**Experiment 9: Creating a REST API with Node.js and Express**

**Aim:** Creating a REST API with Node.js and Express (Node.js, Express, and MongoDB)

**Learning Objective:**

Gain the ability to design and implement a RESTful API using Node.js, Express, and MongoDB, enabling efficient data management and interaction with client applications.

**Tools Used:**

* Node js
* Express js
* MongoDB
* VS Code
* Thunder Client

**Theory:**

**1. Understanding RESTful APIs**

REST (Representational State Transfer) is an architectural style that defines a set of constraints for building web services. It is based on standard HTTP methods, enabling clients to interact with servers through stateless communication. Key principles of RESTful APIs include:

* **Resources**: Everything is considered a resource (e.g., users, products). Each resource is identified by a unique URI (Uniform Resource Identifier).
* **Statelessness**: Each request from the client to the server must contain all the information needed to understand and process the request.
* **HTTP Methods**: Common methods include:
  + **GET**: Retrieve data from the server.
  + **POST**: Create new resources on the server.
  + **PUT**: Update existing resources.
  + **DELETE**: Remove resources from the server.
* **JSON Format**: Data is typically exchanged in JSON (JavaScript Object Notation) format due to its lightweight and easy-to-read structure.

**2. Node.js Overview**

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It allows developers to use JavaScript for server-side programming, enabling asynchronous event-driven architecture, which is particularly useful for building scalable applications. Key features include:

* **Non-blocking I/O**: Node.js can handle multiple requests simultaneously without being blocked, which improves performance.
* **NPM (Node Package Manager)**: A vast ecosystem of libraries and frameworks that can be easily integrated into applications.

**3. Express Framework**

Express is a minimal and flexible web application framework for Node.js that simplifies the process of building web applications and APIs. Key features include:

* **Middleware Support**: Functions that execute during the request-response cycle, allowing for easier handling of requests (e.g., logging, authentication).
* **Routing**: Simplified routing mechanism to define how an application responds to client requests.
* **Error Handling**: Centralized error-handling middleware for managing errors in a consistent manner.

**4. MongoDB Overview**

MongoDB is a NoSQL database that stores data in flexible, JSON-like documents. It allows for dynamic schema design, making it easier to work with unstructured data. Key characteristics include:

* **Document-Oriented Storage**: Data is stored in documents (BSON format), which can have varying fields and data types.
* **Scalability**: Built to scale horizontally, allowing for sharding and replication for large datasets.
* **Mongoose**: An Object Data Modeling (ODM) library for MongoDB that provides a schema-based solution to model application data, simplifying the interaction with the database.

**5. Building a REST API with Node.js, Express, and MongoDB**

Creating a REST API involves the following steps:

1. **Setup**: Install necessary packages, such as Express and Mongoose, and set up the project structure.
2. **Define Data Models**: Create Mongoose schemas that represent the data structure in MongoDB.
3. **Create Routes**: Define API endpoints that correspond to various HTTP methods for handling different requests.
4. **Implement Business Logic**: Connect routes to the appropriate business logic, such as data retrieval, creation, updating, and deletion.
5. **Error Handling**: Implement error-handling middleware to manage potential issues and send appropriate responses to clients.
6. **Testing**: Use tools like Thunder Client or Postman to test the API endpoints and ensure they function correctly.

**Implementation/Code:**







