Course Name:	**Web Programming Laboratory 116U40L501**	**Semester:**	**V**
Date of Performance:	**24 / 10 / 2024**	**Batch No:**	**B - 1**
Faculty Name:	**Prof. Madhura Pednekar**	**Roll No:**	**16014022050**
Faculty Sign & Date:		**Grade / Marks:**	**____/ 25**

Experiment No: 8
Title: To study RESTFUL Api

Aim and Objective of the Experiment:

To study a RESTful API server in Express and Node.js., implementation + Testing application using postman/Thurderclient.

COs to be achieved:

Study interactive web content using RESTFUL api and node js and express.

Problem Statement:

1. **What is a REST API?**
2. **Popular HTTP methods?**
3. **What is Node.js?**
4. **Why use Node.js to build your REST API?**
5. **Prerequisites?**
6. **How to set up a Node.js app?**
7. **How to create a user management API with Node.js and Express?**

Solutions:

1. REST (Representational State Transfer) is an architectural style for designing networked applications.
   It relies on a stateless, client-server, cacheable communications protocol, typically HTTP.
   A REST API allows for interaction with RESTful web services and provides a way to request and manipulate resources over the internet using HTTP methods (GET, POST, PUT, DELETE, etc.).

2. Few of the popular methods of HTTP include:
   - GET: Retrieve data from the server.
   - POST: Send data to the server (e.g., create a new resource).
   - PUT: Update or replace existing resources on the server.
   - PATCH: Partially update an existing resource.
   - DELETE: Remove a resource from the server.

- HEAD: Similar to GET but only retrieves the headers.
- OPTIONS: Describes the communication options for the target resource.

3. Node.js is a JavaScript runtime built on Chrome's V8 engine, designed to build fast, scalable server-side applications.
   It allows developers to write JavaScript for both the server and client sides.
   Node.js is single-threaded but uses non-blocking I/O, making it suitable for real-time applications like chat apps, APIs, etc.

4. Node.js is preferable for REST API because:
   - Non-blocking I/O: Efficient handling of concurrent requests, which makes it ideal for I/O-heavy tasks like API calls.
   - Scalability: Node.js scales well under heavy loads, as it handles multiple requests asynchronously.
   - JavaScript Everywhere: JavaScript can be used both for front-end and back-end, makingdevelopment faster and more consistent.
   - Rich Ecosystem: The npm (Node Package Manager) provides access to thousands of librariesand tools for rapid API development.

5. Basic knowledge of JavaScript and asynchronous programming.
   Installed versions of Node.js and npm.
   Familiarity with HTTP and RESTful architecture.
   Basic understanding of Express.js (web framework for Node.js).

6. **Install Node.js and npm**: Download and install Node.js from nodejs.org.

   Create a New Project Directory:

```
mkdir my-node-app
cd my-node-app
```

   Initialize the Project:

```
npm init -y
```

   This creates a package.json file, which will store metadata and dependencies for the project.

   Install Express.js:

```
npm install express
```

   Create an Entry Point (e.g., app.js):

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Hello World');
});

app.listen(3000, () => {
```

```
  console.log('Server is running on port 3000');
});
```

Run the App:

node app.js

Your Node.js server will be running at http://localhost:3000.

7. **Set up Express and Dependencies**:

npm install express body-parser

Install body-parser to handle incoming request bodies.

Create Routes for User Management (app.js):

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();

app.use(bodyParser.json());

// Dummy database
let users = [];

// Get all users
app.get('/users', (req, res) => {
res.json(users);
});

// Create a new user
app.post('/users', (req, res) => {
const newUser = req.body;
users.push(newUser);
res.status(201).json(newUser);
});

// Get a single user by ID
app.get('/users/:id', (req, res) => {
const userId = req.params.id;
  const user = users.find(u => u.id === userId);
  if (user) {
    res.json(user);
  } else {
    res.status(404).json({ message: 'User not found' });
  }
});

// Update a user by ID
app.put('/users/:id', (req, res) => {
```

```
  const userId = req.params.id;
  const userIndex = users.findIndex(u => u.id === userId);
  if (userIndex !== -1) {
    users[userIndex] = { ...users[userIndex], ...req.body };
    res.json(users[userIndex]);
  } else {
    res.status(404).json({ message: 'User not found' });
  }
});

// Delete a user by ID
app.delete('/users/:id', (req, res) => {
const userId = req.params.id;
  users = users.filter(u => u.id !== userId);
  res.status(204).end();
});

// Start server
app.listen(3000, () => {
  console.log('User management API running on port 3000');
});
```

Post Lab Subjective / Objective type Questions:

1. **Advantages of RESTFUL Api.**
   - Statelessness: REST APIs are stateless, meaning each request contains all the information needed for processing. This improves scalability, as servers don't need to store session information.
   - Flexibility: REST APIs can handle multiple types of calls (GET, POST, PUT, DELETE, etc.), return different data formats (JSON, XML), and be consumed by various clients (web, mobile, IoT).
   - Scalability: RESTful services are easy to scale, as they follow a stateless architecture. Servers can handle requests independently, making load balancing and horizontal scaling easier.
   - Performance: REST APIs often rely on caching (e.g., using HTTP caching headers) to improve response times and reduce server load.
   - Uniform Interface: REST uses a consistent and predefined set of operations (HTTP methods), making the API easier to understand and interact with.
   - Decoupled: REST APIs allow a separation between the client and server, enabling independent development and scaling of each component.

2. Difference between MongoDB and MySQL.

Feature	MongoDB	MySQL
Type	NoSQL (Document-oriented) database	SQL (Relational) database
Data Structure	Stores data as JSON-like documents (BSON) with dynamic schemas (flexible)	Stores data in tables with rows and columns (rigid schema)
Schema	Schema-less (flexible, documents can have different fields)	Schema-based (fixed structure, predefined columns)
Query Language	MongoDB Query Language (MQL)	SQL (Structured Query Language)
Transactions	Supports multi-document ACID transactions (starting from version 4.0)	Supports ACID transactions natively
Performance	Fast for unstructured or semi-structured data, especially when horizontal scaling is required	Fast for structured data with complex relationships (joins)
Joins	Does not support joins natively (requires embedding or linking documents)	Supports joins to combine data from multiple tables

Conclusion:

Successfully learnt and implemented all the topics explained above in our mini project.

Signature of faculty in-charge with Date: