**Experiment Number: 5 Date: 24-02-2025**

# **Demonstrate RESTful Endpoints using Express and HTTP Methods**

## **PRE LAB EXERCISE**

### **Objective:**

* Understand RESTful APIs and their importance in web development.
* Learn the HTTP methods (GET, POST, PUT, DELETE) and their purpose.
* Set up a basic Express.js server for handling API requests.

### **QUESTIONS:**

1. **What is a RESTful API, and why is it important?**

**RESTful API**: A RESTful API follows REST principles, using HTTP methods for communication. It is important because it enables scalable, stateless, and efficient web services.

1. **List the common HTTP methods used in RESTful APIs.**

 **GET** (Retrieve data)

 **POST** (Create data)

 **PUT** (Update data)

 **DELETE** (Remove data)

 **PATCH** (Partial update)

1. **What is the difference between GET, POST, PUT, and DELETE methods?**

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Purpose** | **Request Example** | **Use Case** |
| **GET** | Retrieve data from the server | GET /users | Fetching all users or a specific user |
| **POST** | Send data to the server to create a resource | POST /users (with JSON body) | Adding a new user to the database |
| **PUT** | Update an existing resource (or create if it doesn’t exist) | PUT /users/1 (with JSON body) | Updating user details (e.g., name, age) |
| **DELETE** | Remove a resource from the server | DELETE /users/1 | Deleting a user from the database |

1. **How does Express.js help in building RESTful APIs?**

Express.js is a **lightweight and flexible Node.js framework** that simplifies building **RESTful APIs**. It provides essential features like **routing, middleware support, request handling, and error management**, making it an ideal choice for backend development.

1. **What is the significance of middleware in handling API requests?**

Middleware plays a **crucial role** in Express.js (and other frameworks) for processing API requests before they reach the final route handler. It acts as a **pipeline** where requests pass through different layers, allowing modifications, validations, and additional functionalities.

## **IN LAB EXERCISE**

### **Objective:**

* Implement a RESTful API using Express.js.
* Handle different HTTP methods (GET, POST, PUT, DELETE).
* Test API endpoints using **Postman** or **cURL**.

### **Resources Required:**

* Node.js installed
* Express.js package (npm install express)
* Postman or cURL for API testing

### **Step 1: Install Required Packages**

Open your terminal and run:

npm init -y

npm install express

### **Step 2: Implement RESTful API (**server.js**)**

const express = require('express');

const app = express();

app.use(express.json()); // Middleware to parse JSON

// Sample data

let products = [

    { id: 1, name: "Laptop", price: 1000 },

    { id: 2, name: "Phone", price: 500 }

];

// GET request - Fetch all products

app.get('/products', (req, res) => {

    res.json(products);

});

// GET request - Fetch product by ID

app.get('/products/:id', (req, res) => {

    const product = products.find(p => p.id == req.params.id);

    if (!product) return res.status(404).send('Product not found');

    res.json(product);

});

// POST request - Add a new product

app.post('/products', (req, res) => {

    const newProduct = { id: products.length + 1, name: req.body.name, price: req.body.price };

    products.push(newProduct);

    res.status(201).json(newProduct);

});

// PUT request - Update an existing product

app.put('/products/:id', (req, res) => {

    const product = products.find(p => p.id == req.params.id);

    if (!product) return res.status(404).send('Product not found');

    product.name = req.body.name;

    product.price = req.body.price;

    res.json(product);

});

// DELETE request - Remove a product

app.delete('/products/:id', (req, res) => {

    products = products.filter(p => p.id != req.params.id);

    res.json({ message: 'Product deleted' });

});

// Start server

app.listen(3000, () => console.log('Server running on port 3000'));

### **Step 3: Test the API using Postman or cURL**

#### ****GET All Products:****

* **URL:** http://localhost:3000/products
* **Method:** GET
* **Expected Response:**

[

{ "id": 1, "name": "Laptop", "price": 1000 },

{ "id": 2, "name": "Phone", "price": 500 }

]

A screenshot of a computer

AI-generated content may be incorrect.

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**

#### ****GET Product by ID:****

* **URL:** http://localhost:3000/products/2
* **Method:** GET

**Expected Response:** {"id": 2, "name": "Phone", "price": 500}

**A screenshot of a computer

AI-generated content may be incorrect.**

#### ****POST New Product:****

* **URL:** http://localhost:3000/products
* **Method:** POST
* **Body (JSON):**

{"name": "Tablet", "price": 600}

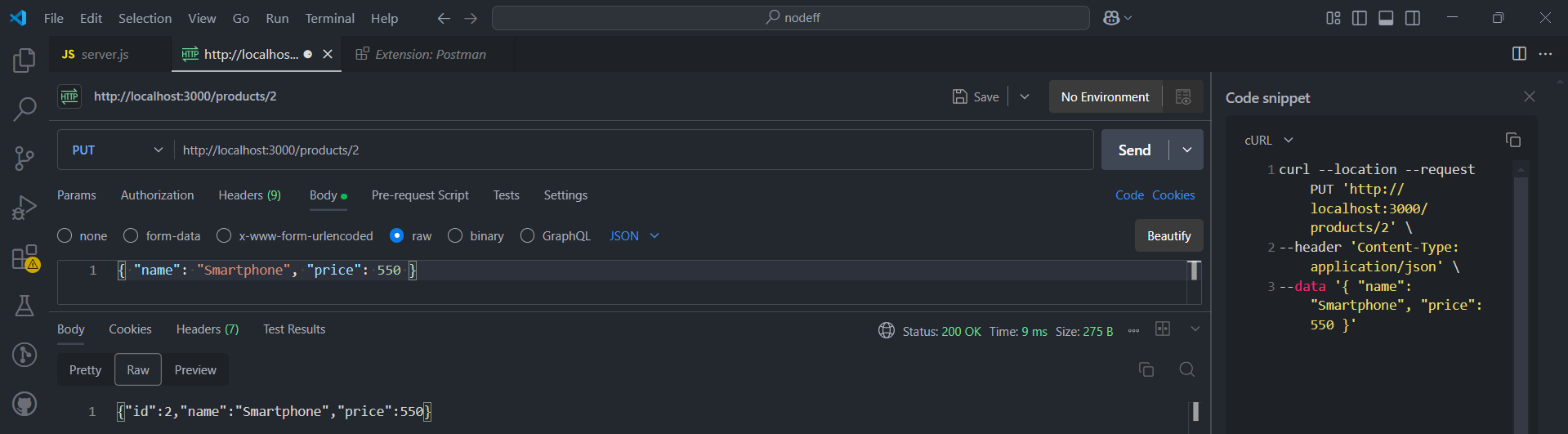
* **Expected Response:** { "id": 3, "name": "Tablet", "price": 600 }

A screenshot of a computer

AI-generated content may be incorrect.

#### ****PUT Update Product:****

* **URL:** http://localhost:3000/products/2
* **Method:** PUT
* **Body (JSON):** { "name": "Smartphone", "price": 550 }
* **Expected Response:** { "id": 2, "name": "Smartphone", "price": 550 }



#### ****DELETE a Product:****

* **URL:** http://localhost:3000/products/1
* **Method:** DELETE
* **Expected Response:** { "message": "Product deleted" }

A screenshot of a computer

AI-generated content may be incorrect.

## **POST LAB EXERCISE**

### **Objective:**

* Verify and analyze API responses.
* Enhance the API by adding error handling and validation.
* Implement authentication and authorization.

### **QUESTIONS:**

1. **What is the purpose of using req.params in Express?**

In Express.js, req.params is an object that contains route parameters (dynamic values) from the URL. It is used to capture values in the URL and pass them to your route handlers.

**Example**:

If you have a route like /books/:id, the :id is a route parameter. You can access it using req.params.id.

app.get('/books/:id', (req, res) => {

const bookId = req.params.id; **// Extracts the id from the URL**

res.send(`Book ID is: ${bookId}`);

});

1. **How can we validate incoming request data in a RESTful API?**

* **Manually in route handlers** - Check the request body or parameters using conditionals.
* **Using Validation Libraries**: You can use libraries like Joi, express-validator, or celebrate to validate and sanitize incoming data.

1. **How can authentication be integrated into a RESTful API?**

 **JWT**: Ideal for stateless authentication, often used for mobile apps and SPAs.

 **OAuth**: Used for delegating access, typically with third-party providers like Google, Facebook, etc.

 **API Key**: Used for simpler services where the user must send a unique key for every request.

 **Session-based Authentication**: Traditional server-side authentication, where the session ID is stored on the server, and a cookie is used to track the session.

1. **How would you modify this API to store data in a MongoDB database?**

Use **Mongoose** to connect Express to MongoDB, define schemas, and replace in-memory storage with database operations (find, save, update, etc.).

1. **What are the benefits of using express.json() middleware in handling API requests?**

### The **express.json()** middleware is used to parse incoming request bodies that are in **JSON** format. It automatically converts the raw data in the request body into a JavaScript object.

### **Benefits:**

### **Automatic Parsing**: It automatically parses the JSON data and makes it available in req.body. This saves you from manually parsing JSON data in every request handler.

### **Error Handling**: If the incoming data is not valid JSON, it sends a 400 (Bad Request) response automatically.

### **Convenience**: Helps streamline the process of handling JSON data in POST, PUT, or PATCH requests without extra code to handle the parsing logic.

### **Enhancements (For Further Learning)**

* Implement **input validation** using express-validator.
* Connect the API with **MongoDB** using mongoose.
* Add **authentication** using JWT.

### **Steps:**

* 1. Install the required packages:

**npm install express mongoose jsonwebtoken bcryptjs express-validator dotenv**

* 1. Code

const express = require('express');

const mongoose = require('mongoose');

const jwt = require('jsonwebtoken');

const bcrypt = require('bcryptjs');

const { body, validationResult } = require('express-validator');

require('dotenv').config();

// Initialize the Express app

const app = express();

app.use(express.json()); // Middleware to parse JSON bodies

// MongoDB connection

mongoose.connect('mongodb://localhost:27017/userDB', {

  useNewUrlParser: true,

  useUnifiedTopology: true,

})

  .then(() => console.log('Connected to MongoDB'))

  .catch((err) => console.log('Error connecting to MongoDB:', err));

// User model

const User = mongoose.model('User', new mongoose.Schema({

  name: String,

  email: { type: String, unique: true },

  password: String,

}));

// Middleware to check JWT token

function authenticateToken(req, res, next) {

  const token = req.header('Authorization')?.split(' ')[1];

  if (!token) return res.status(403).json({ message: 'Access denied' });

  jwt.verify(token, process.env.JWT\_SECRET, (err, user) => {

    if (err) return res.status(403).json({ message: 'Invalid token' });

    req.user = user;

    next();

  });

}

// POST request - Register a new user

app.post('/register', [

  body('name').isString().withMessage('Name must be a string'),

  body('email').isEmail().withMessage('Please provide a valid email'),

  body('password').isLength({ min: 6 }).withMessage('Password must be at least 6 characters long'),

], async (req, res) => {

  // Validate request data

  const errors = validationResult(req);

  if (!errors.isEmpty()) {

    return res.status(400).json({ errors: errors.array() });

  }

  const { name, email, password } = req.body;

  // Check if user already exists

  const existingUser = await User.findOne({ email });

  if (existingUser) {

    return res.status(400).json({ message: 'User already exists' });

  }

  // Hash the password

  const hashedPassword = await bcrypt.hash(password, 10);

  // Save the new user

  const user = new User({ name, email, password: hashedPassword });

  await user.save();

  res.status(201).json({ message: 'User registered successfully' });

});

// POST request - Login user and return JWT

app.post('/login', [

  body('email').isEmail().withMessage('Please provide a valid email'),

  body('password').isLength({ min: 6 }).withMessage('Password is required'),

], async (req, res) => {

  // Validate request data

  const errors = validationResult(req);

  if (!errors.isEmpty()) {

    return res.status(400).json({ errors: errors.array() });

  }

  const { email, password } = req.body;

  // Find user by email

  const user = await User.findOne({ email });

  if (!user) return res.status(400).json({ message: 'User not found' });

  // Check if password matches

  const isMatch = await bcrypt.compare(password, user.password);

  if (!isMatch) return res.status(400).json({ message: 'Invalid credentials' });

  // Create JWT token

  const token = jwt.sign({ id: user.\_id, name: user.name }, process.env.JWT\_SECRET, { expiresIn: '1h' });

  res.json({ token });

});

// Protected route - Get user profile (requires JWT)

app.get('/profile', authenticateToken, async (req, res) => {

  // Fetch user data from database

  const user = await User.findById(req.user.id);

  if (!user) return res.status(404).json({ message: 'User not found' });

  res.json({

    name: user.name,

    email: user.email,

  });

});

// Start the server

app.listen(3000, () => {

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

});

3.Create a .env file with:

JWT\_SECRET=your\_secret\_key

**A black and white background

AI-generated content may be incorrect.**

### **Run the API**

**node jwt.js**

**register**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Login**

**A screenshot of a computer

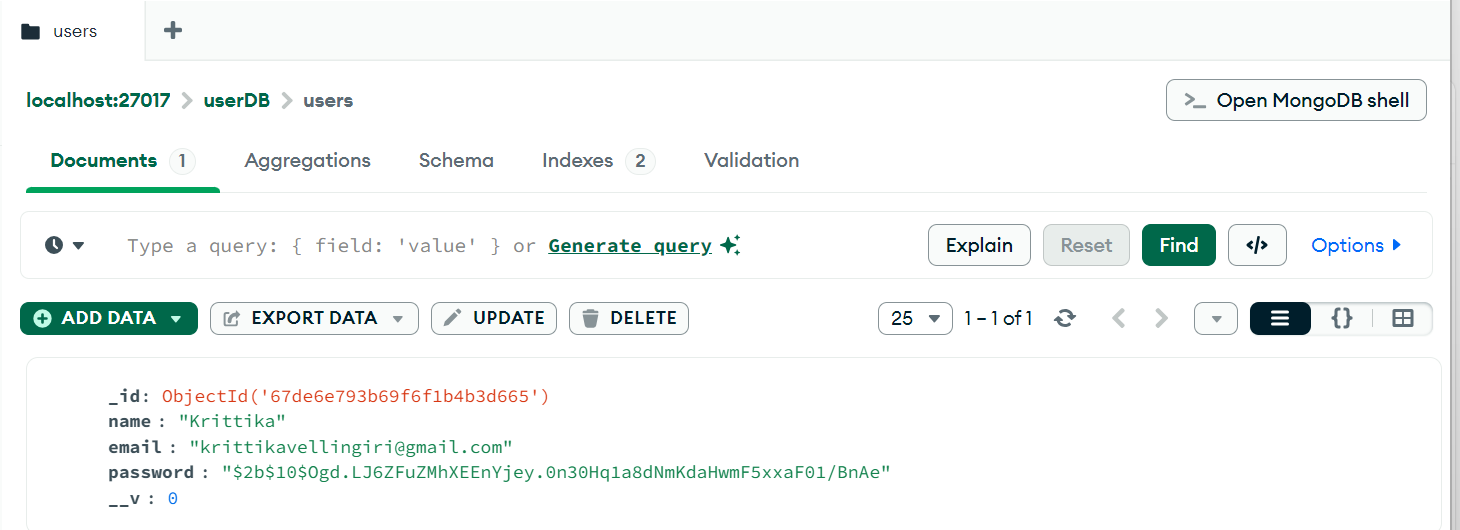
AI-generated content may be incorrect.**

**Input validation**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Mongodb connection**

****

**ASSESSMENT PATTERN.**

|  |  |  |
| --- | --- | --- |
| **Description** | **Max Marks** | **Marks Awarded** |
| Pre Lab Exercise | **5** |  |
| In Lab Exercise | **10** |  |
| Post Lab Exercise | **5** |  |
| Viva | **10** |  |
| **Total** | **30** |  |
| **Faculty Signature** | |  |