**Batch: T6**

**Practical No. 7**

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

**Student Name: Parshwa Herwade**

**Student PRN: 22510064**

Perform following problem statements using ReactJs

Problem Statement 1: Basics of Express.js

Technologies/Frameworks Used:

Express.js: A minimal, flexible Node.js web application framework that provides robust features for building web and mobile applications.

Node.js: A JavaScript runtime environment that executes JavaScript code outside a web browser, primarily on the server-side.

• What is Express.js and how does it differ from Node.js?

Express.js is a framework built on top of Node.js to simplify server-side coding. While Node.js provides a runtime to execute JavaScript on the server, Express.js offers an abstraction layer that handles routing, middleware, and HTTP requests more easily. Express simplifies writing APIs by reducing the amount of boilerplate code.

• How do you create a simple Express.js server?

const express = require('express');

const app = express();

const port = 3000;

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

res.send('Hello World');

});

app.listen(port, () => {

console.log(`Server running on http://localhost:${port}`);

});

• Explain the concept of routing in Express.js. How do you define routes?

Routing in Express.js refers to how endpoints (URLs) respond to client requests. Routes are defined using methods like app.get(), app.post(), etc. Express handles different types of HTTP requests (GET, POST, PUT, DELETE).

Ex. app.get('/home', (req, res) => res.send('Welcome to the Home Page'));

• What is middleware in Express.js, and how does it work?

Middleware functions are those that execute during the lifecycle of a request to the Express.js server. They have access to req, res, and the next middleware function in the application's request-response cycle.

• How do you create and use custom middleware in an Express.js application?

const logger = (req, res, next) => {

console.log(`Request URL: ${req.url}`);

next();

};

app.use(logger);

• What is the difference between application-level middleware and router-level

middleware?

Application-level middleware is bound to an instance of app and can apply to all routes. Router-level middleware is tied to specific router instances.

App-level: app.use(logger);

Router-level: const router = express.Router();

router.use(logger);

• What are req and res in Express.js? Give examples of common properties and

methods associated with each.

ANS. req represents the HTTP request and contains properties like query parameters, body, and headers.

res represents the HTTP response and is used to send data back to the client. Common methods:

req.query: Extracts query parameters.

res.send(): Sends text/html response.

res.json(): Sends a JSON response.

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

Query parameters can be extracted using req.query.

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

const searchTerm = req.query.q;

res.send(`Search results for ${searchTerm}`);

});

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

Express provides methods like app.get(), app.post(), app.put(), and app.delete() to handle respective HTTP methods.

• What are route parameters in Express.js? How do you use them in a route

definition?

Route parameters are named URL segments used to capture values from a URL.

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

const userId = req.params.id;

res.send(`User ID: ${userId}`);

});

Problem Statement 2: Basic Web Server with Express.js

Requirements

• Create a basic Express.js server that listens on port 3000.

• Define three routes:

o GET / - Responds with "Welcome to the Home Page".

o GET /about - Responds with "This is the About Page".

o GET /contact - Responds with "Contact us at: email@example.com".

• Include a 404 error handler that displays a "Page Not Found" message for unknown

routes.

ANS.

Technologies/Frameworks Used:

Express.js: A lightweight framework for Node.js that simplifies the creation of web servers and APIs.

Node.js: JavaScript runtime that allows you to run JavaScript on the server.

Related Theory:

Basic Express.js Server Setup:

To create a basic server with Express.js, you start by importing the express module and creating an application instance. The application will handle HTTP requests on a specified port (in this case, port 3000).

Key Concepts:

Listening on a Port: The app.listen() method tells the application to start listening for incoming requests on the specified port. This is essential for making your server accessible via a web browser or API client.

Defining Routes: Routes are defined using app.get(), app.post(), etc., corresponding to the HTTP methods. Each route can respond to requests with a specific handler function.

Defining Routes:

GET /: Responds with "Welcome to the Home Page."

GET /about: Responds with "This is the About Page."

GET /contact: Responds with a static email address for contact.

404 Error Handling:

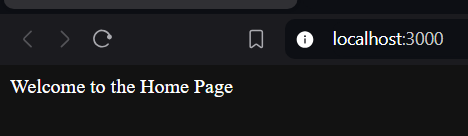
The error-handling middleware at the end of the route definitions catches all requests that do not match any defined route. This is crucial for providing a user-friendly experience, as it helps users understand that they may have navigated to a non-existent page.

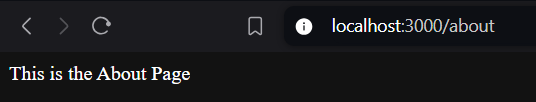
Middleware:

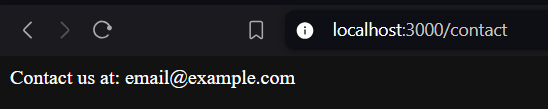
While not explicitly detailed in this example, middleware can also be used for logging, parsing request bodies, or performing authentication checks. The flexibility of Express.js allows developers to include custom or third-party middleware to enhance functionality.

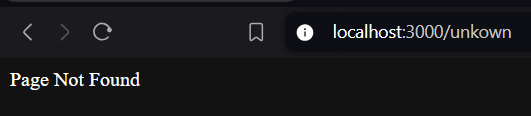
Testing the Server:

Once the server is running, you can use tools like Postman or your web browser to test the routes and ensure they respond correctly. Checking for a 404 response for an undefined route can also be done easily.









Problem Statement 3: Dynamic Route Parameters

Requirements

• Modify the previous server to include the following route:

• GET /users/:id - Responds with "User ID: [id]" where [id] is the dynamic value from

the route.

• Add another route:

• Return a JSON object containing the category and product ID instead of a plain

string.Note:

ANS.

Technologies/Frameworks Used:

Express.js for routing and dynamic URL handling.

Node.js for server-side JavaScript execution.

Related Theory:

Dynamic Route Parameters:

Dynamic parameters allow your application to handle variable segments in the URL, enabling more flexible and powerful routing. Instead of hardcoding values, you can capture them as parameters.

Key Concepts:

Route Parameter Syntax: When defining a route, you can use a colon (:) followed by a name to denote a parameter. For example, in /users/:id, id is a dynamic segment that can change with each request.

Accessing Route Parameters:

The captured parameters can be accessed through req.params, which is an object containing the values specified in the route.

Enhanced Route with JSON Response:

Instead of sending plain text responses, you can return JSON objects. This is particularly useful for APIs, as it provides a structured way to deliver data back to the client.

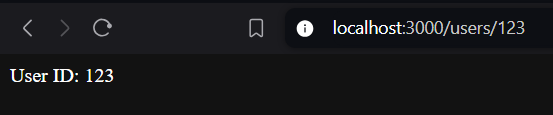
Here, the route captures both category and productId, returning them as a JSON response. This makes the API more informative and easier to work with from a frontend or client-side application.

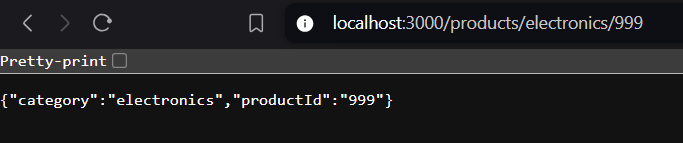
Combining Static and Dynamic Routes:

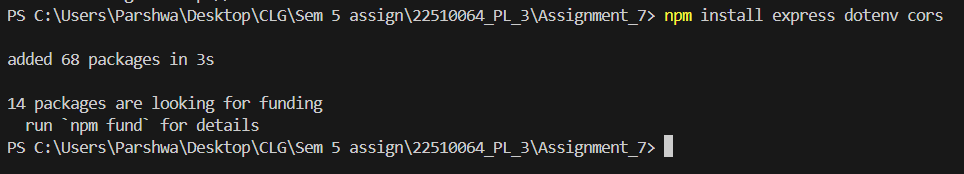
Express allows you to combine static routes (fixed URLs) and dynamic routes, enabling the creation of a more comprehensive API. This approach provides flexibility in handling different resources while maintaining clear URL structures.

Testing Dynamic Routes:

After implementing these dynamic routes, use tools like Postman to make requests to the URLs with different values for id, category, and productId. You can verify that the server responds correctly with the expected output.



  
  
Process:

1. 
2. 
3. 
4. Create a file named server.js and also a .env file and mention port in it.