CODE2CAREER

**WEEK : 5**

**TOPICS : Node.js Part-1**

**📁 Students must create a folder named 'Week 5'. Inside it, create subfolders for each day (Day 1 to Day 6).  
📂 For Days 1 to 4, each topic you study must be practiced by creating a separate `.js` file. Include meaningful code and comments in each file.  
📂 For Days 5& 6, Complete the given tasks as instructed and submit your work inside the Day 5 folder.**

|  |
| --- |
| **DAY 1** |
| * What is a Server * How Can a Computer Become a Server(Beginner level) * Localhost vs Remote Server * What is a Port (Default & Custom Ports) * What Kind of Requests Does a Server Receive?   + HTTP Request Basics   + Request Types (HTTP Methods)   + Structure of the Request   - URL or Endpoint  - Headers  - Body   * What Does the Server Send Back? (HTTP Response)   + HTTP Response Basics   + Parts of an HTTP Response   - Status Code (e.g., 200 OK, 404 Not Found)  - Headers (e.g., Content-Type)  - Body (HTML, JSON, text, etc.)   * What is an API? * What is Node.js? * Key Features of Node.js: * Create a basic Node.js server using the http module that sends a plain text response * Create a basic HTML page with a button that fetches data from your Node server and displays it in a <div>. * Understand routing in Node.js and implement manual routes (/, /about, /contact) with a 404 fallback using the core http module. |
| **Day 2** |
| * What is a Package? * What is npm? * Initializing npm   + What happens when we run npm init   + What is package.json & package-lock.json * What is Express? * Setting Up Express * Writing Your First Express Route * Create an Express server with 3 routes: /, /about, /contact. * Express Routing (Deeper Dive)   + Understanding how route order affects behavior   + Different HTTP method handlers in Express (app.get, app.post, etc.)   + Handling unmatched routes (404 error handling)   + Sending different types of responses (JSON, HTML, Text) * Route Parameters * Query Parameters |
| **Day 3** |
| * Middleware in Express   + What is Middleware?   + Types of Middleware (Built-in, Application-level, Third-party)   + Order of Middleware Execution\   + Example: Custom Logging Middleware   + Using express.json() and express.urlencoded()   + Using express.static() to Serve Static Files * Serve static files in Express by setting up a public folder and using express.static() to access HTML, CSS, and JS files. * What is a Router   + Why we split routes into separate files   + Create a separate route file using express.Router()   + Export the router and import it into the main app   + Mount with app.use() |
| **Day 4** |
| * What are Modules? * Types of Modules   + Core Modules (like fs, path, os)   + Local Modules (your own files)   + Third-party Modules (installed via npm) * Using Modules (Importing & Exporting)   + Import using require() (CommonJS)   + Export using module.exports   + Intro to ES Modules (import, export) – when to use   + Difference between CommonJS and ES Modules * View Engine & Dynamic Rendering   + What is a View Engine & Why use it in Backend?   + Popular Options (EJS, Pug, Handlebars)   + Setting up EJS in Express   + Passing Data with res.render()   + Views Folder & .ejs Files   + Example: Render Profile Page * EJS + Tailwind Practice – Product Cards Page   + Passing Data with res.render()   + Create a dummy array of products (each with name, price, image).   + Use EJS to loop through the array and render product cards dynamically.   + Style the cards using Tailwind CSS in a responsive grid layout.   + Each card must show the image, name, and price. |
| **Day 5 & Day 6** |
| **Login System with Predefined Credentials and Session Handling**   * **Login Page**   + Create a login form using EJS.   + Input fields: username, password.   + Style the form using Tailwind CSS (centered card or form).   + On submit, **POST** data to /login. * **Server-Side Login Validation**   + In /login route:     - Compare credentials with predefined values: username = admin, password = 1234.   + If valid:     - Save session as req.session.isAuth = true     - Redirect to /home.   + If invalid:     - Re-render the login form with an error message (e.g., “Invalid credentials”). * **Home Page**   + Only accessible when session is active (req.session.isAuth === true).   + Show a welcome message like **“Welcome Admin”**.   + Include a **Signout** button.   + Style using Tailwind CSS. * **Signout Route**   + On clicking Signout, destroy the session and redirect to /login. * **Route Protection Middleware**   + Create a middleware to protect /home.   + If the session is not authenticated, redirect to /login. * **Back Button Restriction**   + After logging out, user should not be able to access /home using the browser back button. |