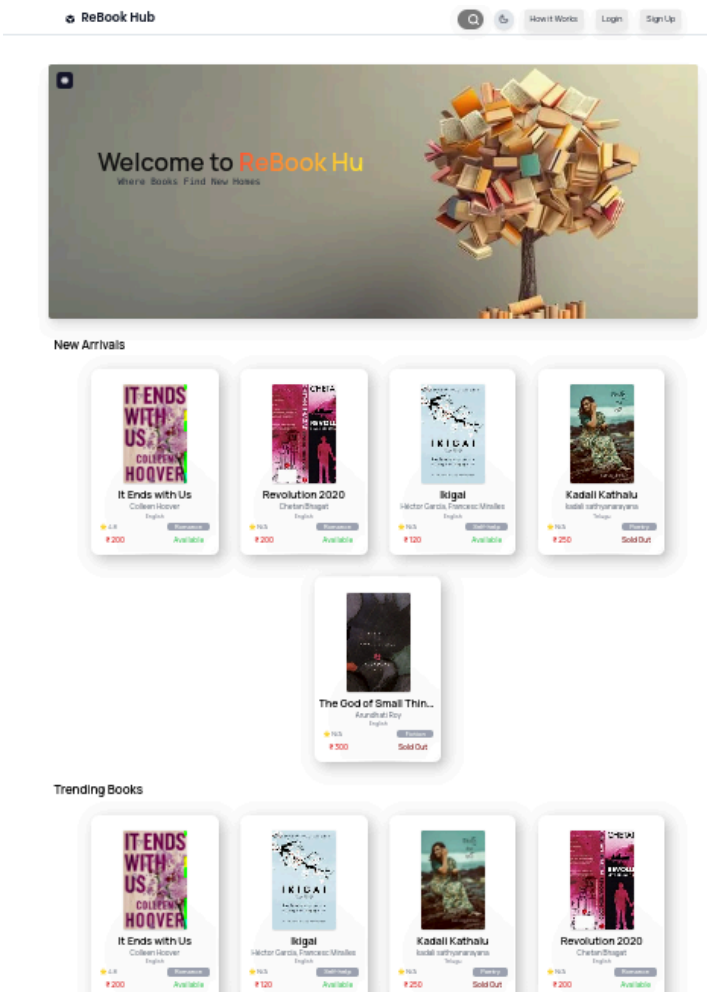


ReBook Hub

Sell What You’ve Read. Buy What You’ll Love.

Sammaiah Guguloth | 22-06-202

Project Description

ReBook Hub is a fully functional **book marketplace platform** that enables users to **buy, sell**, and **manage second-hand books** with ease. The platform provides features such as book listing, advanced search and filtering , order management and secure payment handling.

It is built using the **MERN stack**, which includes **MongoDB**, **ExpressJS**, **ReactJS**, and **NodeJS**, and leverages additional tools and libraries for a modern, performant experience.

ReBook Hub aims to provide:

- A seamless and user-friendly marketplace experience for students and readers to **buy and sell second-hand books**, making educational resources more affordable and accessible.
- A platform for individuals to list their pre-owned books, reach a wider audience, and **earn by sharing knowledge**, while contributing to a **sustainable reading ecosystem**.

In the following sections, we will cover the technical details of the platform, including :

1. System architecture: The high-level overview of the platform's components and diagrams of the architecture.
2. Front-end: The description of the front-end architecture, user interface design, features, and functionalities of the front-end, and frameworks, libraries, and tools used.
3. Back-end: The description of the back-end architecture, features and functionalities of the back-end, frameworks, libraries, tools used, and data models and database schema.
4. API Design: The description of the API design, list of API endpoints, their functionalities, and sample API requests and responses.
5. Deployment: The description of the deployment process, hosting environment and infrastructure, and deployment scripts and configuration.
6. Testing :
7. Future Enhancements: The list of potential future enhancements to the platform, explanation of how these enhancements would improve the platform, estimated timeline and priority for implementing these enhancements.

In summary **ReBook Hub** is a versatile and intuitive book marketplace platform designed to simplify the exchange of second-hand books. It offers users a seamless experience to **buy and sell pre-owned books**, making educational resources and leisure reading more affordable and sustainable. In the following sections, we will explore the **technical architecture and core functionalities** of the platform, providing a comprehensive overview of its features, tools, and implementation strategy.

System Architecture

The **ReBook Hub** marketplace platform consists of three main components: the **frontend**, the **backend**, and the **database**. The platform follows a **client-server architecture**, where the frontend acts as the client, handling user interactions and interface rendering, while the backend and database function as the server, managing business logic, **API** endpoints, authentication, and data persistence.

Frontend

The frontend of **ReBook Hub** is built using **ReactJS**, enabling dynamic and responsive user interfaces. Styling is handled using **Tailwind CSS**, which allows for a clean utility-based approach and includes built-in classes for **smooth transitions and animations** without relying heavily on external libraries. **Framer Motion** is used sparingly for enhanced animation effects on key UI components. The interface supports both **dark and light modes**, delivering a visually pleasing and accessible experience across devices. Communication with the backend is done via **RESTful APIs**.

Backend

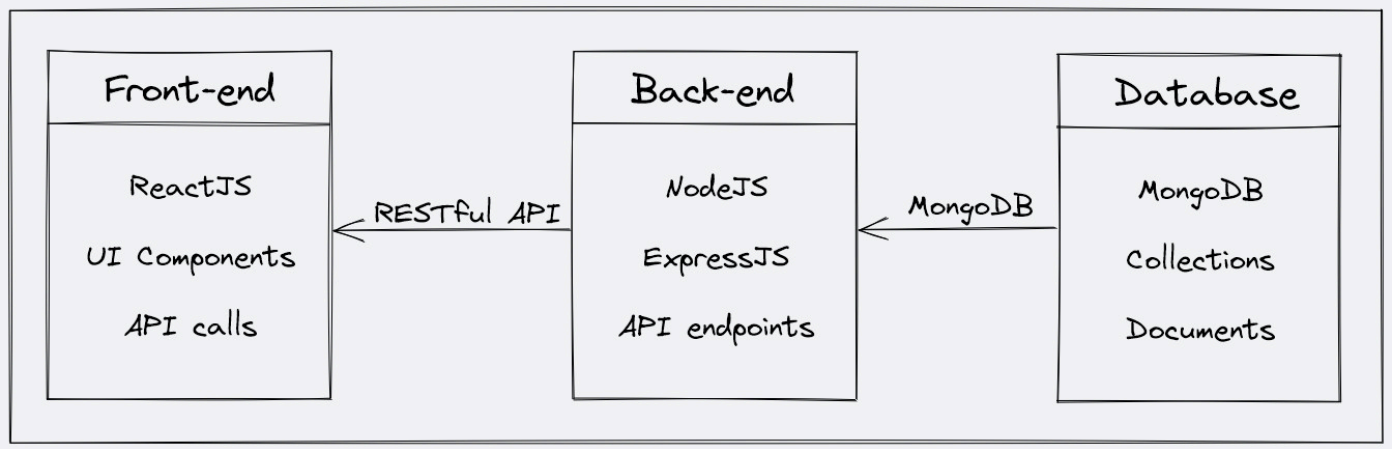
The **backend of ReBook Hub** is built using **NodeJS and ExpressJS**, powerful frameworks for creating scalable and robust server-side applications. It exposes a set of **RESTful APIs** that power core functionalities such as **user authentication**, **book listing and management and order handling**, and **secure payments**. The backend integrates with **Razorpay** to create and verify payment intents, holding payments until delivery is confirmed. It also uses **ImageKit** for fast and optimized image uploads and delivery. A mock **delivery system** is implemented, where sellers update shipment status and **mock webhooks** simulate delivery confirmation, triggering **automatic payment release** to the seller. All user data, order details, and book listings are securely managed and stored in the backend.

Database

The **database of ReBook Hub** is built using **MongoDB**, providing a flexible and scalable solution for storing platform data. It manages essential collections such as **users**, **books** (with cover and up to 5 real images), **orders**, **shipments**, **payments**, **OTP verifications**, **blacklisted tokens**, and **analytics**. This structure supports secure authentication, real-time order tracking, and efficient content management. Detailed model structures are described in later sections.

Architecture Diagram

Here is a high-level diagram that illustrates the architecture of the ReBook Hub.

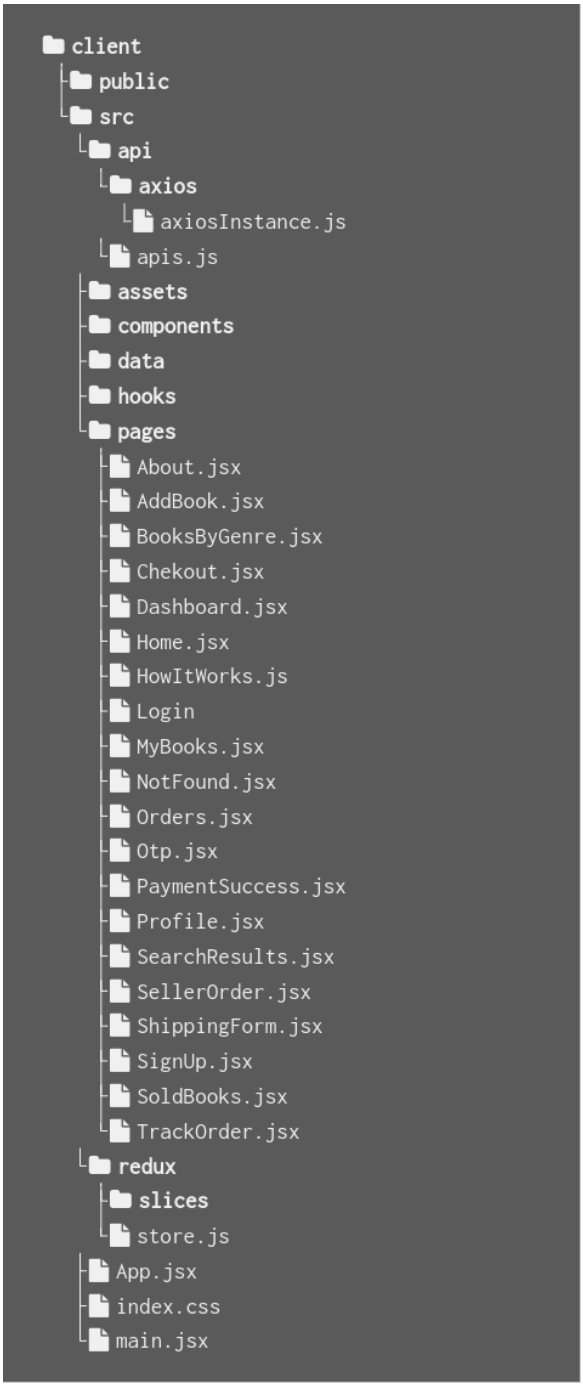

Front-end
ReactJS
UI Components
API calls
RESTful API
Back-end
NodeJS
ExpressJS
API endpoints
MongoDB
Database
MongoDB
Collections
Documents

Frontend

The **frontend** is the part of the platform that users directly interact with. It represents the **visual layer** of ReBook Hub — the layout, buttons, pages, and overall experience users engage with while browsing, buying, or selling books. The frontend of ReBook Hub is developed using **ReactJS** for component-based UI development and styled using **Tailwind CSS** for a clean, modern, and responsive design.

While no Figma file is used, the UI was crafted directly in code following best practices for usability and accessibility. You can view the complete frontend source code on GitHub by visiting:
👉 https://github.com/Sammaiah-Guguloth/ReBookHub-Frontend

Folder structure



Frontend Pages of ReBook Hub

The frontend of **ReBook Hub** provides a user-centric experience designed to make the process of buying and selling second-hand books seamless and intuitive. Built using **ReactJS** with **Tailwind CSS** and smooth in-app transitions, the platform includes all essential pages a modern marketplace needs.

For Buyers & General Users

1. **Homepage:** Features trending books, genres, recommendations, and new arrivals.
2. **Singup page:** New users can register using a clean sign-up form.
3. **Email Verification:** OTP-based verification flow to secure accounts.
4. **Login Page:** Allows users to access their dashboard securely.
5. **Dashboard Overview:** Centralized page to manage **books**, **orders**, and **profile**.
6. **Search Page:** Powerful search and filter system to find books quickly.
7. **Book Details Modal:** Displays price, condition, description, and real images.
8. **Checkout Page:** Allows users to view selected books and proceed to checkout.
9. **Secure Payment Page:** Handles payments via Razorpay with UPI, cards, etc.
10. **Order Tracking Page:** Shows current status and delivery updates.
11. **Profile Page:** View and edit user info, address, and preferences.

For Sellers

1. **Add Book Page:** Form to upload book details, pricing, cover image, and up to 5 real photos.
2. **Seller Confirmation Page:** Allows sellers to accept/decline incoming orders.
3. **Shipping Form Page:** Lets sellers provide tracking details after confirming shipment.
4. **Sales Overview Page:** Dashboard to monitor listed books, sales, and revenue.

For Admins(Future Scope)

1. **Platform Overview Dashboard:** Metrics on total users, orders, and revenue.
2. **User and Book Management:** Ability to moderate listings and user activity.
3. **Refund Oversight Page:** Manual control of dispute and refund cases if needed.

These pages work together to deliver a fluid, responsive experience across both desktop and mobile devices. Built using **ReactJS**, styled with **Tailwind CSS**, and lightly animated using vanilla Tailwind utilities and **Framer Motion**, the UI feels both modern and performant.

👉 *To see the actual flow of the platform in action, visit the live "How It Works" walkthrough here:* **ReBook Hub – How It Works**

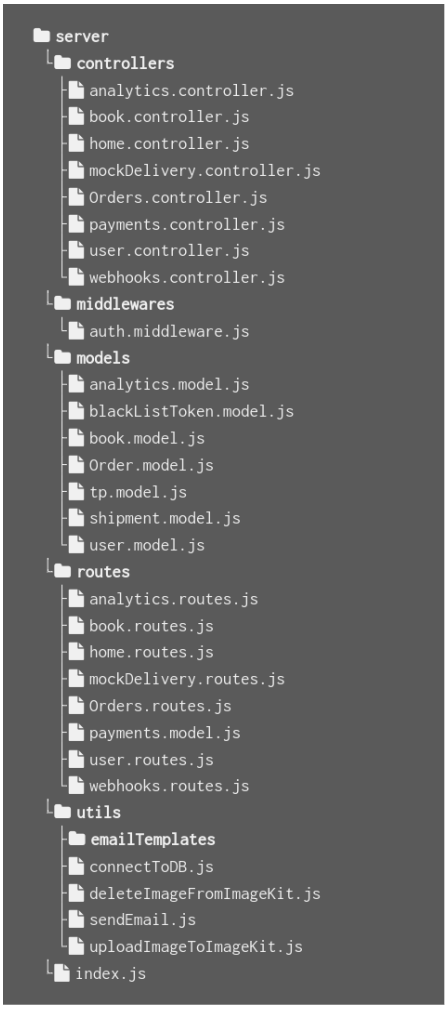
Backend

Architecture Overview

ReBook Hub adopts a **monolithic architecture** using **Node.js** and **Express.js** for backend development and **MongoDB** as its primary NoSQL database. This unified structure ensures streamlined control, enhanced security, and better performance. The backend exposes RESTful APIs for handling user authentication, book listings, payments, and order management.

To simulate real-world logistics, ReBook Hub integrates **mocked shipment APIs** that mimic services like **AfterShip**. These mock APIs help in development and testing by replicating delivery tracking flows and can be **seamlessly replaced with actual shipping APIs** in production. Similarly, a **mock webhook system** is in place to simulate delivery confirmation events—currently triggered manually—which in real deployment can be registered directly with third-party logistics providers. These webhooks handle **automated payment releases**, **order status updates**, and **buyer-seller notifications**, providing a scalable path toward full logistics integration.

Folder structure



Features and Functionalities of the Back-end :

The backend of **ReBook Hub** provides a comprehensive set of features essential for managing a second-hand book marketplace. Key functionalities include:

1. **User Authentication and Authorization** : Users can register and log in securely using email and password. The platform supports **OTP-based email verification** .JWT tokens are used for session handling, and blacklisted tokens ensure security during logout.
2. **Book Listing and Management:** Sellers can create, update, and delete book listings. Each listing includes a cover image, up to five real images, price, condition, and location details. Buyers can view and search books with category and filter options.
3. **Payment Integration:** The platform integrates **Razorpay** for secure payment processing. Payments are held in admin account (can be replaced with escrow services) and released to sellers after delivery confirmation. Failed or canceled orders are auto-refunded. Cron jobs can be added for future scope.
4. **Cloud-Based Image Handling: ImageKit.io** is used to manage all uploaded images. It provides **optimized delivery**, **real-time transformation**, and secure storage for user profile images and book photos.
5. **Mocked Delivery and Webhooks** (Developer Mode): For development purposes, the platform includes **mock shipment APIs** that mimic real tracking systems like AfterShip. A **mock webhook system** is also implemented to simulate delivery events, which can trigger **payment release and notifications**. These can be easily swapped with real logistics APIs during production deployment.

Frameworks, Libraries, and Tools used:

The backend of **ReBook Hub** utilizes a modern set of libraries and tools to ensure robustness, scalability, and ease of development:

1. **Node.js** : Serves as the runtime environment for executing JavaScript on the server side.
2. **Express.js** : A minimal and flexible Node.js web application framework used to build **RESTful APIs** and handle **routing** efficiently.
3. **MongoDB + Mongoose** : MongoDB is used as the primary NoSQL database. **Mongoose** provides a schema-based solution to model data and interact with the database easily.
4. **Razorpay SDK** : Integrates **Razorpay** payment **gateway** for handling secure and reliable payment transactions.
5. **ImageKit SDK** : Manages image uploading, transformation, optimization, and delivery via **ImageKit.io**, ensuring fast **media** handling.
6. **JWT** (jsonwebtoken): Used for user **authentication and authorization**, enabling secure session management via tokens.
7. **bcrypt** : Provides password hashing functionality to securely store and compare user passwords.
8. **express-validator** : Adds robust validation for incoming request data, ensuring reliability and security of inputs.
9. **express-fileupload** : Simplifies the handling of file uploads, especially for book images and profile pictures.

10. **nodemailer** : Enables sending of transactional emails such as **OTP verification**, **order notifications**, and **shipment updates**.
11. **cookie-parser** : Parses cookies attached to the client request object, useful for session and token handling.
12. **cors** : Configures Cross-Origin Resource Sharing (CORS) to allow frontend-backend communication securely during development and production.
13. **dotenv** : Loads environment variables from a .env file, keeping sensitive credentials like database URIs and API keys out of the codebase.
14. **nodemon** : A development utility that automatically restarts the server when file changes are detected, improving development workflow.

Data Models and Database Schema:

The backend of **ReBook Hub** employs a well-structured set of data models using **Mongoose** to define and interact with the MongoDB database. Each schema is tailored to handle specific responsibilities within the marketplace ecosystem. The key models include:

1. **User Schema** : Stores essential user information such as name, email, password (hashed), phone number, profile image, address, verification status, and role (buyer/seller). It also supports OTP verification and password reset.
2. **Book Schema** : Contains data for each book listing including title, description, category, condition, price, a cover image, and up to five real images uploaded by the seller. Also stores the seller's ID, availability status, and timestamps.
3. **Order Schema** : Captures all order-related information such as the buyer ID, seller ID, book ID, order status, payment status, and timestamps for order creation and updates. It connects closely with the shipment and payment flow.
4. **Shipment Schema** : Manages shipping-related details including tracking ID, current location, status, and delivery timeline. Used to simulate or integrate with real-world logistics services like AfterShip.
5. **OTP Schema** : Handles temporary storage of one-time passwords (OTPs) sent via email for user verification and password resets. It includes the target email, OTP code, and expiry time.
6. **Blacklisted Token Schema** : Stores JWT tokens that have been invalidated (e.g., after logout), enhancing backend security by preventing unauthorized reuse of tokens.
7. **Analytics Schema** : Tracks user activity like page visits, book views, clicks, and other relevant actions. This model supports insight gathering and potential future recommendation features.

Overall, the backend of ReBook Hub is designed to offer a **scalable, secure, and developer-friendly foundation**. With well-defined data models, integration-ready APIs, and a focus on real-world commerce features, the platform is prepared for both current functionality and future expansion.

API Design

The ReBook Hub platform's API is designed using the REST architectural style and built with Node.js and Express.js. It supports JSON for request and response formats, and uses standard HTTP methods such as `GET`, `POST`, `PUT`, and `DELETE`. Each endpoint is modular, clearly defined, and integrated with middleware for authentication, validation, and error handling.

Below is a sample list of API endpoints and their core functionalities:

- **user**
  - **/api/v1/user/register** (`POST`) – Create a new user account.
  - **/api/v1/user/send-otp** – Sends otp registering user.
  - **/api/v1/user/login** (`POST`) – Log in using credentials; returns a JWT token.
  - **/api/v1/user/logout** (`POST`) – Logs out the user by clearing cookies and blacklisting token.
  - **/api/v1/user/profile** (`GET`) – Returns the authenticated user's profile.
  - **/api/v1/user/update** (`PUT`) – Updates the details of the authenticated user's profile.
- **book**
  - **/api/v1/book/add** (`POST`) – Add's a book to authenticated user's profile.
  - **/api/v1/book/book-by-id/:bookId** (`GET`) – Get's the book by the id in the params
  - **/api/v1/book/my-books** (`GET`) – Get's all the uploaded books of authenticated user.
  - **/api/v1/book/all-books** (`GET`) – Get's all the books in the database.
  - **/api/v1/book/delete/:bookId** (`DELETE`) – Delete's the book
  - **/api/v1/book/books-by-genre/:genre** (`GET`) – Returns books of the provided genre.
  - **/api/v1/book/books-by-title/:title** (`GET`) – Returns books of the provided title.
  - **/api/v1/book/books-by-author/:author** (`GET`) - Returns books of the specified author.
  - **/api/v1/book/search** (`GET`) – Returns the books on the basis of provided **Query** .
- **payments**
  - **/api/v1/payments/create-order** (`POST`) – Instantiates a Razorpay order.
  - **/api/v1/payments/verify-payement** (`POST`) – Verifies the payment details provided by the frontend and updates the order status.
- **orders**
  - **/api/v1/orders/order-by-id/:orderId** (`GET`) – Returns the order.
  - **/api/v1/orders/accept-order** (`POST`) – Seller accepts the order , notifies the buyer , shipment form sent to seller
  - **/api/v1/orders/reject** (`POST`) – Seller rejects order , notifies the buyer and refund for buyer is instantiated.
  - **/api/v1/orders/order-shipped** (`POST`) – Seller ships the order and provides the shipping details , notifies the buyer , sends link and updates the status
  - **/api/v1/orders/orders-by-buyer/:userId** (`POST`) – Returns the user's orders(bought).
  - **/api/v1/orders/sold-orders/:userId** (`POST`) – Returns the user's orders(sold).
- **analytics**
  - **/api/v1/analytics/update/views/:bookId**(`PUT`) – Updates the view count of the book specified.

    - **/api/v1/analytics/update/cart/:bookId**(`PUT`) – Updates the added to cart count of the book specified.
    - **/api/v1/analytics/update/wishlist/:bookId**(`PUT`) – Updates the wishlisted count of the given bookId.
    - **/api/v1/analytics/update/attempted-purchase/:bookId**(`PUT`) – Updates the attempted purchases count of the provided book id.
- **home**
    - **/api/v1/home/default-feed** (`GET`) – Get's the feed based on analytics of the book.
- **mock-delivery**
    - **/api/v1/mock-delivery/add-shipment** (`POST`) – Add's a new shipment.
    - **/api/v1/mock-delivery/update-shipment** (`POST`) – Update's the shipment status and checkpoints on the specifed shipment.
    - **/api/v1/mock-delivery/track-order/:trackingId** (`GET`) – Returns the shipment details of the given trackingId.
- **webhooks**
    - **/api/v1/webhooks/handle-book-delivered** (`POST`) – Notifies the seller and buyer and instantiates a payout for the buyer.

In conclusion, The **REST API design** of the **ReBook Hub** platform plays a critical role in ensuring seamless communication between the frontend and backend systems. Each endpoint is thoughtfully structured to provide clarity, maintainability, and extensibility across various user flows—such as authentication, book listings, payments, and order tracking.

By adhering to **RESTful principles** and standard HTTP conventions, the API remains scalable and developer-friendly. The sample endpoints outlined above demonstrate how the backend processes data securely and efficiently. With this architecture in place, ReBook Hub is well-positioned to deliver a **smooth, secure, and responsive user experience** while maintaining long-term **stability and scalability**.

Deployment

The **ReBook Hub** platform is fully deployed and accessible online, with the frontend deployed using **Netfliy.com**, and backend hosted using **Render.com**, a modern cloud platform for hosting full-stack applications. Render provides automated deployment from Git repositories, built-in SSL, and scalability, making it an ideal choice for hosting production-ready web apps.

- 🔹 **Frontend:** Deployed on Render and accessible at: https://rebookhub1.netlify.app
- 🔹 **Backend (API):** Deployed separately and accessible at: https://rebookhub-server.onrender.com

The platform uses a **monorepo-like structure** with separate deployment pipelines for frontend and backend services, ensuring modularity and ease of scaling. Environment variables for secure credentials (e.g., Razorpay keys, MongoDB URI, ImageKit auth) are configured through Render's dashboard.

This setup ensures 24/7 availability, HTTPS-secured communication, and streamlined CI/CD for rapid iteration and deployment.

Future Enhancements

As **ReBook Hub** is a personal full-stack project built to demonstrate real-world development skills, there are several planned enhancements that reflect both technical curiosity and a commitment to continuous learning. These improvements are aimed at making the platform more production-ready and feature-complete:

1. **Real Shipment API Integration:** Currently using mocked shipment APIs for development. In the future, I plan to integrate real delivery services like **AfterShip** or **Shiprocket** for live tracking and automatic updates.
2. **Buyer-Seller Rating System:** To increase trust within the platform, a feedback mechanism will allow buyers to rate sellers after delivery.
3. **Wishlist and Personalized Recommendations:** I aim to implement a wishlist feature and build a basic recommendation system using collected analytics data to suggest relevant books to users.
4. **In-App Chat Feature:** Adding a real-time messaging system using **Socket.IO** or **Firebase** will help buyers and sellers communicate directly regarding books and delivery.
5. **Admin Dashboard:** A simple admin interface will be added to manage user reports, book listings, disputes, and platform analytics.
6. **Seller Analytics Panel:** Will add data visualization tools using **Recharts** or **Chart.js** so sellers can track views, orders, and earnings from their dashboard.
7. **Referral Program and Loyalty Credits**: As an experiment in gamification, a system for referral links and reward credits will be explored.