     "id": 422,
4     "isbn": "0940322978",
5     "authorName": "James McCourt",
6     "bookName": "Mazowiecki (New York Review Books Classics)",
7     "price": 4991.72458752954,
8     "quantity": 44,
9     "publicationDate": "2002-01-01",
10    "availability": "IN_STOCK",
11    "publisher": "New York Review of Books",
12    "imageURL": "http://images.amazon.com/images/P/0940322978.01.1zzzzzzz.jpg",
13    "genre": null
14  },
15  {
16  }
]

```

Figure 16: Postman pagination, filtering and sorting response

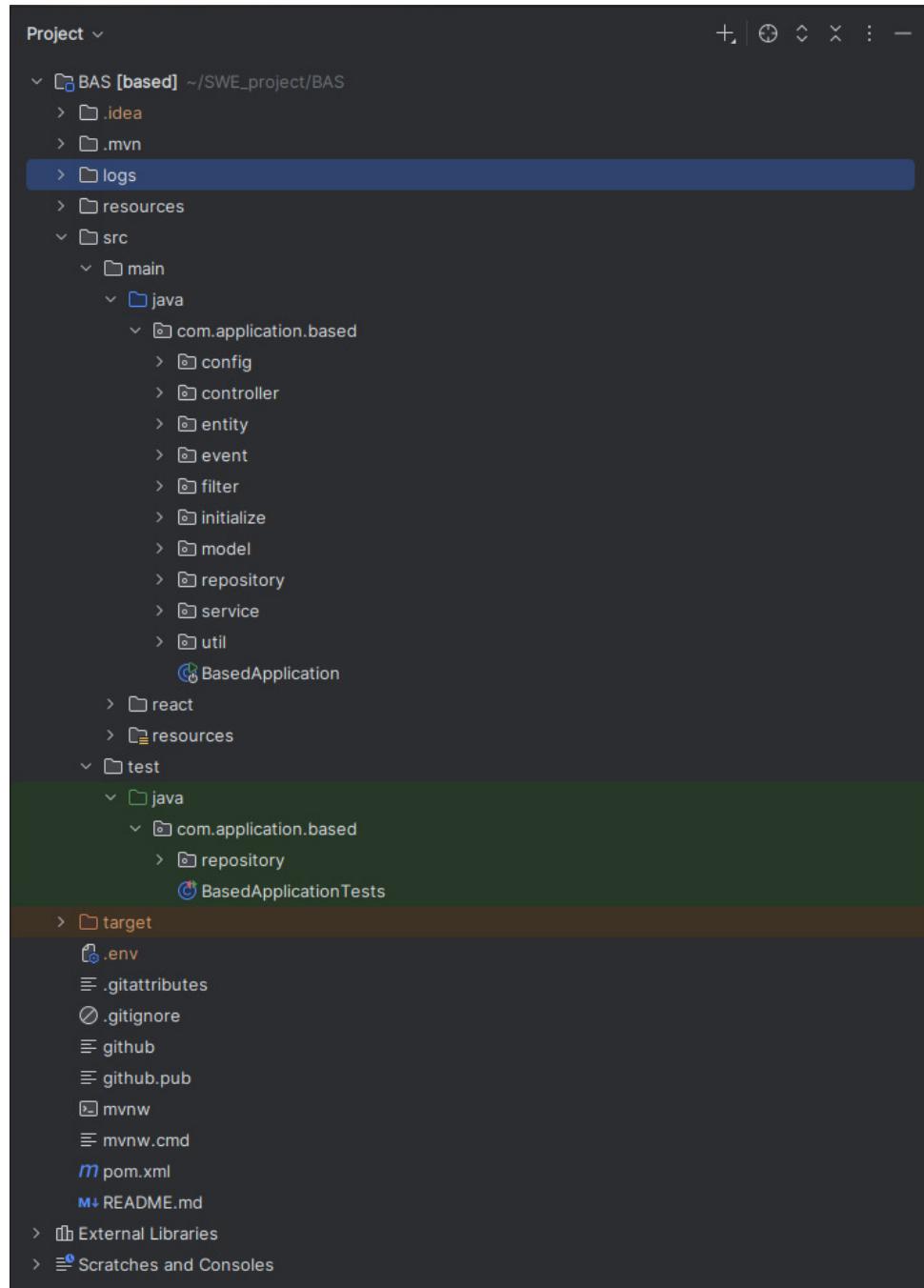


Figure 17: project structure

```

mysql> show tables;
+-----+
| Tables_in_based_db |
+-----+
| blacklistedjwttoken
| book_request
| book_request_requester
| book_request_requesters
| book_sequence
| books
| cart
| invitation_codes
| order_items
| orders
| password_reset_token
| roles
| stock
| stock_sequence
| user_sequence
| users
| users_roles
| verification_token
+-----+
18 rows in set (0.00 sec)

```

Figure 18: All DB relations

books							
	id	author_name	genre	image_url	isbn	price	publication_year
	102	Mark P. O. Morford	IN STOCK	Classical Mythology	0195153448.01.LZZZZZZZ.jpg	3735.0302873984314	2002-01-01
	103	Richard Bruce Wright	IN STOCK	Clara Callan	http://images.amazon.com/images/P/0082005018.01.LZZZZZZZ.jpg	0082005018	935.1751154661836
	104	Carlo D'Este	IN STOCK	In Search In Normandy	http://images.amazon.com/images/P/0086973129.01.LZZZZZZZ.jpg	0086973129	2861.4885258816403
	105	Gina Bari Kolata	IN STOCK	The Flu: The Story of the Great Influenza Pandemic of 1918 and the Search for the Virus That Caused It	http://images.amazon.com/images/P/0374157065.01.LZZZZZZZ.jpg	0374157065	1676.6186078403214
	106	E. J. W. Barber	IN STOCK	Mummin of Urinch!	http://images.amazon.com/images/P/0393045218.01.LZZZZZZZ.jpg	0393045218	3803.971822207357
	107	Amy Tan	IN STOCK	The Kitchen God's Wife	http://images.amazon.com/images/P/0399135782.01.LZZZZZZZ.jpg	0399135782	3567.222949517778
	108	Robert Cowley	IN STOCK	What If?: The World's Foremost Military Historians Imagine What Might Have Been	http://images.amazon.com/images/P/0425176428.01.LZZZZZZZ.jpg	0425176428	2058.167866588686
	109	Scott Turow	IN STOCK	Pleading Guilty	http://images.amazon.com/images/P/0671070432.01.LZZZZZZZ.jpg	0671070432	521.6666437751459
	110	David Cordingly	IN STOCK	Under the Black Flag: The Romance and the Reality of Life Among the Pirates	http://images.amazon.com/images/P/0879425608.01.LZZZZZZZ.jpg	0879425608	1879.2601679821994
	111	Ann Beattie	IN STOCK	Where You'll Find Me: And Other Stories	http://images.amazon.com/images/P/074322678X.01.LZZZZZZZ.jpg	074322678X	261.37583233114435
	112	David Adams Richards	IN STOCK	Nights Below Station Street	http://images.amazon.com/images/P/0771074670.01.LZZZZZZZ.jpg	0771074670	2871.7111418857257
	113	Adam Lebor	IN STOCK	Hitler's Secret Bankers: The Myth of Swiss Neutrality During the Holocaust	http://images.amazon.com/images/P/009652123X.01.LZZZZZZZ.jpg	009652123X	5392.12087937395
	114	Sheila Heti	IN STOCK	The Middle Stories	http://images.amazon.com/images/P/0887841740.01.LZZZZZZZ.jpg	0887841740	3201.48328203471
	115	R. J. Kaiser	IN STOCK	Jane Doe	http://images.amazon.com/images/P/1552041778.01.LZZZZZZZ.jpg	1552041778	3473.7211327721398
	116	Jack Canfield	IN STOCK	A Second Chicken Soup for the Woman's Soul (Chicken Soup for the Soul Series)	http://images.amazon.com/images/P/1556746218.01.LZZZZZZZ.jpg	1556746218	4879.58743611163

Figure 19: All available books in DB

A.2.2 Frontend Screenshots

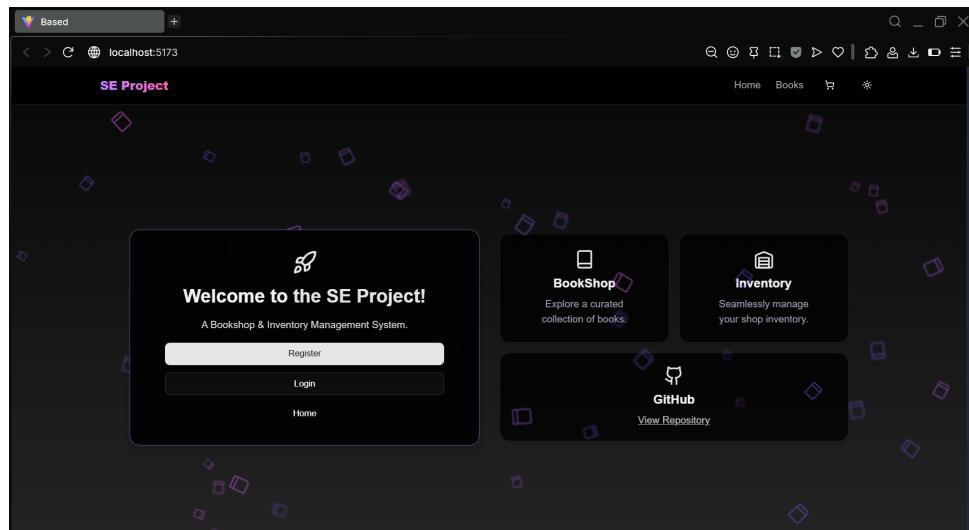


Figure 20: Landing page interface

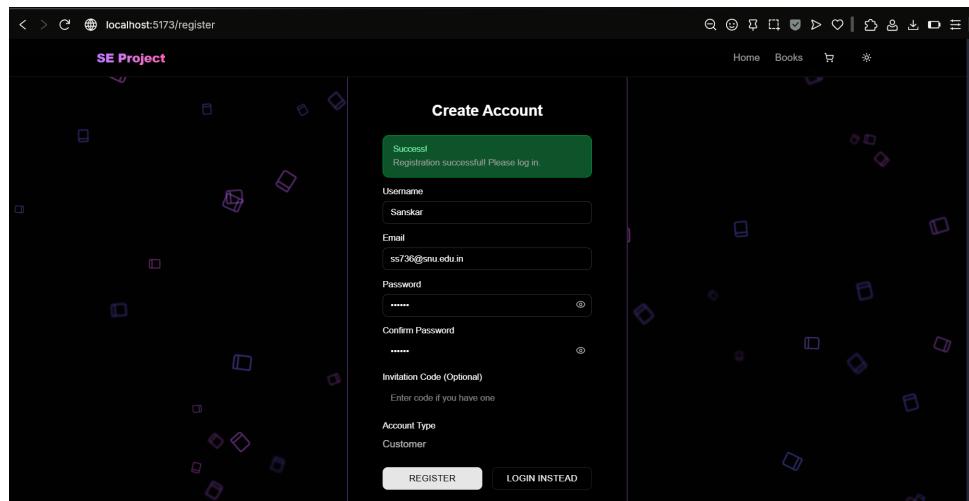


Figure 21: Register page interface

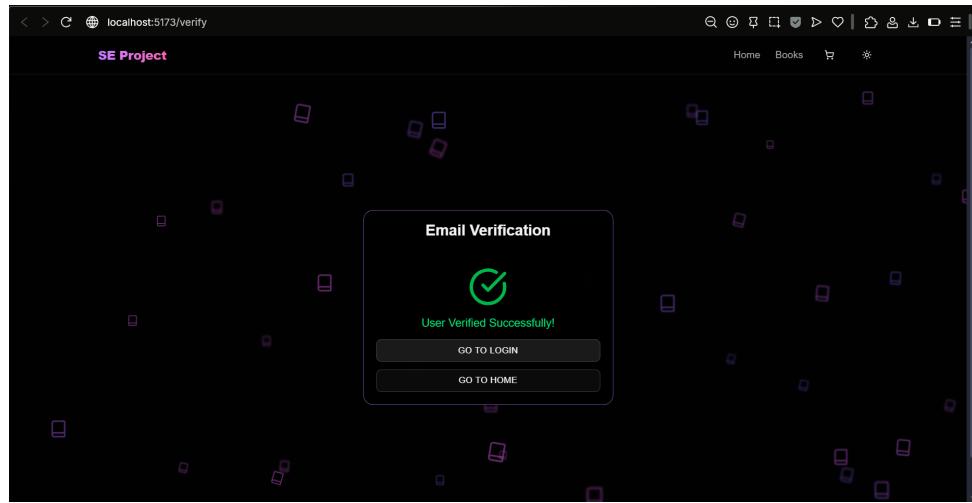


Figure 22: Email verification success page

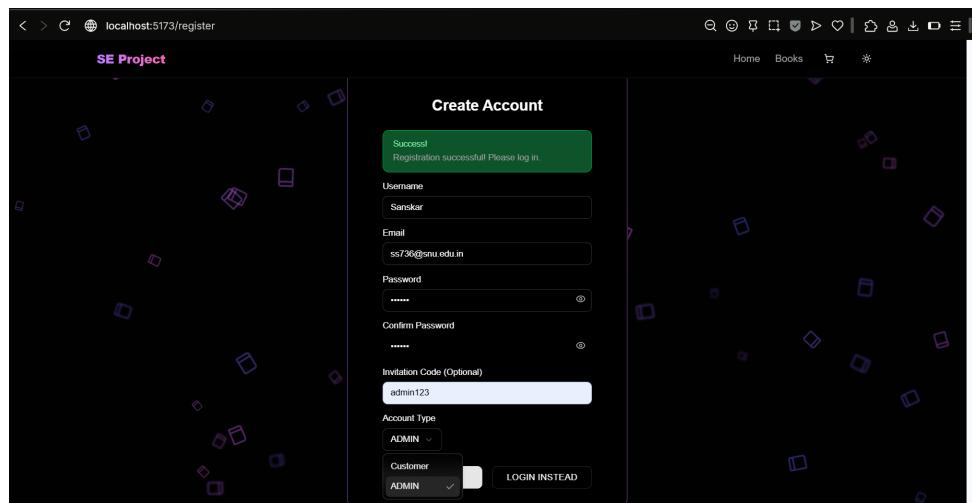


Figure 23: Creating an account with invitation code for admin

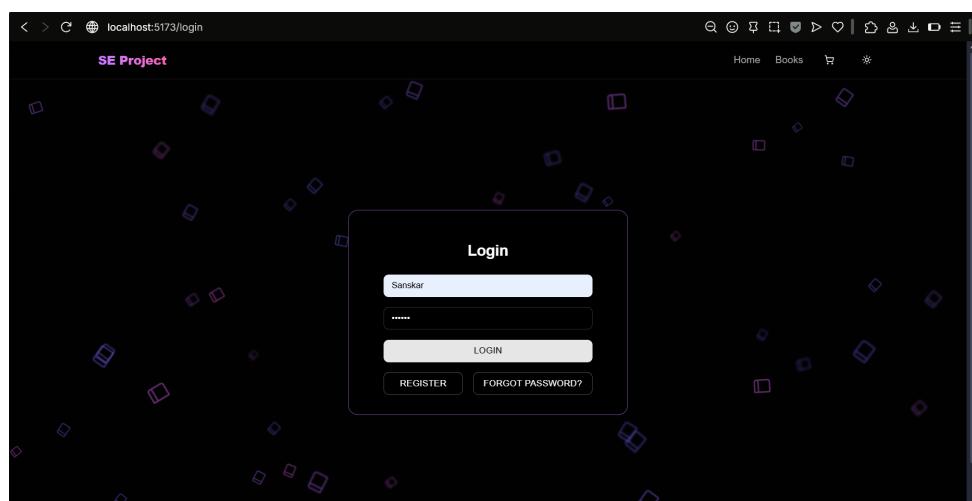


Figure 24: Login page

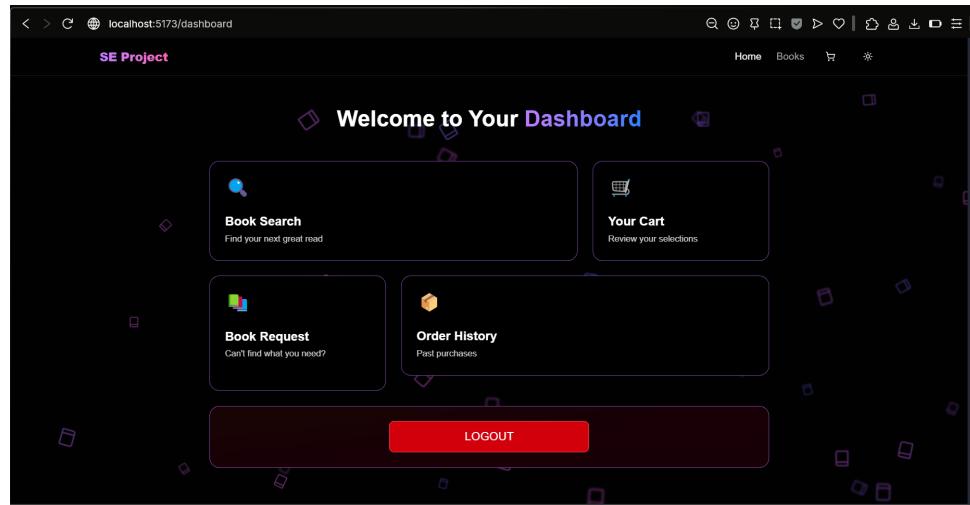


Figure 25: User dashboard

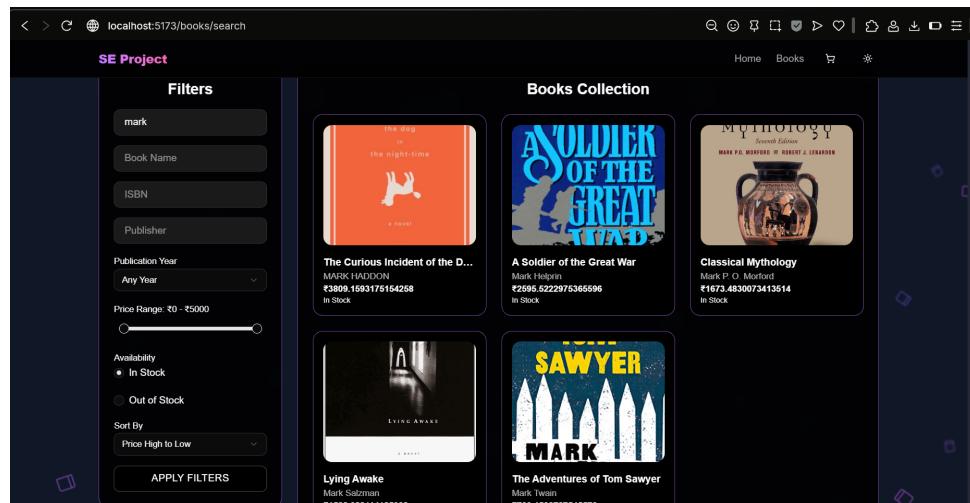


Figure 26: Book search and filter page

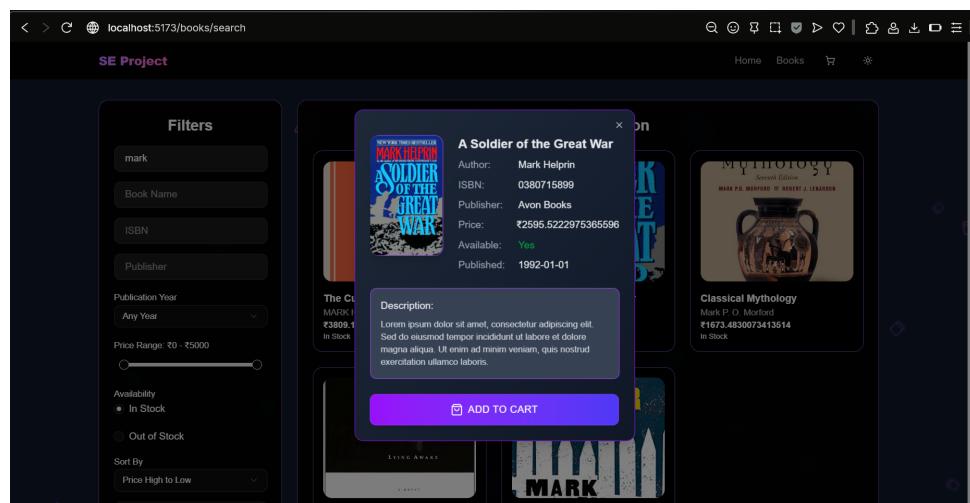


Figure 27: Add to cart dialog

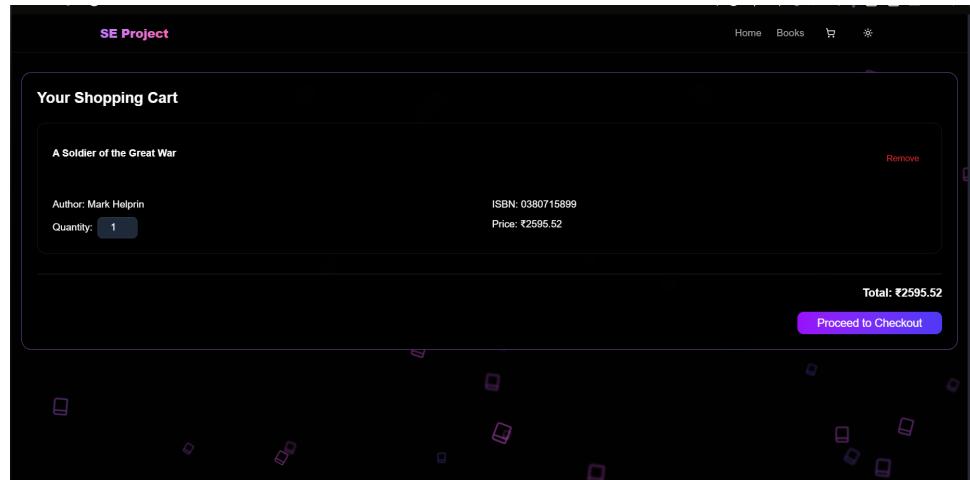


Figure 28: Shopping cart with quantity 1

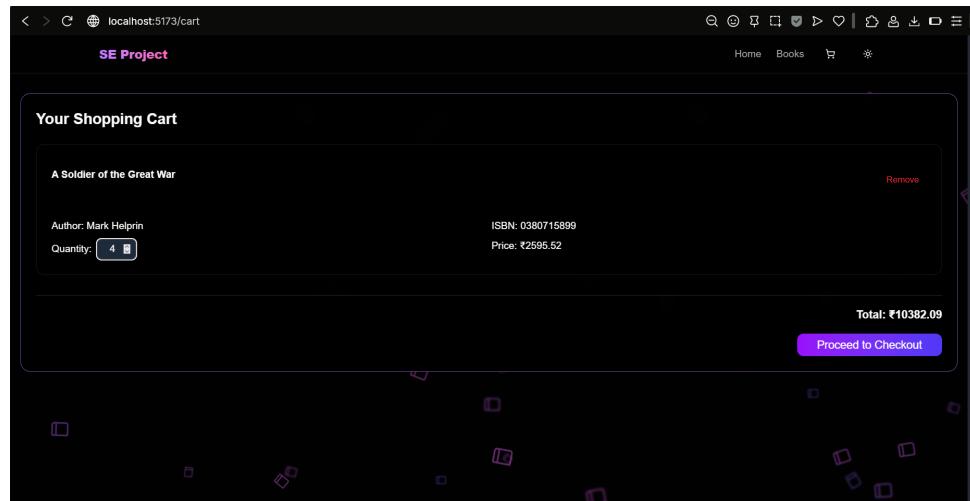


Figure 29: Shopping cart with quantity 4

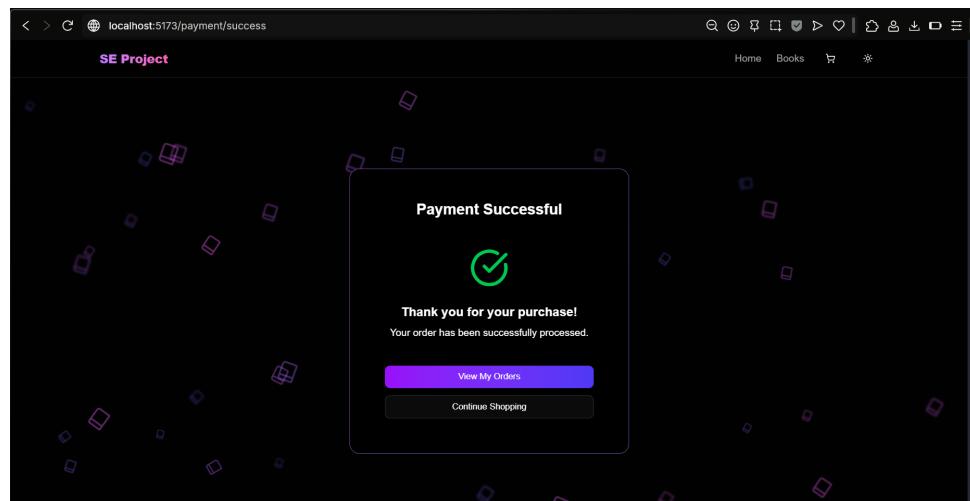


Figure 30: Payment success page

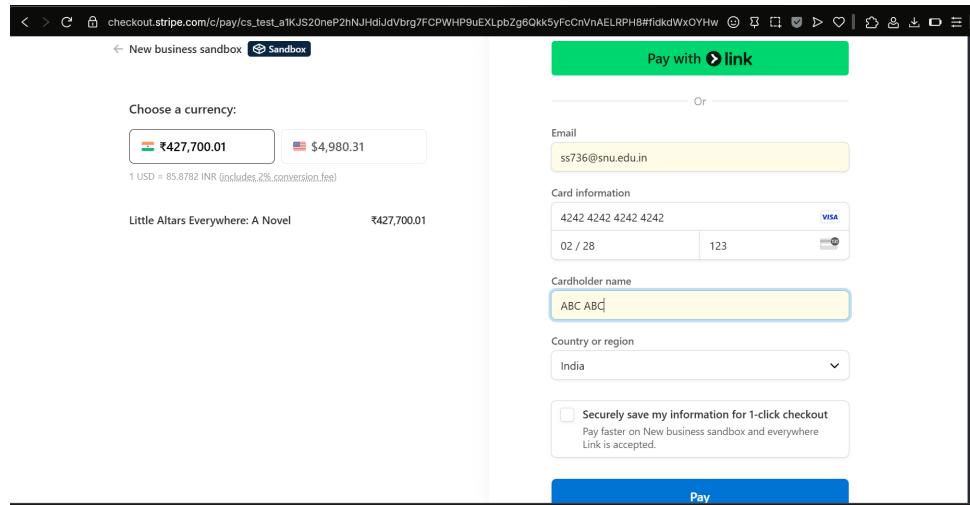


Figure 31: Stripe Payment redirect with fake card info.

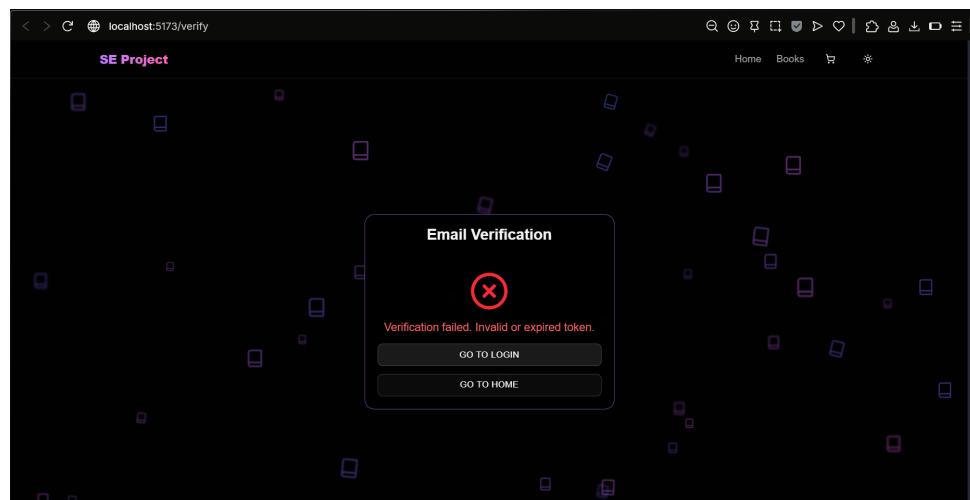


Figure 32: Email verification failure page

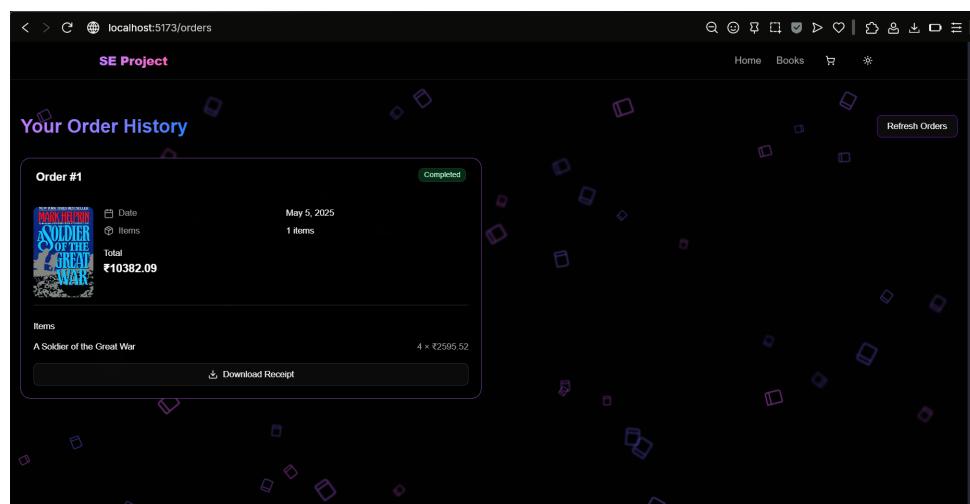


Figure 33: User order history page

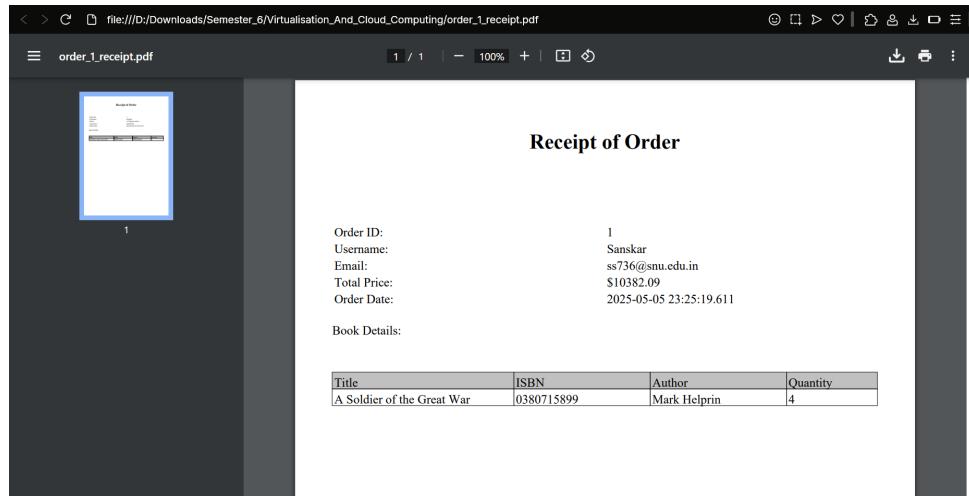


Figure 34: Receipt for order in PDF format

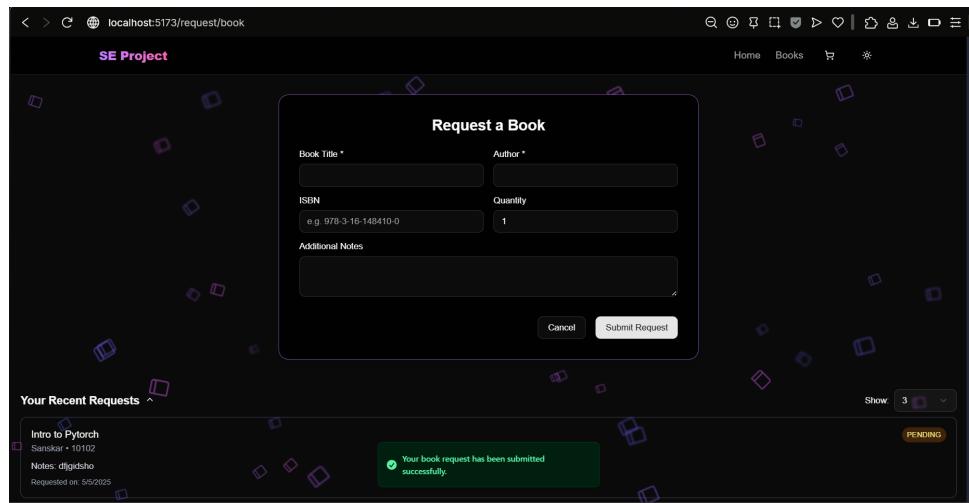


Figure 35: Request book page

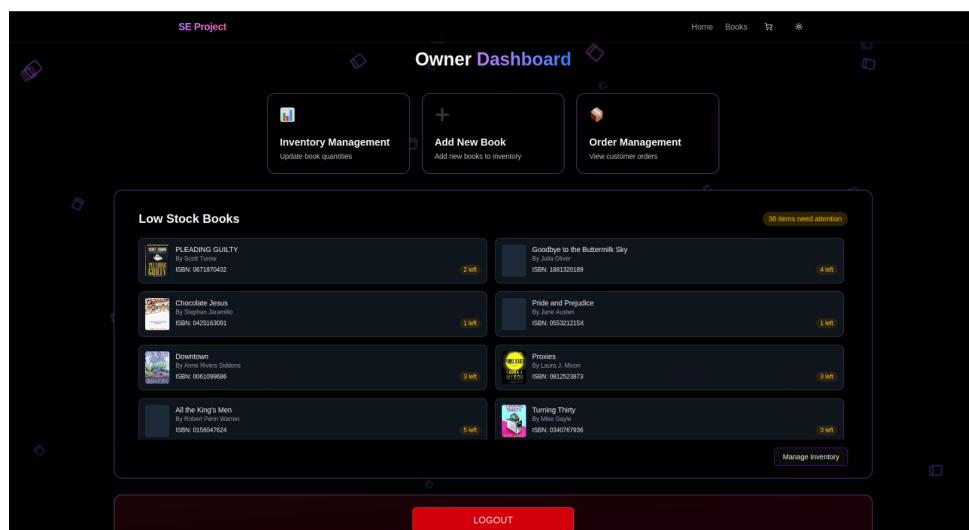


Figure 36: Low stock books and owner dashboard

A.2.3 Logging Screenshot after successful book search

```
Run > BasedApplication <


Console Beans Health Mappings Environment



```

23:23:08 WARN o.s.c.c.i.InitializerDetailsBeanManagerConfigurableInitializerDetailsBeanManagerConfigurable - Global AuthenticationManager configured with an AuthenticationProvider bean. UserDetailsService beans will be used.
23:23:09 INFO o.d.j.r.query.QueryDataSourceFactory - Hibernate is in classpath; If applicable, HQL parser will be used.
23:23:09 INFO o.a.s.w.b.BasedWebApplicationConfiguration - spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view if you don't want this behavior.
23:23:09 INFO o.a.s.w.e.TomcatWebServer - Tomcat started on port 8080 (http://localhost:8080) with context path '/'
23:23:09 INFO o.a.s.w.b.BasedWebApplication - Started BasedWebApplication in 2.797 seconds (process running for 3.358)
23:23:10 INFO o.a.c.c.{Tomcat,localhost:1} - Initializing context dispatcherServlet 'dispatcherServlet'
23:23:10 INFO o.a.s.w.b.DispatcherServlet - Initializing servlet 'dispatcherServlet'
23:24:02 INFO o.a.s.w.b.DispatcherServlet - Completed initialization in 1 ms
usernamePasswordAuthenticationToken [Principal=aniv, authentication-based-service-userInfoDetails@5a22a05b, Credentials=[PROTECTED], Authenticated=true, Details=null, Granted Authorities=[ROLE_CUSTOMER]]
aniv
username
23:24:04 DEBUG c.a.based.service.JwtService - Generating JWT token for aniv** using username authentication eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1ZGZyMjQ1NjY1NjIwIiwiYXpwIjoxMjAxNTkxNjQ1MjE1MDIzNDYwMjAwODZyL98sifNqejJePob2fjg7i1tpVtbAai5m_wPlQ
23:24:05 DEBUG c.a.based.service.JwtService - Validating JWT token
23:24:06 DEBUG c.a.based.service.JwtService - Executing paginated search - Page: 0, Size: 6
23:24:06 DEBUG c.a.based.service.JwtService - Serializing PageImpl instances as-is is not supported, meaning that there is no guarantee about the stability of the resulting JSON
For a stable JSON structure, please use Spring Data's Pageable (globally via @EnableSpringDataWebSupport(T��이스SerializationMode = VIA.DTO))
23:24:12 DEBUG c.a.based.service.JwtService - Validating JWT token
23:24:13 INFO c.a.based.service.BookServiceImpl - Executing paginated search - Page: 0, Size: 6
23:25:09 DEBUG c.a.based.service.JwtService - Validating JWT token
23:25:09 INFO c.a.based.service.BookServiceImpl - Executing paginated search - Page: 0, Size: 6

```


```

Figure 37: System logging configuration

A.2.4 Testing Screenshots

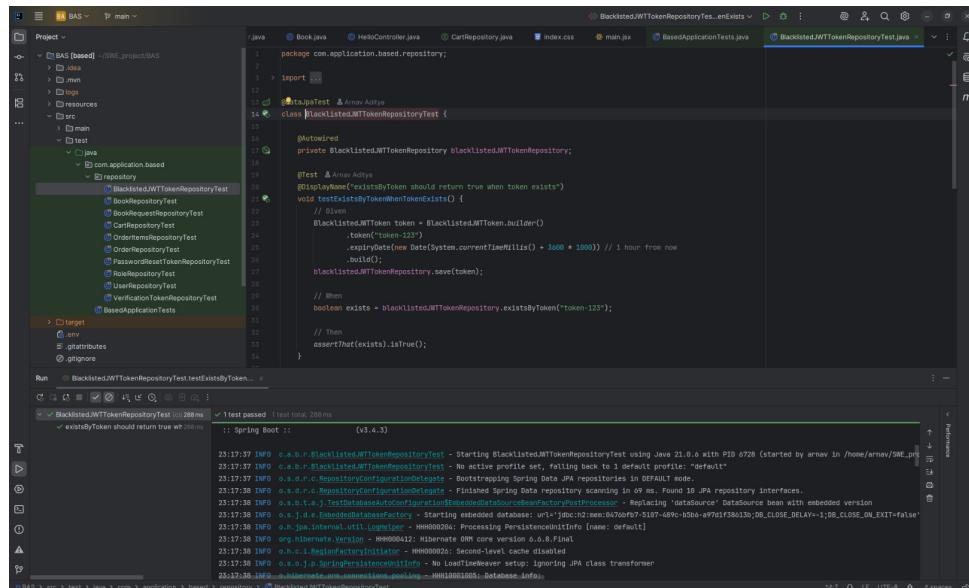


Figure 38: Unit test for repository layer (Blacklisted JWT Token repository success)

The screenshot shows an IDE interface with a project structure on the left and a code editor on the right. The code editor displays a Java test class named `BlacklistedJWTTokenRepositoryTest`. The test method `testExistsByTokenWhenTokenExists()` is annotated with `@Test` and `@DisplayName("existsByToken should return true when token exists")`. It creates a token with an expiration date 1 hour from now and saves it to the repository. Then, it checks if the repository returns true for the same token. The test fails with the error message: "org.junit.AssertionFailedError: Expecting value to be true but was false". The actual value is shown as `false`. The test method also includes a `@Test` block for testing when the token does not exist, which is currently commented out.

```
(@DataJpaTest @RunWith(SpringRunner.class)
public class BlacklistedJWTTokenRepositoryTest {
    @Autowired
    private BlacklistedJWTTokenRepository blacklistedJWTTokenRepository;

    @Test @DisplayName("existsByToken should return true when token exists")
    void testExistsByTokenWhenTokenExists() {
        // Given
        BlacklistedJWTToken token = BlacklistedJWTToken.builder()
            .token("Token-123")
            .expiringDate(new Date(System.currentTimeMillis() + 3600 * 1000)) // 1 hour from now
            .build();
        blacklistedJWTTokenRepository.save(token);

        // When
        boolean exists = blacklistedJWTTokenRepository.existsByToken("Token-123");

        // Then
        assertThat(exists).isTrue();
    }

    @Test @DisplayName("existsByToken should return false when token does not exist")
    void testExistsByTokenWhenTokenDoesNotExist() {
        // Given
        // When
        // Then
    }
})
```

Figure 39: Unit test for repository layer (Blacklisted JWT Token repository fail)

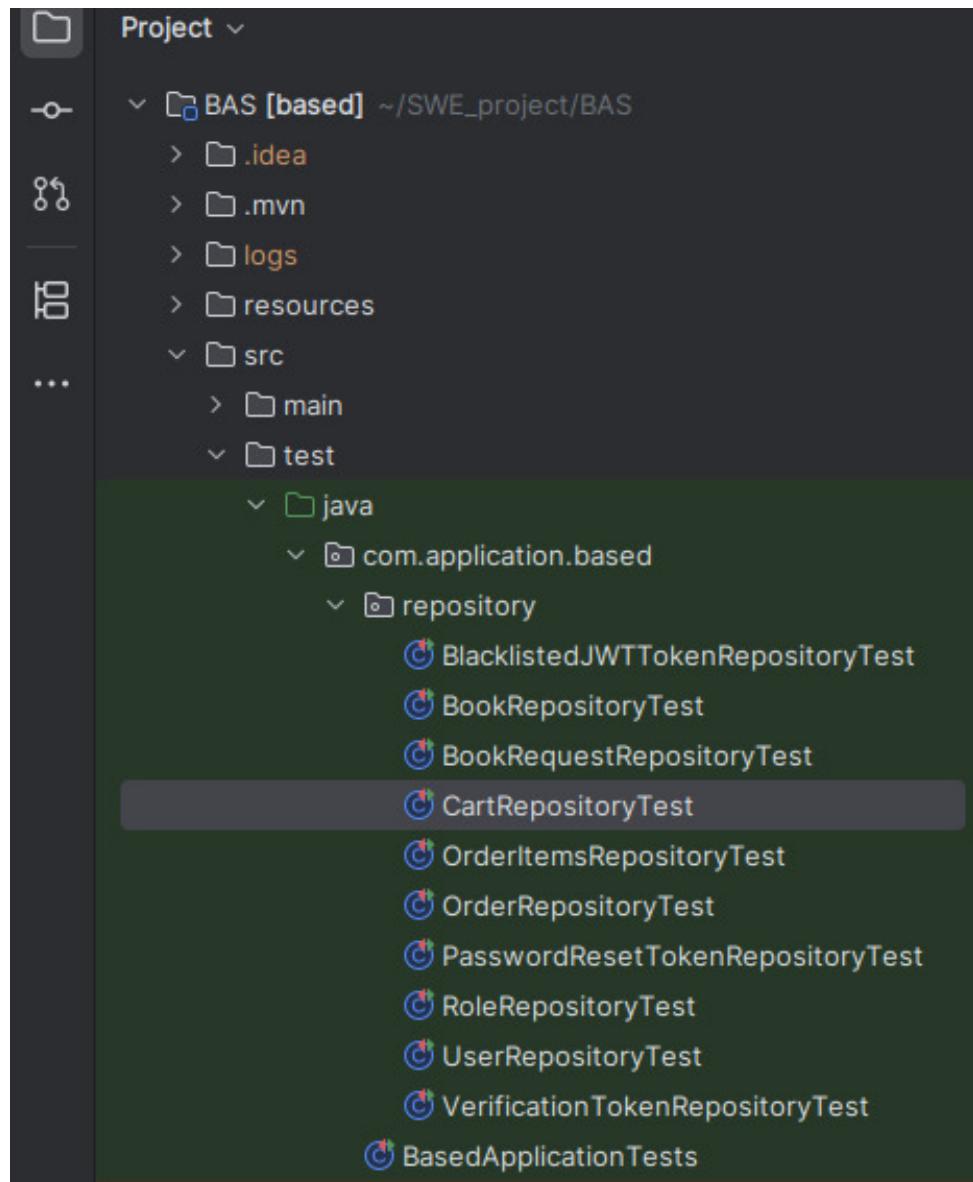


Figure 40: All repository layer tests