**Contextual Problem Statement****: Bookstore App**

Contents

[Introduction 2](#_Toc181892311)

[Preface 2](#_Toc181892312)

[Sprint-Based Approach 2](#_Toc181892313)

[Objectives 2](#_Toc181892314)

[Requirements 3](#_Toc181892315)

[Deliverables 4](#_Toc181892316)

[Versioning 4](#_Toc181892317)

[Tech Stack 4](#_Toc181892318)

[Tools and Libraries 5](#_Toc181892319)

[Testing: 6](#_Toc181892320)

[TDD/BDD Approach: 6](#_Toc181892321)

[Other Considerations: 6](#_Toc181892322)

[Sprint 1 7](#_Toc181892323)

[Objective: 7](#_Toc181892324)

[Solution Outline 7](#_Toc181892325)

[CLI Application Screens 8](#_Toc181892326)

[Sprint 2 9](#_Toc181892327)

[Objective: 9](#_Toc181892328)

[Solution Outline 9](#_Toc181892329)

[Project Structure 11](#_Toc181892330)

[SQL Script to Create Database Schema 12](#_Toc181892331)

[CLI Application Screens 12](#_Toc181892332)

[Sprint 3 15](#_Toc181892333)

[Objective: 15](#_Toc181892334)

[Solution Outline 15](#_Toc181892335)

[Project Structure 16](#_Toc181892336)

[CLI Application Screens 18](#_Toc181892337)

[Sprint 4 21](#_Toc181892338)

[Objective: 21](#_Toc181892339)

[Solution Outline 22](#_Toc181892340)

[Project Structure 24](#_Toc181892341)

[API Endpoint for Authentication and Authorization 25](#_Toc181892342)

[API Endpoints for Admin Options 26](#_Toc181892343)

[API Endpoints for User Options 31](#_Toc181892344)

[Sprint 5 36](#_Toc181892345)

[Objective: 36](#_Toc181892346)

[Solution Outline 36](#_Toc181892347)

[React Application Screens 40](#_Toc181892348)

[References: 42](#_Toc181892349)

# Introduction

Develop an online bookstore application that allows users to browse, search, and purchase  books. The application should manage user accounts, book inventories, and orders.

# Preface

The Bookstore application is a comprehensive, hands-on project designed to integrate the bulk of the tech stack covered during the training phase. This project will be developed over five iterative sprints, each building upon the previous one to enhance functionality and introduce new features.

# Sprint-Based Approach

Each sprint will involve refactoring the existing codebase, introducing new concepts, and ensuring that the application grows in complexity and capability. This iterative approach mirrors real-world software development practices, allowing you to experience the process of incremental enhancement and continuous improvement. By the end of each sprint, the deliverable is a fully functional application, complete with the latest features and refinements.

# Objectives

* **Sprint 1**: Establish the core functionality with a CLI-based login system, leveraging Java’s object-oriented programming principles, basic data types, string manipulation, control flow, exception handling, and arrays.
* **Sprint 2**: Enhance the application by integrating more advanced features such as data persistence with JDBC, refactor and extend the application by incorporating JUnit 5 and Mockito for testing, expand user functionalities, implement data structures with the Collections Framework and incorporate Streams and Lambda Expressions in Java.
* **Sprint 3**: Refactor the application to store Personally Identifiable Information (PII) I in the PostgreSQL database in encrypted format, implement role-based authentication and authorization, implement the Book Order feature for Normal Users and employ Behaviour-Driven Development (BDD) with Cucumber for Java.
* **Sprint 4**: Reengineer the application with Spring Boot. Implement REST API endpoints with Spring REST. Incorporate Spring Data JPA for persistence. Use Spring Testing framework for unit, system and integration tests. Document the REST API endpoints with Swagger API. Dockerize the application and setup a Jenkins pipeline push the image to a container repository and deploy it to a web server. Utilize JMeter for performance and load testing.
* **Sprint 5**: Build the Web UI with the latest version of React.js and TypeScript. Utilize Next.js for tooling and ESLint for linting. Consume the REST APIs produced in the previous Sprint. Update the pipeline to build, test and deploy the Front-End application. Testing can be carried out with Jest and React Testing Library and should include snapshot tests.

This journey will not only solidify your understanding of individual concepts but also teach you how to bring them together in a cohesive, well-structured application.

# Requirements

1. **User Management**:
   * Admins should be able to create accounts and manage user.
   * Admin users should be able to add, update, and remove books from the  inventory.
   * Normal users should be able to browse and search for books, change their password, place an order for a book, manage orders and view their order history.
2. **Book Management**:
   * Implement CRUD operations for books.
   * Each book should have attributes such as title, author, ISBN, price, and quantity.
3. **Order Management**:
   * Users should be able to browse and search for books.
   * Users should be able to add books to their cart and place orders.
   * Implement order processing and order history tracking.
4. **Testing**:
   * Develop **unit tests** for all major components using JUnit 5.
   * Use Mockito to create mock objects for testing database operations and user  interactions.
   * Implement **Behaviour-Driven Development (BDD) tests** using Cucumber for Java:

* Write Gherkin syntax scenarios to define the behaviour of the application.
* Create step definitions to map Gherkin steps to Java methods.
* Integrate Cucumber tests with JUnit 5 to run BDD tests as part of the test suite for the Back-End application.
* Integrate Cucumber tests with Jest to run BDD tests as part of the test suite for the Front-End application.
  + **Performance Testing:** Use JMeter to perform load and performance testing to ensure the application can handle high volumes of users and transactions.

1. **Exception Handling**:
   * Implement robust exception handling to manage various error scenarios.
2. **Persistence and Querying**:
   * Use JDBC and Spring Data JPA for data persistence and implement complex queries to retrieve data.
3. **Web User Interface (UI):**
   * The Front-End will be developed with React.js.
4. **Continuous Integration/Continuous Deployment (CI/CD):**
   * Implement a CI/CD pipeline using Jenkins.
   * Automate the process of building, testing, and deploying the application.
   * Ensure that new code changes are automatically tested and deployed to staging/production environments.
5. **Sample Data Loading:**
   * Load sample data for 50,000 users to test the application's scalability and performance.
   * Ensure the sample data includes a variety of user interactions to simulate real-world usage.
6. **Documentation:**
   * **UML Diagrams:** Create both structural (Class Diagrams, Object Diagrams etc.) and behavioural (Sequence Diagrams, Activity Diagrams etc.) UML diagrams to visually represent the system architecture and workflows.
   * **Swagger for REST APIs:** Document all REST APIs using Swagger to provide a clear and interactive API documentation for developers.
   * **User Flow Documentation:** Develop detailed user flow documents to outline the different user journeys within the application.
   * **4+1 View Architecture Document:** Create a comprehensive 4+1 view architecture document, which includes the Logical, Development, Process, and Physical views of the system along with scenarios to capture use cases and interactions.

# Deliverables

1. **Source Code**: Complete source code of the application.
2. **Test Cases**: Unit tests and test results demonstrating the correctness of the application.
3. **Documentation**: UML diagrams illustrating different facets of the application.

# Versioning

There will be two versions of the bookstore application. One that utilizes a CLI (Command Line Interface) and the other that features a front end developed with React.js and a back end with Spring Boot.

# Tech Stack

1. **Programming Language**

* **Java SE:** For core application development, object-oriented programming, and API interactions.

1. **Frameworks and Libraries:**

* **Spring Boot:** For creating the back end RESTful Web Services.
* **React.js:** For creating the UI and consuming the RESTful Web Services.

1. **Database**

* **PostgreSQL:** For storing user, book, and order data.

1. **Persistence Framework**

* **JDBC (Java Database Connectivity): For direct database interactions using SQL queries.**
* **Spring Data JPA:** For managing database operations, mapping Java objects to database tables, and performing CRUD operations.

1. **Build Tool**

* **Maven**: For project management, dependency management, and building the  application.

1. **Testing**

* **JUnit 5**: For writing and running unit tests.
* **Mockito**: For creating mock objects and performing unit tests with mock dependencies.
* **SpringBootTest**: Testing the Spring Boot back end.
* **Postman**: For Testing the REST APIs.

1. **IDE**

* **IntelliJ IDEA**: For development and debugging the CLI and Spring Boot applications.
* **Visual Studio Code**:  For development and debugging the React.js application.

1. **UML Modeling**

* **Draw.io (now https://app.diagrams.net/):** For creation of the UML diagrams.

# Tools and Libraries

1. **Java SDK**: Ensure you have the Java Development Kit (JDK) ver. 17 installed.
2. **Node**.js: Version 20.15.0 or later.
3. **postgresql**: JDBC driver for connecting Java applications to a PostgresSQL database
4. **Hibernate Core**: For ORM capabilities.
5. **Maven**: To manage dependencies and build the project.
6. NFRs & SLAs:

1. The system must support 1-page/5-page requests per second

1. The system will need to support 150 transactions per second.
2. Concurrent user session – 200
3. Page view times < 3s
4. Responsive web design - supported view port 767px (Mobile and Desktop)
5. CI&CD Pipeline(s) to be deployed via Jenkins | Docker.
6. Different Users should not be able to access each other’s data.
7. Secure coding standards to be followed.
8. User friendly error pages should be displayed to the end users
9. Validation of user input to be carried out in both the front as well as the back-end**.**
10. Automate Performance Testing with JMeter
11. **Only one session per user at a time** or **no concurrent session per user**. If the user tries to open a new session, then either an alert is shown or his previous session is closed.

# Testing:

1. Smoke and Sanity Testing
2. Unit Testing and Regression
3. System Testing
4. [JaCoCo](https://www.eclemma.org/jacoco/) - Code coverage > 90%

* Performance Test with JMeter
* Unit Test Reports with JUnit and Spring Boot Testing Framework
* Error and Exception handling
* Test Data preparation, management and data cleanup should be in place.

# TDD/BDD Approach:

* TDD with Junit to be followed for Java application development.
* TDD with Jest to be followed for ReactJS application development.
* Behaviour Driven Development (BDD) to utilize the Cucumber framework.

# Other Considerations:

* System should support upload of raw data in CSV format.
* At least 50,000 User records need to be stored

# Sprint 1

## Objective:

Develop a CLI-based login system for an online bookstore application, focusing on:

1. Allowing both admin and normal users to log in.
2. Holding user data in memory using arrays.
3. Providing additional privileges for admin users compared to normal users.
4. Utilizing object-oriented programming principles.
5. Employing Java data types and String APIs, classes and interfaces, control flow mechanisms, Enums, and arrays.

## Solution Outline

**1. Java's Object-Oriented Programming Principles**

* **Classes and Objects**: Create User, AdminUser, and NormalUser classes.
* **Inheritance**: AdminUser and NormalUser inherit from User. User implements UserInterface.
* **Encapsulation**: Use private fields and public getters and setters.

**2. Utilizing Java Data Types and String APIs**

* **Attributes**: Use appropriate data types for user attributes (e.g., String for usernames and passwords).
* **String Manipulation**: Handle user input and perform operations like trimming spaces or checking the length.

**3. Managing Control Flow in Java Applications**

* **User Input and Menus**: Use loops and conditionals to navigate user choices.
* **Login Process**: Implement the login process with options for different user types.

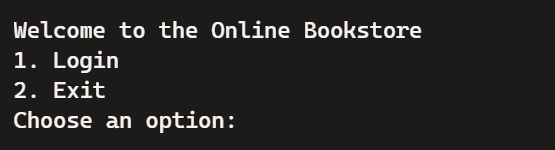
**5. Arrays**

* **In-memory Storage**: Store user data in arrays for quick lookup and access.

# CLI Application Screens

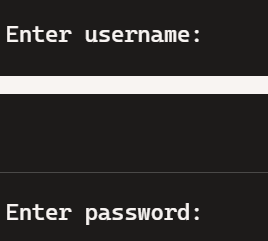
**Initial Screen**

When you run the application, the menu in the command prompt will look something like this:



**Login Screen**

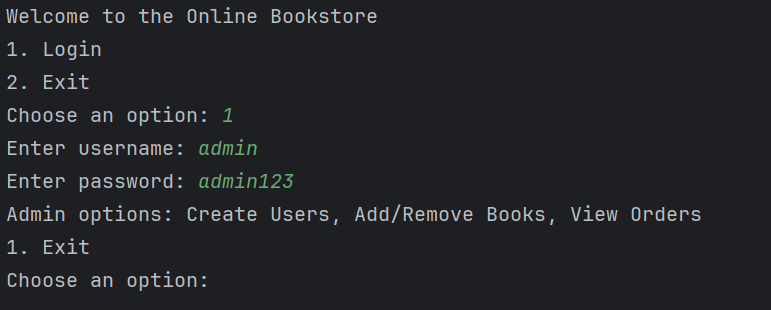
If you choose to log in, it will prompt you:



The admin/normal username and password should be hard coded in the application.

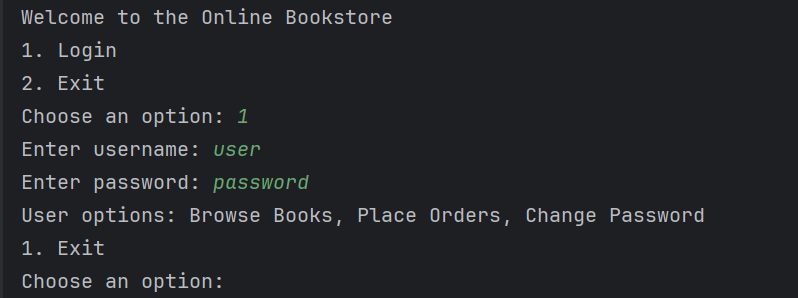
**Admin Menu**

If the login is successful, and the user is an admin, it might display:

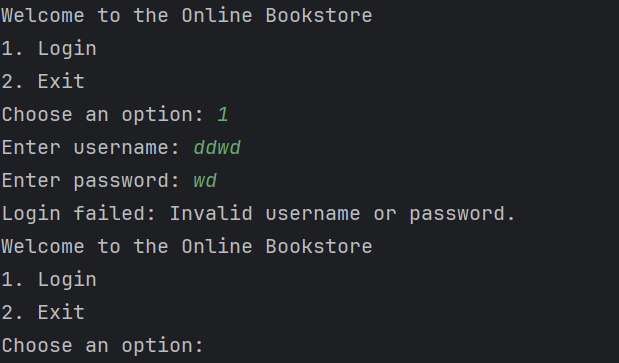


**User Menu (Normal User)**

If the user is a normal user, it could show:



When the login attempt fails, the screen would look something like this:



After the login failure message, the application will loop back to the main menu:

This ensures the user can try logging in again or exit the application.

**Logout**

After Choosing Exit, the application will close.

# 

# Sprint 2

## Objective:

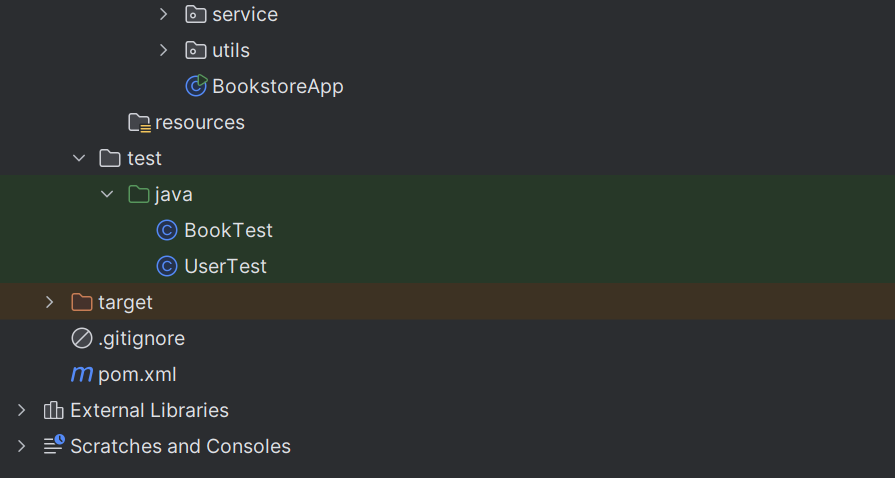
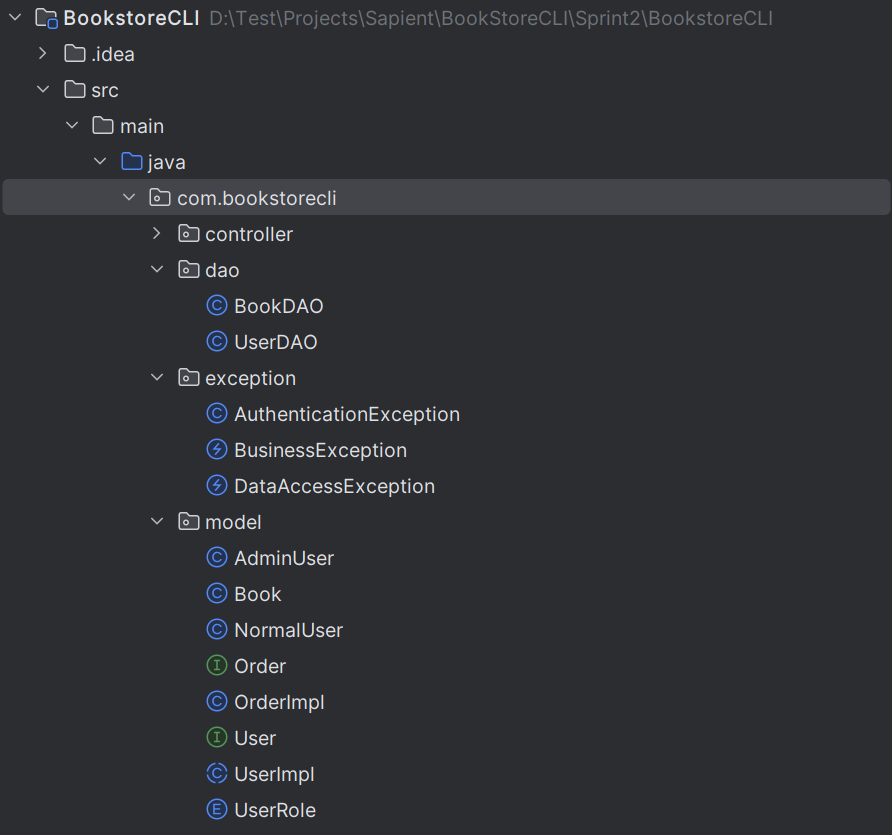
Enhance the "BookstoreCLI" application by integrating more advanced features and topics. In this Sprint, we will focus on:

1. Implementing CRUD operations for books.
2. Using Collections, Streams, and Lambda Expressions for efficient data manipulation.
3. Introducing JDBC for data persistence.
4. Writing unit tests with JUnit 5 and Mockito for mocking external dependencies.
5. Building a layered application where low-level exceptions are wrapped in custom exceptions and propagated to a higher layer. At this higher layer, these exceptions will be handled and a user-friendly error message will be printed to the screen.

## Solution Outline

1. **User Class Enhancements**
   * Extend the User class to include methods for managing user profiles.
   * Update AdminUser and NormalUser classes to reflect new functionalities.
2. **Book Class**
   * Create a Book class with attributes such as title, author, ISBN, price, and  quantity.
   * Implement getters and setters for these attributes.
3. **BookDAO and BookService**
   * Create a BookDAO class to handle CRUD operations for books using JDBC.
   * Create a BookService class to handle business logic for books.
4. **BookController**
   * Create a BookController class to handle user requests related to books.
5. **Collections and Streams**
   * Use List to store books and implement methods to add, update, remove, and  display books.
   * Use Streams and Lambda Expressions to filter, sort, and search books.
6. **Order Class**
   * Create an Order class to represent user orders.
   * Implement methods to add books to the cart, place orders and check total price.
7. **Exception Handling**
   * Enhance exception handling mechanisms to manage errors during CRUD  operations and user interactions.
   * Implement custom exception handling.
8. **Unit Testing**
   * Implement unit tests with JUnit 5 and mocking with Mockito.

# Project Structure



# SQL Script to Create Database Schema

The SQL Script to Create and populate the Database Schema has been provided. We are using a PostgreSQL database as the Data Store.

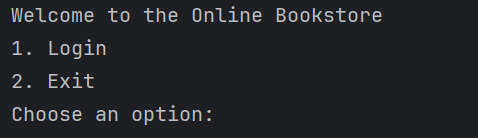
**Instructions**

1. **Create the Database:** Run this SQL script in your SQL client to create the bookstore database, its tables, and populate them with sample data.
2. **Update JDBC Connection**: Ensure your JDBC connection URL matches the bookstore  database and update the username and password in your code accordingly.

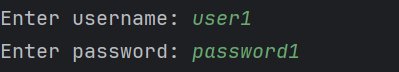
This script sets up the necessary tables and relationships for user management, book management, and order management.

# CLI Application Screens

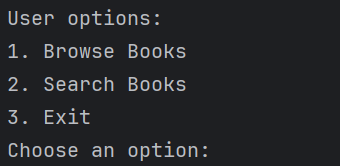
**Initial Screen**



**Login Screen**

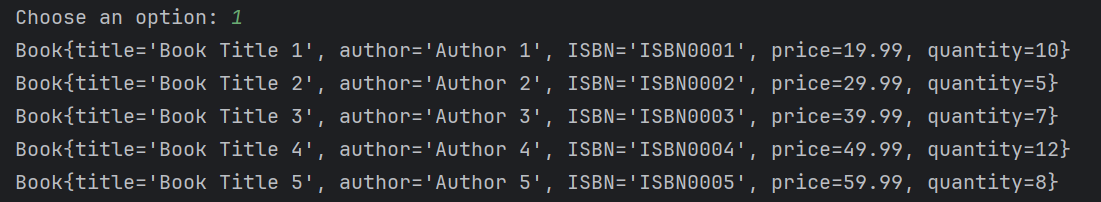


**User Menu (Normal User)**



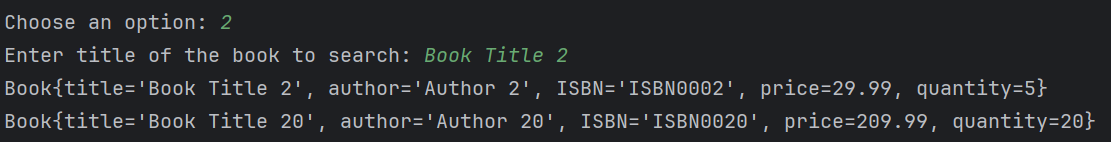
**User Browsing Books**

If the user chooses to browse books:



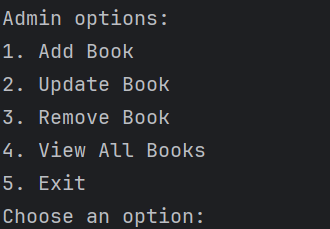
**User Searching for Books**

If the user chooses to search for books:



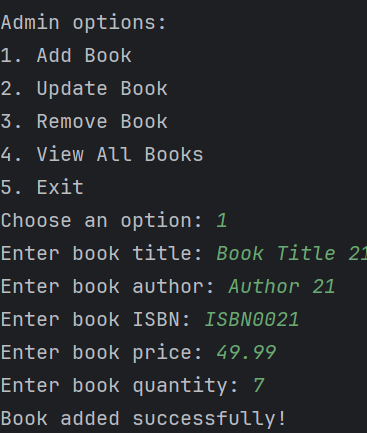
**Admin Menu**

If an admin user logs in, they will see:



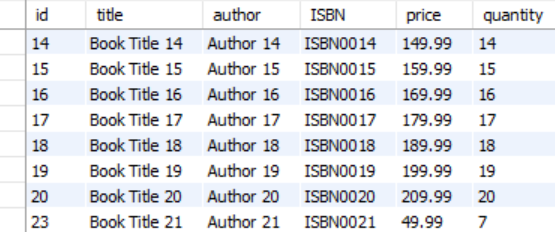
**Admin Adding a Book**

If the admin chooses to add a book:



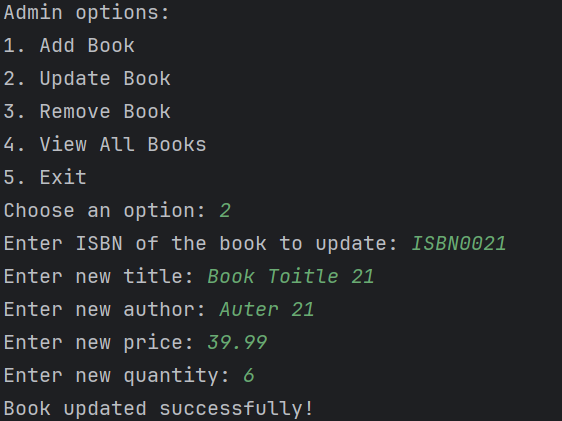
This can be verified in the database :





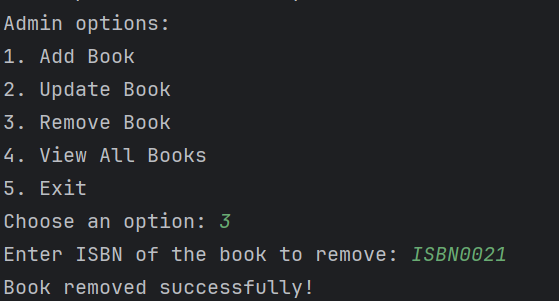
**Admin Updating a Book**

If the admin chooses to update a book:



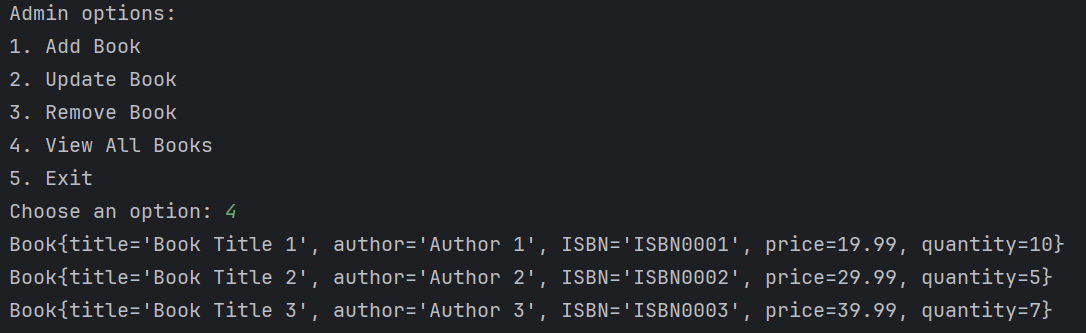
**Admin Removing a Book**

If the admin chooses to remove a book:



**Admin Viewing all Books**

If the admin chooses to view all books:



# Sprint 3

## Objective:

Elevate the "BookstoreCLI" application by incorporating advanced features and addressing sophisticated areas. In this Sprint, we will concentrate on:

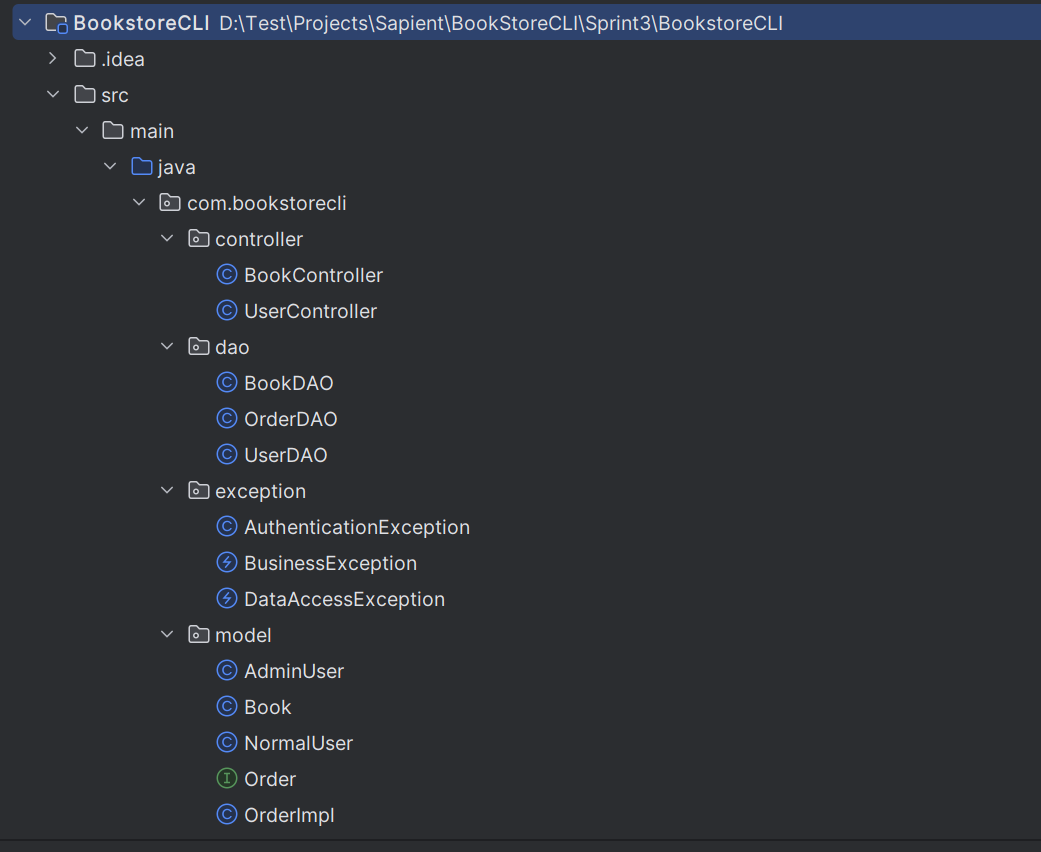
1. Refactor and enhance the "BookstoreCLI" application with a focus on security, new features, and advanced development practices.
2. Ensure Personally Identifiable Information (PII) is stored in the PostgreSQL database with encryption.
3. Implement role-based authentication and authorization to enforce secure access control within the application.
4. Introduce an option for users to change their passwords.
5. Create a Book Order feature for Normal Users, allowing them to place and manage their book orders.
6. Utilize Behaviour-Driven Development (BDD) with Cucumber for Java to guarantee the application aligns with business requirements and user expectations through comprehensive scenarios and acceptance criteria.
7. Enable admins to import data into the PostgreSQL database from a CSV file.

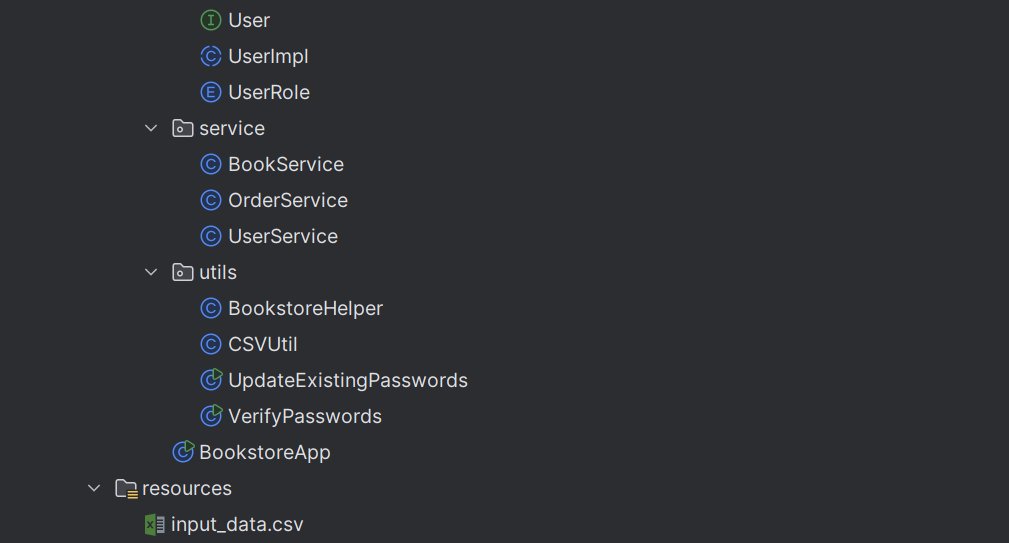
## Solution Outline

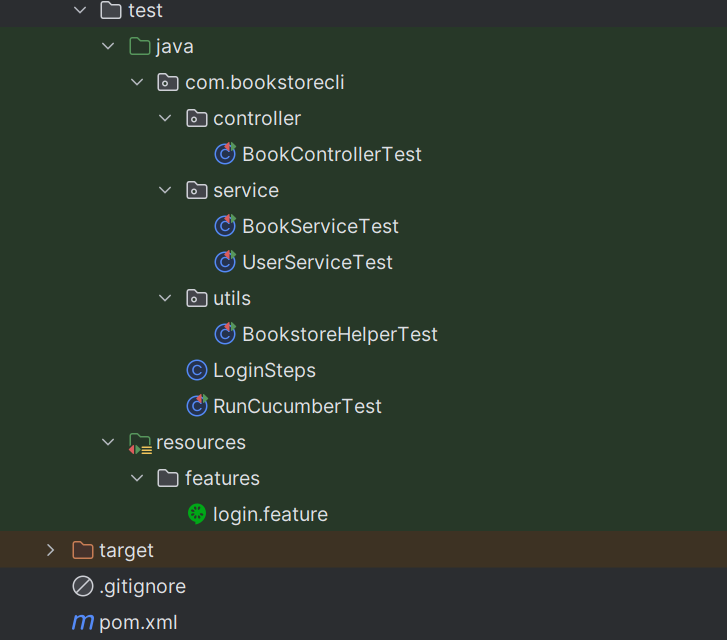
**New Classes:**

1. **SecurityManager:**
   * Handles encryption and decryption of PII.
   * Manages user roles and permissions.
2. **AuthenticationService:**
   * Handles user authentication and session management.
   * Validates user credentials and assigns roles.
3. **AuthorizationService:**
   * Checks user permissions for accessing different features and resources.
4. **OrderService:**
   * Manages the creation, retrieval, and updating of book orders.
   * Interacts with the database to store and retrieve order information.
5. **OrderController:**
   * Handles user requests related to book orders and interacts with the OrderService.
   * Ensures the correct data is passed between the user interface and the service layer.
6. **CucumberTestRunner:**
   * Configured for running BDD scenarios using Cucumber and Serenity.
   * Ensures that tests are executed and results are captured in detailed reports.
7. **CSVUtil:**
   * Utilizes Java NIO's FileChannel, Buffer, and Path classes to read from and write to CSV files.
   * Facilitates loading data into the PostgreSQL database from CSV files.
   * Handles file operations efficiently and ensures data integrity during the process.

# Project Structure

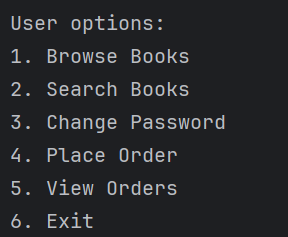






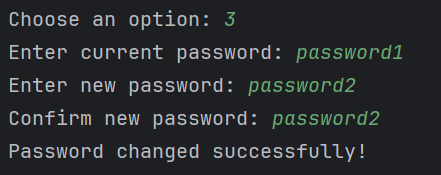
# CLI Application Screens

**User Menu (Normal User)**

****

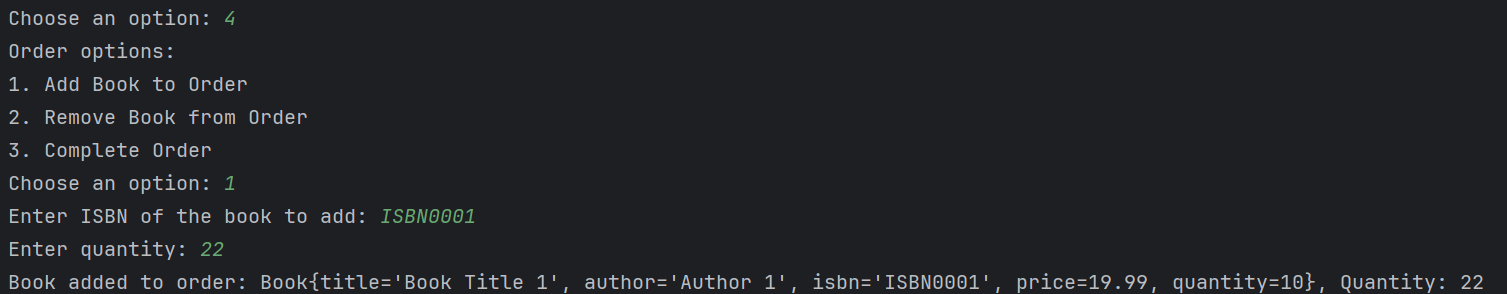
**User Change Password Option**

If the user chooses to change his password:



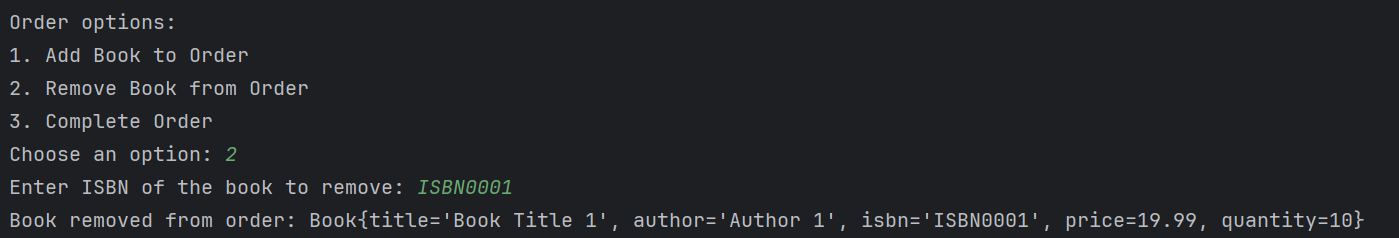
**User Placing an Order**

If the user chooses to place an Order for a book:



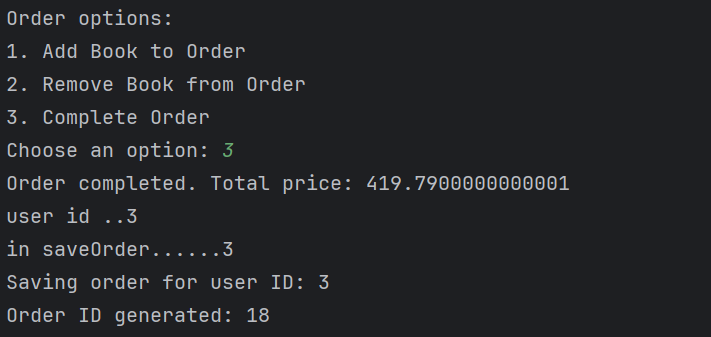
**User Removing a book from an Order**

If the user chooses to remove an Order for a book:



**User Completing a book from an Order**

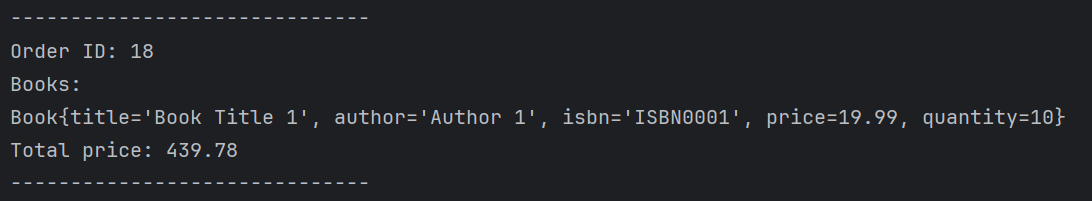
If the user chooses to complete an Order for a book:



**User Viewing all Orders**

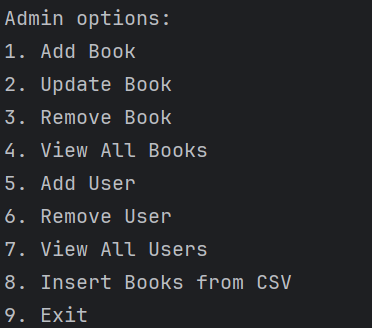
If the user chooses to view all orders placed:





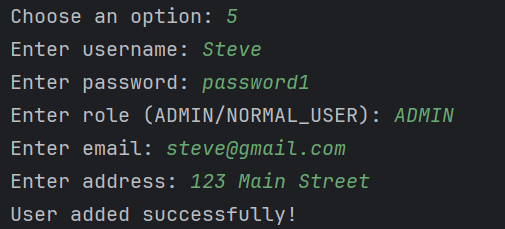
**Admin Menu**

If an admin user logs in, they will see:



**Admin Adding a User**

If the admin chooses to add a user:



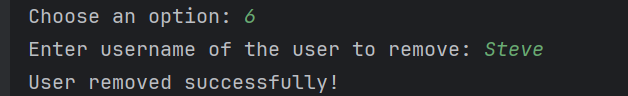
This can be verified in the database:





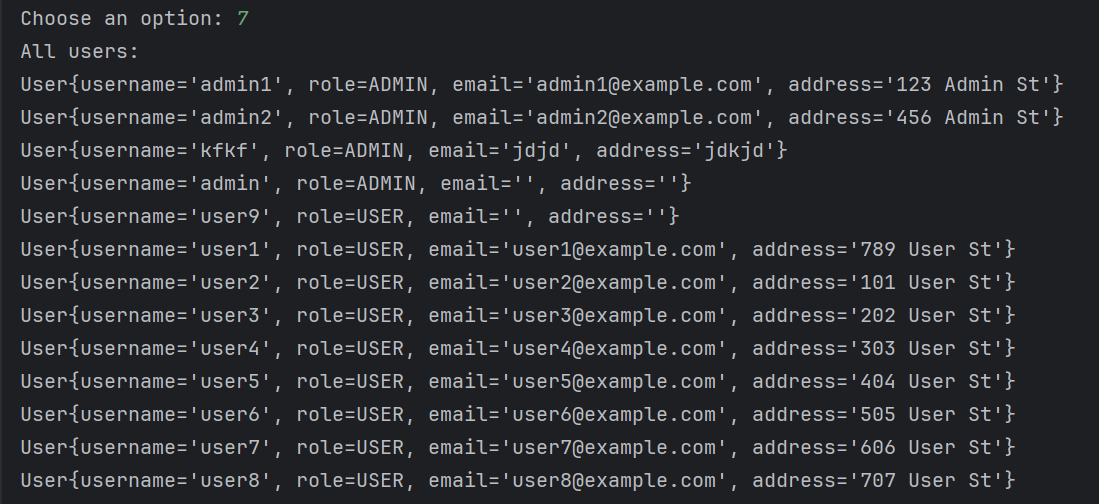
**Admin Removing a User**

If the admin chooses to remove a user:

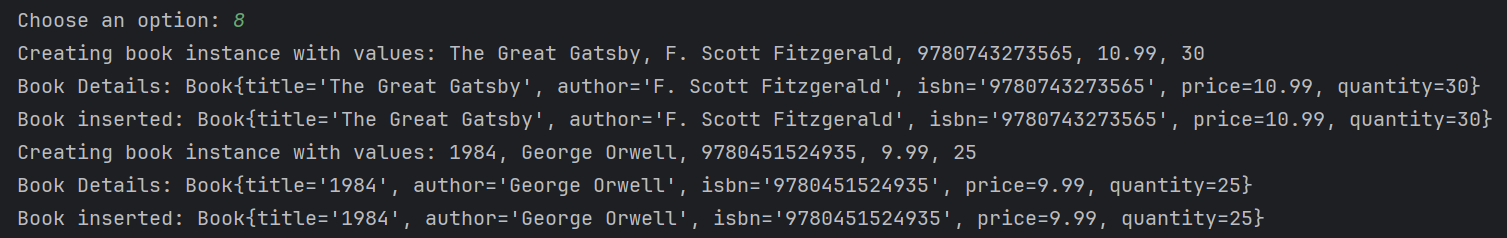


**Admin Viewing all Users**

If the admin chooses to view all users:



**Admin Loading data into PostgreSQL database**



# Sprint 4

## Objective:

Reengineer the "BookstoreCLI" application by integrating advanced features and tackling complex areas. This Sprint will focus on:

1. Redesigning the application using Spring Boot.
2. Developing REST API endpoints with Spring REST.
3. Integrating Spring Data JPA for data persistence.
4. Applying the Spring Testing framework for unit, system, and integration testing.
5. Documenting the REST API endpoints with Swagger API.
6. Containerizing the application and publishing the image to a container repository.
7. Conducting performance and load testing using JMeter.

## Solution Outline

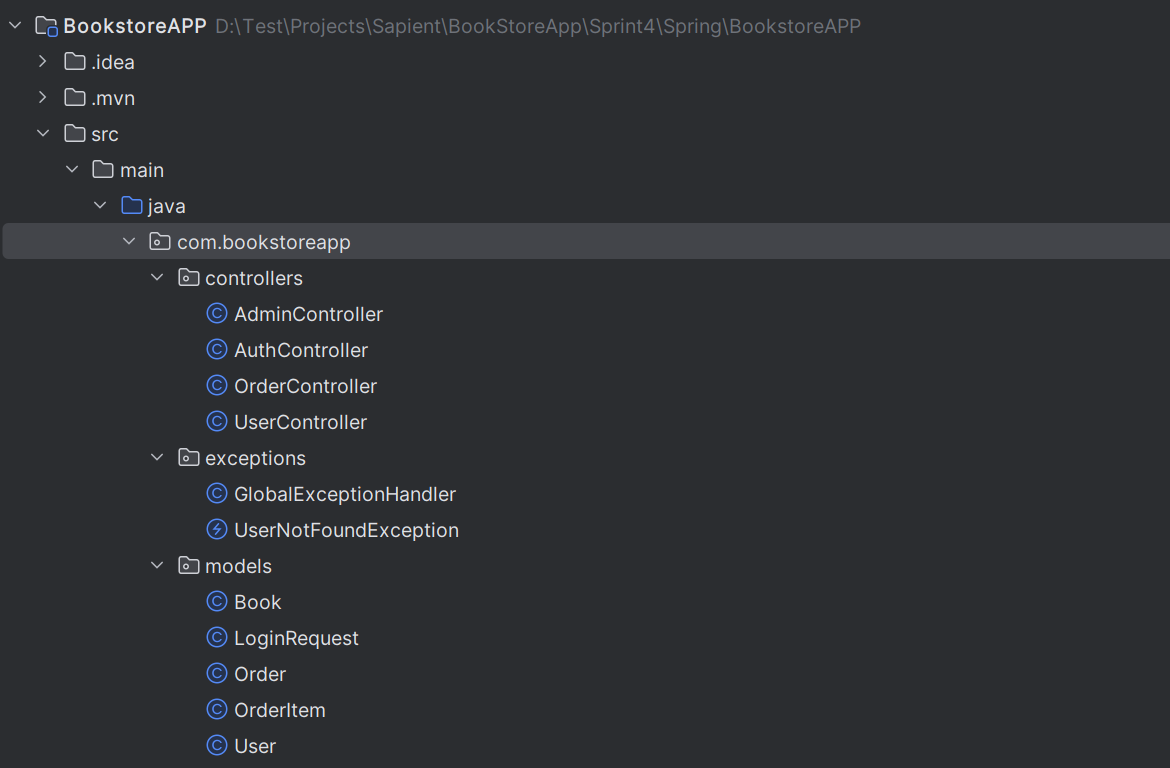
**New Classes:**

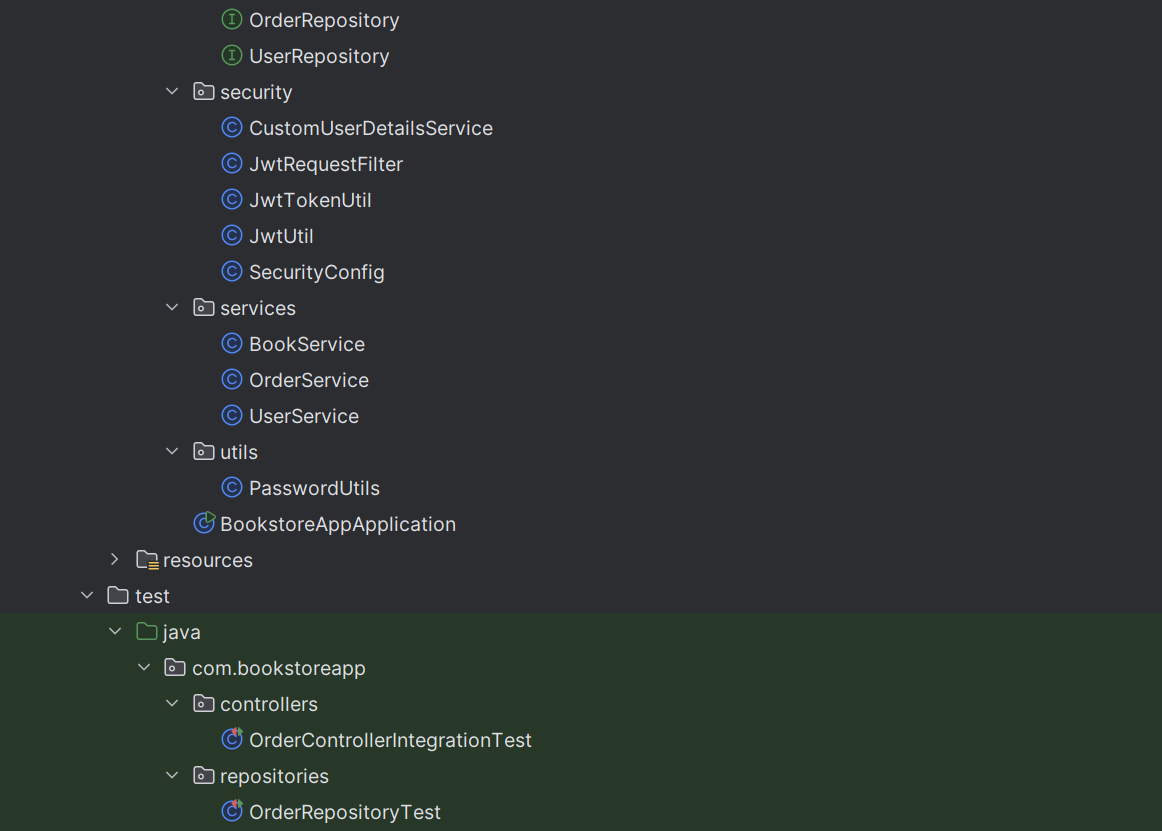
1. **BookstoreApplication:**
   * Entry point for the Spring Boot application.
   * Configures and runs the Spring Boot application.
2. **BookController:**
   * REST controller that handles HTTP requests for book-related operations.
   * Maps endpoints for creating, retrieving, updating, and deleting books.
3. **UserController:**
   * REST controller for handling user-related HTTP requests.
   * Manages endpoints for user registration, authentication, and profile management.
4. **OrderController:**
   * REST controller for handling book order-related HTTP requests.
   * Manages endpoints for placing, viewing, and managing book orders.
5. **BookService:**
   * Service layer class that contains business logic for book-related operations.
   * Interacts with the BookRepository to perform CRUD operations on books.
6. **UserService:**
   * Service layer class that manages business logic for user-related operations.
   * Handles user authentication, registration, role assignment, and profile updates.
7. **OrderService:**
   * Service layer class responsible for managing book orders.
   * Interacts with the OrderRepository to handle CRUD operations on book orders.
8. **BookRepository:**
   * Spring Data JPA repository interface for performing database operations on books.
   * Extends JpaRepository to provide methods for CRUD operations.
9. **UserRepository:**
   * Spring Data JPA repository interface for user-related database operations.
   * Extends JpaRepository to provide CRUD methods for user entities.
10. **OrderRepository:**
    * Spring Data JPA repository interface for managing book orders in the database.
    * Extends JpaRepository to provide CRUD operations for order entities.
11. **SecurityConfig:**
    * Configures Spring Security for role-based authentication and authorization.
    * Defines security policies, user roles, and access control mechanisms.
12. **SwaggerConfig:**
    * Configures Swagger for API documentation.
    * Sets up Swagger UI for interactive REST API documentation.
13. **Dockerfile:**
    * Defines the Docker image for the application.
    * Specifies the base image, copies the application files, and sets the entry point.
14. **JMeterTestPlan:**
    * JMeter test plan for performance and load testing.
    * Configures test scripts, thread groups, and assertions to evaluate application performance.
15. **Test Classes (e.g., BookControllerTest, UserServiceTest):**
    * JUnit test classes for unit, system, and integration tests.
    * Utilizes Spring Testing framework to ensure robust testing coverage.

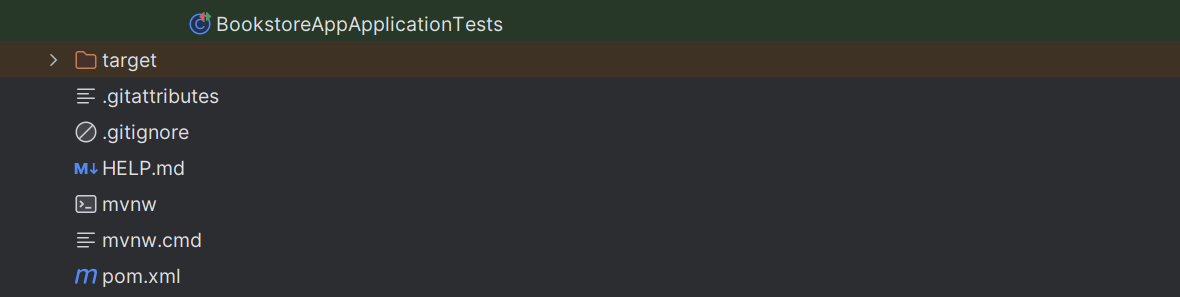
**Additional Features:**

* **Encryption and Security:**
  + Ensure PII is encrypted in the PostgreSQL database.
  + Implement role-based access control to secure endpoints.
* **User Features:**
  + Allow users to change their passwords.
* **Admin Features:**
  + Enable admin users to import data from CSV files into the PostgreSQL database.

# Project Structure







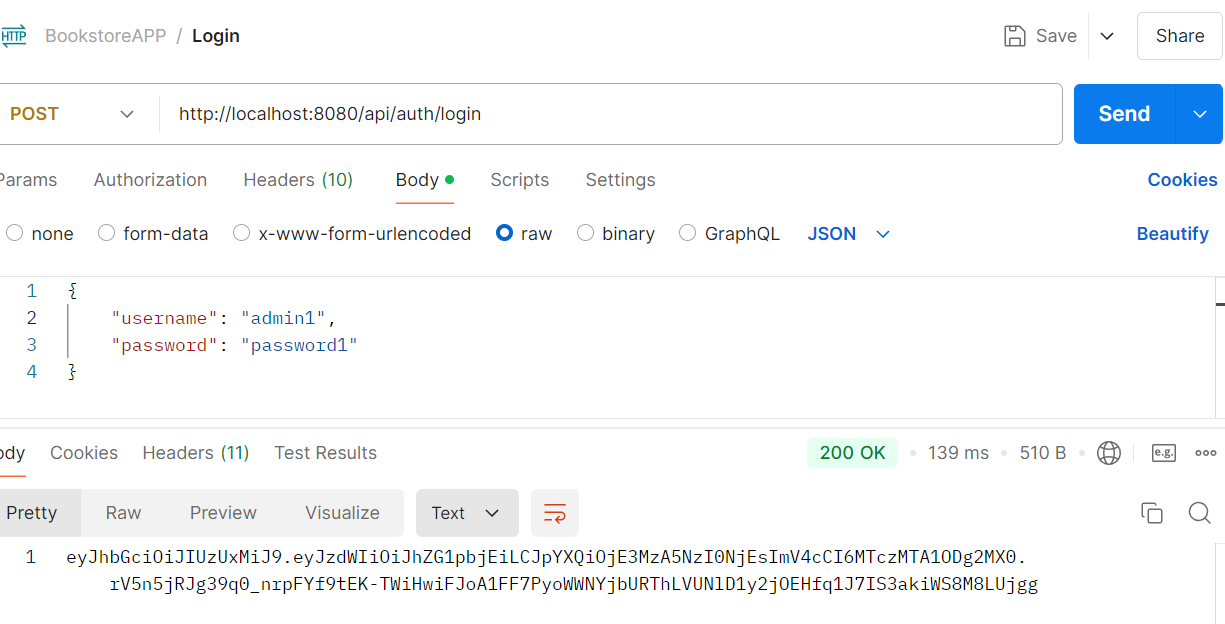
# API Endpoint for Authentication and Authorization

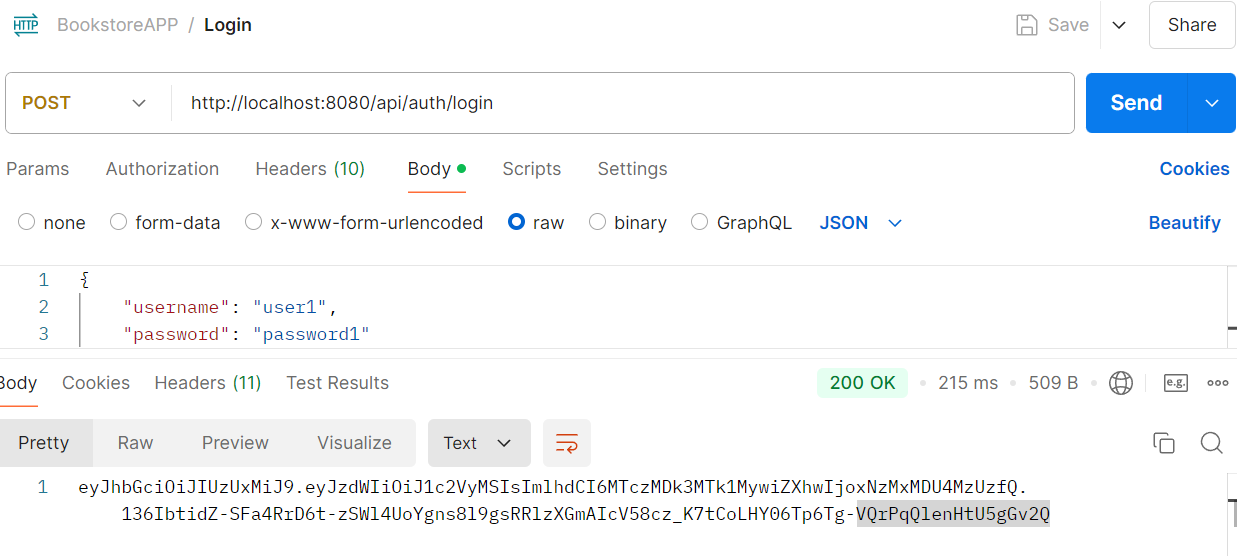
**Login Admin/User**

* **URL:** /api/users/login
* **Method:** POST
* **Description:** Authenticate a user and initiate a session. Upon successful authentication, a JSON Web Token (JWT) is generated. This token must be included in the Authorization header of every subsequent request to authenticated endpoints.

**How Token Generation Works:**

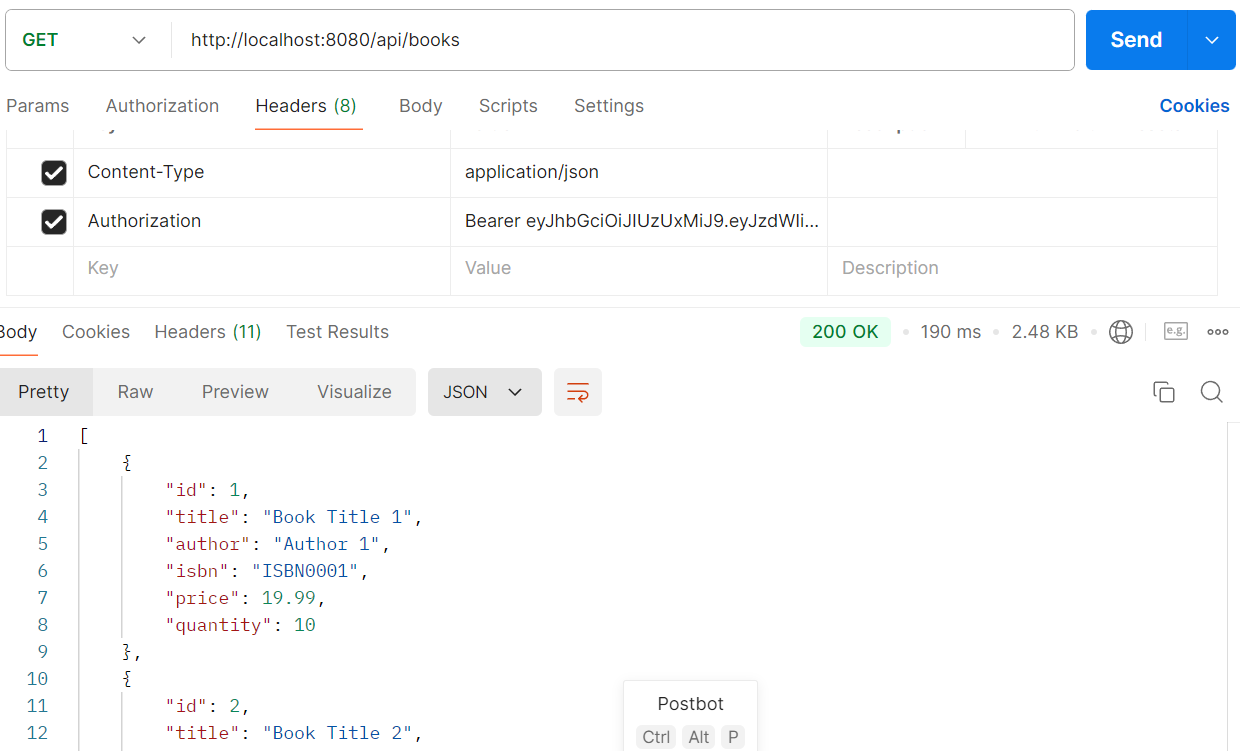
1. **User Authentication:** The provided username and password are validated against the stored credentials in the database.
2. **Token Creation:** If the credentials are valid, a JWT is generated. This token typically contains the user's ID, roles, and expiration time, signed with a secret key.
3. **Token Response:** The generated JWT is sent back to the client as part of the response.



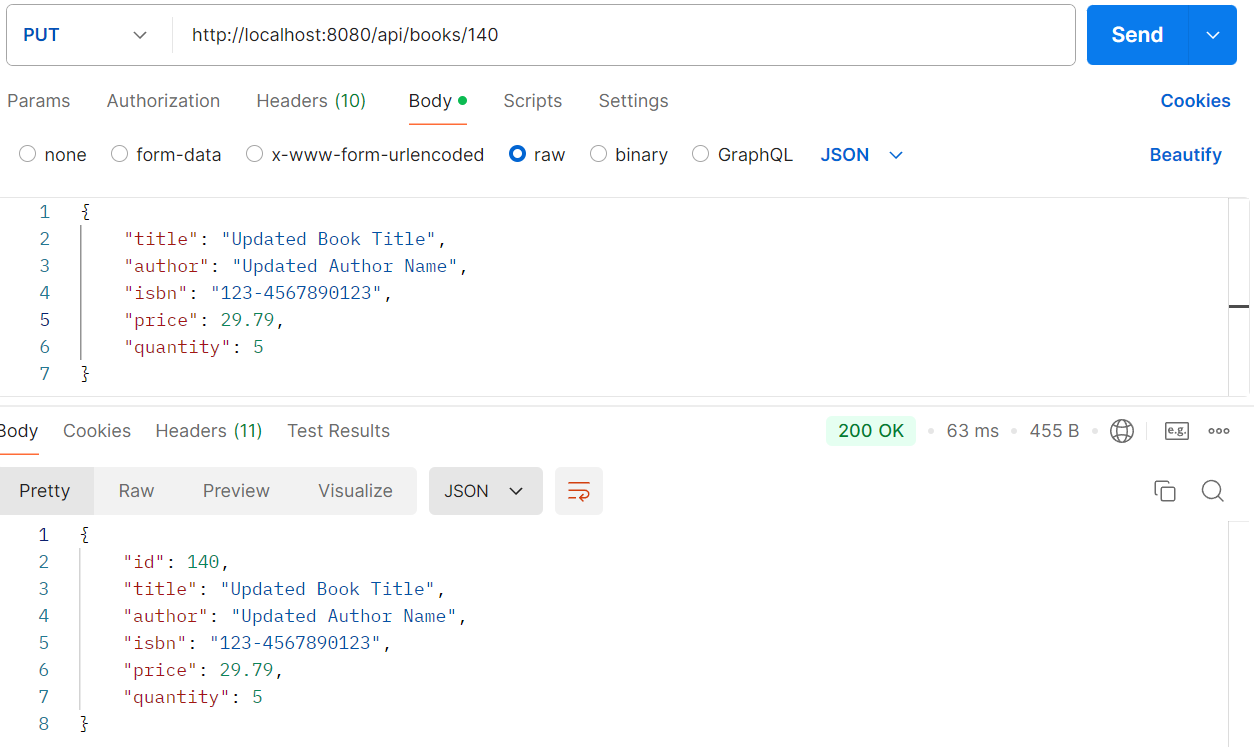


# API Endpoints for Admin Options

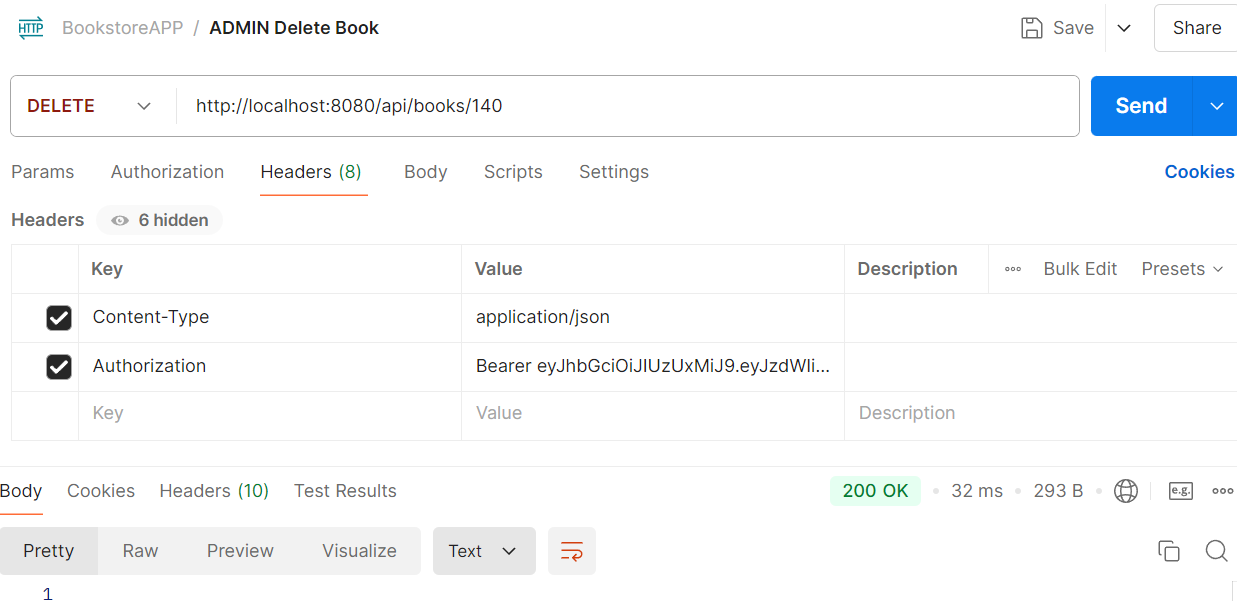
1. **Add Book:**
   * **URL**: /api/books
   * **Method**: POST
   * **Description**: Add a new book.



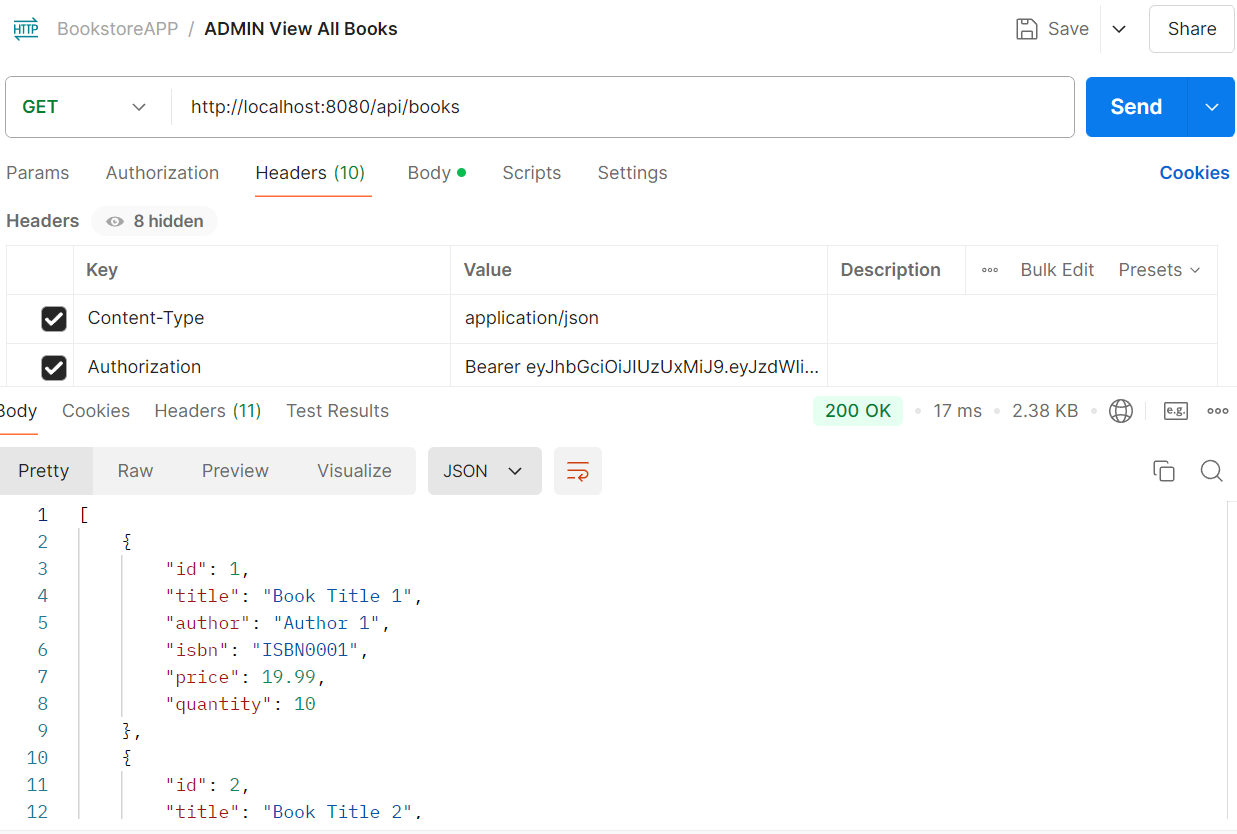
1. **Update Book:**
   * **URL**: /api/books/{bookId}
   * **Method**: PUT
   * **Description**: Update an existing book.



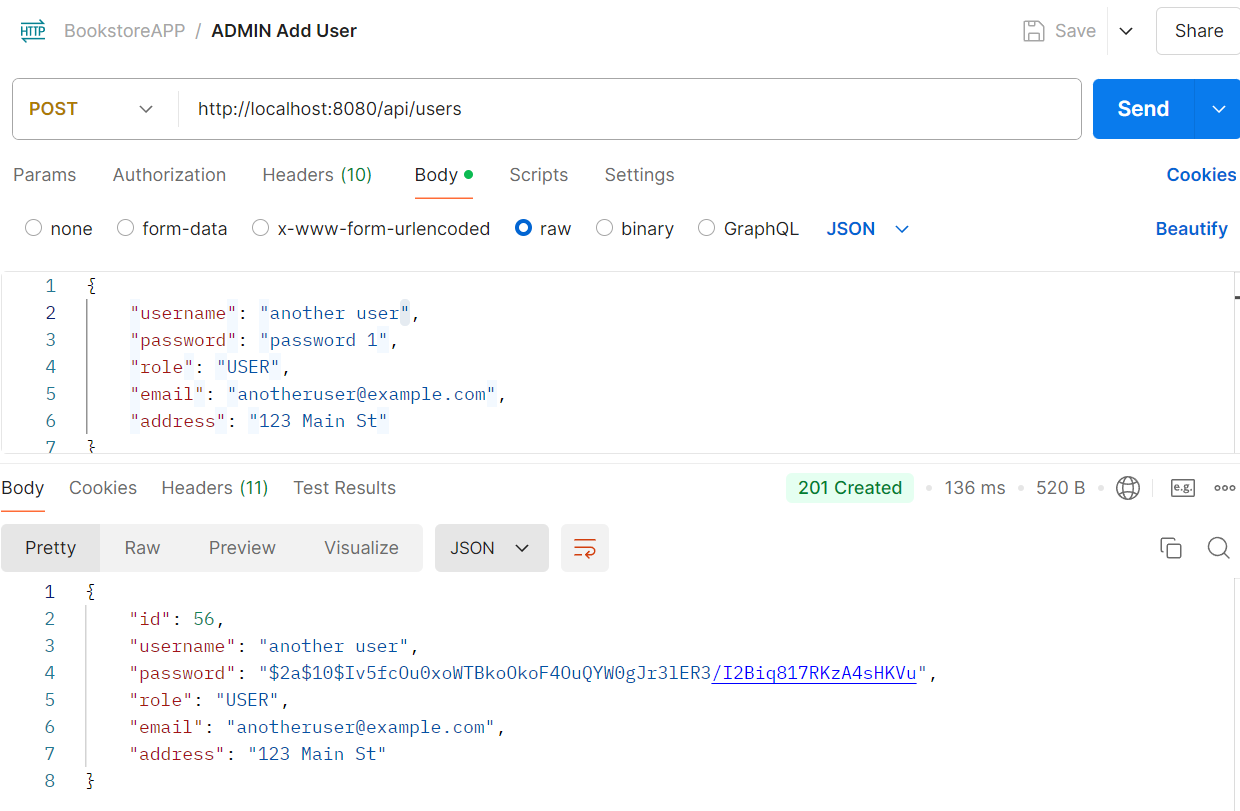
1. **Remove Book:**
   * **URL**: /api/books/{bookId}
   * **Method**: DELETE
   * **Description**: Remove a book.

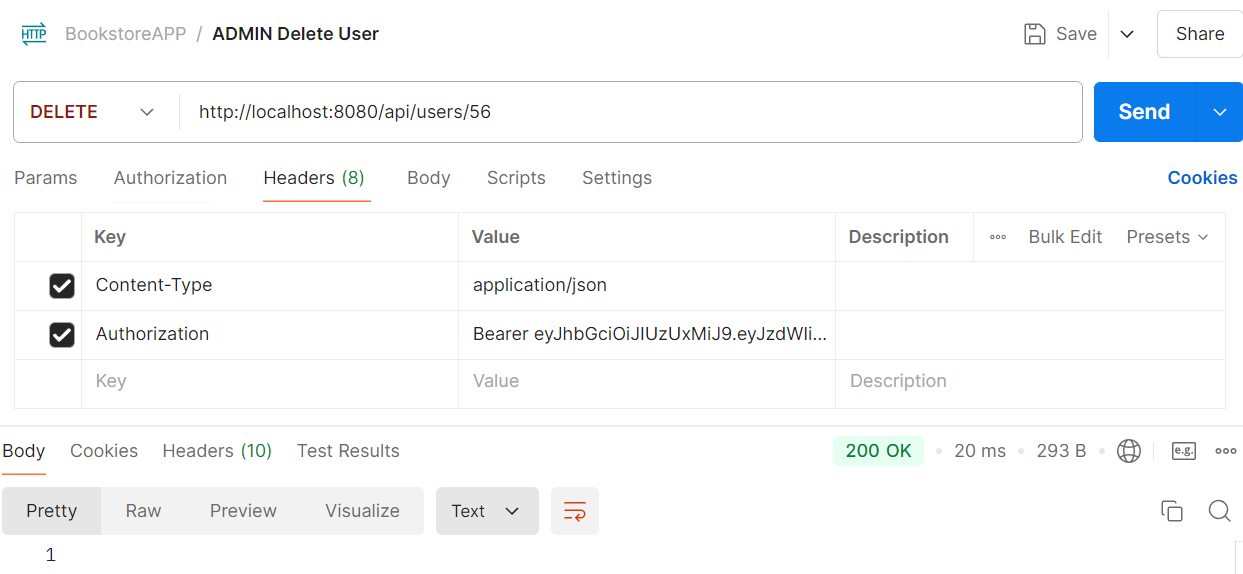


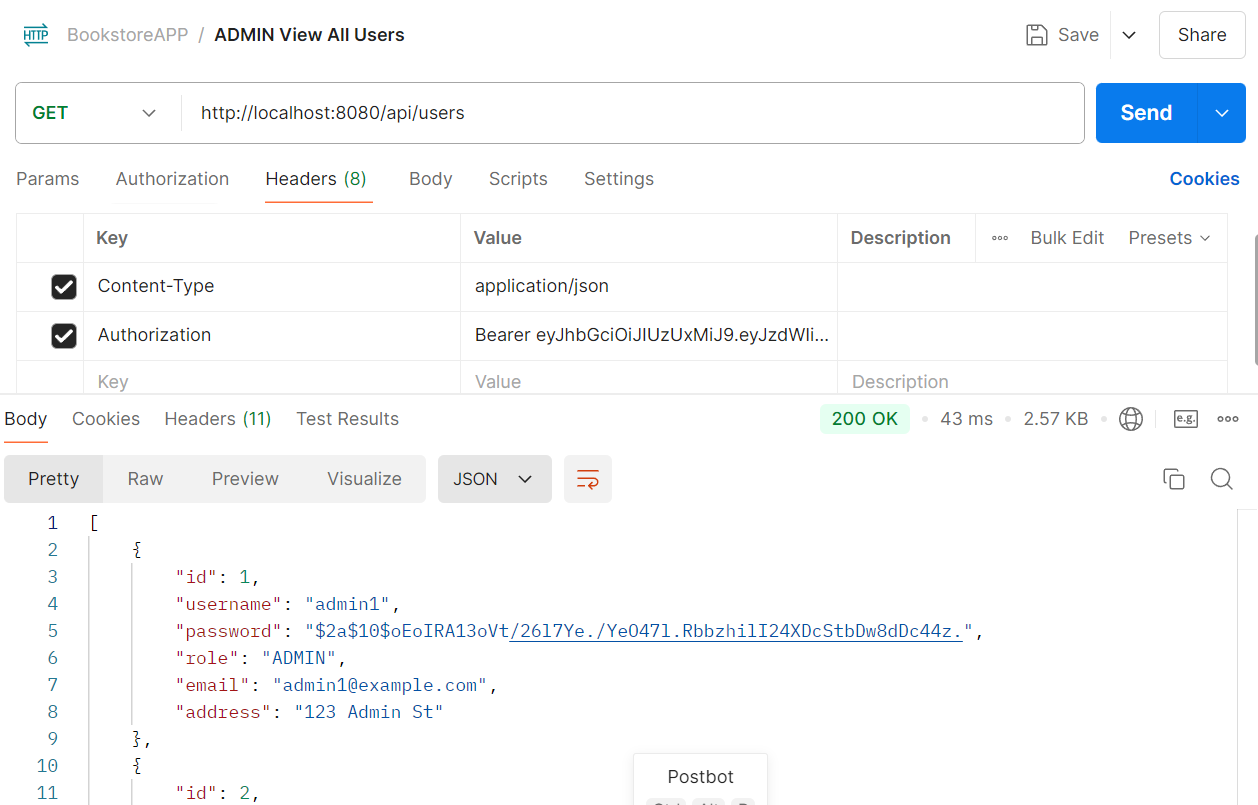
1. **View All Books:**
   * **URL**: /api/books
   * **Method**: GET
   * **Description**: Get a list of all books.



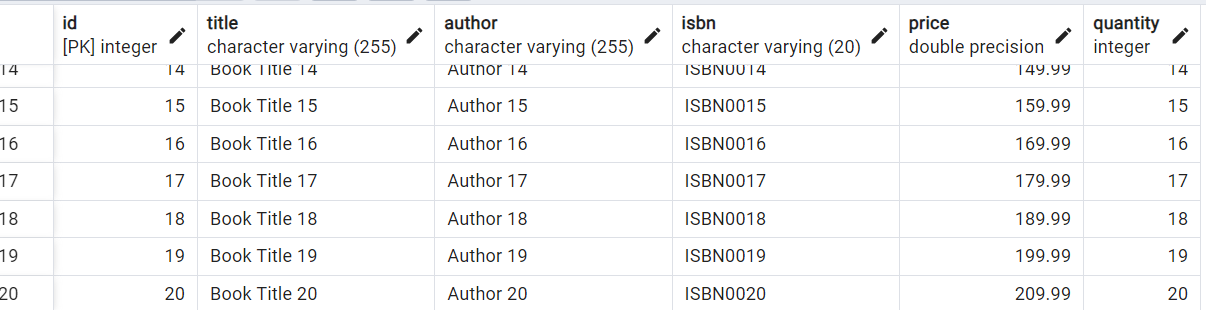
1. **Add User:**
   * **URL**: /api/users
   * **Method**: POST
   * **Description**: Add a new user.

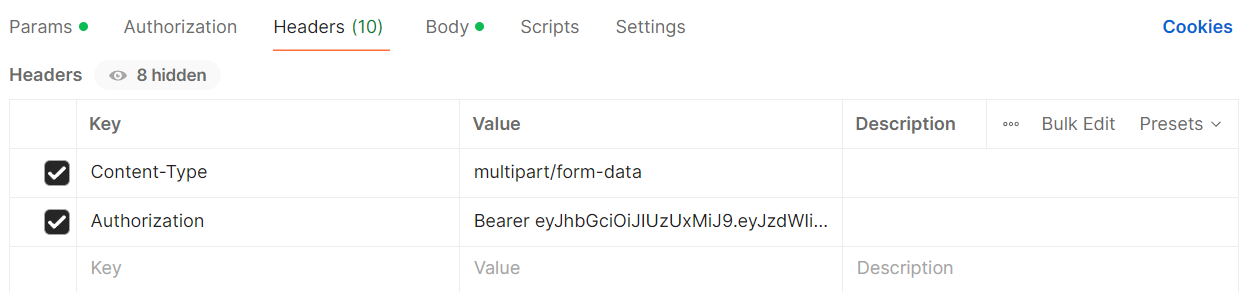
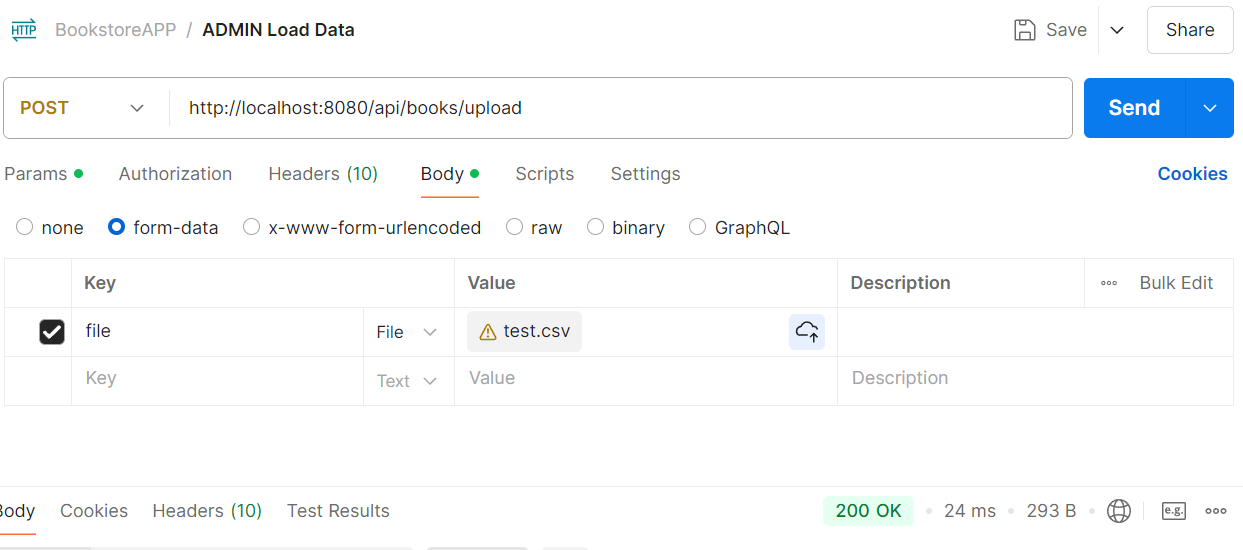


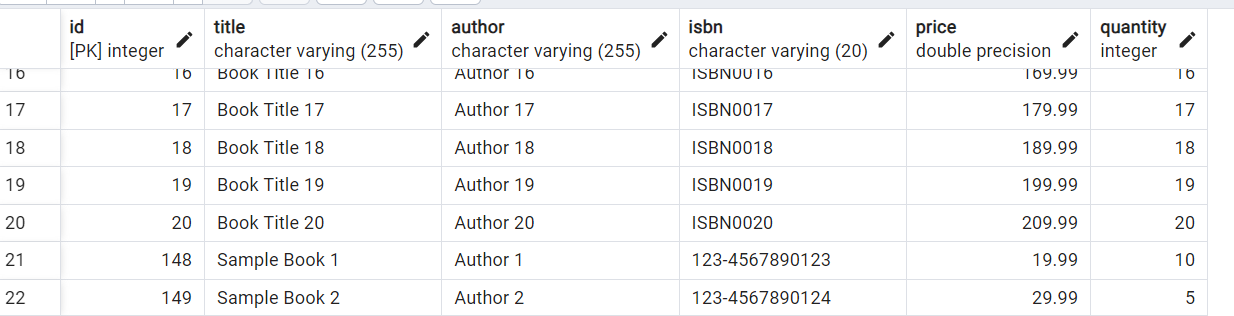
1. **Remove User:**
   * **URL**: /api/users/{userId}
   * **Method**: DELETE
   * **Description**: Remove a user.****
2. **View All Users:**
   * **URL**: /api/users
   * **Method**: GET
   * **Description**: Get a list of all users.



1. **Insert Books from CSV:**
   * **URL**: /api/books/upload
   * **Method**: POST
   * **Description**: Upload and insert books from a CSV file into the database.

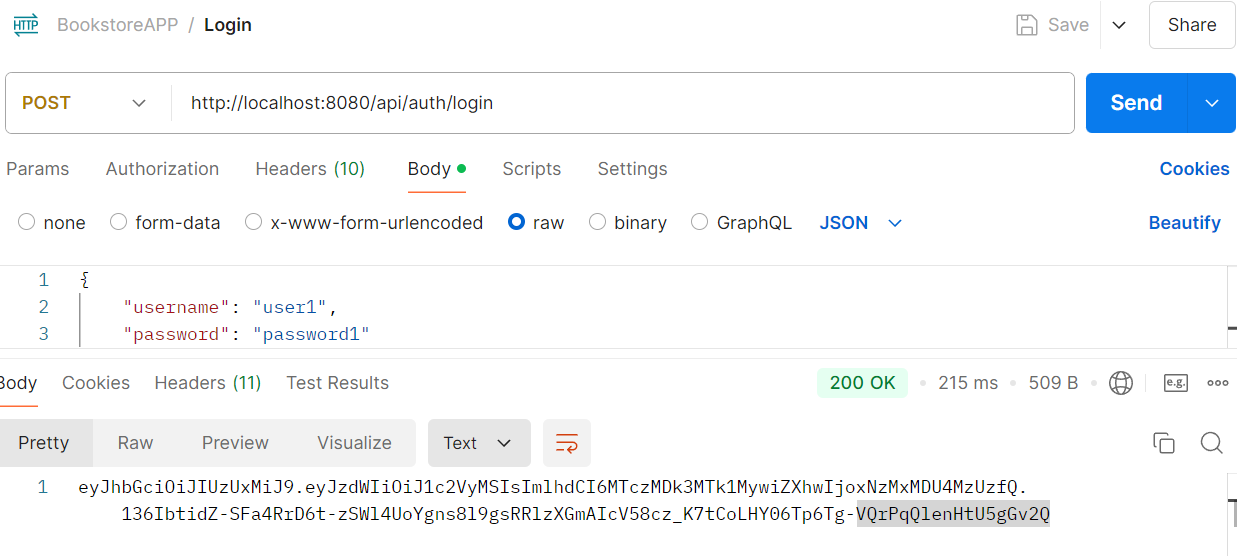


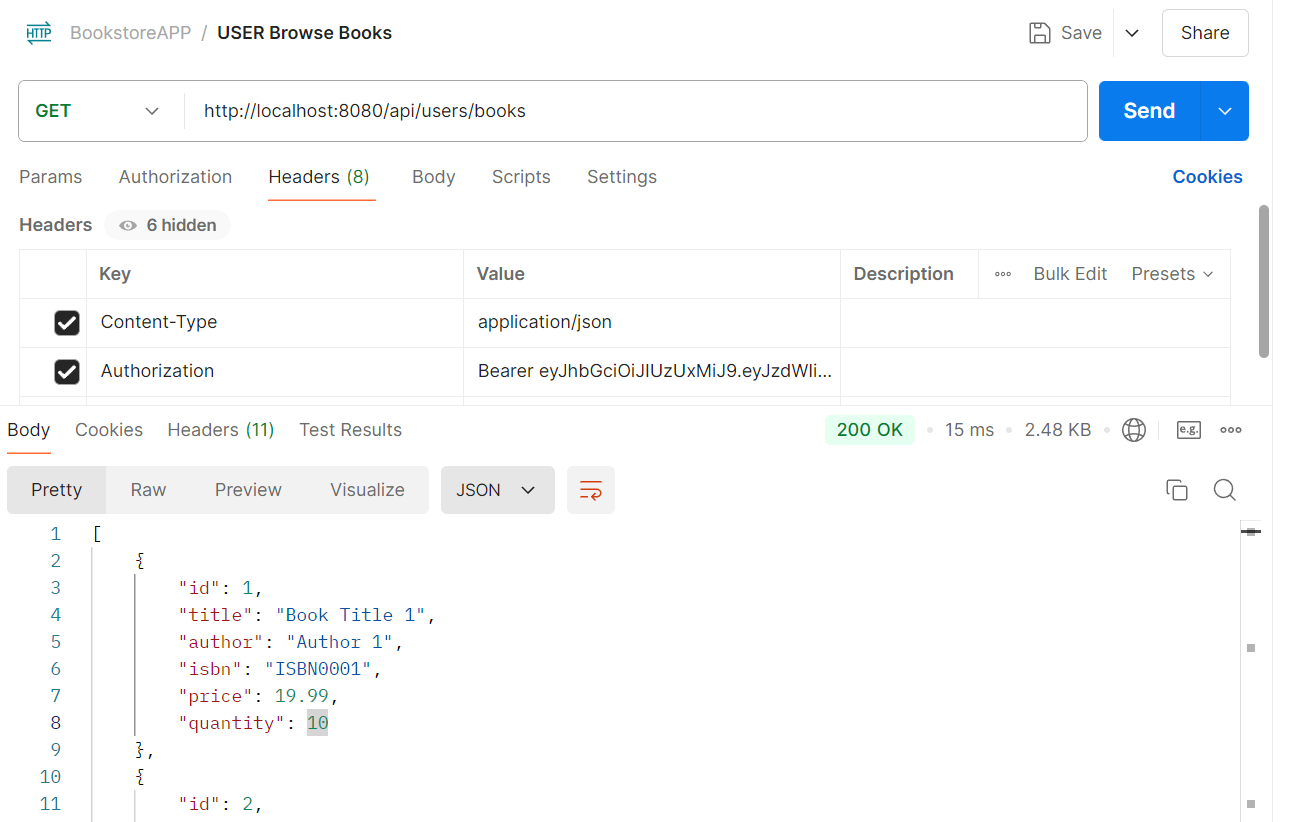


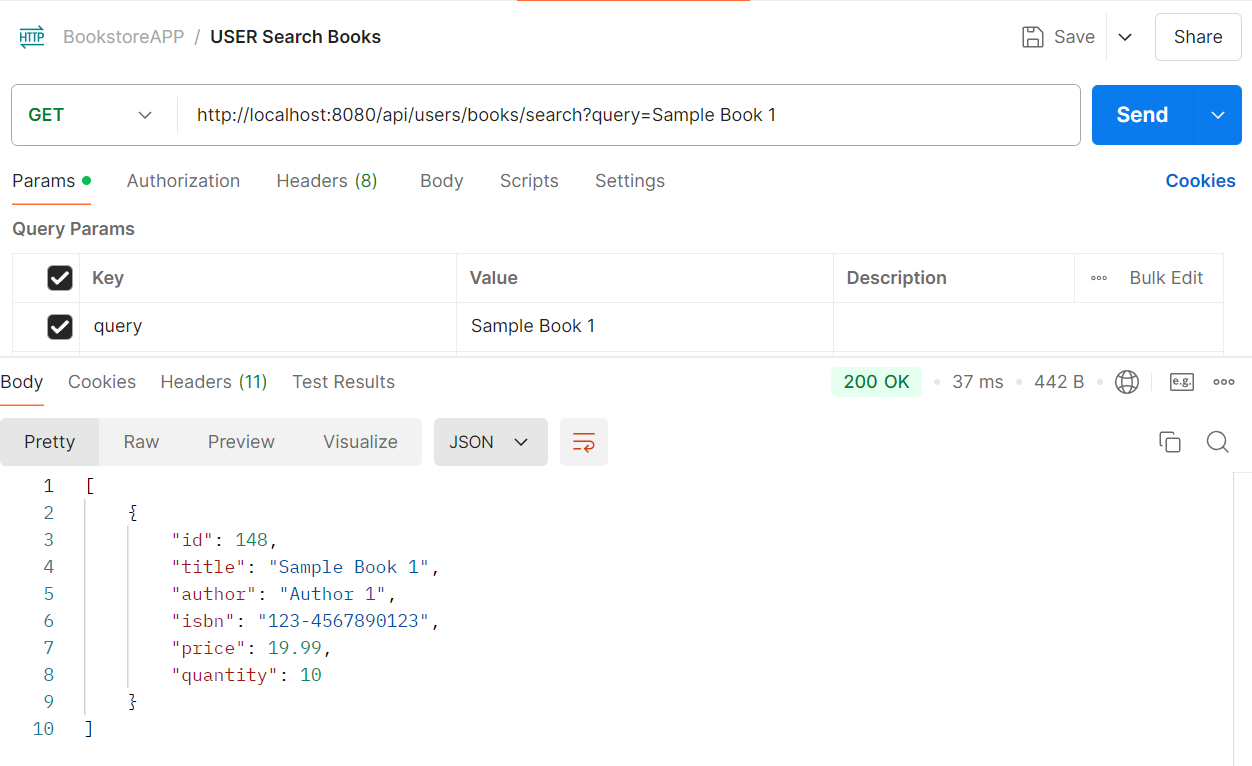
# API Endpoints for User Options

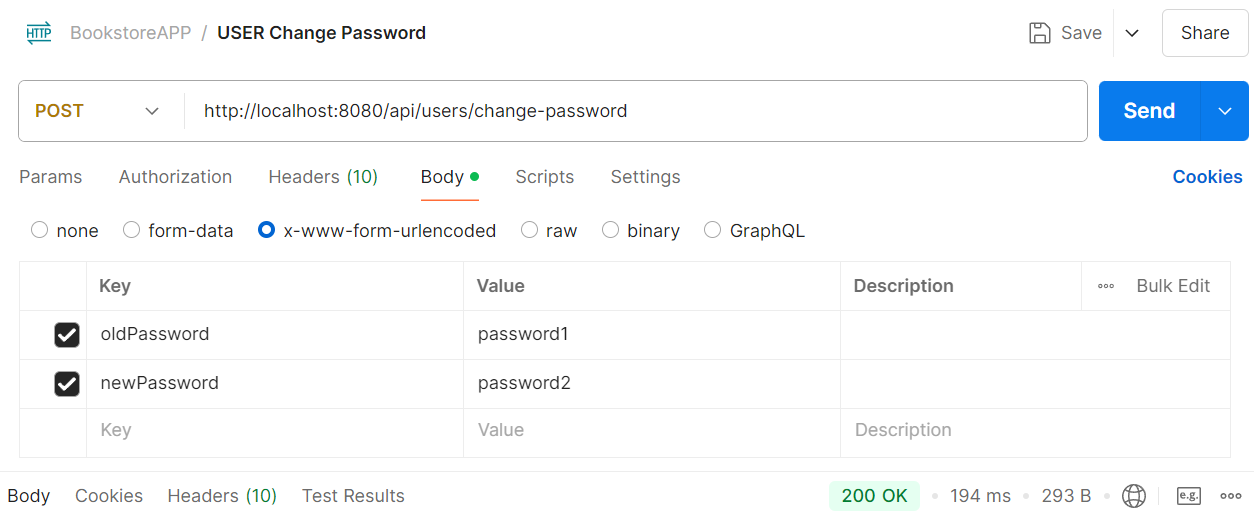
1. **Login User:**
   * **URL**: /api/users/login
   * **Method**: POST
   * **Description**: Authenticate a user and initiate a session.



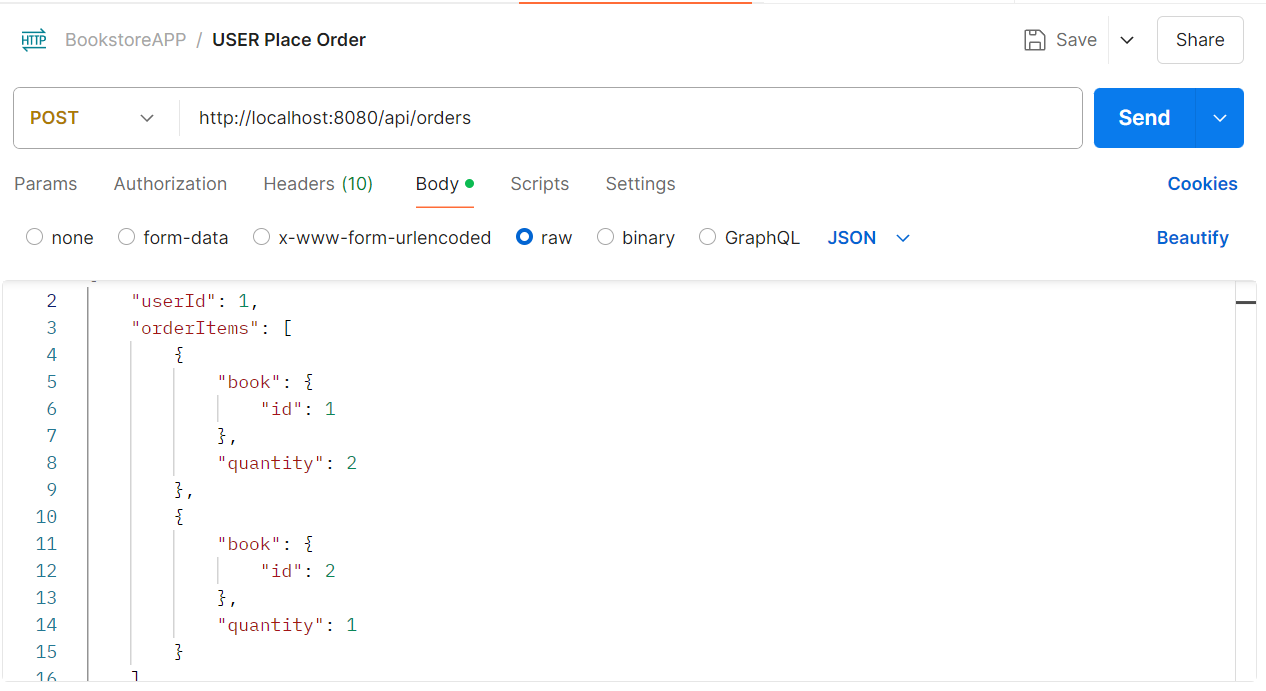
1. **Browse Books:**
   * **URL**: /api/users/books
   * **Method**: Get
   * **Description**: List the books in the inventory.



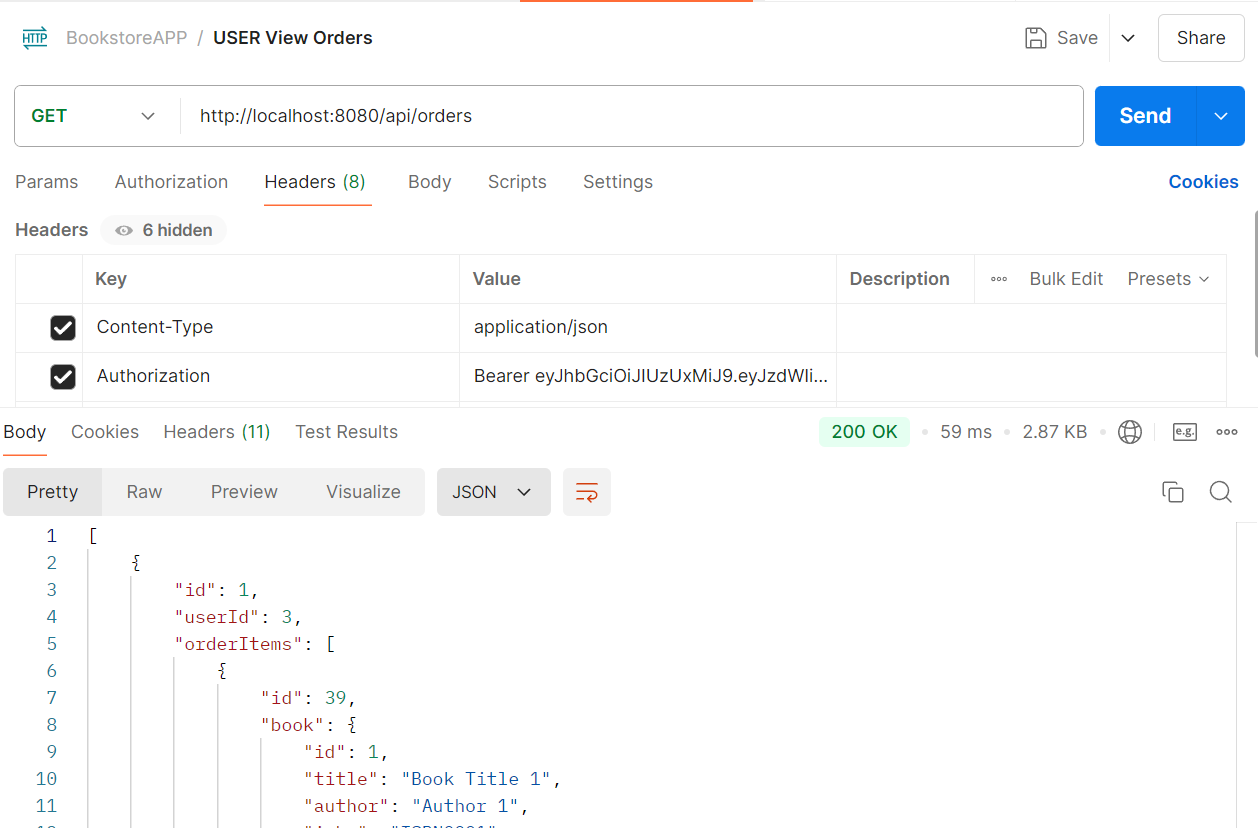
1. **Search Books:**
   * **URL**: /api/users/books
   * **Method**: Get
   * **Description**: This endpoint allows users to search for books based on a query string. The query string is passed as a request parameter, and the endpoint returns a list of books that match the query.
   * **Parameters:**
   * **query** (String): The query string used to search for books. This parameter is required.
   * 
2. **Change Password:**
   * **URL**: /api/users/{userId}/password
   * **Method**: PUT
   * **Description**: Change the user's password.



1. **Place Order:**
   * **URL**: /api/orders
   * **Method**: POST
   * **Description**: Place a new book order.

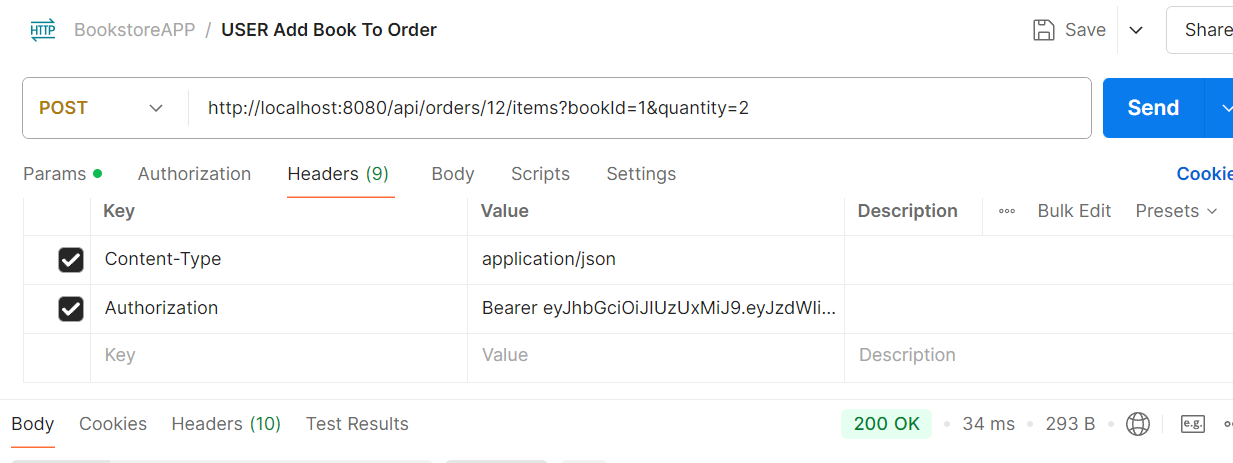
 

1. **View User Orders:**
   * **URL**: /api/orders
   * **Method**: GET
   * **Description**: Get a list of all orders placed by the user.

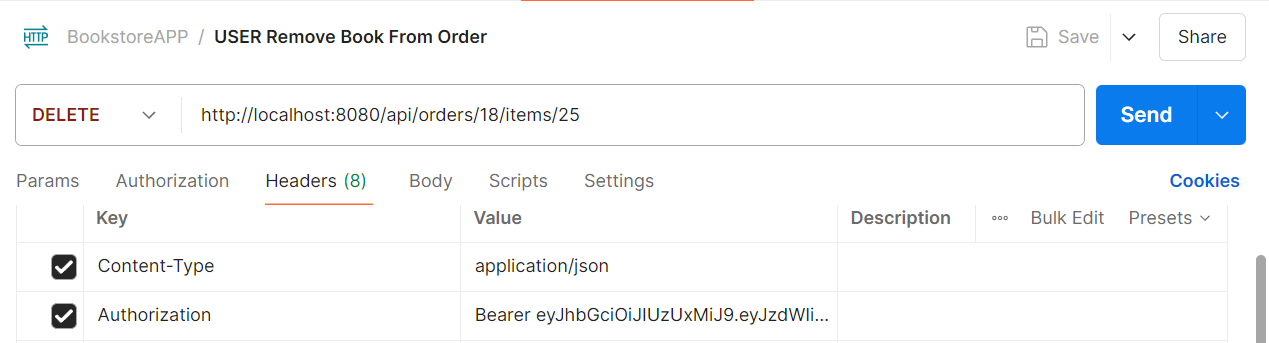


1. **Add Book To Order:**
   * **URL**: /api/orders/{orderId}/items
   * **Method**: POST
   * **Description**: This endpoint allows adding a book to an existing order. It requires the order ID, the book ID, and the quantity of the book to add. Upon successful addition, it returns a 200 OK status with no content.
   * **Parameters:**

* **orderId** (Long): The ID of the order to which the book will be added. This parameter is required and is passed as a path variable.
* **bookId** (Integer): The ID of the book to add to the order. This parameter is required and is passed as a request parameter.
* **quantity** (Integer): The quantity of the book to add to the order. This parameter is required and is passed as a request parameter.

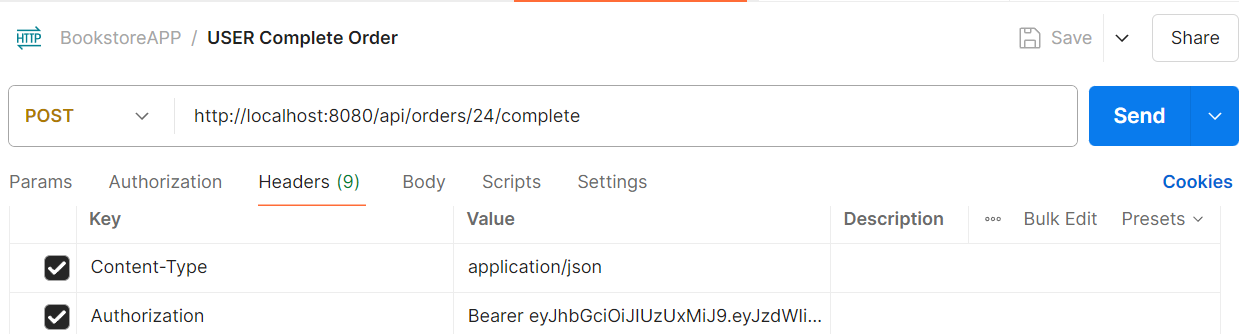


1. **Cancel Order:**
   * **URL**: /api/orders/{orderId}
   * **Method**: DELETE
   * **Description**: Cancel an existing order.



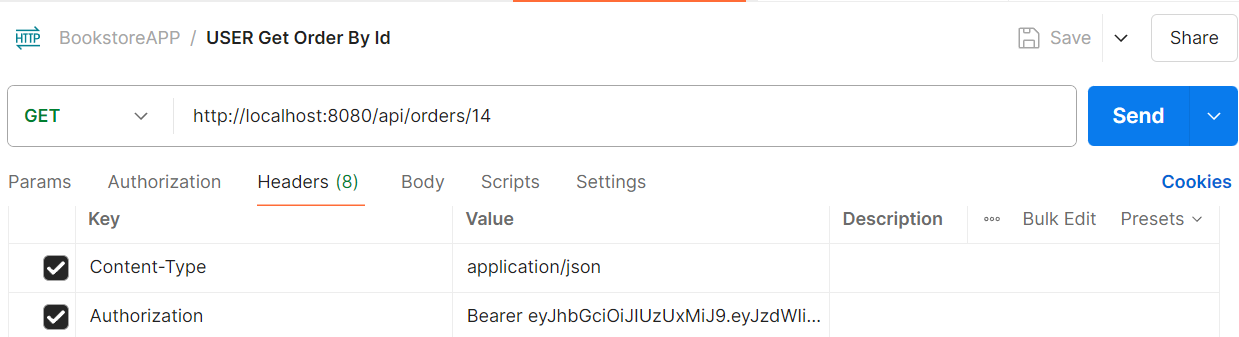
1. **Complete Order:**

* **URL**: /api/orders/{orderId}/complete
* **Method**: POST
* **Description**: Mark an existing order as completed.



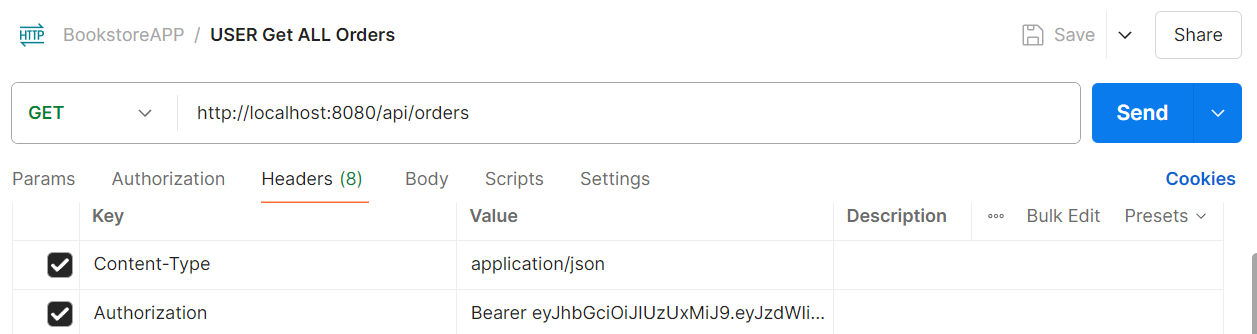
1. **Get Order by ID:**

* **URL**: /api/orders/{orderId}
* **Method**: GET
* **Description**:Retrieve the details of an order by its ID.



1. **Get All Orders for the Logged-in User:**

* **URL**: /api/orders
* **Method**: GET
* **Description**: Retrieve a list of all orders placed by the currently authenticated user.



# Sprint 5

## Objective:

Create the web user interface for the "BookstoreCLI" application leveraging cutting-edge front-end technologies and industry best practices. This Sprint will focus on:

1. Developing the Web UI with the latest versions of [React.js](https://React.js) and TypeScript.
2. Utilizing [Next.js](https://Next.js) for enhanced tooling and server-side rendering.
3. Implementing ESLint for linting and ensuring code quality.
4. Integrating and consuming the REST APIs created in the previous Sprint.
5. Updating the CI/CD pipeline to incorporate build, test, and deployment steps for the front-end application.
6. Performing extensive testing using Jest and React Testing Library (RTL), including snapshot tests to ensure high test coverage and component verification.

## Solution Outline

**New Components and Tools:**

1. **React Components:**
   * **App**: The root component that sets up the application structure.
   * **Header**: Displays the navigation bar and links to different parts of the application.
   * **BookList**: Shows a list of books fetched from the API.
   * **BookDetail**: Displays detailed information about a specific book.
   * **OrderHistory**: Displays the logged-in user's order history.
   * **Cart**: Manages the shopping cart functionality.
   * **Login**: Handles user login functionality.
   * **PlaceOrder**: Manages the process of placing a new order.
   * **CompleteOrder**: Handles the completion of an existing order.
   * **AddBookToOrder**: Allows users to add books to an existing order.
   * **LoadBooks**: Admin component to upload books from a CSV file into the database.
   * **ManageUsers**: Admin component to manage user accounts, including adding and removing users.
   * **ChangePassword**: Allows users to change their passwords.
2. [**Next.js**](https://Next.js) **Setup:**
   * **Custom \_app.js**: To initialize pages with common layouts and configurations.
   * **Custom \_document.js**: To customize the HTML document structure.
   * **Pages Directory**: Each file in this directory will automatically become a route.
     + [**index.ts**](https://index.js): The home page of the application.
     + [**books.ts**](https://books.js): The books listing page.
     + **book/[id].ts**: Dynamic routing for individual book details.
     + [**login.ts**](https://login.js): User login page.
     + [**orders.ts**](https://orders.js): User's order history page.
     + [**cart.ts**](https://cart.js): Shopping cart page.
     + [**place-order.ts**](https://place-order.js): Page for placing a new order.
     + [**complete-order.ts**](https://complete-order.js): Page for completing an order.
     + [**add-book-to-order.ts**](https://add-book-to-order.js): Page for adding books to an order.
     + [**load-books.ts**](https://load-books.js): Admin page for uploading books from CSV.
     + [**manage-users.ts**](https://manage-users.js): Admin page for managing user accounts.
     + [**change-password.ts**](https://change-password.js): Page for users to change their passwords.
3. **TypeScript Setup:**
   * [**tsconfig.json**](https://tsconfig.json): Configuration file for TypeScript settings.
   * **Type Definitions**: Define types for components, props, state, and API responses.
4. **ESLint Configuration:**
   * **.eslintrc.js**: Configuration file for linting rules and settings.
   * **Prettier Integration**: To ensure consistent code formatting.
5. **API Integration:**
   * **Axios**: For making HTTP requests to the REST APIs.
   * **Services**:
     + **bookService**: Contains methods to interact with book-related API endpoints.
     + **userService**: Contains methods for user authentication, profile management, and changing passwords.
     + **orderService**: Contains methods to manage orders and order history.
6. **State Management:**
   * **React Context API**: For managing global state (e.g., user authentication status, shopping cart).
7. **Testing:**
   * **Jest and React Testing Library**:
     + **Unit Tests**: Test individual components and functions.
     + **Integration Tests**: Test component interactions and API calls.
     + **Snapshot Tests**: Ensure UI components do not change unexpectedly.
   * **Cucumber**:
     + **BDD Tests**: Write feature files and step definitions to test user stories and application behaviour.
   * **JMeter**:
     + **Performance Tests**: Simulate user load to assess performance.
     + **Load Tests**: Evaluate how the application behaves under heavy load.
8. **CI/CD Pipeline:**
   * **Jenkins**: For automating the build, test, and deployment processes.
   * **Steps**:
     + **Build**: Compile the React application.
     + **Test**: Run unit, integration, snapshot, BDD, and performance/load tests.
     + **Deploy**: Deploy the built application to a web server or cloud platform.

**Menu Options:**

**Admin Menu Options:**

* Add Book
* Update Book
* Remove Book
* View All Books
* Add User
* Remove User
* View All Users
* Insert Books from CSV

**User Menu Options:**

* Home
* View All Books
* Place Order
* Complete Order
* Add Book to Order
* View Order History
* Manage Shopping Cart
* Change Password
* Login/Logout

**Additional Features:**

* **Responsive Design**: Ensure the web UI is mobile-friendly and works well on various screen sizes.
* **User Authentication**: Implement login, logout, and user session management.
* **Error Handling**: Graceful handling of API errors and displaying user-friendly messages.
* **Performance Optimization**: Optimize the application for fast load times and responsiveness.

**Detailed Steps for BDD Testing with Cucumber:**

1. **Feature Files**: Write feature files in the features directory, describing the behaviour of the application in plain language.
2. **Step Definitions**: Implement step definitions in TypeScript files to link the steps in feature files to the application code.
3. **Running Tests**: Use Cucumber to run the BDD tests as part of the testing process.

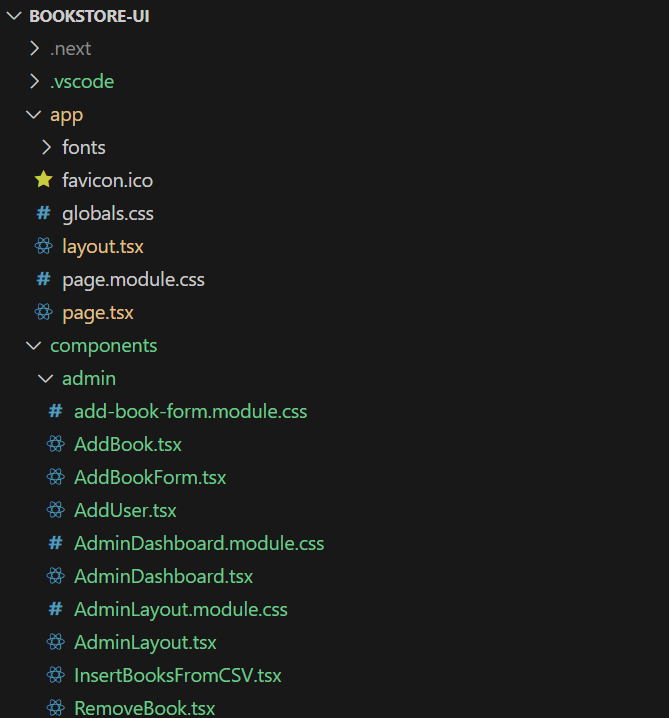
**CI/CD Integration:**

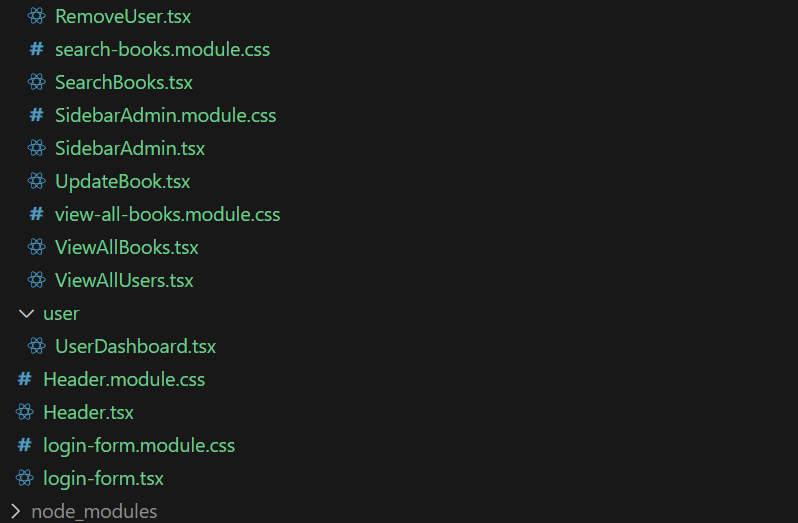
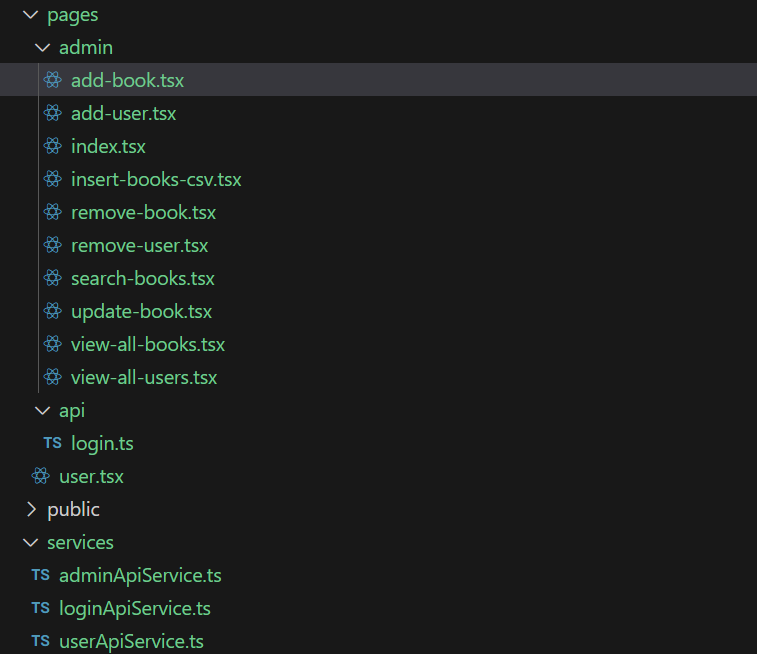
* **Cucumber**: Include steps in the Jenkins pipeline to run Cucumber BDD tests.
* **JMeter**: Integrate JMeter for performance testing in the Jenkins pipeline.

**\*\* The UI design and project structure of the React.js front-end is at your discretion. Pls. ensure all requirements are met.**

**Below are a suggested project structure and minimal UI to give you a feel for the Front-End application.**

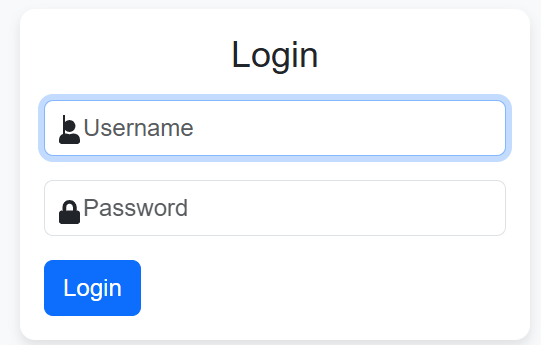
# Project Structure



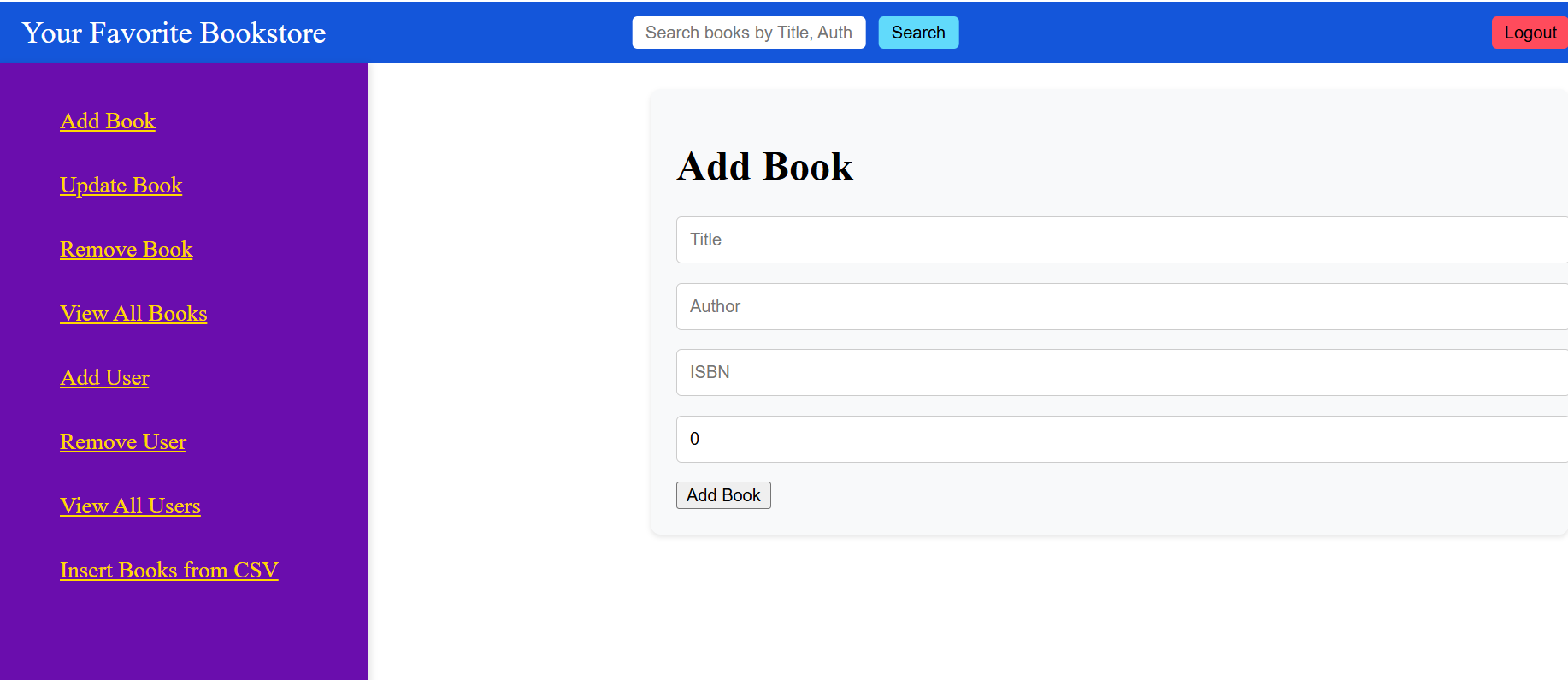
  

## React Application Screens

**Landing Page**

****

**Admin Menu Options**

****

**User Menu Options**

Similar to the Admin Dasboard above.

# References:

* Bookstore App – Solution Outline.docx
* ASDE\_Training\_USER\_STORIES - 2025.xlsx: – Goals and Tasks for each of five Sprints in the Learning Phase.
* 2025.ASDE Training TOC.xlsx: TOC for the Learning Phase.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*