Book Exchange Platform

The Book Exchange

Submitted by:

Ameya Bhujbal

2023TM93772

FSAD, BITS Mtech

Web Application Design Document

1. Project Overview

Purpose & Objectives

Purpose

The Book Exchange Platform addresses the challenge faced by book lovers who accumulate books and desire fresh reading material but are constrained by the limited reach of traditional book swaps or lending practices. By providing a digital platform, the solution aims to overcome these barriers, fostering an inclusive and dynamic book-sharing ecosystem that promotes reading and community-building.

Objectives

1. **Facilitate Easy Book Exchanges**: Provide a centralized, user-friendly platform to list, browse, and request books for exchange or lending.
2. **Enhance Accessibility**: Connect book enthusiasts across different locations and communities, enabling exchanges beyond local circles.
3. **Encourage Sustainability**: Promote the reuse of books, reducing waste and encouraging environmentally friendly reading habits.
4. **Foster Community Engagement**: Create a space where users can interact, share recommendations, and discuss their favorite books.
5. **Leverage Technology for Efficiency**: Utilize robust search, filtering, and secure transaction features to ensure seamless exchanges.

2. Functional Requirements

2.1 User Stories

2.1.1 User Story 1: User Authentication

As a user, I want to securely register, log in, and manage my account, So that I can access and use the book exchange platform.

Acceptance Criteria:

- The platform must allow users to register with a valid email and password.
- Passwords must be stored securely using encryption.
- Users should be able to reset their password via a password recovery system.
- Users should be able to log out from their account.

2.1.2 User Story 2: Book Listing

As a user, I want to list books that I want to exchange or lend, So that others can browse and request the books I offer.

Acceptance Criteria:

- Users should be able to add a book to their list by providing details such as title, author, genre, condition, and availability status.
- Each book listing must have a unique ID associated with a user's profile.
- Users should be able to edit or delete book listings at any time.
- The book listing must be displayed in the user's profile and searchable by others.

2.1.3 User Story 3: Book Search

As a user, I want to search for books based on criteria such as title, author, genre, and location,

So that I can easily find books that interest me.

Acceptance Criteria:

- A search bar allows users to input keywords like title, author, or genre.
- Search results can be filtered by availability status, genre, and location.
- Detailed information about books is displayed when users click on search results.
- Results are paginated or load incrementally to optimize performance for large datasets.

2.2 Features & Functionality

Backend (Node.js with Express)

- Server:
  - The backend is built with Express.js, offering a robust framework to define RESTful APIs for user authentication, book management, and search functionalities.
  - Body-parser: Parses incoming request bodies for handling form data (e.g., book details, user registration).
  - Cookie-parser: Helps manage cookies for user session tracking.
- Database Integration:
  - pg (PostgreSQL): For relational database operations, ensuring structured storage of users, books, transactions, and reviews.
- Security:
  - bcrypt: Encrypts passwords for secure user authentication.
  - cors: Handles cross-origin requests to enable secure interaction between frontend and backend (not listed but recommended).
- Testing Backend:
  - Jest: Unit and integration testing for backend endpoints.
  - Supertest: Testing HTTP requests to ensure the API works as intended.

Frontend (React)

- User Interface:
  - Built with React.js to create dynamic and responsive single-page applications (SPAs).
  - React Router DOM: Implements navigation for pages like login, book listings, and book details.
  - React Bootstrap: Provides pre-styled UI components (e.g., modals, forms, buttons) for a polished and consistent design.
- Styling:
  - Bootstrap: For default styling and responsive design.
  - Sass: Enables advanced styling features like nested rules and variables.
  - Sass-loader & PostCSS: Processes styles efficiently during development and production builds.
- Frontend Testing:
  - @testing-library/react & jest-dom: Write unit and integration tests for React components.
  - @testing-library/user-event: Simulate user interactions to test UI behavior.

Build & Development Tools

- Development Workflow:
  - Nodemon: Automatically restarts the server on code changes for efficient backend development.
  - Webpack: Bundles frontend assets (JavaScript, CSS, images) for production.
  - Webpack-dev-server: Provides a live development server with hot reloading.
- Concurrent Development:
  - Concurrently: Runs both backend (nodemon) and frontend (webpack-dev-server) in parallel for seamless development.
- Babel:
  - Transpiles modern JavaScript (ES6+) and JSX into browser-compatible JavaScript.
  - @babel/preset-react: Handles JSX syntax.
  - @babel/preset-env: Ensures backward compatibility with older browsers.

2.3 Key Features Enabled by This Setup

1. **User Authentication**:
    - Securely handle user registration and login using **Express.js** and **bcrypt**.
    - Maintain session data with cookies and possibly JWT for secure authentication.
2. **Book Listings**:
    - Use **PostgreSQL** to store book details and retrieve them via RESTful APIs.
    - Dynamically render book listings on the frontend using **React**.
3. **Book Search**:
    - Implement advanced filtering and sorting with PostgreSQL queries and expose them through backend APIs.
    - Display filtered results dynamically using React's state management.
4. **Responsive Design**:
    - Ensure mobile and desktop responsiveness with **React Bootstrap** and custom styling via **Sass**.
5. **Testing**:
    - Use **Jest** and **Testing Library** to write comprehensive tests for both backend (API endpoints) and frontend (React components).
6. **Real-Time Updates**:
    - Enable hot-reloading during development using **webpack-dev-server** and **concurrently**.
7. **Production Optimization**:
    - Minimize the frontend build size and optimize performance using **Webpack** with production settings.

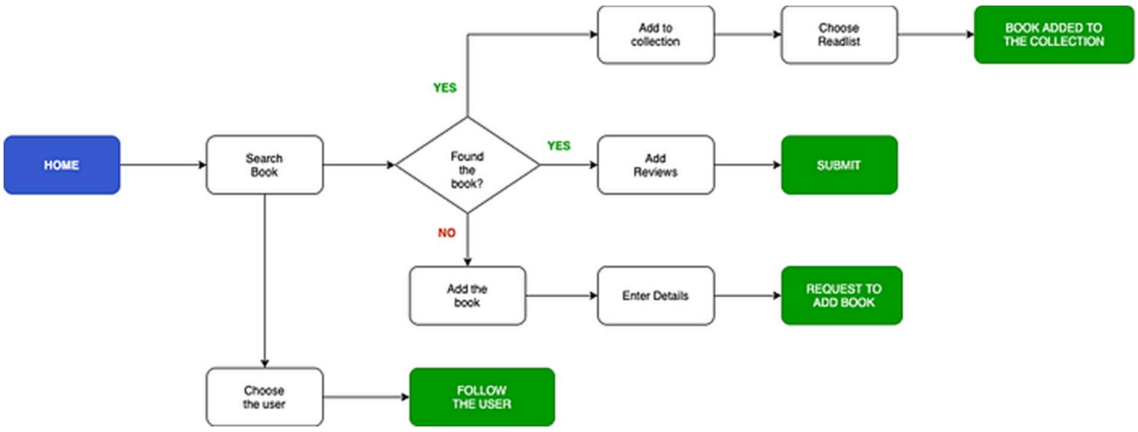
3. Architecture Overview

3.1 System Architecture

The system architecture for the Book Exchange Platform is a **three-tier architecture** consisting of:

1. **Presentation Layer (Frontend)**
2. **Application Layer (Backend)**
3. **Data Layer (Database)**

It is designed to ensure scalability, maintainability, and security while offering a seamless user experience.



3.2 Components of the Architecture

1. Presentation Layer (Frontend)

- **Technology**: React.js, React Router DOM, Bootstrap
- **Description**:
    - This layer provides the user interface, allowing users to interact with the platform.
    - Responsive and dynamic single-page application (SPA) for functions like login, book listing, and search.
    - Communicates with the backend via RESTful APIs.
- **Key Features:**
    - Search bar with advanced filters.
    - Book details and user profiles rendered dynamically.
    - Mobile responsiveness and cross-browser compatibility.

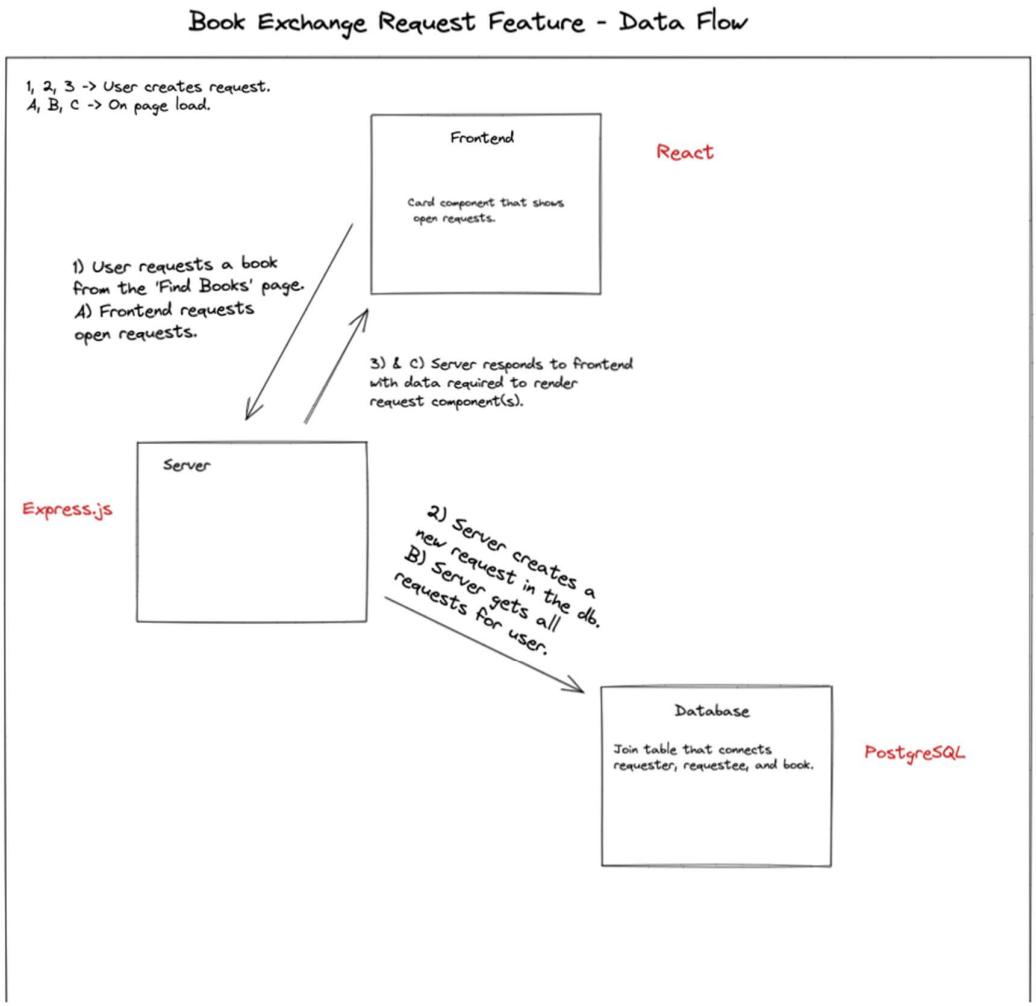
---

2. Application Layer (Backend)

- **Technology**: Node.js, Express.js
- **Description**:
    - Acts as the middle layer to process user requests and business logic.
    - Provides APIs for user authentication, book management, and search.
    - Ensures data validation and implements security measures.
- **Key Features:**
    - RESTful API design for CRUD operations.
    - Authentication with password hashing (bcrypt).
    - Session management using cookies or JWT.

4. **Data Layer (Database)**

- **Technology**: PostgreSQL
- **Description**:
  - Stores structured data, including user profiles, book listings, and transaction logs.
  - Optimized with indexes for quick search and filtering.
- **Key Features:**
  - Relational schema for handling relationships between users and books.
  - ACID compliance to ensure secure and reliable transactions.


Book Exchange Request Feature - Data Flow
1, 2, 3 -> User creates request.
A, B, C -> On page load.
Frontend
React
Card component that shows open requests.
1) User requests a book from the 'Find Books' page.
A) Frontend requests open requests.
3) & C) Server responds to frontend with data required to render request component(s).
Server
Express.js
2) Server creates a new request in the db.
B) Server gets all requests for user.
Database
Join table that connects requester, requestee, and book.
PostgreSQL

4. Data Flow

Frontend to Backend Interaction:

1. **User Authentication**:
    - User enters credentials on the frontend.
    - Request sent to /auth/login API.
    - Backend validates credentials and generates a token (JWT or session cookie).
2. **Book Listings**:
    - Users can add/edit/delete books via /books API.
    - Data includes title, author, genre, condition, and location.
    - Stored in the database with associations to the user profile.
3. **Book Search**:
    - Search queries (e.g., genre, title, location) are sent to /books/search.
    - Backend fetches matching records from PostgreSQL and sends paginated results.

5. API Design

API design for your **Book Exchange Platform** involves structuring the routes, methods, data formats, and error handling to support key actions like verifying users, adding books, finding books, and managing exchanges. Here's how the API design could look for your platform, based on your requirements:

Defining the Resources

Resources in your platform are mainly centered around **users** and **books**. The API will handle operations related to:

- **Users**: Actions like registering, verifying, and viewing user profiles.
- **Books**: Actions like adding books to the inventory, searching for books, requesting books, and managing the book list.

HTTP Methods:

GET: Retrieves data from the server. Used for searching books, viewing user profiles, or fetching a list of books owned by a user.

POST: Sends data to the server to create new resources (e.g., registering a user, adding a new book).

DELETE: Removes a resource from the server (e.g., deleting a book).

PUT/PATCH: If your platform supports updating books or user profiles, these methods will modify the existing data.

5.1 API Endpoints

Route	Method	Description	Parameters	Request Body	Response Codes	Response Example
/verifyUser	POST	Verifies the identity of a user (via credentials or token).	username, password, token (optional)	username: string, password: string, token: string (optional)	200 OK 401 Unauthorized	{ "status": "success", "userID": "12345", "message": "User verified successfully" }
/register	POST	Registers a new user on the platform.	None	username: string, email: string, password: string, fullName: string, contactNumber: string (optional)	201 Created, 400 Bad Request	{ "status": "success", "userID": "12345", "message": "Registration successful" }
/seeuser	GET	Retrieves details of a specific user.	userID: string	None	200 OK, 404 Not Found	{ "status": "success", "user": { "username": "johndoe", "email": "john@example.com", "books": [...] }, "message": "User found" }
/findBook	GET	Searches for books by title, author, ISBN, or condition.	title, author, ISBN, condition	None	200 OK, 404 Not Found	{ "status": "success", "books": [{ "title": "Book 1", "author": "Author 1", "ISBN": "123456", "condition": "New", "owner": "user123" }] }

Route	Method	Description	Parameters	Request Body	Response Codes	Response Example
/addOldBook	POST	Adds a user's old book to the platform.	None	title: string, author: string, ISBN: string, condition: string, owner: string	201 Created, 400 Bad Request	{ "status": "success", "bookID": "56789", "message": "Book added successfully" }
/findOldBook	GET	Searches for old books listed by other users.	title, author, ISBN, condition	None	200 OK, 404 Not Found	{ "status": "success", "books": [{ "title": "Book 1", "author": "Author 1", "ISBN": "123456", "condition": "Used", "owner": "user123" }] }
/requestBook	POST	Allows a user to request a book from another user..	None	bookID: string, userID: string, message: string (optional)	200 OK, 400 Bad Request	{ "status": "success", "message": "Book request sent successfully" }
/deleteOldBook	DELETE	Removes an old book from a user's inventory.	bookID: string	None	200 OK, 404 Not Found	{ "status": "success", "message": "Book deleted successfully" }
/getMyOldBookList/	GET	Retrieves a list of old books owned by a specific user.	userID: string	None	200 OK, 404 Not Found	{ "status": "success", "books": [{ "title": "Book 1", "author": "Author 1", "ISBN": "123456", "condition": "Used", "owner": "user123" }] }

5.2 Detailed Explanation of Design

Resources and Relationships:

- **User**: Each user has a profile and a list of books they own. A user can request books from other users, and they can add, update, or remove books from their collection.
- **Book**: Each book is represented by parameters like title, author, ISBN, condition, and owner (a user).

Error Handling:

Clear, descriptive error messages are crucial for debugging and improving user experience.

- **400 Bad Request**: Invalid input or missing required fields.
- **401 Unauthorized**: User needs to authenticate.
- **404 Not Found**: Resource (book, user) does not exist.
- **500 Internal Server Error**: Unexpected server-side issues.

Security Considerations

- **Authentication**: For endpoints like /verifyUser, use OAuth or JWT (JSON Web Tokens) for securing user sessions.
- **Authorization**: Ensure that users can only modify or access their own data (e.g., adding or deleting books).
- **Input Validation**: Always validate user inputs to prevent SQL injection, cross-site scripting (XSS), or other common attacks.

Performance Considerations

- **Caching**: Implement caching strategies to avoid querying the database repeatedly for the same data (e.g., frequently searched books).
- **Pagination**: For endpoints like /getMyOldBookList, return paginated results if there are many books.
- **Rate Limiting**: Protect your API from abuse by limiting the number of requests a user can make in a given time frame.

6. Implementation Structure

```
BookExchangeApp/
├── client/
│   ├── components/
│   │   ├── ExchangeRow.jsx
│   │   ├── MyBookRow.jsx
│   │   ├── Nav.jsx
│   │   └── SearchBookRow.jsx
│   ├── routes/
│   │   ├── Exchange.jsx
│   │   ├── Login.jsx
│   │   ├── MyPage.jsx
│   │   ├── NotFound.jsx
│   │   ├── Register.jsx
│   │   ├── Root.jsx
│   │   └── Search.jsx
│   ├── stylesheets/
│   │   └── styles.scss
│   ├── App.jsx
│   └── index.js
├── public/
│   ├── index.html
│   └── styles.css
├── server/
│   ├── server.js
│   ├── routes/
│   │   └── api.js
│   ├── controllers/
│   │   ├── apiController.js
│   │   ├── cookieController.js
│   │   ├── dbController.js
│   │   └── userController.js
│   ├── models/
│   │   ├── Book Exchange 2.0_postgres_create.sql
│   │   └── booksModels.js
```

7. Hosting and Deployment

7.1 Deployment Environment

Technologies in Use:

- **Frontend**: React (react, react-dom, react-router-dom).
- **Backend**: Express.js.
- **Database**: PostgreSQL (pg).
- **Build Tools**: Webpack, Babel.
- **Styling**: Sass (sass, sass-loader), Bootstrap.
- **Testing**: Jest, Testing Library.
- **Others**: Concurrently (to run both backend and frontend in development).

7.2 Hosting:

- **Frontend**: Deploy to a static hosting service **AWS S3**.
- **Backend**: Deploy to **Node.js** hosting services: **AWS EC2**.
- **Database**: Use a managed database service like **AWS RDS** for PostgreSQL.

7.3 CI/CD Pipeline

Automate Builds and Deployments:

- Use GitHub Actions or the hosting platform's CI/CD pipelines to automate the deployment process.
- Trigger deployments on pushes to the main branch.

7.4 Testing and Monitoring

- **Testing**:
  - Run unit tests using Jest: npm run test.
  - Use tools like Postman or Swagger to test API endpoints.
- **Monitoring**:
  - Integrate logging (e.g., Winston, Morgan) to track application errors.
  - Use monitoring services like **New Relic**, **Sentry**, or **LogRocket** to observe performance and issues.

8. Testing Plan

8.1 Unit Testing

Focus: Testing individual components or functions in isolation.

What to Test:

- **Backend**:
  - Controllers (e.g., adding a book, verifying a user).
  - Services/Utility functions (e.g., hashing passwords, generating tokens).
  - Database queries (mock the database).
- **Frontend**:
  - React components (e.g., book listing, user profile page).
  - Form validation logic.
  - API calls (using mocks for backend endpoints).

Tools:

- **Backend**: Jest, Supertest (for HTTP requests).
- **Frontend**: Jest, React Testing Library.

8.2 Integration Testing

Focus: Testing how different modules work together.

What to Test:

- Backend endpoints connected to the database.
- Frontend interacting with backend APIs.
- End-to-end workflows (e.g., user registering, searching for a book, requesting it).

Tools:

- **Backend**: Supertest, Postman.
- **Frontend**: Cypress (end-to-end testing).
- **Database**: Mock PostgreSQL with tools like pg-mock or use a test database.

8.3 Security Testing

Focus: Identifying vulnerabilities in the application.

What to Test:

- **Authentication**:
  - Token validation.
  - Password hashing and storage (e.g., using bcrypt).
- **Authorization**:
  - Ensure users can only access their own data (e.g., /getMyOldBookList).
- **Input Validation**:
  - Prevent SQL injection and XSS.
- **Rate Limiting**:
  - Protect APIs from abuse or brute force attacks.

Tools:

- **Manual Testing**: Use Postman to simulate attacks.
- **Automated Testing**: OWASP ZAP, Burp Suite.
- **Code Analysis**: ESLint (static analysis), Snyk (dependency vulnerabilities).

8.4 Test Coverage

- Use a tool like Jest's coverage reporter to measure the percentage of code covered by tests.
- Set a minimum coverage threshold (e.g., 80% for statements, branches, functions, and lines).

9. Project Timeline & Milestones

- Stage 1: Basic Initial Development (2 months)
    - 3 User stories implementation: Done with basic implementation. It needs further enhancement tasks.
- Stage 2: Feature Development (1 month)
- Stage 3: Complete Launch (1 month)

10 Resource Allocation

- Developer Team: 2 Full Stack Developers (1 core developer, 1 with testing background), 1 UI/UX Designer, 1 DevOps Engineer.

---

12. Future Implementations

12.1. Enhanced User Features

Wishlist Feature:

- Allow users to create and manage a wishlist of books they want.
- Notify users when a book from their wishlist is available.

Rating and Reviews:

- Add a rating system for users and books.
- Allow users to leave reviews for book conditions or exchange experiences.

12.2. Community and Social Features

Book Clubs and Groups:

- Allow users to create or join book clubs and communities.
- Enable group discussions and book recommendations.

Social Media Integration:

- Allow users to share book listings or exchanges on platforms like Twitter, Facebook, or Instagram.

12.3 Mobile Application

- Build native mobile apps for Android and iOS using **React Native** or **Flutter**.
- Provide offline functionality for browsing books or managing exchanges.