**Batch -** T5

**Practical No. -** 7

**Title –** Study and implementation of Express.Js

**Student Name -** Sharaneshwar Bharat Punjal

**Student PRN -** 23520011

**Perform following problem statements using Express.Js**

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

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

* **Express.js**: A minimal and flexible web application framework built on top of Node.js. It simplifies the process of handling HTTP requests, defining routes, and integrating middleware for web and API development.
* **Node.js**: A runtime environment that allows JavaScript to run on the server-side. It provides the core functionality to interact with the operating system, file system, and networking.

**Difference**:

* Node.js provides the core features (like creating HTTP servers), whereas Express.js is an abstraction layer that simplifies and organizes these features for web development.

**2. How do you create a simple Express.js server?**

To create a simple Express.js server:

const express = require('express');

const app = express();

// Define a simple route

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

res.send('Hello, World!');

});

// Start the server

const port = 3000;

app.listen(port, () => {

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

});

* This code starts an Express server and responds with "Hello, World!" when visiting the root URL

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

**Routing** refers to how an application responds to client requests to specific endpoints (URLs).

* A **route** in Express.js is defined using methods like app.get(), app.post(), etc. Each route method takes a path and a callback function.
* Example:

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

res.send('About page');

});

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

res.send('Form submitted');

});

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

**Middleware** are functions that execute in the request-response cycle before sending the response to the client. Middleware functions have access to req and res objects and can modify them or end the request-response cycle by sending a response or passing control to the next middleware using next().

**Types of middleware**:

* Built-in middleware (e.g., express.json())
* Third-party middleware (e.g., cors)
* Custom middleware

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

To create a custom middleware:

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

console.log(`${req.method} ${req.url}`);

next(); // Pass control to the next middleware/route

};

app.use(logger);

* This middleware logs the HTTP method and URL of each request and then passes control to the next middleware or route handler using next().

**6. What is the difference between application-level middleware and router-level middleware?**

* **Application-level middleware**: Applies to the entire app. You use app.use() to mount the middleware, and it runs for all routes unless restricted to specific paths.

app.use((req, res, next) => {

console.log('This runs for every request');

next();

});

* **Router-level middleware**: Applies to a specific router or group of routes. You define it using router.use().

const router = express.Router();

router.use((req, res, next) => {

console.log('This runs only for routes in this router');

next();

});

**7. What are req and res in Express.js? Give examples of common properties and methods associated with each.**

* **req (request object)**: Represents the HTTP request and contains data like query parameters, request body, headers, etc.
  + Common properties:
    - req.params: Route parameters (e.g., /user/:id → req.params.id)
    - req.query: Query parameters (e.g., /search?term=abc → req.query.term)
    - req.body: Body of a POST request (used with body-parsing middleware like express.json())
* **res (response object)**: Used to send back the desired HTTP response.
  + Common methods:
    - res.send(): Sends a string or object as the response.
    - res.json(): Sends a JSON response.
    - res.status(): Sets the HTTP status code.

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

Query parameters are extracted from the req.query object. For example:

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

const searchTerm = req.query.term; // URL: /search?term=abc

res.send(`You searched for: ${searchTerm}`);

});

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

Express.js provides specific methods to handle different HTTP requests:

* app.get(): Handles GET requests.
* app.post(): Handles POST requests.
* app.put(): Handles PUT requests (typically used for updating data).
* app.delete(): Handles DELETE requests.

Example:

app.get('/user', (req, res) => { res.send('GET user'); });

app.post('/user', (req, res) => { res.send('POST user'); });

app.put('/user/:id', (req, res) => { res.send(`PUT user ${req.params.id}`); });

app.delete('/user/:id', (req, res) => { res.send(`DELETE user ${req.params.id}`); });

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

Route parameters are named segments in the URL that are captured and available in req.params.

* Example:

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

const userId = req.params.id; // URL: /user/123 → req.params.id = '123'

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

});

In this example, :id is a route parameter that captures part of the URL and makes it available in req.params.

**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".**

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

**GET /contact - Responds with "Contact us at:** [**email@example.com**](mailto:email@example.com)**".**

**Include a 404 error handler that displays a "Page Not Found" message for unknown routes.**

const express = require('express');

const app = express();

require('dotenv').config();

const PORT = process.env.PORT || 4000;

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

    res.send('Welcome to the Home Page');

});

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

    res.send('This is the about Page');

});

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

    res.send('Contact us at: email@example.com');

});

app.use((req, res) => {

    res.status(404).send('Page Not Found');

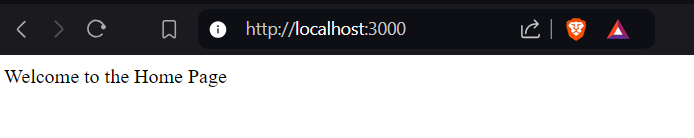
});

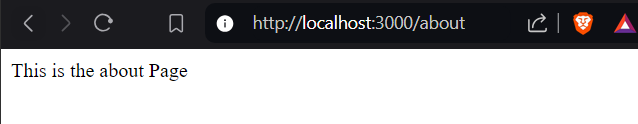
app.listen(PORT , () =>{

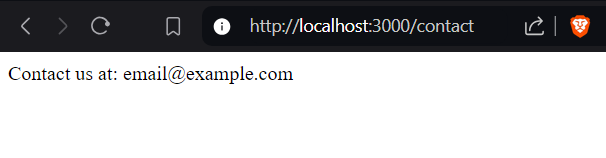
    console.log(`server is running on ${PORT}`);

});

**Output :-**







**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: o GET /products/:category/:productId - Responds with "Category: [category], Product ID: [productId]".**

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

const express = require('express');

const app = express();

require('dotenv').config();

const PORT = process.env.PORT || 4000;

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

    res.send('Welcome to the Home Page');

});

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

//     const userId = req.params.id;

//     res.status(400).send({

//         user : userId,

//     });

// });

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

//     const category = req.params.category

//     const productId = req.params.productId

//     res.status(400).send({

//         category : category,

//         productId : productId

//     });

// });

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

    const userId = req.params.id; // Access the dynamic parameter

    res.send(`User ID: ${userId}`);

});

// Route for getting product details

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

    const category = req.params.category; // Access category

    const productId = req.params.productId; // Access product ID

    // Create a JSON response

    res.json({

        category: category,

        productId: productId

    });

});

app.listen(PORT, () => {

    console.log(`server is running on the port ${PORT}`);

});

**Output :-**

