**What is Express?**

**Express** is a **web framework** for **Node.js**. A **framework** is a tool that helps you build applications more easily by providing pre-built methods, libraries, and structure to work with.

**Why do we use Express?**

* **Node.js alone** is not enough for handling HTTP requests (like serving a webpage when a user visits a URL). Express helps us do that efficiently by providing simple methods and middleware.
* **Routing**: It helps us handle different URLs (routes) and return the right response for each one.
* **Middleware**: Express lets us add functionality between receiving a request and sending a response, like serving static files or checking if a user is logged in.

In summary, Express simplifies building web servers and APIs.

**How Express Works**

When you create an Express app, it listens for incoming **HTTP requests** (like a user visiting a webpage). Once the request is received, it matches it to a **route** you define and sends an appropriate **response** back (such as an HTML page, an image, or JSON).

For example:

* User visits http://localhost:3000/ → Express responds by sending index.html back to the browser.

**Setting Up Express**

**1. Installing Express**

To start using Express, we first need to install it. You can do this by initializing a Node.js project and installing Express via npm.

# Initialize a new Node.js project

npm init -y # This creates a package.json file

# Install Express

npm install express

This will create a node\_modules folder and a package.json file with Express added as a dependency.

**Creating an Express Server**

Once Express is installed, we can set up a simple server.

**Basic Server Code:**

// Step 1: Import Express

const express = require('express');

// Step 2: Create an Express app

const app = express();

// Step 3: Define a Route

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

res.send('Hello World!'); // Send a simple text response when visiting the homepage

});

// Step 4: Start the server on port 3000

app.listen(3000, () => {

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

});

**Explanation of the Code:**

* **const express = require('express');**: This imports Express so you can use it in your app.
* **const app = express();**: This creates an Express **application** (app) that will handle incoming HTTP requests.
* **app.get('/', (req, res) => {...});**: This defines a **route**. A route is a URL path (like /) and the method (in this case, GET) that will trigger a function when the user visits that URL.
  + **req** (request): Represents the incoming request, such as details about the URL or data sent by the client.
  + **res** (response): Represents the response we will send back to the client (the browser).
* **app.listen(3000, () => {...});**: This makes the app listen on port 3000 for incoming requests. When you run the server, you can visit http://localhost:3000 in your browser.

**Serving HTML Files in Express**

There are several ways to serve **HTML files** in Express. Let's explore the most common methods:

**Method 1: Use res.sendFile() to Send HTML Files**

**Why Use res.sendFile()?**

When a user visits a specific route (e.g., /), we want to send them an HTML file as the response. res.sendFile() lets us do this by providing the file path.

**Example:**

const express = require('express');

const path = require('path'); // Helps with working with file paths

const app = express();

// Serve the home page (index.html) when a user visits '/'

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

res.sendFile(path.join(\_\_dirname, 'index.html')); // Send the 'index.html' file

});

// Start the server

app.listen(3000, () => {

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

});

**Explanation:**

* **path.join(\_\_dirname, 'index.html')**: The path module is used to ensure that the file path is correct. \_\_dirname refers to the current directory where your server file is located, so this will point to index.html in the same folder.
* **res.sendFile()**: This method sends the HTML file as a response to the user's browser.

**Method 2: Serve Static Files Using express.static()**

**Why Use express.static()?**

If you have many static files (like images, stylesheets, or JavaScript), it’s better to place them in a specific directory (like public/) and let Express serve them automatically.

**Example:**

const express = require('express');

const path = require('path');

const app = express();

// Serve static files from the 'public' folder

app.use(express.static(path.join(\_\_dirname, 'public')));

// Serve the home page (index.html)

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

res.sendFile(path.join(\_\_dirname, 'public', 'index.html')); // Send 'index.html' from 'public' folder

});

// Start the server

app.listen(3000, () => {

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

});

**Explanation:**

* **app.use(express.static(...))**: This middleware tells Express to serve files from the public directory. So if you place your CSS, JS, or image files in public/, Express will automatically serve them without needing to manually define routes.
* **Example file structure**:

project/

├── public/

│ ├── index.html

│ ├── style.css

│ └── image.jpg

└── server.js

* If you visit http://localhost:3000/style.css, Express will serve the style.css file from the public/ folder.

**Method 3: Dynamically Render HTML with EJS**

**Why Use EJS?**

If you want to include dynamic content in your HTML (for example, user names, list items, etc.), you can use a **template engine** like **EJS**. EJS allows you to embed JavaScript in your HTML files, making it easier to generate dynamic pages.

**Example:**

1. Install EJS:

npm install ejs

1. Set up the server to use EJS:

const express = require('express');

const app = express();

// Set EJS as the view engine

app.set('view engine', 'ejs');

// Route for home page with dynamic content

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

res.render('index', { title: 'Home Page', message: 'Welcome to my website!' });

});

// Start the server

app.listen(3000, () => {

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

});

1. Create an index.ejs file in the **views** folder:

<!-- views/index.ejs -->

<html>

<head>

<title><%= title %></title>

</head>

<body>

<h1><%= message %></h1>

</body>

</html>

**Explanation:**

* **app.set('view engine', 'ejs');**: This tells Express to use **EJS** as the template engine for rendering dynamic HTML pages.
* **res.render('index', { title: 'Home Page', message: 'Welcome to my website!' });**: This renders the index.ejs file and passes dynamic data (the title and message variables) to it.
* **<%= title %> and <%= message %>**: These are EJS syntax tags that insert the values of title and message into the HTML when it’s rendered.

**Summary and Key Points**

1. **Express** is a framework for Node.js that helps you handle HTTP requests, define routes, and send responses.
2. Use **res.sendFile()** to serve HTML files directly when users visit a specific route.
3. Use **express.static()** to serve static files (like CSS, JavaScript, and images) from a specific folder.
4. **EJS** lets you dynamically generate HTML by embedding JavaScript into HTML files, which is useful for creating dynamic content.

These are the most common ways to work with HTML and static assets in Express. As a beginner, you can start with simple routes and gradually explore using EJS and static file serving as your application grows.