**Full Stack Development with MERN**

**Project Documentation**

**1. Introduction**

* **Project Title:** Book Store
* **Team Members:**

Janarthanan. B\_413021104012 (Team leader) – **Frontend developer**

Praisy Bakkiyam. A\_413021104027 – **Backend developer**

Shivani. R\_413021104037-**Data analyst**

Karthika. J\_413021104017-**Backend developer**

**2. Project Overview**

* **Purpose:** The purpose of a bookstore goes beyond just selling books—it's a place that serves various cultural, educational, jhghdfgsocial functions.
* **Goals:**

### ****Providing Access to Knowledge and Learning****

* **Education**:

Bookstores are key in providing educational resources, from textbooks and academic guides to self-help books and non-fiction works. They support learning across various levels, from school to lifelong education.

### ****Supporting the Publishing Industry****

* **Selling Books**:

Bookstores are the retail arm of the book industry, offering books to consumers. They help support authors, publishers, and distributors by providing a marketplace for books to reach readers.

* **Promotion and Marketing**:

Bookstores often play an important role in promoting new releases, bestsellers, or niche genres, influencing trends in reading and the book market.

* **Features:**

### 1. ****Education and Knowledge Sharing****

* **Resource for Learning**: Bookstores offer a vast array of informational and educational materials, catering to various interests and fields of study.
* **Access to Diverse Perspectives**: They provide access to books that encourage critical thinking and expose readers to different cultures, philosophies, and viewpoints.

### ****2.Support for Authors and Publishers****

* **Platform for Emerging Authors**: Bookstores often promote new and local authors, giving them visibility and a chance to reach readers.
* **Collaboration with Publishers**: They work closely with publishers to bring new titles and genres to the market, contributing to the literary ecosystem.

1. **Architecture**

**Frontend Architecture:**

**Technology: ReactJs**

**Frameworks and Libraries**

* **JavaScript Frameworks**: Popular frameworks like React, Angular, and Vue.js help in building dynamic UIs and manage the application state effectively.
* **CSS Frameworks**: Frameworks such as Bootstrap, Tailwind CSS, and Foundation offer pre-built components and utilities to streamline styling.
* **State Management Libraries**: Libraries like Redux or MobX help manage application state in complex applications, providing a centralized store for state management.

**Component-Based Architecture**

* **Reusable Components**: Breaking down the UI into small, reusable components enhances maintainability and allows for easy updates.
* **Separation of Concerns**: Each component encapsulates its logic, styles, and template, promoting cleaner code and easier debugging.

**Routing**

* **Client-Side Routing**: Libraries like React Router or Vue Router manage navigation between different views without requiring a full page reload, creating a smoother experience.
* **Dynamic Routing**: Allows routes to be created dynamically based on application state or user interactions.

When designing a bookstore application architecture with user roles such as **User**, **Seller**, and **Admin**, we need to consider how each role interacts with the system, the components required for each role, and how the data flows through the application. Below is a detailed architecture description that incorporates these roles into a React-based application.

### Overview of Roles

**User Role Components**:

* + **User Dashboard**: Displays user-related information, order history, and recommendations.
  + **Book List**: Shows all available books for users.
  + **Book Detail**: Displays detailed information about a selected book.
  + **Cart**: Manages the user's cart and checkout process.

**Seller Role Components**:

* + **Seller Dashboard**: Displays seller-specific metrics, orders, and book management options.
  + **Add Book**: Form for sellers to add new books.
  + **Manage Books**: Allows sellers to edit or delete their books.
  + **Order History**: Displays orders received by the seller.

**Admin Role Components**:

* + **Admin Dashboard**: Overview of application metrics (e.g., total users, total sales).
  + **Manage Users**: Admin functionality to manage user accounts.
  + **Manage Books**: Admin functionality to manage all books in the application.
  + **Reports**: View sales reports and analytics.

### Set Up Your Development Environment

**Install Node.js and npm**: Make sure you have Node.js installed. This includes npm (Node Package Manager), which you’ll use to install packages.

**Create a New React App**:

bash

Copy code

npx create-react-app book storecd bookstore

**Book List Component**: In components/BookList.js, display the list of books.

**Book Component**: In components/Book.js, create a component to display individual book details.

**Add Book Component**: In components/AddBook.js, create a form to add new books

### Combine Everything in App.js

1. **Update App.js** to use the context and components.

. 2. Run Your Application: npm start

### Enhance Your Project

You can enhance your bookstore project with additional features, such as:

* **Styling**: Use CSS or a UI library like Material-UI or Bootstrap.
* **Routing**: Use React Router for different pages (e.g., Home, About).

### ****Component-Based Architecture****

The application will be structured around reusable components, where each component can cater to different roles. Here’s how the main components might be organized:

#### **Components for a Bookstore Application:**

* **App Component**: The root component that initializes the application.
* **Context Provider**: A component (like Book Provider) that uses Reacts Context API to provide global state (e.g., list of books) to all child components.
* **Page Components**:
  + **Home Page**: Displays featured books or categories.
  + **Book List**: Lists all the books available.
  + **Book Detail**: Displays detailed information about a selected book.
  + **Add Book**: A form for adding new books.

### ****Routing****

Using **React Router** to manage navigation within the application:

* **Browser Router**: Wraps the application to enable routing capabilities.
* **Route Components**: Define routes for different pages (e.g., /, /add, /books/:id).
* **Link Components**: For navigating between different views without reloading the application.

### ****Styling****

* **CSS Modules**: For component-scoped styles to avoid conflicts.
* **Styled Components**: For dynamic styling using JavaScript.
* **CSS Frameworks**: Integration with frameworks like Bootstrap or Tailwind CSS for responsive design.
* **Testing**

**End-to-End Testing**: Using tools like Cypress for testing the entire application

**Manual Testing**

### ****Deployment****

* **Build Process**: Running npm run build to create an optimized production build.
* **Hosting**: Deploying to platforms like Vercel, Netlify, or GitHub Pages.

**b . Backend Architecture:**

**Technology: Node.js and Express.js**

**Core Technologies**

* **Node.js**: A JavaScript runtime built on Chrome's V8 engine that allows developers to execute JavaScript on the server-side, enabling full-stack JavaScript development.
* **Express.js**: A minimal and flexible Node.js web application framework that provides a robust set of features for building web and mobile applications.

**Routing**

* **Express Router**: Use the Router class from Express to define routes for different resources (e.g., users, products). Each route corresponds to a specific URL and HTTP method (GET, POST, PUT, DELETE).
* **RESTful API Design**: Structure routes to adhere to REST principles, allowing for clear and intuitive interactions with resources.

**Middleware**

* **Built-in Middleware**: Use Express built-in middleware like express.json() for parsing JSON request bodies, and express.static() for serving static files.
* **Custom Middleware**: Create custom middleware functions for handling tasks such as logging, authentication, and error handling.
* **Third-Party Middleware**: Utilize popular middleware packages like body-parser, cors, and morgan for additional functionality.

**Database Integration**

* **MongoDB:** Object Data Mode ling (ODM) libraries like **Mongoose** for MongoDB to interact with the database.

**Authentication and Authorization**

* **JWT (JSON Web Tokens)**: Implement JWT for stateless authentication, allowing users to log in and receive a token to access protected routes.
* **Session Management**: Use libraries like express-session for managing sessions if a stateful approach is needed.

**Error Handling**

* **Global Error Handler**: Define a centralized error handling middleware that captures errors and sends appropriate HTTP responses.
* **Error Logging**: Implement logging for errors using libraries like winston or morgan to track issues and performance.

**Testing**

* **Integration Testing**: Test how different parts of the application work together, ensuring that routes and database interactions function as expected.

**c. Databace:**

**Technology: MongoDB**

**MongoDB Schema Design**

**User Schema**

* **Fields**:
  + username: String (unique)
  + email: String (unique)
  + password: String (hashed)
  + createdAt: Date
  + updatedAt: Date
  + posts: Array of ObjectIDs (references to Post)

**Post Schema**

* **Fields**:
  + title: String
  + content: String
  + author: ObjectID (reference to User)
  + createdAt: Date
  + updatedAt: Date

### Interactions with MongoDB

First, ensure that you have a MongoDB database set up and that you can connect to it. Here’s an example of how to connect to a MongoDB instance using Mongoose.

**4.Setup Instructions**

1. **Prerequisites:** To develop a full-stack flight booking app using React JS, Node.js, and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application.
2. **Installation of MongoDB**: Set up a MongoDB database to store hotel and booking information. Install MongoDB locally using a cloud-based MongoDB service.
3. **React.js:** React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications.
4. **HTML, CSS, and JavaScript**: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

**Installation:** Step-by-step guide to clone, install dependencies, and set up the environment variables.

#### **1.Node.js**

* **Description**: A JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js allows you to run JavaScript on the server-side.
* **Installation**: Download from the [Node.js official website](https://nodejs.org/). It is recommended to install the LTS (Long-Term Support) version for stability.

#### **2. npm (Node Package Manager)**

* **Description**: Comes bundled with Node.js and allows you to install and manage packages (libraries) for your Node.js applications.
* **Usage**: Use npm commands like npm install <package> to install dependencies.

#### **3. Express.js**

* **Description**: A minimal and flexible Node.js web application framework that provides robust features for web and mobile applications.
* **Installation**: Install via npm:
* bash
* Copy code
* npm install express

#### 4. **MongoDB**

* **Description**: A NoSQL database that uses a flexible, document-oriented data model.
* **Installation**: You can run MongoDB locally or use a cloud-based service like MongoDB Atlas.
  + **Local Installation**: Follow the installation instructions on the [MongoDB official website](https://www.mongodb.com/try/download/community).
  + **MongoDB Atlas**: Sign up for a free tier account at [MongoDB Atlas](https://www.mongodb.com/cloud/atlas).

#### **5.** **Mongoose**

* **Description**: An ODM (Object Data Modeling) library for MongoDB and Node.js that provides a straightforward way to model your data.
* **Installation**: Install npm:
* bash
* Copy code
* npm install mangoose

**5. Folder Structure:**

Static files like index.html, favicon, and manifest files.

**Src:**

The primary source folder containing all application logic and resources.

**Assets :**

1. **Purpose**: Store static resources.
2. **Subfolders**:
   * /images: Icons, logos, or general images.
   * /icons: SVGs or other icons.
   * /styles: Global CSS, SCSS, or theme file

**Components:**

1. **Purpose**: Reusable UI components, further subdivided by type or functionality.

* /public: Contains static files that are served directly by the server. The index.html is the main entry point for the React application.
* /src: The main source folder for the application. All React components, styles, and logic reside here.
* /assets: A directory for images, fonts, and other static assets.
* /components: Contains reusable React components that can be shared across different parts of the application.
* /pages: Contains components that represent different pages or views in the application, often corresponding to routes in the app.
* /hooks: Custom React hooks that encapsulate shared logic.
* /context: Contains files related to the React Context API, used for managing state across the application.
* /redux: If using Redux for state management, this folder contains actions, reducers, and the Redux store setup.
* /services: Contains functions that handle API calls, typically using libraries like Axios.
* /utils: Utility functions that are used throughout the application.
* /styles: Contains global styles and styles specific to components.
* env: Contains environment variables for configuration, such as API keys and URLs.
* package.json: Lists project dependencies and scripts for running the application.
* README.md: Documentation for the project, explaining how to set it up and use it.

**6. Running the Application**

* Provide commands to start the frontend and backend servers locally.
  + **Frontend:** npm start in the client directory.
  + **Backend:** npm start in the server directory.

**7. API Documentation**

#### **User Login**

* **Endpoint**: /auth/login
* **Method**: POST
* **Request Body**:

Json

{

"email": "string",

"password": "string"}

**Responses**:

* **200 OK**

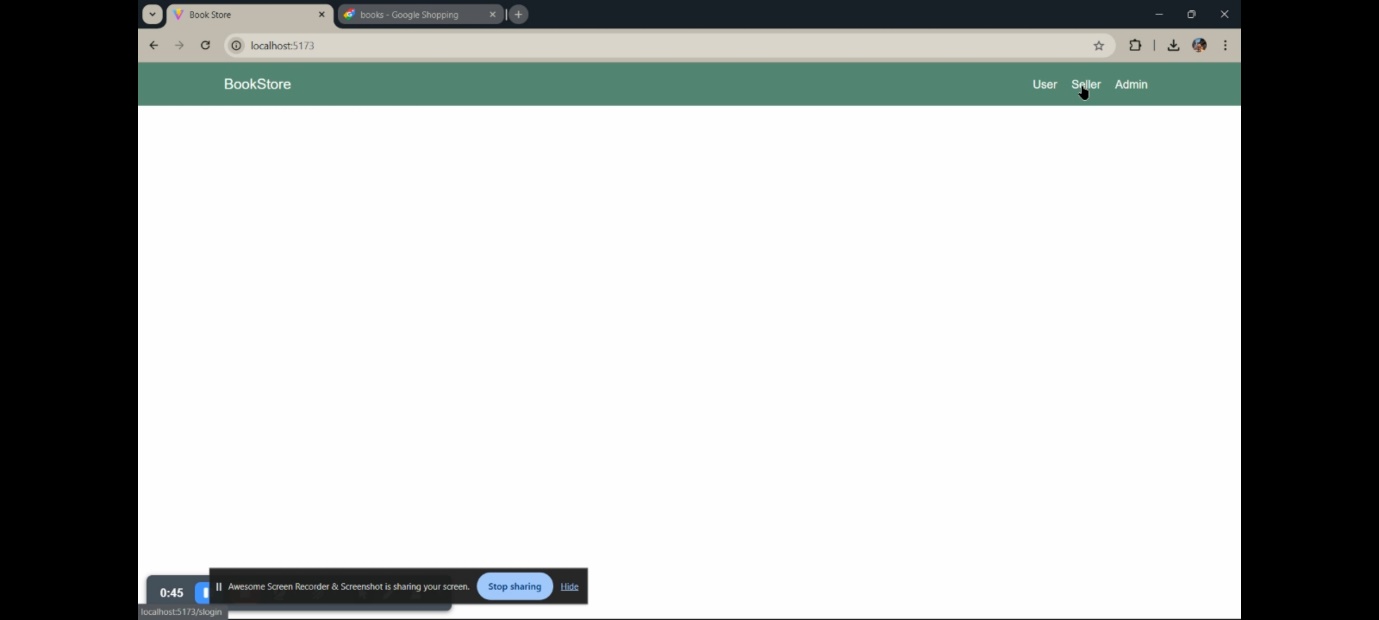
**8. Authentication**

Authentication verifies the identity of users. In this project, it is implemented using **JSON Web Tokens (JWT)**.

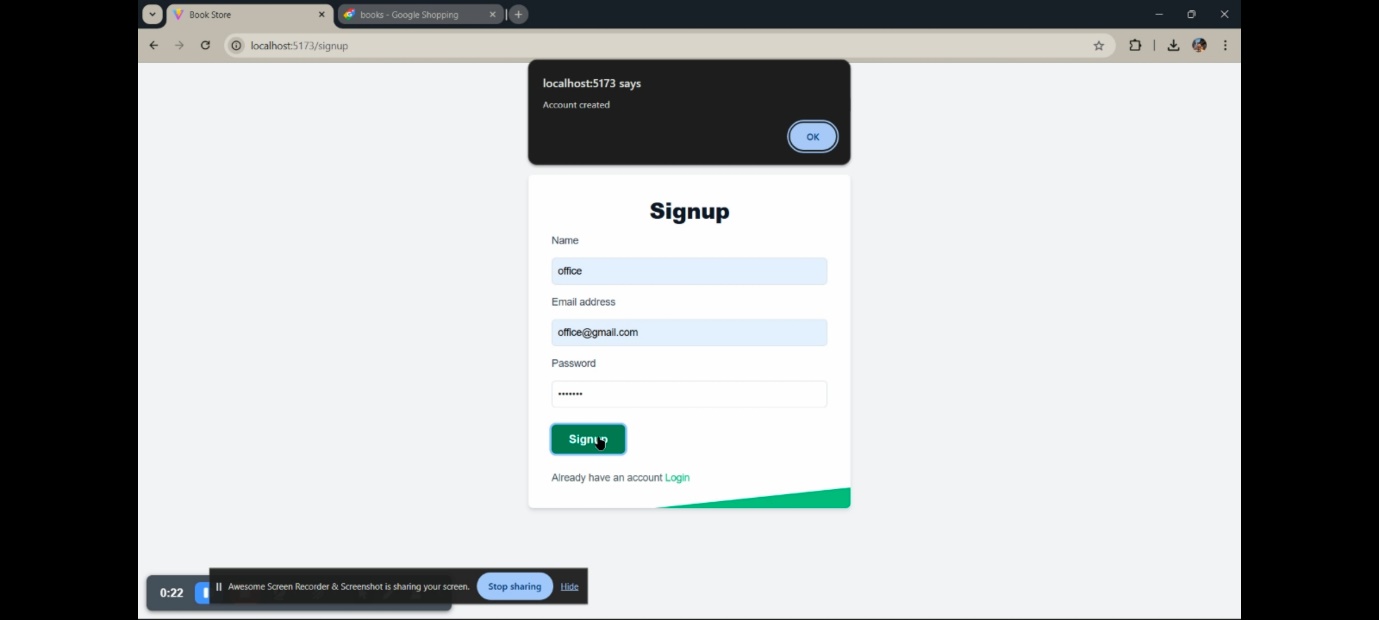
1. **User login**:
   * Endpoint: /api/auth/signup
   * Users provide their details (e.g., name, email, and password).
   * Passwords are securely hashed using a library like **bcrypt** and stored in the database.
   * After successful registration, the user is notified (but no token is issued yet).
2. **Seller Login**:
   * Endpoint: /api/auth/login
   * Users provide their credentials (email and password).
   * The password is verified against the hashed version stored in the database.
   * Upon successful authentication:
     + A **JWT** is generated and returned to the user.
     + This token contains the user’s ID, email, and other claims (e.g., roles) in its payload.
     + The token is signed using a secret key to ensure its integrity.
3. **Admin Login**:
   * Endpoint: /api/auth/login
   * Users provide their credentials (email and password).
   * The password is verified against the hashed version stored in the database.
   * Upon successful authentication:
     + A **JWT** is generated and returned to the user.
     + This token contains the user’s ID, email, and other claims (e.g., roles) in its payload.
     + The token is signed using a secret key to ensure its integrity.
4. **Token Generation**:
   * The JWT is generated using libraries like **json web token**.
   * Claims include:
     + sub (subject): User ID.
     + iat (issued at): Timestamp of token issuance.
     + exp (expiration): Token expiry time .
5. **Storing the Token**:

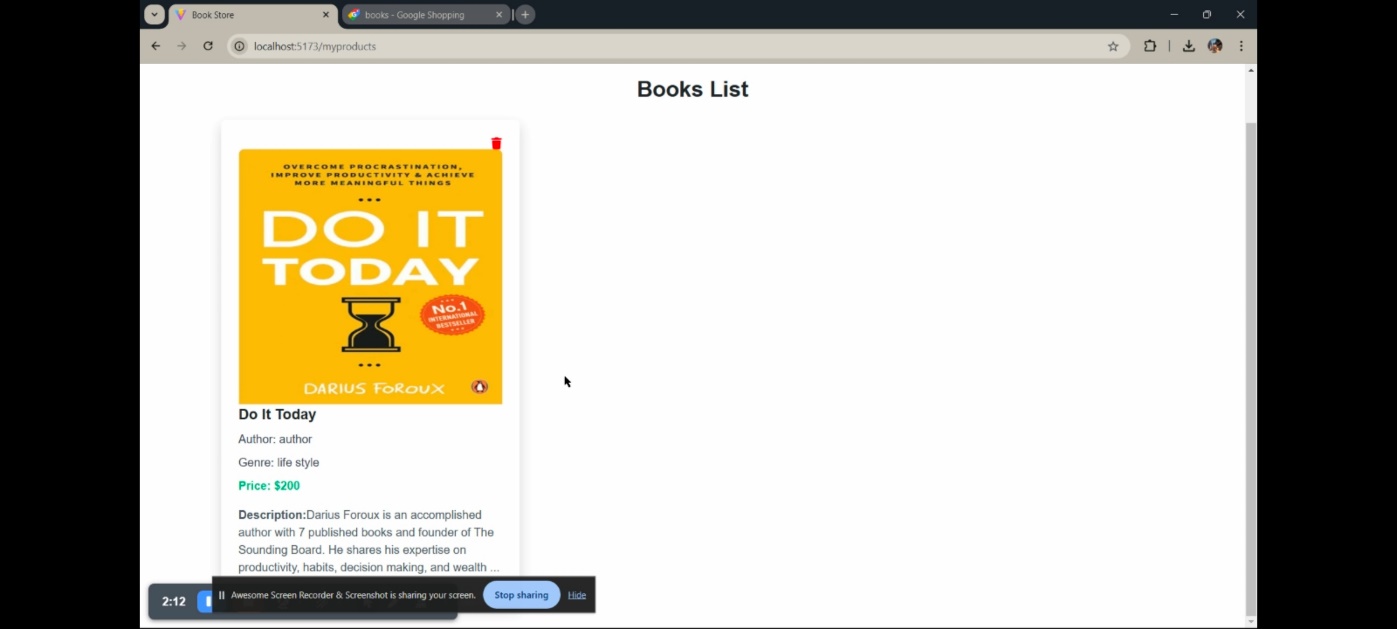
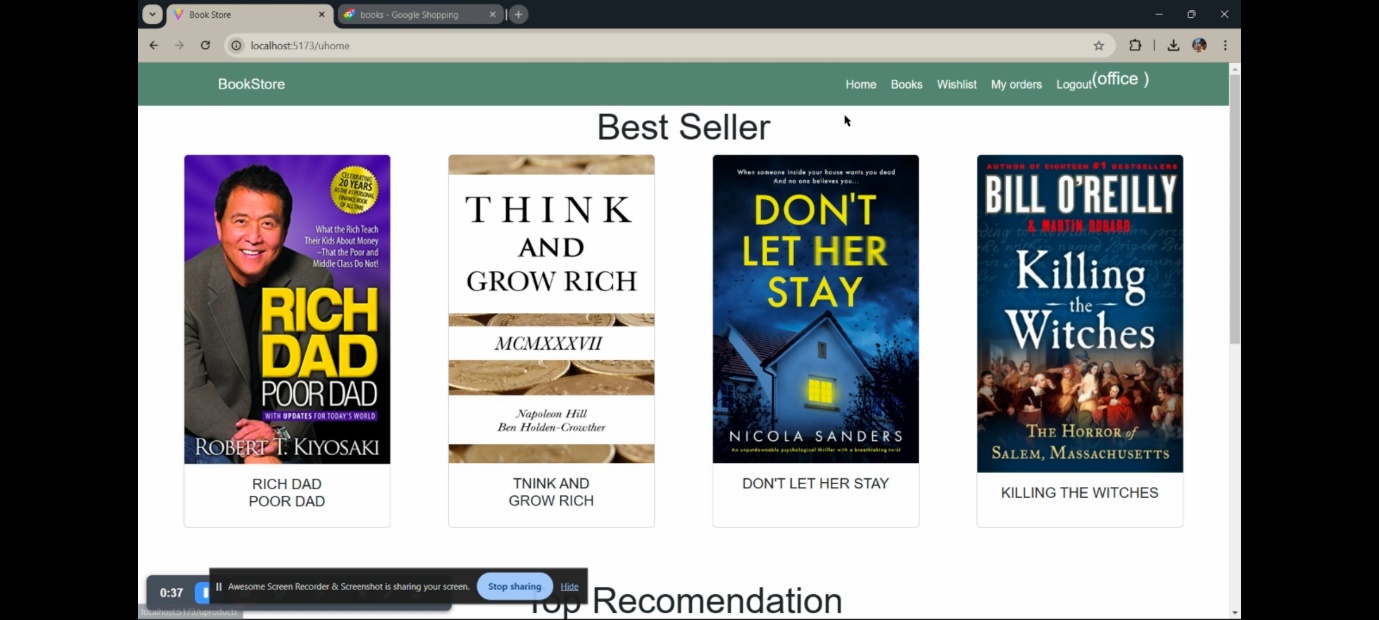
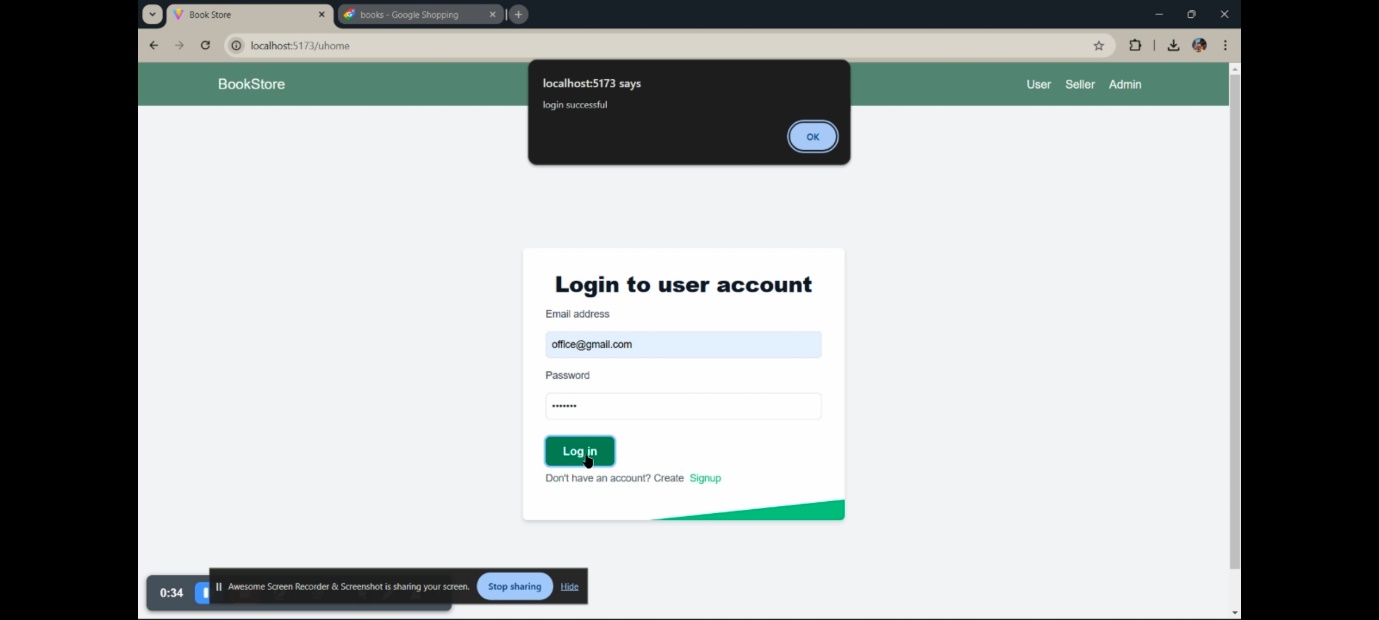
* The client stores the JWT in **localStorage** or **cookies** (with HttpOnly for security).
* For mobile apps, secure storage mechanisms like **Keychain** (iOS) or **Keystore** (Android) are used

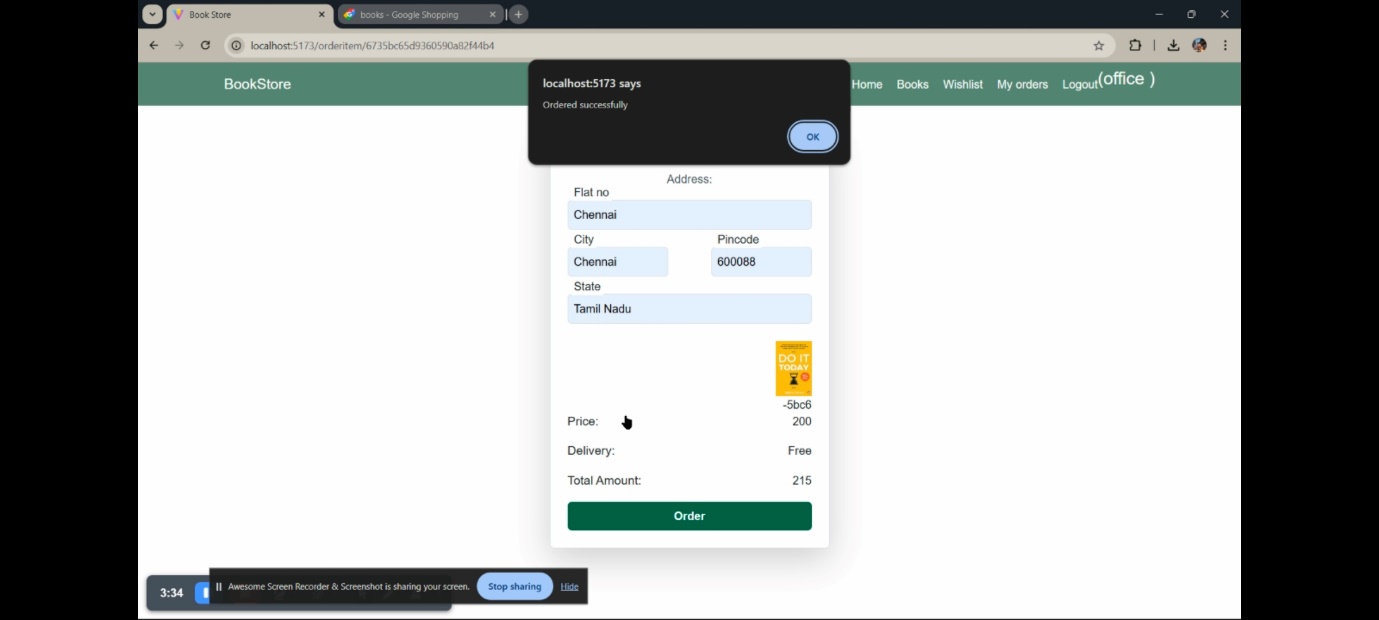
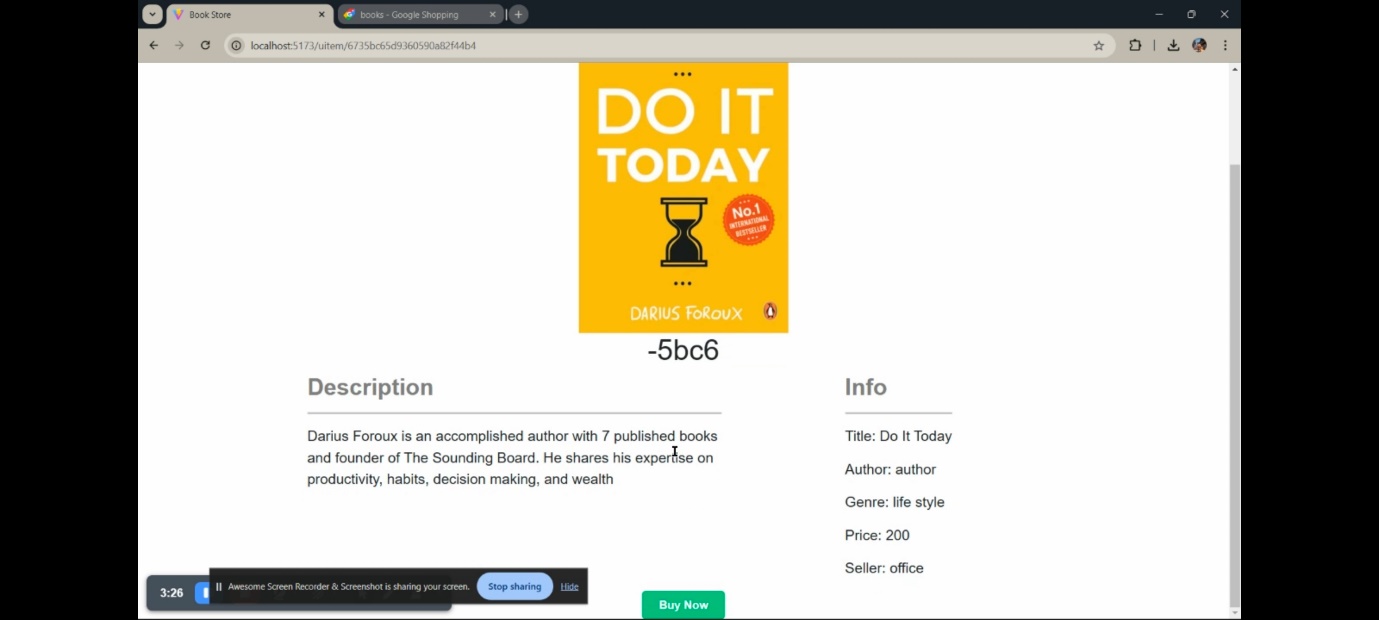
**9. User Interface**



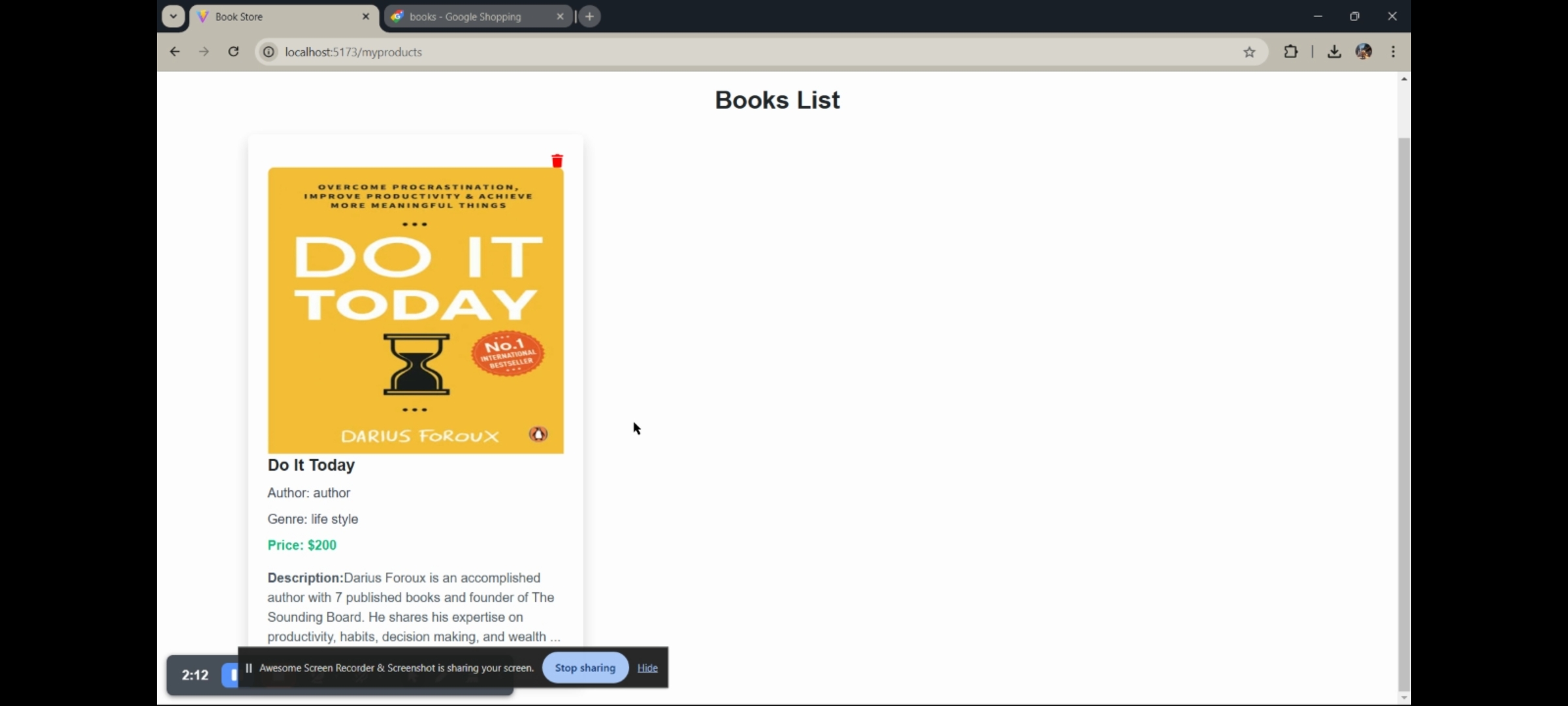
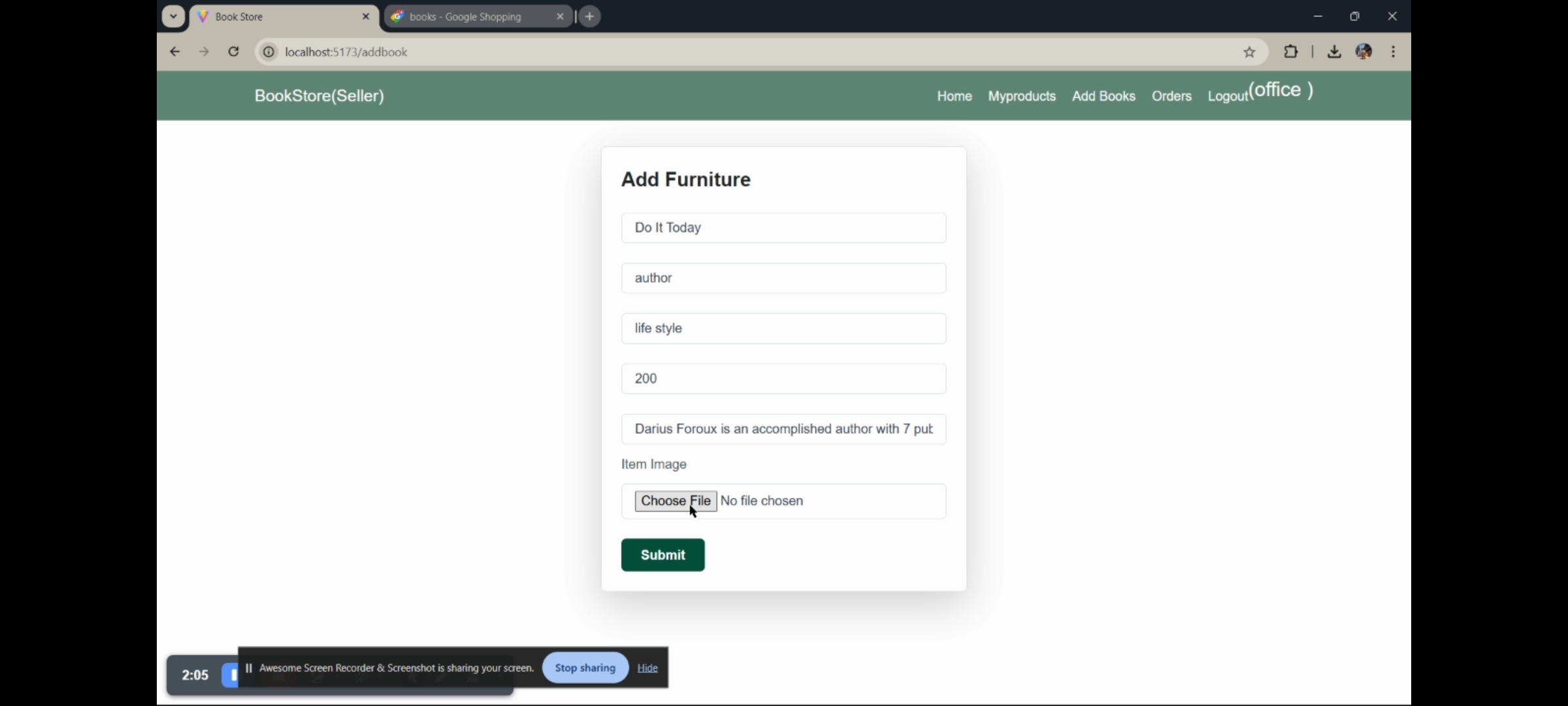
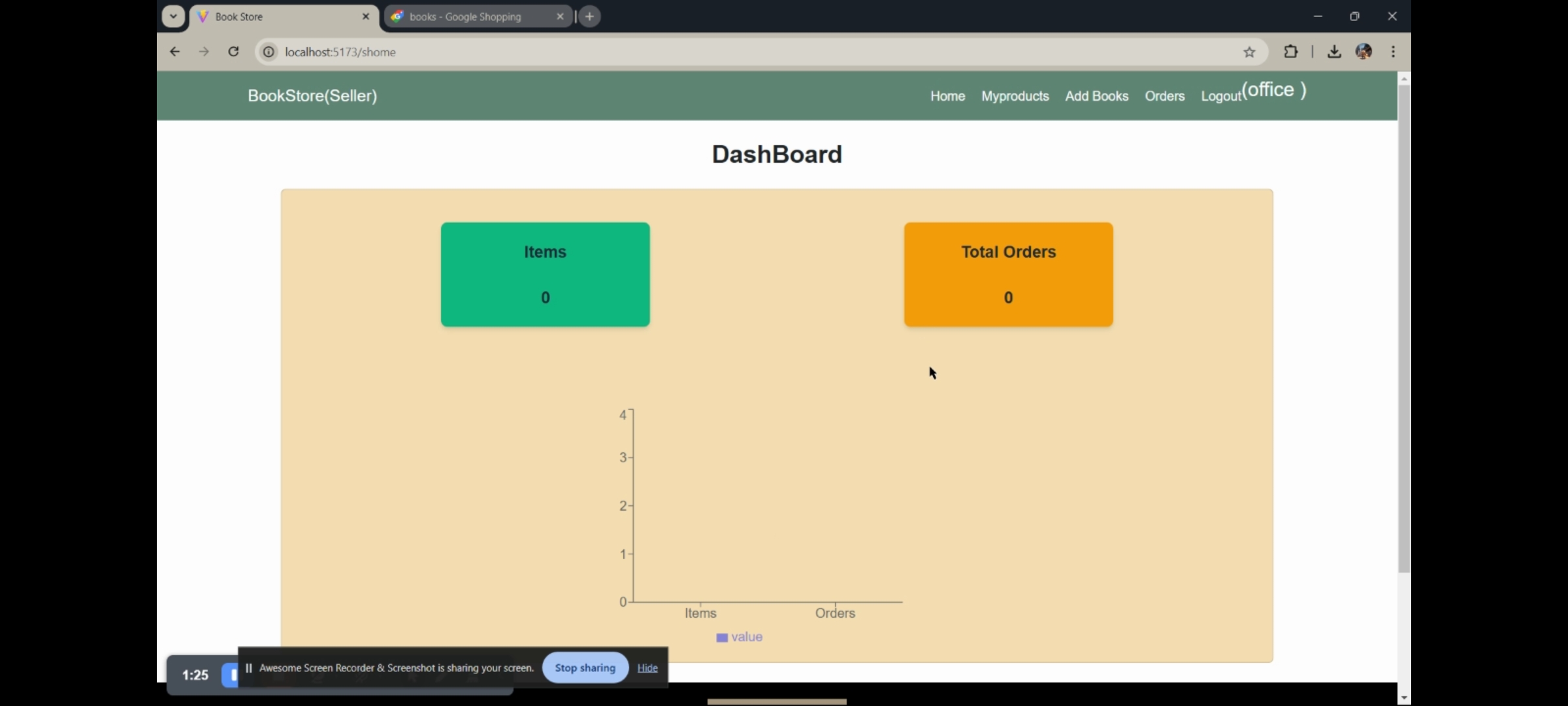
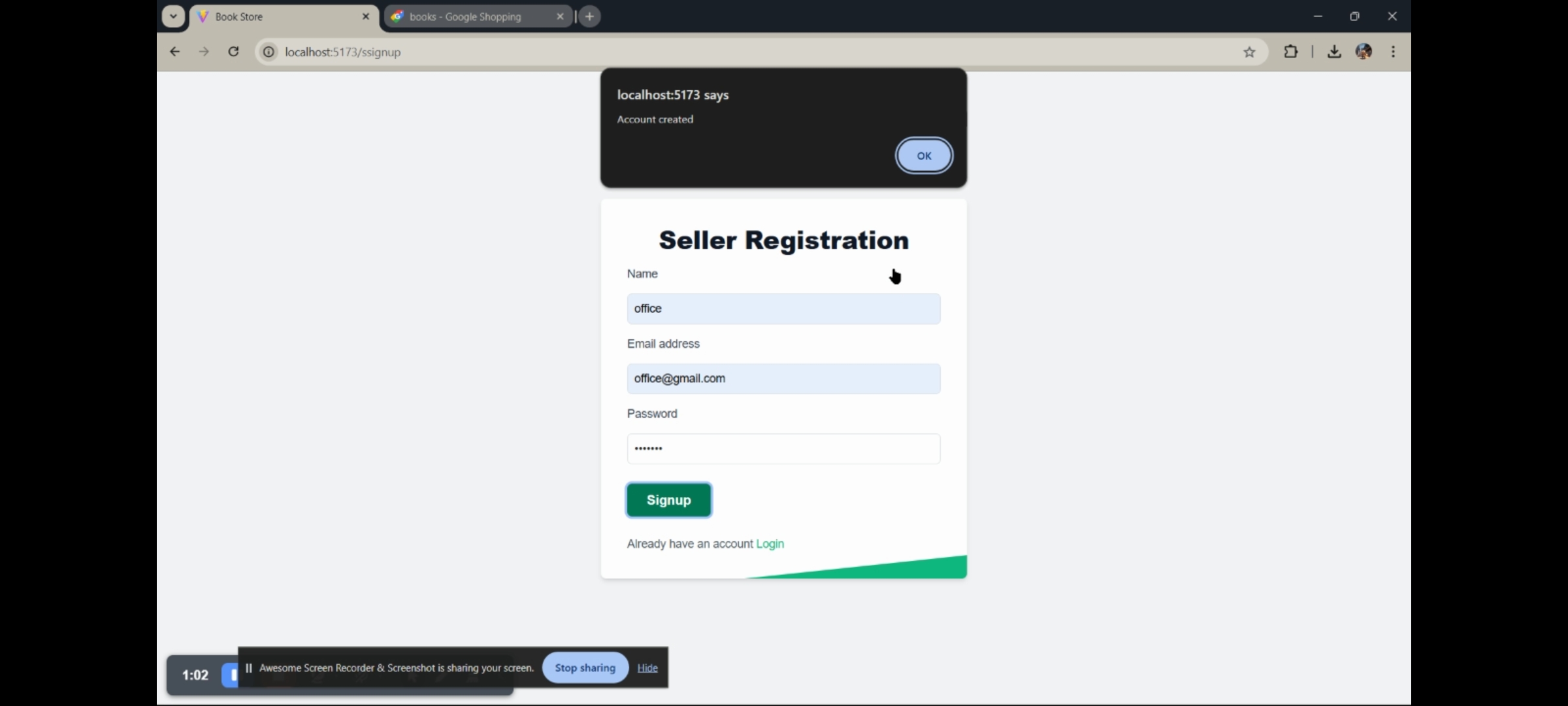
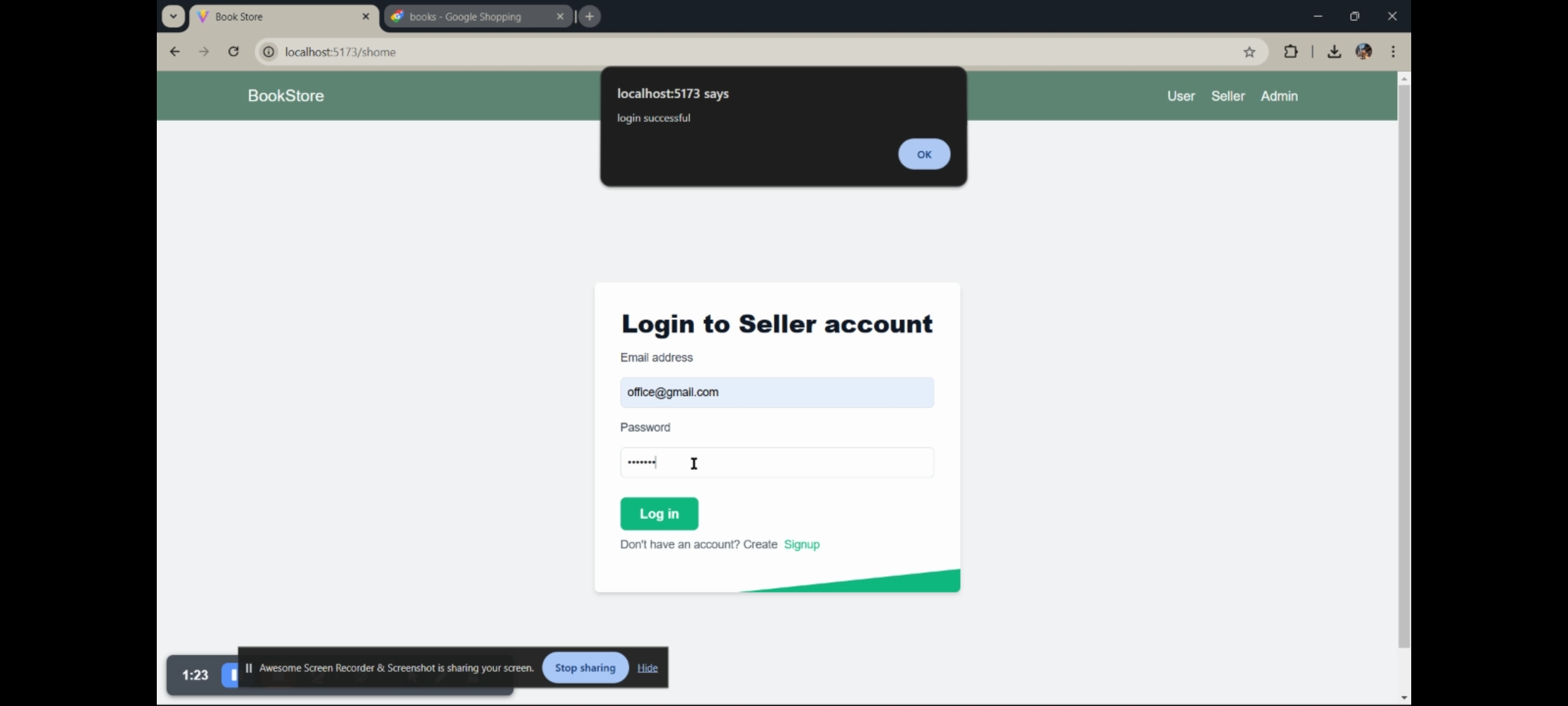
**User :**

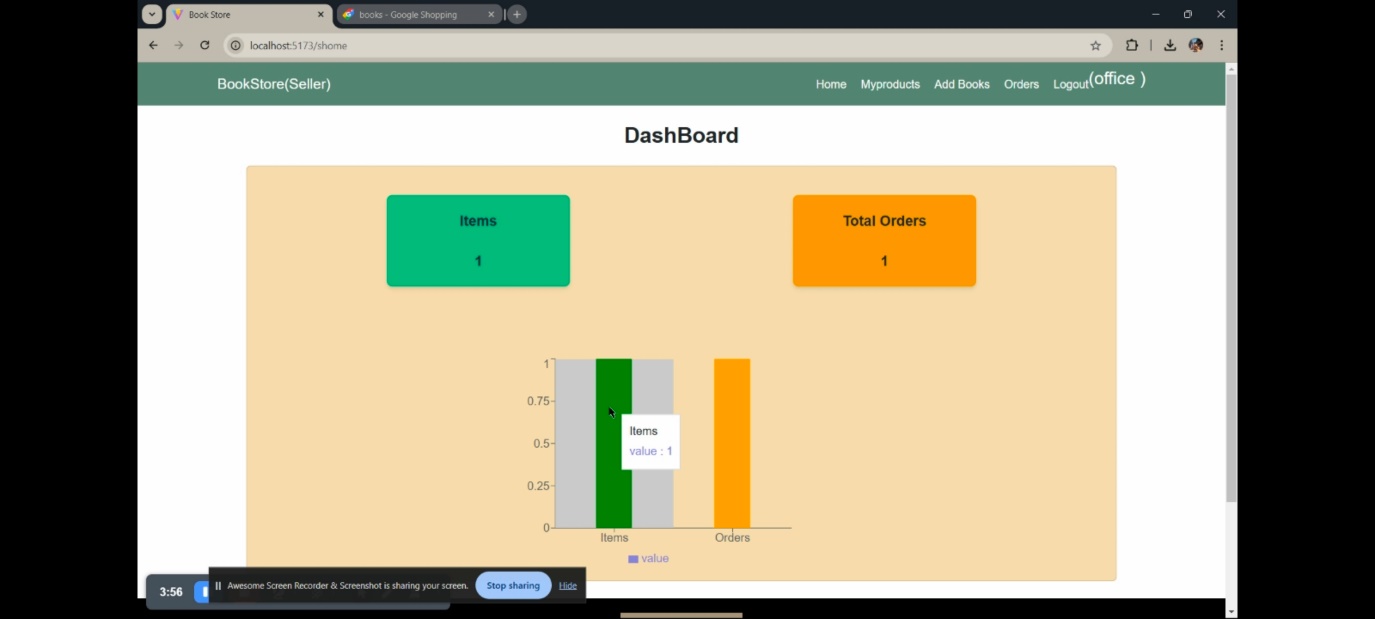




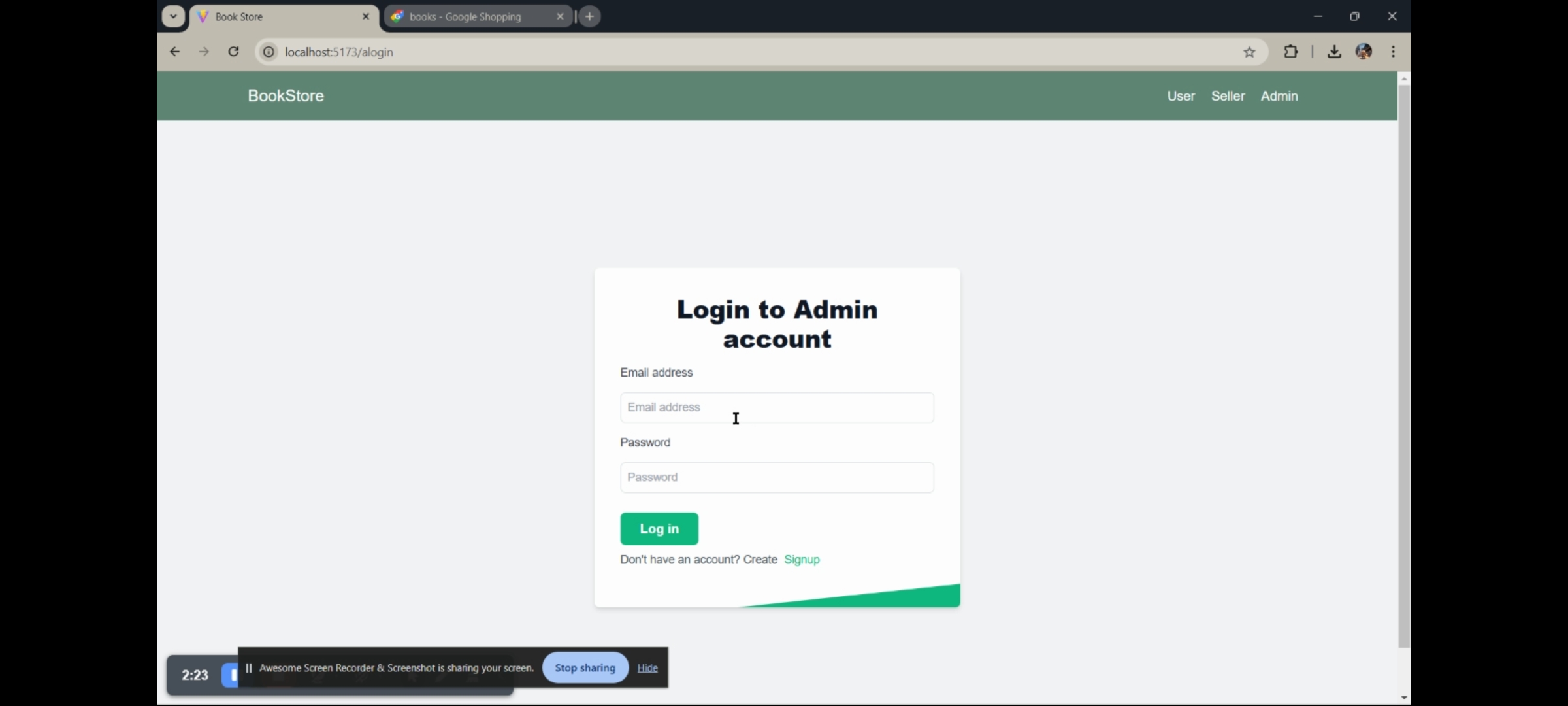


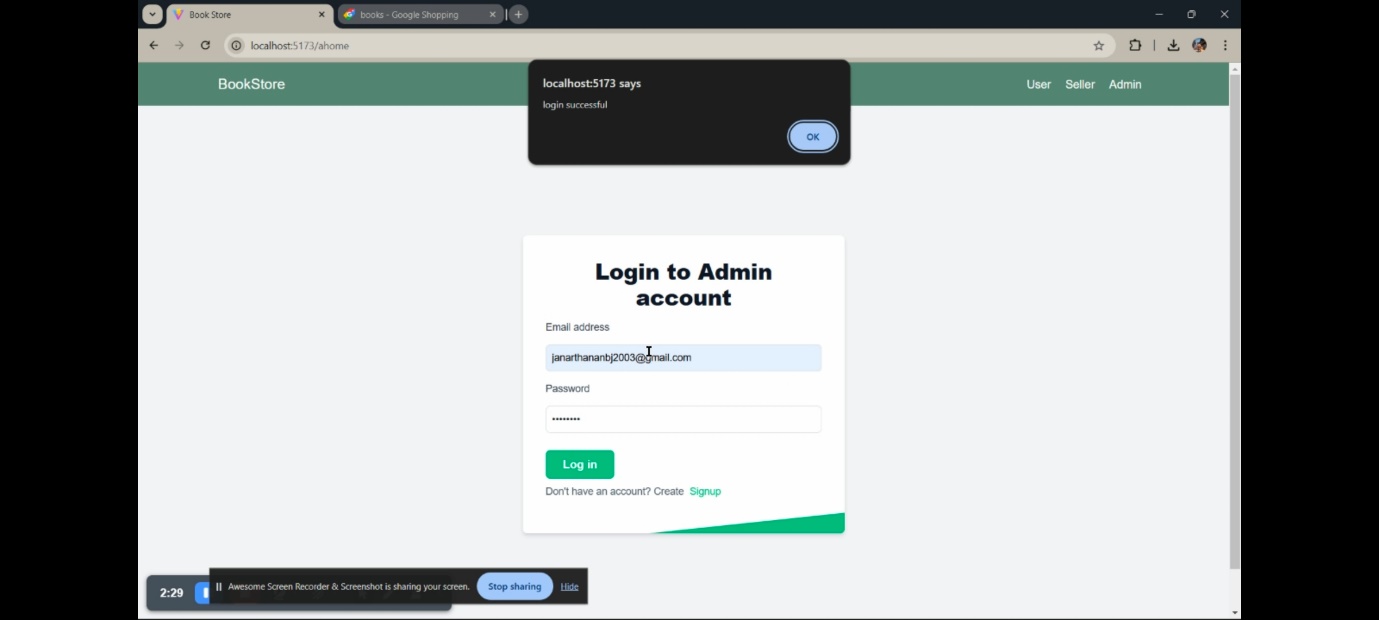
**Seller:**

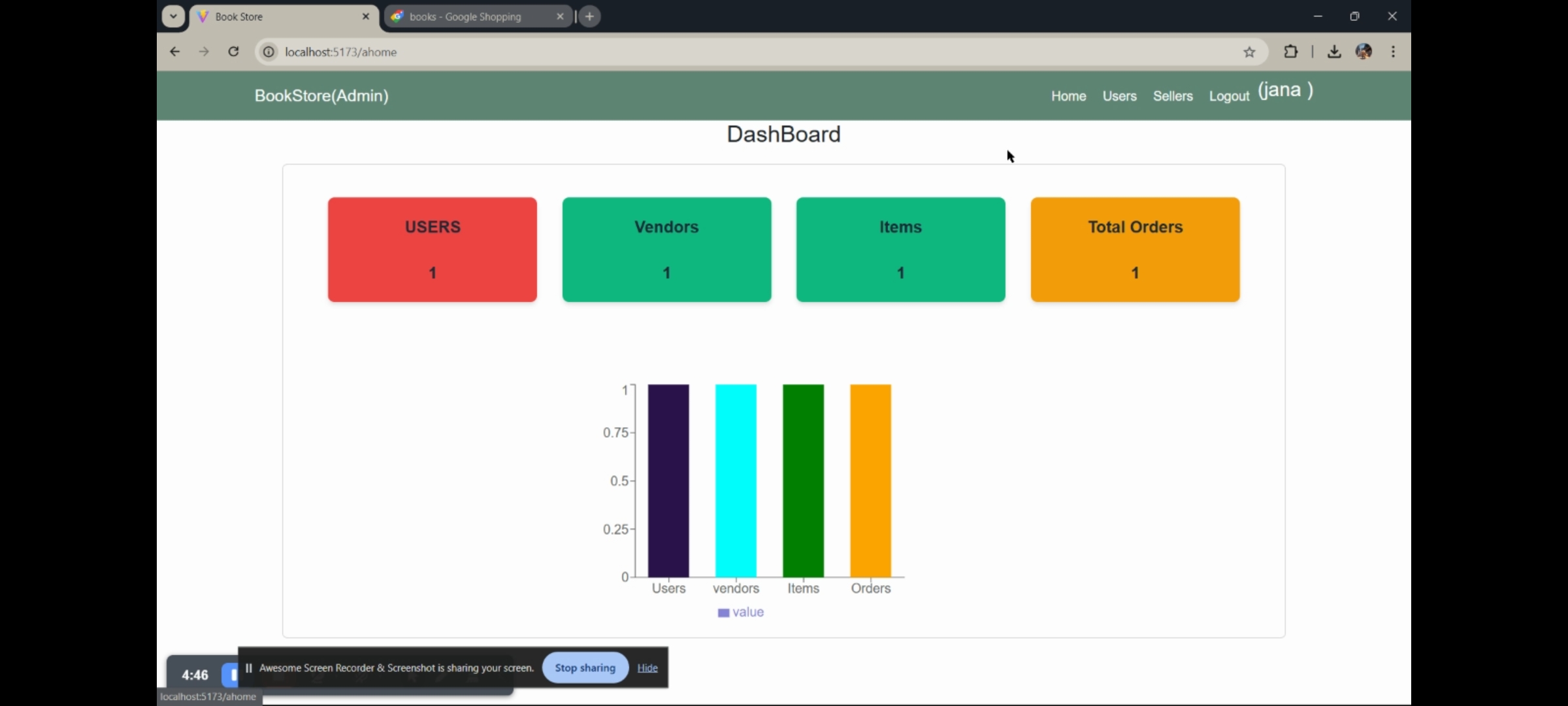




**Admin:**







**10. Testing**

* Manual Testing

**11. Screenshots or Demo**

* Demo video link:

**12. Known Issues**.

* The page is not fully responsive on some mobile devices, causing some form fields to overlap or appear off-screen.

**13. Future Enhancements**

* Expand the application to support multiple currencies and languages to accommodate international users.