Full Stack Development with MERN -- BOOKSTORE

Project Documentation

1. Introduction

Project Title: Bookstore App -- BookNest

Team Members:

Tanya Maheshwari 22BCE10169
Shreyansh Sharma 22BCE11026
Akshit Singh 22BCE10386
Neha Pal 22BCE10367

2. Project Overview

Purpose:

The Bookstore App is a full-stack web application designed to allow users to browse, add, update, and delete books. It also supports user registration and login functionalities, making it an ideal platform to manage a digital bookstore.

Features:

- User authentication and authorization
- Book listing and search functionality
- Add/update/delete books (CRUD operations)
- Responsive user interface
- Integration with MongoDB for persistent data storage

3. Architecture

Frontend:
Developed using React.js. The frontend interacts with the backend API to display dynamic data and handles user authentication, routing, and UI rendering.

Backend:
Implemented using Node.js and Express.js. The backend exposes RESTful API endpoints for CRUD operations and manages business logic.

Database:
MongoDB is used as the database to store user and book data. Mongoose is used for object modeling and schema validation.

4. Setup Instructions

Prerequisites:

- Node.js (v14 or higher)
- MongoDB (local or cloud instance)

Installation:

1. Clone our repository

   git clone <repository-url>

2. Navigate to the client and server directories

   cd client && npm install (to install required node modules)

   cd ../server

3. Install server dependencies

   Run the following command inside the `server` directory:

   npm install bcrypt cors dotenv express jsonwebtoken mongoose

4. Install development dependencies

   Still inside the `server` directory, install `nodemon` for development:

   npm install --save-dev nodemon

5. Set up environment variables

   Create a `.env` file in the `server` directory and add the following:

   MONGO_URI=your_mongo_connection_string

   PORT=5000

5. Folder Structure

Client:

Contains the React app organized as:

- src/components: Reusable components
- src/pages: Route-specific page components
- src/App.js: Main app structure and routing

Server:

Organized as:

- controllers/: Logic for handling API requests
- models/: Mongoose models for User and Book
- routes/: API routes
- middleware/: Auth and error-handling middleware
- server.js: Entry point for the Express server

6. Running the Application

1. Clone the repository

git clone <repository-url>

2. Navigate to the client and server directories and install dependencies

cd client && npm install

cd ../server && npm install

3. Set up environment variables in a .env file in the server directory

MONGO_URI=your_mongo_connection_string

PORT=5000

7. API Documentation

Book Routes:

- GET /api/books — Get all books
- POST /api/books — Add a new book
- PUT /api/books/:id — Update a book
- DELETE /api/books/:id — Delete a book

User Routes:

- POST /api/users/register — Register a new user
- POST /api/users/login — Login a user

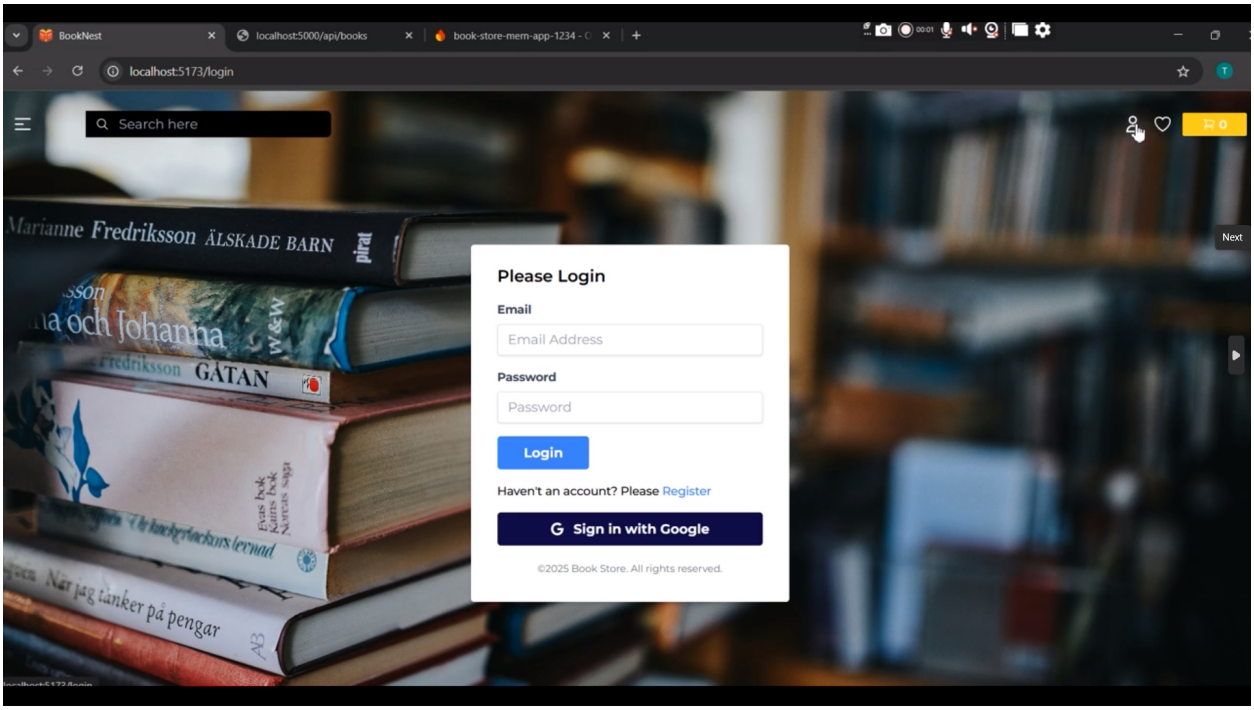
8. Authentication

The application uses **Firebase Authentication** to handle secure user login and registration. The authentication flow includes:

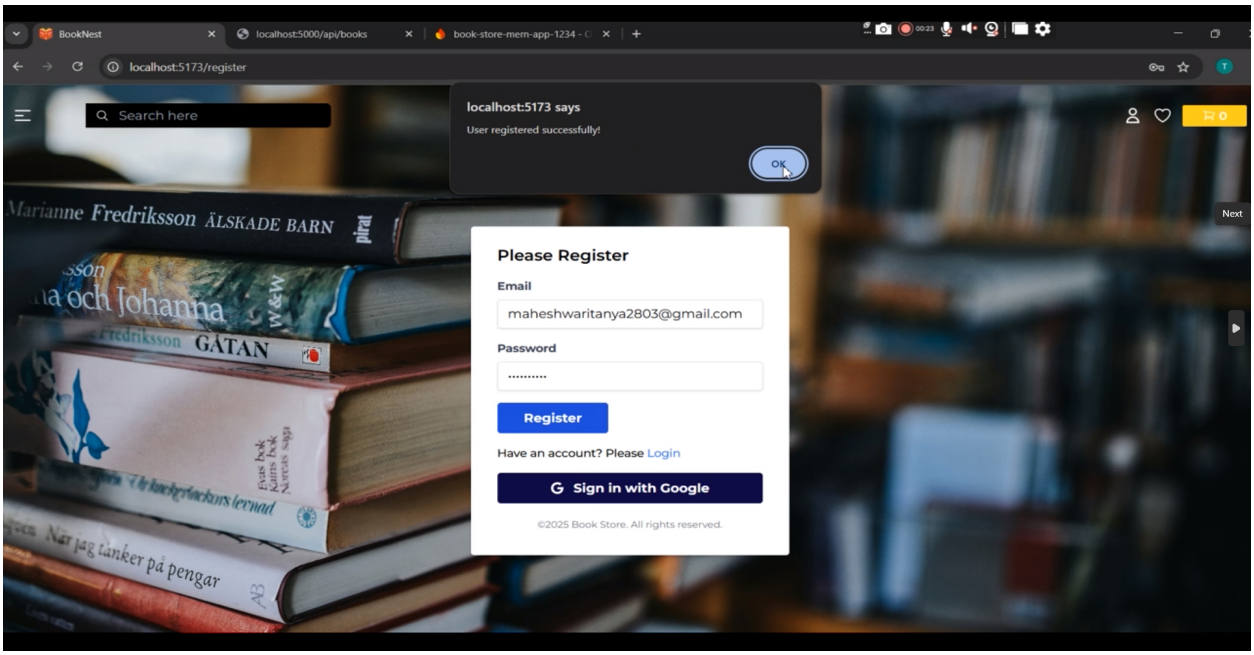
- **Email/Password Authentication** for users to register and log in.
- **Token-based sessions**, where Firebase issues an ID token upon successful authentication.
- **Route protection** in both frontend (React) and backend (Express) to ensure that only authenticated users can access certain resources.
- **Firebase Auth SDK** is used on the client side to manage authentication state and logout functionalities.

9. User Interface

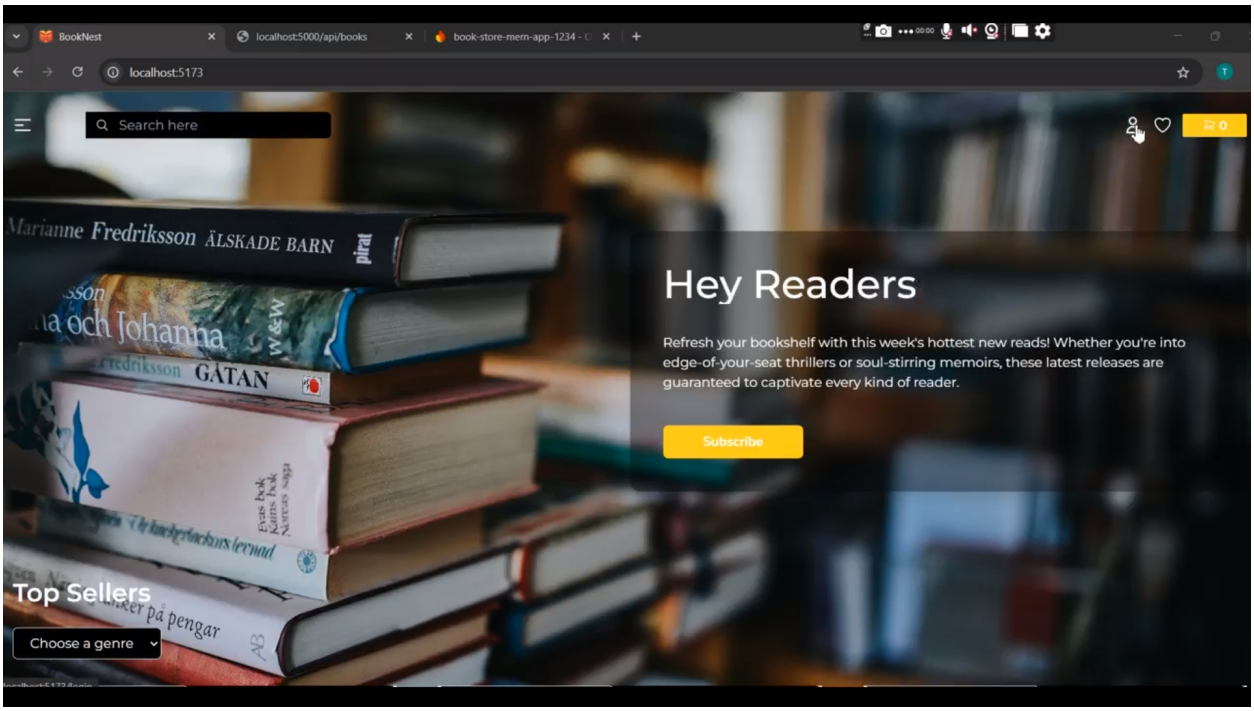
1. Initial Login/Registeration Page :



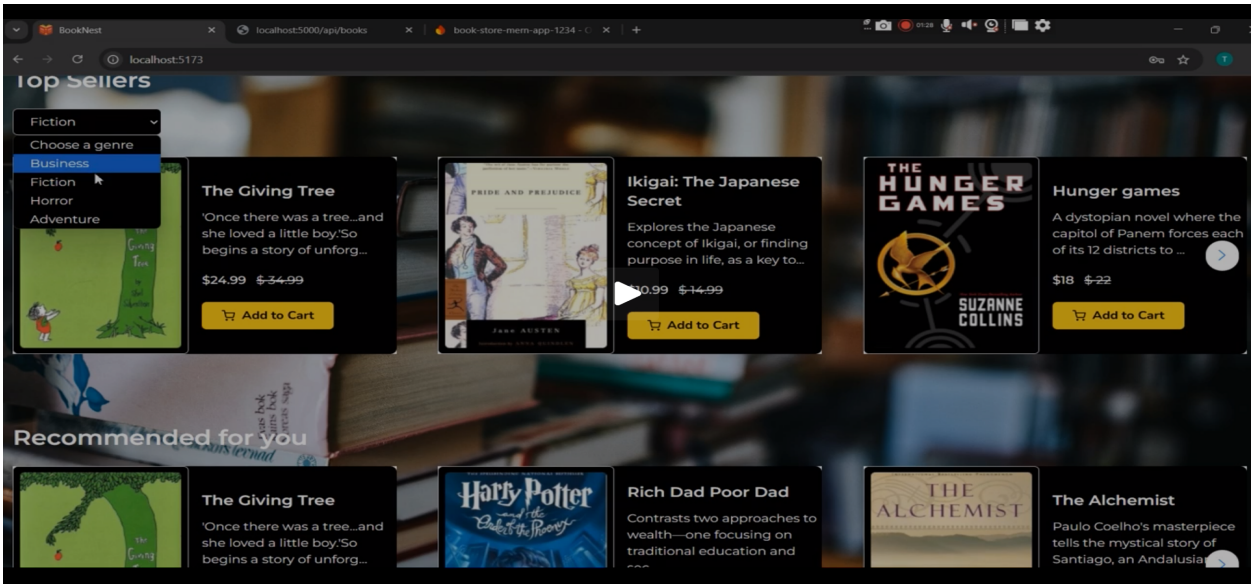
2. Upon Successful Registration:



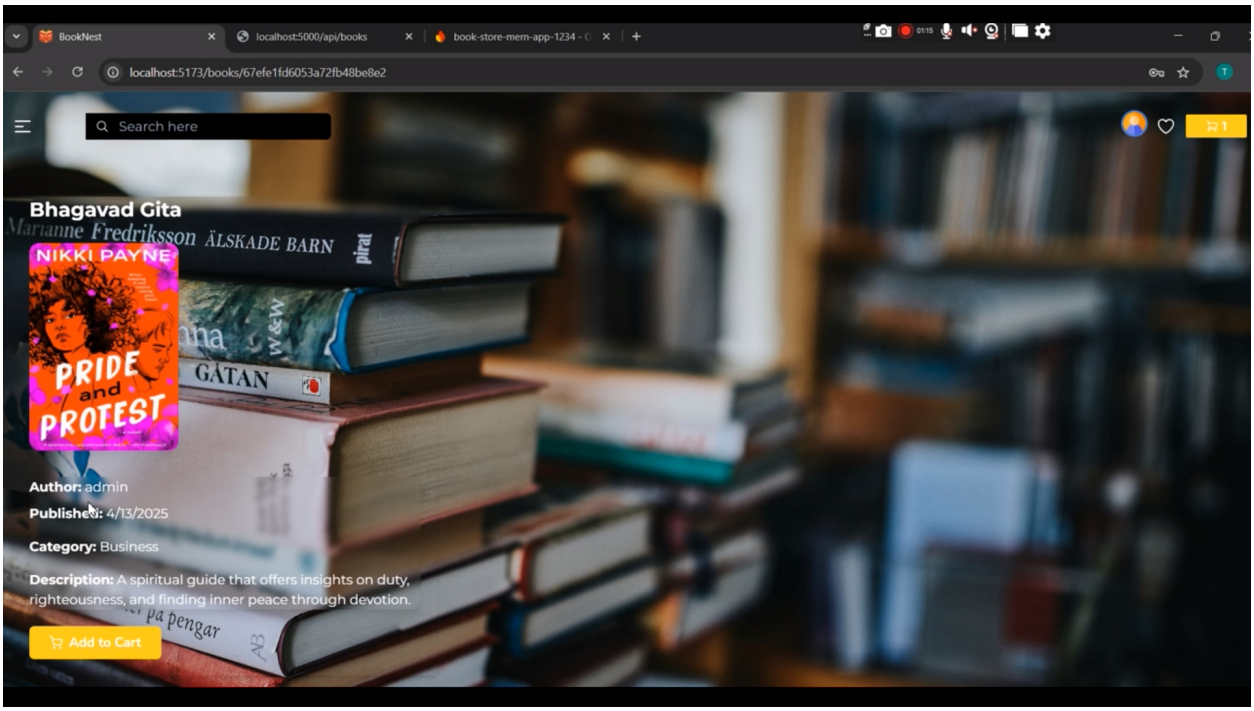
3. Home Page UI:



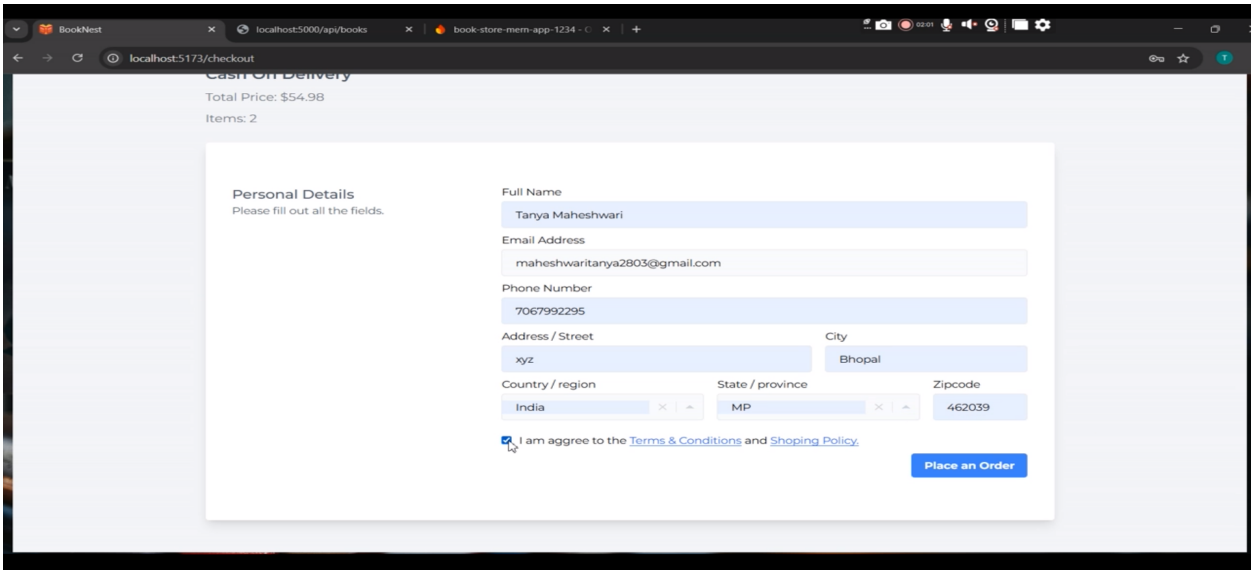
4. Book Library UI:



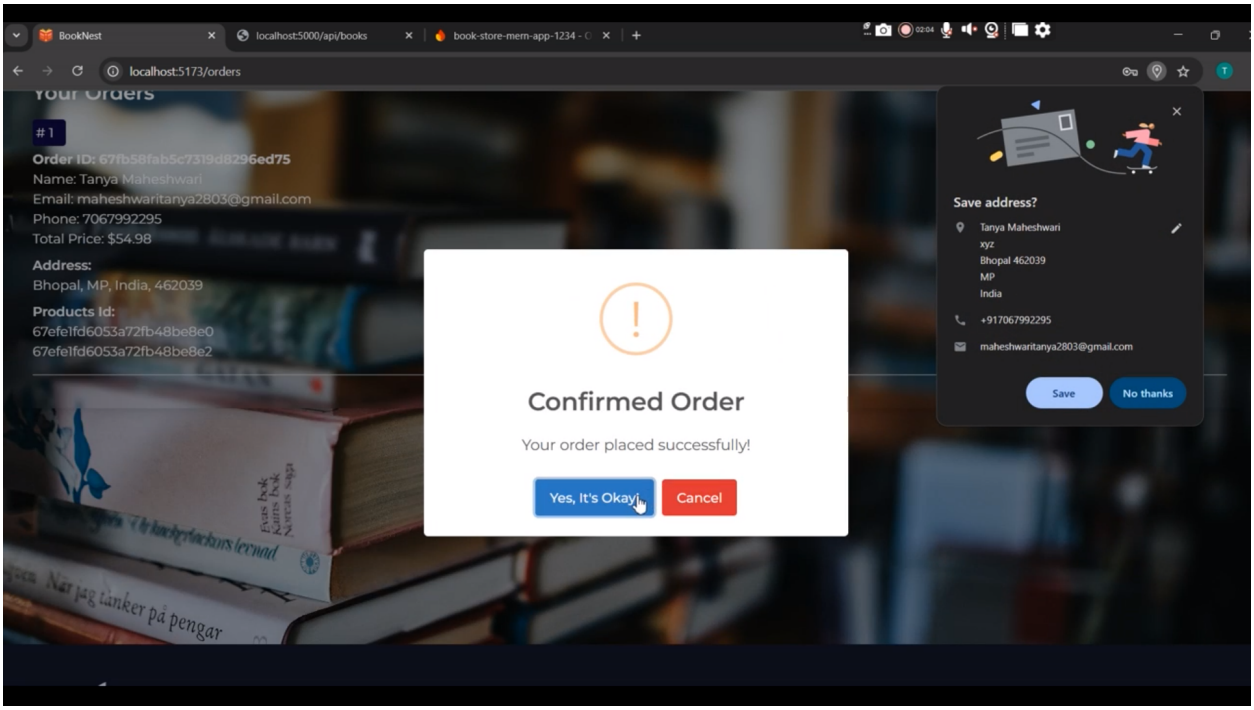
5. Choosing a book:



6. Ordering menu :



7. Confirming Order:



10. Testing

Testing Environment:

- **Frontend Stack**: React.js
- **Backend Stack**: Node.js / Express.js with MongoDB
- **Testing Tools**: Postman, Chrome DevTools
- URL/Location: localhost

Test Cases:

Test Case ID	Test Scenario	Test Steps	Expected Result	Actual Result	Pass/Fail
TC-001	User Registration	1. Navigate to the sign-up page2. Enter valid details3. Click 'Register'	Account should be created and user redirected to dashboard	Account created and redirected successfully	Pass
TC-002	User Login with Valid Credentials	1. Go to login page2. Enter valid email and password3. Click 'Login'	User should be logged in and directed to dashboard	Login successful and dashboard loaded	Pass
TC-003	Add Book to Cart	1. Search/select a book2. Click 'Add to Cart'	Book should appear in cart	Book added to cart and shown in cart page	Pass
TC-	View Order	1. Log in2. Go to	List of	Order	

004	History	'Order History' from dashboard	previous orders should be displayed	history displayed	Pass
TC-005	View Personalized Recommendations	1. Log in2. Visit 'Recommendations' page	Books based on user history should be displayed	Irrelevant books shown	Fail

Bug Tracking:

Bug ID	**Bug Description**	**Steps to Reproduce**	**Severity**	**Status**	**Additional Feedback**
BG-001	Irrelevant books in recommendations section	1. Log in with test user2. Go to 'Recommendations'3. Observe book list	Medium	Open	Books shown are not related to user's genre
BG-002	Cart not updating item quantity	1. Add book to cart2. Increase quantity3. Click 'Update Cart'	High	In Progress	Quantity stays the same after clicking update
BG-003	Broken link in footer	1. Scroll to footer2. Click ‘Contact’ link	Low	Open	Leads to 404 page

11. Screenshots or Demo

Demo Link:

https://drive.google.com/drive/folders/1P4LtrP0PdnP7KCFMG_gQbsOXvJIKGKRz?usp=sharing

12. Known Issues

- 🔄**Token Expiry Handling**: The current implementation does not automatically refresh expired Firebase ID tokens, which may log users out unexpectedly.
- **No Offline Support**: The app doesn't support offline login or cache user session locally.
- 🔁**Redundant Re-Renders**: Some components re-render on every auth state change, leading to unnecessary API calls.
- **No Role-Based Access Control (RBAC)**: Currently, there is no role separation (e.g., admin vs user), which limits permission-based feature access.
- ⚠**Minimal Error Feedback**: Error messages from Firebase are not user-friendly or customized.

13. Future Enhancements

 Role-Based Access Control (RBAC): Implement user roles and restrict access to specific pages based on role.

 Refresh Token Logic: Add automatic token refresh mechanism to ensure persistent login.

 Social Logins: Integrate Google and GitHub sign-in for quicker access.

 Toast Notifications: Improve UI feedback by adding styled toast alerts for auth errors and success.

 Mobile Responsiveness: Enhance the UI for mobile users with responsive design improvements.

 Two-Factor Authentication (2FA): Integrate an extra layer of security for user accounts.

 Unit Testing for Auth Flow: Add Jest tests to validate login, registration, and protected route behaviors.