* **What is the latest version of angular ?**

**11.0.8**

### 0.1 What are Single Page Applications (SPA)?

Single-page applications are web applications that load once with new features just being mere additions to the user interface. It does not load new HTML pages to display the new page's content, instead generated dynamically. This is made possible through JavaScript's ability to manipulate the DOM elements on the existing page itself. A SPA approach is faster, thus providing a seamless user experience.

1. **Explain Components, Modules and Services in Angular**

ng new angularApp  
  
**Components**  
In Angular, components are the basic building blocks, which control a part of the UI for any application.  
A component is defined using the **@Component** decorator. Every component consists of three parts, the template which loads the view for the component, a stylesheet which defines the look and feel for the component, and a class that contains the business logic for the component.  
For creating a component, inside the command terminal, navigate to the directory of the application created, and run the following command:

ng generate component test

selector: 'app-test',

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

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

})

export lass TestComponent implements OnInit {

constructor() {}

ngOnInit() {

}

}

As we can see in the above image, our component is defined with **@Component** decorator.

**Modules**  
A 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 ModuleEvery 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**.  
  
In the application that we created before, one can see that the root module is defined inside **app.module.ts** and this is how it looks:

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

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

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

import { TestComponent } from './test/text.component';

@NgModule({

declarations: [

AppComponent,

TestComponent

],

imports: [

BrowserModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

We can see in the above image that the component we created earlier is already imported in the declarations array.  
  
To create a feature module, run the following command:

ng g m test-module

The module is created inside the src/app/test-module/test-module.module.ts file:

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

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

@NgModule({

declarations: [],

imports: [

CommonModule

]

})

export class TestModuleModule { }

As one can see, **CommonModule** is imported since this is a feature module.

**Services** Services are objects which get instantiated only once during the lifetime of an application. 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.  
  
To create a service, run the following command:

ng g s test-service

The service will be created inside src/app/test-service.service.ts:

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

@Injectable({

providedIn: 'root'

})

export class TestServiceService {

constructor() { }

}

Any method/function defined inside the TestServiceService class can be directly used inside any component by just importing the service.

1. **What are lifecycle hooks in Angular? Explain a few lifecycle hooks.**

Every component in Angular has a lifecycle, different phases it goes through from the time of creation to the time it's destroyed. Angular provides **hooks** to tap into these phases and trigger changes at specific phases in a lifecycle.

**ngOnChanges( )** This hook/method is called before **ngOnInit** and whenever one or more input properties of the component changes.  
This method/hook receives a SimpleChanges object which contains the previous and current values of the property.  
  
**ngOnInit( )** This hook gets called once, after the **ngOnChanges** hook.  
It initializes the component and sets the input properties of the component.  
  
**ngDoCheck( )** It gets called after **ngOnChanges** and **ngOnInit** and is used to detect and act on changes that cannot be detected by Angular.  
We can implement our change detection algorithm in this hook. **ngAfterContentInit( )** It gets called after the first **ngDoCheck** hook. This hook responds after the content gets projected inside the component.  
  
**ngAfterContentChecked( )** It gets called after **ngAfterContentInit** and every subsequent **ngDoCheck**. It responds after the projected content is checked.  
  
**ngAfterViewInit( )** It responds after a component's view, or a child component's view is initialized.  
  
**ngAfterViewChecked( )** It gets called after **ngAfterViewInit**, and it responds after the component's view, or the child component's view is checked.  
  
**ngOnDestroy( )** It gets called just before Angular destroys the component. This hook can be used to clean up the code and detach event handlers.  
  
Let’s understand how to use **ngOnInit** hook, since it’s the most oftenly used hook. If one has to process lot of data during component creation, it’s better to do it inside **ngOnInit** hook rather than the constructor

### 3. Explain string interpolation and property binding in Angular.

String interpolation and property binding are parts of **data-binding** in Angular.  
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 and property binding allow only one-way data binding.  
String interpolation uses the double curly braces **{{ }}** to display data from the component. Angular automatically runs the expression written inside the curly braces, for example, {{ 2 + 2 }} will be evaluated by Angular and the output 4, will be displayed inside the HTML template. 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. **How are observables different from promises?**

The first difference is that an Observable is **lazy** whereas a Promise is **eager**.

|  |  |
| --- | --- |
| Promise | Observable |
| Emits a single value | Emits multiple values over a period of time |
| Not Lazy | Lazy. An observable is not called until we subscribe to the observable |
| Cannot be cancelled | Can be cancelled by using the unsubscribe() method |
| What are directives in Angular? A directive is a class in Angular that is declared with a **@Directive** decorator. Every directive has its own behaviour and can be imported into various components of an application.  **When to use a directive?** Consider an application, where multiple components need to have similar functionalities. The norm thing to do is by adding this functionality individually to every component but, this task is tedious to perform. In such a situation, one can create a **directive** having the required functionality and then, import the directive to components which require this functionality.  **Types of directives** **Component directives** These form the main class in 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. Explain the concept of Dependency Injection? Dependency injection is an application design pattern which is implemented by Angular. It also forms one of the core concepts of Angular.  **So what is dependency injection in simple terms?** Let’s break it down, dependencies in angular are nothing but services which have a functionality. Functionality of a service, can be needed by various components and directives in an application. Angular provides a smooth mechanism by which we can inject these dependencies in our components and directives. So basically, we are just making dependencies which are injectable across all components of an application. 7. How can you initialize a select box with options on page load? You can initialize a select box with options on page load by using ng-init directive.   * <div ng-controller = " apps/dashboard/account " ng-switch * On = "! ! accounts" ng-init = " loadData ( ) ">  **8: Explain Lazy Loading in Angular ?** Answer  Lazy loading is one of the most useful concepts of Angular Routing and brings down the size of large files. This is done by lazily loading the files that are required occasionally.  Angular 8 comes up with support for **dynamic imports** in our router configuration. This means that we use the import statement for lazy loading the module and this will be understood by the IDEs, webpack, etc. 9. **What are the utility functions provided by RxJS?** **Mid**  [C:\Users\808034\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6DE41F81.tmp](https://www.fullstack.cafe/interview-questions/angular" \o "Angular Interview Questions)**[Angular](https://www.fullstack.cafe/interview-questions/angular" \o "Angular Interview Questions)**[120](https://www.fullstack.cafe/interview-questions/angular" \o "Angular Interview Questions)  Answer  The RxJS library also provides below utility functions for creating and working with observables. 1. Converting existing code for async operations into observables 2. Iterating through the values in a stream 3. Mapping values to different types 4. Filtering streams 5. Composing multiple streams 10 . **What is AOT?** **Mid**  [C:\Users\808034\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\23C8E37.tmp](https://www.fullstack.cafe/interview-questions/angular" \o "Angular Interview Questions)**[Angular](https://www.fullstack.cafe/interview-questions/angular" \o "Angular Interview Questions)**[120](https://www.fullstack.cafe/interview-questions/angular" \o "Angular Interview Questions)  Answer  The Angular Ahead-of-Time compiler pre-compiles application components and their templates during the build process. Apps compiled with AOT launch faster for several reasons.   * Application components execute immediately, without client-side compilation. * Templates are embedded as code within their components so there is no client-side request for template files. * You don't download the Angular compiler, which is pretty big on its own. * The compiler discards unused Angular directives that a tree-shaking tool can then exclude.  11. **Why should ngOnInit be used, if we already have a constructor?** **Mid**  [C:\Users\808034\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\4A370C5D.tmp](https://www.fullstack.cafe/interview-questions/angular" \o "Angular Interview Questions)**[Angular](https://www.fullstack.cafe/interview-questions/angular" \o "Angular Interview Questions)**[120](https://www.fullstack.cafe/interview-questions/angular" \o "Angular Interview Questions)  Answer   * The Constructor is a default method of the class that is executed when the class is instantiated and ensures proper initialization of fields in the class and its subclasses. * ngOnInit is a life cycle hook called by Angular2 to indicate that Angular is done creating the component.   Mostly we use ngOnInit for all the initialization/declaration and avoid stuff to work in the constructor. The constructor should only be used to initialize class members but shouldn't do actual "work". So you should use constructor() to setup Dependency Injection and not much else. ngOnInit() is better place to "start" - it's where/when components' bindings are resolved.  12. What are Pipes in Angular?  Pipes are simple functions designed to accept an input value, process, and return as an output, a transformed value in a more technical understanding. Angular supports several built-in pipes. However, you can also create custom pipes that cater to your needs.  Some key features include:   1. Pipes are defined using the pipe “|” symbol. 2. Pipes can be chained with other pipes. 3. Pipes can be provided with arguments by using the colon (:) sign. | Observable provides operators like map, forEach, filter, reduce, retry, retryWhen etc. |