Angular Introduction

Frontend Frameworks

Understanding Single Page Applications (SPAs)

Overview of Angular

Setting up the development environment

Introduction to typescript

Creating your first Angular app

Project Registration

Team Size: 4 - 6 members

Registration Form: Click here to access the form, the deadline is 2/11/2025

Idea Selection: choose one idea from this list

What are frontend frameworks ?

Frontend frameworks are essential tools in modern web development.

They offer pre-written code modules, and predefined ways to solve common development challenges.

Frameworks handle low-level details, allowing developers to focus on building features rather than reinventing the wheel.

They are used to build single page applications

What is a single page application ?

A single page application loads a single HTML page and dynamically updates its content as the user interacts with the app, without requiring a full page reload.

Traditional multi-page Web Applications (Old Style):

- Every time you click a link, the browser sends a request to the server.
- The server processes the request, generates a new HTML page, and sends it back.
- The browser unloads the current page and loads the new one.

Drawbacks:

- **Slower Navigation**: Each action requires a full page reload, which can be slow.
- **Disrupted Experience**: Reloading the page interrupts the user's flow.
- **Increased Server Load**: The server has to render and send complete pages each time.

What is a single page application ?

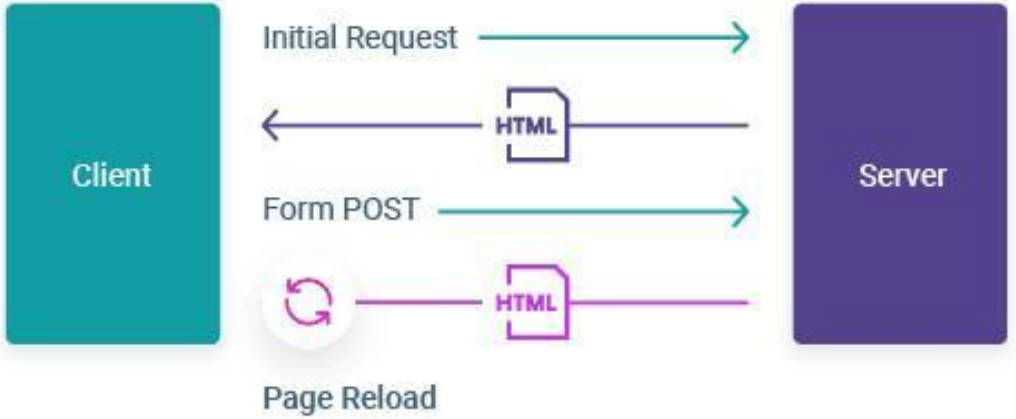
Single Page Applications Flow:

- The browser loads the initial HTML, CSS, and JavaScript files once.
- When you interact with the app, JavaScript handles updating the content dynamically.
- Data is fetched from the server using APIs, and only the necessary data (JSON) is transferred.

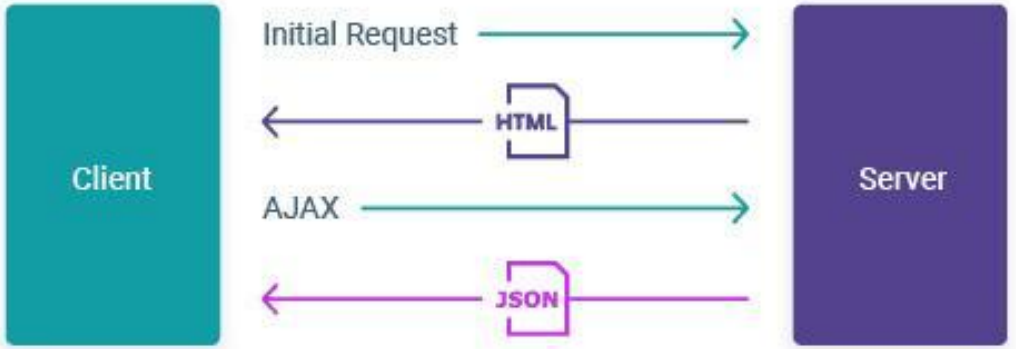
Benefits:

- **Faster Interactions**: Only parts of the page are updated, resulting in quicker responses.
- **Smooth User Experience**: No full page reloads mean interactions feel seamless.
- **Reduced Server Load**: Less data is transferred with each interaction.

Multi-Page Lifecycle



SPA Lifecycle



Frontend frameworks features

The most common features frontend frameworks provide are:

Component-Based: Applications are built using independent, self-contained components, making development more manageable.

Data Binding: Synchronizes data between the model (State) and the view (UI), reducing the amount of boilerplate code.

Routing and Navigation: navigation within single-page applications without full page reloads.

State Management: Handle the state of the application efficiently, ensuring consistency across different parts of the UI.

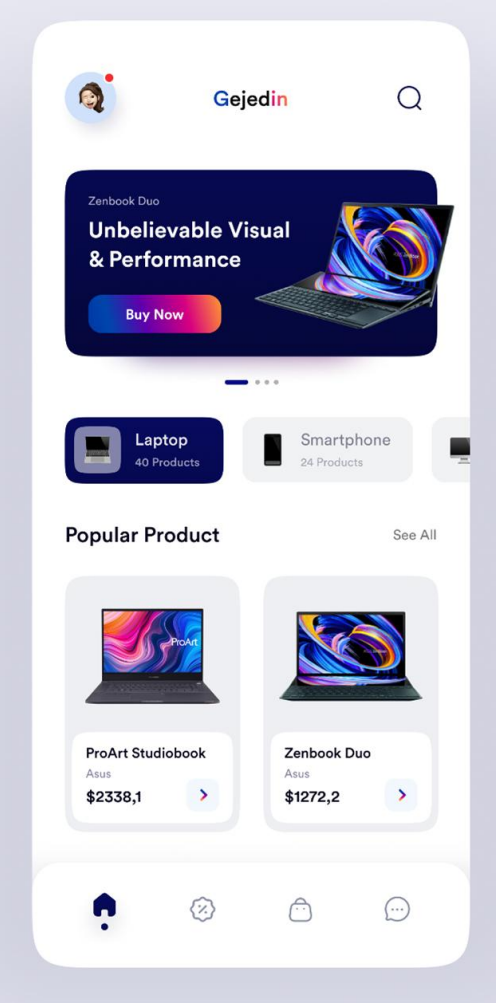
Component Based

Component-based applications are divided into small, reusable pieces called components.

Each component encapsulates its own:

- Structure (HTML)
- Behavior (TypeScript/JavaScript)
- Styling (CSS)

This makes each component self-contained easy to manage, and reuse. Example: A product Card

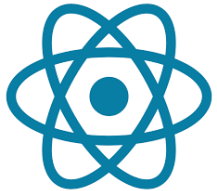


Data Binding

Data binding synchronizes data between the model (the code state) and the view (the UI). Angular uses data binding to keep the view updated with the latest data. Without needing to interact with the Document Object Model (DOM) using **getElementById(id)** for example.

Most popular frameworks

React is a lightweight and flexible JavaScript library focused on building component-based user interfaces that promotes reusability. It has high popularity and a wide ecosystem, offering developers a vast array of resources and community support. However, it often requires additional third-party libraries for state management and routing, which can add complexity to projects.



Angular is a full-featured framework, offering built-in tools like dependency injection, routing, and form handling to accelerate complex application development. It is widely used by large enterprises.

Vue.js is a framework that is easy to learn and integrate, offering excellent developer experience. However, it is less popular than React and Angular, leading to fewer job opportunities and a smaller ecosystem.

Why Choose Angular ?

Comprehensive: Everything you need is included.

TypeScript: Uses a language that offers type safety.

Strong Community: Lots of job opportunities and help available.

Backed by Google: Regular updates and improvements.

Angular Development Environment

To get started with Angular, you need to set up a development environment that includes **Node.js**, the **Angular CLI**, and a code editor such as **Visual Studio Code.**

What is Node.js

Node.js is a JavaScript runtime environment that allows you to run JavaScript code outside of a web browser.

Traditionally, JavaScript was used mainly for front-end development, executed in the browser to create interactive web pages.

However, with Node.js, JavaScript can also be used for server-side development, enabling developers to create full-stack applications using a single language

Node.js includes **npm** (Node Package Manager), which allows developers to easily install and manage dependencies in their projects. Similar to **pip** in python.

It is needed for Angular development because the **Angular CLI** depends on it for executing commands, running the local development server, managing packages with npm, and building/compiling the project. Node.js can be downloaded from here

Installing the Angular CLI

The Angular Command Line Interface (CLI) is a tool that provides commands to generate components, services, and modules, run the development server, and build the project for deployment.

Once Node.js is installed, you can install the Angular CLI globally using npm: `npm install -g @angular/cli`

The -g flag means that the CLI will be available globally on your system.

Verify the Angular CLI installation using: `ng version`

If the installation was successful, you should see the version of the Angular CLI displayed.

Installing VSCode

Visual Studio Code (VSCode) is a popular choice due to its wide range of extensions for Angular development, lightweight nature, and built-in Git support.

After installing VSCode, install the Angular language service extension which provides autocompletion, error detection, and navigation features specifically tailored for Angular projects.

Introduction to Typescript

In Angular development, the language used is **TypeScript**, which is a superset of JavaScript.

This means that TypeScript builds on top of JavaScript by adding new features, particularly related to type safety.

JavaScript: is a dynamically typed language, meaning types are determined at runtime, which can lead to type-related errors that are only caught when the code is executed.

TypeScript: It adds static typing, allowing you to define variable types (e.g., string, number, boolean) at compile time. This means errors can be caught during development before the code is run.

TypeScript needs to be compiled (transpiled) into JavaScript because browsers do not natively understand TypeScript. The TypeScript compiler (tsc) converts TypeScript code into JavaScript.

JavaScript Without Types

```
// JavaScript is a dynamically typed language,
// which means that variables can be reassigned to any type.
let number = 1; // Variables have no types
console.log(number); // Outputs 1

number = "Hello"; // Variables can be reassigned to any type
console.log(number); // Outputs Hello
```

JavaScript Without Types

```
// Functions can also be called with any type of arguments.
// This can lead to unexpected results.
function sum(numberOne, numberTwo) {
    return numberOne + numberTwo;
}

console.log(sum(1, 2)); // Outputs 3
// The number 1 is converted to a string and concatenated with the string "2"
console.log(sum(1, "2")); // Outputs 12
console.log(sum("3", "3")); // Outputs 33
```

Typescript type annotations

TypeScript enhances code safety by adding static typing with type annotations, which allows developers to specify and enforce types for variables, function parameters, and return values.

Let's see how TypeScript can make the original JavaScript code safer by preventing unintended type-related errors.

The following code will be written inside a **.ts** file instead of a **.js** file

Typescript type annotations

```
// TypeScript has a type system that helps you to catch errors during
development.
// A variable's type is defined using a colon after the variable name.
let number: number = 1;
console.log(number); // Outputs 1
// Error: Type 'string' is not assignable to type 'number'.
number = "Hello";
// TypeScript can infer types based on the value assigned to the variable.
let name = "Basem"; // name has a type of string
name = 1; // Error: Type 'number' is not assignable to type 'string'.
```

Function type annotations

```
// Function parameters can have types as well.
// The function return type is defined using a colon after the parameters.
function sum(numberOne: number, numberTwo: number): number {
    return numberOne + numberTwo;
}
console.log(sum(1, 2)); // Outputs 3
// Error: Argument of type 'string' is not assignable to parameter of type
'number'.
console.log(sum(1, "2"));
console.log(sum("3", "3"));
```

Typescript variable declaration

In TypeScript variables can be declared using 3 keywords: `const - let - var`

`const` is used when you want to declare a variable that should not be changed after its initial assignment. Once you set the value, you cannot reassign it to something else, although if the variable is an object or an array, you can still change the contents

`let` is used to declare a variable that can be updated or changed. It is "block-scoped," meaning the variable only exists within the block (such as a loop, conditional, or function) where it is declared. This helps to avoid accidentally using or changing the variable outside of that specific block, making the code more predictable.

`var` is the older way to declare variables in JavaScript and TypeScript, and it behaves differently from let and const. It is "function-scoped," which means it is accessible throughout the entire function, even if declared inside a block. This can cause unexpected behavior because var is also subject to "hoisting," where the variable is moved to the top of its scope, potentially leading to bugs.

Typescript variable declaration

```
const count = 1;
count = 2; // Variables declared with const cannot be reassigned.

let name = "Basem";
name = "Ali"; // Variables declared with let can be reassigned.
```

Typescript variable declaration

```
function isEven(value: number) {
    // Variables declared with var are function-scoped.
    // This means that they are accessible anywhere within the function.
    // As if the variable was declared at the top of the function.
    even = false;
    // Variables declared with let are block-scoped.
    // This means that they are only accessible within the block they are declared in.
    odd = false; // Error: Cannot find name 'odd'.
    if (value % 2 === 0) {
        var even = true;
        let odd = false;
    }
}
```

Typescript data types

```typescript
const isEven: boolean = true;
const sum: number = 0;
const message: string = "Hello";
const numbers: number[] = [1, 2, 3];
const positions: Array<number> = [1, 2, 3];
const mixed: (string | number)[] = [1, "Ali", 3];
// age has a type of number
const age = 1;
// The type any can be used to represent any type.
let x: any = 1;
x = "message";
```

Null & Undefined Types

In TypeScript, the type **`undefined`** indicates that a variable has been declared but hasn't been assigned a value.

It also represents the default return value of functions without a specified return.

It occurs automatically for uninitialized variables or absent function parameters.

On the other hand, the type **`null`** is an explicit assignment used to denote the intentional absence of any value, often used to clear references or indicate an empty state.

If a variable can be **`null`** or **`undefined`** it must specified in its type

```
let name: string | null = "Basem";
name = null; // name can be null.
```

Equality Checks

In TypeScript (and JavaScript), equality (==) and strict equality (===) are used to compare values, but they work differently.

Equality (==) : Compares two values for equality after performing type conversion if the types are different. This means that it tries to convert the values to the same type (usually a number) before comparing.

```
console.log(5 == "5"); // Output: true (string "5" is converted to number 5)
// null and undefined are considered equal to each other but not equal to any other values.
console.log(null == undefined); // Output: true
console.log(null == 0); // Output: false
```

Strict Equality (===): Compares both the value and the type without performing type conversion. The values must be of the same type and have the same value to be considered equal.

```
console.log(5 === "5"); // Output: false (different types: number vs string)
console.log(null === undefined); // Output: false (different types: null vs undefined)
```

In general always use strict equality to avoid unexpected bugs.

Loops

```
const numbers = [4, 8, 12, 16, 20,
24];
// Normal for loop
for (let i = 0; i < numbers.length;
i++){
    console.log(numbers[i]);
}
// For of loop
// element is the value of the current
element
for (const element of numbers) {
    console.log(element);
} // Output: 4, 8, 12, 16, 20, 24
```

```
// For in loop
// Used to iterate over indexes of an
array, or keys of an object.
for (const index in numbers) {
    console.log(index);
} // Output: 0, 1, 2, 3, 4, 5
const object = { name: "Ali", age: 21 };
for (const key in object) {
    console.log(key);
} // Output: name, age
```

Optional function parameters

```
function greet(firstName: string, lastName: string): string {
    return "Hello " + firstName + " " + lastName;
}
// Functions must be called with the correct number of arguments.
greet("Omar"); // Error: Expected 2 arguments, but got 1.

// To make a parameter optional, add a question mark after the parameter name.
function hello(firstName: string, lastName?: string): string {
    return "Hello " + firstName + " " + lastName;
}

let greetings = hello("Omar");
// Output: Hello Omar undefined
console.log(greetings);
```

Classes

```
class Person {
    // Public variables are accessible from outside the class
    public name: string; // public is the default access modifier
    private age: number; // private variables are only accessible within the class
    protected address: string; // protected variables are accessible within the class and its subclasses
    // The constructor is defined using the constructor keyword
    constructor(name: string, age: number, address: string) {
        this.name = name;
        this.age = age;
        this.address = address;
        // Any function call within the class must be called using this.
        this.printName();
    }
    private printName() {
        console.log(this.name);
    }
}
```

Inheritance

```
class Employee extends Person {
    private salary: number;
    constructor(name: string, age: number, address: string) {
        // The super keyword is used to call the constructor of the parent class.
        super(name, age, address);
    }
}
const ali:Person = new Person("Ali", 25, "Cairo");
const omar:Employee = new Employee("Omar", 30, "Giza");
// Objects can be created using object literals by specifying key-value pairs
const object = { productName: "Laptop", price: 1000 };
console.log(object.productName); // Output: Laptop
```

Interfaces

Interfaces are used to define the **shape or structure** of an object. They specify the types of properties and methods that an object should have, but they do not implement any functionality. Interfaces help enforce type checking, making code more predictable and easier to maintain.

```
interface Transport {
    engineType: string;
    move(): void;
}
```

```
class Car implements Transport {
    engineType: string;
    constructor(engineType: string) {
        this.engineType = engineType;
    }
    move() {
        console.log("Car is moving");
    }
}
```

Knowledge Check

```
// Can address be assigned to null
?
let address:string = "Cairo";
address = null;

// Are undefined and null the same
?
console.log(undefined === null);

// Can array contain different
types ?
const array = [1, 2, 3];
array.push("Name");
```

```
interface Point {
    x: number;
    y: number;
}
// Can origin have the type of
Point ?
const origin = { x: 0, y: 0 };
```

Creating An Angular Project

To create a new Angular project, use the ng new command followed by your project's name:

```
ng new app
```

Starting with Angular version 15, projects use something called standalone components by default, which simplify the project structure and reduce boilerplate code.

However, since most existing projects still use a module-based structure, we'll focus on that approach. The differences between the two structures are straightforward, making it easy to switch to standalone components later if you wish.

To create a new project without standalone components (using the module-based structure), use:

`ng new app --standalone=false` Then accept the default options by pressing enter.

```
ahmed@omen MINGW64 ~/Downloads/HCI Lab 3 Angular
$ ng new app --standalone=false
? Which stylesheet format would you like to use? CSS             [ https://developer.mozilla.org/docs/Web/CSS                    ]
? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? no
CREATE app/angular.json (2937 bytes)
CREATE app/package.json (1073 bytes)
CREATE app/README.md (1091 bytes)
CREATE app/tsconfig.json (1045 bytes)
CREATE app/.editorconfig (331 bytes)
CREATE app/.gitignore (629 bytes)
CREATE app/tsconfig.app.json (439 bytes)
CREATE app/tsconfig.spec.json (449 bytes)
CREATE app/.vscode/extensions.json (134 bytes)
CREATE app/.vscode/launch.json (490 bytes)
CREATE app/.vscode/tasks.json (980 bytes)
CREATE app/src/main.ts (256 bytes)
CREATE app/src/index.html (302 bytes)
CREATE app/src/styles.css (81 bytes)
CREATE app/src/app/app-routing.module.ts (255 bytes)
CREATE app/src/app/app.module.ts (411 bytes)
CREATE app/src/app/app.component.html (20239 bytes)
CREATE app/src/app/app.component.spec.ts (1069 bytes)
CREATE app/src/app/app.component.ts (214 bytes)
CREATE app/src/app/app.component.css (0 bytes)
CREATE app/public/favicon.ico (15086 bytes)
✓ Packages installed successfully.
    Successfully initialized git.
```

Angular Project Structure

package.json: Lists project details and dependencies, and contains commands for running the app.

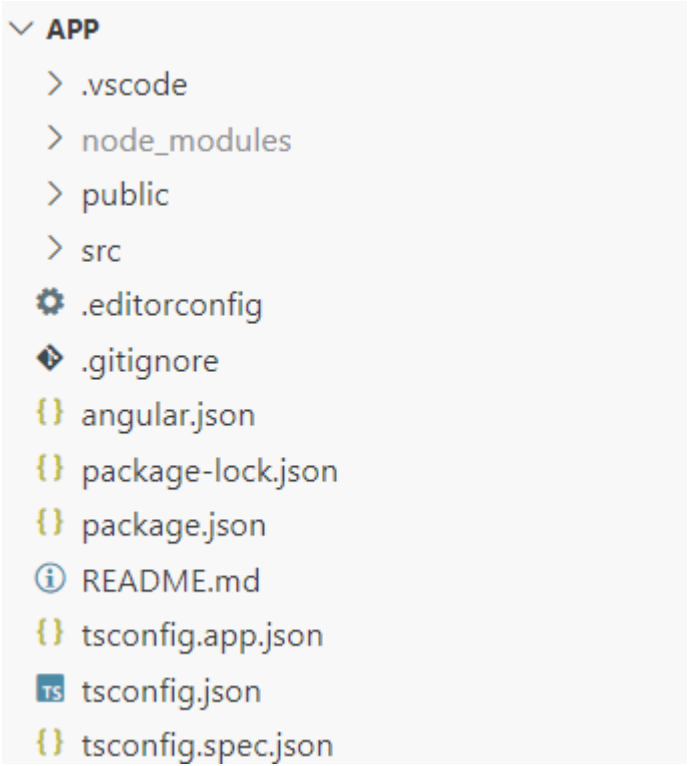
node_modules: Stores all the project's installed dependencies.

angular.json: Configures how the Angular app is built and run.

tsconfig.json: settings for how TypeScript compiles the project.

src: The folder where the app's code lives

public: Holds static files like images and fonts for the app.



The Source Folder

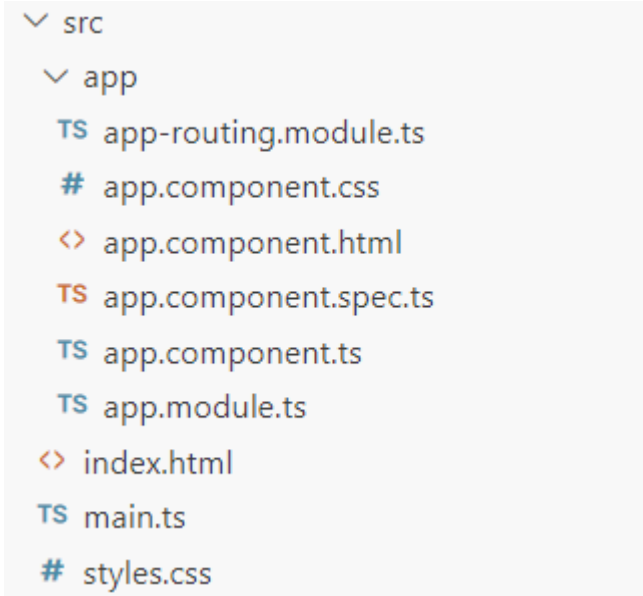
main.ts: The entry point of the application. It bootstraps (starts) the Angular app.

index.html: The main HTML file for the app. It loads the Angular application into the browser.

styles.css: A global stylesheet that applies styles across the entire app.

app.module.ts: This file is the main module for the app. It lists all the parts (components) the app has, brings in extra tools or features (called modules), and connects any services the app might need (like getting data from a server).

app.component.ts: The first component that shows up on the screen when you open the app. It handles what you see and how the app responds when you interact with it (like clicking buttons or entering data).



The main.ts file

platformBrowserDynamic() is a function in Angular that sets up the environment for your application to run in a web browser.

It returns a platform object (PlatformRef) that provides the bootstrapModule() method, which you use to compile and launch your root module (AppModule), effectively starting your Angular application.

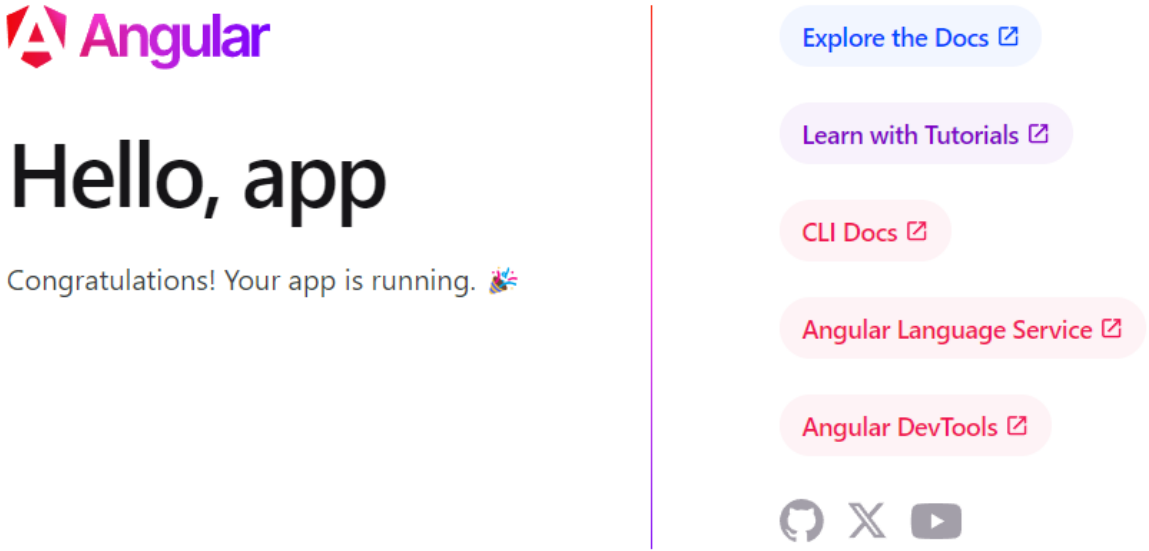
```
import { platformBrowserDynamic } from
'@angular/platform-browser-dynamic';

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

platformBrowserDynamic()
  .bootstrapModule(AppModule, {
    ngZoneEventCoalescing: true,
  })
  .catch((err) => console.error(err));
```

Running The Application

To Start the Angular application use: `ng serve --open`



Thank You ;)

Any Questions ?