**Full Stack Development with MERN**

**API Development and Integration Report**

|  |  |
| --- | --- |
| Date | 10 July 2024 |
| Team ID | SWTID1719939027 |
| Project Name | Project – Book Nest |
| Maximum Marks | 12 |

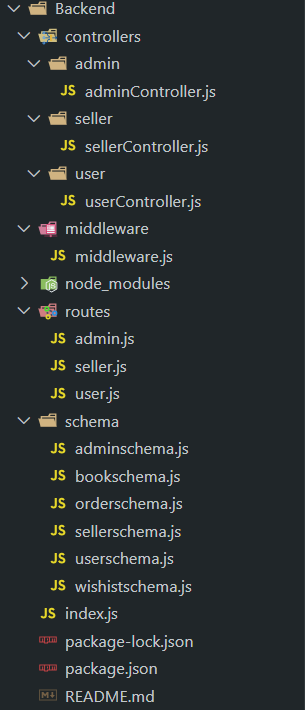
**Project Title:** Book Nest  
**Date:** 10 July 2024  
**Prepared by:** Yavish Sahrawat

**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. **/controllers**
   * Contains functions to handle requests and responses.

• admin

adminController.js

• seller

sellerController.js

• user

userController.js

1. **/schema** : Includes Mongoose schemas and models for MongoDB collections.
   * adminschema.js
   * bookschema.js
   * orderschema.js
   * sellerschema.js
   * userschema.js
   * wishlistschema.js
2. **/routes**
   * Defines the API endpoints and links them to controller functions.

• admin

admin.js

• seller

seller.js

• user

user.js

1. **/middlewares**
   * Custom middleware functions for request processing.

middlewar.js

1. **/config**
   * Configuration files for database connections, environment variables, etc.
   * Index.js

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

* **POST /user/register** - Registers a new user.
* **POST /user/login** - 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 /seller/register** - Registers a new seller.
* **POST /seller/login** - Authenticates a seller and returns a token.

**Seller Management**

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

**Admin API Endpoints**

* **POST /admin/register** - Registers a new admin.
* **POST /admin/login** - Authenticates an admin and returns a token.

**Admin Management**

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

**Books**

* **GET /books** - Retrieves all books.
* **POST /books** - Adds a new book.
* **GET /books/** - Retrieves a book by ID.
* **PUT /books/** - Updates a book by ID.
* **DELETE /books/** - Deletes a book by ID.

**Orders**

* **GET /orders** - Retrieves all orders.
* **POST /orders** - Creates a new order.
* **GET /orders/** - Retrieves an order by ID.
* **PUT /orders/** - Updates an order by ID.
* **DELETE /orders/** - Deletes an order by ID.

**Wishlist**

* **GET /wishlist** - Retrieves all wishlisted items.
* **POST /wishlist** - Adds a new item to the wishlist.
* **GET /wishlist/** - Retrieves a wishlisted item by ID.
* **DELETE /wishlist/** - Removes an item from the wishlist by ID.

**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.
  + **Endpoints**:
    - POST /user/login - Authenticates a user and returns a token.
    - POST /user/signup - Registers a new user and returns a token.
    - POST /user/details - Retrieves user details using a token.
* **Data Fetching**: Frontend components make API calls to fetch necessary data for display and interaction.
  + **Endpoints**:
    - GET /user/books - Retrieves all books.
    - GET /user/orders - Retrieves orders for a specific user.
    - GET /user/wishlist - Retrieves wishlist items for a specific user.

**Error Handling and Validation**

Describe the error handling strategy and validation mechanisms:

* **Error Handling**: Centralized error handling using middleware.
  + Example:
    - In each controller function, errors are caught and a JSON response with success: false and an 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 bcrypt before storing in the database.