**Full Stack Development with MERN**

**API Development and Integration Report**

|  |  |
| --- | --- |
| Date | 17th July 2024 |
| Team ID | SWTID1720068764 |
| Project Name | Book-Store |
| Maximum Marks | 10 Marks |

**Project Title:** **Book Nest** (Online Book Store)  
**Date:** 17th July 2024

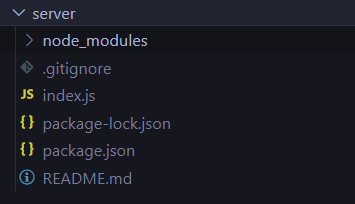
**Team:** Sricharan R, Krishna Vamsi S, Nithin Krishna K, Praneeth P

**Objective**  
The objective of this report is to document the API development progress and key aspects of the backend services implementation for the [Your Project Title] project.

**Technologies Used**

* **Backend Framework:** Node.js with Express.js
* **Database:** MongoDB
* **Authentication:** [e.g., JWT, OAuth]

**Project Structure**  
Provide a screenshot of the backend project structure with explanations for key directories and files.



**Key Directories and Files**

**index.js:** This file contains the entire server setup, middleware configuration, MongoDB connection, and API route definitions**.**

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

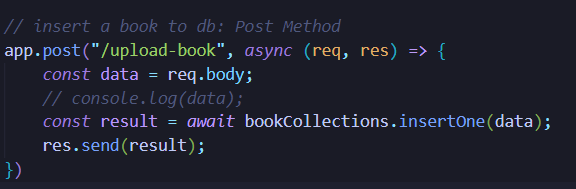
The API provides several endpoints to interact with the book inventory:

**1. Base Endpoint**

* **GET /**
  + Description: Returns a simple "Hello World!" message to verify the server is running.
  + Response: "Hello World!"

**Book Management Endpoints**

* **POST /upload-book**
  + Description: Insert a new book into the database.
  + Request Body: JSON object containing book details (title, author, image URL, category, description, PDF URL).
  + Response: JSON object with the result of the insertion.



* **GET /all-books**
  + Description: Retrieves all books from the database. Can filter by category if the category query parameter is provided.
  + Query Parameters: category (optional)
  + Response: JSON array of books.

A computer code with colorful text

Description automatically generated

* **PATCH /book/**
  + Description: Updates the details of an existing book identified by its ID.
  + Request Body: JSON object containing the updated book details.
  + Response: JSON object with the result of the update.

A screen shot of a computer program

Description automatically generated

* **DELETE /book/**
  + Description: Deletes a book from the database identified by its ID.
  + Response: JSON object with the result of the deletion.

A screen shot of a computer code

Description automatically generated

* **GET /book/**
  + Description: Retrieves the details of a single book identified by its ID.
  + Response: JSON object of the book details.

A computer code on a black background

Description automatically generated

**User Authentication & Management**

### Google Firebase Authentication

User authentication is managed using Google Firebase Authentication. This service provides secure and scalable authentication solutions, including:

* **Sign-In Methods**: Supports various sign-in methods such as email/password, phone authentication, and social providers like Google.
* **User Management**: Handles user registration, login, and password management.
* **Security**: Ensures secure handling of user credentials and session management.

A screenshot of a computer

Description automatically generated

**Integration Details**

Firebase Authentication is integrated into the application to manage user sign-ins and access control. It simplifies the authentication process and provides a robust solution for securing API endpoints.

A computer screen shot of a program

Description automatically generated

**Integration with Frontend**

**Overview**

The backend communicates with the frontend through RESTful APIs, which provide the necessary data and functionality required by the frontend components. This interaction is crucial for dynamic and responsive web applications. Below are the key aspects of this integration:

**Key Points of Integration**

**1. Data Fetching**

Data fetching is the process where frontend components make API calls to retrieve data from the backend. This data can include information such as book details, user information, or any other relevant data needed for display or interaction.

* **API Calls**: Frontend components use HTTP methods like GET, POST, PATCH, and DELETE to interact with the backend. These API calls are typically made using libraries like Axios or the Fetch API.
  + **GET Requests**: Used to retrieve data from the server. For example, fetching a list of all books or details of a specific book.
  + **POST Requests**: Used to send new data to the server. For example, adding a new book to the database.
  + **PATCH Requests**: Used to update existing data on the server. For example, updating book details.
  + **DELETE Requests**: Used to remove data from the server. For example, deleting a book from the database.
* **State Management**: Once data is fetched, it is usually stored in the state of the frontend application. This state is then used to render the user interface. State management libraries like Redux or Context API can be used to handle and share state across components.
* **Error Handling**: Proper error handling is crucial for providing a good user experience. If an API call fails, the front end should handle this gracefully, showing appropriate error messages or fallback content.
* **Loading States**: During data fetching, it’s important to handle loading states to inform users that data is being loaded. This can be achieved through loading spinners or placeholder content.

**2. Authentication and Authorization**

With Google Firebase Authentication integrated, the frontend must handle user authentication and authorization:

* **User Sign-In**: The frontend should include functionality to allow users to sign in using Google or other authentication methods supported by Firebase. Once signed in, the user's authentication state is typically stored in the frontend state or context.
* **Authenticated Requests**: For API requests that require authentication, the frontend must include authentication tokens (e.g., Firebase ID tokens) in the request headers. These tokens verify the user's identity and grant access to protected resources.
* **Session Management**: The frontend should handle user sessions, ensuring users remain signed in across page reloads and navigating to protected routes or components based on their authentication state.

**3. User Interaction**

* **Forms and Data Submission**: User interactions such as filling out forms to add or update book information trigger API calls to send data to the backend. The front end should validate user inputs before sending them to the server.
* **Dynamic Content Updates**: Based on user actions or changes in state, the frontend may need to update the displayed content dynamically. This can be achieved by re-fetching data from the backend or using WebSockets for real-time updates.

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

* **Error Handling:** Centralized error handling using middleware.
* **Validation:** Input validation using libraries like Joi or express-validator.

**Security Considerations:**

Outline the security measures implemented:

**Recommendations**

1. **Environment Variables**: Store sensitive data like MongoDB URI and Firebase configuration in environment variables.
2. **Input Sanitization**: Prevent injection attacks by sanitizing inputs.
3. **Authentication and Authorization**: Firebase Authentication provides robust user management and session control.
4. **HTTPS**: Use HTTPS to encrypt data between the client and server.
5. **Rate Limiting**: Implement rate limiting to prevent DDoS attacks.
6. **CORS**: Configure CORS properly to restrict access to trusted domains only.