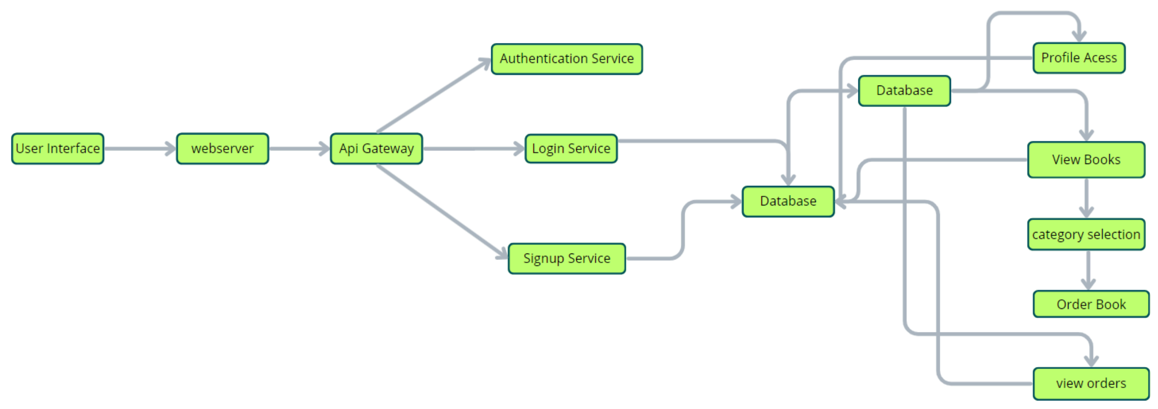
BOOK STORE

TECHNICAL ARCHITECTURE



The BookStore application is designed with a robust and efficient technical architecture to ensure a smooth, secure, and enjoyable reading and shopping experience for users.

1. User Interface (UI)
The front-end interface is where customers interact with BookStore. They can browse books, search by title or author, read detailed descriptions, check prices, and make purchases. The design is intuitive and user-friendly, making navigation and book discovery effortless on any device.

2. Web Server
The web server hosts the BookStore application and delivers dynamic web pages to users. It ensures fast loading times, responsive interactions, and a seamless browsing and shopping experience.

3. API Gateway
The API Gateway acts as the central hub for all client requests. Whether it's fetching book details, processing orders, or managing user accounts, the API Gateway routes

each request to the correct service in the backend, ensuring smooth communication between components.

4. Authentication Service
This service securely handles user login, registration, and authorization. It protects sensitive data like passwords, personal details, and payment information, ensuring only authorized users can access specific features.

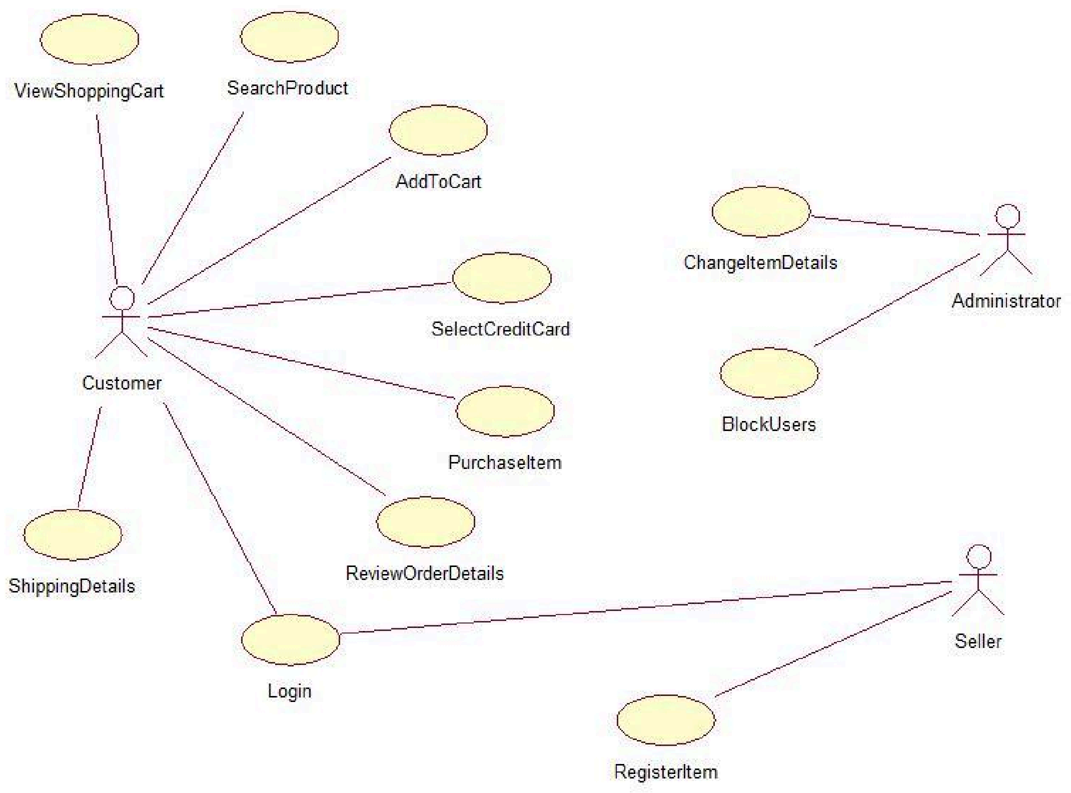
5. Database
The database stores all the important data for BookStore, including:
Book details (title, author, genre, description, price, availability)
User profiles and preferences
Purchase history and reviews
Inventory and stock levels

6. View Books
This feature allows users to explore the entire BookStore collection. They can view books by category, genre, author, or popularity, making it easy to find their next great read.

7. Category Selection
Readers can filter books by specific genres or categories—such as fiction, non-fiction, science, romance, or children's books—for a more personalized browsing experience.

8. Order Management Service
The Order Management Service handles everything from adding books to the cart, specifying quantities, and completing secure purchases, to tracking order status in real-time. It ensures that users are informed every step of the way, from checkout to delivery.

ENTITY RELATION DIAGRAM



User-Book Relationship:
Type: Many-to-Many (M:M). A single user can read or interact with many books, and a single book can be accessed by many users.

Implementation: Introduce an intermediate entity, "Interaction", with foreign keys to both User and Book tables. This table could store additional information like reading progress, reviews, or ratings.

Book-Inventory Relationship:
Type: One-to-Many (1:M). Each book can have multiple copies in inventory, but each copy belongs to one book.

Implementation: Maintain a separate Inventory table with fields like BookID (foreign key), quantity, location, and condition.

User-Order Relationship:
Type: One-to-Many (1:M). A single user can place multiple orders, but each order belongs to one user.
Implementation: Keep the UserID foreign key in the Order table to track user purchase history.

Additional Relationships:
Book-Author Relationship: Many-to-Many (M:M). A book can have multiple authors, and an author can write multiple books. (Similar to User-Book, use an intermediate "WrittenBy" table)
Book-Genre Relationship: Many-to-Many (M:M). A book can belong to multiple genres, and a genre can have many books. (Similar to User-Book, use an intermediate "CategorizedAs" table)
Review-User Relationship: Many-to-One (M:1). A review is written by one user, but a user can write many reviews. (Keep UserID as a foreign key in the Review table).

FEATURES

User Registration and Authentication: Allow users to register accounts securely, log in, and authenticate their identity to access the book store platform.

Book Listings: Display a comprehensive list of available books with details such as title, author, genre, description, price, and availability status.

Book Selection: Provide users with options to select their preferred books based on factors like genre, author, ratings, and popularity.

Purchase Process: Allow users to add books to their cart, specify quantities, and complete purchases securely. Upon successful completion, an order is generated, and the inventory is updated accordingly.

Order Confirmation: Provide users with a confirmation page or notification containing details of their order, including book information, total price, and order ID.

Order History: Allow users to view their past and current orders, providing options to track shipments, review purchased books, and rate their shopping experience.

Organizer Dashboard: Offer administrators an interface to manage book listings, inventory levels, user accounts, orders, and other platform-related activities.

Create Item: Organizer can create items and add new items and he can get the items and he can update items.

Admin Dashboard: Offer administrators an interface to manage book listings, inventory levels, user accounts, orders, and other platform-related activities. Manage the users and organizers.

ROLES AND RESPONSIBILITIES

1. User

Registration – Create an account by providing essential details such as name, email, and password.
Profile Management – Update personal information like name, email, and password.
Book Browsing – Explore available books, browse genres, and search for specific titles or authors.
Logout – Securely log out from the BookEase app.

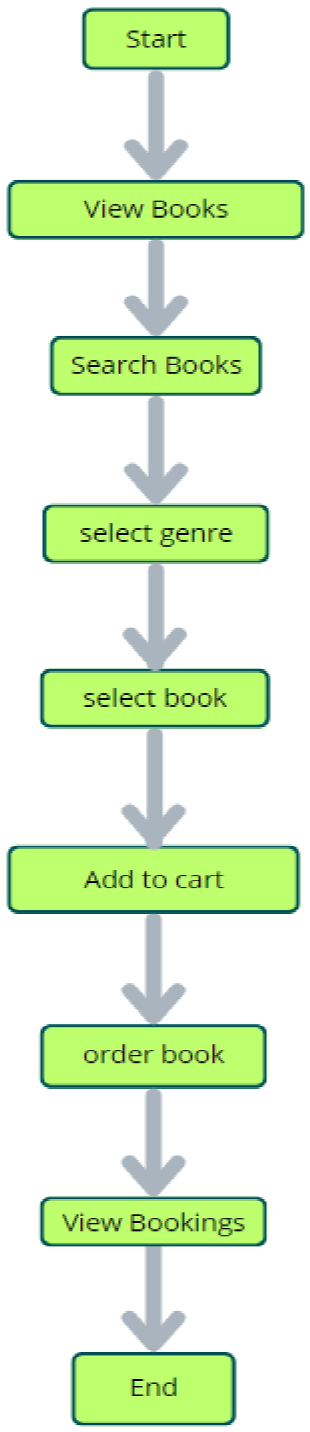
2. Seller

Registration – Create a seller account by providing details such as business name, email, and password.
Profile Management – Update business information like name, email, and password.
Book Listing – Add new books with complete details (title, author, genre, description, price, quantity).
Order Fulfillment – Process and ship orders placed by users in a timely manner.
Logout – Securely log out from the BookEase app.

3. Admin

System Management – Oversee all system functionalities, configurations, and security.
User Management – Create, update, or delete user accounts; manage user ratings.
Book Management – Add, update, or remove book listings from the platform.
Seller Management – Approve new seller accounts, update seller details, and manage ratings.
Logout – Securely log out from the BookEase app.

USER FLOW



Start

Users open the BookEase app to explore a wide collection of books.

Home Page

The home page showcases featured books, categories, and recommendations.

Users can navigate to different sections such as categories, profile, or search.

Access Profile

Users can log in or open their profile to view/update personal details, preferences, and order history.

Book Selection

Users browse through available books with details like title, author, genre, and price.

Search and filter options help them find specific books.

Book Purchase

Users select books, specify quantities, and choose formats (e-book, paperback, special edition).

Selected items are added to the cart.

Order Confirmation

Users review their cart, confirm details, and proceed with secure payment.

After placing the order, they receive a confirmation with book details, price, and order ID.

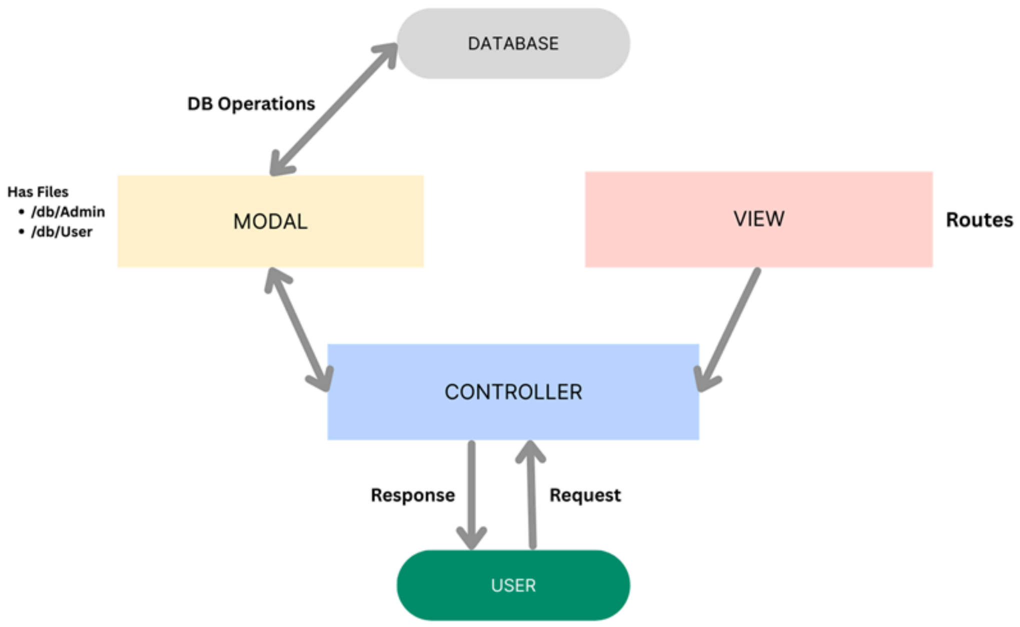
View Orders

Users can check current and past orders, track delivery status, and view payment history.

End

The flow completes once users finish their intended actions, such as making a purchase or browsing books.

MVC PATTERN



The Book store backend application follows the Model-View-Controller (MVC) architectural pattern, a software design approach that separates an application into three interconnected layers. This separation allows for modularity, easier maintenance, and scalability.

Model Layer (Data Layer)

The Model layer is responsible for handling all data-related logic. This includes the definition of data schemas and the operations performed on the database using those schemas. The models are implemented using Mongoose, which provides a schema-based solution to model application data for MongoDB.

Controller Layer

The Controller layer acts as an intermediary between the view (routes) and the model. It receives incoming requests, processes the input (which may include validation or transformation), calls the appropriate methods from the model, and then returns a response to the client.

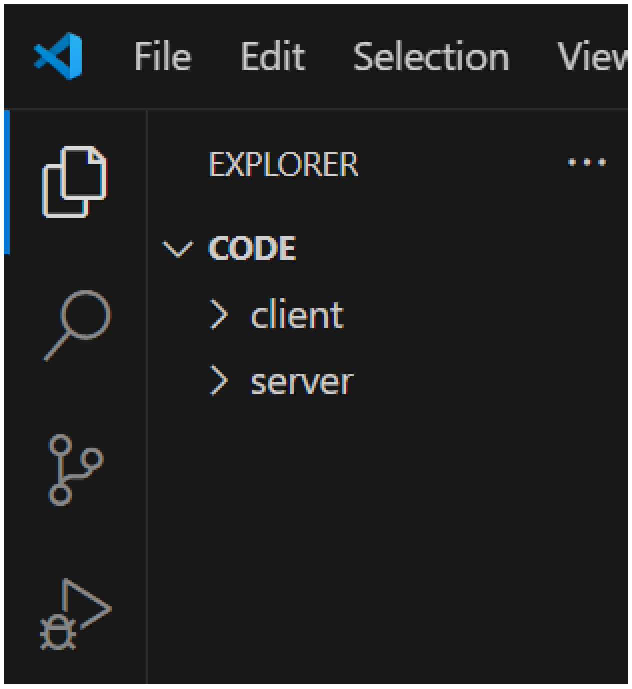
View Layer (Routing Layer)
In the context of a backend REST API, the View is implemented as the routing layer, where various endpoints are defined. These endpoints determine how the backend responds to different HTTP requests (GET, POST, PUT, DELETE) and are responsible for invoking the appropriate controller functions.

Advantages of Using MVC in This Project
Separation of Concerns: Each layer has a clearly defined responsibility, improving readability and maintainability.
Scalability: New features can be added easily by creating new routes, controllers, and models.
Reusability: Logic in controllers and models can be reused across multiple parts of the application.
Testing: Each layer can be tested independently, especially the controllers and models.
Collaboration-Friendly: Multiple developers can work simultaneously on different layers without conflict.

PROJECT SETUP AND CONFIGURATION

Creating project folder

1. Create a new folder with your <project name>.
2. Inside that folder create two new folders.
3. Name one as Client.
4. Name another one as Server.
5. Now open that folder in VS Code.



Client setup (installing react app)

Open the Client folder in the terminal of VScode.

```
npm create vite@latest . -- --template react
```

Select React framework from the given options.

```
Select a framework:
| React
```

Select JavaScript variant from the given options.

```
Select a variant:
| JavaScript
```

Now lets navigate to the client folder by giving the following command.

```
cd client
```

To install all the packages run the following command.

```
npm install
```

To start the React server type the following command.

```
npm run dev
```

Server setup (npm init)

Open Server folder in terminal of VScode.

```
npm init -y
```

Create files:

server.js

Create folders:

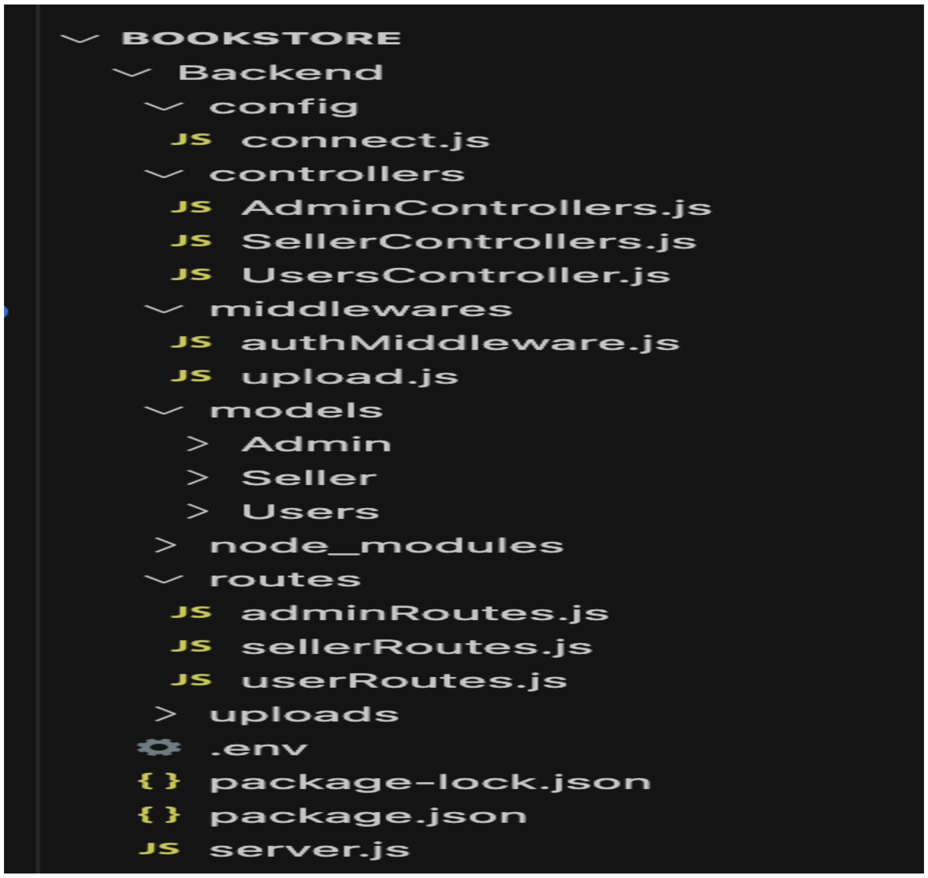
models

controllers

routes

BACKEND DEVELOPMENT:

Backend Structure



```
BOOKSTORE
  Backend
    config
      connect.js
    controllers
      AdminControllers.js
      SellerControllers.js
      UsersController.js
    middlewares
      authMiddleware.js
      upload.js
    models
      > Admin
      > Seller
      > Users
    > node_modules
    routes
      adminRoutes.js
      sellerRoutes.js
      userRoutes.js
    > uploads
    .env
    package-lock.json
    package.json
    server.js
```

Development And Explanation

1)config/connect.js:

Handles the connection to your MongoDB database using Mongoose or another ORM.
Ensures the backend can store and retrieve data.

2) controllers/:
Contains all business logic for handling requests and responses.
AdminControllers.js – Manages admin operations like user/seller approval, book management, and system control.
SellerControllers.js – Handles seller actions such as adding books, updating inventory, and managing orders.
UsersController.js – Deals with user actions like browsing books, making purchases, and leaving reviews.

3) middlewares/:
Reusable functions that run before certain routes to handle security and other processes.
authMiddleware.js – Verifies user or seller/admin tokens to protect private routes.
upload.js – Handles file uploads (e.g., book cover images) using tools like Multer.

4) models/:
Defines the MongoDB schema for each entity.
Admin/ – Schema for admin details and permissions.
Seller/ – Schema for sellers, including their books and profile data.
Users/ – Schema for user accounts, orders, and preferences.

5) routes/:
Maps HTTP requests (GET, POST, PUT, DELETE) to controller functions.
adminRoutes.js – Endpoints for admin actions.
sellerRoutes.js – Endpoints for seller-related operations.
userRoutes.js – Endpoints for user browsing, purchases, and account management.

6) uploads/:
Stores uploaded files such as book images, profile pictures, or documents.

Entry Point – server.js
Initializes Express server.

Connects to MongoDB via db/config.js.

Uses middleware (express.json(), cors, etc.).

Registers all route files.

Listens on a specific port (e.g., 8000).

DATABASE DEVELOPMENT:

Configure MongoDB

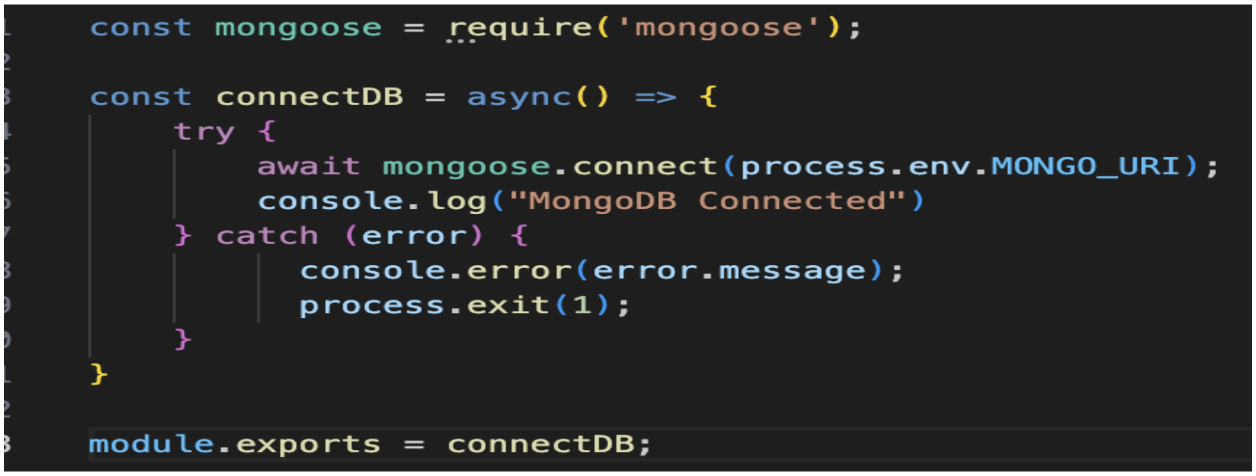
Install Mongoose

In your Node.js project directory, run:

```
npm install mongoose
```

Create Database Connection

Now, make sure the database is connected before performing any of the actions



```
const mongoose = require('mongoose');

const connectDB = async() => {
    try {
        await mongoose.connect(process.env.MONGO_URI);
        console.log("MongoDB Connected")
    } catch (error) {
          console.error(error.message);
          process.exit(1);
    }
}

module.exports = connectDB;
```

through the backend. The connection code looks similar to the one provided below

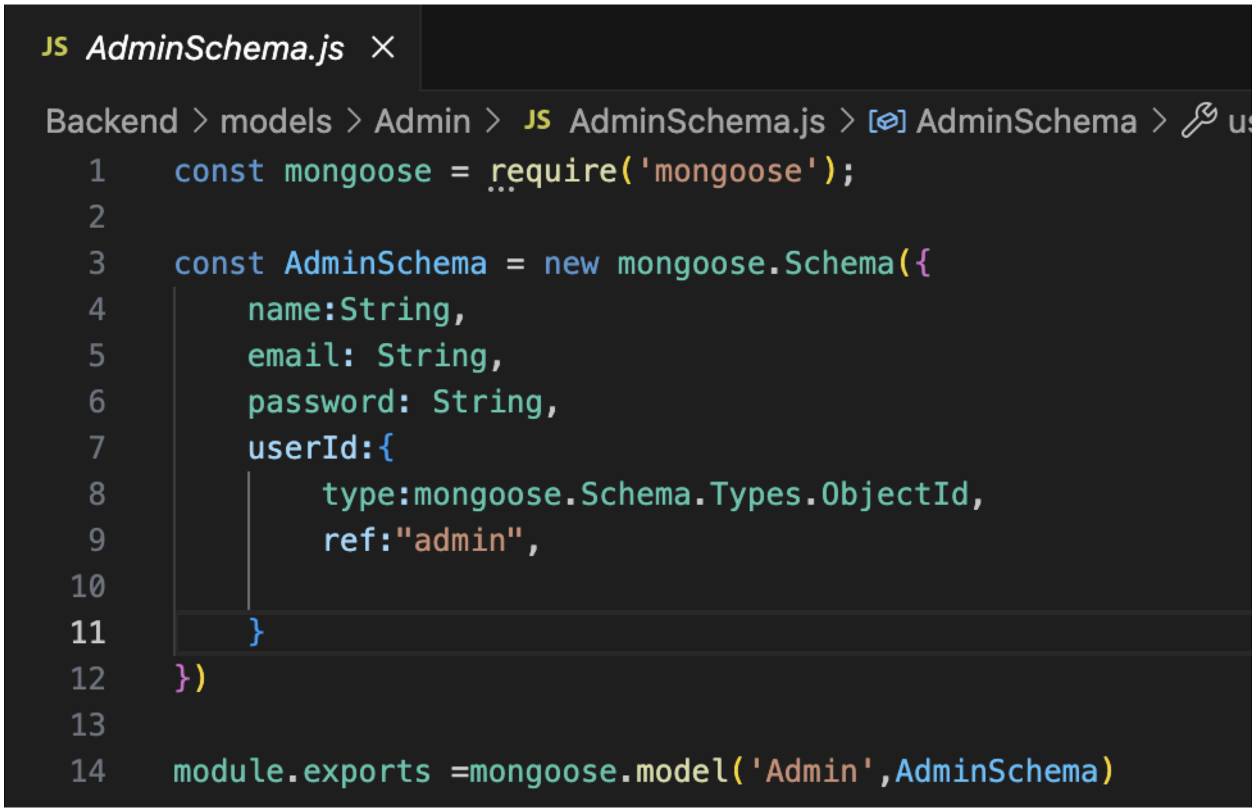
Create Schema and Models

Create Schemas:

Firstly, configure the Schemas for MongoDB database, to store the data in such a Pattern. Use the data from the ER diagrams to create the schemas.

The Schemas for this application look alike to the one provided below.

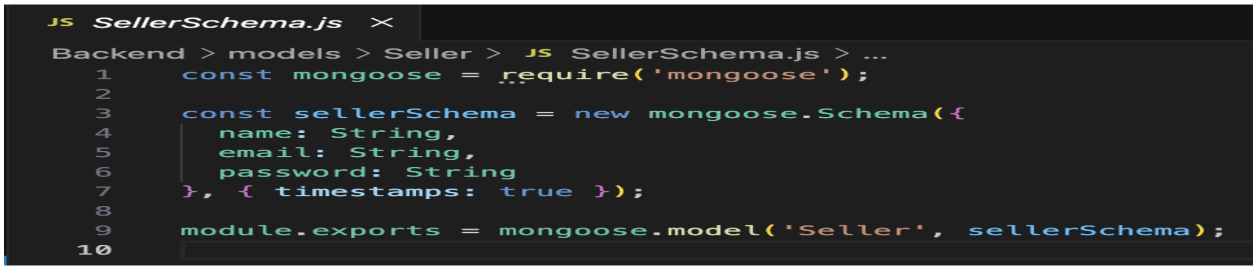
AdminSchema: Schema for admin details and permissions.



AdminSchema.js
Backend > models > Admin > AdminSchema.js > AdminSchema > us

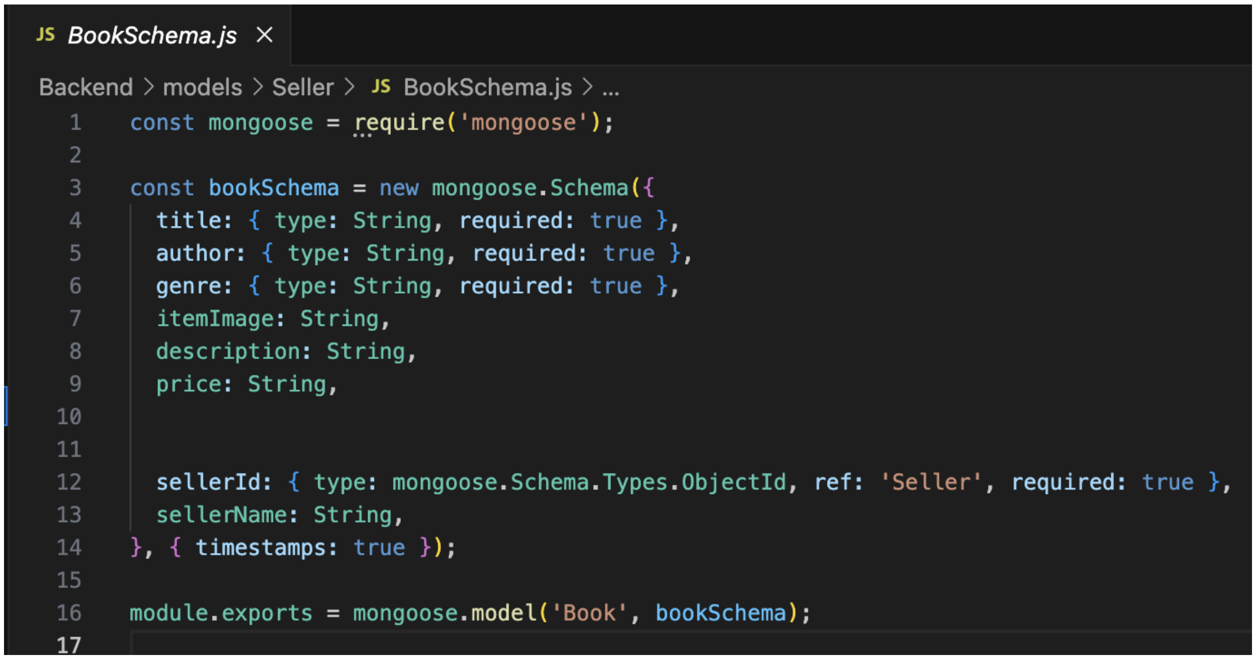
```javascript
const mongoose = require('mongoose');

const AdminSchema = new mongoose.Schema({
    name:String,
    email: String,
    password: String,
    userId:{
        type:mongoose.Schema.Types.ObjectId,
        ref:"admin",

    }
})

module.exports =mongoose.model('Admin',AdminSchema)
```

SellerSchema: Schema for sellers, including their books and profile data.



```javascript
const mongoose = require('mongoose');

const sellerSchema = new mongoose.Schema({
  name: String,
  email: String,
  password: String
}, { timestamps: true });

module.exports = mongoose.model('Seller', sellerSchema);
```

BookSchema:

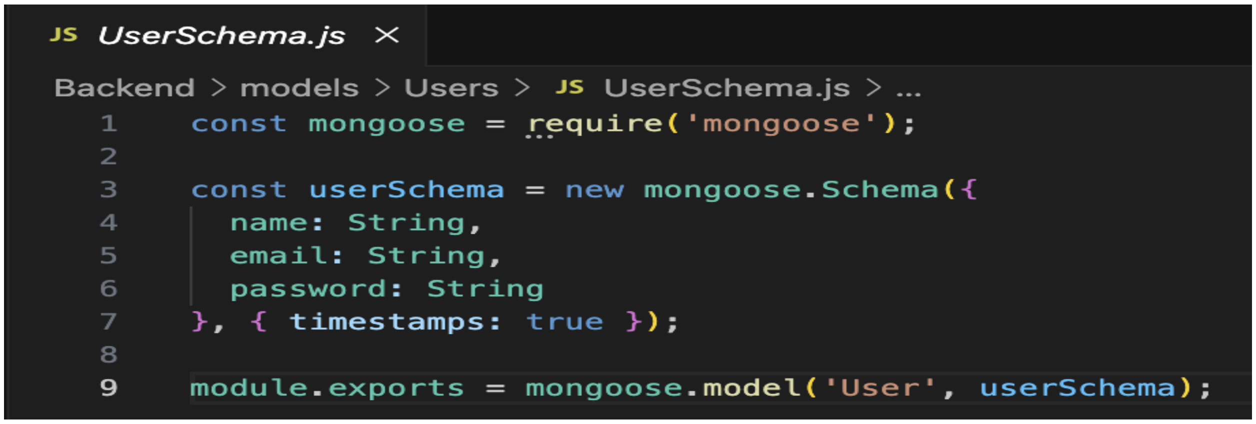


BookSchema.js

Backend > models > Seller > BookSchema.js > ...

```javascript
const mongoose = require('mongoose');

const bookSchema = new mongoose.Schema({
  title: { type: String, required: true },
  author: { type: String, required: true },
  genre: { type: String, required: true },
  itemImage: String,
  description: String,
  price: String,


  sellerId: { type: mongoose.Schema.Types.ObjectId, ref: 'Seller', required: true },
  sellerName: String,
}, { timestamps: true });

module.exports = mongoose.model('Book', bookSchema);
```

UserSchema: Schema for user accounts, orders, and preferences

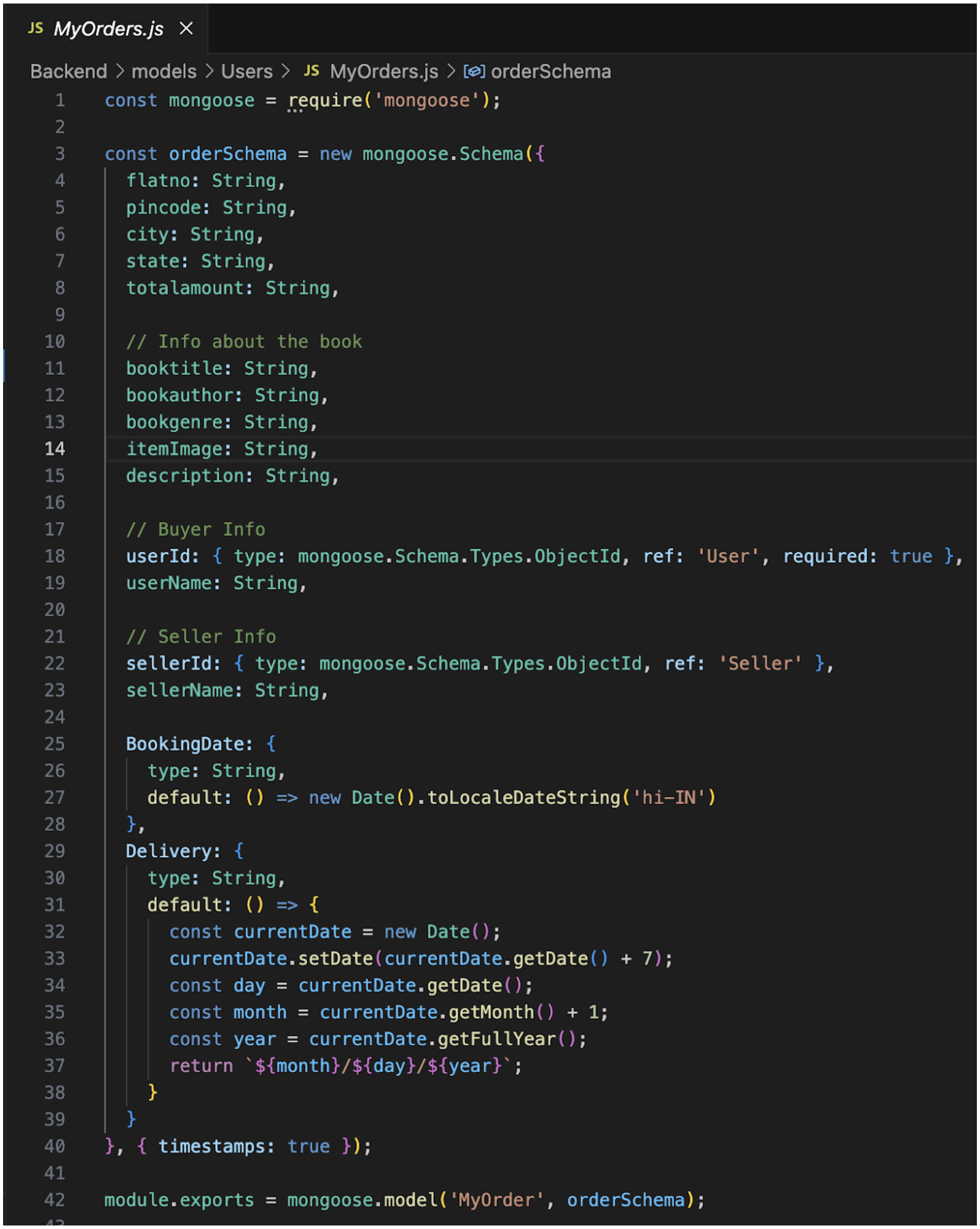


UserSchema.js

Backend > models > Users > UserSchema.js > ...

```javascript
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  name: String,
  email: String,
  password: String
}, { timestamps: true });

module.exports = mongoose.model('User', userSchema);
```

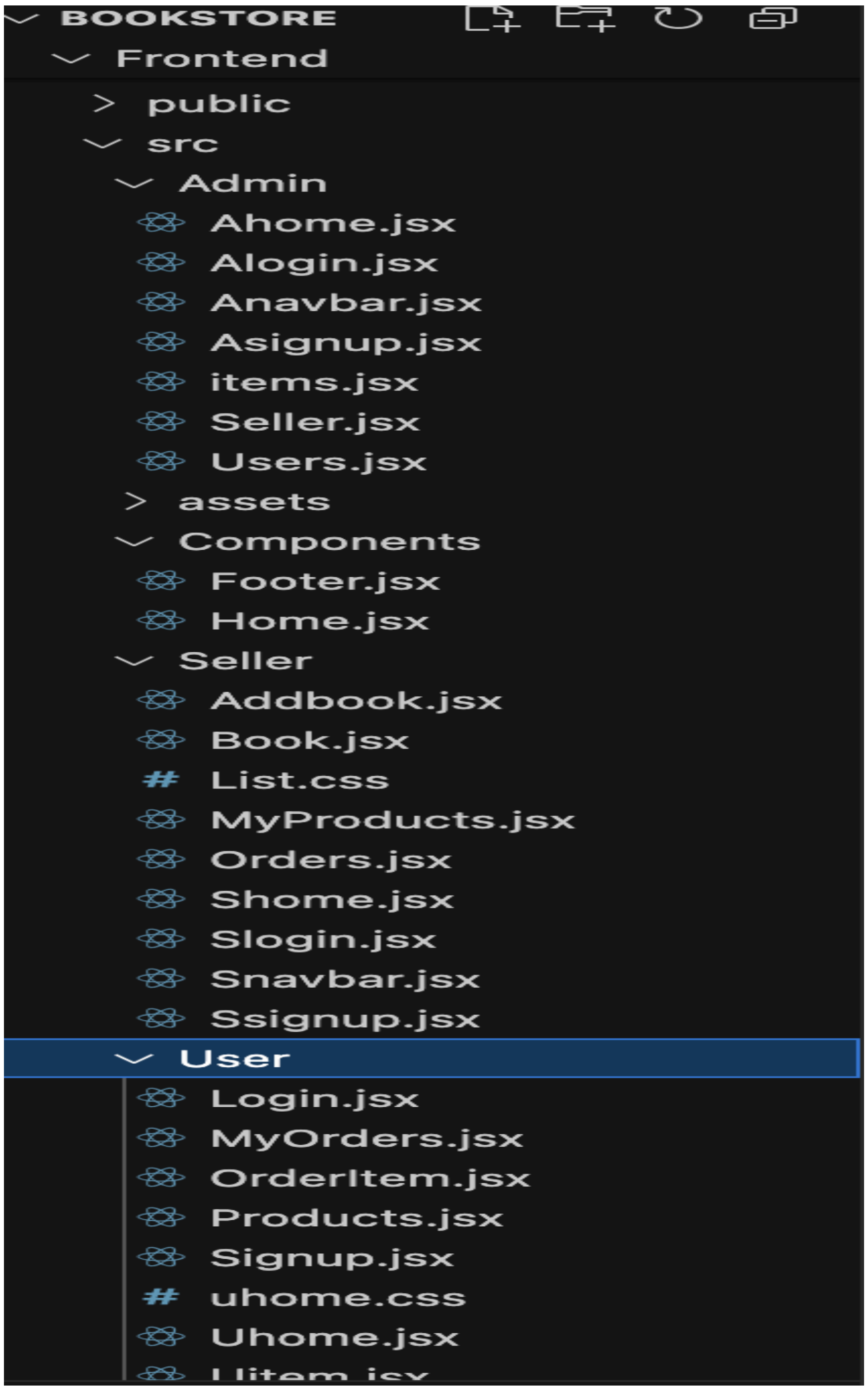
MyOrderSchema:



```javascript
const mongoose = require('mongoose');

const orderSchema = new mongoose.Schema({
  flatno: String,
  pincode: String,
  city: String,
  state: String,
  totalamount: String,

  // Info about the book
  booktitle: String,
  bookauthor: String,
  bookgenre: String,
  itemImage: String,
  description: String,

  // Buyer Info
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  userName: String,

  // Seller Info
  sellerId: { type: mongoose.Schema.Types.ObjectId, ref: 'Seller' },
  sellerName: String,

  BookingDate: {
    type: String,
    default: () => new Date().toLocaleDateString('hi-IN')
  },
  Delivery: {
    type: String,
    default: () => {
      const currentDate = new Date();
      currentDate.setDate(currentDate.getDate() + 7);
      const day = currentDate.getDate();
      const month = currentDate.getMonth() + 1;
      const year = currentDate.getFullYear();
      return `${month}/${day}/${year}`;
    }
  }
}, { timestamps: true });

module.exports = mongoose.model('MyOrder', orderSchema);
```

FRONTEND DEVELOPMENT:

Frontend Structure



```
BOOKSTORE
  Frontend
    public
    src
      Admin
        Ahome.jsx
        Alogin.jsx
        Anavbar.jsx
        Asignup.jsx
        items.jsx
        Seller.jsx
        Users.jsx
      assets
      Components
        Footer.jsx
        Home.jsx
      Seller
        Addbook.jsx
        Book.jsx
        List.css
        MyProducts.jsx
        Orders.jsx
        Shome.jsx
        Slogin.jsx
        Snavbar.jsx
        Ssignup.jsx
      User
        Login.jsx
        MyOrders.jsx
        OrderItem.jsx
        Products.jsx
        Signup.jsx
        uhome.css
        Uhome.jsx
        Uitem.jsx
```

Development and explanation

src/Admin/

Admin-specific React components and pages.

Ahome.jsx – Admin dashboard home page showing system stats, quick actions, and an overview of store activity.

Alogin.jsx – Admin login page with form and authentication logic.

Anavbar.jsx – Navigation bar for admin panel with links to dashboard, users, sellers, and books.

Asignup.jsx – Page for creating new admin accounts.

items.jsx – Displays a list of all books in the store with options for admin to edit or remove.

Seller.jsx – Admin page listing all sellers with options to approve, block, or view details.

Users.jsx – Admin page listing all users with options to manage accounts or view purchase history.

src/Components/

Reusable components used across all roles.

Footer.jsx – Footer section of the app with links, copyright, and branding info.

Home.jsx – Landing page for the website showing highlights, categories, and navigation to other sections.

src/Seller/

Seller-specific pages and tools.

Addbook.jsx – Form page for sellers to add new books with title, author, price, stock, and image upload.

Book.jsx – Component for displaying a single book's details in seller view.

List.css – Stylesheet for book listing layouts in seller pages.

MyProducts.jsx – Page showing all books listed by the seller with edit/delete options.
Orders.jsx – Shows orders received for the seller's books with status update controls.
Shome.jsx – Seller dashboard home page with sales stats and quick navigation.
Slogin.jsx – Seller login page for authentication.
Snavbar.jsx – Navigation bar for seller dashboard with links to products, orders, and profile.
Ssignup.jsx – Seller signup page for creating an account.

4) src/User/
User-facing pages and shopping flow.
Login.jsx – User login form with authentication handling.
MyOrders.jsx – Page showing the user's past and current orders with tracking info.
OrderItem.jsx – Component to display individual order details.
Products.jsx – Product listing page where users browse and filter available books.
Signup.jsx – User signup form for new accounts.
Uhome.jsx – User home page showing featured books, categories, and recommendations.
Uitem.jsx – Component for displaying individual book details in the user shop.

PROJECT EXECUTION

Steps for Execution

Step 1: Set Up the Frontend (React App):
a) Open a terminal and navigate into the client folder:

```
cd client
```

b) Once installation is done, start the React development server:

```
npm run dev
```

c) The app should now be running on:
http://localhost:5173

Step 2: Set Up the Backend (Express Server)

a) Open a new terminal tab/window or split the terminal.

b) Navigate into the server folder:

```
cd ../server
```

Step 3: Configure Environment Variables

Inside the server folder, create a new file named .env (no file extension).

In that .env file, add your MongoDB connection string:

```
MONGO_URI=mongodb://localhost:27017/Nutrition_Assistant
```

Step 4: Start the Backend Server:

Inside the same server folder, run the backend server using:

```
nodemon index.js
```

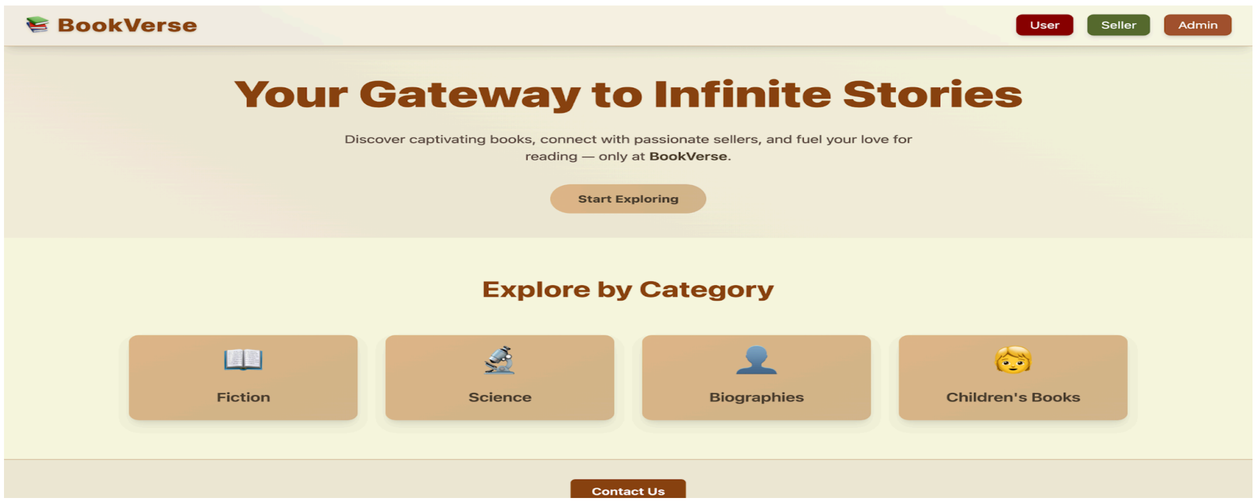
The server should start on:

http://localhost:8000

Demo ScreenShots

Output ScreenShots:

HomePage:

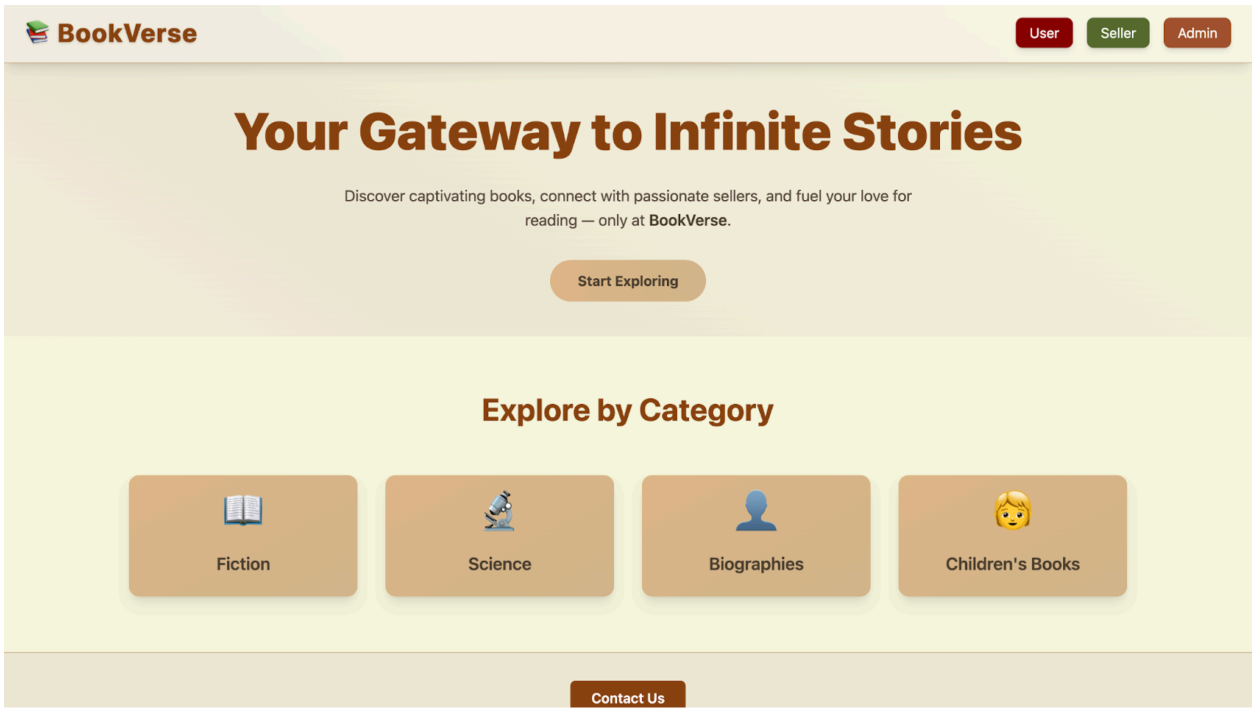


Your Gateway to Infinite Stories

Discover captivating books, connect with passionate sellers, and fuel your love for reading — only at **BookVerse**.

Explore by Category

Admin Login :



BookVerse

User Seller Admin

Your Gateway to Infinite Stories

Discover captivating books, connect with passionate sellers, and fuel your love for reading — only at **BookVerse**.

Start Exploring

Explore by Category

Fiction

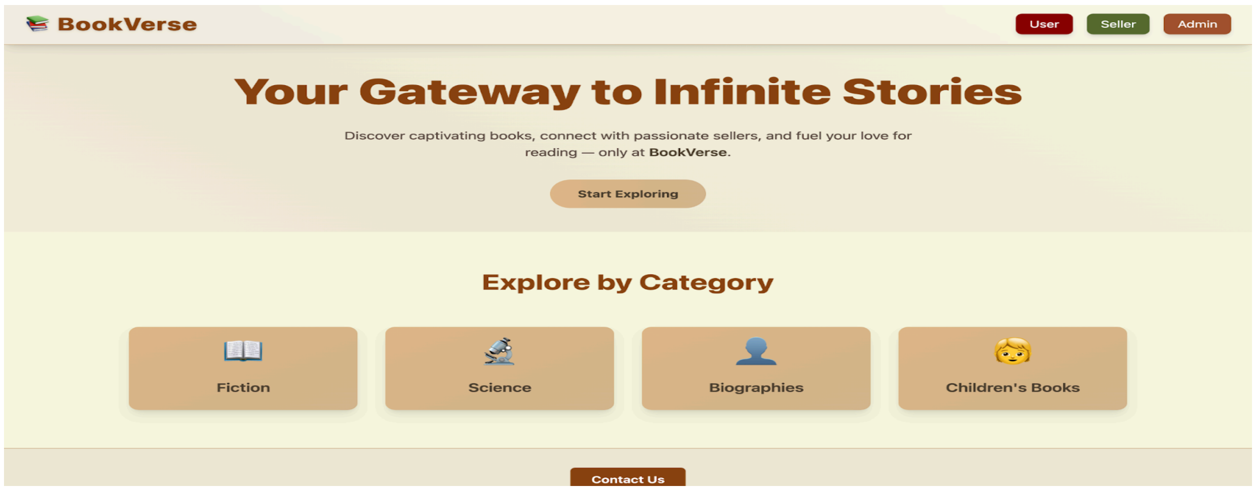
Science

Biographies

Children's Books

Contact Us

Seller login :



BookVerse

User Seller Admin

Your Gateway to Infinite Stories

Discover captivating books, connect with passionate sellers, and fuel your love for reading — only at **BookVerse**.

Start Exploring

Explore by Category

Fiction

Science

Biographies

Children's Books

Contact Us