**Batch- T4**

**Practical No. 7**

**Title of Assignment: Study and implementation of Express.js**

**Student Name: Daivik Karbhari**

**Student PRN: 22510111**

**Problem Statement:**

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

1. What is Express.js and how does it differ from Node.js?
   * **Node.js** is a runtime environment that allows you to run JavaScript on the server side. It provides the platform for running server-side JavaScript and managing things like HTTP requests, filesystem interactions, and more.
   * **Express.js**, on the other hand, is a web framework built on top of Node.js that simplifies the process of building web applications and APIs. It provides higher-level abstractions like routing, middleware, request handling, and more, making it easier to create and manage HTTP servers.
   * **Difference:**
     + Node.js is a platform, whereas Express.js is a web framework for Node.js.
     + Express.js provides a simpler API to handle routing, request/response cycles, and middleware compared to writing plain Node.js applications
2. How do you create a simple Express.js server?

Code:

*const* express = require('express');

*const* app = express();

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

*res*.send('Hello World');

});

app.listen(3000, () *=>* {

  console.log('Server running on http://localhost:3000');

});

In this code:

* We initialize an Express application.
* Define a route that responds with "Hello World" when a GET request is made to the root (/).
* Start the server on port 3000.

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

**Routing** in Express.js refers to defining endpoints (URLs) that a server responds to. A route consists of:

* The HTTP method (GET, POST, PUT, DELETE, etc.)
* The path (URL endpoint)
* A callback function that executes when a request is made to the defined route.

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

*res*.send('Hello World');

});

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

**Middleware** functions are functions that execute during the lifecycle of an Express request. Middleware functions can:

* Execute code
* Make changes to the request and response objects (req, res)
* End the request-response cycle
* Call the next middleware function in the stack

Middleware is commonly used for tasks like logging, authentication, parsing request bodies, error handling, etc.

Example of logging middleware:

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

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

  someFunction();

});

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

* To create custom middleware, you define a function that accepts req, res, and next as arguments and either handles the request/response or passes control to the next middleware.
* Example of custom middleware:
* *const* customMiddleware = (*req*, *res*, *next*) *=>* {
* console.log('Custom Middleware');
* next();
* };
* app.use(customMiddleware);

1. What is the difference between application-level middleware and router-level middleware?
   1. **Application-level middleware** applies to the entire application, regardless of the specific route.

Example:

//application level middleware

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

  console.log('Application-level middleware');

  next();

});

**Router-level middleware** applies to specific route handlers or routers.

// router level middleware

*const* router = express.Router();

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

  console.log('Router-level middleware');

  next();

});

app.use('/api', router);

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

req (Request): Represents the incoming request. Contains details like URL, query parameters, headers, body, etc.

* Common properties:
  + req.params: Route parameters (e.g., /user/:id)
  + req.query: Query parameters (e.g., /search?term=abc)
  + req.body: The body of a POST request (if body parsing middleware like express.json() is used)

res (Response): Represents the outgoing response. Contains methods to send data back to the client.

* Common methods:
  + res.send(): Send a response body.
  + res.json(): Send a JSON response.
  + res.status(): Set the HTTP status code.

1. How would you extract query parameters from a URL in an Express.js route?
   1. Query parameters are part of the URL that comes after the ? symbol. In Express.js, they can be accessed via req.query.
   2. Example: Visiting /search?term=express would output Search Term: express.

* app.get('/search', (*req*, *res*) *=>* {
* *const* searchTerm = *req*.query.term;
* *res*.send(`Search Term: ${searchTerm}`);
* });

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

* Express.js allows you to define routes for different HTTP methods using functions like app.get(), app.post(), app.put(), and app.delete().
* Each function corresponds to the HTTP method it's meant to handle.
* app.get('/user', (*req*, *res*) *=>* {
* *res*.send('GET request to /user');
* });
* app.post('/user', (*req*, *res*) *=>* {
* *res*.send('POST request to /user');
* });
* app.put('/user', (*req*, *res*) *=>* {
* *res*.send('PUT request to /user');
* });
* app.delete('/user', (*req*, *res*) *=>* {
* *res*.send('DELETE request to /user');
* });

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

**Route parameters** are named segments of a route that are dynamic and act as placeholders for actual values. Here, :id is a route parameter that captures a value from the URL (e.g., /user/123 would capture 123 as the id). The value can be accessed via req.params.

app.get('/user/: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:
  + 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".
* Include a 404 error handler that displays a "Page Not Found" message for unknown routes.

Code:

*const* express = require('express');

*const* app = express();

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(3000, () *=>* {

  console.log('Server running on http://localhost:3000');

});

**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.Note:

*const* express = require('express');

*const* app = express();

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

*res*.send(`User ID: ${*req*.params.id}`);

});

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

*res*.json({ category: *req*.params.category, productId: *req*.params.productId });

});

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

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

});

app.listen(3000, () *=>* {

  console.log('Server running on http://localhost:3000');

});