WebServices,Restful Api with Js

Index.js

This code is a basic **RESTful API** built using **Express.js**. It performs **CRUD** operations on a list of users. Let's break down the code and its functionality step by step:

1. Setting Up Express

```
const express = require('express');
const app = express();
const PORT = 3000;
```

- **express**: A Node.js framework for building web applications and APIs.
- **app**: An instance of an Express application.
- **PORT**: The port number where the server will listen for requests.

2. Middleware to Parse JSON Requests

```
app.use(express.json());
```

- **app.use(express.json()):** This middleware parses incoming requests with JSON payloads. It enables the server to handle JSON data in request bodies.

3. Sample Data

```
let users = [
  { id: 1, name: "Vijay Kumar" },
  { id: 2, name: "Priya Jhosi" }
];
```

- **users:** A list (array) of user objects. Each user has an **id and a name**. This is a simple **in-memory data store** used for demonstration purposes.

4. GET /users - Retrieve All Users

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

- This route handles **GET** requests to /users.
- It responds with a JSON array of all user objects.

5. GET /users/

- Retrieve a User by ID

```
// GET /users/:id - Retrieve a user by ID
app.get('/users/:id', (req, res) => {
  const user = users.find(u => u.id === parseInt(req.params.id));
  if (!user) return res.status(404).send("User not found");
  res.json(user);
});
```

- **req.params.id**: Extracts the id parameter from the request URL **(e.g., /users/1).**
- **users.find(...):** Searches for a user with the matching id.
- **404 Status Code:** If no user is found, it returns a **404 Not Found** error.
- If a user is found, it returns the user object as **JSON**.

6. POST /users - Create a New User

```
// POST /users - Create a new user
app.post('/users', (req, res) => {
  const newUser = {
    id: users.length + 1,
    name: req.body.name
  };
  users.push(newUser);
  res.status(201).json(newUser);
});
```

- **POST Request**: Adds a new user to the list.
- req.body.name: Gets the name from the request body **(JSON payload).**
- **id: users.length + 1:** Assigns a new id based on the current length of the users array.
- **users.push(newUser):** Adds the new user to the array.
- **201 Status Code:** Indicates that a new resource has been created successfully.
- Returns the newly created user as **JSON.**

7. PUT /users/

- Update a User's Name

```
// PUT /users/:id - Update a user's name
app.put('/users/:id', (req, res) => {
  const user = users.find(u => u.id === parseInt(req.params.id));
  if (!user) return res.status(404).send("User not found");
  user.name = req.body.name;
  res.json(user);
});
```

- **PUT Request**: Updates an existing user's name.
- **req.params.id:** Gets the user id from the URL.
- If the user is not found, it returns a **404 Not Found** error.
- **user.name = req.body.name:** Updates the name property of the user.
- Returns the updated user object as **JSON**.

8. DELETE /users/

- Delete a User

```
// DELETE /users/:id - Delete a user
app.delete('/users/:id', (req, res) => {
  users = users.filter(u => u.id !== parseInt(req.params.id));
  res.send("User deleted");
});
```

- **DELETE Request**: Removes a user from the list.
- **users.filter(...):** Filters out the user whose id matches the given id.
- Returns a message **"User deleted"** as a response.

9. Starting the Server

```
// Start the server
app.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}`);
});
```

- **app.listen(PORT)**: Starts the Express server on the specified port **(3000).**
- The **callback** function logs a message indicating the server is running and the URL where it can be accessed.

Summary

- This API supports basic **CRUD** operations:

- **Create**: Add a new user **(POST /users).**

  http://localhost:3000/users/

  json data

  ```
  {
      "id":"3",
      "name": "Akash Kumar"

  }
  ```

- **Read**: Get all users (GET /users) or a specific user by ID **(GET /users/:id). All Records- http://localhost:3000/users/**

  With Id: http://localhost:3000/users/1

- **Update**: Modify a user's name **(PUT /users/:id).**

  JSON Data:

```
{
        "id":"3",
        "name": "Ajay Kumar"

}
```

- **Delete**: Remove a user **(DELETE /users/:id).**

  http://localhost:3000/users/3

- **Express Middleware** is used to parse JSON data.
- Data is stored in-memory using a simple array (users), so changes are not persistent and will reset when the server restarts.

How to Test

1. **Install Dependencies**: Run npm install express.
   - **npm install express**
2. **Start Server**: Run node <filename>.js.
   - **node index.js**
3. **Use a tool like Postman or curl** to send requests to:
   - **http://localhost:3000/users**
   - **http://localhost:3000/users/1**
   - **And other endpoints for testing create, update, and delete operations.**

This is a typical setup for building and testing **RESTful APIs with Express.js.**