**Express.js Tutorial**

* Express is a web application framework that is built on the top of Node.js
* It is designed to built APIs and make connections to databases.
* It simplifies the development of server-side applications by offering an easy-to-use API for routing, middleware, and HTTP utilities.
* Express.js supports asynchronous routing, which allows you to handle non-blocking I/O operations, such as database queries or external API requests, without freezing the server.

**Installing Express:**

1. Download and Install Node.js
2. Creating a directory for our project and make that our working directory.

mkdir dir\_name

cd dir\_name

1. Using npm init command to create a package.json file for our project.

npm init

1. Installing Express: npm install express
2. Verify installation: npm --version express
3. Create file “server.js” and run using command: node server.js

**Creating an Express.js Server:**

1. Use **require(‘express’)** to import the Express module.
2. Call **express()** to create an Express application instance.
3. Define the **port** for the application, typically 3000.
4. Set up a basic **GET route** with app.get(‘/’, (req, res) => res.send(‘Hello World!’)).
5. Use **app.listen()** method to listen to your desired PORT and to start the server.

**Example:**

const express = require("express");

const app = express();

const port = 3000;

// Routing

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

    res.send("Home page"); // sends message to client

});

// start the server

app.listen(port, ()=>{

    console.log(`Server is running on port: ${port}`);

});

**Note:** Routes are the endpoints of the server, which are configured on our backend server and whenever someone tries to access those endpoints they respond accordingly to their definition at the backend.

**Syntax**: app.httpMethod(path, callback) // httpMethods = get, post, put, delete

**Response Methods:**

1. **res.send():** It sends the HTTP response. The body parameter can be a String or a Buffer object or an object or an Array.
2. **res.sendFile():** transfers the file at the given path and it sets the Content-Type response HTTP header field based on the filename extension.
3. **res.sendStatus()** function is used to set the response HTTP status code to statusCode and send its string representation as the response body.
4. **res.status()** function set the HTTP status for the response. It also allows to add message using above functions.
5. **res.download()** function transfers the file at the path as an ‘attachment’. Typically, browsers will prompt the user to download.
6. **res.end()** function is used to end the response process. This method actually comes from the Node core, specifically the response.end() method of HTTP.ServerResponse.
7. **res.get()** function returns the HTTP response header specified by the field. The match is case-insensitive.
8. **res.json()** function sends a JSON response. This method sends a response (with the correct content-type) that is the parameter converted to a JSON string using the JSON.stringify() method.
9. **res.redirect()** function redirects to the URL derived from the specified path, with specified status, an integer (positive) which corresponds to an HTTP status code.
10. **res.render()** function is used to render views using a template engine (such as EJS, Handlebars, etc.), allowing dynamic content to be injected into the template before sending it to the client.

Syntax: res.render(view [, locals] [, callback])

**EJS Template Engine:**

* Template engine is used to create dynamic HTML pages from server.
* Popular Template Engines are EJS and PUG.
* EJS stands for Embedded JavaScript, which is a templating engine used to generate HTML with JavaScript. It allows you to embed JavaScript code into your HTML templates.

**Installation:** npm I ejs

* In order to set ejs in our project we write: app.set(‘view engine’, ‘ejs’);
* Create a folder named: “**views**” which has .ejs files (similar to html files).
* To render any file in our project, use app.render(“filename”, {name: “hanni”});
* Essential feature of Template Engine is that they allow us to create reusable components called as “**Partials**”.
* The **<% %>** tags in EJS are used to execute JavaScript code within an EJS template.
* **<%= %>** is used to output the value of a variable to the HTML.
* **'<%- %>'**is majorly used  to directly insert HTML markup into your templates.

Data

Template Engine

HTML

Document

<p>Hello, <%= name%></p>

**Middleware in Express:**

Middleware in Express refers to functions that process requests before reaching the route handlers. These functions can modify request and response objects, end the request-response cycle, or call the next middleware function.

* Middleware functions are executed in the order they are defined.
* They can perform tasks like authentication, logging, or error handling.
* Middleware helps separate concerns and manage complex routes efficiently.

**Syntax:**

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

console.log('Middleware executed');

next();

});

**Working of Middleware in Express.js:**

* In Express.js, middleware functions are executed sequentially in the order they are added to the application.
* When a request is received, it is passed through the middleware functions in the order they were defined.
* Each middleware can perform a task and either send a response or call the**next()**function to pass control to the next middleware function.

**Types of Middleware:**

1. **Application-level middleware:**It effects the entire application and executes for all routes. It is used by **app.use()** or **app.METHOD()** where METHOD = get, post, put, delete.
2. **Router-level middleware**: Associated with specific routes and executes for routes defined within that router. It is used by using**router.use()** or **router.METHOD()**.

For example:

router.get(“/user”, (req,res)=>{ console.log(“Middlewae”) });

1. **Error-handling middleware:**It is also an Application level middlewarebut it only handles errors during the request-response cycle. Defined with four parameters (err, req, res, next).
2. **Built-in middleware:**Pre-written middleware, provided by Express which we can use directly. (e.g., express.static(), Express.json(), etc.).
3. **Third-party middleware**: Developed by external packages (e.g., body-parser, morgan, etc.).