Lab Exercises -3

(React Programs on API, Error Handling, Nested Routes, Express)

[Execute the below programs and upload a Single PDF with the given problem statements, Codes, and outputs for all the execution.]

1. **Building a Blog API**
   You are tasked with creating a basic blogging platform API using Express.js. Implement the following:
   - A route to fetch all blog posts (GET /api/posts).
   - A route to fetch a single blog post by its ID (GET /api/posts/:id).
   - A route to create a new blog post (POST /api/posts) with title and content in the request body.
   - A route to update a blog post by ID (PUT /api/posts/:id).
   - A route to delete a blog post by ID (DELETE /api/posts/:id).

Ensure that:

- The API returns appropriate HTTP status codes (e.g., 200, 201, 404).
- Error handling is implemented for invalid IDs or missing data in the request body.
- Use middleware to log all incoming requests with their method and URL.

2. **E-commerce Product Management**
   You are developing an API for managing products in an e-commerce application. Implement the following features:
   - A route to list all products (GET /products). Support query parameters like ?category=electronics to filter products by category.
   - A route to fetch details of a specific product by its ID (GET /products/:id).
   - A route to add a new product (POST /products) with fields like name, price, and category in the request body. Validate that all required fields are provided.

- Use middleware to check if the Content-Type header is set to application/json for POST requests. If not, return an error response.

3. **User Authentication System**
   Build an Express.js-based API for user authentication with the following routes:

   - POST /register: Accepts user details (name, email, password) in the request body and registers a new user. Validate that all fields are provided and return an error if any field is missing.
   - POST /login: Accepts email and password in the request body and checks if they match a registered user. Return success or failure accordingly.
   - GET /profile: Returns the profile of the logged-in user. Use middleware to simulate authentication by checking if a valid token is passed in the headers (e.g., Authorization: Bearer <token>). Return an error if no token is provided.

4. **Error Handling in an Order Management System**
   Create an API for managing customer orders with the following routes:

   - GET /orders: Fetches all orders. If no orders exist, return a message saying "No orders found."
   - POST /orders: Creates a new order with details like customer name, product ID, and quantity in the request body. Validate that all fields are provided; otherwise, return an error response with status code 400.
   - Add global error-handling middleware that catches unhandled errors and returns a JSON response with status code 500 and an error message like "Internal Server Error."

5. **Nested Routes for Library Management**
   You are designing an API for managing books and their authors in a library system. Implement nested routes using express.Router() as follows:

- /authors (GET): Lists all authors.
- /authors/:authorId/books (GET): Lists all books written by a specific author based on their ID.
- /authors/:authorId/books/:bookId (GET): Fetches details of a specific book written by the author.

Requirements:

- Use modular routing (express.Router()) for authors and books routes separately and mount them in your main application file using app.use().
- Implement middleware that logs the current timestamp whenever any of these routes are accessed.

6. **RESTful API for Task Management**
   Develop a task management API where users can manage their daily tasks with the following features:
   - GET /tasks: Fetches all tasks with support for filtering tasks based on their status using query parameters (e.g., ?status=completed).
   - POST /tasks: Adds a new task with fields like title, description, and status (default status should be "pending"). Validate that title and description are provided; otherwise, return an error response.
   - PATCH /tasks/:id: Updates the status of a task by its ID (e.g., change from "pending" to "completed"). Return an error if the task ID does not exist.
7. **Middleware for Logging and Authentication**
   Create an Express.js application where you implement middleware for logging and authentication:
   - **Middleware 1**: Logs every incoming request's method, URL, and timestamp before passing control to the next middleware or route handler. This should apply globally across all routes.
   - **Middleware 2:** Checks if an API key is passed as part of query parameters (e.g., /api/resource?apiKey=12345). If no API key is provided or it’s invalid, return an error response with status code 401 ("Unauthorized"). Apply this middleware only to routes under /api.

****