Lecture-2 Express and CRUD operation

Agenda

- Review of last class
- Installation : express , postman ,
- Intro to express and res.json
- Get req with Express
- Error handling in express
- Template routes in express
- POST req in express

Setup of a Nodejs Project with Express

- npm init -y
- npm i express nodemon

Introduction to Express.js

Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for building web applications and APIs. It simplifies common tasks like routing, handling HTTP requests, and managing middleware

1. Basic Setup:

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

2. Creating Routes:
   Define routes for different HTTP methods and URLs:

```
app.get('/hello', (req, res) => {
  const jsonResponse = {
    message: 'Hello, you accessed the GET route!',
    timestamp: Date.now()
  };
```

```
  res.json(jsonResponse);
});
```

3. Start the server

```
const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

Nodemon

Nodemon is a utility that monitors changes in your Node.js application and automatically restarts the server. It's useful during development to save time and effort when making changes to the code.

Using Nodemon

1. After installing Nodemon as dev dependecies

```
npm install -D nodemon
```

2. **Setting Up `package.json` Scripts**: Within `package.json`, you can define a script that runs your Node.js application using `nodemon`.

```
`"scripts": {   "dev": "nodemon your_app_file.js" }`
```

3. **Running Your Application with Nodemon**: To start your application using `nodemon`, you'd execute the script defined in `package.json` via the terminal:

```
npm run dev
```

Handling GET and POST Requests with Express

Express.js provides methods for handling different types of HTTP requests like GET, POST, PUT, DELETE, etc.

GET Request

We've seen an example of handling a GET request earlier. To handle a GET request for a specific route:

```
app.get('/hello', (req, res) => {   // Handle the GET request });
```

POST Request

Handling a POST request involves parsing incoming data. Here's an example:

```
app.post('/submit', (req, res) => {
    const formData = req.body;   // Process the submitted data
});
```

However, to handle POST requests with Express, Express's built-in `express.json()` middleware to parse incoming request bodies.

Error Handling in Express Using Try and Catch

Express allows you to handle errors gracefully by using `try` and `catch` blocks in your route handlers or middleware.

Example of Error Handling

```
app.get('/route', async (req, res) => {
    try {     // Code that might throw an error
        const result = await someAsyncOperation();
        res.json(result);
    } catch (error) {     // Handle the error gracefully
        res.status(500).json({ error: error.message });
    } });
```

In this example, an asynchronous operation `someAsyncOperation` is attempted within a `try` block. If an error occurs, it's caught in the `catch` block, and a 500 status along with an error message is sent as the response.

Template Routes

Template routes in Express help organize and structure your application's endpoints, making it easier to manage and perform actions on different resources within your application.

Example of getById Route

Assuming you have a collection of items and you want to retrieve a specific item by its ID:

Express Route Setup:

```
// Assume you have an array of items (for demonstration purposes)
const items = [   { id: 1, name: 'Item 1' },   { id: 2, name: 'Item 2' },   {
id: 3, name: 'Item 3' } ];
const express = require('express');
const app = express();
// GET route to fetch item by ID
app.get('/items/:id', (req, res) => {
const itemId = parseInt(req.params.id); // Extract ID from URL parameter
const foundItem = items.find(item => item.id === itemId);
if (!foundItem) {
return res.status(404)
.json({ error: 'Item not found' });
}else{
res.json(foundItem); // Respond with the found item

}

});  // Start the server
const PORT = 3000; app.listen(PORT, () => {   console.log(`Server running on
port ${PORT}`); });
```

Usage and Explanation:

- The route `/items/:id` is defined to handle GET requests where `:id` is a placeholder for the item's unique identifier.
- Inside the route handler function, `req.params.id` extracts the `id` parameter from the URL.

- The `find` method is used to search for the item with the corresponding ID within the `items` array.
- If the item is found, it's sent as a JSON response using `res.json()`. If not found, a 404 error response is sent.

Requesting the Route:

Assuming your server is running locally, you can make a GET request to fetch an item by its ID using tools like cURL or a web browser:

- To get the item with ID 2: `http://localhost:3000/items/2`

This URL triggers the `getById` route in your Express application, responding with the item information for the specified ID.

Understanding the POST Route and `express.json()` Middleware in Express

```
function createPost(req, res) {
    try {
        console.log("req.body", req.body);
        const postsArr = jsonPosts.posts;
        postsArr.push(req.body);
        res.status(201).json({
            message: "post created "
        })
    } catch (err) {
        res.status(500).json({
            response: "something went wrong on our end"
        })
    }


}
app.use(express.json());
app.post("/posts", createPost);
```

`express.json()` Middleware

Purpose: The express.json() middleware is used to parse incoming requests with JSON payloads. It is a built-in middleware function in Express.
Usage: By using app.use(express.json()), you enable your Express application to automatically parse JSON data in the body of incoming requests. This is essential for handling POST requests where the client sends data in JSON format.

POST Route in Express

Route Definition: The POST route is defined using `app.post("/posts", createPost)`.
Path: /posts - This is the endpoint where the client will send POST requests to create new posts.
Handler Function: createPost - This function is called whenever a POST request is made to the /posts endpoint.

`createPost` Handler Function. The createPost function handles the logic for creating a new post.