**Batch: T7**

**Practical No: 7**

**Title of Assignment: Study and Implementation of ExpressJS**

**Student Name: Shreeyash Shripad Dongarkar**

**Student PRN: 22510025**

**Problem Statement 1: Basics of Express.js**

**1) What is Express and How does it differ from Node.js?**

**=>** Express is a popular web framework for Node.js that simplifies the process of building web applications and APIs. It provides a minimalistic structure with robust features to manage HTTP requests, routes, middleware, and views, making it easier to handle web server functionalities.

1. Node.js:
   * Definition: Node.js is a JavaScript runtime environment that allows you to run JavaScript on the server side.
   * Role: It provides the core functionalities to build server-side applications, handle file system operations, manage databases, and create servers.
   * Purpose: Node.js is great for creating high-performance, scalable applications, but it doesn't provide built-in tools for handling routes, views, or other web-related features.
2. Express:
   * Definition: Express is a lightweight web application framework built on top of Node.js.
   * Role: It simplifies the process of managing HTTP requests, responses, and routes, providing an easier way to create web servers and APIs.
   * Purpose: Express abstracts the low-level details of Node.js, allowing developers to build web applications more efficiently with a clear structure.

Key Differences:

* Level of Abstraction: Node.js provides core functionalities, while Express adds layers of abstraction to manage routes, middleware, and HTTP requests.
* Routing: Express has a powerful and intuitive routing system, which isn’t provided out of the box by Node.js.
* Middleware: Express allows the use of middleware functions to handle requests, errors, and more, streamlining the handling of complex applications.
* Simplification: Express simplifies common tasks like handling requests, sending responses, managing sessions, and working with cookies.

**2) How do you create a simple Express.js Web Server?**

**=>** To create a simple Express.js web server

Step 1: Install Node.js and Express

1. Install Node.js: If you don’t already have Node.js installed, download and install it from here.
2. Create a Project Folder: Open your terminal and create a new project folder:

mkdir express-server

cd express-server

1. Initialize the Project: In your project folder, initialize a new Node.js project with:

npm init -y

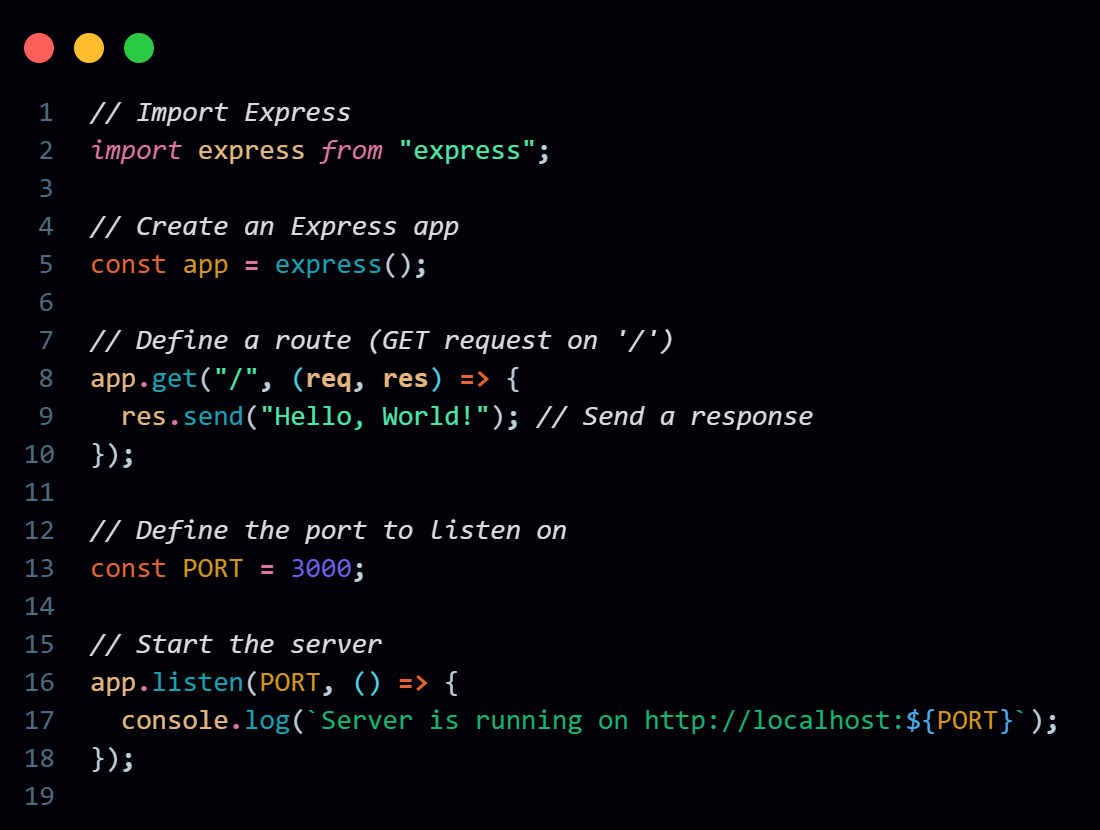
This will create a package.json file in your project folder.

1. Install Express: Install Express.js using npm:

npm install express

Step 2: Create the Express Server

1. Create a Server File: Create a file named server.js (or index.js) in your project folder.
2. Write the Code: Open the server.js file and add the following code to set up a simple web server:

**

Step 3: Run the Server

1. Start the Server: In your terminal, run the following command to start your Express server:

node server.js

1. Visit the Server: Open your browser and go to http://localhost:3000. You should see "Hello, World!" displayed.

**3) Explain the concept of routing in Express.js. How do you define routes in it?**

=> Routing in Express.js refers to the mechanism that allows you to define how your application responds to client requests for specific endpoints (URLs). Each route is associated with a specific HTTP method (GET, POST, PUT, DELETE, etc.) and a URL path. When a request matches a route, Express executes the corresponding callback function to handle the request.

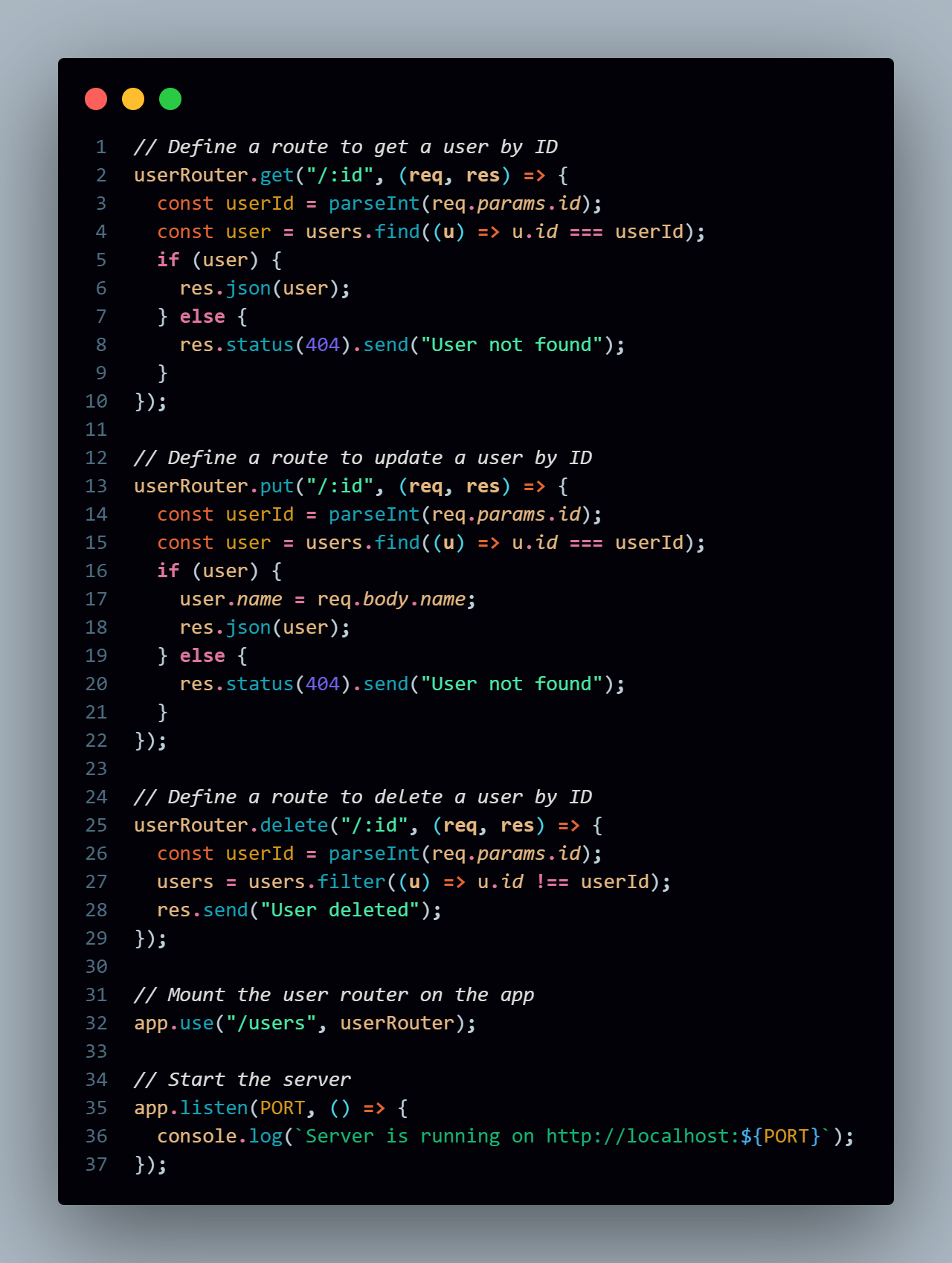
Routing is crucial for building RESTful APIs and web applications, as it helps organize different application endpoints and manage how each endpoint responds to requests.

**Defining Routes in Express.js**

You can define routes in Express.js using the following methods:

1. **Using app.get(), app.post(), app.put(), app.delete() Methods**: These methods are used to define routes for specific HTTP methods.
2. **Using Route Parameters**: You can define dynamic routes with parameters that capture values from the URL.
3. **Using Router**: Express provides a Router class that helps to modularize routes. This is useful for organizing routes in larger applications.
4. **Middleware Functions**: Middleware can also be used in routes to perform actions on incoming requests before they reach the route handler. This is useful for tasks such as authentication, logging, or modifying request data.

****

****

**4) What is middleware in Express.js and how does it work?**

**=>** Middleware in Express.js refers to functions that have access to the request object (req), the response object (res), and the next middleware function in the application’s request-response cycle. Middleware functions can perform various tasks such as modifying the request and response objects, ending the request-response cycle, and calling the next middleware function in the stack.

Middleware is a fundamental concept in Express.js and is used to handle a wide range of functionalities, including logging, authentication, request parsing, response compression, error handling, and more.

**How Middleware Works**

1. **Order of Execution**: Middleware functions are executed in the order they are defined in your application. When a request is made, it is passed through each middleware function in the order specified.
2. **Next Function**: Each middleware function takes three parameters: req, res, and next. The next function is called to pass control to the next middleware in the stack. If next is not called, the request will hang, as the server won't know how to proceed.
3. **Ending the Request-Response Cycle**: A middleware function can end the request-response cycle by sending a response to the client using res.send(), res.json(), or similar methods. In such cases, the next function should not be called.

**Types of Middleware**

1. **Application-level Middleware**: Defined using app.use() or app.METHOD() functions. It applies to all incoming requests or specific routes.
2. **Router-level Middleware**: Similar to application-level middleware, but applied to specific router instances using router.use().
3. **Built-in Middleware**: Express provides some built-in middleware functions for handling requests, such as:
   * express.json(): Parses incoming JSON request bodies.
   * express.urlencoded(): Parses incoming request bodies with URL-encoded payloads.
   * express.static(): Serves static files from a directory.
4. **Error-handling Middleware**: Defined with four parameters: (err, req, res, next). This middleware is specifically for handling errors that occur in the application.

**5) What is the difference between application-level middleware and router-level middleware?**

|  |  |  |
| --- | --- | --- |
| Feature | Application-Level Middleware | Router-Level Middleware |
| Definition | Defined with app.use() or app.METHOD() | Defined with router.use() or router.METHOD() |
| Scope | Applies to all routes in the application | Applies only to routes defined in that router |
| |  | | --- | | **Usage** |  |  | | --- | |  | | Used for global functions (e.g., logging, authentication) | Used for modular route handling (e.g., user, product routes) |
| Organization | Less organized for specific route groups | Helps organize routes and middleware related to specific features |

**6) How do you create and use a custom middleware in Express.js?**

=> Creating and using custom middleware in Express.js is straightforward. Custom middleware allows you to add functionality to your application that fits your specific requirements. This middleware can perform various tasks like logging, authentication, request validation, modifying request or response objects, and much more.

Steps to Create and Use Custom Middleware

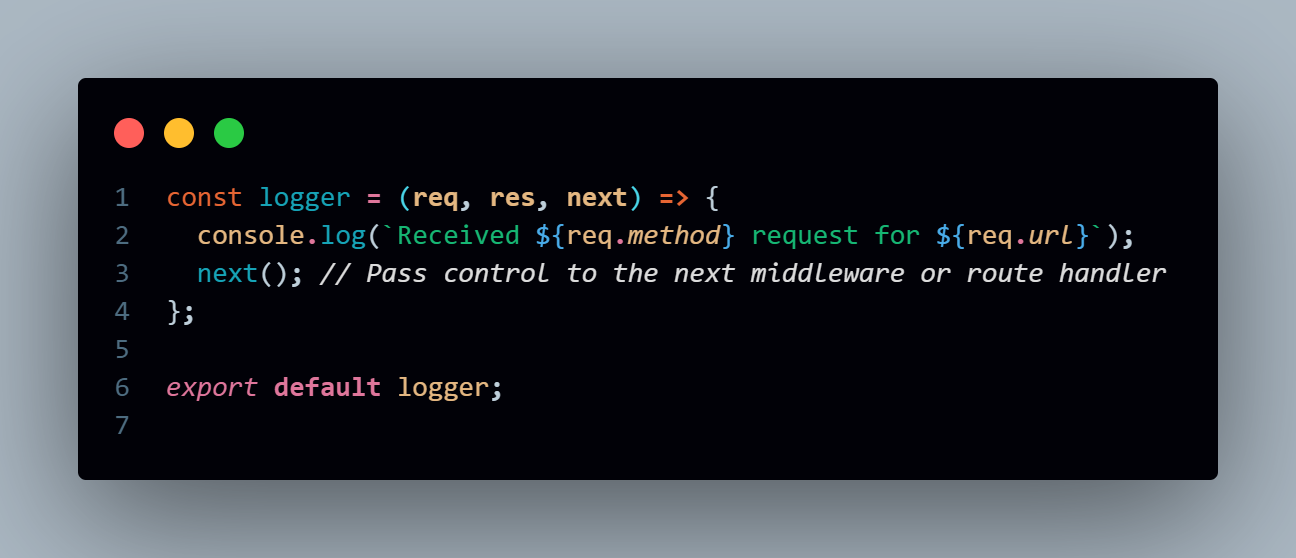
1. Create a Custom Middleware Function

A custom middleware is simply a function that takes three parameters: req (request object), res (response object), and next (a function that passes control to the next middleware in the stack).

2. Use the Middleware in the Application

You can use the custom middleware globally (application-level) or for specific routes (router-level).

First we need to set up Express.js Application and then we can create a middleware as:



Then we can use custom middleware as follows:



**7) What is req and res in Express.js? Give examples of common properties and methods associated with it.**

=> In Express.js, req (request) and res (response) are objects that represent the HTTP request and response, respectively. They are essential to building routes and handling client-server interactions in Express.

1) req (Request Object)

The req object represents the incoming HTTP request to the server. It contains details about the request, such as the HTTP method (GET, POST, etc.), headers, URL, and any data sent by the client.

Common Properties and Methods of req:

* req.method: The HTTP method used for the request (GET, POST, etc.).
* req.url: The URL of the request.
* req.params: An object containing route parameters (from the URL path).
* req.query: An object containing query string parameters (after the ? in the URL).
* req.body: Contains data sent in the request body (for POST or PUT requests). Requires middleware like express.json() or express.urlencoded() to parse the body.
* req.headers: An object containing the headers of the request.

2. res (Response Object)

The res object represents the HTTP response that the server sends back to the client. It is used to send data, status codes, or files back to the client.

Common Properties and Methods of res:

* res.status(code): Sets the HTTP status code of the response.
* res.send(data): Sends a response body of various types (string, object, array, etc.).
* res.json(data): Sends a JSON response. It automatically sets the Content-Type to application/json.
* res.redirect(url): Redirects the client to another URL.
* res.render(view, data): Renders a view template and sends the rendered HTML as the response (used in template engines like Pug, EJS).
* res.set(field, value): Sets a specific HTTP response header.

**8) How would you extract query parameters from a URL in an Express.js route?**

=> In Express.js, you can easily extract query parameters from a URL using the req.query object. Query parameters are the part of the URL that comes after the ? symbol, and they are typically used to pass data to the server in a GET request.

http://localhost:3000/search?term=express&sort=asc

In this case:

* term=express is a query parameter.
* sort=asc is another query parameter.

You can access these parameters in the route handler using req.query.

**Explanation:**

* **req.query.term**: This extracts the value of the term query parameter.
* **req.query.sort**: This extracts the value of the sort query parameter.

**9) How does Express.js handle different HTTP methods (GET, POST, PUT, DELETE)?**

=> Express.js handles different HTTP methods (GET, POST, PUT, DELETE) by providing specific routing methods that correspond to each HTTP method. These methods are used to define routes that respond to various types of HTTP requests.

**1. GET Method**

The GET method is used to retrieve data from the server. It’s commonly used for fetching data without making any changes to the server's data.

**2. POST Method**

The POST method is used to send data to the server, often to create new resources. It’s commonly used when submitting form data or uploading files.

**3. PUT Method**

The PUT method is used to update an existing resource. It generally replaces the current representation of the resource with the new data provided.

**4. DELETE Method**

The DELETE method is used to delete a resource from the server.

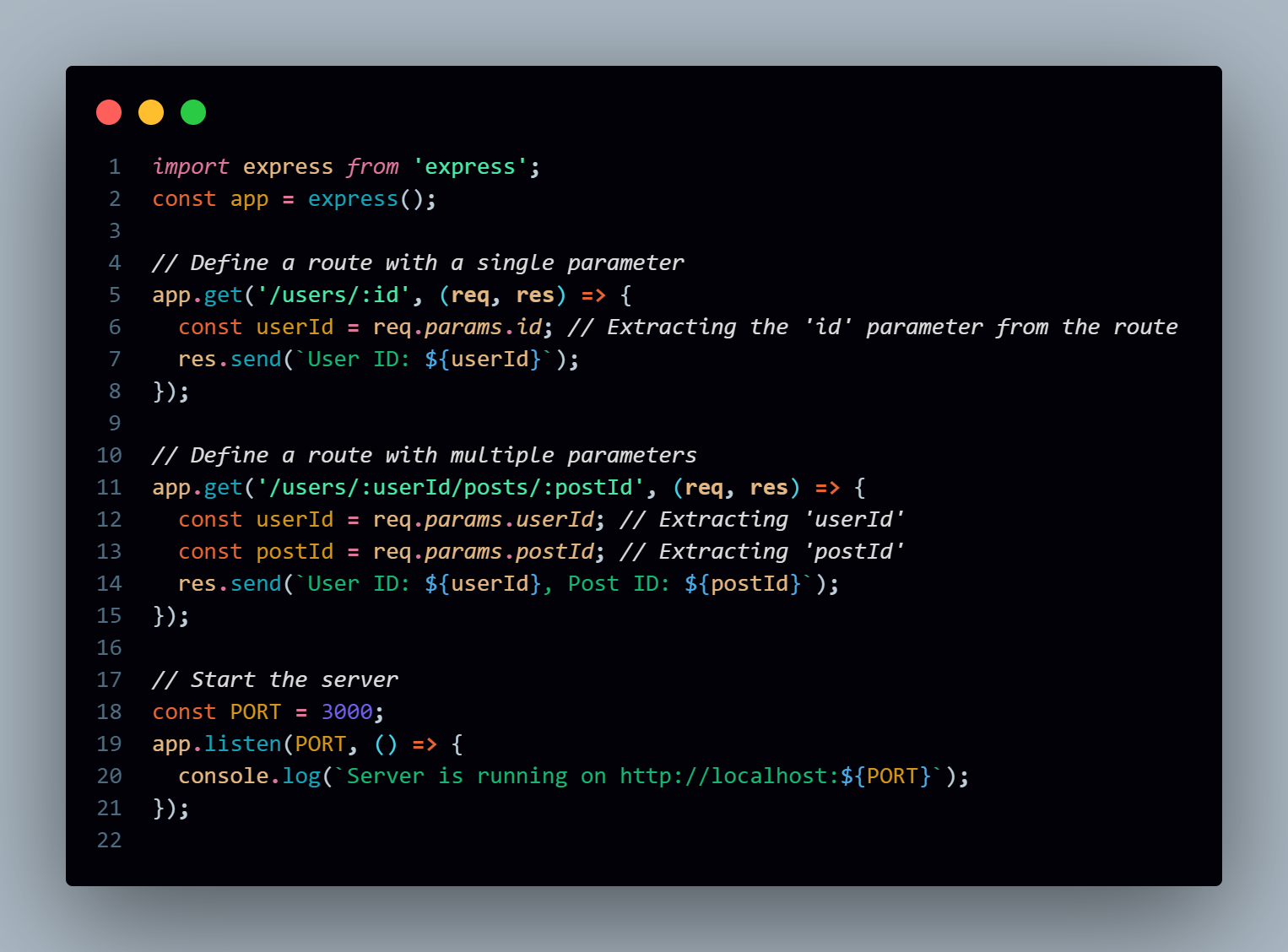


**10) What are route parameters in Express.js? How do you use them in a route definition?**

=> Route parameters in Express.js are dynamic segments of a URL that allow you to capture values at specific points in the path. They are used to capture values from the URL and can be accessed in the route handler through the req.params object.

**Syntax for Defining Route Parameters**

Route parameters are defined in the path by prefixing a colon (:) before the parameter name. For example, in the route /users/:id, the id is a route parameter.

****