**Full Stack Development with MERN**

**API Development and Integration Report**

|  |  |
| --- | --- |
| Date | 15-07-2024 |
| Team ID | SWTID1719929609 |
| Project Name | Project – BOOK NEST |
| Maximum Marks | 6 |

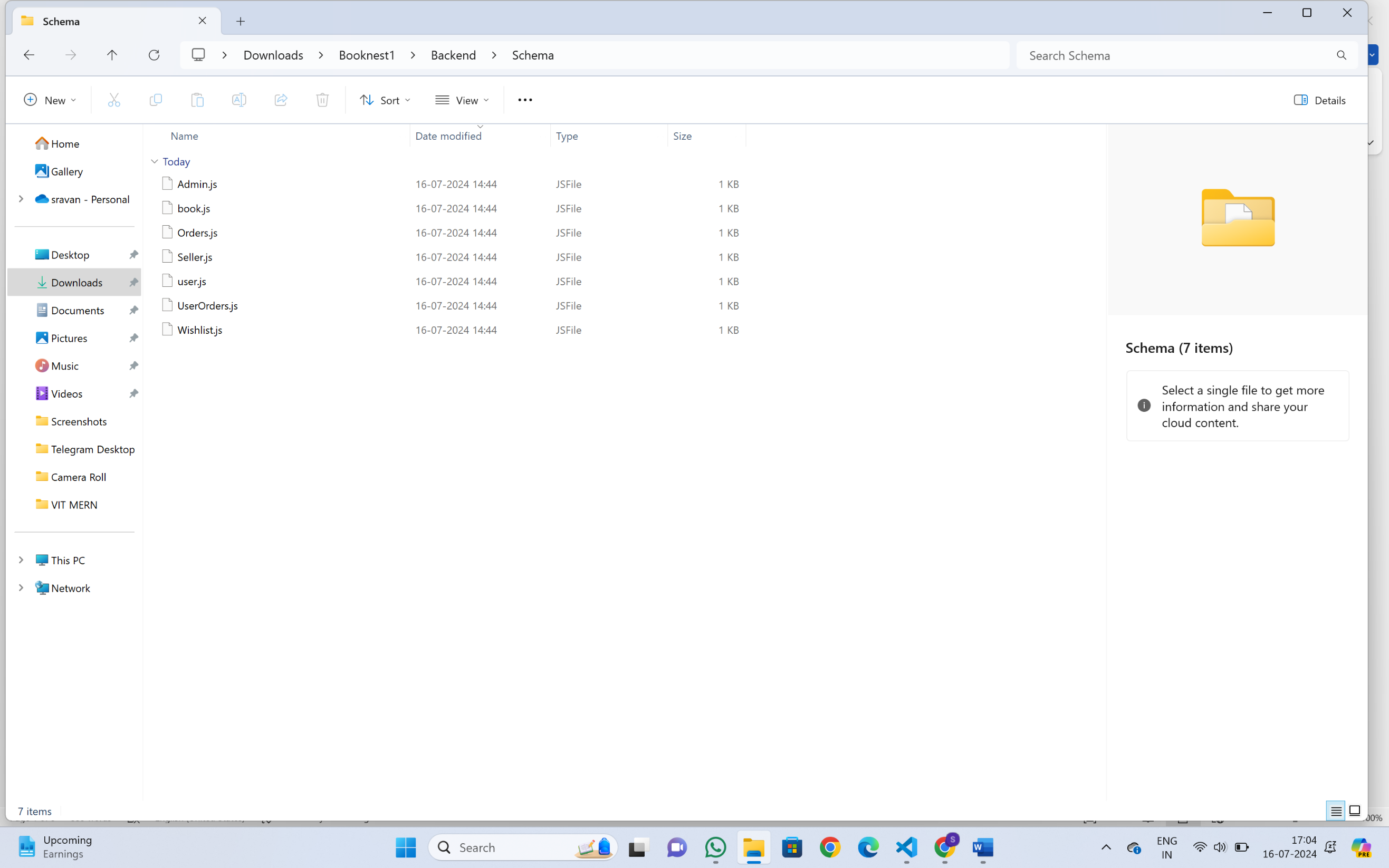
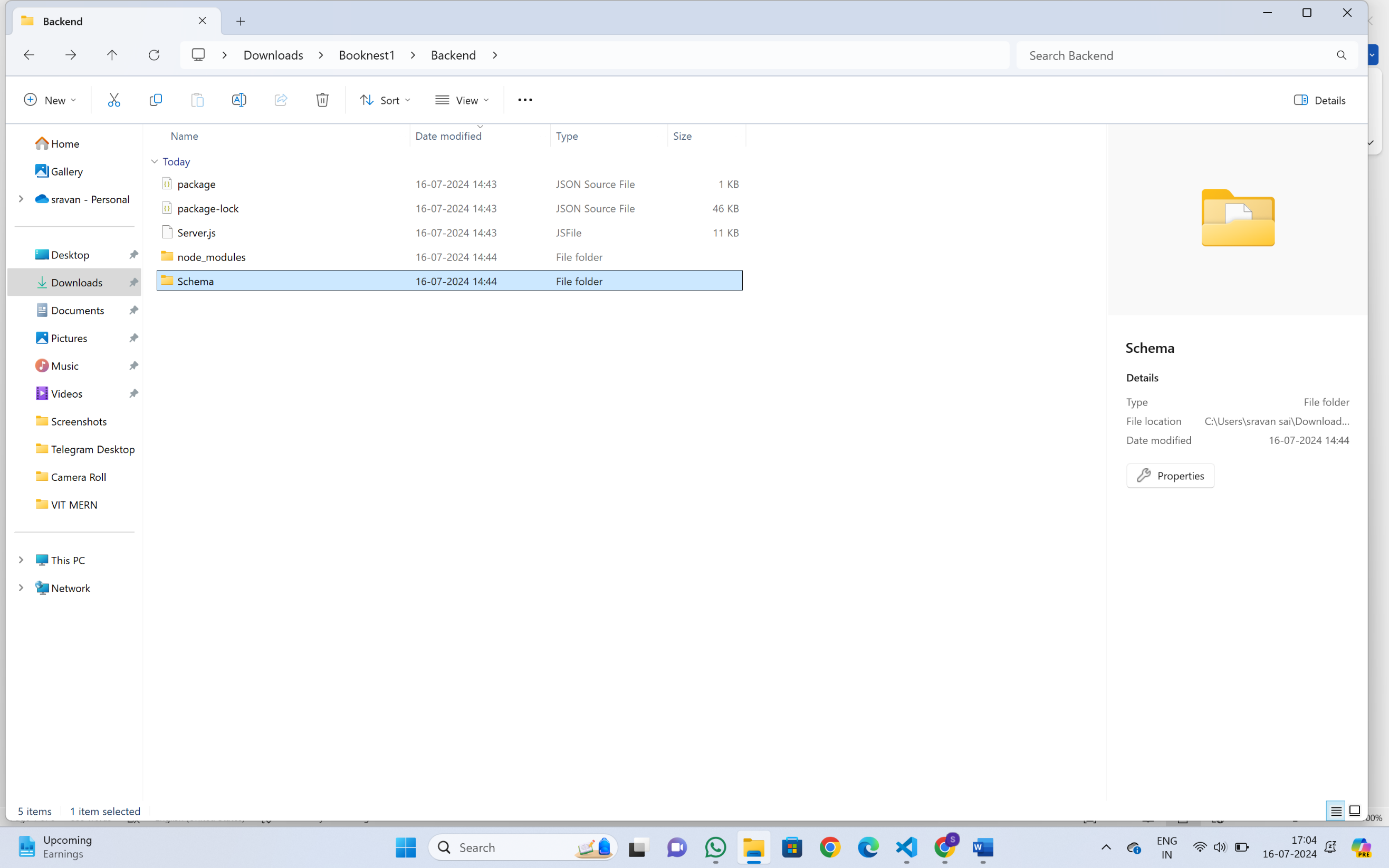
**Project Title:** Book Nest  
**Date:** 15-07-2024  
**Prepared by:** Team Book Nest

**Objective**  
The objective of this report is to document the API development progress and key aspects of the backend services implementation for the Book Nest project.

**Technologies Used**

* **Backend Framework:** Node.js with Express.js
* **Database:** MongoDB
* **Authentication:** JWT

**Project Structure**



**Key Directories and Files**

**1./server.js :** Contains functions to handle requests and responses.

2.**/Schema** : Includes mongoose schemas and MongoDB collections.

1.Admin.js 2. Book.js 3.Orders.js 4.Seller.js 5.user.js 6.UserOrders.js 7.Wishlist.js

**User Authentication API Endpoints**  
A summary of the main API endpoints and their purposes:

* **POST /usersignup** - Register a new user.
* **POST /userlogin** - Authenticates a user and returns a token.

**User Management**

* **GET /user/-** Retrieves user information by ID.
* **PUT /user/**- Updates user information by ID.

**Seller API Endpoints**

* **POST/sellersignup**- Registers a new Seller
* **POST /sellerlogin** – Authenticates a seller and returns a token.

**Seller Management**

* **GET /seller/** - Retrieves seller information by ID
* **POST /addbooks** – updates seller information

**Admin API Endpoints**

* **POST/ausers**- Registers a new admin
* **POST /adminlogin**– Authenticates a admin and returns a token.

**Admin Management**

* **GET /ausers** - Retrieves seller information by ID

**Books**

* **GET /allbooks** - Retrieves all books
* **POST /addbooks**  - adds a new book
* **GET/seller/getbooks/:id** - retrieves a book by ID
* **DELETE/deletebook/:id** – deletes a book by ID
* **DELETE/seller/deletebook/:id –** deletes a book addes by seller by ID

**Orders**

* **GET /getorders/:id**- Retrieves order by id
* **GET/getuserorders/:id**  - Retreives user orders by ID
* **POST/adduserorder** : Adds orders placed by users
* **GET/seller/getorders/:id** : Retrieves orders places in seller page by ID
* **GET/aorders** - retrieves orders placed in admin page

**WishList**

* **GET /wishlist/:user**- Retrieves wishlisted items by user
* **POST/wishlist/add**- Adds a new item to the wishlist
* **POST/wishlist/delete** :Deletes a item in wishlist
* **GET/deletewishbook/:id** : Deletes wishlisted item by ID
* **GET/aorders** - retrieves orders placed in admin page

**Integration with Frontend**  
The backend communicates with the frontend via RESTful APIs. Key points of integration include:

* **User Authentication:** Tokens are passed between frontend and backend to handle authentication.

**END POINTS:**

* **POST /usersignup** - Register a new user and returns a token
* **POST /userlogin** - Authenticates a user and returns a token.

**Data Fetching:** Frontend components make API calls to fetch necessary data for display and interaction.

**ENDPOINTS**:

**GET /allbooks** - Retrieves all books

**GET /getorders/:id**- Retrieves order by id

**GET /wishlist/:user**- Retrieves wishlisted items by user

**Error Handling and Validation**  
Describe the error handling strategy and validation mechanisms:

* **Error Handling:** Centralized error handling using middleware.

**Example:** In server.js,errors are caught and a JSON response with error message is returned

* **Validation:** Input validation using libraries like Joi or express-validator.

Example: user input such as email and password are validated during signup and login processes

**Security Considerations**  
Outline the security measures implemented:

* **Authentication:** Secure token-based authentication.

Example: JWT(JSON Web Token) is used for creating and verifying tokens during user login and signup

* **Data Encryption:** Encrypt sensitive data at rest and in transit.

Example : Passwords are hashed using bcyrpt before storing in the database