1. **What is Angular Framework & TypeScript ?**

Angular is a TypeScript-based open-source front-end platform that makes it easy to build applications within web/mobile/desktop. TypeScript is a superset of JavaScript created by Microsoft that adds optional types, classes, async/await, and many other features, and compiles to plain JavaScript. Angular built entirely in TypeScript and used as a primary language.

1. **What is Angular Application Workflow?**

***angular.json => main.ts => app.module.ts => app.component.ts => app-root => index.html;***

**angular.json** file will contain all the configurations of the app. While building the app, the builder looks at this file to find the entry point of the application. Inside the build section, the **main** property of the **options** object defines the entry point of the application (default case is main.ts).

The **main.ts** file creates a browser environment for the application to run, and, along with this, it also calls a function called **bootstrapModule**, which bootstraps the application. AppModule is getting bootstrapped by **platformBrowserDynamic().bootstrapModule(AppModule)** which is existing in **main.ts**.

The **AppModule** is declared in the **app.module.ts** file. This module contains declarations of all the components. This **AppModule** bootstraps the **AppComponent** which is defined in **app.component.ts** file. This file interacts with the webpage and serves data to it. Each component is declared with three properties: 1). **Selector** - used for accessing the component; 2). **Template/TemplateURL** - contains HTML of the component; 3). **StylesURL** - contains component-specific stylesheets

After this, Angular calls the **index.html** file. This file consequently calls the root component that is **app-root**. The root component is defined in **app.component.ts**. The HTML template of the root component is displayed inside the **<app-root>** tags. This is how every angular application works.

1. **What are the key components of Angular?**

**Component**: Components are the basic building blocks, which control a part of the UI for an application. A component is defined using the @Component decorator. Every component consists of three parts, a selector: to represent in the UI, a templateUrl: to load the view for the component and a styleUrls: to apply the styles for the component. ng g c test / ng generate component test will generate the test component.

**Modules**: Module is a place where we can group *components, directives, services, and pipes.* Module decides whether the components, directives, etc can be used by other modules, by exporting or hiding these elements. Every module is defined with a @NgModule decorator. By default, modules are of two types: *Root Module & Feature Module*. Every application can have only one root module whereas, it can have one or more feature modules. A root module imports *BrowserModule*, whereas a *feature* module imports *CommonModule*. *ng g m test-module / ng generate module test-module;*

**Templates**: A template is a HTML view where you can display data by binding controls to properties of an Angular component. We can define it inline using the template property, or we can define the template in a separate HTML file and link to it in the component metadata using the @Component decorator's templateUrl property.

**Services**: The main objective of a service is to share data, functions with different components of an Angular application. A service is defined using a @Injectable decorator. A function defined inside a service can be invoked from any component or directive. ng g s test-service / ng generate service test-service;

**Metadata**: Metadata Provides additional information about the component to the Angular. Angular uses this information to process the class. We use the @Component decorator to provide the Metadata to the Component.

1. **Data-Binding & String Interpolation**

**Data-binding** is a feature in angular, which provides a way to communicate between the component(Model) and its view(HTML template). Data-binding can be done in two ways, one-way binding and two-way binding. In Angular, data from the component can be inserted inside the HTML template. In one-way binding, any changes in the component will directly reflect inside the HTML template but, vice-versa is not possible. Whereas it is possible in two-way binding.

**String interpolation** is a one-way data-binding technique that outputs the data from TypeScript code to HTML view and it uses the double curly braces {{ }} to display data from the component. Angular automatically runs the expression written inside the curly braces.

Using **property binding**, we can bind the DOM properties of an HTML element to a component's property. Property binding uses the square brackets [ ] syntax.

1. **Directive and its types**

Directives are classes that add additional behaviour to elements in Angular applications. They execute whenever the Angular compiler finds them in the DOM. Directive is a class in Angular that is declared with **a @Directive** decorator.

**Component directives**: Instead of @Directive decorator we use @Component decorator to declare these directives. These directives have a view, a stylesheet and a selector property.

**Structural directives:** These directives are generally used to manipulate DOM elements. Every structural directive has a ‘ \* ’ sign before them. We can apply these directives to any DOM element. Ex: \*ngIf & \*ngFor;

**Attribute Directives**: These directives are used to change the look and behaviour of a DOM element. Let’s understand attribute directives by creating one: **ng g directive blueBackground;**

import { Directive, ElementRef } from '@angular/core';

@Directive({ selector: '[appBlueBackground]' })

export class BlueBackgroundDirective {

constructor(el:ElementRef) { el.nativeElement.style.backgroundColor = "blue"; } }

1. **What are the differences between Component and Directive?**

| **Component (**use @Component) | **Directive (**use @Directive) |
| --- | --- |
| used to create UI widgets | used to add behaviour to DOM elements |
| used to break up the application into smaller components | used for design re-usable components |
| Only one component can be present per DOM | Many directives can be used per DOM |
| @View decorator or templateurl/template are mandatory | Directive doesn't use View |

1. **What is lifecycle hooks available?**

Angular calls lifecycle hook methods on directives and components as it creates, changes, and destroys them.

* 1. **Constructor**: While creating a component/directive by calling new keyword on the class.
  2. **ngOnChanges**: Responds when Angular sets/resets data-bound input properties.
  3. **ngOnInit**: While initialization of the directive/component after Angular first displays the data-bound properties happens.
  4. **ngDoCheck**: Detect & act upon changes that Angular can't or won't detect on its own.
  5. **ngAfterContentInit**: Response after Angular projects external content into the component's view.
  6. **ngAfterContentChecked**: Response after Angular checks the content projected into the component.
  7. **ngAfterViewInit**: Response after Angular initializes the component's views and child views.
  8. **ngAfterViewChecked**: Response after Angular checks the component's views and child views.
  9. **ngOnDestroy**: Clean-up phase just before Angular destroys the directive/component.

1. **What is metadata?**

Metadata is used to decorate a class so that it can configure the expected behaviour of the class. The metadata is represented by decorators. The purpose of these decorators is to accept a metadata object that provides relevant information about the component.

**Class decorators**, e.g. @Component and @NgModule;

**Method decorators** Used for methods inside classes, e.g. @HostListener

**Parameter decorators** Used for parameters inside class constructors, e.g. @Inject, Optional

**Property decorators** Used for properties inside classes, e.g. @Input and @Output;

1. **What is angular CLI?**

Angular CLI (**Command Line Interface**) is a command line interface to scaffold and build angular apps using nodejs style (commonJs) modules.

1. **What is the difference between constructor and ngOnInit?**

TypeScript classes has a default method called **constructor** which is normally used for the initialization purpose. Whereas **ngOnInit** method is specific to Angular, especially used to define Angular bindings. Even though constructor getting called first, it is preferred to move all your Angular bindings to ngOnInit method. In order to use ngOnInit, you need to implement OnInit.

1. **What is dependency injection in Angular?**

A class asks for dependencies from external sources rather than creating them itself. Angular comes with its own dependency injection framework for resolving dependencies (services or objects that a class needs to perform its function). So you can have your services depend on other services throughout your application.

1. **What is pipe & parameterized pipes?**

Pipes are simple functions designed to accept an input value, process, and return as an output, a transformed value in a more technical understanding. The parameterized pipe can be created by declaring the pipe name with a colon ( : ) and then the parameter value. If the pipe accepts multiple parameters, separate the values with colons.

1. **What is HttpClient and its benefits?**

Most of the Front-end applications communicate with backend services over HTTP protocol using either XMLHttpRequest interface or the fetch() API. Angular provides a simplified client HTTP API known as **HttpClient** which is based on top of XMLHttpRequest interface. This client is available from @angular/common/http package.

1. **How do you perform Error handling?**

If the request fails on the server or failed to reach the server due to network issues then HttpClient will return an error object instead of a successful response. In this case, you need to handle in the component by passing error object as a second call-back to subscribe() method.

1. **What is RxJS?**

RxJS (Reactive Extensions for JavaScript) is a library for composing asynchronous and callback-functions using Observables. Many APIs such as HttpClient produce and consume RxJS Observables.

1. **What are Observables-Subscribing & Promises ?**

An Observable instance begins publishing values only when someone subscribes to it. So you need to subscribe by calling the **subscribe()** method of the instance, passing an observer object to receive the notifications.

**Observables** and **Promises** help us work with asynchronous functionality in JavaScript. Promises deal with one asynchronous event at a time, while observables handle a sequence of asynchronous events over a period. The first difference is that the Promise is eager, whereas the Observable is lazy.

|  |  |
| --- | --- |
| **Observables** | **Promises** |
| Emit multiple values over a period. | Emit a single value at a time. |
| Are lazy: they’re not executed until we subscribe using the subscribe() method. | Are not lazy: execute immediately after creation. |
| These are cancellable using unsubscribe() | Are not cancellable. |
| Provide operators like forEach, filter, reduce, etc., | Don’t provide any operators |
| Deliver errors to the subscribers. | Push errors to the child promises. |

1. **What are Angular Router, Router Outlet, Router Link & Activatedroute ?**

**Angular** **Router** is a mechanism in which navigation happens from one view to the next. The **RouterOutlet** is a directive, and it acts as a placeholder spot in the template where the router should display the component’s output. The **RouterLink** is a directive on the anchor tags give the router control over those elements. **ActivatedRoute** contains the information about a route associated with a component loaded (URL & Its params).

1. **What is AOT & Why do we need compilation process?**

The Angular components and templates cannot be understood by the browser directly. Due to those Angular applications require a compilation process before they can run in a browser. For example, In AOT compilation, both Angular HTML and TypeScript code converted into efficient JavaScript code during the build phase before browser runs it. Faster Rendering, Fewer Ajax Requests, Quick detection of template errors are the advantages of AOT.

1. **What is @Input & @Output Decorators ?**

Both are used to pass the different types of data form parent to child component and child to parent component. @**Input** decorator is used to pass data (property binding) from parent to child component (should be annotated with @Input decorator). @**Output** decorator is used to pass the data from child to parent component. @Output binds a property of EventEmitter class.

1. **Version Changes:**

Ang 1: [Oct-10] In JavaScript, Supports MVC, doesn’t support mobile;

Ang 2: [Sep-16] Complete rewritten, Written in TypeScript, support’s mobile

Ang 3: Skipped

Ang 4: [Mar-17] Intro of HttpClient

Ang 5: [Nov-17] Replaced @angular/http with @angular/common/http library.

Ang 6: [May-18] Available RxJS

Ang 7: [Oct-18] drag and drop interfaces => @angular/cdk/drag-drop

Ang 8: [May-19] Intro of Ivy compiler as opt-in feature; JIT is default upto 8;

Ang 9: [Nov-19] Ivy as default compiler

Ang 10: [Jun-20] TypeScript 3.9,

Ang 11: [Nov-20] Typescript 4.0., Deprecating support of IE9&IE10 and IE Mobile, **Hot Module Replacement** (HMR) helps in replacing the modules without a browser refresh.

Ang 12: [May-21] TypeScript 4.2, New Dev Tools, Nullish Coalescing, Ivy Everywhere.

Ang 13: [Nov-21] Deprecated for IE11, TypeScript 4.4 Rxjs 7.4 is now as default for new apps.

1. **AngularJS vs Angular**

AngularJS uses MVC architecture But in Angular replaces controller with Components. AngularJS uses JavaScript whereas Angular uses TypeScript. About mobile support AngularJS does not support whereas Angular can. AngularJS will not support DI where Angular can.

1. **IVY Rendering**

Ivy Compiler is default in Angular 9 onwards which provides huge improvements in application with **better debugging, fast testing, Improved type checking, Improved build errors & Improved build times**. Ivy has very big focus on tree-shaking (compiler looks at code and figure out exactly which libraries are required in project and eliminates the unused code. Also improves application loading time due to **reducing the bundle size**.

1. **What is an ngModule ?**

NgModules are containers that reserve a block of code to an application. @NgModule takes a metadata object that generally describes the way to compile the template of a component and to generate an injector at runtime.

1. **Pipes, Inbuilt Filters, Versions,**
2. **Routing** – An Angular router is responsible for interpreting a browser URL as an instruction to navigate to a client-generated view. The router is bound to links on a page to tell Angular to navigate the application view when a user clicks on it.
3. **Dependency Injection (DI)** – Angular makes use of DI to provide required dependencies to new components. Typically, dependencies required by a component are services. A component’s constructor parameters tell Angular about the services that a component requires. So, a dependency injection offers a way to supply fully-formed dependencies required by a new instance of a class.