**Angular**

**Angular** is a popular open-source framework developed by Google for building dynamic, single-page web applications (SPAs). Angular is a framework which provides complete developer platform for Single Page Application and Progressive Web Applications. It uses TypeScript, which is a superset of JavaScript, and provides a powerful set of tools for developing complex applications with a focus on maintainability, scalability, and performance.

**Single Page Application:** Every time a new route triggers, it wont load whole page, this feature allows to create whole project in one HTML called SPA.

**Progressive Web App:** These apps work similar in different platforms.

**Framework:** Framework contains all the packages needed for the application development. (Unlike library which is pre written code used for specific purpose.)

**Key Concepts of Angular:**

1. **Component-based Architecture**:
   * A component consists of an HTML template for the UI, a TypeScript class for handling logic, and metadata that defines how the component should behave.
   * Angular applications are built using components, which are self-contained building blocks with their own logic and templates.
2. **Modules**:
   * Angular applications are organized into modules. A module is a container that groups related components, services, and other code.
   * The root module, AppModule, bootstraps the Angular application.
3. **Templates and Data Binding**:
   * Angular uses templates to define the view, and it supports powerful data-binding mechanisms.
     + **Interpolation**: Binding data from the component to the template using {{ }}.
     + **Property Binding**: Setting properties of DOM elements with [property]="expression".
     + **Event Binding**: Handling user actions with (event)="handler()".
     + **Two-way Binding**: Synchronizing data between the component and the UI using [(ngModel)].
4. **Services and Dependency Injection**:
   * Angular promotes the use of services to share logic between components. Services are typically used for things like fetching data from APIs.
   * Dependency injection (DI) is a key feature, allowing Angular to efficiently provide instances of services where needed.
5. **Routing**:
   * Angular provides a powerful routing system to build single-page applications. The Angular Router maps URLs to components and helps manage navigation within the app without reloading the page.
6. **Directives**:
   * Directives are special markers in the DOM that tell Angular to attach a specific behavior to an element. Angular provides built-in directives like \*ngIf and \*ngFor, and you can create custom directives for reusable behaviors.
7. **Forms**:
   * Angular has two approaches to managing forms:
     + **Template-driven forms**: Simple and easy to use, but less scalable.
     + **Reactive forms**: More powerful and suitable for complex scenarios, offering better control over form validation and dynamic form creation.
8. **Pipes**:
   * Pipes are used to transform data in templates. For example, the | date pipe formats a date, and the | uppercase pipe converts text to uppercase. You can also create custom pipes.
9. **Testing**:
   * Angular encourages a test-driven development approach. It comes with built-in tools for unit testing components and services using frameworks like Jasmine and Karma.
10. **Performance Optimization**:
    * Angular provides various techniques for optimizing performance, including Ahead-of-Time (AOT) compilation, lazy loading of modules, and change detection strategies.

**Advantages of Angular:**

* **Structured Code**: Angular’s architecture promotes clean, maintainable, and testable code.
* **TypeScript**: Leveraging TypeScript allows for better tooling, refactoring, and catching errors at compile time.
* **Large Ecosystem**: Angular has a vast ecosystem of tools, libraries, and extensions.
* **Support from Google**: Being developed and maintained by Google ensures regular updates, strong community support, and stability.

**Getting Started with Angular:**

To start building Angular applications, you typically use the Angular CLI (Command Line Interface), which provides a powerful set of tools to scaffold projects, generate components, and run development servers.

1. **Install Angular CLI**:

npm install -g @angular/cli

1. **Create a New Angular Application**:

ng new my-angular-app

cd my-angular-app

ng serve

1. **Develop**: You can now start building your application by creating components, services, and other features using the Angular CLI and writing code in TypeScript.

Angular is widely used in the industry for building large-scale applications, and its comprehensive framework helps developers manage complex applications with ease.

**Advantages of Angular**:

1. **Component-based Architecture**: Promotes reusability, modularity, and organized code structure.
2. **Two-way Data Binding**: Simplifies synchronization between the model and the view.
3. **TypeScript Support**: Provides static typing, better tooling, and early error detection.
4. **Dependency Injection**: Enhances modularity and testability by decoupling components from their dependencies.
5. **Powerful CLI**: Automates tasks like project setup, scaffolding, testing, and building applications.
6. **Built-in Routing and State Management**: Supports advanced navigation and state handling in single-page applications.
7. **Comprehensive Documentation and Community**: Offers extensive resources and community support for problem-solving and learning.
8. **Performance Optimizations**: Features like Ahead-of-Time (AOT) compilation and lazy loading improve app performance.
9. **Cross-platform Development**: Enables building web, mobile (via Ionic), and desktop applications.
10. **Enterprise-ready**: Suited for large-scale, complex applications due to its robustness and scalability.

**Disadvantages of Angular**:

1. **Steep Learning Curve**: Complex concepts can be overwhelming for beginners.
2. **Verbose Code**: Requires more boilerplate code compared to other frameworks.
3. **Complexity**: Overkill for small or simple applications due to its extensive features.
4. **Performance Issues**: May struggle with very large applications if not optimized properly.
5. **Frequent Updates and Breaking Changes**: Updates often introduce breaking changes that require codebase adjustments.
6. **Heavy Framework**: Larger bundle size can affect initial load times compared to lighter frameworks.
7. **Complex Integration with Legacy Systems**: Challenging to integrate with non-modern JavaScript frameworks or legacy systems.
8. **Verbose Testing**: Writing tests, especially for complex components, can be time-consuming and complicated.
9. **Opinionated Structure**: Enforces specific conventions and architecture, limiting flexibility for developers who prefer more freedom.

**Q: Difference Between Angular and AngularJS**

|  |  |  |
| --- | --- | --- |
| **Category** | **Angular** | **AngularJS** |
| Creator | Google | Google |
| Language supported | JavaScript and Typescript | JavaScript |
| Mobile Development friendly | Compatible for mobile-development | Not compatible |
| Architecture | It uses components and directives. | Support model-view-controller (MVC) and model-view-view-model (MVVM) architectures. |
| Testing | Supports unit testing with Karma | Testing is done through third-party applications |
| CLI | Comes with Angular CLI | No support for CLI |
| Dependency Injection | Uses hierarchal dependency injection | Does not use dependency injection. Uses directives |
| Performance | Supports server-side rendering which offers a speedy performance | Overall performance is slow as compared to Angular |
| Example | Gmail and Upwork | Netflix and Lego |

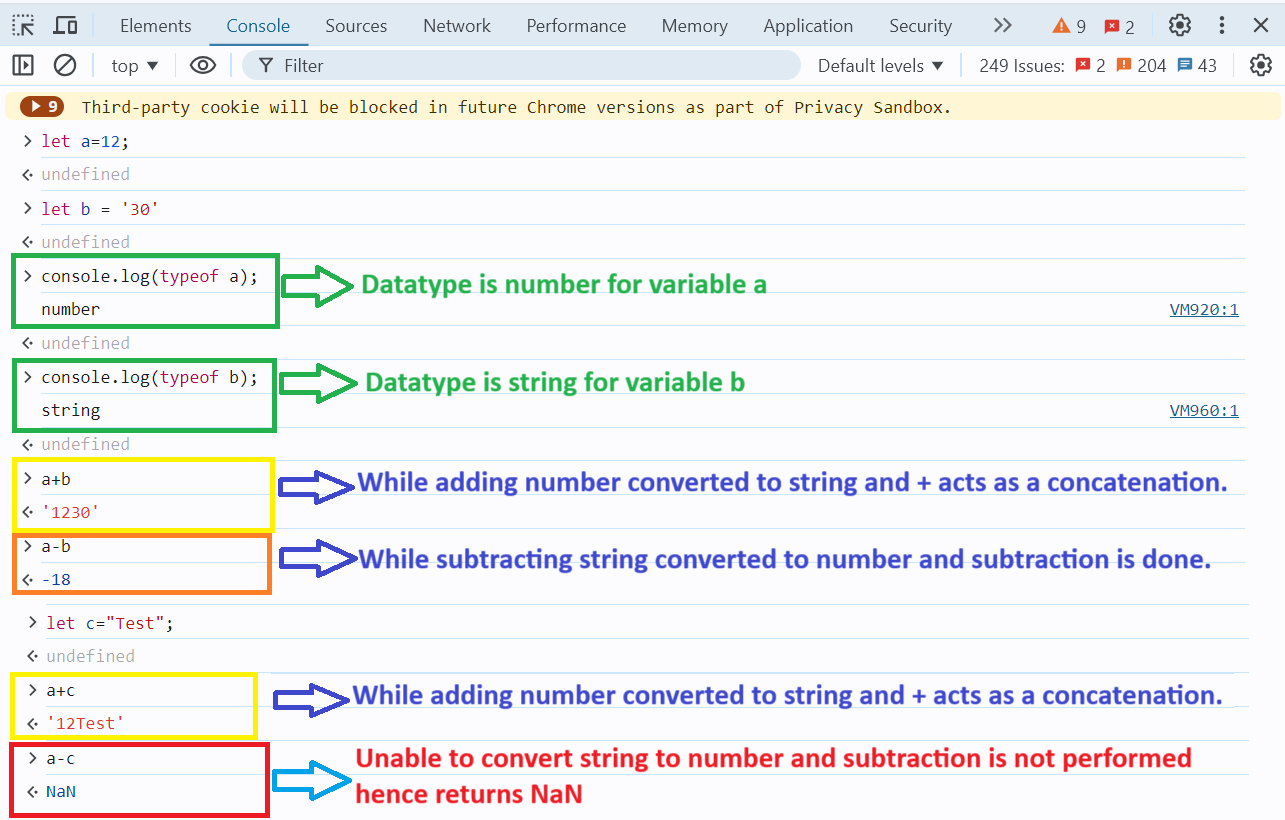
**Q: Difference Between React & Angular:**

|  |  |  |
| --- | --- | --- |
| **Field** | **React** | **Angular** |
|  |  |  |
|  | It is a **JavaScript library**. | Angular is **a framework.** |
| **Written** | React.js written in JavaScript. | Written in Microsoft’s Typescript language, which is a superset of ECMA Script 6 (ES6). |
| **Dependency Injection** | React.js Does not use the DI concept. | Angular Hierarchical DI system used. |
| **Routing** | Routing is not easy in React JS. | Routing is comparatively easy as compare to React JS. |
| **Scalability** | It is highly scalable. | It is less scalable than React JS. |
| **Data Binding** | It supports Uni-directional data binding that is one way data binding. | It supports Bi-directional data binding that is two data binding. |
| **DOM** | It has virtual DOM. | It has regular DOM. |
| **Testing** | It supports Unit Testing. | It supports both Unit testing and Integration testing. |
| **Used as** | React.js is a JavaScript library. As it indicates react js updates only the virtual DOM is present and the data flow is always in a single direction. | Angular is a framework. Angular updates the Real DOM and the data flow is ensured in the architecture in both directions. |
| **Released** | It was released in 2013. | It was released in 2010. |
| **Architecture** | React.js is more simplified as it follows MVC ie., Model View Control. This like angular includes features such as navigation but this can be achieved only with certain libraries like Redux and Flux. Needs more configuration and integration. | The architecture of angular on the other hand is a bit complex as it follows MVVM models ie., Model View-ViewModel. This includes lots of tools and other features required for navigation, routing, and various other functionalities. |
| **Performance** | React.js holds JSX hence the usage of HTML codes and syntax is enabled. But this doesn’t make react js a subset of HTML. This is purely JavaScript-based. | Angular, on the other, is a mere subset of HTML. |
| **Preference** | React.js is preferred when the dynamic content needed is intensive. As react js holds more straightforward programming and since it is reliable many apps such as Instagram, Facebook, and Twitter still prefer to react js over angular. | Angular is platform-independent and hence is compatible to work in any platform. Hence, the HTML app which is compatible with all the browsers can prefer angular. One major app which uses angular is YouTube. |

**Q: Difference Between TypeScript and JavaScript**

* TypeScript is known as an Object-oriented programming language whereas JavaScript is a prototype-based language.
* TypeScript has a feature known as Static typing but JavaScript does not support this feature.
* TypeScript supports Interfaces but JavaScript does not.

|  |  |  |
| --- | --- | --- |
| **Feature** | **TypeScript** | **JavaScript** |
|  | Developed by Microsoft | Developed by Google |
| Typing | Provides static typing. | Dynamically typed. |
| Tooling | Comes with IDEs and code editors. | Limited built-in tooling. |
| Syntax | Similar to JavaScript, with additional features like static typing. | Standard JavaScript syntax. |
| Compatibility | Backward compatible with JavaScript. | Cannot run TypeScript in JavaScript files. |
| Debugging | Stronger typing can help identify errors. | May require more debugging and testing. |
| Type | Object Oriented Programming Language. | Prototype Based Language. |
| Learning curve | Can take time to learn additional features. | Standard JavaScript syntax is familiar. |
| Example | Let a=20;  A=’Mahesh’ //Invalid | Let a=20;  A=’Mahesh’ //Valid |
| Execution | Browser doesn’t understand TypeScript. | Browser understands JavaScript, html, css. |
|  | Needs to be converted to JavaScript before it reaches to browser. | Browser will directly understand the code. |



This is the official bug reported over angular website also, mentioning Angular is dynamically typed, meaning that the datatype will change according to the value.

And that will create big problem as like below

**Ex:**

let a=33;

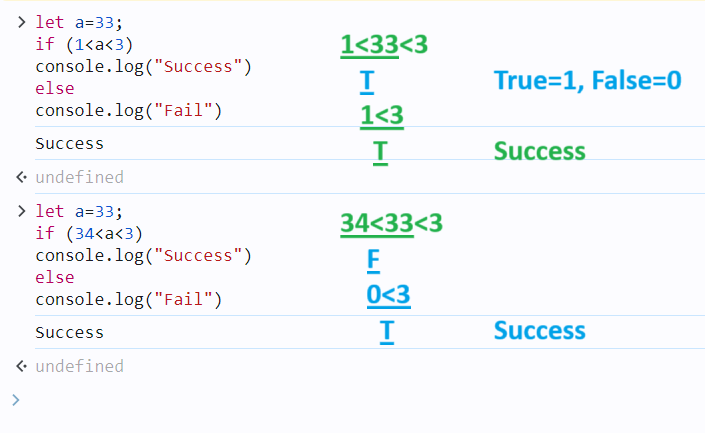
if (1<a<3)

console.log("Success")

else

console.log("Fail")

Success



**Basics of TypeScript:**

TypeScript is an open-source programming language developed by Microsoft that builds on JavaScript by adding static types. TypeScript is a superset of JavaScript, meaning any valid JavaScript code is also valid TypeScript code. However, TypeScript adds additional syntax to allow developers to specify types (such as string, number, boolean, etc.) for variables, function parameters, and return values.

To install TypeScript, you typically use npm (Node Package Manager), which comes with Node.js. (npm cannot be separately installed it comes with Node.js)

**1. Install Node.js**

* **Node.js**: If you haven’t installed Node.js yet, download and install it from [nodejs.org](https://nodejs.org). Installing Node.js will also install **npm (Node Package Manager).**
* **To update** npm use: **npm install -g npm**

**2. Install TypeScript Globally**

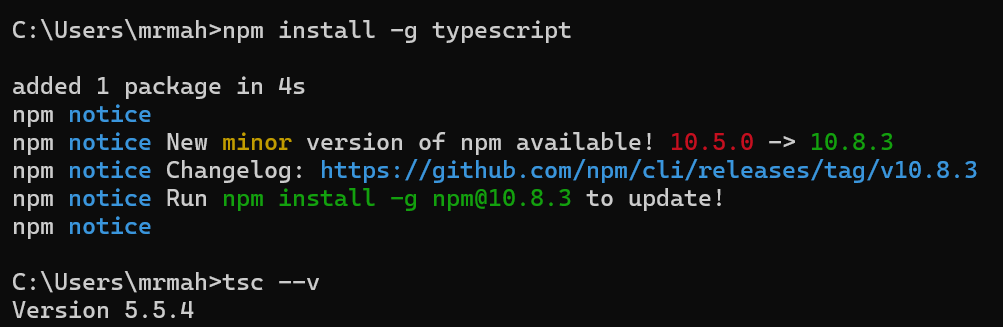
* Open your terminal (Command Prompt, PowerShell, or a terminal in your code editor) and run the following command to install TypeScript globally:

**npm install -g typescript@latest**

This installs the TypeScript compiler globally on system, so you can use it from any directory.

**3. Verify the Installation**

* To verify that TypeScript is installed correctly, run the following command in your terminal: **tsc –version**
* To Update Typescript use : **npm install -g typescript**



**4. Compile TypeScript Code**

* Once installed, you can compile a TypeScript file (.ts) into JavaScript by using the tsc command. For example: **tsc example.ts**

**Optional: Initialize a TypeScript Project**

* If you are starting a new TypeScript project, you can initialize it with a tsconfig.json file by running: **tsc --init**

This file helps configure TypeScript options such as the target ECMAScript version, module resolution, and more. Once installed and configured, ready to start developing in TypeScript!

**Note:**

Npm : Node Package Manager

-g : To install it globally in system. If not mentioned it will install in current folder.

@latest/version no: Will install particular version or latest version.

* In Visual Studio Code check the Auto Save option under file menu.
* Browser only understand HTML, CSS, JavaScript, it wont understand TypeScript.
* Install **Live Server** extension in VSCode editor.
* JavaScript Code must be added at the end of body not at the end of head and CSS at end of head section.

First.html:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

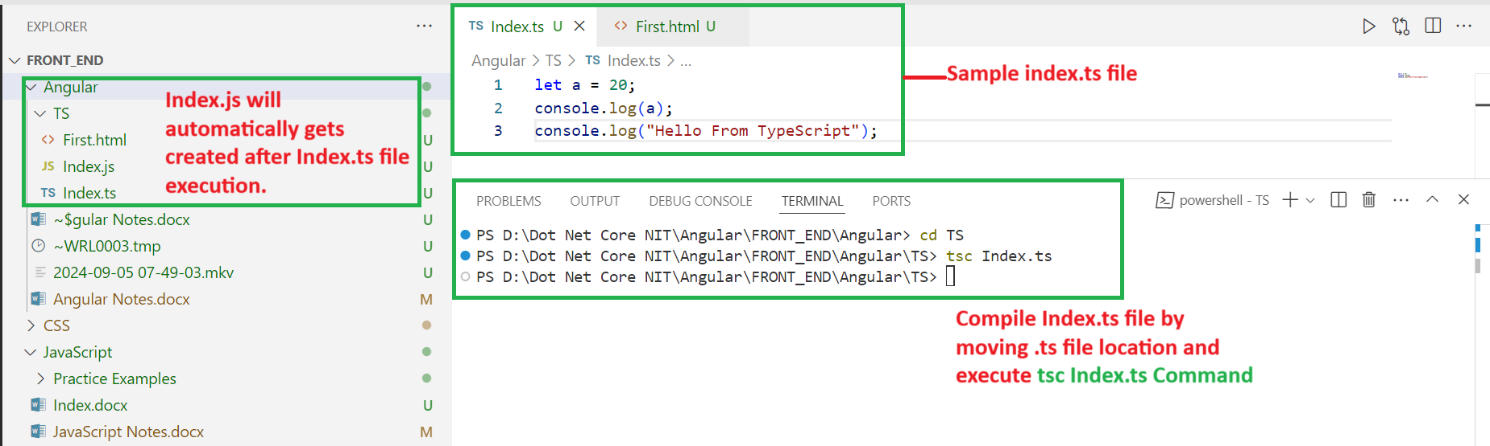
</head>

<body>

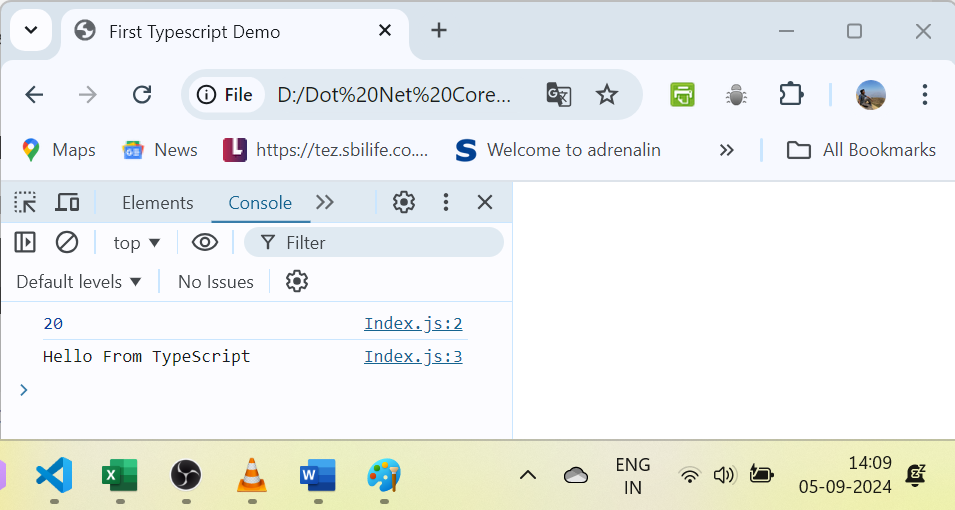
    <script src="Index.js"></script>

</body>

</html>



Output:



The watch mode in TypeScript allows you to automatically recompile your TypeScript code whenever changes are made to your .ts files. This is useful during development, as it eliminates the need to manually run the tsc command every time you make a change.

**How to Use the watch Command**

You can enable watch mode by using the --watch (or -w) flag with the tsc command.

**1. Basic Usage**

To watch a single TypeScript file for changes and automatically recompile it, run:

**tsc --watch <filename>.ts**

For example, if you have a file named app.ts, you would run:

**tsc --watch app.ts**

This will watch the file for any changes and recompile it automatically when changes are detected.

**2. Watch an Entire Project**

If you have multiple TypeScript files in your project and you want to watch all of them, it's more efficient to use the tsconfig.json configuration file. You can enable watch mode for the entire project by running:

**tsc --watch**

This will watch all the files specified in your tsconfig.json and automatically recompile them whenever changes are made.

06-09-2024 Friday

**Type inference:**

Type inference in TypeScript is a feature that allows the TypeScript compiler to automatically determine the type of a variable, function return value, or expression based on the value assigned or the context in which it is used. This means you don't always have to explicitly specify the type, making the code more concise while still maintaining type safety.

**How Type Inference Works**

TypeScript infers types based on the value assigned to a variable or the return value of a function. This inferred type is then used throughout the code, providing type safety without requiring explicit type annotations.

**Example: Variable Type Inference**

let message = "Hello, TypeScript!"; // TypeScript infers 'message' as string

let count = 42; // TypeScript infers 'count' as number

let isActive = true; // TypeScript infers 'isActive' as boolean

In this example:

* message is inferred to be of type string because it is initialized with a string.
* count is inferred to be of type number because it is initialized with a number.
* isActive is inferred to be of type boolean because it is initialized with a boolean.

**Example: Function Return Type Inference**

TypeScript can also infer the return type of a function based on the values returned within the function.

function multiply(a: number, b: number) {

return a + b; // TypeScript infers the return type as number

}

let result = multiply(5, 10); // TypeScript infers 'result' as number

Here, TypeScript infers that the return type of the multiply function is number because the result of adding two numbers is a number.

**Example: Inference with Arrays**

TypeScript infers the type of array elements based on the initial values provided.

let numbers = [1, 2, 3, 4, 5]; // TypeScript infers numbers as number[]

let fruits = ["apple", "banana", "cherry"]; // TypeScript infers fruits as string[]

In this example:

* numbers is inferred to be of type number[] (an array of numbers).
* fruits is inferred to be of type string[] (an array of strings).

**Example: Contextual Typing**

TypeScript can also infer types based on the context, such as event handling in the DOM.

window.addEventListener("click", (event) => {

console.log(event.clientX); // TypeScript infers 'event' as MouseEvent

});

In this case, TypeScript infers that event is of type MouseEvent because it's being used in the context of a click event listener.

**Benefits of Type Inference**

* **Conciseness**: Reduces the need to explicitly specify types, leading to cleaner and more readable code.
* **Type Safety**: Even without explicit types, TypeScript still checks types at compile-time, catching potential errors.
* **Better Tooling**: With inferred types, editors and IDEs can provide better autocompletion, refactoring, and error-checking features.

**When to Use Explicit Types**

While type inference is powerful, there are situations where explicitly declaring types might be better:

* **Clarity**: Explicit types can make your code more understandable, especially in complex situations.
* **Public APIs**: When defining public interfaces or APIs, explicit types can help other developers understand how to use your code correctly.
* **Complex Types**: In cases where the inferred type is complex or where the default inference might be too general (e.g., any), explicit typing is useful.

**Summary**

Type inference in TypeScript allows the compiler to automatically determine types based on the assigned values or context, providing a balance between type safety and code simplicity. While explicit types are sometimes necessary, type inference helps reduce the need for boilerplate type annotations in many cases.

**Type Annotation:**

Type annotation in TypeScript is the practice of explicitly specifying the type of a variable, function parameter, function return type, or expression. This provides more clarity, improves code readability, and ensures type safety by explicitly defining what types are expected.

**How to Use Type Annotations**

Type annotations are added by placing a colon : followed by the type after the variable name, function parameter, or function return type.

**Example: Variable Type Annotation**

let name: string = "Alice";

let age: number = 25;

let isStudent: boolean = true;

**In this example:**

* name is explicitly annotated as string.
* age is explicitly annotated as number.
* isStudent is explicitly annotated as boolean.

**Example: Function Parameter and Return Type Annotations**

You can also use type annotations for function parameters and return types:

function greet(name: string): string {

return "Hello, " + name;

}

let greeting: string = greet("Alice");

console.log(greeting); // Outputs: Hello, Alice

**In this example:**

* The name parameter is annotated as string.
* The function greet is annotated to return a string.

**Example: Object Type Annotation**

You can annotate the types of properties in an object:

let user: { name: string; age: number; isAdmin: boolean } = {

name: "Alice",

age: 25,

isAdmin: true

};

**In this example:**

* The user object is annotated with an object type that specifies name as string, age as number, and isAdmin as boolean.

**Example: Array Type Annotation**

Type annotations can also be used to specify the type of elements in an array:

let numbers: number[] = [1, 2, 3, 4, 5];

let fruits: string[] = ["apple", "banana", "cherry"];

**In this example:**

* numbers is annotated as an array of numbers (number[]).
* fruits is annotated as an array of strings (string[]).

**Example: Function Type Annotation**

You can annotate both the parameter types and return type for a function type:

let add: (a: number, b: number) => number = function(a: number, b: number): number {

return a + b;

};

console.log(add(5, 10)); // Outputs: 15

**In this example:**

* add is annotated as a function type that takes two number parameters and returns a number.

**Example: Union Type Annotation**

TypeScript allows you to specify that a variable can be of multiple types using union types:

let value: string | number;

value = "Hello"; // OK

value = 42; // OK

// value = true; // Error: Type 'boolean' is not assignable to type 'string | number'.

**In this example:**

* value is annotated to be either a string or a number. It can hold values of either type, but nothing else.

**Example: Optional Parameter Annotation**

You can annotate parameters as optional by using a question mark (?):

function greet(name: string, age?: number): string {

if (age) {

return `Hello, ${name}. You are ${age} years old.`;

} else {

return `Hello, ${name}.`;

}

}

console.log(greet("Alice")); // Outputs: Hello, Alice.

console.log(greet("Bob", 30)); // Outputs: Hello, Bob. You are 30 years old.

**In this example:**

* The age parameter is optional (age?: number), so it may or may not be provided.

**Summary**

Type annotations in TypeScript allow you to explicitly specify the types of variables, function parameters, return values, objects, arrays, and more. This enhances type safety, improves code clarity, and makes it easier for others (and tools like IDEs) to understand and work with your code.

**Variable declaration and its meaning:**

In TypeScript, variable declaration can be done using three main keywords: let, const, and var. Each has its own scope, mutability, and behavior. Here’s an overview of how they work:

**1. let Declaration:**

* **Scope**: Block-scoped (local to the block {} where it's declared).
* **Reassignment**: Can be reassigned.
* **Hoisting**: Hoisted but not initialized, meaning the variable exists in memory but cannot be used before it's declared.

let x: number = 5;

x = 10; // Reassignment is allowed

**2. const Declaration:**

* **Scope**: Block-scoped (local to the block {}).
* **Reassignment**: Cannot be reassigned (immutable reference). However, if it's an object or array, the contents can be mutated.
* **Hoisting**: Hoisted but not initialized, similar to let.

const y: string = "Hello";

// y = "World"; // Error: cannot reassign a const variable

const obj = { name: "Alice" };

obj.name = "Bob"; // Object contents can be mutated

**3. var Declaration:**

* **Scope**: Function-scoped (if declared within a function) or global-scoped (if declared outside any function). It ignores block-level scope.
* **Reassignment**: Can be reassigned.
* **Hoisting**: Hoisted and initialized, meaning the variable can be used before it's declared, but will return undefined if accessed before the declaration.

var z: boolean = true;

z = false; // Reassignment is allowed

function testVar() {

var inside = "inside";

console.log(inside); // "inside" is accessible here

}

**Q. Differences between let, const, and var:**

* **Scope**: let and const are block-scoped, while var is function-scoped.
* **Hoisting**: var is hoisted and initialized with undefined, while let and const are hoisted but not initialized (they exist in a "temporal dead zone" until their declaration is reached).
* **Mutability**: const creates immutable bindings (though object properties can be mutated), while let and var allow reassignment.

**Type Annotations in TypeScript:**

TypeScript allows you to add explicit type annotations when declaring variables. For example:

let count: number = 10; // Variable 'count' is of type 'number'

const name: string = "John"; // Constant 'name' is of type 'string'

These annotations help TypeScript's type-checking system to ensure that variables are used correctly throughout the code.

|  |  |  |
| --- | --- | --- |
| **var** | **let** | **const** |
| The scope of a [*var*](https://www.geeksforgeeks.org/javascript-var/)variable is functional or global scope. | The scope of a[*let*](https://www.geeksforgeeks.org/javascript-let/) variable is block scope. | The scope of a *[const](https://www.geeksforgeeks.org/javascript-const/" \t "_blank)* variable is block scope. |
| It can be updated and re-declared in the same scope. | It can be updated but cannot be re-declared in the same scope. | It can neither be updated or re-declared in any scope. |
| It can be declared without initialization. | It can be declared without initialization. | It cannot be declared without initialization. |
| It can be accessed without initialization as its default value is “undefined”. | It cannot be accessed without initialization otherwise it will give ‘referenceError’. | It cannot be accessed without initialization, as it cannot be declared without initialization. |
| These variables are hoisted. | These variables are hoisted but stay in the temporal dead zone untill the initialization. | These variables are hoisted but stays in the temporal dead zone until the initialization. |

**When to Use let and const**

**var** can be tricky because its scope is either global or within a function, which can lead to bugs. To avoid these issues:

* Use **let**when you know a variable’s value might change later in your code.
* Use **const**for variables that should never change once you set them.

Using let and const makes your code easier to understand and helps prevent errors caused by unexpected variable changes.

**Arrays in TypeScript:**

In TypeScript, arrays can be categorized based on the types of elements they hold and how they are defined. Here’s a breakdown of the different types of arrays in TypeScript:

**1. Homogeneous Array**

A homogeneous array is an array where all elements are of the same type.

**Example:**

let numbers: number[] = [1, 2, 3, 4, 5];

let strings: string[] = ["apple", "banana", "cherry"];

In this case, both numbers and strings arrays contain elements of a single type (number and string, respectively).

**Alternate Syntax:**

let numbers: Array<number> = [1, 2, 3, 4, 5];

Here, the Array<number> syntax is another way of declaring an array of numbers.

**2. Heterogeneous Array**

A heterogeneous array allows different types of elements within the same array. This can be achieved using **union types** or **tuples**.

**Using Union Types:**

let mixedArray: (number | string | boolean)[] = [1, "hello", true, 2, "world"];

This array can hold number, string, and boolean types.

**Using Tuples:**

let tupleArray: [number, string, boolean] = [1, "Alice", true];

Tuples enforce the exact type and order of elements.

**3. Array of Arrays (Multidimensional Array)**

In TypeScript, you can create an array of arrays, which is often referred to as a multidimensional array. This type is commonly used for matrices or grid-like data.

**Example (2D array):**

let matrix: number[][] = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

];

This represents a 2D array (array of arrays), where each inner array is a row in the matrix.

**4. Array of Objects**

You can also have arrays where each element is an object. This is common when dealing with data like JSON.

**Example:**

interface Person {

name: string;

age: number;

}

let people: Person[] = [

{ name: "Alice", age: 25 },

{ name: "Bob", age: 30 }

];

In this example, people is an array of Person objects, where each object has properties name and age.

**5. Readonly Array**

A **readonly** array is an immutable array whose elements cannot be modified once assigned. You can declare this using readonly keyword.

**Example:**

let readonlyNumbers: readonly number[] = [1, 2, 3, 4];

// readonlyNumbers[0] = 10; // Error: Cannot assign to '0' because it is a read-only property.

In this case, you cannot modify the elements of the readonlyNumbers array.

**6. Generic Arrays**

TypeScript arrays can also be created using **generics**, which makes the array flexible and reusable across different data types.

**Example:**

function getArray<T>(items: T[]): T[] {

return new Array().concat(items);

}

let numberArray = getArray<number>([1, 2, 3, 4]);

let stringArray = getArray<string>(["apple", "banana", "cherry"]);

In this example, the getArray function is a generic function that can work with any type of array, depending on what type T is provided.

**7. Array with Optional Elements (Sparse Array)**

A sparse array is an array in which some of the elements are intentionally left undefined.

**Example:**

let sparseArray: number[] = [1, , 3, 4];

console.log(sparseArray); // Output: [1, empty, 3, 4]

In this array, the second element is intentionally left empty.

**Summary of Types:**

* **Homogeneous Array**: All elements are of the same type.
* **Heterogeneous Array**: Contains elements of different types (using union types or tuples).
* **Array of Arrays**: Multidimensional arrays or nested arrays.
* **Array of Objects**: Arrays containing objects, often used with interfaces.
* **Readonly Array**: Immutable arrays that cannot be modified.
* **Generic Arrays**: Flexible arrays defined with generics to work with any type.
* **Sparse Arrays**: Arrays with missing or undefined elements.

Each of these array types has different use cases, depending on the structure and requirements of the data you are working with.

**Continued Angular:**

**Note:** Angular use command line interface to generate / delete/update any of its components, directives etc..,

This technique of using command prompt for everything is called scaffolding.

**IQ :**

**What is Scaffolding in Angular?**

**Scaffolding** refers to the automatic generation of code or project structure using the Angular CLI (Command Line Interface). It allows developers to quickly create components, services, modules, and other Angular artifacts with predefined templates and structures. This helps maintain consistency and speeds up development by reducing the manual effort of writing boilerplate code.

**Key Scaffolding Commands in Angular:**

**How to Scaffold in Angular**

To scaffold an Angular project or component, you use the ng generate command or its shortcut ng g. Here are some commonly used commands and examples:

**1. Scaffold an Angular Project**

To create a new Angular project, you run the following command: bash

**ng new my-angular-app**

This command scaffolds the entire project structure with a basic setup, including the following:

* src/ folder containing the main application code
* app/ folder with an initial component
* Configuration files such as angular.json, package.json, and tsconfig.json

**2. Scaffold Components**

Components are fundamental building blocks in Angular. You can scaffold new components like this: bash

**ng generate component my-component**

or using the shorthand: bash

**ng g c my-component**

This command generates:

* A TypeScript file (my-component.component.ts)
* An HTML template (my-component.component.html)
* A CSS or SCSS stylesheet (my-component.component.css)
* A test file (my-component.component.spec.ts)

**Example:** Running ng g c header will generate the following: bash

src/app/header/header.component.ts

src/app/header/header.component.html

src/app/header/header.component.css

src/app/header/header.component.spec.ts

**3. Scaffold Services**

Angular services handle business logic and data management. To scaffold a service, use: bash

**ng generate service my-service**

or the shorthand: bash

**ng g s my-service**

This creates a TypeScript file for the service with basic boilerplate code: bash

src/app/my-service.service.ts

**4. Scaffold Modules**

Modules organize related components, services, and other Angular constructs. To scaffold a module: bash

**ng generate module my-module**

or the shorthand: bash

**ng g m my-module**

This command generates a new module file: bash

src/app/my-module/my-module.module.ts

**5. Scaffold Other Angular Artifacts**

Angular CLI can scaffold other constructs such as **directives**, **pipes**, **guards**, and **classes**.

* **Directives**: bash

ng g directive my-directive

* **Pipes**: bash

ng g pipe my-pipe

* **Guards**: bash

ng g guard my-guard

**Example Workflow:**

Let’s say you want to create a new Angular project for an online store. You can scaffold it as follows:

1. **Create a new Angular project**: bash

ng new online-store

1. **Generate a product component**: bash

cd online-store

ng g c product

1. **Generate a service to fetch product data**: bash

ng g s product-service

1. **Generate a module to group related components**: bash

ng g m products

This scaffolding process saves time by creating all the required files with boilerplate code and directory structure for you, allowing you to focus on writing the business logic.

**Benefits of Scaffolding in Angular:**

1. **Speeds up Development**: Reduces manual work and automates the creation of files and structure.
2. **Consistency**: Ensures that all generated components, services, and other parts follow Angular best practices and conventions.
3. **Reduces Errors**: By using predefined templates, scaffolding minimizes the risk of errors in boilerplate code.
4. **Focus on Business Logic**: Developers can focus on core functionality instead of writing repetitive code.

Scaffolding in Angular enhances productivity by quickly generating the necessary components and services for your project.

**Angular 17**

[*https://www.youtube.com/watch?v=uJIbc2YE58E*](https://www.youtube.com/watch?v=uJIbc2YE58E) *ARC Tutorial*

Angular Folder Structure and Files

1. Parent folder will be the main project folder
2. .angular : Ignore this folder (internally used for caching, memory management etc)
3. .vscode : Ignore this folder (internally used for caching, memory management etc)
4. Node\_modules :
   1. Packages will be installed in this folder whenever you install/add new packages using npm install or ng add.
   2. You don’t have to go through these folder or files.
   3. Unless and until you don’t make any changes to core libraries or modules.
5. .editorconfig: Make your custom editor changes in this file.
6. .gitignore: we can add fodders/files that we want to ignore while committing to git.
7. angulare.json :
   1. this is the file configured styles, js for deploying in a pipeline.
   2. It is having all the configuration details of angular application like what is the version, where is the schema located, what is the prefix that you want to add to angular project, root:”” meaning it’s a parent directory, schematics decides what kind of style we want to design that could be css, scss etc, we can add different configuration specific to the different projects separately.
   3. Under the architect we can see the different settings while build , serve , production, development with default defaultconfiguration.
8. package.json :
   1. In this we can het various entries of new packages installed along with there versions.
   2. When we run npm install inside project : the module listed will be installed
   3. here we add our scripts that can be used to run application with single command and multiple settings. Like “arc-build”:”ng serve && json-server –watch db.json”
9. package-lock.json :
   1. same details of package.json + dev dependencies broken down in details.
   2. Don’t touch this manually
10. Tsconfig.app.json :
    1. Tells you the typescript configuration for your project
    2. Don’t touch this manually – for dev purpose
11. Tsconfig.spec.json: typescript test specific configuration
12. ReadMe.md : starting file : documentation of your project
13. SRC :
    1. Source code of project
    2. App :
       1. This is actual code of project/ application
       2. **Every component in Angular has 4 files**
          1. .html : Template / HTML code
          2. .scss/.css/.less : Styles
          3. .spec.ts : used for unit testing
          4. .ts : Component class / logical piece of component.
       3. App.component.ts : It has selector attribute with app-root value and which will ring the bell app-root in index.html i.e. the code present inside the app will be injected dynamically as a SPA inside the index.html.
       4. App.component.spec.ts : There is no end to end Unit framework for unit testing it is shifted to Jasmine
          1. Jasmine is for writing unit tests
          2. Karma is to Test Runner



* 1. Assets : use this folder to serve the assets which are public and may contains images, videos, js files that are kept as public.
  2. Favicon : favicon for application specific.
  3. Index.html : Angular is a SPA(Single Page Application), there is only one html file index.html. when we develop/build the app : the index.html contains only **<app-root>** First component to be initialized.
  4. main.ts : Entry point to project. This is the first file to be called which decides which component to be rendered next ex: 1. Main.ts – picks appComponent and go to index.html and into <app-root></app-root>
  5. styles.scss : Global styling for your project. The extension varies from project to project like .css/.less/.scss etc. depends on your initial setup for specific project.

1. a

Creating First Application in Angular:

1. Goto specific folder where you want to create Angular application and write

ng new <app\_Name> --standalone=false

Note:

ng : Next Generation

new : used for new element

app\_Name : Application Name

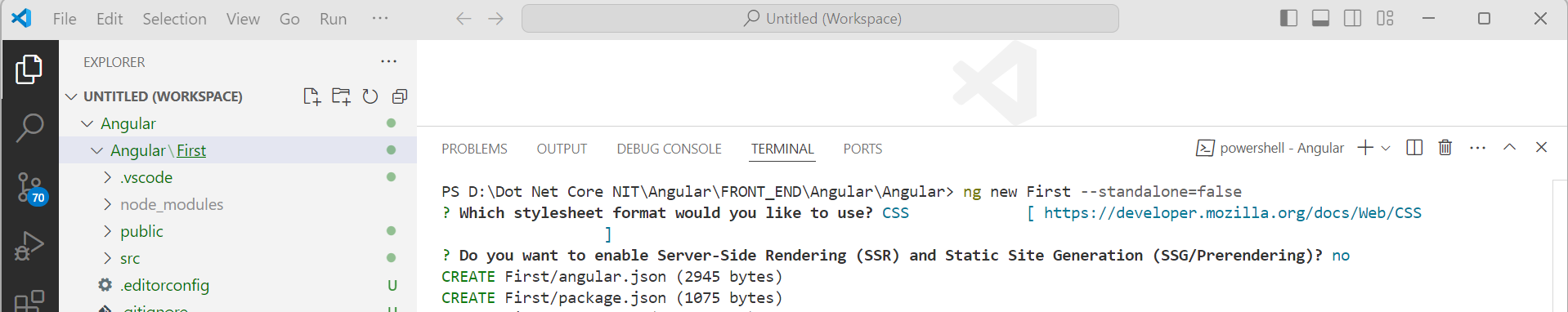
**--standalone=false** : use --standalone=false when you are working with the traditional Angular structure that relies on modules and when you want the component to be declared inside an NgModule, as was the standard in Angular versions prior to v14.

**--standalone=false** means that the generated component or directive will **not be standalone**. It will be part of an Angular module, and you will need to declare it within an NgModule.

Ex:

ng new First –standalone=false

First : Name of application



1. To run application use

ng serve --o

serve : To run application

--o will open the application in browser with default port number

1. Delete default code available in app.component.html file, and write your own without writing boiler plate code as the boiler plate code is already in index.html and only app.component.html code will be injected.

<div class="card">

  <img src="https://pbs.twimg.com/profile\_images/787106179482869760/CwwG2e2M\_400x400.jpg" alt="Mahesh PRofile photo">

  <div><strong>Name : </strong> Mahesh Baradkar</div>

  <div><strong>Email : </strong>agilemahesh33&#64;gmail.com</div>

  <div class="socialLinks">

    <a href="https://www.facebook.com">facebook</a>

    <a href="https://www.LinkedIn.com">LinkedIn</a>

    <a href="https://www.GitHub.com">GitHub</a>

  </div>

</div>

1. Add css styles into app.component.css.

.card{

    padding: 10px;

    display: flex;

    flex-direction: column;

    justify-content: center;

    align-items: center;

    height: max-content;

    background-color: white;

    border: 5px solid #1c6125;

    border-radius: 33px;

}

img{

    height: 100px;

    width: 100px;

    border-radius: 50px;

    box-shadow: rgba(29, 236, 10, 0.17) 0px -23px 25px 0px inset, rgba(0, 0, 0, 0.15) 0px -36px 30px 0px inset, rgba(0, 0, 0, 0.1) 0px -79px 40px 0px inset, rgba(0, 0, 0, 0.06) 0px 2px 1px, rgba(0, 0, 0, 0.09) 0px 4px 2px, rgba(0, 0, 0, 0.09) 0px 8px 4px, rgba(0, 0, 0, 0.09) 0px 16px 8px, rgba(0, 0, 0, 0.09) 0px 32px 16px;

}

.socialLinks{

    display: flex;

    justify-content: space-between;

    gap: 20px;

}

1. Add common Global style Styles.css

/\* You can add global styles to this file, and also import other style files \*/

body{

    display: flex;

    justify-content: center;

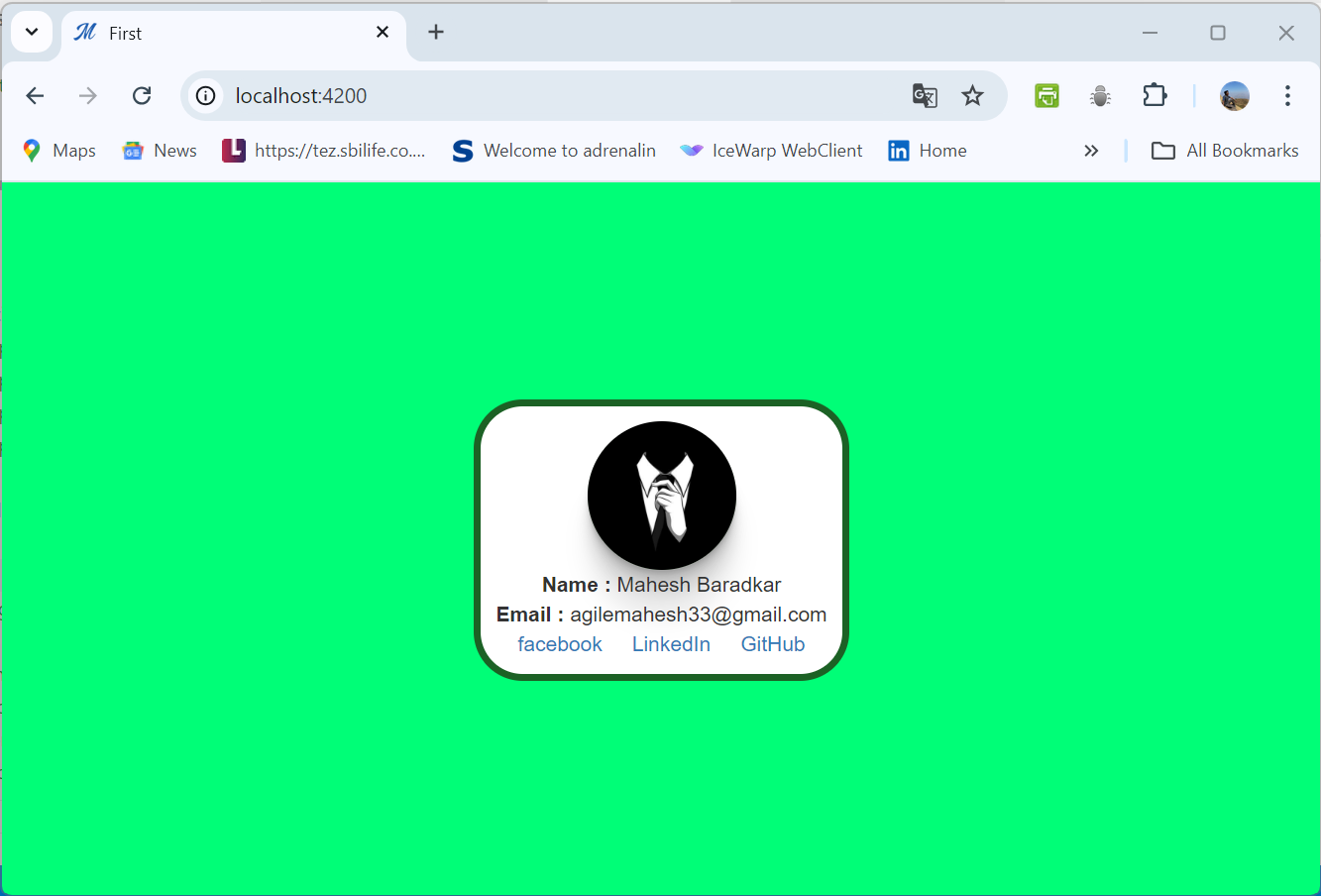
    align-items: center;

    height: 100vh;

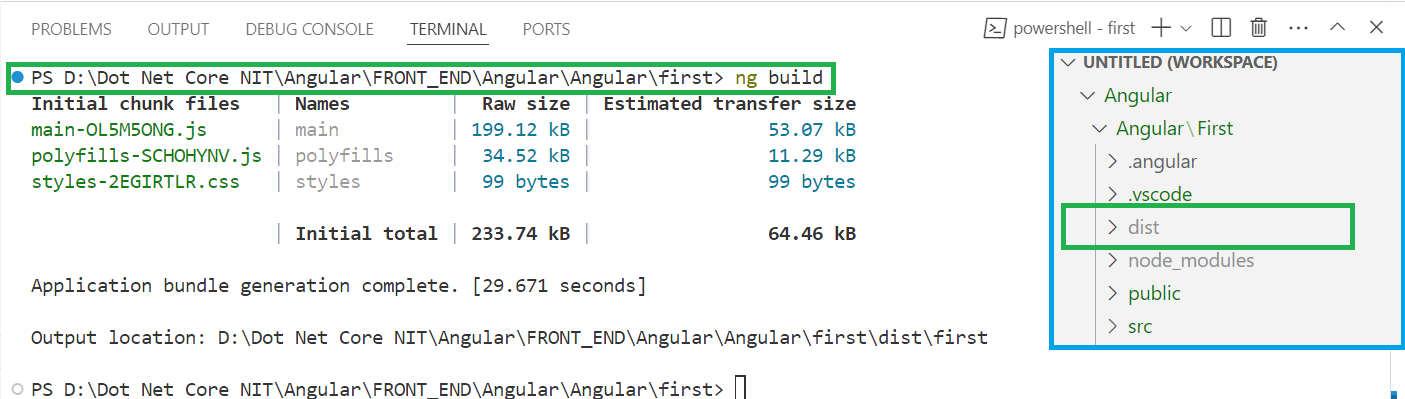
    background-color:rgb(0, 254, 119)

}

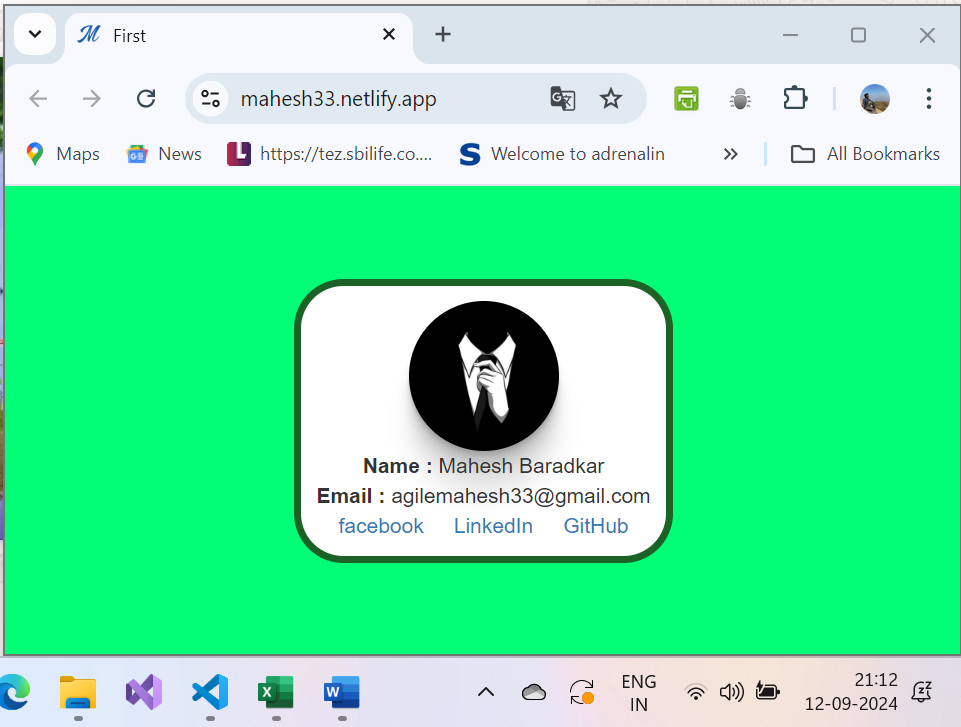
1. Output of the above will look like



1. To push the above code to live server follow below steps:
   1. Stop current execution by pressing ctrl+c in VS Code.
   2. To build the project execute “ng build” command so that the project will be ready to move on live server.
   3. “ng build” command also creates dist folder which can then be directly copied to the live server.



* 1. Create or login to <https://app.netlify.com/> for temporary deployment.
  2. Upload dist folder that is created in step c. which will generate new url for the browser.
  3. We can change URL but that should be unique.



1. sdf