BOOK REVIEW - Book Recommendation App in Springboot & React

About the Project:

“Book Review" is a book-tracking application built using Spring Boot and PostgreSQL. The app is similar to Goodreads and allows users to keep track of books they have read, are currently reading, and want to read in the future. Users can also rate and review books, create and join book clubs, and discover new books to read through personalized recommendations.

The app utilizes PostgreSQL as its database management system, allowing high scalability and fast data retrieval. Additionally, Spring Boot is used to handling the application's backend services, such as user authentication, book data management, and communication with the Open Library API for search functionality.

Project Details:

By working on this project you will gain a deep understanding of various concepts related to Spring Boot and PostgreSQL. You will learn how to use Spring Boot to create web applications, handle user authentication, and access a PostgreSQL database to store and retrieve data. You will also learn about the basics of PostgreSQL and how to use it to store and manage data for your application.

You will also learn about RESTful web services, which is a way of creating web services that follow the principles of Representational State Transfer (REST). This means that the web services can be accessed using standard HTTP requests and responses, and they return data in a format that is easy to parse and use, such as JSON.

Prerequisites of the Project:

In order to work on the project, "Book Review" there are certain prerequisites that must be met:

1. Backend(Springboot) - JAVA(Preferred), Python, Nodejs, GO
    - Spring Security(Authentication)
    - REST API
2. SQL(PostgreSQL) - Preferred, MySQL, Cassandra.
3. FrontEnd(React) - React(Preferred), Python

Project Module Breakdown:

The "Book Review" project has been divided into two modules:

Module 1: Backend Module:

Task 1: System Design, Schema Design, Setup, and Installation for Backend.

Task 2: This module will handle the application's backend services and communication with the PostgreSQL database. It will be developed using Spring Boot and expose RESTful web services to interact with the front end.

Module 2: (Front End Module)

Task 1: This module will handle storing the books read by a user, fetch book data from the Open Library API and save it to the database.

Task 2: The application's frontend functionality, such as user interface, navigation, and user interactions. It will be developed using React. The frontend module will consume the RESTful web services provided by the backend module to display data and perform actions on the user's behalf.

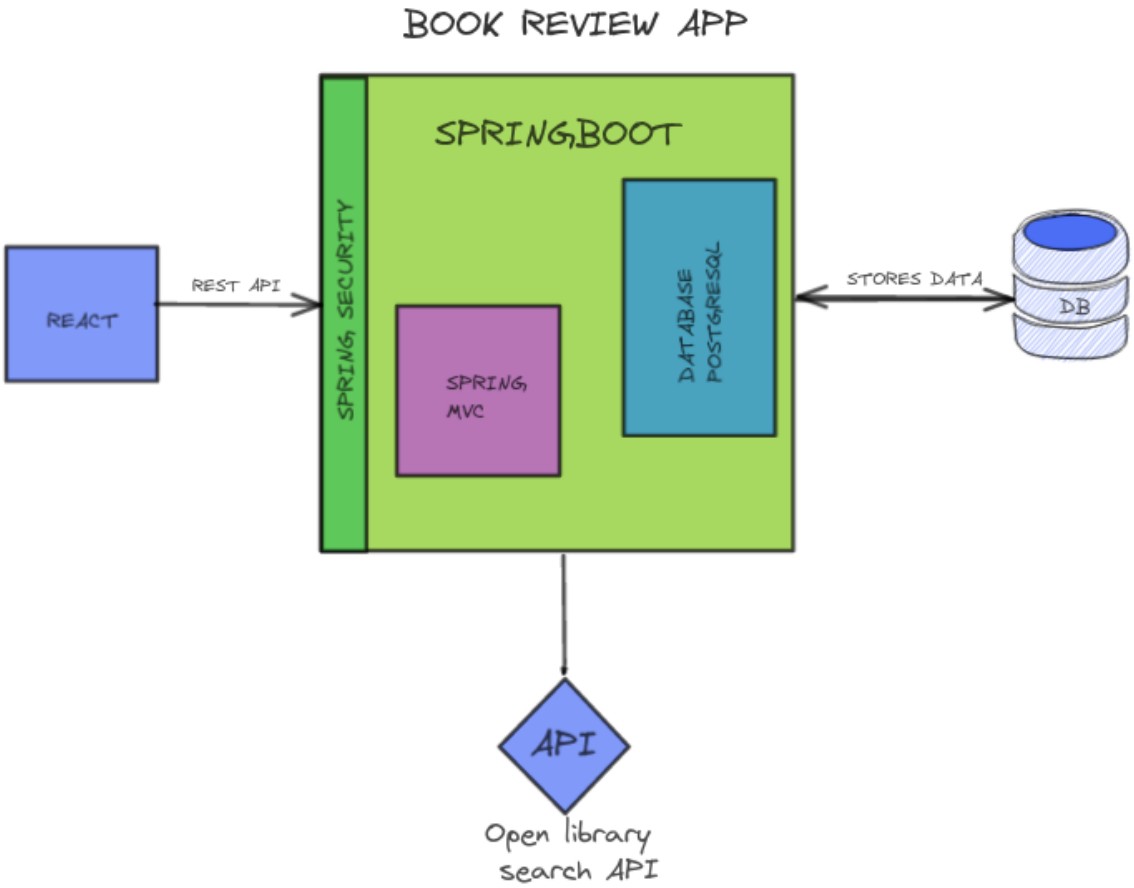
Module 1: System Design, Schema Design, Setup, Installation, And Data Management

In this module, you will learn about the requirements and setup for a Spring Boot application with PostgreSQL. This module includes understanding the requirements like system design and schema design and setting up the environment by installing PostgreSQL and SpringBoot. Also using open library API to manipulate the book data using this API.

Task 1: Understanding the Requirements and Design the Schema

System Design: The system design for the "Book Review" project can be divided into several major components:

1. User Interface: The frontend of the application, developed using any front-end framework React, will provide users with a simple and intuitive interface to interact with the application. It will allow users to search for books, view book details, add books to their library, and leave reviews and ratings.
2. RESTful Web Services: The backend of the application, developed using Spring Boot, will expose a set of RESTful web services to handle user authentication, book data management, and communication with the PostgreSQL database. The frontend module will consume these web services to display data and perform actions on the user's behalf.
3. PostgreSQL Database: This will store the application's data, including user information, book information, and reviews. It will be optimized for high scalability and fast data retrieval, allowing the application to handle many users and books.
4. Open Library API: The application will use the Open Library API to access book data and provide users with additional information about the books they are interested in. The API will be integrated into the backend module and will be used to retrieve data such as book titles, authors, publication dates, and cover images.
5. User Authentication: The application will handle user authentication using the Spring Security framework. Users can register, login, and log out into the application.



Schema Design:

The "Book Review" project will have several different types of data that need to be stored and managed, including books, authors, and users. Here's a possible schema design for these entities in PostgreSQL, you can even add more or less and create an ER diagram:

1. Book:
    - book_id (UUID): unique identifier for each book
    - title (text): title of the book
    - author (text): name of the author

    - description (text): brief description of the book
    - publication_date (date): date the book was published
    - cover_image (text): URL of the cover image of the book
    - reviews (map<text,text>): map containing reviews and ratings of the book
    - PRIMARY KEY (book_id)
2. Author:
    - author_id (UUID): unique identifier for each author
    - name (text): name of the author
    - bio (text): brief biography of the author
    - PRIMARY KEY (author_id)
3. User:
    - user_id (UUID): unique identifier for each user
    - name (text): name of the user
    - PRIMARY KEY (user_id)

Subtask 1: Create a Schema Diagram using this website DBDiagram

Submission 1: Submit the ER-Diagram

Subtask 2: Setup and Installation:

Steps to create a Spring Boot project that uses Postgres as the database and Docker Compose to manage the dependencies, along with a React frontend:

1. Create a new Spring Boot project using a tool such as the Spring Initializer.
    a. For this project, we will use Maven as our build tool, Spring version 3.0.2 (or whatever is latest in the 3.x series).
    b. Include the dependencies for Spring Web, Spring Data JPA, and Postgres.
    c. Fill in the rest of the details and generate the project.
    d. Unzip the downloaded code and open it in IntelliJ IDEA
2. Create a `docker-compose.yml` file in the root directory of the project.
    a. In this file, you will define the services for the Spring Boot application, Postgres, and any other dependencies you may need.
    b. In the docker-compose.yml file, define a service for Postgres, specifying the image to use and the environment variable POSTGRES_PASSWORD=password to specify the password for the

database. Refer to the image documentation for more environment variables to customize the docker container.

3. To create the React application you can use `npx create-react-app --template typescript frontend` to create a new project.
    a. Run the following command to install React Router: `npm install react-router`
    b. Run the following command to install React Router DOM: `npm install react-router-dom`
4. Run docker-compose up to start the services defined in the docker-compose.yml file.
5. Run `mvnw spring-boot:run` to start the spring server
6. Run npm start to start the frontend, and open http://localhost:3000 on your browser
7.

Submission 2: Submit the zip file for the setup and installation

Subtask 3: To implement the register and login features in the app, you can follow these steps:

1. Create the user model as discussed above and a user repository to manage entries to the database.
2. Create a Registration service to add a new user. Make sure to configure a password encoder to store encoded passwords.
3. Create the RegisterController and DTO. The register controller will have a single endpoint POST /register and accepts register DTO as its body. The registered DTO should contain the username and password. The controller should pass the result to the Register service and return a 200 response
4. In the react frontend, create new pages for registration and new user welcome and configure it with react-router.
5. On the registration page, build a form using react-hook-forms and make a request to our register endpoint on submitting the form. On success response from the server return the user to the welcome page.
6. We will use JWT tokens for authenticating our react frontend with spring boot backend, we will use jwt library to generate and validate tokens on the backend.
7. You will need to create a login page that will allow the user to enter their credentials and send a request to the server to authenticate. Use react-hook forms to build this page.

8. Create a LoginController to handle the incoming request. Validate it with an authentication manager and return a token if details are correct, 401 otherwise.
9. Store the resulting token after successful login.
10. After login, Into your React application, you will need to make sure that you are sending the JWT with each request to the server. This can be done by storing the JWT in the browser's local storage and adding it to the Authorization header of each request.
11. To validate the JWT token passed by the front end, use a Spring Security Request filter
    a. Create a new class that implements the Filter interface. This class will validate the JWT and set the authentication details in the SecurityContext.
    b. In the doFilter method, you will need to extract the JWT from the request's Authorization header.
    c. Then, you will need to validate the JWT using JJWT
    d. If the JWT is valid, you will need to extract the user's information from the JWT and create an Authentication object.
    e. Finally, you will need to set the Authentication object in the SecurityContext by calling the SecurityContextHolder.getContext().setAuthentication(authentication) method.
    f. Alternatively, you can use AbstractAuthenticationProcessingFilter as well

Submission 3: Write the Register and Login Authentication.

Module 1 solution submission form: Click here

Module 2: Integrating with Open Library API, Controllers for Book Review and Search

Module 2 of the Book Review application involves integrating with the Open Library API to search for books and display information about them. The module includes the following tasks:

1. Calling the Open Library API
2. Writing controllers for searching books
3. Writing controllers for reviewing books
4. Testing: You will write unit tests for each of the controllers using JUnit and MockMVC. These tests will verify that the controllers return the correct information and respond to requests as expected.

The overall goal of Module 2 is to create a complete end-to-end flow for searching and reviewing books in the Book Review application. By the end of the module, users should be able to search for books, view details about them, and leave reviews.

Subtask 1:

To store the books read by a user, fetch book data from the Open Library API, and save it to the database, you can follow these steps:

1. Create the Books and Authors Model as we designed
2. Create the Books and Authors DAO: In the data access layer, you will implement the logic for interacting with the database. This includes tasks such as inserting new records into the books and authors tables, fetching records from the tables, and updating existing records.
3. Create an open library API client which calls open API to get the search results.
4. Create the Books and Authors Service: In the service layer, you will implement the business logic for saving the fetched book and author data to the database. This

includes creating new records in the books and authors' tables and updating existing records.
5. Create a Controller for searching for a book: Using a spring boot controller and a service, we will search for a book from Open Library Search API. We will parse the results and show them results to the user.
6. Frontend: Search for Book Page and add it to Personal Library Page

for example, the 'useEffect' hook is used to fetch the list of books from the backend API when the component is first rendered. The fetch function is used to make a GET request to the API endpoint and the response is converted to JSON using the json method. The resulting data is then stored in the component's state using the setBooks function.

Finally, the list of books is displayed in an unordered list, with each book being represented as a list item. The key for each list item is set to the book's id to ensure that React can render the list efficiently.

Subtask 2:

To rate and review the books

To create an API to rate and review books, you could follow these steps:

1. Create a Book model to store information about books, such as the title, author.
2. Create a Review model to store information about user reviews, such as the reviewer's username, rating, and review text.
3. Create a ReviewRepository to manage entries in the database for the Review model.
4. Create a ReviewController with endpoints to handle requests to create and retrieve reviews.
5. In the frontend, create a page to display book information and allow users to submit a review. This can be done using React and making requests to the ReviewController endpoints.
6. When a user submits a review, make a POST request to the /review endpoint with the review information, including the book ID and the user's rating and review text.

7. In the ReviewController, use the ReviewRepository to save the review information to the database.
8. To retrieve reviews for a specific book, make a GET request to the /reviews/{bookID} endpoint, where bookID is the ID of the book you want to retrieve reviews for.
9. In the ReviewController, use the ReviewRepository to retrieve the reviews for the specified book ID and return them in the response.

Subtask 3: Testing

1. Test the APIs in Postman
2. Write Unit Test Cases for Backend services
3. Test the UI

Module 2 solution submission form: Click here

Submission Process & Guidelines:

Each module is to be completed in one week. Submissions will be on weekly basis. Module 1 submission is to be done in the first week and module 2 submission in the next consecutive week.

Guidelines for submissions are mentioned below:

1. A google form will be circulated for module file submissions.
2. For module 1 you have to submit ER-Diagram in PNG format and 2 zip files as mentioned below:
    a. ER-Diagram for the App
    b. Setup and Installation Code - .zip format or GitHub URL
    c. Authentication Code - .zip format or GitHub URL

3. Create your own youtube, zoom or vimeo video and attach the link that consists of the walkthrough of a code setup and authentication in the submission form itself. .

Evaluation Criteria:

1. The project submission guidelines and prerequisites are mandatory and non-negotiable.
2. The submissions that are completed will be considered for evaluation.
3. Copied code will not be considered for evaluation.

Session Schedule & Timelines:

Day	Date	Time	Session
Wednesday	**25th January**	**6:00 PM PST**	**Project Kickoff Call/ Introduction to Module 1**
Saturday	**30th January**	**6:00 PM PST**	**Doubt Solving Session for Module 1**
Friday	**6th February**	**6:00 PM PST**	**Module 1 Submission**

**Any changes in the dates and time of the sessions will be informed in advance.*

Important Links related to the Live Project:

1. Live Project Kick-Off Session Click here
2. Doubt Clearing Session for Module 1: Click here
3. Introduction to Module 2: Click here
4. Team members google sheets:
5. Module solution submission form: Click here