**🔹 Introduction: Setting Up an Angular Application**

**1️⃣ What is Angular? Why are we learning this?**

📌 **"Angular is a front-end framework developed by Google. It is used to create single-page applications (SPAs), where users can navigate between different sections of a website without full-page reloads. This makes applications faster and more efficient."**

📌 **"In this lab, we are setting up our first Angular project to understand the project structure and see how Angular applications are executed."**

📌 **Connecting React and Angular Topics:**

**Component-Based Architecture**

**React:** Functional & Class Components

**Angular:** Components with Templates, Styles, and TypeScript

**Connection:** Both frameworks use component-based architecture but differ in syntax and state management.

**Data Binding & Events**

React: One-way binding (useState, props)

Angular: Two-way binding with ngModel

Connection: Show how useState in React works similarly to Angular’s [(ngModel)].

**Routing**

React: React Router

Angular: Angular Router (RouterModule)

Connection: Compare route definition in both frameworks.

**Service & State Management**

React: Context API, Redux

Angular: Services, Dependency Injection

Connection: Services in Angular act like centralized state management in React.

**Fetching Data from APIs**

React: fetch or Axios

Angular: HTTP Client (HttpClientModule, Observables)

Connection: Compare Promises in React vs. Observables in Angular.

**2️⃣ What Happens When We Run ng serve?**

* The ng serve command **compiles the application**, **bundles the required files**, and **runs a local development server**.
* It enables **Hot Module Replacement (HMR)**, meaning any changes we make to the code will be automatically reflected in the browser **without restarting the server**.

Example:  
**"Think of this like launching a machine. We are setting up our Angular application, which compiles into small JavaScript files that can run in a browser. The terminal is showing us that everything is built correctly, and the app is ready to be viewed in a web browser."**

**🛠 Step 1: Install Node.js and Angular CLI**

**Check if Node.js is Installed**

Open the terminal (Command Prompt/PowerShell in Windows, or Terminal in macOS/Linux) and run:

node -v

If Node.js is not installed, download it from [Node.js official site](https://nodejs.org/" \t "_new) and install it.

**Install Angular CLI**

Run the following command to install Angular CLI globally:

Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass

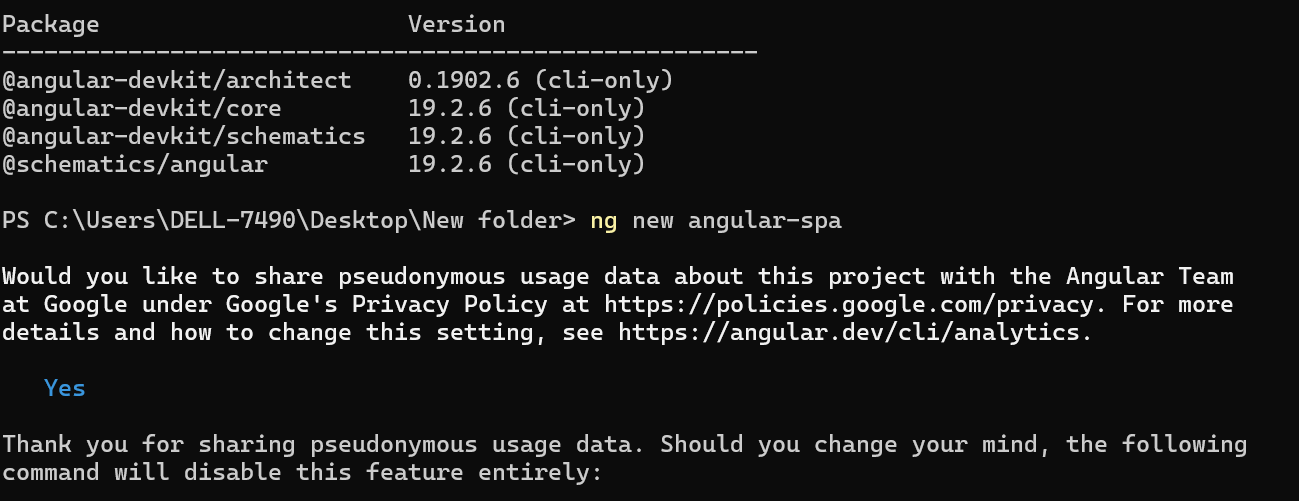
npm install -g @angular/cli ----> angular command line interface-tool to

automate and simplify development tasks.

Verify installation:

ng version





**📂 Step 2: Create a New Angular Project**

Run the following command:

ng new angular-spa

we're telling Angular CLI to **create a new project named** angular-spa.

### ⚙️ What Happens Behind the Scenes?

Angular CLI will:

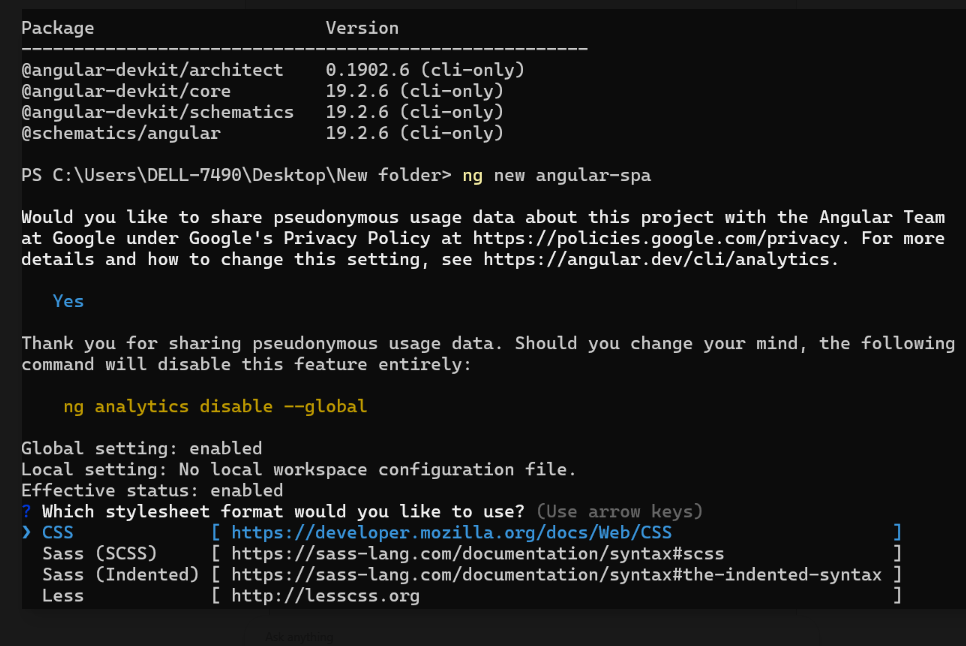
📁 Create a new folder named angular-spa.

📦 Set up all the files and folders required for a full Angular app.

📜 Configure important files like angular.json, package.json, tsconfig.json.

📥 Automatically install all required node packages (npm install).

✅ Create the base app structure (with AppComponent).



### ****What to Do Next?****

Use the **arrow keys** to select an option and press **Enter** to continue.  
For simplicity, choose **CSS** (already selected) and press **Enter**.

If it still doesn’t move forward after selecting CSS:

1. Press **Enter** again.
2. If it gets stuck, close the terminal and restart the command:

ng new angular-spa



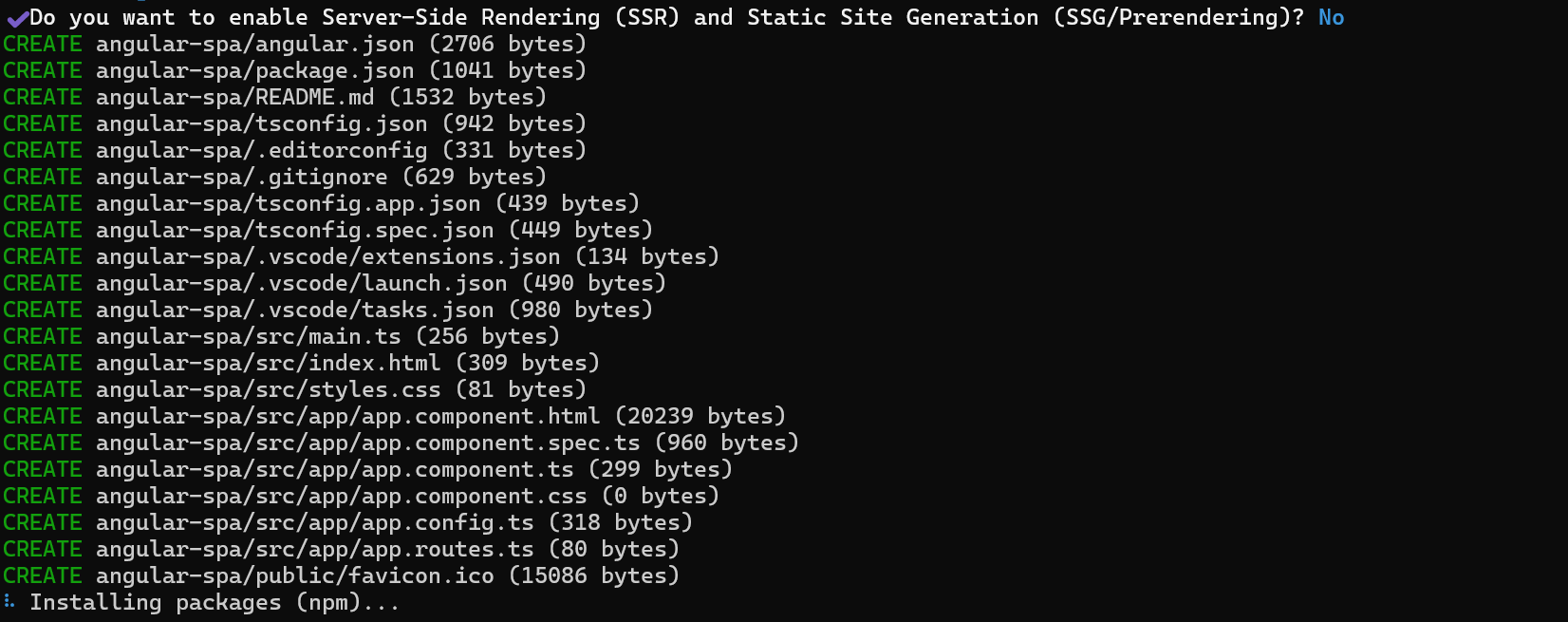
It is asking if you want to enable **Server-Side Rendering (SSR)** and **Static Site Generation (SSG/Prerendering)**.

**What Should You Choose?**

* **For a basic Angular Single Page Application (SPA), choose No** (default option).
* Type **N** and press **Enter**.

If you want SSR later, you can always add it using:

ng add @angular/universal



**What's Happening?**

1. The Angular CLI is creating all the necessary files for your project. ✅
2. Now it's installing required **npm packages** (dependencies). ⏳

**What to Do Next?**

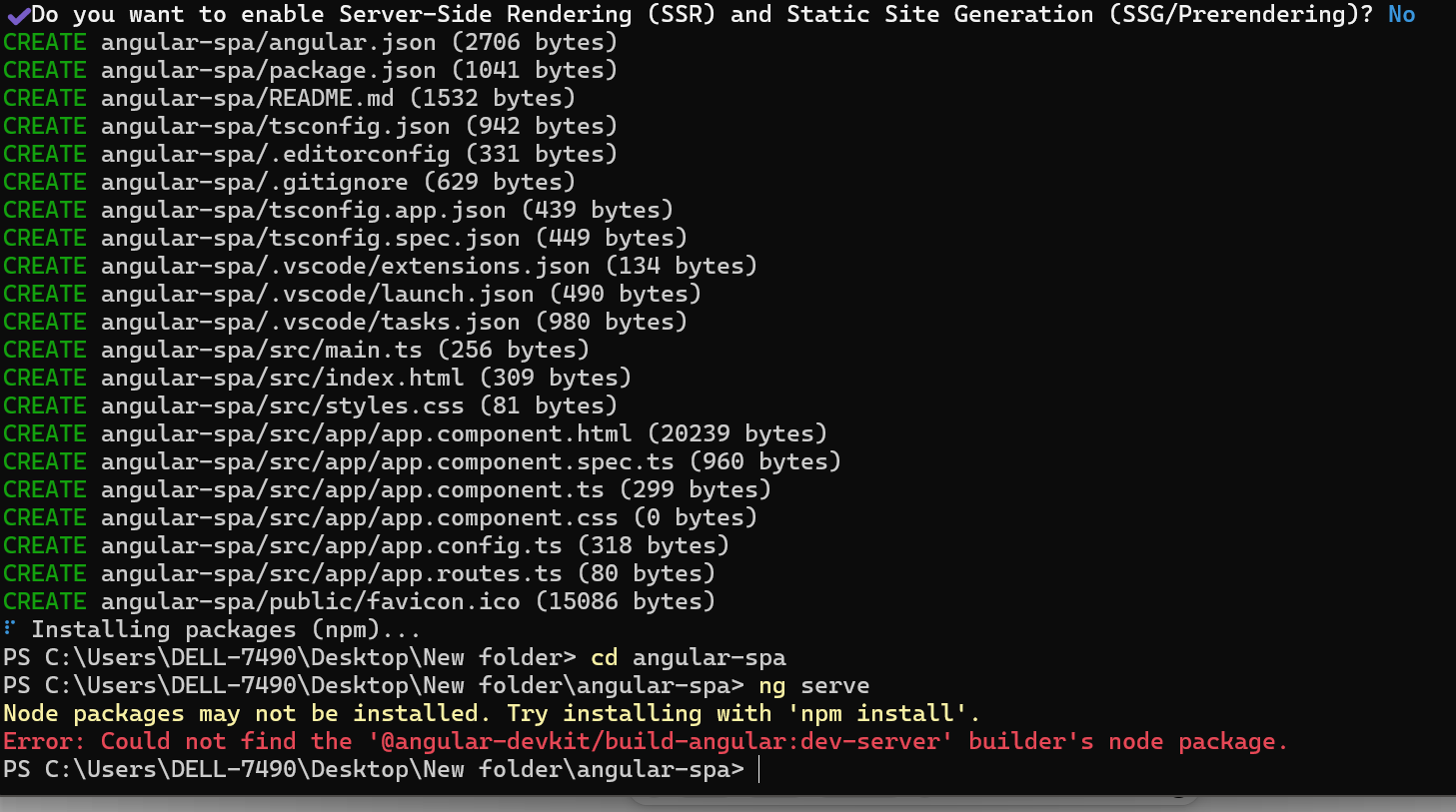
* Wait for the package installation to complete.
* Once done, navigate to your project folder:

cd angular-spa

Start the development server:

ng serve

**ERROR:**



The error **"Could not find the '@angular-devkit/build-angular:dev-server' builder's node package"** indicates that some required dependencies were not installed properly.

### ****Solution****

Follow these steps to fix it:

#### **1️⃣ Run** npm install

Inside your project folder (angular-spa), run:

npm install

This will install all missing dependencies listed in package.json.

#### **2️⃣ Try Running Again**

Once the installation completes, start your Angular project:

ng serve

#### **3️⃣ If the Issue Persists, Try Cleaning Cache**

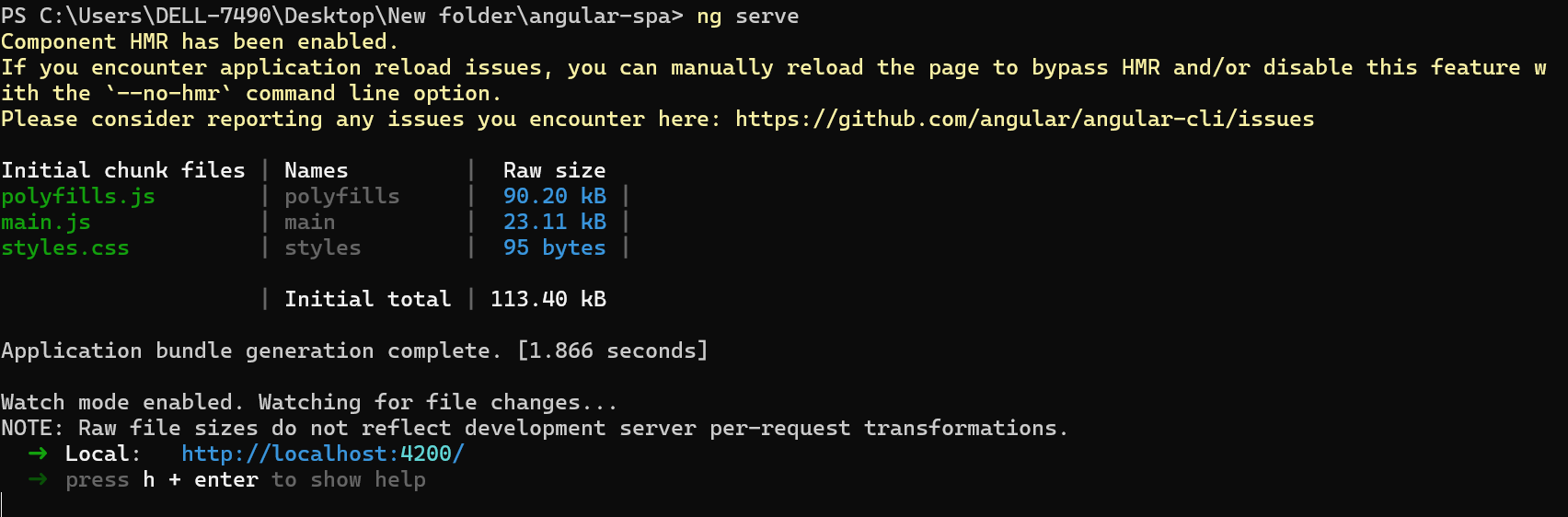
If the error still appears, try clearing the npm cache and reinstalling:

npm cache clean --force

rm -rf node\_modules package-lock.json

npm install

ng serve



## ****🔹 Understanding the Terminal Output (Step-by-Step Analysis)****

### ****Terminal Output Breakdown****

🔹 **Key sections of the output after running ng serve:**

Initial chunk files | Names | Raw size

-------------------------------------------------

polyfills.js | polyfills | 90.20 kB

main.js | main | 23.11 kB

styles.css | styles | 95 bytes

-------------------------------------------------

Application bundle generation complete. [1.866 seconds]

💡 **What does this mean?**  
1️⃣ polyfills.js: **Contains code to support older browsers** that don’t fully support modern JavaScript features.  
2️⃣ main.js: **The core Angular application logic**, including components, modules, and services.  
3️⃣ styles.css: **Contains global CSS styles** used by the application.

📌 **"These files work together to render our Angular application in a web browser."**

**Next, we see:**

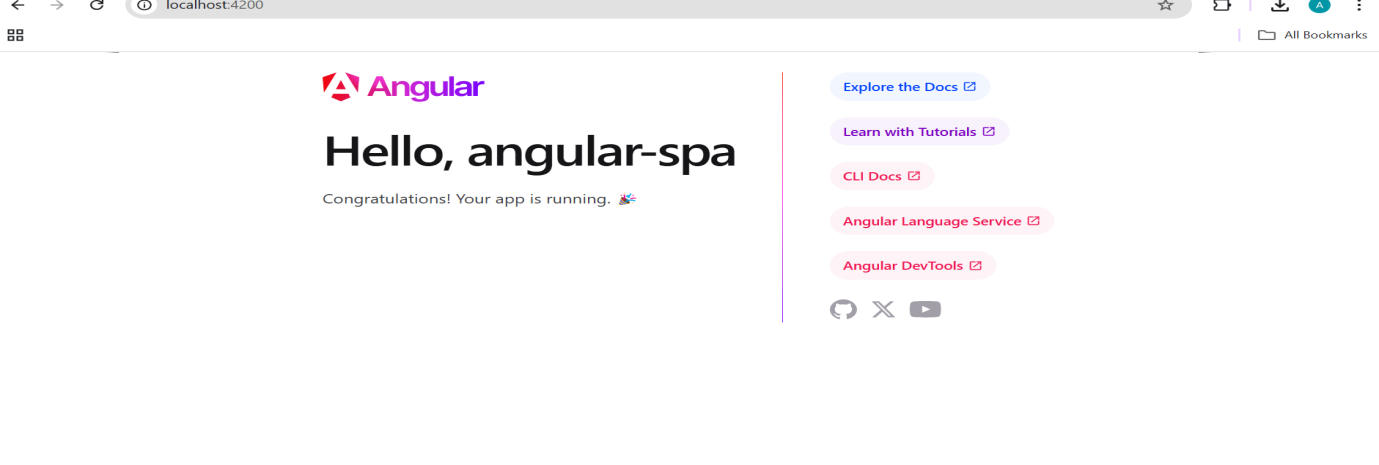
Watch mode enabled. Watching for file changes...

Local: http://localhost:4200/

💡 **This means:**  
✔️ The Angular app is running successfully.  
✔️ Any code changes will automatically reflect in the browser.  
✔️ We can now open **http://localhost:4200/** in a web browser.

Open your browser and visit:

<http://localhost:4200>



## ****🔹 Understanding the Browser Output****

### ****What Happens When We Open**** http://localhost:4200/****?****

📌 **"Now, let's open our browser and check the output. You will see a default Angular welcome page."**

🔹 **Breakdown of the browser output:**  
✅ The text **"Hello, angular-spa"** confirms that our application is running successfully.  
✅ The page has links to Angular documentation and tools.  
✅ This default page is created using Angular’s **root component (app.component.ts)** and template (app.component.html).

💡 **"This is the default template provided by Angular when we create a new project. In our next labs, we will modify this page by adding our own components and styling."**

**🔹 Connecting This Lab to Real-World Applications**

💡 **"Why does this matter?"**

📌 **"Understanding this setup is crucial because all Angular applications start from this structure. Once we know how an Angular app loads, we can start building complex features like dynamic pages, user authentication, and data fetching."**

Example:

* **Gmail**: Runs on Angular, allowing smooth navigation between emails without reloading the page.
* **YouTube**: Uses Angular for faster loading and dynamic UI updates.

**🔹 Conclusion: Key Takeaways & Next Steps**

**📌 What We Learned Today**

✅ We installed and set up an Angular project.  
✅ We ran ng serve and analyzed its output.  
✅ We understood the **core files** in an Angular application.  
✅ We viewed the default **Angular Welcome Page** in a browser.

**📌 What’s Next?**

🔹 In the next session, we will:  
✅ Create our **own components** to display dynamic content.  
✅ Modify app.component.html to add a **custom message** instead of the default welcome page.  
✅ Learn about **data binding**, which lets us dynamically update content.

**"Now that we know how an Angular app runs, let's move to the next step—modifying and creating components!"** 🚀

**🛠 Understanding the Core Files in an Angular Application**

When we create an Angular application using ng new project-name, several files and folders are generated. These files define the **structure** of the application and help manage different functionalities such as components, routing, styles, and dependencies.

Let’s **break down the most important files and their roles** in an Angular application.

**📂 1. Main Project Structure**

When we open our project folder, we see the following structure:

pgsql

angular-spa/

│── node\_modules/

│── src/

│ ├── app/

│ │ ├── app.component.html

│ │ ├── app.component.ts

│ │ ├── app.component.css

│ │ ├── app.module.ts

│ ├── assets/

│ ├── environments/

│ ├── index.html

│ ├── main.ts

│ ├── styles.css

│ ├── angular.json

│── package.json

│── tsconfig.json

Each of these files serves a specific purpose. Let’s go through them one by one.

**📌 2. Important Files in an Angular Application**

**(A) src/ Folder – The Main Codebase**

This is where our application’s source code resides.

**🔹 index.html (The Entry Point)**

📄 **Purpose**:

* This is the main **HTML file** that gets loaded first when the Angular application starts.
* It contains a placeholder <app-root></app-root> where the Angular application is injected.

📌 **Example:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>AngularSPA</title>

</head>

<body>

<app-root></app-root> <!-- Angular application loads here -->

</body>

</html>

💡 **Explanation:**

* The <app-root></app-root> tag is where the main **Angular component (AppComponent)** is injected.
* Everything we build in Angular will be displayed inside this tag.

**🔹 main.ts (Bootstraps the App)**

📄 **Purpose**:

* This is the **entry point of the Angular application** in TypeScript.
* It tells Angular which module should be loaded first when the app starts.

📌 **Example:**

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule)

.catch(err => console.error(err));

💡 **Explanation:**

* This file **imports AppModule** (which contains all the components of our app).
* Then, it calls bootstrapModule(AppModule), which **starts the Angular application**.

**(B) app/ Folder – The Heart of an Angular Application**

This is where we define all our **components, modules, and services**.

**🔹 app.module.ts (The Main Module)**

📄 **Purpose**:

* Every Angular application has at least one **module**, and app.module.ts is the root module.
* It tells Angular what **components, directives, and services** belong to the application.

📌 **Example:**

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

💡 **Explanation:**

* declarations: Lists the **components** that belong to this module.
* imports: Lists the **other modules** that this module depends on.
* bootstrap: Specifies which component should be loaded first (in this case, AppComponent).

**🔹 app.component.ts (The Main Component)**

📄 **Purpose**:

* This is the **main component** that loads when the app starts.
* It contains the **logic** for the root component.

📌 **Example:**

import { Component } from '@angular/core';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'My First Angular App';

}

💡 **Explanation:**

* selector: 'app-root' → This defines the **HTML tag** <app-root> that is used in index.html.
* templateUrl → Links this component to its **HTML template (app.component.html)**.
* styleUrls → Links this component to its **CSS styles (app.component.css)**.
* The title variable can be used in the HTML template to display dynamic content.

**🔹 app.component.html (Component Template)**

📄 **Purpose**:

* This file contains the **HTML code** for the root component.

📌 **Example:**

<h1>Welcome to {{ title }}!</h1>

💡 **Explanation:**

* The {{ title }} syntax is called **Interpolation**, which dynamically binds the title variable from app.component.ts.

**🔹 app.component.css (Component Styles)**

📄 **Purpose**:

* This file contains the **CSS styles** for app.component.html.

📌 **Example:**

h1 {

color: blue;

font-family: Arial, sans-serif;

}

💡 **Explanation:**

* Any styles here **apply only to app.component.html** and not to the entire application.

**(C) Other Important Files**

| **File** | **Purpose** |
| --- | --- |
| package.json | Lists all installed dependencies (e.g., Angular, Bootstrap) and scripts. |
| angular.json | Configuration file for the Angular CLI (build options, styles, etc.). |
| tsconfig.json | TypeScript configuration for compiling Angular code. |
| styles.css | Global styles that apply to the entire Angular application. |

**📌 Summary: How These Files Work Together**

1️⃣ **index.html** → Loads the Angular application and displays <app-root>.  
2️⃣ **main.ts** → Bootstraps AppModule.  
3️⃣ **app.module.ts** → Declares AppComponent and other components.  
4️⃣ **app.component.ts** → Defines the main component logic.  
5️⃣ **app.component.html** → Displays the component’s UI.  
6️⃣ **app.component.css** → Styles the component.

**📌 Key Takeaways**

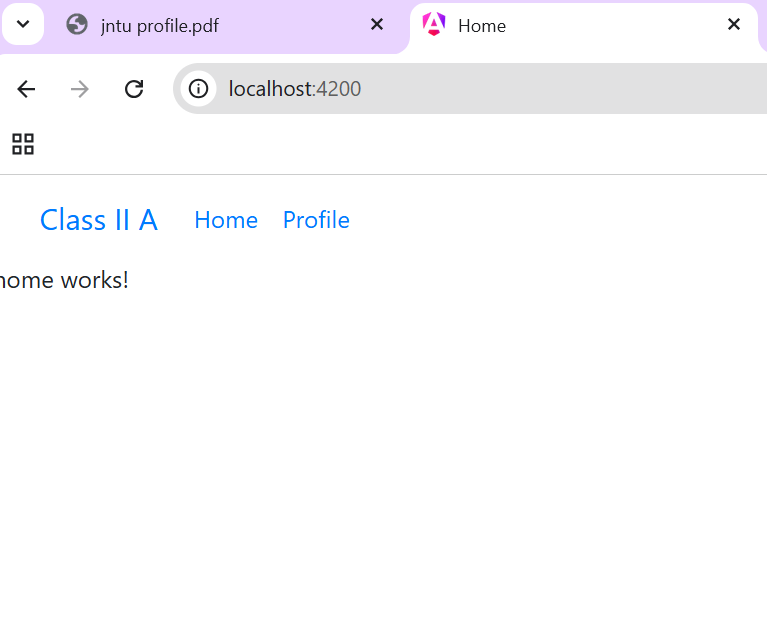
✅ **Angular apps follow a modular structure**, where each component has its own **HTML, CSS, and TypeScript files**.  
✅ **index.html is the main entry point**, and <app-root> is replaced by the root component.  
✅ **Components (.ts files) control the logic**, while **templates (.html files) display the UI**.  
✅ The **Angular CLI manages dependencies** and builds the app using package.json and angular.json.

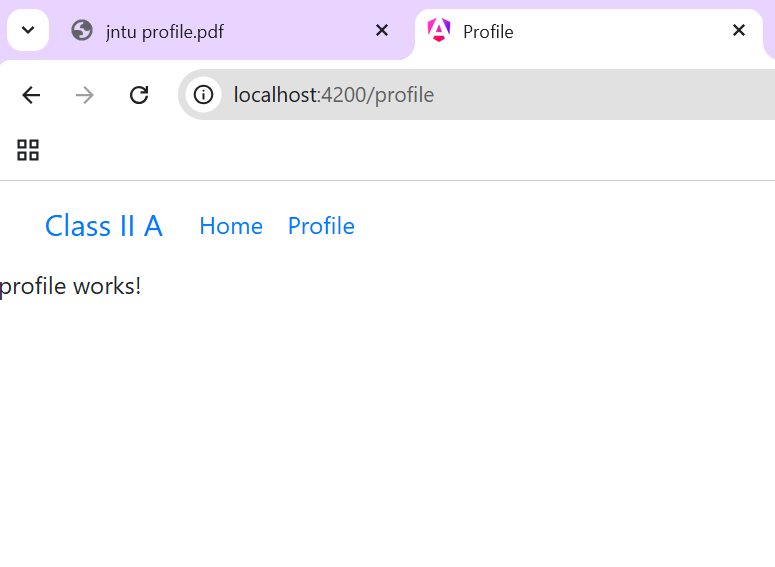
**📌 What’s Next?**

In the next session, we will:  
✔️ Create **new components** to display different sections of a webpage.  
✔️ Modify the **app.component.html** file to display dynamic content.  
✔️ Learn about **data binding**, which allows real-time updates in Angular applications.

💡 **"Now that we understand the core files, let's start modifying them to build a real-world Angular project!"** 🚀

**After this let’s turn our application to something with menu items.**

****

****

**So to achieve this lets do the routing.**

**Step 1: index.html (entry point):**

**Modify Index.html 🡪**

<!doctype html>

<html lang="en">

<head>

  <meta charset="utf-8">

  <title>Prog10</title>

  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">

  <link rel="icon" type="image/x-icon" href="favicon.ico">

  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.4.1/dist/css/bootstrap.min.css" crossorigin="anonymous">

</head>

<body>

  <app-root></app-root>

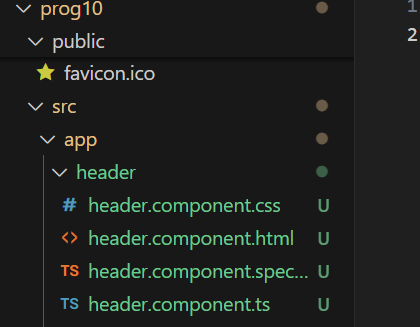
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.4.1/dist/js/bootstrap.min.js"

                crossorigin="anonymous" ></script>

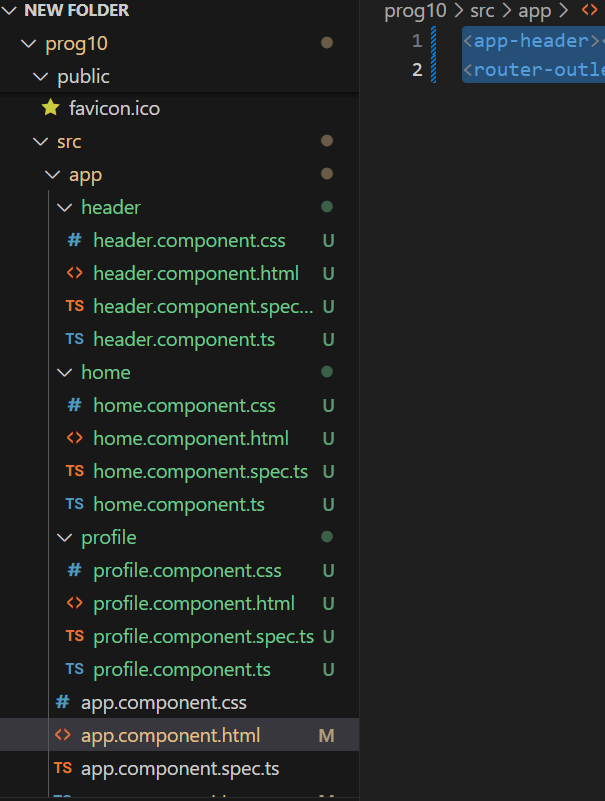
</body>

</html>

**Step 2: creating different menu items 🡪web pages🡪components**

* **cd prog10**
* **ng generate component header or ng g c header**
* 

**Similarly create two more components 🡪home and profile**

****

**Step 3:**

**Lets now move to the parent file ie. App.component.html and app.component.ts**

* **Delete everything from app.component.html and replace it with**
* <app-header></app-header>
* <router-outlet></router-outlet>

**🔧 1. Purpose of Replacing app.component.html**

By default, app.component.html might contain static content, But when you're building a **multi-page Angular application with navigation**, this static content doesn't change with routing. So you update it to **dynamically load different components** based on the URL route.

### 🔹 <app-header></app-header>

#### 👉 What it is:

A **custom component tag** that loads the **Header Component** (typically navigation bar or top header).

### 🔹 <router-outlet></router-outlet>

#### 👉 What it is:

A **placeholder** for loading components based on the active route.

**🧠 Summary**

| **Element** | **Purpose** | **Example Outcome** |
| --- | --- | --- |
| <app-header> | Always display the navigation/header bar | Nav links visible on all pages |
| <router-outlet> | Render page components based on URL route | Shows Home/Profile/Dashboard/etc based on routing |

**✅ The <router-outlet> directive comes from the @angular/router package.**

**📦 Import Location**

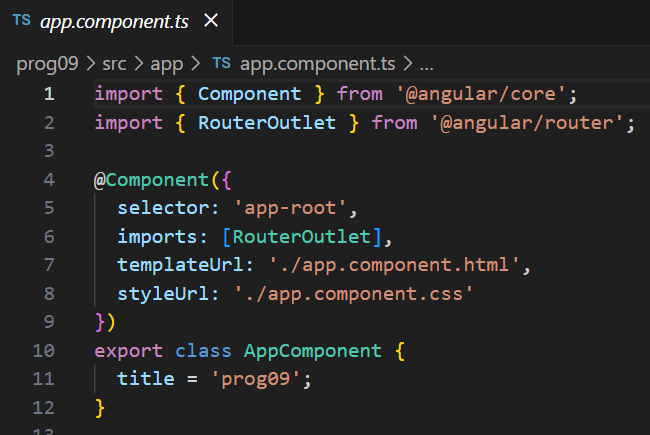
To use routing (including <router-outlet>), you must import RouterModule from @angular/router.

import { RouterModule } from '@angular/router';

But where we are importing it, ofcourse in its script file app.component.ts

Step 4: Now lets modify app.component.ts:

**BY default:**

****

**We need to update it with adding router module and header component**

import { Component } from '@angular/core';

import { RouterModule,RouterOutlet } from '@angular/router';

import { HeaderComponent } from './header/header.component';

@Component({

  selector: 'app-root',

  imports: [RouterOutlet,HeaderComponent,RouterModule],

  templateUrl: './app.component.html',

  styleUrl: './app.component.css'

})

export class AppComponent {

  title = 'prog10';

}

| **Import** | **From** | **Purpose** |
| --- | --- | --- |
| Component | @angular/core | Defines Angular components using metadata |
| RouterModule | @angular/router | Enables routing, navigation, and route configuration |
| RouterOutlet | @angular/router | Directive that acts as a placeholder for routed components |

**🔹 1. import { Component } from '@angular/core';**

**✅ What is @angular/core?**

* This is the **core Angular package**.
* It provides essential Angular features like components, directives, services, decorators, etc.

**📦 Component**

* Component is a **decorator** used to define an Angular component.
* It tells Angular: “Hey, this is a component, and here’s its metadata (selector, template, style, etc.)”.

**🔹 2. import { RouterModule, RouterOutlet } from '@angular/router';**

**✅ What is @angular/router?**

* It’s the **Angular Router module**—the official Angular library for handling navigation between views/pages.
* It gives you routing tools like routerLink, RouterModule, and <router-outlet>.

**📦 RouterModule**

* A **NgModule** that provides all the routing features.
* You use it to configure your routes with RouterModule.forRoot(routes) or RouterModule.forChild(routes).

**🔷 What is NgModule in Angular?**

**✅ A NgModule is a container that groups related components, directives, pipes, and services together.**

It helps Angular know:

* What parts of your app exist
* What they depend on
* What should be shared or hidden

****

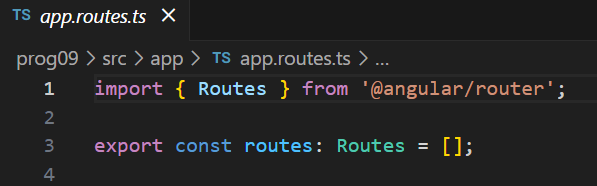
**📦 RouterOutlet**

* This is the **directive behind** the <router-outlet> element in HTML.
* You typically don’t use RouterOutlet directly in code. Angular uses it internally to render components as per the route.

**Step 5:**

**Now we need to modify app.routes.ts**

**From this :**

** To this:**

import { Routes } from '@angular/router';

import { HomeComponent } from './home/home.component';

import { ProfileComponent } from './profile/profile.component';

export const routes: Routes = [

        {

            path:'',

            component: HomeComponent,

            title:'Home',

        },

        {

            path:'profile',

            component: ProfileComponent,

            title:'Profile'

        }

];

**As we are here to form routes for home and profile that’s what we are setting here.**

**Step 6: After setting the routes we need to include it in all our component files 1.home.component.ts,header.component.ts, profile.component.ts. We do this by importing in each of these files**

import { RouterModule } from '@angular/router';

**Also update the imports:**

imports: [RouterModule],

**in each of the files.**

**Step 7: Our header page is having navigation links hence we have to update it.**

**Header.component.js:**

<nav class="navbar navbar-expand-lg bg-body-tertiary">

    <div class="container-fluid">

      <a class="navbar-brand" [routerLink]="['/']">Class II A</a>

      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

        <span class="navbar-toggler-icon"></span>

      </button>

      <div class="collapse navbar-collapse" id="navbarSupportedContent">

        <ul class="navbar-nav me-auto mb-2 mb-lg-0">

          <li class="nav-item">

            <a class="nav-link" [routerLinkActiveOptions]="{exact:true}" routerLinkActive="active" aria-current="page" [routerLink]="['/']">Home</a>

          </li>

          <li class="nav-item">

            <a class="nav-link" [routerLinkActiveOptions]="{exact:true}" routerLinkActive="active" [routerLink]="['/profile']">Profile</a>

          </li>

        </ul>

      </div>

    </div>

  </nav>

**Step 8: save all**

**Step 9:ng serve 🡪in the terminal.**

**WEEK 10:**

**What is a REST API?**

**REST API** stands for:

**R**epresentational **S**tate **T**ransfer **A**pplication **P**rogramming **I**nterface

It's a way for two applications (like **frontend Angular** and **backend server**) to **talk to each other over the internet** using HTTP (like GET, POST, PUT, DELETE).

**📦 Example Scenario (like your Angular project)**

Let’s say:

* Your Angular app needs to show a user’s profile.
* The data (name, email, etc.) is stored on a **server**.

Instead of hardcoding the data, Angular will:

* Make a **GET request** to a REST API like:

http://localhost:3000/users/1

The REST API responds with data in **JSON** format:

{

"id": 1,

"name": "John Doe",

"email": "john@example.com"

}

Then Angular displays this on your webpage.

**🔁 Common HTTP Methods in REST API**

| **Method** | **What it Does** | **Example Use** |
| --- | --- | --- |
| GET | Read/fetch data | Get user profile |
| POST | Create new data | Register a new user |
| PUT | Update entire data | Update full profile |
| PATCH | Update part of the data | Change only the email |
| DELETE | Delete data | Remove a user |

**📱 Example 2: Student Management System**

You are building a **Student Dashboard** using Angular.

The backend (server) has a REST API that manages student data.

**🎯 Objective:**

Show student details (name, class, marks) by **fetching from the server**.

**🛠️ Angular Frontend:**

When the Angular app starts, it sends a **GET request** to:

http://localhost:3000/students/101

**🧾 REST API (Backend Server) responds:**

{

"id": 101,

"name": "Riya Sharma",

"class": "10th Grade",

"marks": 92

}

**🎨 Angular displays on the screen:**

Name: Riya Sharma

Class: 10th Grade

Marks: 92

**Week 10: Fetch user details from server using REST API and show in profile menu using Angular.**

For week 1o, After following up all the steps from week 9 we need to just modify our

Profile.component.html:

<div class="d-flex justify-content-center align-items-center vh-100">

    <div class="card shadow-sm w-50 mt-5 p-2 " \*ngIf="user">

    <img class="avatar d-block mx-auto" src="https://cdn-icons-png.freepik.com/512/5733/5733290.png" alt="Icon"/>

    <h2 class="text-center">hi,{{user.name}}</h2>

    <table class="table table-bordered mt-3">

            <tr>

                <th>Username</th>

                <td>{{user.username}}</td>

            </tr>

            <tr>

                <th>Email</th>

                <td>{{user.email}}</td>

            </tr>

            <tr>

                <th>Phone</th>

                <td>{{user.phone}}</td>

            </tr>

            <tr>

                <th>City</th>

                <td>{{user.address?.city}}</td>

            </tr>

            <tr>

                <th>ZIP Code</th>

                <td>{{user.address?.zipcode}}</td>

            </tr>

            <tr>

                <th>Website</th>

                <td>{{user.website}}</td>

            </tr>

    </table>

</div>

</div>

**And**

**profile.component.js**

import { CommonModule } from '@angular/common';

import { HttpClient,HttpClientModule } from '@angular/common/http';

import { Component,OnInit } from '@angular/core';

@Component({

  selector: 'app-profile',

  standalone:true,

  imports: [HttpClientModule,CommonModule],

  templateUrl: './profile.component.html',

  styleUrl: './profile.component.css'

})

export class ProfileComponent implements OnInit{

  user:any

  constructor(private http:HttpClient){

  }

  ngOnInit(): void {

    this.http.get('https://jsonplaceholder.typicode.com/users/4').subscribe((data:any)=>{

      console.log(data);

      this.user=data;

    });

  }

}

**✅ Step-by-Step Explanation**

ts

import { CommonModule } from '@angular/common';

📦 This imports **basic Angular directives** like \*ngIf, \*ngFor, etc.  
🧠 Think of it as: *"Bring in the common tools needed for an Angular template to work."*

ts

import { HttpClient, HttpClientModule } from '@angular/common/http';

📦 This gives us the power to **fetch data from a REST API** (like using fetch() in JavaScript).

* HttpClientModule: Registers HTTP features in the app.
* HttpClient: Lets us actually **make GET/POST requests**.

🧠 Think of it like: *"We're installing the internet in our app!"*

ts

import { Component, OnInit } from '@angular/core';

📦 Component is used to **create a UI block** (like a page or section).  
📦 OnInit is a **lifecycle hook** — code inside ngOnInit() runs **automatically when the component loads**.

🧠 Like: *"Let’s make a profile box that runs code when it shows up."*

ts

@Component({

selector: 'app-profile',

standalone: true,

imports: [HttpClientModule, CommonModule],

templateUrl: './profile.component.html',

styleUrl: './profile.component.css'

})

🧩 This tells Angular:

“I’m creating a new **standalone** component. Here's how to render it, style it, and what tools it uses.”

* selector: Custom HTML tag like <app-profile>
* standalone: true: Doesn’t need to be declared in a module
* imports: What this component depends on (HTTP + Angular templates)
* templateUrl: HTML file for the component’s view
* styleUrl: CSS file for styles

🧠 Like: *"This is my self-contained profile card, and here’s what it looks like."*

ts

export class ProfileComponent implements OnInit {

🧠 This defines the **actual logic** for the component.  
It says: *"We’re creating a ProfileComponent that does something when it loads."*

ts

user: any

🧠 We’re making a variable user to **store the fetched user data** later.

ts

constructor(private http: HttpClient) { }

This is called **Dependency Injection**.

📦 HttpClient is injected so we can use it to make HTTP requests.

🧠 Like: *"Give this component internet access, please!"*

ts

ngOnInit(): void {

🧠 This function runs **automatically** when the component is shown.  
It’s the best place to load data.

ts

this.http.get('https://jsonplaceholder.typicode.com/users/4')

🧠 We are making a **GET request** to a free fake API that gives us user data (user #4).

ts

.subscribe((data: any) => {

console.log(data);

this.user = data;

});

📦 .subscribe() is used to **handle the response** when it comes back.  
We:

* Print it in the console
* Store it in the user variable to use it in the HTML

🧠 Like: *"When data arrives, save it and show it!"*