****

**Assignment No : 5**

## 5. Create a SPA with React.js that includes multiple components, state management, and routes. Integrate with a public API to fetch and display data Build a RESTful API using Express.js that performs CRUD operations on a resource (e.g., products, users). Include error handling and middleware for logging.

**Objective:**

* To create a RESTful API using Express.js that performs CRUD operations on a resource (e.g., products).
* To demonstrate the use of Express.js for building a simple and efficient backend service that interacts with various HTTP methods (GET, POST, PUT, DELETE).
* To implement middleware for logging requests and error handling, ensuring robust and secure operations.
* To practice handling API requests and responses, managing data in an in-memory array, and showcasing basic error management.

**Theory:**

1. **Express.js and RESTful API Design:** The project employs Express.js to create a RESTful API, adhering to REST principles for structuring URLs and managing resources. CRUD operations are implemented to interact with product resources.
2. **Middleware in Express:** Express.js middleware functions are used to manage requests, responses, and errors. The use() method sets up middleware for logging request details and handling JSON parsing, demonstrating the core structure of Express.js.
3. **Routing and URL Patterns:** The project organizes API routes to handle HTTP requests such as GET, POST, PUT, and DELETE, ensuring a clean and well-organized route system. Each route corresponds to a specific API operation, promoting readability and maintainability.
4. **Error Handling:** Basic error-handling mechanisms are implemented to ensure the API can gracefully handle unexpected inputs or failed operations. Custom error messages are returned for invalid requests, missing resources, or malformed inputs, ensuring that users receive meaningful feedback.
5. **State Management**: Although the project uses an in-memory array to store products (simplified for the purpose of demonstration), the structure allows for easy expansion to integrate database solutions such as MongoDB or PostgreSQL in future enhancements.
6. **Logging:** A logging middleware is used to log incoming requests to the console, demonstrating how to monitor and track API usage in real-time.

**Implementation Details:**

1. **Route Design:** Each CRUD operation is tied to a specific route. POST /products creates a new product, GET /products fetches all products, GET /products/:id fetches a product by its ID, PUT /products/:id updates an existing product, and DELETE /products/:id removes a product. This setup ensures clear, REST-compliant API interaction.
2. **In-Memory Data Storage:** The API stores product data in an array for simplicity. Each product contains an ID, name, and price. The use of in-memory storage simplifies testing and focuses on API structure rather than database integration.
3. **Middleware for Logging and JSON Parsing:** Two middleware functions are used: one for logging the request method and URL, and another for parsing incoming JSON data. This ensures that all requests are properly formatted and that developers have visibility into API requests.
4. **Error Handling:** The API includes error handling at both the route and global levels. If a product is not found by ID, the API returns a 404 Not Found error. Invalid or incomplete input results in a 400 Bad Request response, showcasing how to handle client-side errors.
5. **User Interface (UI):** Although this project doesn’t implement a UI, it is designed to interact with any frontend that consumes its API. Potential UIs could include web applications built with React, Angular, or Vue.js.

**Conclusion:**

1. **Express.js Simplicity:** This project demonstrates Express.js's simplicity and effectiveness in creating RESTful APIs. It highlights how to build lightweight, scalable APIs with clear route structures and middleware integration.
2. **API Design and State Management:** The application showcases how to effectively design RESTful APIs that handle CRUD operations. While the project uses in-memory data, it sets the stage for more advanced state management techniques, such as database integration.
3. **Modular and Scalable Structure:** The project promotes a modular structure, making it easy to extend. Future enhancements could include the introduction of more resources (e.g., Users, Orders), additional middleware for authentication, or integration with a database for persistent data storage.
4. **Error Handling and Logging:** Error handling plays a critical role in maintaining API robustness, and the project demonstrates practical approaches to managing common API issues. Logging ensures that the API remains transparent, aiding in debugging and monitoring.
5. **Learning Outcomes:** This project provides hands-on experience in building backend services with Express.js, managing data via API endpoints, and implementing basic middleware. It’s a solid introduction to building real-world web services using Node.js and Express.
6. **Future Enhancements:** Possible improvements include integrating a database (e.g., MongoDB, MySQL) for persistent storage, adding user authentication and authorization, implementing advanced validation using libraries like Joi or express-validator, and integrating request logging with a service like Winston or Morgan for production-ready logging.

**GitHub Repo with Source Code :**

[FSDL Assn 5](https://github.com/RudradevArya/FSDL/tree/main/?tab=readme-ov-file#assn-5)

<https://github.com/prathameshparsai18/prathamesh-FSDL/upload/main>

**Output (Screenshots):**

1. **A screenshot of a computer

   Description automatically generatedGET Request**



1. **POST Request**



A screenshot of a computer

Description automatically generated

1. A screenshot of a computer

   Description automatically generated**PUT Request**



1. **DEL Request**



A screenshot of a computer

Description automatically generated