NOVELLE

A Full-Stack MERN Bookstore Application

NM2024TMID00542

Salmaan Ahmed K N - 24A96D292A531EE93A34DED1F60CD998

Rahaman A - 8E27B033355B87C97B947EF7E2E1A720

Santheep S - 6CAF8B380586EC1A0CA451E5597ACBA0

Suraj R - 2F2374139FE7867AAD547BBF5C52CBA7

Table of Contents

Introduction

The project titled **Novelle – A Comprehensive Bookstore Solution** is a simple online bookstore application developed using the **MERN stack**. While the concept of an online bookstore is not new, the focus of this project is to provide a reliable, user-friendly, and efficient platform for users to browse, purchase, and manage books.

Team Details

Our team consists of four members, each bringing their expertise to create a fully functional solution

Name	Role	Responsibilities
Salmaan Ahmed K N	Tech & Design Lead	Overseeing the entire planning, technical development, and design of the app, ensuring a smooth user experience.
Rahaman A	Backend Developer	built the database schema, routes, and implemented authentication tokens.
Santheep S	Frontend Developer	Developed the UI based on the provided design and integrated it with the backend.
Suraj R	System Tester	Testing the overall system, ensuring proper functionality across the platform.

Although **Novelle** is a simple bookstore app, the strength of this project lies in its solid technical foundation, ensuring reliable functionality, secure user authentication, and effective book management. The aim was to create a clean, straightforward platform that users can easily navigate while enjoying a smooth and secure shopping experience.

Project Overview

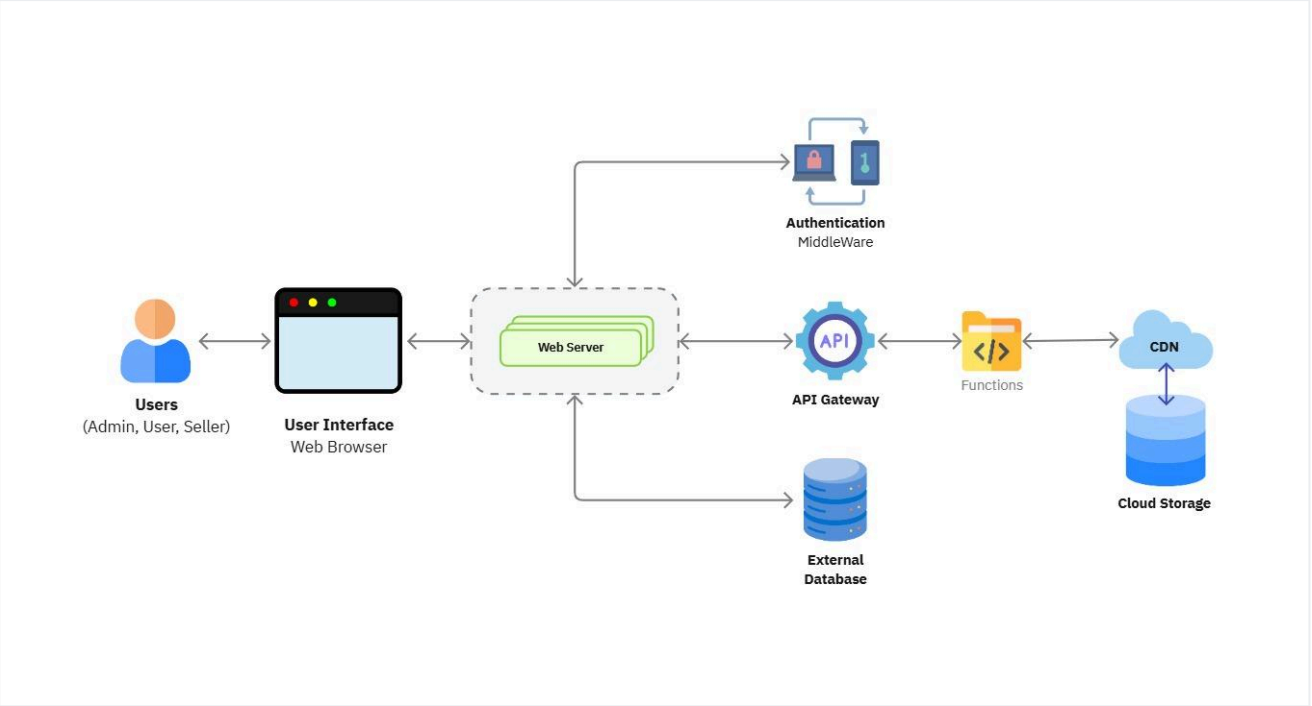
Purpose

The purpose of **Novelle** is to provide a simple yet efficient online bookstore platform catering to three main types of users: **Customers, Sellers**, and **Admins**. While the platform is centered around an intuitive and smooth user experience, it also offers specific functionalities to meet the needs of each user group. The overall goal is to create a reliable, secure, and user-friendly platform for browsing, purchasing, and managing books, ensuring an easy interaction between all users.

Features

Feature	Description
User Registration and Authentication	Allows users to create accounts, securely log in, and manage their profiles.
Book Browsing	Customers can explore a wide range of books, and view detailed information for each book.
Book Purchase	Customers can add books to their cart, select quantities, and securely complete purchases through the checkout process.
Order Management	Customers can view their order history, track active orders, and rate their purchases.
Seller Dashboard	Sellers can register on the platform, list new books, manage inventory, and fulfil customer orders.
Admin Dashboard	Admins can manage users (Customers and Sellers), monitor book listings, update platform settings, and generate reports to track overall performance and activities.
Secure Transactions	All transactions, whether by Customers or Sellers, are securely processed to ensure safety and privacy for all parties involved.

Architecture



High-Level Architecture the Book Store Web Application

Component	Technology / Tool	Description
Frontend	React, Vite	The frontend is developed with **React** for building the user interface, and **Vite** is used as the build tool and development server for fast, efficient development. The design is custom-built from scratch without using any external UI frameworks like Bootstrap or Material UI.
Backend	Node.js, Express.js	The backend is built using **Node.js** with **Express.js** for handling API requests, user authentication, and server-side logic.
Database	MongoDB	**MongoDB** is used for storing all data, ensuring flexibility and scalability for the app's needs.

Setup Instructions

Prerequisites

- Node.js (version 14 or higher)
- MongoDB (preferred MongoDB Atlas)
- Git (for cloning the repository)

Installation Steps

1. Clone the Repository

```
git clone https://github.com/salmaanahmeddd/novelle-book-store-nm
cd <project_folder>
```

2. Install Backend Dependencies: Navigate to the Backend folder and run

```
cd Backend
npm install
```

3. Install Frontend Dependencies: Navigate to the Frontend folder and run

```
cd Frontend
npm install
```

4. Set Up Environment Variables:

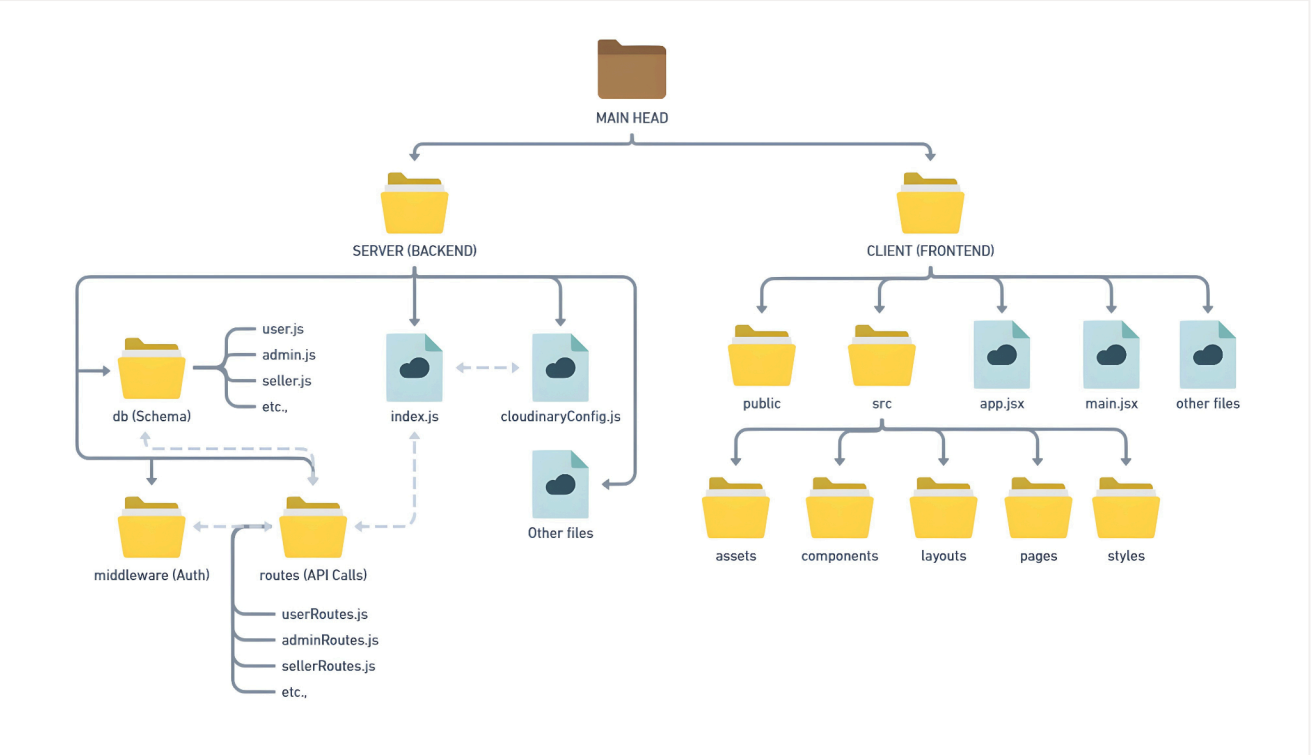
In the Backend folder, create a .env file and add the following variables (adjust values as needed):

```
MONGO_URI=<your_mongodb_connection_string>
JWT_SECRET=<your_jwt_secret>
CLOUDINARY_URL=<your_cloudinary_url>
PORT=5000
```

For Frontend, create a .env file in the Frontend folder and add the following:

```
VITE_APP_API_URL=http://localhost:5000
```

Folder Structure



Client (Frontend)

The frontend is built with Vite and React, organised to maintain a clean structure.

Folder	Description
components	Contains reusable UI components, like the header, login popups, and book lists.
layouts	Holds layout files for different user roles, such as the admin dashboard layout.
pages	Main role-based pages (e.g., Home, Admin).
styles	Holds CSS files for custom styling without using frameworks.

Server (Backend)

The backend is built with Node.js and Express and is structured to support scalability and easy management of API endpoints.

Folder	Description
db	Contains schema files for database collections..
middleware	Houses middleware files for token verification and other security functions.
routes	Holds route files for different entities (e.g., users, books, orders).

Running the Application

1. Run the Backend Server: In the Backend folder, run:

```
cd Backend
npm start
```

This will start the backend server at http://localhost:5000 (or your defined port).

2. Run the Frontend Development Server: In the Frontend folder, run:

```
cd Frontend
npm run dev
```

This will start the React app on http://localhost:5173

API Documentation

This document provides details of the API endpoints exposed by the backend of the Novelle Bookstore. The API supports operations for Users, Sellers, and Admin roles.

TEST THE APIs HERE: https://postman.com

Base URL

Type	Description
Production	https://novelle.onrender.com (Initial load may take 50s on the free server plan)
Local	http://localhost:5000/ (or your localhost)

Customer Endpoints

Endpoint	Method	Description	Parameters
`/users/register`	POST	Register a new customer.	`name, email, password`
`/users/login`	POST	Log in an existing customer.	`email, password`
`/users/all`	GET	Retrieve a list of all customers.	None
`/users/profile`	GET	Retrieve the profile of a customer.	`email (Query Parameter)`

Use these credentials for user login:

```
{
  "email": "user@novelle.in",
  "password": "user"
}
```

Admin Endpoints

Endpoint	Method	Description	Parameters
`/admin/signup`	POST	Register a new admin.	`name, email, password`
`/admin/login`	POST	Log in an existing admin.	`email, password`
`/admin/check-auth`	GET	Check the authentication status of the logged-in admin.	None
`/admin/admins`	GET	Retrieve a list of all admins.	None
`/admin/profile`	GET	Retrieve the profile by their email.	`email (Query Parameter)`
`/admin/logout`	POST	Log out the admin by clearing the authentication cookie.	None

Use these credentials for admin login:

```
{
  "email": "admin@novelle.in",
  "password": "admin"
}
```

Seller Endpoints

Endpoint	Method	Description	Parameters
`/sellers/register`	POST	Register a new seller.	`name, email, password`
`/sellers/login`	POST	Log in a seller and get a JWT token.	`email, password`
`/sellers/all`	GET	Retrieve a list of all sellers.	None
`/seller/profile`	GET	Retrieve the profile by their email.	`email (Query Parameter)`

Use these credentials for seller login:

```
{
  "email": "seller@novelle.in",
  "password": "seller"
}
```

Books Endpoints

Endpoint	Method	Description	Parameters
`/books/add`	POST	Add a new book to the store with image upload to Cloudinary.	`title, author, genre, description, price,sellerID, itemImage (body, form-data)`
`/books/`	GET	Get all books or filter by genre/author.	`genre, author (query parameters)`
`/books/:id`	GET	Get a specific book by its ID.	`id (URL parameter)`
`/books/update/:id`	PUT	Update a specific book's details.	`id (URL parameter), title, author, genre, description, price, itemImage (body, form-data)`
`/books/delete/:id`	DELETE	Delete a book from the store.	`id (URL parameter)`

To add a book, first log in via the seller login endpoint to receive a Bearer token and seller ID, then send a POST request to the `/books/add` endpoint with the token in the Authorization header as Bearer `<your_token>` and include the book details in the request body as form-data.

Use these credentials for seller login:

```
{
  "email": "seller@novelle.in",
  "password": "seller"
}
```

Sample Output from the login POST:

```
{
    "token":
"eyJhbGciOiJIUzI1ZXJJZCI6IjY3MzcwOTY0YmJhZDRiNDM3NGRjNjM2MSIsImlhdCI6MTczMTY2MD
EzNywiZXhwIjoxNzMxNjYzNzM3fQ.f4X02y6lGvVQT6QepTxhIJdXS54t-Ld2uEj5d3d24VY",
    "sellerId": "673709gregbbad4b4374dc6361"
}
```

Orders Endpoints

Endpoint	Method	Description	Parameters
`/orders/place`	POST	Place a new order for a book.	`bookId, userId, address, state, city, pincode (body)`
`/orders/`	GET	Retrieve all orders with populated details.	None
`/orders/:id`	GET	Retrieve a specific order by its ID.	`id (URL parameter)`
`/orders/delete/:id`	DELETE	Delete an order by its ID.	`id (URL parameter)`

To place an order, first send a POST request to `https://novelle.onrender.com/books/` to get the book ID by choosing a book. Then, log in as a user to obtain the user ID and a token in the response. Use the user ID in the order request input along with the book ID.

Use these credentials for userlogin:

```
{
  "email": "user@novelle.in",
  "password": "user"
}
```

Sample Output from the login POST:

```
{
    "token":
"eyJhbGciOiJIUzI1ZXJJZCI6IjY3MzcwOTY0YmJhZDRiNDM3NGRjNjM2MSIsImlhdCI6MTczMTY2MD
EzNywiZXhwIjoxNzMxNjYzNzM3fQ.f4X02y6lGvVQT6QepTxhIJdXS54t-Ld2uEj5d3d24VY",
    "userID": "673709gregbbad4b4374dc6361"
}
```

Sample Body

```
{
  "bookId": "62a2f7f3f1a7b3c0e1234567",
  "userId": "62a2f7f3f1a7b3c0e7654321",
  "address": "123 Main St, Apt 4B",
  "state": "California",
  "city": "Los Angeles",
  "pincode": "90001"
}
```

Authentication

In the **Novelle** project, user authentication and authorization are implemented securely using **JWT (JSON Web Tokens)** for managing login sessions across different user roles. The system ensures that only authorized users can access protected resources based on their assigned roles: **Admin**, **User**, or **Seller**.

Key Features of Authentication

1. JWT Tokens:

- Upon successful login, a JWT is generated for the user containing their unique identifier (`userId`, `adminId`, or `sellerId`) and role.
- The token is set as an **HTTP-only cookie** for secure transmission and stored on the client for subsequent authenticated requests.

2. Role-Based Access Control (RBAC):

- Access to certain endpoints is restricted based on user roles (e.g., Admin, User, Seller).
- Middleware such as `authorizeRole` ensures that users with incorrect roles are denied access.

3. Token Validation:

- Every request to a protected route is intercepted by the `verifyToken` middleware.
- The middleware extracts and validates the JWT from the request header or cookies.
- If valid, the token's payload (user ID, role) is attached to the request for further processing.

4. Security Practices:

- Passwords are hashed using **bcrypt** before storage in the database.
- Tokens are signed with a secret key stored in environment variables (`JWT_SECRET`).

Authentication Flow

1. Registration:

- Users, admins, and sellers can register using their respective endpoints:
  - `/users/register`
  - `/admin/signup`
  - `/sellers/register`
- Passwords are hashed, and upon successful creation, users receive a confirmation response.

2. Login:

- Each user role has a dedicated login endpoint:
  - `/users/login`
  - `/admin/login`
  - `/sellers/login`
- Valid credentials result in a JWT being issued and set as a cookie.

3. Token Verification:

- Middleware (`verifyToken`) checks the validity of the token on every protected route.
- Decoded information from the token is used to authorize user actions.

4. Role Authorization:

- Specific routes implement authorizeRole to restrict access based on the user role:
  - Admin-only routes: `/admin/admins, /admin/dashboard`
  - Seller-specific routes: `/sellers/all, /seller/dashboard`
  - User-specific routes: `/user/dashboard`

5. Logout:

- Logging out involves clearing the JWT cookie:
  - `/users/logout`
  - `/admin/logout`
  - `/sellers/logout`

Endpoints Overview

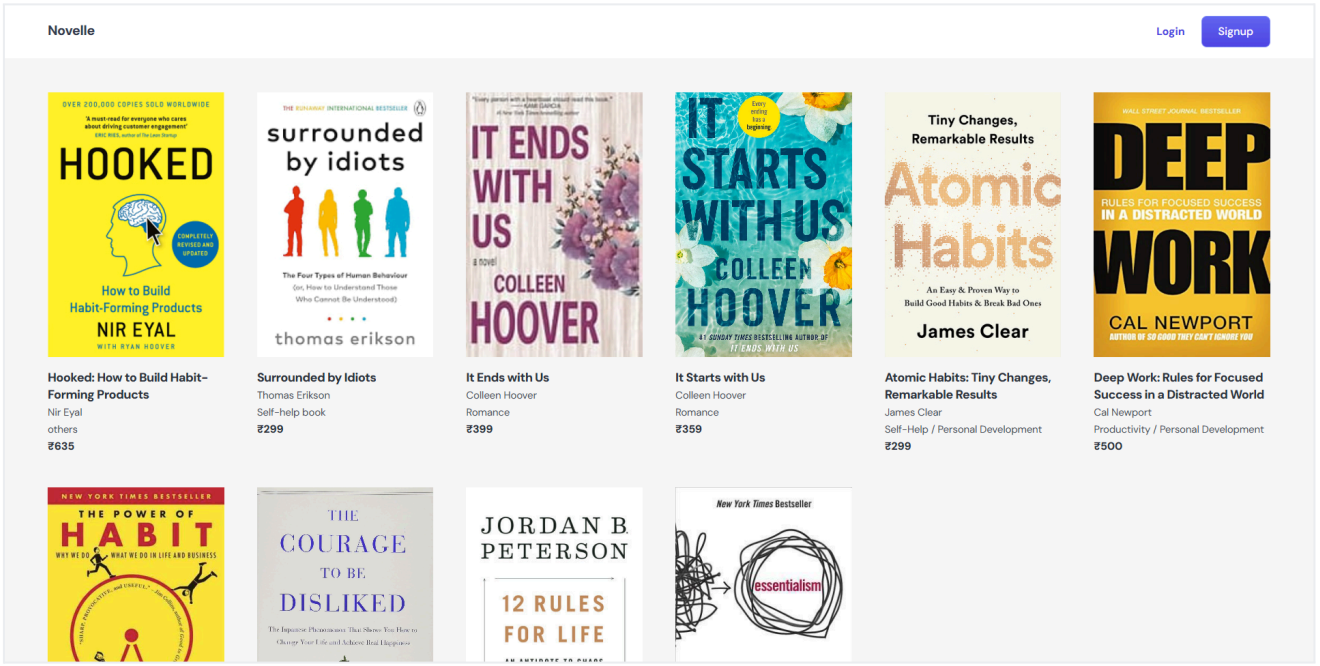
Endpoint	Method	Description	Role
`/users/register`	POST	Registers a new user.	`User`
`/users/login`	POST	Logs in an existing user.	`User`
`/admin/signup`	POST	Registers a new admin.	`Admin`
`/admin/login`	POST	Logs in an existing admin.	`Admin`
`/sellers/register`	POST	Registers a new seller.	`Seller`
`/sellers/login`	POST	Logs in an existing seller.	`Seller`
`/check-auth`	GET	Verifies if a token is valid for any role.	`All roles`

Common Error Responses

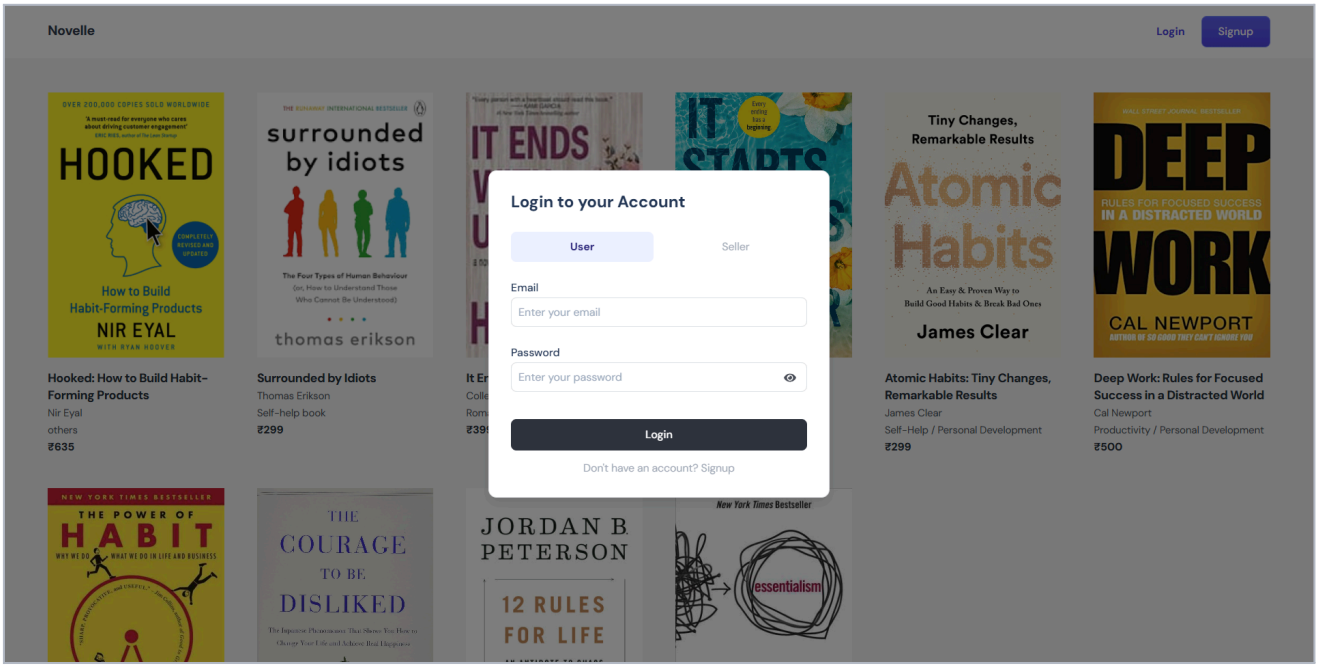
- **401 Unauthorized**: Invalid or expired token.
- **403 Forbidden**: Access denied due to insufficient role permissions.
- **400 Bad Request**: Missing or invalid input data.

For saving and managing image data in the **Novelle** project, we used **Cloudinary**. Cloudinary enables efficient and secure image uploads, storage, and retrieval. When a seller adds a book with an image, the image is uploaded to Cloudinary via its API, and the secure URL is stored in the database for later use.

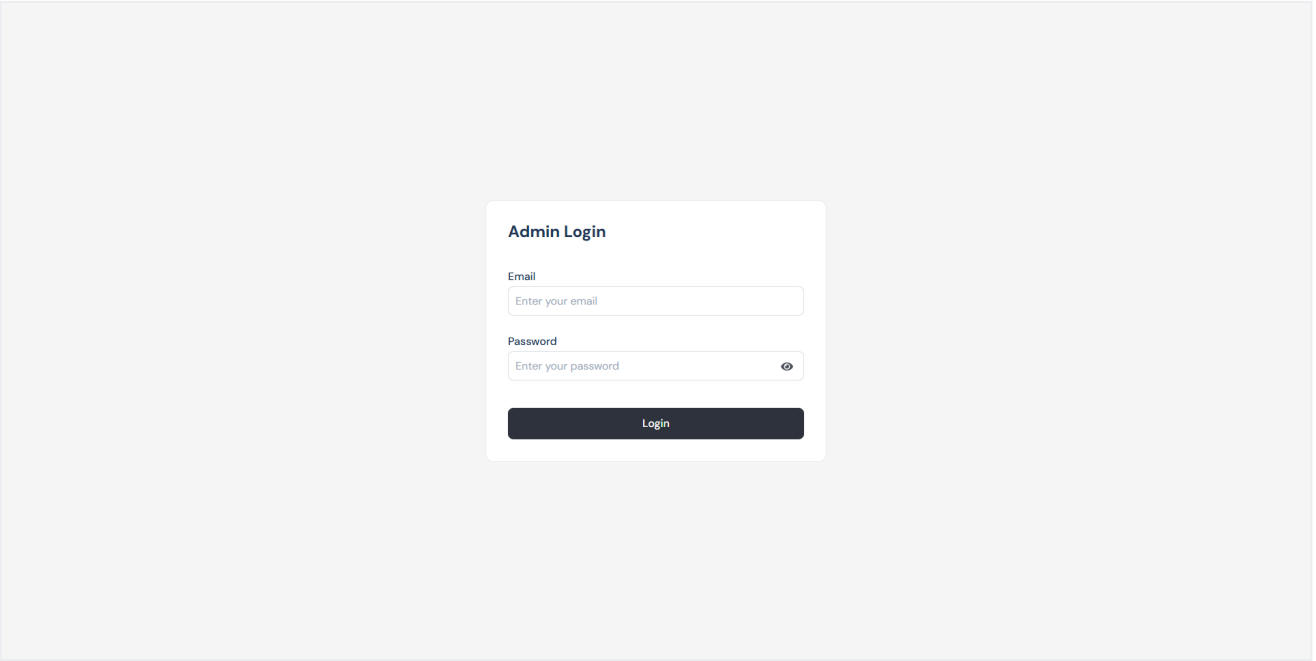
User Interface



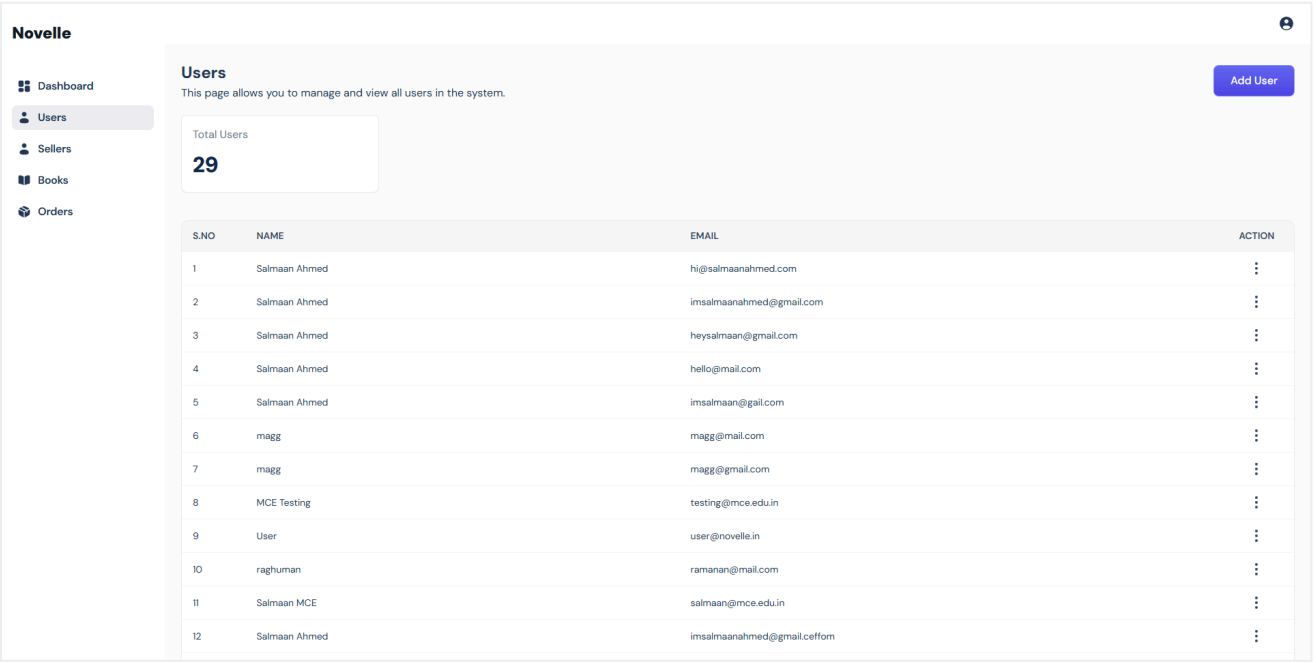
Home Page - User/Seller



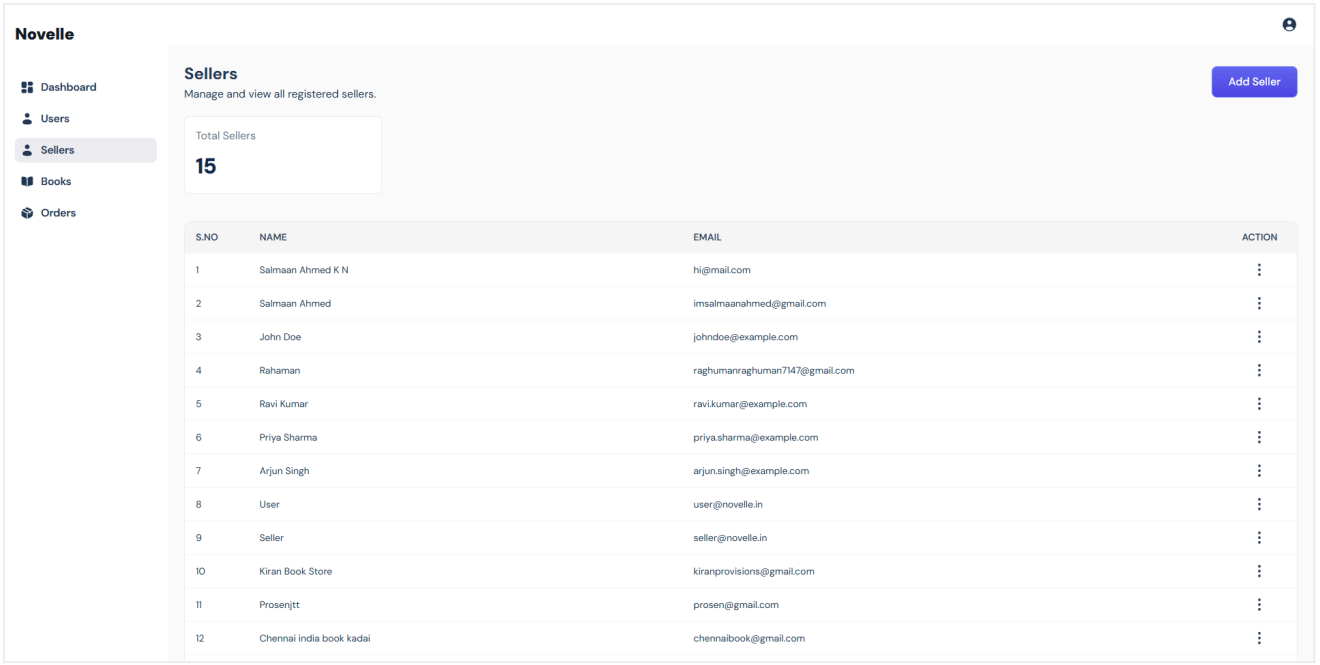
Login Modal - User/Seller



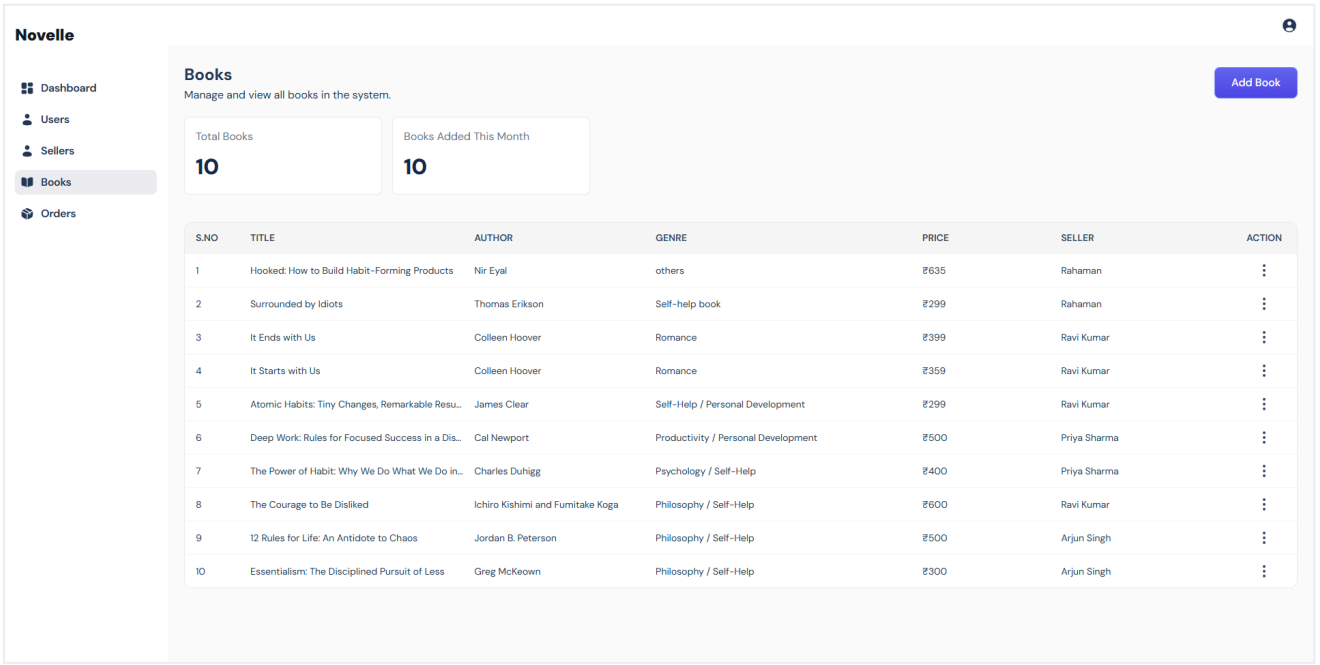
Admin Login – Admin



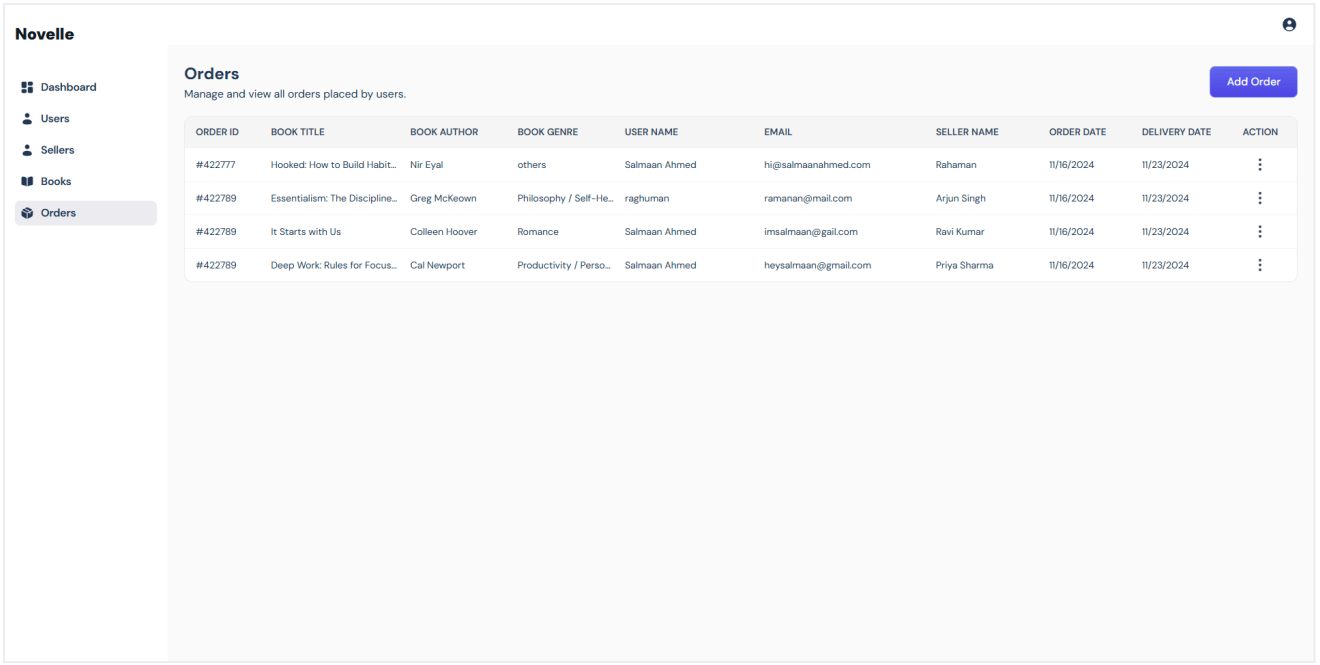
Users Details – Admin



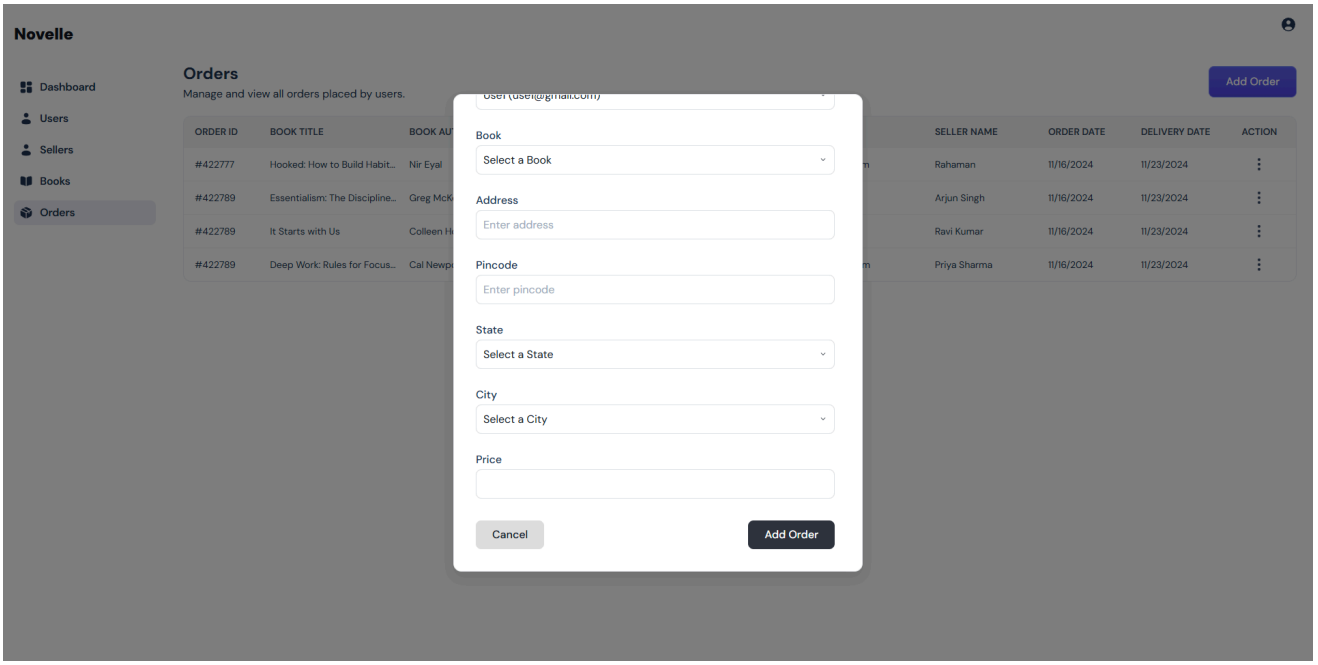
Sellers Details - Admin



Books Details - Admin



Order Details - Admin



Add Order Details - Admin

Testing

For testing, no technical-based testing was conducted during the development phase. Instead, we focused on basic system testing to ensure that the core functionalities, such as user login, order placement, and data flow between the front end and back end, were working as intended. Additionally, API endpoints were tested using Postman to verify the expected responses and error handling. The user interface was manually tested across different devices and browsers to ensure compatibility and responsiveness.

Aspect	Tool/Technology Used	Purpose
Login Testing	Manual (Browser, Devices)	Verify login functionality for all roles (User, Admin, Seller) on multiple devices and browsers.
Functionality Testing	Manual (Browser, Devices)	Ensure all core features (order placement, wishlist management, etc.) are working as expected.
UI Testing	Manual (Browser, Devices)	Check if the UI is responsive, intuitive, and displays correctly across different screen sizes and browsers.
API Testing	Postman	Test API endpoints for expected behavior, validate parameters, and ensure correct status codes and error messages.

Demo Links

Client (Frontend)

The frontend is hosted on Vercel.com, which pulls requests from GitHub and is updated every time changes are pushed to the GitHub repository.

Page	Link	Description
User Home	https://novelle-store.vercel.app/	User homepage for browsing.
Admin Home	https://novelle-store.vercel.app/admin/	Admin dashboard for management.

Server (Backend)

The backend is hosted on Render.com, which also pulls requests from GitHub and is updated with every push to the repository.

Note: Since the backend is hosted on a free plan, it may take approximately 50 seconds to load the server after a long period of inactivity.

Page	Link	Description
Home	https://novelle.onrender.com/	Base endpoint for backend API.
Users	https://novelle.onrender.com/users/	List and manage users.
Admins	https://novelle.onrender.com/admin/admins	Manage admin users.
Sellers	https://novelle.onrender.com/sellers/	View and manage sellers.
Books	https://novelle.onrender.com/books/	List of books available.
Orders	https://novelle.onrender.com/orders/	View order details and statuses.

Known Issues

Issue	Description
Login Issue in Incognito	Users experience login issues in incognito mode, causing immediate logouts.
Role-based Access Control (RBAC) Conflicts	Occasional conflicts in RBAC, allowing unauthorized users to access admin-specific functions.
Mobile Responsiveness	UI issues on mobile screens, particularly with the book list display and order form layout.
Error Handling for Missing Fields	API endpoints lack clear error messages for missing or invalid fields, leading to vague responses.

Future Enhancements

Enhancements	Description
Payment Gateway Integration	Integrating a payment gateway for users to make payments directly on the platform for their orders.
Advanced AI-Based Search and Filtering	Implementing AI algorithms to suggest books based on user preferences and refine filtering options.
Order Tracking	Adding real-time order tracking to allow users to monitor the status of their orders, from placement to delivery.
User Reviews and Ratings	Enabling users to leave reviews and ratings for books they purchase to assist other customers.