PS 2: TITLE: BOOK EXCHANGE PLATFORM

Group-29

Students Name

Sourav Bose 2021MT70503

Harichandana Geelapaturu 2021MT70587

Git Hub Link: <https://github.com/chandanakookie/Book-exchange-application.git>

Problem Statement:

Book lovers frequently accumulate a collection of books they have read and look for other recommendations. They are always eager to explore new reading material. Traditional methods of exchanging books, such as localbook swaps or lending among friends, are limited in scope and accessibility. Therefore, it is imperative to have adigital platform that can facilitate book exchanges on a larger scale. This platform should connect users withsimilar reading interests, enabling them to trade books easily and efficiently. The objective of this project is to

develop a full-stack web application that serves as a centralized platform for users to exchange, lend, and borrowbooks with other users. The platform should provide a user-friendly interface, robust search and recommendationfeatures, and secure transaction capabilities.

Features:

User Authentication: Implement a secure user authentication system to allow users to register, log in, andmanage their accounts.

Book Listing: Enable users to list books they want to exchange or lend, including details such as title, author, genre, condition, and availability status.Book Search: Provide users with advanced search and filtering options to discover books based on criteria likegenre, author, title, location, and availability.Exchange Requests: Allow users to send and receive exchange requests for specific books, including negotiation.

options for terms such as delivery method and duration.Messaging System: Implement a messaging system to facilitate communication between users regarding bookexchanges, including negotiation details, logistics, and scheduling. Notifications: Send notifications to users for exchange requests, messages, and updates related to their book

listings or requests.User Profiles: Enable users to create profiles with information about their reading preferences, favourite genres,and books they currently own or wish to acquire. Transaction Management: Provide tools for users to track the status of their exchange transactions, includingpending requests, accepted exchanges, and completed transactions.

Expected outcome.

The book exchange platform should provide users with a convenient and efficient way to discover new readingmaterial, share their favourite books with others, and connect with fellow book enthusiasts in their community. Byfacilitating book exchanges, the platform aims to promote a culture of reading and foster a sense of communityamong users.

*Answer*

To develop a full-stack web application for a book exchange platform, we'll break down the project into several key components and outline a high-level implementation plan. This plan will include the required features, technology stack, and steps to ensure the platform meets the desired objectives.

Key Features

1. User Authentication:

- Secure user registration, login, and account management.

- Password encryption and user data protection.

2. Book Listing:

- Users can list books with details (title, author, genre, condition, availability).

- Book cover images and descriptions.

3. Book Search:

- Advanced search and filtering by genre, author, title, location, and availability.

4. Exchange Requests:

- Users can send and receive exchange requests.

- Options for negotiating terms such as delivery method and duration.

5. Messaging System:

- In-app messaging to facilitate communication regarding exchanges.

- Chat history and notifications.

6. Notifications:

- Real-time notifications for exchange requests, messages, and updates.

7. User Profiles:

- Profiles with reading preferences, favorite genres, and book collections.

- User activity and reputation system (e.g., ratings, reviews).

8. Transaction Management:

- Tools to track the status of exchanges (pending, accepted, completed).

- History of past transactions.

Technology Stack

- Frontend:

- React.js for a responsive and dynamic user interface.

- Redux for state management.

- CSS/Bootstrap for styling.

- Backend:

- Node.js with Express.js for the server.

- RESTful API design for communication between frontend and backend.

- Database:

- MongoDB for storing user data, book listings, messages, and transaction history.

- Authentication:

- JWT (JSON Web Tokens) for secure authentication.

- Bcrypt for password hashing.

- Real-time Features:

-

Socket.io

for real-time messaging and notifications.

Implementation Steps

# Phase 1: Project Setup and Basic Functionality

1. Initial Setup:

- Set up the project repository (e.g., GitHub).

- Initialize frontend (React) and backend (Node.js, Express) environments.

- Configure MongoDB database.

2. User Authentication:

- Implement user registration and login routes.

- Set up JWT for secure token-based authentication.

- Create user profile management.

3. Basic UI:

- Design the main components (header, footer, navigation).

- Create initial routes and pages (home, login, register).

# Phase 2: Core Features

4. Book Listing:

- Create a form for users to list books.

- Develop routes to add, edit, and delete book listings.

- Display user book listings on their profile.

5. Book Search:

- Implement search and filtering functionalities.

- Design search results page with book details.

6. Exchange Requests:

- Enable users to send exchange requests.

- Develop negotiation options and request tracking.

7. Messaging System:

- Set up

Socket.io

for real-time messaging.

- Create messaging interface and backend routes.

8. Notifications:

- Implement notification system for new messages, requests, and updates.

- Design notification UI and backend logic.

# Phase 3: Enhanced Functionality and Testing

9. User Profiles:

- Enhance profile pages with reading preferences and activity.

- Add features for rating and reviewing other users.

10. Transaction Management:

- Develop transaction tracking interface.

- Implement backend logic for managing transaction status.

11. Testing:

- Perform unit and integration tests for all components.

- Conduct user acceptance testing (UAT) to gather feedback.

12. Deployment:

- Deploy the application to a cloud provider (e.g., Heroku, AWS).

- Set up environment variables and configure production settings.

# Phase 4: Post-Launch Enhancements

13. User Feedback and Iteration:

- Gather user feedback and identify areas for improvement.

- Implement enhancements based on user suggestions.

14. Maintenance and Support:

- Regularly update dependencies and libraries.

- Monitor the application for bugs and performance issues.

Conclusion

This project plan outlines a comprehensive approach to developing a full-stack book exchange platform. By following these steps and utilizing the specified technology stack, the platform will provide a seamless and engaging experience for users, fostering a community of book enthusiasts and promoting a culture of reading and sharing.

let's create a minimal version of the full-stack application with the specified features. We'll start with the backend using Spring Boot and H2 Database, and then move to the frontend using Angular.

Backend: Spring Boot

**1. Setting up Spring Boot Project**

First, create a new Spring Boot project using Spring Initializer with dependencies for Web, Security, JPA, and H2.

`pom.xml`:

xml

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-rest</artifactId>

</dependency>

</dependencies>

**2. Configuration**

`application.properties`:

```properties

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.h2.console.enabled=true

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

jwt.secret=mysecretkey

```

**3. User Authentication**

User Entity:

```java

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String username;

private String password;

private String role;

// getters and setters

}

```

User Repository:

```java

public interface UserRepository extends JpaRepository<User, Long> {

User findByUsername(String username);

}

```

User Service:

```java

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public User save(User user) {

user.setPassword(new BCryptPasswordEncoder().encode(user.getPassword()));

return userRepository.save(user);

}

public User findByUsername(String username) {

return userRepository.findByUsername(username);

}

}

```

Security Configuration:

```java

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private UserService userService;

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(username -> {

User user = userService.findByUsername(username);

return new org.springframework.security.core.userdetails.User(user.getUsername(), user.getPassword(), Collections.singletonList(new SimpleGrantedAuthority(user.getRole())));

}).passwordEncoder(new BCryptPasswordEncoder());

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests()

.antMatchers("/register", "/login").permitAll()

.anyRequest().authenticated()

.and()

.addFilter(new JwtAuthenticationFilter(authenticationManager()))

.addFilter(new JwtAuthorizationFilter(authenticationManager()));

}

}

```

**JWT Authentication and Authorization Filters (omitted for brevity, but they should handle generating and validating JWT tokens).**

**Authentication Controller:**

```java

@RestController

public class AuthController {

@Autowired

private UserService userService;

@PostMapping("/register")

public ResponseEntity<?> register(@RequestBody User user) {

user.setRole("ROLE\_USER");

return ResponseEntity.ok(userService.save(user));

}

@PostMapping("/login")

public ResponseEntity<?> login(@RequestBody AuthRequest authRequest) {

// Handle authentication and return JWT

}

}

class AuthRequest {

private String username;

private String password;

// getters and setters

}

```

**# 4. Book Management**

Book Entity:

```java

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

private String genre;

private String condition;

private boolean available;

@ManyToOne

@JoinColumn(name = "user\_id")

private User owner;

// getters and setters

}

```

Book Repository:

```java

public interface BookRepository extends JpaRepository<Book, Long> {

List<Book> findByGenre(String genre);

List<Book> findByTitleContaining(String title);

}

```

Book Controller:

```java

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@PostMapping

public Book addBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@GetMapping("/search")

public List<Book> searchBooks(@RequestParam(required = false) String title,

@RequestParam(required = false) String genre) {

if (title != null) {

return bookRepository.findByTitleContaining(title);

} else if (genre != null) {

return bookRepository.findByGenre(genre);

} else {

return bookRepository.findAll();

}

}

}

```

**Frontend: Angular**

**1. Setting up Angular Project**

Use Angular CLI to create a new Angular project.

```sh

ng new book-exchange

cd book-exchange

ng add @angular/material

```

**2. Authentication Service**

`auth.service.ts`:

```typescript

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

@Injectable({

providedIn: 'root'

})

export class AuthService {

private baseUrl = 'http://localhost:8080';

constructor(private http: HttpClient) { }

register(user: any): Observable<any> {

return this.http.post(`${this.baseUrl}/register`, user);

}

login(credentials: any): Observable<any> {

return this.http.post(`${this.baseUrl}/login`, credentials);

}

}

```

# 3. User Registration Component

`register.component.ts`:

```typescript

import { Component } from '@angular/core';

import { AuthService } from '../services/auth.service';

import { Router } from '@angular/router';

@Component({

selector: 'app-register',

templateUrl: './register.component.html',

styleUrls: ['./register.component.css']

})

export class RegisterComponent {

user = { username: '', password: '' };

constructor(private authService: AuthService, private router: Router) { }

register() {

this.authService.register(this.user).subscribe(() => {

this.router.navigate(['/login']);

});

}

}

```

`register.component.html`:

```html

<form (ngSubmit)="register()">

<input [(ngModel)]="user.username" name="username" placeholder="Username" required>

<input [(ngModel)]="user.password" type="password" name="password" placeholder="Password" required>

<button type="submit">Register</button>

</form>

```

**# 4. Login Component**

`login.component.ts`:

```typescript

import { Component } from '@angular/core';

import { AuthService } from '../services/auth.service';

import { Router } from '@angular/router';

@Component({

selector: 'app-login',

templateUrl: './login.component.html',

styleUrls: ['./login.component.css']

})

export class LoginComponent {

credentials = { username: '', password: '' };

constructor(private authService: AuthService, private router: Router) { }

login() {

this.authService.login(this.credentials).subscribe((response: any) => {

localStorage.setItem('token', response.token);

this.router.navigate(['/']);

});

}

}

```

`login.component.html`:

```html

<form (ngSubmit)="login()">

<input [(ngModel)]="credentials.username" name="username" placeholder="Username" required>

<input [(ngModel)]="credentials.password" type="password" name="password" placeholder="Password" required>

<button type="submit">Login</button>

</form>

```

**# 5. Book Listing and Search**

Book Service:

`book.service.ts`:

```typescript

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

@Injectable({

providedIn: 'root'

})

export class BookService {

private baseUrl = 'http://localhost:8080/books';

constructor(private http: HttpClient) { }

getBooks(): Observable<any> {

return this.http.get(this.baseUrl);

}

searchBooks(title: string, genre: string): Observable<any> {

return this.http.get(`${this.baseUrl}/search`, { params: { title, genre } });

}

addBook(book: any): Observable<any> {

return this.http.post(this.baseUrl, book);

}

}