Angular Interview Questions & Answers

CREDO SYSTEMZ
Simplifying IT

Table of Content

Introduction to Angular

Angular CLI

Components and Directives

Modules

Pipes

Routing and Navigation

Angular Forms

Dependency Injection

Observables and RxJS

Angular Services and HTTP

Unit Testing

1. What is Angular?

Angular is a JavaScript-based framework developed by Google. Angular allows developers to build applications for browsers, mobile, and desktop using web technologies like HTML, CSS, and JavaScript.

Angular2+ is different from Angular1.x and it is completely re-written from scratch using Typescript. At the time of writing this, book the latest version of Angular was 7 which was released on 19 Oct 2018.

2. Explain evolution history of Angular

The evolutionary history of Angular is given below –
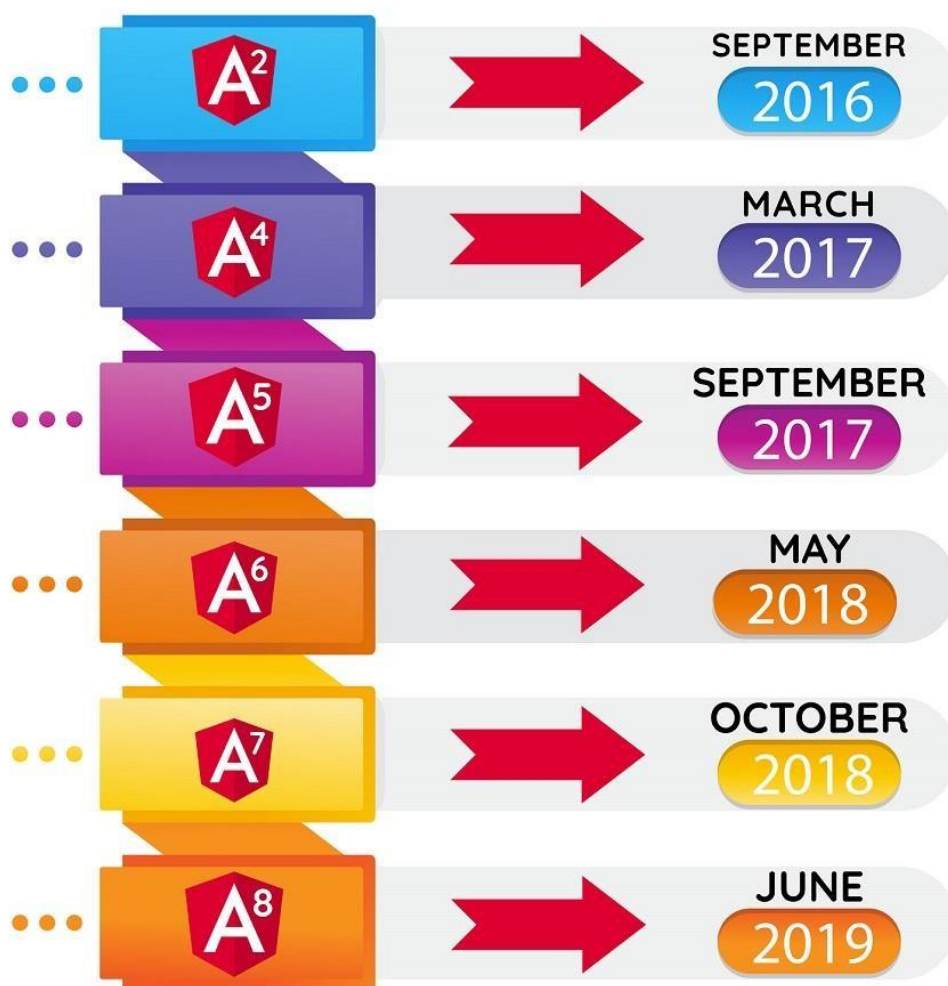
3. What are the differences between Angular2+ and Angular 1.x?

There are following differences between Angular2+ and Angular 1.x are given below:

Angular 2+	Angular 1.x
Based on components	Based on controller and scope
Improved DI	Supports DI
Mobile First	Not built with mobile first
Supports ES5/6, TS or Dart	Supports ES5/6 or Dart
Supports Angular CLI	Doesn't have CLI
The class is the only way to define services in Angular2	factory, service, provider, value and constant are used for services
Runs on client-side & server-side	Run on only client-side
bootstrapModule() function is used to initialize	ng-app and angular.bootstrap() function are used to initialize
Supports Pipe	Support filters
Supports camelCase and PascalCase syntaxes likengModel, ngForm and NgForm	Supports spinal-case and camelCase syntaxes like ng-model, ng-class and ngModel
Use Html DOM elements properties and events	Uses its own directives like ng-click, ng-show and ng-src etc.
Use () for events and [] for attributes	Doesn't support () and [] based syntax

4. Why uses Angular?

There are multiple reasons to use Angular over other front-end frameworks which are listed below.

- Angular is full-featured Single Page Application (SPA) framework
- The product of Google and completely open-source
- Follows MVC (Model View Controller) architecture
- Simple Dependency Injection implementation
- It supports Typescript and ES6 standard
- Able to create a re-usable component, modules, routes
- Supports to enable testing using jasmine and karma
- Great community support

5. Important fundamentals of Angular

Angular has large numbers of feature which makes it more useful over other front-end frameworks, below is the list of important features.

- Component
- Module
- Data bindings
- Services
- Dependency Injection
- Directives

- Templates
- Application Bootstrapping
- Navigation
- Native mobile development

6. What are the new features of Angular 7?

Recently Angular has released its newer version Angular 7, and with this version, we got several updated as well as the new features and these are listed below.

- CLI Prompt
- Angular Elements
- Updates in Angular Material
- Virtual scrolling
- Drag and Drop
- Setting budgets
- Checking bundle size
- Ivy rendering engine
- Typescript 3.1 supports
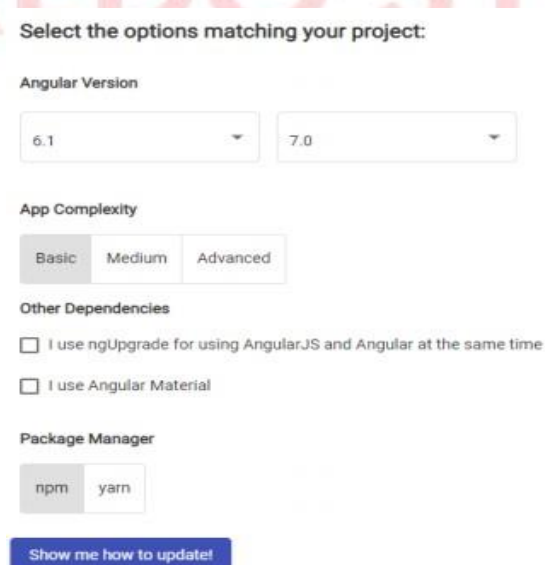
7. How to update your app to Angular 7?

If you want to work on the latest version of the Angular then we need to update our Angular CLI as well as Core angular packages by using below command.

8. How to update your older Angular app to Angular?

We can update our old angular app to the latest Angular 7 by following few steps carefully, but keep in mind that this updated guide can be applicable from Angular 2.0.

You can follow official guide to update your old angular app to a newer version and it will look like this.

9. What is a new feature called CLI prompt?

We know that it is difficult to keep in mind all those npm commands, but sometimes we may forget a few of them and get stuck, and at the end, we will find it with the help of different search engine. Using CLI prompts, now we can use different commands very easily like ng new and ng add.

```
PS C:\> ng new
? What name would you like to use for the project? Ng7Demo
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE Ng7Demo/angular.json (3777 bytes)
CREATE Ng7Demo/package.json (1315 bytes)
CREATE Ng7Demo/README.md (1024 bytes)
CREATE Ng7Demo/tsconfig.json (408 bytes)
CREATE Ng7Demo/tslint.json (2837 bytes)
CREATE Ng7Demo/.editorconfig (245 bytes)
CREATE Ng7Demo/.gitignore (503 bytes)
CREATE Ng7Demo/src/favicon.ico (5430 bytes)
```

When we type ng new and then press the enter key, it will ask three different named and the sequence is listed below.

- Application name
- Enable routing
- The format of the stylesheet

After providing appropriate values, it will create a new Angular 7 project.

10. How to create Angular Elements?

In order to use Angular Elements into our app, we need to add new core element package with the appropriate name as explained in below command.

   ng add @angular/elements --project=my-ngelement

Here the core package for elements is @angular/elements, and to create new element we need to pass the project name just like the above command.

11. Explain Angular Elements in Detail?

Angular Elements are the collection of components and it can be used by almost any framework which needs browser support.

It is just a simple package kind of collection where we have simple HTML tag and it acts as an independent element which is controlled and managed by the JavaScript.

This is also called web components, which is used to instantiate a JavaScript class to a normal HTML tag, before Angular 7, Angular 6.1 has a feature called ViewEncapsulation.ShadowDeom, but after the release of Angular 7, now we can use which is <slot>, let's see the simple example.

```
<my-element-demo message="My Custom Element">
</my-element-demo>
```

So, whenever we put this custom element into our HTML template, the instance will be created for the class and the element will be added to the HTML DOM.

```
Export class AppModule { constructor(private injector: Injector) {
constel = createCustomElement(DemoComponent, { injector });
customElements.define('my-custom-element', el);
}
```

As you can see into the above lines of code, we are going to inject the injector, and then we need to import elements as well like below.

```
Import { NgModule, Injector } from '@angular/core';
Import { createCustomElement } from '@angular/elements';
```

createCustomElement() is responsible to define custom elements globally using module file, and for registering our component as a new custom element we can use *customeElements.define()* method by providing element name.

12. What's new in Angular Material?
With the release of Angular 7, there are some highlighted changes which are listed below.

- Supports Virtual scrolling
- Supports Drag and Drop
- Can use native <select> in <mat-form-field>
- Updated component styles based on Material Design Specification 2018
- 250+ bug/performance fixes

13. How to update Angular Material to the latest version?
To work with latest Materialized components, we need to update our latest angular material package using below command.

ng update @angular/material

14. Explain Virtual Scrolling in Angular
Virtual scrolling is one of the new features added with the Release of Angular 7, which allow us to scroll the long list of elements, images, and other elements as well.

For virtual scrolling, there is a new module added called ScrollingModule to enable virtual kind of scrolling into our Angular application.

By using virtual scrolling, we can load and add different elements into the DOM and the same way we can also remove it based on the business requirements.

We need to import ScrollingModule from @angular/cdk/scrolling and then we can use this

feature, below is the simple Virtual scrolling example.

```
// app.module.ts

import{ NgModule } from '@angular/core';
import{ BrowserModule } from '@angular/platform-browser'; import{ FormsModule }
from '@angular/forms';
import{ AppComponent } from './app.component';
// Importing ScrollingModule
import{ ScrollingModule } from '@angular/cdk/scrolling';

@NgModule({
imports: [BrowserModule, FormsModule, ScrollingModule], declarations:
[AppComponent],
bootstrap: [AppComponent]
})
Export class AppModule{ }
```

Here in this module file, we have imported ScrollingModule from @angular/cdk and need to add
inside imports array.

To render the list of items, we will use the following code snippets.

```
// app.component.ts
import{ Component, ChangeDetectionStrategy } from '@angular/core'; @Component({
selector: 'my-app',
templateUrl: './app.component.html', styleUrls: ['./app.component.css'],
changeDetection : ChangeDetectionStrategy.OnPush,
})
Export class AppComponent {
// List of items from 1 to 100
myItems = Array.from({ length:100 }).map((_, i) =>`Item No : ${i}`);
}
```

Do not forgot to add some stylesheet into app.component.css.

```
/* app.component.css */

.viewport { height: 300px; width: 300px;
border: 1pxsolidblack; text-align: center;
}

.item {
height: 50px;
}
```

After running the above example, you will the output like this.
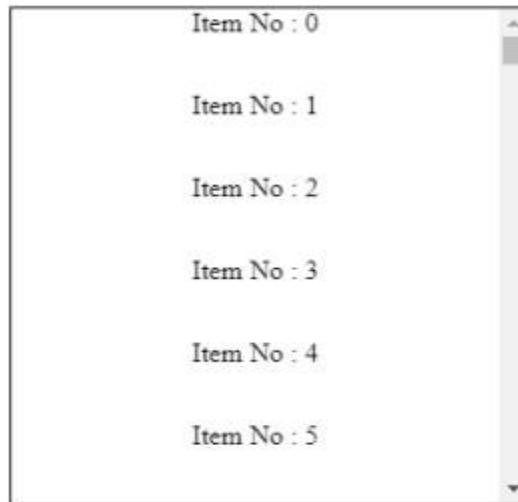
15. What is the Drag and Drop feature in Angular 7?

Another great feature released with Angular 7 is Drag and Drop and this feature can be used with @angular/cdk/drag-drop from the DragDrop module.

We know that in our older projects, we have developed our custom logic or any third-party library to implement drag and drop functionality but Angular 7 brings us this new feature which is a kind of plug and play element.

Not just only drag and drop but different other functionality can be implemented which are listed below.

* Simple Drag and Drop
* Reordering or sorting the list of items
* Use custom drag handle
* Transfer item from one list to another
* Different orientation can be possible
* Locking with axis

16. How to implement simple Drag and Drop?

To implement Drag and Drop we have a module called DragDropModule from @angular/cdk specification.

There are mainly two different directives to use Drag and Drop features.

* cdkDrag
* cdkDrop

In this example, we will implement simple draggable div using cdkDrag, follow the steps explained below.

```
// app.module.ts
import{ NgModule } from '@angular/core';
import{ BrowserModule } from '@angular/platform-browser'; import{ FormsModule } from
'@angular/forms';
import{ AppComponent } from './app.component';
import{ BrowserAnimationsModule } from '@angular/platform-browser/animations';
// Imported DragDropModule from CDK
import{ DragDropModule } from '@angular/cdk/drag-drop';

@NgModule({
imports:        [BrowserModule, FormsModule, DragDropModule], declarations:
[AppComponent],
bootstrap: [AppComponent]
})
Export class AppModule{ }
```

We have added DragDropModule which enables to use Drag and Drop feature, now let's implement the template part where we are going to usecdkDrag directive along with HTML div.

```
// app.component.html
<div class="box" cdkDrag>You can drag me</div>
```

Now we are done with our simple example, let's run this see that how its look like.

You can drag me

Now we can drag the div easily by just using simple directive cdkDrag, it's like magic using @angular/cdk without much complex implementation.

17. What is Budget bundle in Angular 7?

We know that size of the app is such a crucial thing, and for that we need to optimize the complete size of the application, using Angular 7 we can use Budget bundler to be notified whenever our application reached to the maximum size limit.

We can adjust these maximum size limits by using the angular.json file as explained below.

```
"budgets": [{
"type":"initial", "maximumWarning":"10mb", "maximumError":"15mb"
}]
```

This is how we can do budget bundling which is directly reflected in the application performance, so we should make sure that our application will be as small as possible.

18. How to check the current bundle size of an Angular app?

We should know about the application bundling size so then we can work on app optimization, but how do we know about the current project bundle size, it's pretty easy to check by using below command.

ng serve

And you will get complete bundling details like this into your console.

```
Time: 24086ms
chunk {main} main.js, main.js.map (main) 4.12 MB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 223 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.08 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 186 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 8.09 MB [initial] [rendered]
i ⌈wdm⌋: Compiled successfully.
```

19. What is Ivy Renderer engine in Angular?

Ivy in Angular is a view engine or we can say view rendering pipeline, which just like a compiler which takes component and template data than it will be converted into normal HTML and JavaScript, at last, reflected the DOM.

It is always interesting that how ivy renderer engine works and we can get to know more about Angular in depth by implementing such stuff.

Thus, Ivy renderer is not completely ready to use as suggested in this diagram.

The main goal of Ivy renderer engine is to optimize the speed and performance of the Angular application and provide backward compatibility by reducing bundle size.

20. What version of TypeScript, Angular 7 support?

It is official now that Angular 7 supports TypeScript 3.1, and it brings us many new features which are listed below.

- Mappable tuple and array types
- Easier properties on function declarations
- typesVersions used for version redirect
- Matching behavior
- Multiple fields
- Refactor from .then() to awaits

And also included some breaking changes like removed vendor specific declaration and difference in narrowing functions.

21. What's new in Angular 6?

Below are the new features as well as breaking changes.

- Angular Elements
- Ivy Rendering engine
- Bazel Compiler
- RxJs 6.0
- Typescript 2.7
- ng add
- ng update
- Angular Material + CDK Components
- Angular Material Starter component

22. What's new in Angular 5?

Below are the new features as well as breaking changes.

- Angular CLI v1.5
- Optimized and Incremental compilation
- HttpClient
- Typescript 2.4
- Rxjs 5.5.2
- Progressive web app
- Updates on Reactive forms and Template driven forms
- i18N Pipes
- Router life-cycle events
- Lambdas in Providers
- Faster AOT builds
- Angular CDK

23. What's new in Angular 4?

Below are the new features as well as breaking changes.

- Angular Universal
- Animations
- Typescript 2.1/2.2
- <ng-template>
- Pipes
- ngIf with else
- Http
- ParamMap in Router

24. What's new in Angular 2?

Below are the new features as well as breaking changes.

- Component based architecture
- Views
- Directives
- Modules
- Templates
- Metadata
- supports Typescript
- Services
- Router

25. What are the prerequisites for learning Angular?

There are some prerequisites to learn Angular, which are listed below.

- Basic knowledge of JavaScript
- Basic knowledge of HTML
- Basic knowledge of CSS
- Knowledge of Typescript add an advantage
- Any IDE to work with Angular

26. What IDE can be used to develop an Angular application?

To get started with the Angular application, we need IDE to get started with any technology, specifically for the Angular, we can choose any of the IDE listed below.

- Visual Studio
- Visual Studio Code
- Sublime Text
- Web storm
- Atom

- Angular IDE
- Brackets

These are some IDE which we can use to develop the Angular application end to end.

27. How to set up the Angular Environment?

Angular is a platform to create Single Page Application (SPA) using Html and Typescript, and main building block is Component, Modules, and Services etc.

To get started with the Angular, we need to set up the development environment, which requires following tools.

- **Nodejs**

To get started with Angular, we need to have node installed; if you don't have nodejs already installed into your machine then you can find the setup.

- **Npm**

Npm stands for Node Package Manager, and it contains the command-line client and using this we can install the various packages into our application.

For that, it manages the package registry with the complete details about the package, name, version, issue etc.

- **Angular CLI**

CLI is one of the important parts while getting started with Angular development and it is used to scaffold and build Angular app faster.It manages the entire tedious tasks like creating files, building the project, serving the project and many more tasks.

To use Angular CLI, we need to install the CLI by using below npm command.

`npm install -g @angular/cli`

After installing the CLI, if you want to know the current version then we can find the version like this.

```
PS C:\> ng --version

    _                      _                 ____ _     ___
   / \   _ __   __ _ _   _| | __ _ _ __     / ___| |   |_ _|
  / △ \ | '_ \ / _` | | | | |/ _` | '__|   | |   | |    | |
 / ___ \| | | | (_| | |_| | | (_| | |      | |___| |___ | |
/_/   \_\_| |_|\__, |\__,_|_|\__,_|_|       \____|_____|___|
               |___/

Angular CLI: 7.0.1
Node: 10.12.0
OS: win32 x64
Angular:
...
```

Any text editor tool as IDE

We can choose any of the code editors which are listed above in the IDE question. For Angular development, Visual Studio Code is a nice choice over the other code editors.

After following all of the above steps, now we are ready to get started with the Angular world.

28. What is the problem in Angular JS so we are using Angular?

The main reason to introduce Angular was that it's based on OOPs concepts. All the OOPs concepts can be supported by Angular. The second thing is that Angular is designed for web as well as mobile devices.

29. What is the need of TypeScript in Angular?

We use TypeScript in Angular for creating the application with components, services etc. and when we create the application, we can use the OOPs concepts.

30. What is the difference between various versions of Angular?

Version AngularJS is the first version of Angular released in 2010. It is also known as Angular 1. Angular 2 is the second version of Angular released in 2016.

JavaScript/TypeScript JavaScript is used in AngularJS whereas in Angular 2, TypeScript version 1.8 is used.

AngularJS is based on controller and view communication using $scope whereas Angular 2 is a component based approach.

Reusability of code We cannot reuse the code in AngularJS but Angular 2 provides us with the facility to reuse the code.

31. How can we run two Angular project simultaneously?

For running an Angular project, we use the following command.

ng serve

With the help of this command, the Angular project runs on port number 4200, i.e., localhost:\\4200. Now, if we want to run another project or want to change the port number of the same project, then we can use the following command.

ng serve --port 4210

32. What is bootstrapping in Angular? Is it possible to start Angular in any other way rather than app.module? If yes, then how?

Bootstrapping means starting an Angular application. Yes, it is possible. We can start Angular using our defined component instead of "app.module". For this, we should follow the following steps.

- Make a module file, say "test.module.ts", in app folder.

- Make a component using command "ng g c test" in app folder.

- Go to "test.module" file and register your component in this file and use "TestComponent" in declaration and bootstrap.

- Go to the "main.ts" file, put your module name "TestModule" into bootStrapModule method.

- Go to "index.html" file and put the test component tag inside body tag.

33. How does an Angular application get start?

An Angular application gets loaded or started by the following ways.

34. What is the Architecture Overview of Angular?
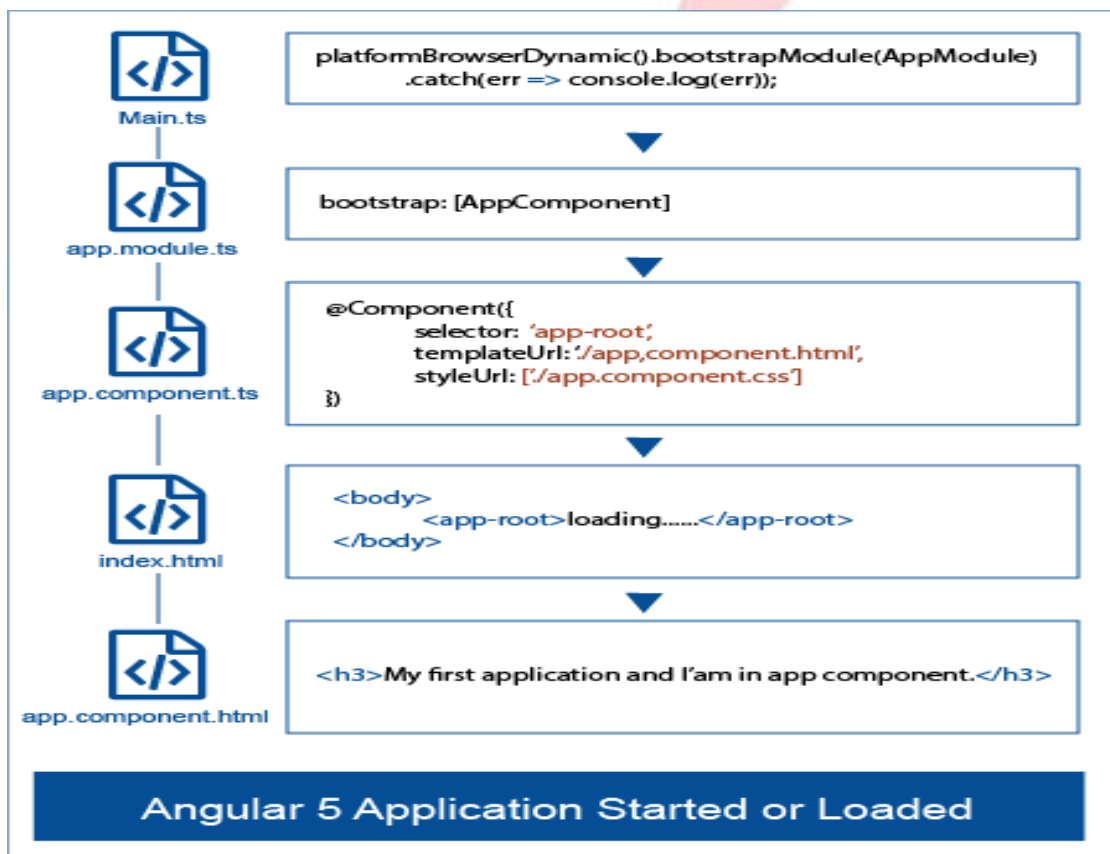
35. What is webpack?

Webpack is a tool used by an Angular CLI to create bundles for JavaScript and stylesheets. Webpack injects these bundles into index.html file at runtime.

36. We cannot reload our Angular application after changing in code. It automatically changes this code. How?

Just because of HMR (Hot Module Replacement/Reload) which is a feature of webpack. Due to HMR, we are able to see our changes without reloading the whole Angular application on a web browser.

37. Which files are bundled and injected in index.html at runtime by webpack?

The following files are bundled and injected in index.html at runtime.

- inline.bundle.js
- polyfills.bundle.js
- styles.bundle.js
- vendor.bundle.js
- main.bundle.js

38. What is the purpose of main.ts file?

The main.ts file is the main file which is the start point of our application. As you have read before about the main method the same concepts are here in the Angular application. It's the file for the bootstrapping the application by the main module as .bootstrapModule (AppModule). It means according to main.ts file, Angular loads this module first.

39. What is the role of package.json file?

It's a most important file for the Angular application. There are many settings in this file including dependencies and devDependencies. When we run npm install, Angular installs all the dependencies as defined in this file.

40. What is the role of tsconfig.json file?

The tsconfig.json is the configuration file and there are many setting for compiler. According to these settings the browser can understand it .

41. How package.json file is different from package-lock.json file?

Whenever npm modifies the node_module tree or package.json file, then package- lock.json is automatically generated.

- This file is intended to be committed into source repositories, and serves various purposes:
- Describe a single representation of a dependency tree such that teammates, deployments, and continuous integration are guaranteed to install exactly the same dependencies.
- Provide a facility for users to "time-travel" to previous states of node_modules without having to commit the directory itself.
- To facilitate greater visibility of tree changes through readable source control diffs.
- And optimize the installation process by allowing npm to skip repeated metadata resolutions for previously-installed packages.

42.What is Angular CLI?

Angular Command Line Interface is a command line tool which is used to create, initialize, scaffold and   to manage the whole angular application.

We can also use Angular Console tool to work with the command line to generate or work with different part of the Angular application, using CLI we can manage everything using commands.

Command Line Interface

You can find the complete details of Angular CLI and different commands into the CLI introduction part later on this series.

43.What are the features of Angular CLI?

Angular CLI is a powerful command-line tool by which we can create new files, update the file, build and deploy the angular application and other features are available so that we can get started with Angular development within few minutes.

- Packaging application and release for the deployment
- Testing Angular app
- Bootstrapping Angular app
- Various code generation options for Component, Module, Class, Pipes, Services, Enums, and many other types are also supported.
- Running the unit test cases
- Build and deployment our Angular application

44.How to use different Angular CLI Commands?

Angular Command Line Interface is the tool and by using this tool, we can scaffold, initialize, generate different files, and can build or deploy the angular application easily.

To use different commands, we should install the package by using npm below command.

npm install -g @angular/cli

45.What is Angular Console?

Angular CLI is a powerful tool and by using this we can develop an Angular application without following complex stuff.

Angular Console is a way to achieve different command line features using a simple GUI which is completely alternative to know all of the CLI commands.

To download the GUI tool, you can get the executable file from and after downloading the tool, it will look like this.
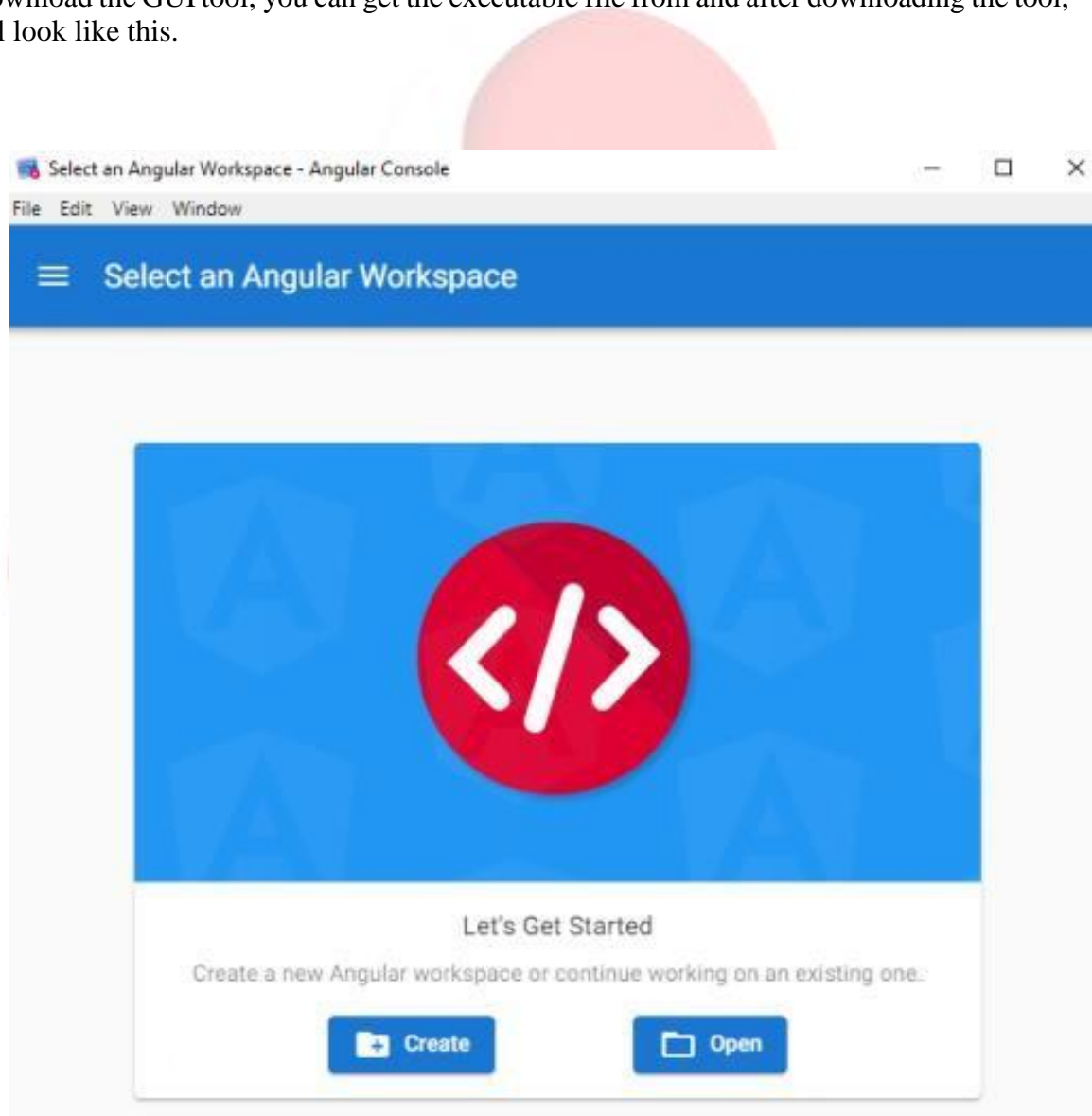
46.What are the advantages of Angular Console?

There are few advantages and features provided by Angular Console.

- Build CLI Commands visually
- Internal terminal for the output
- Import the existing Angular project
- Trivial code generation
- Run the custom NPM scripts
- Install various extensions

47.How to create a new Angular app?

To create a new Angular app using CLI command, then we can use below command.
 ng new <your_app_name>
It will take a few seconds or minutes based on our internet connection speed and then the new angular app will be created with the basic configuration.

48.How to serve the Angular app?

After creating a new Angular app, now it's time to run our app, for that we can use below command to serve our app.
 ng serve
After executing above given command, our application will start building and it can also be helpful to rebuild the app after any changes occurred.

If you want to open app automatically into a new window from the browser then we can use the flag –o like this.
ng serve –o

49.How to change default port number other than 4200?

If you have created new Angular app and server the app using ng serve, then our app will be executed on URL localhost:4200, but we can change it using additional flag along with ng serve which is –port.

ng serve --port 4300

50. How to change the default protocol other than HTTP?

By default, the Angular application works on HTTP protocol which is secure to use, thus we can change it to HTTPS which is an encrypted connection between client and server.

We can use additional flag along with command ng serve which is –ssl, and we can enable it using Boolean value true/false.
ng serve --ssl=true
 Now we will be able to run our Angular app using HTTPS securely, by default –ssl value is false.

51. How to update your project's dependencies?

To update the Angular's core packages to 7, we can use the ng update command like this.

ng update @angular/cli @angular/core

 It will update the above packages, and make sure that your project is fulfilled with all prerequisites.

52. How to generate various types of files using Angular CLI?

Angular CLI provides various features and one of them is to create the new file using just a single line of command. By using a command, we can generate various kind of files like Component, Module, Pipe and many more are listed below.

Command:

ng generate <file_type> <file_name>

Angular CLI commands for creating various types of files:

- *ng generate class <name>*
- *ng generate component <name>*
- *ng generate module <name>*
- *ng generate service <name>*
- *ng generate pipe <name>*
- *ng generate application <name>*
- *ng generate library <name>*
- *ng generate directive <name>*
- *ng generate enum<name>*
- *ng generate interface <name>*
- *ng generate guard <name>*
- *ng generate appShell<name>*
- *ng generate universal <name>*

 After running all of the above command, now we will be able to generate ready-to-use file which is one of the great features provided by Angular CLI.

53. How to update all dependencies of the Angular app?

We can update specific dependency as well as we can also update all of the project's dependencies using additional flag --all like this.

ng update --all=true

Now you can open the package.json file and can see that all of the dependencies are updated.

54. How to add a new dependency to the project?

Now we know that we can update dependencies, the same way we can also add dependency using the command.

ng add <dependency_name>

For example, we are going to add angular-material to our project.

ng add @angular/material

55. How to build an Angular application using CLI?

For the deployment purpose, we need to build our code to uglify all JavaScript files and to get ready to submit directory to the server.

Basically, it uses web pack build tools and the regarding data can be fetched from the angular.json file. We can use below command to build the project.

ng build

After executing the above command, the output folder will be created called /dist, but we can change the default directory to store the build output folder.

56. How to get project configuration details?

Sometimes it may possible that we need to know the details about project configuration like project name, default output directory, routing configuration, testing, and other important details.

By using below command, we will get the complete data of the angular.json file which is the global configuration file for the project.

ng config

When we open the PowerShell or command line, we can see the complete content of the configuration file.

57. How to test the Angular app?

Testing is the crucial task for any application, In Angular, we can test the angular application using Karma by using below command..

ng test

It will open a new browser window automatically and will give you brief information about the test cases prepared by us and if there is something wrong then it will show you the complete error details.

When we run this command, it will get the name of the project from the angular.json file where our testing related configuration is implemented.

58. How to get the version of Angular CLI?

To get the current version of the Angular CLI, we can use below command.

ng –version

After running above command, we will get the output like this.

```
Angular CLI: 7.1.2
Node: 10.12.0
OS: win32 x64
Angular: 6.1.10
... animations, common, compiler, compiler-cli, core, forms
... http, language-service, platform-browser
... platform-browser-dynamic, router

Package                             Version
------------------------------------------------------------
@angular-devkit/architect           0.11.2
@angular-devkit/build-angular       0.11.2
@angular-devkit/build-optimizer     0.11.2
@angular-devkit/build-webpack       0.11.2
@angular-devkit/core                7.1.2
@angular-devkit/schematics          7.1.2
@angular/cli                        7.1.2
@ngtools/webpack                    7.1.2
@schematics/angular                 7.1.2
@schematics/update                  0.11.2
rxjs                                6.2.2
typescript                          2.9.2
webpack                             4.23.1
```

59. How to specify the type of the stylesheet using the command?

Angular 7 provides the CLI prompt where we can choose stylesheet type from the available option, but for an older version, CSS was the default format for styling the pages.

But for that, we can use additional flag --style to specify any other supported stylesheet type than css, below is the simple example by which we can use specific stylesheet format while creating a new Angular app.

ng new <project_name> --styles <type>

Example:

ng new ngDemo --styles scss

After running above command, now our default stylesheet format will be SCSS, not the css, this is how we can change the stylesheet type.

60. How to get help from the CLI?

It is possible that sometimes we forgot any command or don't know about the specific command that how to use and for what the specific command is used for then we can ask for the help, where we will get the list of supported commands by CLI.

ng help

After running above command, we will get the list of different command with the appropriate description like this.

```
Available Commands:
  add Adds support for an external library to your project.
  build (b) Compiles an Angular app into an output directory named dist/ at the given output path. Must
  config Retrieves or sets Angular configuration values.
  doc (d) Opens the official Angular documentation (angular.io) in a browser, and searches for a given
  e2e (e) Builds and serves an Angular app, then runs end-to-end tests using Protractor.
  generate (g) Generates and/or modifies files based on a schematic.
  help Lists available commands and their short descriptions.
  lint (l) Runs linting tools on Angular app code in a given project folder.
  new (n) Creates a new workspace and an initial Angular app.
  run Runs a custom target defined in your project.
  serve (s) Builds and serves your app, rebuilding on file changes.
  test (t) Runs unit tests in a project.
  update Updates your application and its dependencies. See https://update.angular.io/
  version (v) Outputs Angular CLI version.
  xi18n Extracts i18n messages from source code.
```

61. Why do we use Angular CLI while we have another methods for Angular application creation?

Yes, we can create the application with another methods also, such as using Visual Studio (any version - 2015, 2017) but with Angular CLI, it's way easier to build a robust application.

62. Can we create the Angular application without Angular CLI?

Yes, with the help of Visual Studio. Any version of VS can do; such as - Visual studio 2015 or Visual Studio 2017 or later version.

63. What is npm and what is the need for Angular application?

NPM (Node Package manager) is the package manager which we are using for installing the dependencies which is required for the application.

In Angular, we have package.json file in which all the dependencies have added which we have installed. When we need any new dependencies, we can simply install using below command.

npm install <package name > --save .

It adds the installed package in node_modules folder and also, adds the dependencies in package.json file.

64. How Webpack is different to SystemJS?

Webpack is a totally different from SystemJS. It does not do same as SystemJS but in case of SystemJS, the systemjs.config.js allows us to configure the way in which module names are matched with their corresponding files. WebPack is a module bundler which bundles a file for application.

65. What is transpilation concept of Angular?

Transpiling is the process to convert the code from one high level language to another high level language. All modern browsers can only understand JavaScript and in Angular we wrote all code in TypeScript. So, in Angular, transpilation means that what you have written in TypeScript gets converted to another high level language, which is JavaDcript.

66. How can we run our Angular project? Which is the default port used by Angular?

We can run our Angular application using "ng serve" command, and by default, Angular uses port no 4200.

67. What is the difference between ng serve and ng serve –open?

ng serve and ng serve --open - both the commands are used to run our Angular application. However, if we use "ng serve" to run a project, then we need to open the browser manually and give address "localhost:\\4200" to see the project. In case of "ng serve --open" command, it opens the browser and runs the command automatically.

68. What are the commands for the following?

For creating new component	ng g c MyComponentName
For creating new service	ng g s MyServiceName
For creating new module	ng g m MyModuleName
For creating new directive	ng g d MyDirectiveName
For creating new pipe	ng g p MyPipeName
For creating routing guard	ng g g GuardName
For creating class	ng g cl MyClassName
For creating interface	ng g i MyInterfaceName
For creating enum	ng g e MyEnumName

69. What is the role of Angular-cli.json file?

The Angular-cli.json is the configuration file of Angular application in which we have many configuration settings which is required for Angular application.

70. What is the use of "assets" folder in Angular?

Whenever we build our Angular application with "npm run build" or "ng build --prod" commands, then Angular CLI moved all of our assets into the dist folder. It will do the same when it sees that there are images inside the assets folder. So, we can say that assets folder is used in Angular for maintaining the Angular assets as image etc which doesn't have to be modified while compiling.

71. What is Component in Angular?

An Angular component mainly contains a template, class, and metadata. It is used to represent the data visually. A component can be thought as a web page. An Angular component is exported as a custom HTML tag like as:

<my-app></my-app>.

When we create new create an Angular app using CLI, it will create default App component as the entry point of application. To declare a class as a Component, we can use the Decorators and decorator to declare a component looks like the below example.

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

@Component({
selector:                                                 'my-app',
template: `<h1>{{ message }}</h1>`, styleUrls: ['./app.component.css']
})
exportclassAppComponent                                                    {
```

```
message          =          "Hello          From          Angular";
}
```

For creating a component, we can use decorator @component like the above example. If we are using Angular CLI than new component will be created using the below ng command.

ng generate component componentName

72. What is the Template Expression?

Template expressions are the same expression that we use with JavaScript. But the difference is that we can use it within our Html template, so it looks like that we are using JavaScript along with Html but keep in mind that not every JavaScript expressions are supported like (new), (++) and (--).

Let's see a simple example of the mathematical calculation using template expression.

```
App.component.html
<p>Addition  :  {{  1  +  1  }}</p>
<p>Subtraction :  {{  25 - 50 }}</p>
<p>Multiplication : {{ 25 * 50 }}</p>
<p>Division : {{ 25 / 50 }}</p>
```

Here we have used {{ }} brackets in order to use an expression within our template, and you can see the output like this.

Addition : 2

Subtraction : -25

Multiplication : 1250

Division : 0.5

73. How to manage a web page using Angular Component?

A webpage containing an article with comments, categories, news feed and header, the footer can be managed using the angular components as given below:
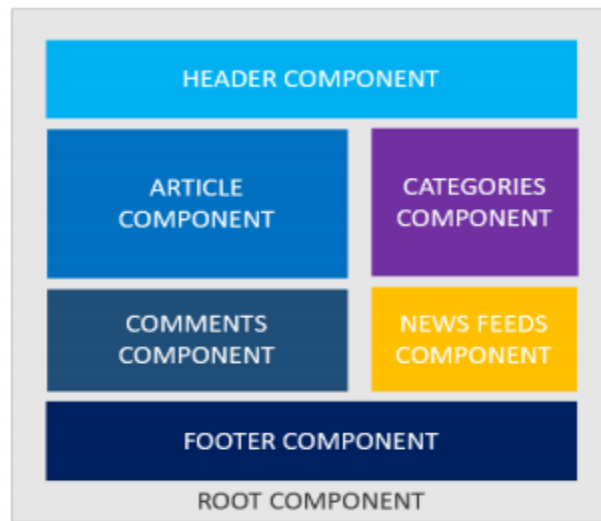
ROOT COMPONENT

74. What are the Template Statements?

Template statement will be triggered when an event will be raised using any binding target such as component, directive or a simple element.

Expression will always be the right side of the event using a (=) symbol like the following example.

```
<input type="button" (click)="clickevent()" value="Submit"/>
```

In the above example, we have a template statement and we are using the event clickevent (). This is called a template statement.

Same as template expression, not every JavaScript expression are supported in Template statements.

75. What is Data Binding in Angular?

Data binding is one of the important core concepts which is used to do the communication between the component and DOM./In other words, we can say that the Data binding is a way to add/push elements into HTML Dom or pop the different elements based on the instructions given by the Component. There are mainly three types of data binding supported in order to achieve the data bindings in Angular.

- One-way Binding (Interpolation, Attribute, Property, Class, Style)
- Event Binding
- Two-way Binding

Using these different binding mechanisms, we can easily communicate the component with the DOM element and perform various operations.

76. What is Interpolation?

Interpolation in Angular is used to show the property value from component to the template or used to execute the expressions.
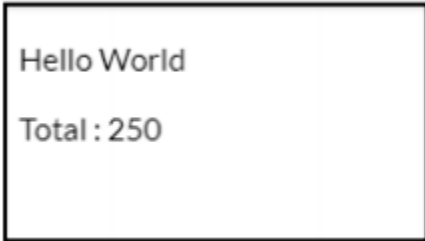
Generally, interpolation can be used to execute the JavaScript expressions and append the result to the Html DOM.

Another use of Interpolation is to print the data or we can say one-way data, which is coming from the component i.e. from a variable or the item of an array.

And interpolation can be used to achieve Property binding as well by using curly braces {{ }}, let's see a simple example.

```
App.component.ts

myMessage: string = "Hello World";

    App.component.html

<p>{{ myMessage }}</p>
<p>Total : {{ 25 + 50 + 75 + 100 }}</p>
```

The same way, we can use interpolation to different places or elements like the source of the image, values for the classes, and other use as well. The output will look like this.

```
Hello World

Total : 250
```

77. What is Property Binding?

Property binding is generally used to show or update the value of the variables in component and vice versa. We can also call it one-way data binding thus it is not used to update the two-way binding mechanism, and value will be reflected at the component level.

Let's see the simple example by taking simple string value from component to the template.

App.component.ts

We are going to create a string variable with a default value like this.

myMessage: string = "Hello World";

And to show this message into the template, we will use a {{ }} symbol which is called

interpolation like this.

App.component.html

`<h1>`{{ myMessage }}`</h1>`

Whenever we change the value of the myMessage variable, at that time it will be reflected in the template but not vice versa.

78. How Does Event binding work in Angular?

User activities are a common thing in a web application to do some actions, at that time click or mouse event will be used to fulfill the goal of an activity.

When the user clicks on any element, at that time it generates the event and into the event, we will perform the various operations to achieve something, for that we can use Event binding in Angular.

For event binding, the targeted event will be on the left side and the event name will on the right side. The event name is surrounded by the punctuations like (), [] or preceded by the prefix like (on-, bind-).

Let's understand event binding by using a simple example, where we will use click event on a button.

```
App.component.html

<button (click)="clickEvent()"> Click me to generate an event
    </button>
```

Here in this template file, we have used the click event and it's surrounded by the ()

App.component.ts

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

@Component(
{ selector: 'my-app',
templateUrl: './app.component.html',
```

```
styleUrls: ['./app.component.css']
})
Export class AppComponent { clickEvent() {
console.log("Button Clicked");
}
}
```

In the component file, we have created one method for handling click event, whenever the user clicks on the button, at that time our clickEvent () will be triggered.

79. How to implement Two-way data binding in Angular?

Two-way binding is one of the strongest binding mechanisms, where two different bindings are merged i.e. input and output bindings.

It means that Event binding works with the two different binding mechanisms at a time and generates the output based on the event which was triggered.

For two-way data binding, we will use the punctuation [( )] in order to bind the two-way data.

It can be happen using the directive called ngModel which listen to the input for any changes and return the output.

We will understand more by implementing the simple example, where we will type the text into the input and at the same time, we will get the value of the same input value.

App.component.html

```
Full Name : <input type="text"
[(ngModel)]="fullName"> Your Name is : {{ fullName }}
```

App.component.ts

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

@Component({ selector: 'my-app',
templateUrl: './app.component.html', styleUrls: ['./app.component.css']
})
Export class AppComponent { fullName:
string = '';
}
```

Here in this example, fullName is our variable into the component, and from our template, we are binding the value of textbox and returns to the same variable which holds the current value of fullName.

As we have discussed that it is combination of two binding mechanism which is equivalent to below statement.

```
<input type="text" [ngModel]="fullName" (ngModelChange)="fullName=$event">
```

80. What is Directive in Angular?

The directive in Angular basically a class with the (@) decorator and duty of the directive is to modify the JavaScript classes, in other words, we can say that it provides metadata to the class.

Generally, @Decorator changes the DOM whenever any kind of actions happen and it also appears within the element tag.

The component is also a directive with the template attached to it, and it has template- oriented features. The directive in Angular can be classified as follows.
1. Component Directive
2. Structural Directive
3. Attribute Directive

81.What is Component Directive in Angular?

A component is completely a form of Component Directive, and it is a simple directive with their own templates.

82.What is Structural Directive in Angular?

The structural directive is used to manipulate the DOM parts by letting us add or modify DOM level elements.

For example, we are adding records every time when we submit the form. To generate a new directive, we can use the following command.
ng generate directive mydirective

And the generated directive file can look like this.

```
// mydirective.directive.ts
import{ Directive } from '@angular/core'; @Directive({
selector:'[appMydirective]'
})
Export class MydirectiveDirective { constructor() { }
}
```

83.What is Attribute Directive in Angular?

Attribute directive is used to change the behavior or the appearance of different elements, and these elements act as an attribute for the DOM elements.

This is the same as custom or other directive but the difference is that we are changing the behavior of elements dynamically.

Let's see the simple directive example and create new directive named attrdemo.directive.ts and source code can look like this.

```
import{ Directive, ElementRef, Input, HostListener } from '@angular/core';
@Directive({
selector: '[appAttdirdemo]'
})
Export class AttdirdemoDirective {
@Input() appAttdirdemo: string; constructor(privateelref: ElementRef) { }
@HostListener('mouseover') onMouseOver() { this.changefontsize(this.appAttdirdemo);
}
@HostListener('mouseleave') onMouseLeave() { this.changefontsize('15px');
}
changefontsize(size) {
this.elref.nativeElement.style.fontSize = this.appAttdirdemo;
}
}
```

In this example, we are going to change the font size of a string using our custom directive, please find below code where I have used directive.

```
<h3 [appAttdirdemo]="'12px'"> This is a Attribute Directive</h3>
```

As you can see that I have used appAttdirdemo which is my directive name and based on a mouse event, it will change the font size.

84.How many types of built-in attribute directives are available?

Basically, there is three type of built-in attribute directives are available to use which are listed below.

- **NgStyle**

To add or remove the different styles of an HTML element

- **NgClass**

To set the different classes of an HTML element

- **NgModel**

We have used NgModel attribute directive which is used to implement two-way data binding.

85.What are various Structural directives in Angular?

Structural directives are used to manipulate the DOM by adding or removing different HTML elements at a time.

We can use the structural directives with the host element, and that it performs the different task based on the directive values.

Probably we can identify the Structural directive, and the main reason is that it contains (*) asterisk as a prefix with all Structural directive.

There are three types of Structural directives are available which are listed below.

- **NgIf**

Can add or remove the element based on the condition

- **NgSwitch**

Similar to switch case and it will render the element based on the matching condition based on the value

- **NgFor**

Used to repeat the element until it reaches to the total number of items into the array/list

86. What is the use of ngIf directive?

While developing an application, quite a few times we have a situation that we need to show some part of the page based on the specific condition, at the same time ngIf directive comes into the picture.

the ngIf directive will add or remove the element based on the condition i.e. true/false, let's see the simple
example to show specific div based on the condition.

In the component file, create one Boolean variable and provide a default value.

```
isVisible: boolean = false;
```

And in the template, we are going to create a simple div and also using a ngIf directive like this.

```
<div *ngIf="isVisible">
<h1>I am visible !!!</h1>
</div>
```

So whenever isVisible variable's value will be changed, based on that value the element will be added or removed from the native DOM.

87.What is the use of ngFor directive in Angular?

NgFor directive in Angular is used to iterate the array item; it will be iterated until it reaches to the total number of items in the list.

App.component.ts

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

@Component({ selector:'my-app', templateUrl:'./app.component.html', styleUrls:
['./app.component.css']
})
Export class AppComponent implementsOnInit { items = []; constructor() {
this.items = newArray();
}

ngOnInit() { this.items.push('Item 1');
this.items.push('Item 2');
this.items.push('Item 3');
this.items.push('Item 4');
this.items.push('Item 5');
}

}
```

And in the app.component.html file, we are going to create an unordered list by using ngfor directive.

```
<ul>
<li *ngFor="let item of items">{{ item }}</li>
</ul>
```

New list item will be added until it reaches the last item in the array, and the output will look like this.

- Item 1
- Item 2
- Item 3
- Item 4
- Item 5

88.What is the use of ngSwitch directive in Angular?

NgSwitch is completely similar to the switch case that we have used with JavaScript to select single element amongst the different available elements based on the condition.

In Angular, to use NgSwitch, we should use three different directives which are listed below.

- **NgSwitch**

This directive takes the value of the expression and based on the value, the specific test case will be selected.

- **NgSwitchCase**

NgSwitchCase acts as a specific case if it matches with the value of expression than it will add or remove an element from the HTML Dom.

- **NgSwitchDefault**

If none of the cases will be matched, then the element under the default case will be added to the HTML dom.

Let's see a simple example, in which we are going to pass the static value to the NgSwitch directive and will perform the operation based on the value.

App.component.html

```html
<div [ngSwitch]="selectedOption">
<p *ngSwitchCase="'option1'">Option 1 Selected</p>
<p *ngSwitchCase="'option2'">Option 2 Selected</p>
<p *ngSwitchCase="'option3'">Option 3 Selected</p>
<p *ngSwitchDefault>No Option Selected</p>
</div>
```

In our template file, we have used selectedOption as expression value and based on the value we will select any single option based on the matching value.

App.component.ts

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

@Component({ selector: 'my-app',
templateUrl: './app.component.html', styleUrls: ['./app.component.css']
})
export class AppComponent { selectedOption: string = 'option2';
}
```

As you can see into the above example, by default we have used value for expression which is option2 when we pass it to the NgSwitch. It will find the matching expression value from the available cases, and if one of them is the same as expression, then it will manipulate the Dom element. And if there are no matched cases found then it will render the default case element into the HTMLDom.

89.What are various Angular Lifecycle hooks?

Angular has its own Component lifecycle, and every lifecycle event will occur based on the data-bound changes.

Right from the initialization of a component to the end of the application, events of the lifecycle are processed based on the current component status.
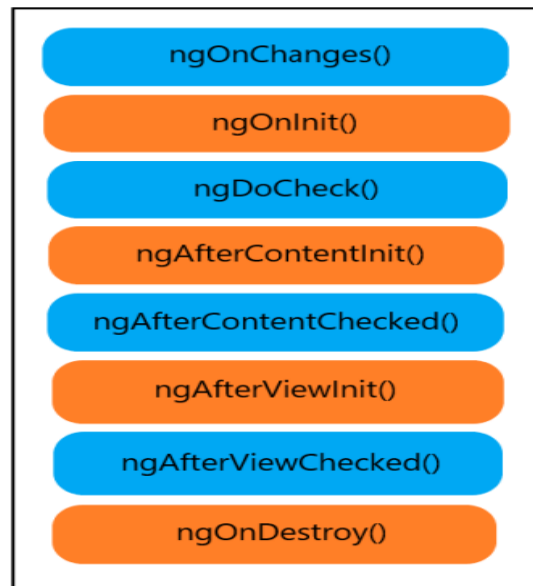
ngOnChanges()
This is the first method to be called when any input-bound property will be set and always respond before ngOnInit() whenever the value of the input-bound properties are changed. **ngOnInit()**
It will initialize the component or a directive and set the value of component's different input properties. And one important point is that this method called once after ngOnChanges ().

ngDoCheck()
This method normally triggered when any change detection happens after two methods ngOnInit() and ngOnChanges().

ngAfterContentInit()
This method will be triggered whenever component's content will be initialized and called once after the first run through of initializing content.

ngAfterContentChecked()
Called after every ngDoCheck() and respond after the content will be projected into our component.

ngAfterViewInit()

Called when view and child views are initialized and called once after view initialization.

ngAfterViewChecked()

Called after all of the content is initialized, it does not matter either there are changes or not but still this method will be invoked.

ngOnDestroy()

Called just before destroys of a component or directive and we need to unsubscribe any pending subscription which can harm us as extreme memory leaks.

90.Methods to Share Data between Angular Components

While developing an application, there is a possibility to share data between the components.

And sharing data between the components is our day-to-day kind of stuff where communication between components is a must either parent to child, child to the parent or communicate between un-related components.

To enable data sharing between components, different ways are possible which are listed below.

- Sharing data using @Input()
- Sharing data using @Output and EventEmitter
- Sharing data using @ViewChild
- Using Services

91.How to share data using @Input ()

This is the common way to share data from Parent component to child using @Input () decorator which allows us to get the input value from the parent component to the child component.

Let's learn the concept using a simple counter example, and for that, we should have two different components app and child component, and inside the child component, we are going to get the counter value.

The first step is to take two different buttons for increment and decrement the counter value, and for showing the counter value, we will use the child

```html
<!-- app.component.html -->
<div>
<p>Simple counter using @Input()</p>
<button (click)="increment()">+</button>
<app-child [counterValue]="counterValue"></app-child>
<button (click)="decrement()">-</button>
</div>
```

component.

Then we need to implement two different action methods for the increment and decrement like this.

```typescript
// app.component.ts
import { Component } from '@angular/core';

@Component({
selector: 'my-app',
templateUrl: './app.component.html'
})
export class AppComponent { counterValue: number = 0;

increment() {
this.counterValue++;
}

decrement() {
this.counterValue--;
}
}
```

We have implemented two methods to increase the counter value and another one is to decrease the counter value, but you may have noticed that we have used <app-child> which is our child component.

Open child component and declare @Input like this.

```typescript
// child.component.ts
import { Component, Input } from '@angular/core';

@Component({
selector: 'app-child',
template: `<h3>{{ counterValue }}</h3>`
})
export class ChildComponent {

@Input() counterValue: number;
}
```

In our child component, we have @input () decorator which accepts the input from the parent component and display the value into the child component template.

This is how we can achieve parent to child component communication via @Input () decorator.

92.How to share data using @Output and EventEmitter ()

This is another way to share the data from child to parent component by emitting the event and which can listen by the parent component.

@Output is a decorator which becomes the output for the parent component and in order to get the message from the child, we can use EventEmitter.

Let's learn by doing a simple example in which the user needs to enter their full name which is coming from the child component and print the full name into the parent component.

Create a new component called the child and paste the following lines of code into the child component.

```
// child.component.ts
import { Component, Input, Output, EventEmitter } from '@angular/core'; @Component({
selector: 'app-child', template: `<div>
<input type="text" [(ngModel)]='fullName'/>
<button (click)="submit()">Submit</button>
</div>
<div>
<p>Your Name is : {{ fullName }}</p>
</div>`
})
export class ChildComponent {

fullName: string;
@Output() inputChange = new EventEmitter();
```

```
submit() {
this.inputChange.emit({ fullName: this.fullName });
}

}
```

Let me explain a bit about this code, here we have used the simple textbox to enter the full name and when the user changes the value, at that time it will be reflected the fullName which is the model value.

And when the user clicks on the submit button, at that time using @Output, we are going to emit an event called inputChange which can be accessible by the parent component.

Now in parent component, we will use the value sent by the child component like this.

```
// app.component.ts
import { Component } from '@angular/core';

@Component({
selector: 'my-app',
templateUrl: './app.component.html', styleUrls: ['./app.component.css']
})
export class AppComponent {

changeInput(fullName) { console.log(fullName);
}
}
```

Using method changeInput, we will get the value of fullName using the event implemented into the child component, the template part looks like this.

```
<!-- app.component.html -->
<app-child (inputChange)='changeInput($event)'></app-child>
```

Here in this template file, we have event inputChange which is implemented in the child component, and whenever the value of the textbox will be changed at that time the event will be emitted directly. This is how we can use @Output and EventEmitter to communicate from child to parent component.

93.How to share data using @ViewChild ()

This is the way to share the data from child to parent component using @ViewChild () decorator.

ViewChild is used to inject the one component into another component using @ViewChild() decorator.

One thing to keep in mind about the ViewChild is that we need to implement AfterViewInit lifecycle hook to get the data because the child won't be available after the view has been initialized.

Let's see one simple example that how to use @ViewChild to inject the child component into the parent Component.

Here in this example, we are going to implement the counter example, and into the child component, there are two different methods to increment and decrement the counter value.

Now let's inject the child component into the parent using ViewChild like this.

```
// app.component.ts
import { Component, ViewChild } from '@angular/core'; import { ChildComponent } from
'./child/child.component';

@Component({
selector: 'my-app',
templateUrl: './app.component.html', styleUrls: ['./app.component.css']
})
export class AppComponent { @ViewChild(ChildComponent)
private childComponent: ChildComponent;

add() {
this.childComponent.increment();
}
```

```
sub() {
this.childComponent.decrement();
}
}
```

In our app component, we have injected child component using ViewChild and using different methods of child increment () and decrement (), we can add or remove the counter value directly into our parent component.

```
<!-- app.component.html -->
<button type="button" (click)="add()">+</button>
<app-child></app-child>
<button type="button" (click)="sub()">-</button>
```

When we try to click on the plus or minus button, the counter value will be reflected, this is how we can inject the child component functionality into the parent component using @ViewChild ().

94.How to share data using the Service

We have seen different ways to share data between the components like @Input(), @Output and @ViewChild, but these approaches are used for the component related to each other like a child to parent or parent to child.

But what if we don't have any direct relationship between the components, for this situation, we can use
Services to share data amongst different components.

One of the useful approaches is rxjs's BehaviorSubject because by using getValue() function, we can get the raw data as last value.

For example, we will use two component and one service file to communicate between them.

```
// MyDataService.service.ts
import { Injectable } from '@angular/core'; import { BehaviorSubject } from 'rxjs';

@Injectable({ providedIn: 'root' }) export class MyDataService {

private source = new BehaviorSubject('Message From the Service'); currentMessage =
this.source.asObservable();
constructor() { } changemessage(mymessage: string) {
this.source.next(mymessage)
}

}
```

Here in this service file, we have used rxjs BehaviorSubject which is going to
send the message as a raw data.

```
// app.component.ts
import { Component } from '@angular/core';
import { MyDataService } from './mydataservice.service';

@Component({
selector: 'my-app',
templateUrl: './app.component.html', styleUrls: ['./app.component.css']
})
export class AppComponent { mymessage: string;
constructor(private dataService: MyDataService) { }

ngOnInit() {
this.dataService.currentMessage.subscribe(message => this.mymessage = message)
}

}
```

In our app component, we are going to inject the mydataservice file from where we
are going to send the message.

Our next move is to create new component called newmessage, from where we are
sending new message to the parent component.
// newmessage.component.ts

```
import { Component } from '@angular/core';
import { MyDataService } from '../mydataservice.service';

@Component({
selector: 'app-newmessage', template: `
{{ message }} | <button (click)="sendNewMessage()">Send New Message</button>
`,
})
export class NewMessageComponent { message: string;
constructor(private dataService: MyDataService) { }

ngOnInit() {
this.dataService.currentMessage.subscribe(message => this.message = message)
}

sendNewMessage() { this.dataService.changemessage("Sending New Message")
}

}
```

In our newmessage component, we are also using the mydataservice file send the
message, but here we are going to send a new message to the parent component using
method changemessage ().

Our last step is to use child component into the parent template like this.

```
<!-- app.component.ts --> This is the message :
<strong>
{{ mymessage }}
</strong>
<app-newmessage></app-newmessage>
```

When we run this example, just click on the button to send the new message, and
a new message will be sent from the newmessage to the parent component.

When we use changemessage () function to any of the components, the new data will
be reflected in all other components.

95.Explain Change Detection in Angular

Change Detection in Angular detects any changes into and based on the changes it
will update the HTML DOM. Using zode.js along with Angular, it detects the async
events and allows Change Detection to the components. It's always traversing from
the UI tree from the root component to the child component and updates the DOM
based on the updated state. There are two different Change Detection strategies are
supported. ChangeDetectionStrategy.Default In this Default strategy, every
component will be checked against any changes.

ChangeDetectionStrategy.OnPush

OnPush Detection strategy detects the changes against any new reference are passed to them which is immutable. This strategy is based on the two things like the reference of the reference type changed and if changed than their value has changed into heap memory or not. We will see the simple example using OnPush strategy than we will get to know about the mechanism.

Let's assume we have 3 different component called app, child and subchild and basically we are going to push records and will see how it works.

```typescript
// app.component.ts
import { Component } from '@angular/core';
```

```typescript
import { ChildComponent } from './child.component';

@Component({ selector: 'my-app', template: `
<child [users]="userList"></child>
`,
})
export class AppComponent { userList: any[] = [];
// Push 5 users constructor(){
for(let i=0; i<5; i++){ let num = i + 1; this.userList.push({
name: 'user ' + num,
})
}
}
}
```

In our app component, we are going to push the five records into the user array, and you may notice that we are going to pass the records to the child component.

Now let's open child component and the code snippet look like this.

```typescript
// child.component.ts
import { ChangeDetectionStrategy, Component, Input } from '@angular/core';

@Component({
selector: 'child', template: `
<h3>List of the users</h3><hr/>
<sub-child *ngFor="let user of users" [user]="user"></sub-child>
`,
changeDetection: ChangeDetectionStrategy.OnPush // Change detection strategy
})
export class ChildComponent { @Input() users: any[];
}
```

Into our child component, you may notice that we have imported ChangeDetectionStrategy which is part of @angular/core package.

And to enable Detection strategy, we can use ChangeDetection along with its value type either Default or OnPush strategy.

From parent component, we are getting the list of the users and at the same time we are passing these records to the subchild component and it will look like this.

```ts
// subchild.component.ts
import { Component, Input } from '@angular/core';
```

```ts
@Component({
selector: 'sub-child', template: `
<p>{{ user.name }}</p>
`,
})
export class SubchildComponent { @Input() user: any;
ngDoCheck() {
console.log("DoCheck Event Occured");
}

}
```

Our subchild module will be called five times because we are going to render five different users, and you can see that we have used ngDoCheck() lifecycle hook which is used to detect every change in the child component.

Whenever we open browser console, we can see that change detection happens five times as we have pushed five different records from the parent component.

```
DoCheck Event Occured
DoCheck Event Occured
DoCheck Event Occured
DoCheck Event Occured
DoCheck Event Occured
```

This is how OnPush Change Detection strategy works by enabling the immutable state object which directly reflects on the performance.

96.Explain Content Projection in Angular

Content Projection in Angular is one of the important concepts. It is also called Transclusion which is used to define a fixed part of the template and at the same time, we can render the dynamic content as well.

Content Projection will work by providing markups as element's content, and then we

can project our content into the specific div or a specific position into the template.

We can implement Content Projection by using **<ng-content>** directive to place the projected content into our template.

Let's learn Content Projection by using a different example and we are going to use 2 different components which are app and child component.

```html
<!-- app.component.html -->
<app-child>
This is content for projection
</app-child>
```

Here in the above template file, I have used <app-child> directive which is my child component.

```ts
// child.component.ts
import { Component, Input } from '@angular/core';

@Component({
selector: 'app-child',
template: `<ng-content></ng-content>`
})

export class ChildComponent {}
```

This is our child component, and you may have noticed that we have used <ng-content> directive to project the content. For this example, I have used a simple line of string inside the
<app-child> directive which is going to render inside the projected element.

Our line of the string will be rendered inside the child template's <ng-content>, this is how easy to do the content projection.

We can also use <ng-content> along with the selector, so that we can project the content more specifically.

```html
<!-- app.component.html -->
<app-child>
<footer>
This is my footer section
</footer>
</app-child>
```

In our template file, we have used the tag <footer>, the same way we can also create different div based on the requirement.

Now our goal is to render the footer part into our projected content from the child component.

In this example, we have used "select" attribute along with ng-content which is used to define the selector for our slot.

We can also use selector with css class, below is the simple example for that.

```
// child.component.ts
import { Component, Input } from '@angular/core';

@Component({
selector: 'app-child',
template: `<ng-content select="footer"></ng-content>`
})



export class ChildComponent {}
```

In our template file, we have used two different div element with the different CSS classes, now let's select any single div inside the content.

```
<!-- app.component.html -->
<app-child>
<div class="my-css">
This is my footer section
</div>
<div class="my-css1">
This is my footer section 1
</div>
</app-child>
```

Our child component tries to load the content using CSS class selector which we have used inside our app template and based on the class name, it will render the specific elements content.

When we run above example, the output looks like this.

```
// child.component.ts
import { Component, Input } from '@angular/core';

@Component({
selector: 'app-child',
template: `<ng-content select=".my-css"></ng-content>`
})

export class ChildComponent {}
```

It means that only first div matched whose class name is my-css and another was ignored to be projected inside the child component.

This is how we can Project the Content based on the attribute, HTML tag, and CSS class name.

97. Can we add more than one component in bootstrap in @NgModule?

Yes, we can add more than one components within bootstrap in @NgModule.

98. What is Attribute binding? What is the benefit of using it?

Attribute binding is used to bind attribute of an element with the field of a component. It is useful when we need to bind attribute,i.e. attribute cannot be bind with property as well as interpolation method. For example –

```
<tr>
  <td colspan="{{1 + 1}}">Employee's Record</td>
</tr>
<!-- Above line generates an error, because "colspan" is an attribute
  not a property, so, we bind it like -->
<td [attr.colspan]="1+1">Employee's Record</td></tr>
```

99. What is template reference variable?

If you want to access DOM properties of your element then you can use "Template Reference Variable", or we can say, to allow for elements to access other elements from within a template, you can create reference variables on elements, known as template reference variables.

100. How many ways we can create template reference variable?

You can create template reference variable by two ways i.e. by using "ref" keyword; in the below example, "myVariable" is a template variable.

```
<input type="text" placeholder="Click on button" ref-myVariable>
```

By using "#" , in the below example "myVariable" is a template variable.

```
<input type="text" placeholder="Click on button" #myVariable>
```

101. Can we use template reference variable using select (combo box). If yes, then how?

Yes, we can use template reference variable with select. For example,

```
<select #myValue (change) = "setStudentRecord(myValue.value)">

</select>
```

Look at the above code, #myValue is a template reference variable. The selected value of select box can be accessed by myValue.value.

102. Which binding we will use for dynamic css?
NgStyle and NgClass we are using for dynamic css. For example:
NgStyle:

```
<div [ngStyle]="{'background-color':person.country === 'UK' ? 'green' : 'red' }">

<div>
```

In the above we are checking the country value which is of person object and showing the style according to condition.

NgClass:

```
<div [ngClass]="{'text-success':person.country === 'UK'}">
</div>
```

As you can see in the above, we are checking the country value and according to value applying the class using Ngclass.

103. What is component decorator?

Component decorator allows you to mark a class as an Angular component and provide additional metadata that determines how the component should be processed, instantiated and used at runtime as you can see below:

```
@Component({
    selector: 'my-comp',
    templateUrl: './mycomp.component.html',
    styleUrls: ['./mycomp.component.css']
});
```

104. What is the mandatory property of @Component() decorator function?

"selector" and "template or templateUrl" are the mandatory properties for @Component() decorator function. If you miss the selector property then how can you use it, because selector is used to render the template, and, if you miss the template property then it generates compile time error like:
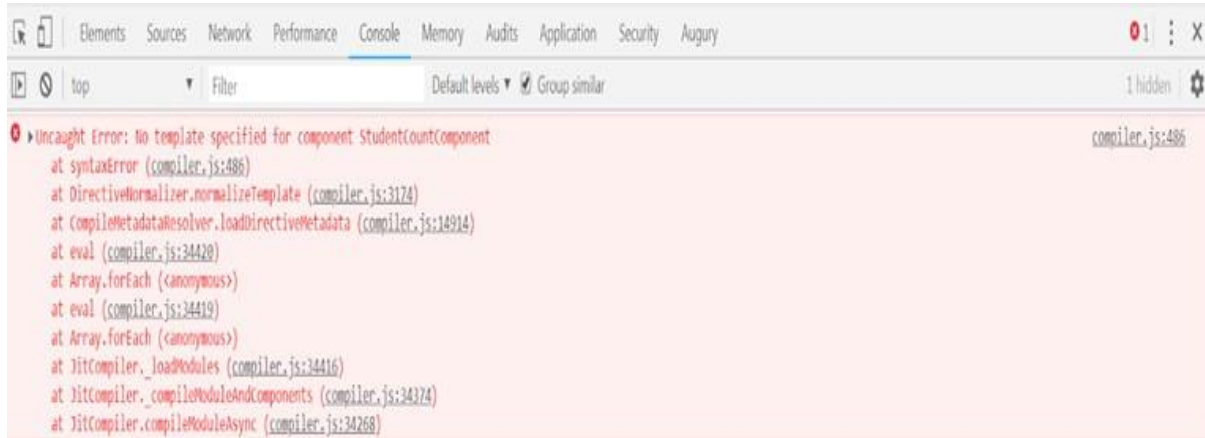
105. What is ViewEncapsulation?

ViewEncapsulation decides whether the styles defined in a component can affect the entire application or not. There are three ways to do this in Angular:

Emulated: When we will set the ViewEncapsulation.Emulated with the @Component decorator, Angular will not create a Shadow DOM for the component and style will be scoped to the component.One thing need to know necessary it's the default value for encapsulation.

Native: When we will set the ViewEncapsulation.Native with the @Component decorator,Angular will create Shadow DOM for the component and style will create Shadow DOM from the component so style will be show also for child component.

None: When we will set the ViewEncapsulation.None with the @Component decorator, the style will show to the DOM's head section and is not scoped to the component. There is no Shadow DOM for the component and the component style can affect all nodes in the DOM.

106. What is the difference between templeteUrl and template?

The templateUrl and template both are the part of @Component decorator. We are using templateUrl when we will add the path of any template file means html file and use template when we need to add only html text of entire html page which required for the component

107. What is the difference between styleUrl and style?

The styleUrl and style, both are the part of @Component decorator. We prefer using styleUrl when we add the path of any style file and style when we need to add only required style for the page required for the component.

108. What is the difference between providers present in component and present in app.module.ts file?

Basically, provider is used to implement the concept of dependency injection (DI). It takes a list of services name. So, if we inject service name into Provider inside the app.module.ts file then it behaves like global service which present all over the application and we can use any component, however, if we inject service name into Provider inside component file then it behave local service which we can only use in current component.

109. What is Dynamic Component?

Component templates are not always fixed. An application may need to load new components at runtime. So when we create component in such a way that they can load runtime, these types of component called Dynamic Component.

110. What is the use of ComponentFactoryResolver Service?

Whenever, we create a Dynamic Component then we need to use this service. This service can be used to render a component instance into another component's template. However, this is similar to the $compile method in AngularJS but this is better than the old $compile method as it encapsulates each template into a child component.

111. What is nested component?

When we will create any two component as first component and second component. Now if we will use one component selector within another component so its called nested component

For example: If we have one component as:

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

@Component({
 selector: 'app-second',
 template: `
 <h2>My second Component</h2>

 `
})
export class SecondComponent {
 constructor() {}
}
```

Another Component as:

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

@Component({
 selector: 'app-first',
 template: `
 <h2>My First Component</h2>
 <app-second></app-second>
 `
})
export class SecondComponent {
 constructor() {}
}
```

You can see in above, I am using the selector of one component with in another component so it's the example of nested component. We can say one component as Parent as in above First component where we have added the selector of second component and child as Second component.

112. What is the role of selector?

We are using selector of any component for identifying this component in templates.

113. If there is nested component (parent and child) then order of lifecycle hook is same or different?

No the order of lifecycle hook is different as you can show below
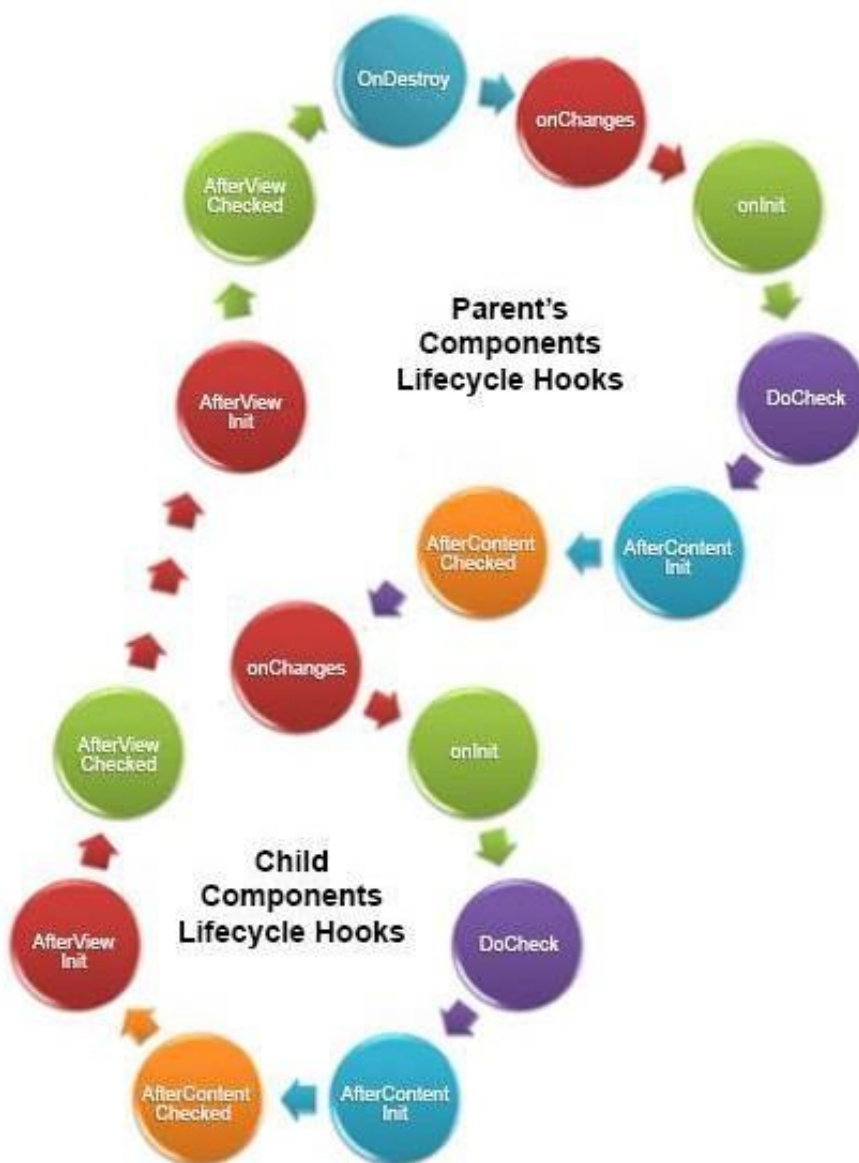
114. What is the difference between @ViewChild and @ViewChildren?

@ViewChild and @ViewChildren both are decorator, The basic difference between both are @ViewChild() provides the instance of another component or directive in a parent component and then parent component can access the methods and properties of that component or directive. In this way, by using @ViewChild() a component can communicate with another component or a directive. But if we want to access multiple child references then we will use @ViewChildren using query list.

115. What is the difference between @ContentChild and @ContentChildren?

@ContentChild and @ContentChildren both are decorators and we are using thes when we want to fetch single child element or all child elements from content DOM means within <ng-content></ng-content>.

116. What is the difference between ngDoCheck and ngOnChange?

ngDoCheck is very similar to ngOnChanges hook, the major difference is that ngOnChanges does not detect all the changes made to the input properties. It detects changes for those properties which are passed by value. However, ngDoCheck detects changes for those properties also which are passed by reference such as arrays.

117. How can we pass data from component to component?

Components Communication in Angular

Child to Parent   **Parent to Child**   **Child to Child**
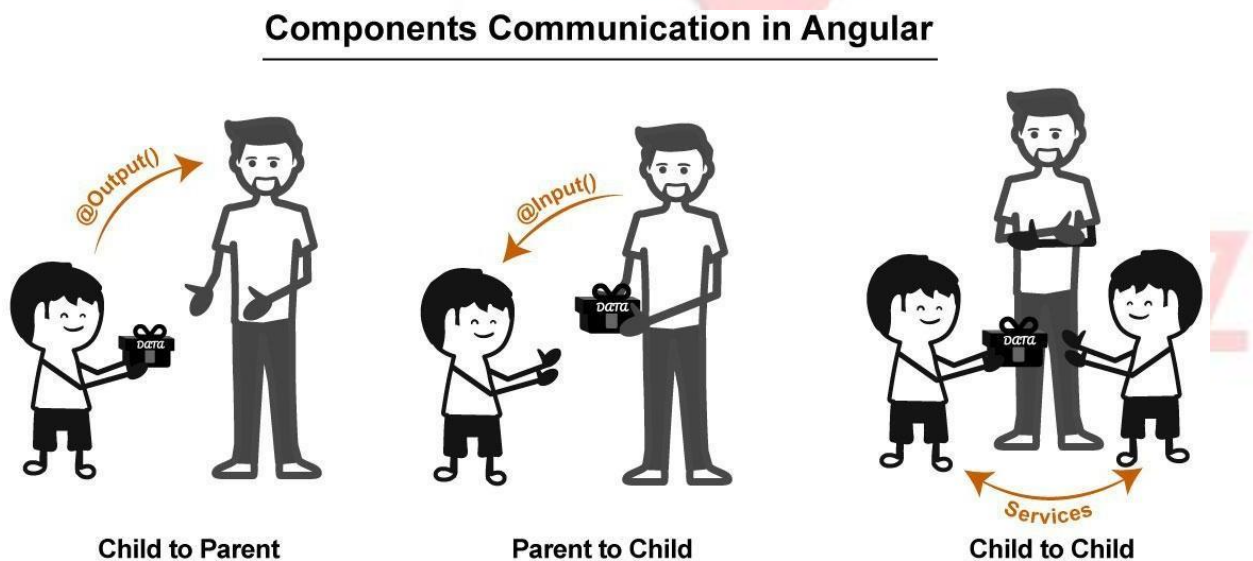
We can pass the data from component either parent and child relationships or not. If component are separate and we want to pass data between component so we can use services with get and set properties.set for setting the values which you want and get for getting the values which added by one component .

118. What is Event Emitter?

As we are using component in Angular and there are many events which we are using so component emits the event using @output and event emitter. When we pass the value from child to parent, the Parent can emit the value when wanted using event emitter.

For example:

```
@Output() Change: EventEmitter<number> = new EventEmitter<number>();
```

119. What is the difference between ngIf and hidden

They both are different due to their rendering methods, i.e. *ngIf can render and add element into DOM if condition is true, whereas hidden property had already rendered and added elements into DOM, it only shows and hides these elements on the basis of condition.
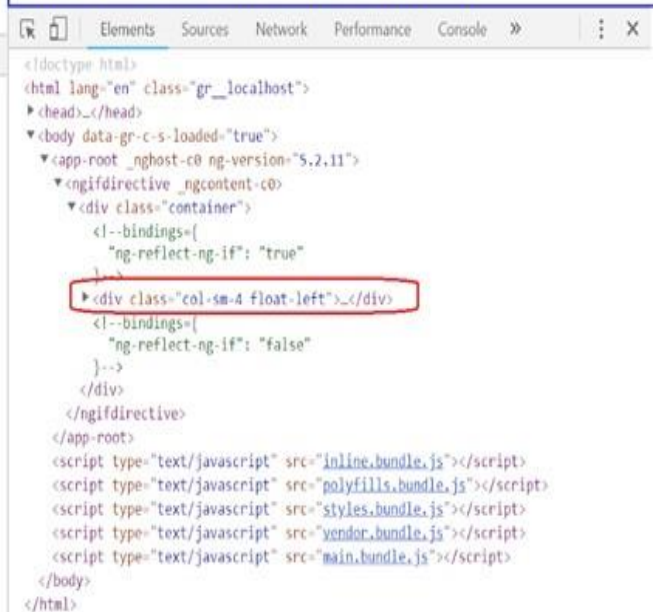
120. How can we use "then" and "else" keywords with *ngIf directive? Explain with an example

With the help of "ng-template" and "template reference variables" we can use "else" and "then" keywords with ngIf, because if condition is true then it will execute the "then" part otherwise execute "else" part. For example

```
<div *ngIf="technology=='angular'; then edu1 else edu2"> </div>
<div class="container">
    <ng-template #edu1>
        <div class="col-sm-4 float-left">
            <div class="card">
                <p class="bg-dark text-white p-2">EDU Learning Zone</p>
                <div class="p-3">
                    <p>Hello! How are you ?</p>
                </div>
            </div>
        </div>
    </ng-template>
    <ng-template #edu2>
        <div class="col-sm-4 float-left">
            <div class="card">
                <p class="bg-primary text-white p-2">EDU Learning Zone<
                <div class="p-3">
                    <p>Hello! How are you ?</p>
                </div>
            </div>
        </div>
    </ng-template>
</div>
```

121. What is the difference between ngFor and ngForOf?

ngFor and ngForOf are not the different things - they are actually the selectors of the NgForOf directive.

Whenever, we use *ngFor, the Angular compiler converts it into *ngForOf, like below.

We write

```
<div *ngFor="let record of items"> </div>
```

Due to *, first it converts into

```
<template [ngFor]="let record of items">
  <div> </div>
</template>
```

Then, due to "let", which is present in "let record of items", it again compiles and converts into

```
<template ngFor let-item="$implicit" [ngForOf]="items">
  <div>...</div>
</template>
```

122. Why "*" is prefix with structural directive? Can we use structural directive without using *?

Angular translates the * into a <ng-template> element, wrapped around the host element, for example

```
<div *ngIf="student"class="name">
  {{student.name}}
</div>
```

Angular convert this into

```
<ng-template [ngIf]="student">
    <div class="name">{{student.name}}</div>
</ng-template>
```

*Yes, we can use structural directive without using *, but in that case, line of code will increase like*

```
<ng-template [ngIf]="student">
    <div class="name">{{student.name}}</div>
</ng-template>
```

123. How many structural directives can we implement on a single element?

one, if we want to use more than one structural directive on a single element then we need to use <ng-container> like in the below line, there are 2 structural directives on one element i.e. *ngFor and *ngIf, which is not possible

```
<tr *ngFor="let record of studentList" *ngIf="record.gender=='Male'">
```

So, we should use ng-container like

```
<ng-container *ngFor="let record of studentList">
  <tr *ngIf="record.gender='Male'">
    <td>{{record.name}}</td>
  </tr>
</ng-container>
```

124. Can we provide our own mechanism for tracking the elements? If yes, then how?

Yes, we can provide our own mechanism for tracking items in a list by using trackBy. We need to pass a function to trackBy, and this function takes two arguments, an "index" and the "current item". trackBy is used for performance optimization.
For example

```html
<ul>
  <li *ngFor="let record of studentList; index as i;  trackBy: studentRecord">
      {{i + 1}}: {{record.studentId}} -
      {{record.name}} - {{record.age}}
  </li>
</ul>
```

In component, define function which is used in track by

```typescript
studentRecord(index: number, student: Student) {
  return student.studentId;
}
```

125. What is ng-template?

ng-template is an Angular element for rendering HTML. It is never displayed directly. In fact, before rendering the view, Angular replaces the <ng-template> and its contents with a comment.

126. What is Host Listner?

@HostListener is the decorator in Angular which we are using during the custom directives creation so you can say, we can use it for events on the host element or component.

127. What is ElementRef?

ElementRef is a class that can hold a reference to a DOM element. In Angular we can use it when we will play with custom directive. We are following below steps when we will use ElementRef:

Import the ElementRef from @Angular/core as:

```typescript
import { ElementRef } from '@angular/core';
```

 Inject it within constructor of component as:

```typescript
constructor(private hostElement: ElementRef) {

}
```

Now we can use as DOM manipulation with the help of hostElement which we have added in the constructor.

128. What is the purpose of @HostBinding?

@HostBinding is also the decorator which we will use at the time of custom directive creation. In Angular, we use it for setting the properties on the element or component that hosts the directive.

129. What are the difference between Patch() and Put()?

Patch and Put both will use for update the data, but the major difference is that Put is used for updating all the data (all fields), however, Patch is used for updating partial data (a few fields).

For example-

Suppose, we have a "address form" object in which few fields of address related and if user want to change their address means all the fields of "address from" object so, in this case we will use Put method, but if user want to change only street not locality or area so, in this case we use Patch method.

In short, we can say that

PUT - For update a resource (by replacing it with a new version) PATCH - For update part of a resource (if available and appropriate)

130. How many decorators are there in Angular?

We have mainly following decorator as:

Class decorators
> Class decorator is Top level decorator and they allow us to tell Angular that a particular class is a component, or module.
> For example : @Component and @NgModule

Property decorator
> These are second most common decorators that we are using within Angular application. We can create specific properties with in class with the help of property decorator and use with the class when communication within parent and child or child and parent.
> Example : @Input and @Output

Parameter decorator
> Parameter decorators also important and play a important role within Angular application. We can use when we have to inject the parameter .
> Example: @Inject

Method decorator.

Method decorator are also a most common decorators that we are using within Angular application. We can create specific methods within our class with required functionality. Example: @HostListner.

131. What is backtick and how we use it in Angular?

In Angular 2-4 or later version we are using Backtick for multiple line HTML. For example, if we want the long HTML file data within separate lines, then we can use the Backtick.

132. What is DOM Shadowing?

Dom Showing allows us to hide DOM logic behind other elements. We can say it enables us to apply the style according to scope and requirement of application.

133.What is Module in Angular?

The module in Angular is a very essential part of the application, which is used to organize our angular app into a distributive manner.

Or we can say that Modules plays an important role to structure our Angular application.

Each Angular application must have one parent module which is called app module and it contains the list of different items which are listed below.

- Components
- Service providers
- Custom modules
- Routing modules
- Pipes

Module file group all of them into a single file; the same module will be useful for bootstrapping the application.
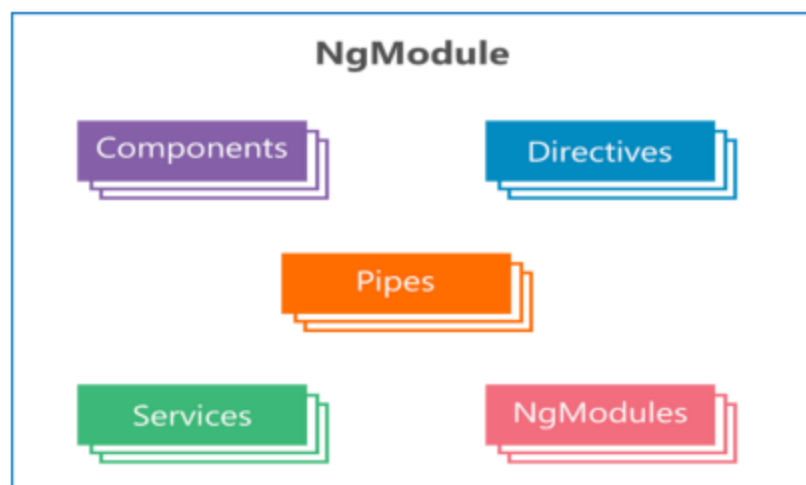
134.How to create a new Module using Angular CLI?

To get started with the modules, we need to create the module file by using below ng command.

ng generate module mycustommodule

135.What different types of Modules are supported by Angular?

Modules in Angular are categorized into 5 types which are listed below.

- Routing module
- Service module
- Features module
- Widget module
- Shared module

136. Explain Angular Module basic structure?

Every module file we create has a specific structure which looks

like this. App.module.ts

```
import{ BrowserModule } from '@angular/platform-browser'; import{ NgModule } from '@angular/core';
import{ FormsModule } from '@angular/forms'; import{ AppComponent } from './app.component';

// Routing
import{ routing } from'./routes';

@NgModule({ declarations: [
AppComponent,
],
imports: [
BrowserModule, FormsModule, routing,
],
exports: [],
providers: [], bootstrap: [AppComponent]
})
Export class AppModule{ }
```

As you can see into the above code snippet, by default module file can be declared using decorator @NgModule, and it will contain the various metadata options like:

- Declaration
- Imports
- Exports
- Providers
- Bootstrap

137. What are the various NgModule metadata properties?

There is a list of metadata options is available to use with NgModule which are listed below.

- Declaration
- Imports
- Exports
- Providers
- Bootstrap
- entryComponent
- Id
- Jit
- Schemas

138. What are built-in Modules?

Angular has built-in library modules starting with the @angular as a prefix. Built-In library & third part modules can be installed using npm manager. Built-In modules, components, services, directives etc. can be imported by using built-in library modules.

139. What are the different Feature Modules?

Feature modules can be categorized into the following group of modules.

- Domain feature modules
- Routed feature modules
- Routing modules
- Widget feature modules
- Service feature modules

140.How to create a Feature Module?

Feature module can be created using ng command along with the appropriate module name like this. ng generate module myfeaturemodule

It will create the feature module file and contains the code snippet like this.

```
import{ NgModule } from '@angular/core'; import{ CommonModule } from
'@angular/common';

@NgModule({ declarations: [], imports: [
CommonModule
]
})
Export class MyfeaturemoduleModule{ }
```

141.What is the difference between the JavaScript module and NgModule?

JavaScript module can be written into the separate JavaScript file and can export them, but NgModule is a Typescript class decorated with a @NgModule decorator.

142.How to create a Module with routing using CLI?

We can configure the routes along with the Modules by using the command, which generates the complete routing file as explained below.

ng generate module mymodule--routing

Here in this command, you can notice that we have used the additional flag –routing which is used to create a Module for the routing as well as the normal module file.

After running above command, it will create one folder along with the two files.

1. Mymodule-routing.module.ts
2.

```
import{ NgModule } from '@angular/core';
import{ Routes, RouterModule } from '@angular/router'; const routes: Routes = [];
@NgModule({
imports: [RouterModule.forChild(routes)], exports: [RouterModule]
})
Export class MymoduleRoutingModule{ }
```

   3. Mymodule.module.ts

   4.

```
import{ NgModule } from '@angular/core'; import{ CommonModule } from
'@angular/common';
import{ MymoduleRoutingModule } from './mymodule-routing.module';

@NgModule({ declarations: [],
imports: [
CommonModule, MymoduleRoutingModule
]
```

143. What is Entry Component?

The bootstrapped component is an entry component which loads the DOM during the bootstrapping process. In other words, we can say that entry components are the one, which we are not referencing by type, thus it will be included during the bootstrapping mechanism.

There are mainly two types of entry component declarations can be possible.

1.   Using bootstrapped root component

```
import{ NgModule } from '@angular/core'; import{ CommonModule } from
'@angular/common';
import{ MymoduleRoutingModule } from './mymodule-routing.module';

@NgModule({ declarations: [], imports: [
CommonModule, MymoduleRoutingModule
]
})
Export class MymoduleModule{ }
```

We can specify our component into bootstrap metadata in an app.module file like this.

2.   Using route definition

```
Const myRoutes: Routes = [
{
path:'',
component: DashboardComponent
}
];
```

Another way to define the entry component is to use it along with the routing configuration like this.

144. What are the Providers?

Providers are used to injecting the token to a dependency via constructor of the component, directives and other classes as well.

Most of the time, Providers are used with the Services where we inject service anywhere from our Angular application.

• Using providers into the Module file

We can add services to the providers metadata like this. providers: [MydataService]

• Using providers into the Component file

Can also specify the services using component-based providers like this.

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

@Component({ selector: 'app-login',
templateUrl: './login.component.html', styleUrls: ['./login.component.css'], providers: [
MyAuthService ]
})

Export class AppComponent { selectedOption: string = 'option2';
}
```

145. What is Pipe in Angular?

During the web application development, we need to work with data by getting it from the database or another platform, at that we get the data into a specific format and we need to transform it to a suitable format, and for this purpose, we can use pipes in Angular.

The pipe is a decorator and it is used to transform the value within the template.

It is just a simple class with @Pipe decorator with the class definition and it should implement the PipeTransform interface which accepts the input value and it will return the transformed value.
To create a new pipe, we can use the below command.

146. The defaults file structure of Angular Pipes

When we create a new pipe, a new file will be created and the file structure will be like this.

Mycustompipe.pipe.ts

```
import{ Pipe, PipeTransform } from '@angular/core';

@Pipe({
name:'mycustompipe'
})
Export class MycustompipePipe implements PipeTransform {

transform(value: any, args?: any): any { return null;
}
}
```

We will use the transform () method which is used to get the arguments and values and it will return the converted value to the template to transform the desired data.

147.What is the use of PipeTransform Interface?

When we want to create custom pipe at a time we should implement an interface named PipeTransform which accepts input and returns the transformed value using transform() method.

There are two parameters provided with the transform () method.

- **Value -** A value to be transformed
- **Exponent -** Additional arguments along with the value like size, volume, weight etc.

148.What are built-In Pipes in Angular?

There are different types of in-built pipes available in Angular, which are listed below.

- Date pipes
- Currency pipes
- Uppercase pipes
- Lowercase pipes
- Titlecase pipes
- Async pipes
- Slice pipes

- Decimal pipes
- Json pipes

And many more types of pipe are available to use for data transformation.

149. How to use Pipes?

To use in-built pipes in angular we can use symbol pipe followed by pipe name as
described below.

Syntax:
{{ value | pipe_name }}

Here you can see the simple example, in which you will learn about title case pipe, upper case, andlower
case pipes.

```
<p>{{ 'this is first line' | titlecase }}</p>
<p>{{ 'This is Second Line' | uppercase }}</p>
<p>{{ 'This is Third Line' | lowercase }}</p>
```

And you can see the output with transformed value like this.

150.What are different types of Pipes in Angular?

Mainly there are two types of Pipes are available.

- Pure pipes
- Impure pipes

151.What is a Pure Pipe?

By default, in Angular, pipes are pure, and every pipe created in the angular are
pure by nature.

The pure pipe will be used only when it requires pure changes in terms of the value, and value can be String, Number, and Symbol etc. And other object values like Array, Functions, and others as well.

```
This Is First Line

THIS IS SECOND LINE

this is third line
```

152.Explain Pure Pipe using an Example?

The pure pipe can be created using command **ng generate pipemycustompipe** and below you can find a simple example of the pure pipe.

Mycustompipe.pipe.ts

```
import{ Pipe, PipeTransform } from '@angular/core';

@Pipe({
name:'mycustompipe', pure:true
})
Export class MycustompipePipe implements PipeTransform { transform(value: any, args?:
any): any {
if(!value) {
returnvalue = 'Value Not Found';
}
return value = value;
}

}
```

And below is the HTML file where we are going to use our

custom pipe. App.component.html

```
<p>
{{ 'Pure Pipe Example' | mycustompipe }}
</p>
```

Here in this example, if we don't provide the value from HTML than our default value will be used, you can see the output.

```
Pure Pipe Example
```

As you can see that I have provided value Pure Pipe example so it will return the same value but if we don't
provide any value than default value will take place like this.

```
<p>
{{ '' | mycustompipe }}
</p>
```

Our output now looks like this.

```
Value Not Found
```

153.Explain Impure Pipe using an example?

We know that by default every pipe we create is a pure pipe, but we can also create impure pipe by providing pure to false as describe below.

Impurepipe.pipe.ts

```
import{ Pipe, PipeTransform } from '@angular/core';

@Pipe({ name:'impurepipe', pure:false
})
Export class Impurepipe implements PipeTransform {
transform(value: any, args?: any): any {
if(!value) {
return value = 'Value Not Found';
}
return value = value;
}
}
```

In this example, you can see that using pure option we can set it to true or false, so if we want to set a pipe to impure than value would be false just like this example.

154. What is Pipes chaining?

Sometimes we need to use multiple transformations for the data, at that time chain of pipes is used, using the pipe chaining we can get the combinational transformed value.

We can use multiple in-built pipes or we can use a combination of built-in pipes with a custom pipe, in our example, we are going to transform the date value using uppercase and the date pipe.

In the component, we are going to use the default date.
examdate = new Date('Jan 2018');

And from HTML template, we are going to transform its value using a chain of the pipe. After Date :

```
<b>
{{ examdate | date | uppercase }}
</b>
```

155. What is Parameterized Pipe?

Using Pipes, we can also pass the parameter to transform the value in a specific manner; it can be an expression to achieve the desired output.

If we want to pass additional parameters along with a pipe, then we can use (:) colon symbol as described below in the example.
Date :

```
<b>
{{ examdate | date:'dd:MM:yyyy' }}
</b>
```

156. What is an AsyncPipe in Angular?

We can use the temporary property to hold the content return by observable or promise and later them we bind the same content to the template. But if you want to get rid of this temporary property and bind this content directly to the template then you should use AsyncPipe.

157. What is Routing in Angular?

Routing in Angular allows the user to navigate to the different page from the current page. When we try to enter URL in the browser at a time it navigates to a specific page and performs appropriate functions implemented into the pages.

Routing is one of the crucial tasks for any application, because the application may have a large number of pages to work with.

158. How to compose navigation URL?

While developing web applications using Angular, we should add one tag <base> which is used to compose the navigation URL.

If our Angular application root is app directory than it will look like this in the index.html file.

159. What is RouterModule?

RouterModule in Angular provides different directives and providers to navigate from one page to another page.

And this module makes sure that navigation transforms the URL in the correct manner and also manages different URL transitions.

Soon, we will cover the actual use of RouterModule later on this series of Routing.

160.How to use Routing in Angular?

Angular has its own library package @angular/router it means that routing is not a part of core framework @angular/core package.

In order to use Routing in our Angular application, we need to import RouterModule like this into our root module file.

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

161. What is the Route in Angular?

The route is the Array of different route configuration and contains the various properties which are listed below.

Property	Description
Path	String value which represents route, matcher
pathMatch	Matches the path against the path strategy
Component	The component name which is used to indicate the component type
Matcher	To configure custom path matching
Children	Children are the array of different child routes
loadChildren	Load children are used to loading the child routes as lazily loaded routes

These are the primary properties and also other properties are supported like data, canActivate, resolve, canLoad, canActivateChild, canDeactivates.

And by using these different properties, we can perform many operations like loading the child route, matching the different route paths; specify the route path for specific component etc.

162. What are different types of methods in RouterModule?

There are mainly two static methods are available which are listed below.

1.   **forRoot():** It performs initial navigation and contains the list of configured router providers and directives. You can find the simple example using forRoot below. RouterModule.forRoot([
{ path:'', redirectTo:'dashboard', pathMatch:'full' },

{ path:'dashboard', component:DashboardComponent },
{ path:'about', component:AboutComponent },

{ path:'contact', component:ContactComponent }
])

2.   **forChild():** forChild is used to create the module with all of the router directive and provider registered routes.

163. What isRouterLink?

RouterLink in Angular is a Directive and used to provide a link to the specific route.

We use routerLink to transfer the route from one component to another by identifying the route value which is configured in the router module.

The basic syntax of routerLink will be like this.

```
<a [routerLink]="['/home']"> Home Page</a>
```

164.What are the different properties of a RouterLink?

There is a number of properties are available to use with RouterLink which are listed below.

- routerLink
- queryParams
- fragment
- preserveFragment
- replaceUrl
- preserveQueryParams
- urlTree
- skipLocationChange
- queryParamsHandling

165.What are various Router events?

When the navigation process started, at that time different event will be trigger at a specific point of time using the property Router.events.

There are different events are available which are listed below.

- NavigationStart
- RouteConfigLoadStart
- RouteConfigLoadEnd
- RoutesRecognized
- GuardsCheckStart
- ChildActivationStart
- ActivationStart
- GuardsCheckEnd
- ResolveStart
- ResolveEnd

- ChildActivationEnd
- ActivationEnd
- NavigationEnd
- NavigationCancel
- NavigationError
- Scroll

166. What is RouterOutlet?

RouterOutlet is a directive provided by Angular which is used to load the different components based on the router state.

Whenever the user clicks on the link, at a time the activated component will be rendered and added to HTML DOM inside the router-outlet directive.

To load the different components, we need to use a directive like below into our HTML template.

```
<router-outlet></router-outlet>
```

Here it is the simple example, to load different components inside the router-outlet.

```
<div class="container">
<a routerLinkActive="active" routerLink="/page1">Page 1</a> |
<a routerLinkActive="active" routerLink="/page2">Page 2</a> |
<a routerLinkActive="active" routerLink="/page3">Page 3</a>
<router-outlet></router-outlet>
</div>
```

167. How to use routerLink?

routerLink property is used to generate the link to transfer the user from one page to

another. For that you can see the simple example like below.

```
<a [routerLink]="['/home']" fragment="homelink"> Go To Home Page</a>
```

And it will generate the link just like the normal URL for navigating to the home page.

localhost:4200/home

168.How to use RouterLinkActive?

RouterLinkActive in Angular is used to provide a CSS class whenever any link will be clicked by the user and using this attribute, we can change styles of that link.

For example, we have two different links, one is Dashboard and another is Aboutus, now I'm going to create a class and providing it to the link as explained below.

```html
<div class="container">
<a routerLinkActive="active" routerLink="/dashboard" routerLinkActive="demo-active">Dashboard</a> |

<a routerLinkActive="active" routerLink="/about" routerLinkActive="demo-active">About Us</a> |
<router-outlet></router-outlet>
</div>
```

In the above example, I have used attribute routerLinkActive with the class name demo- active and the style is given below.

.demo-active { color: red;
}

Now when we run our demo, we can see that activated link's color is changed now from blue to red like this.

Dashboard | About Us |

169.What are the different Routing Strategies in Angular?

n Angular, there are two primary routing strategy can be implemented which are.

a. Path Location Strategy

By default, in Angular, Path location strategy is used to implement routing using HTML 5 pushState API.

b. Hash Location Strategy

To enable Hash-based routing, we need to use Hash location strategy when Hash (#) will be appended to the URL.

By using these two different routing strategies, we can implement routing in our Angular app because every time whenever URL will be changed at that time our application sends the request to the server and will render appropriate data based on the component loaded.

170.Explain Path location routing strategy in Angular?

Every time when we create an Angular application, by default Path location strategy will be implemented so we don't need to put extra efforts from our side.

It used HTML5 history API called pushState which allow us to push the different URL than the current URL string, let's see a simple example.

Our angular application working on URL:

localhost:4200/Employees

And we want to change it to somewhat like this. *localhost:4200/Recruiter/Messages/12345*

So that by using pushState, we can achieve the above example easily and can modify different URL string based on the requirements.

And one of the most important things to keep in mind that browser would not request for getting data again and again and when we click again on the back button, we will be navigated to the previous page.

When we open the file index.html from the src directory, you can find the below line.

<base href="/">

The above statement will identify that Path location strategy was implemented into our application.

171.Explain Hash location routing strategy in Angular?

This is the routing strategy which is used to generate a hash (#) based URLs.

We know that Path location strategy is implemented by default in the Angular application, but if we want to implement Hash location strategy than we need to follow one more step as described below.

While implementing hash-based routing strategy, we need to pass Boolean useHash to true, and after doing this we will be able to see hash (#) into the URL.
Export const routing: ModuleWithProviders =
RouterModule.forRoot(appRoutes, { useHash:true

});

Now run your angular app, and you can see the output
like this. https://localhost:4200/#/dashboard

One of the most meaningful about this strategy is that it never sent the remaining part after # from the URL to the server, let's see a simple example.

Assume that we have URL like **localhost: 4200/Employees/#/marketing/165** than a request to the server will be only to **localhost: 4200/Employees**.

In short, hash location strategy enables us to implement routing when we don't want to confuse the server by sending hash fragment and it is the ideal solution for the URL which may contain bookmarks related terms so that we can directly be redirected to the specific part of the page.

172.How to generate a routing module along with the normal module?

When we create a new Angular application, at that time routing will not be implemented by default, we need to configure routing manually but we can generate routing module automatically using CLI command ng generate, let's see that how we can do this.

ng generate module <module_name> --routing

As you can see that I have used –routing along with ng command which is used to generate a default routing module and includes @angular.router package inside the newly created module.

173.Explain the types of Route loading

In Angular, basically, there are three types can be used for Route loading which is listed below.

- Pre-Loading
- Eager Loading
- Lazy Loading

174. What is Pre-Loading?

Preloading is a completely different thing than lazy loading and eager loading, preloading means to load the module after any eager module will be loaded, and after eager module loaded at startup, the router looks for the modules which are unloaded and it can be preloaded.

Let's understand by example, assume that we have three different module Home, About and Contact.

Now Home is the first page when our app will be executed, so remaining two modules will remain unloaded, but at some point of time user may want to contact about the business owner to contact them using contact page so that contact page should be ready to load after the Home page.

We can say that the Homepage will be loaded first, then after contact page should be preloaded afterward, which is called pre-loading.

175.What is Eager Loading?

We have seen an example that how preloading works, but there is the difference between both of them is Eagerly loaded module will be executed at a time of application startup.

By default, all the modules in our application are eagerly loaded it means every module will be loaded in the beginning like we have three modules Home, about and Contact than every module will be loaded which is completely opposite to the lazy loading approach.

For example, we have three components which are eager loaded.

// app-routing.module.ts

```
import{ NgModule } from '@angular/core';
import{ Routes, RouterModule } from '@angular/router'; import{ HomeComponent } from
'./home/home.component'; import{ AboutComponent } from './about/about.component';
import{ ContactComponent } from './contact/contact.component';

const routes: Routes = [
{
path:'', redirectTo:'home', pathMatch:'full',
},
{
path:'home', component:HomeComponent
},
{
path:'about', component:AboutComponent
},
{
path:'contact', component:ContactComponent
},
];

@NgModule({
imports: [RouterModule.forRoot(routes)], exports: [RouterModule]
})
Export class AppRoutingModule{ }
```

We have three different components, Home, About and Contact, so whenever application loaded at that time all the modules will be loaded at application startup.

176. What is Lazy Loading?

We are loading every module at once, but do you know that the performance of the application may decrease, for that we need to load only necessary modules which required for application startup.

When the user navigates to a specific route and the component and module are loaded for the single page is called lazily loaded modules and it will be directly reflected the app bundle size.

Here in this question, we are talking about Lazy loading, in which the current screen will only be loaded and rest of the screen will not be loaded at application startup, we can also call it On-Demand loading.

Let's see one simple example of a routing module where we are loading the lazily loaded module, for that let's assume we have three different pages Home, About and Contact pages.

```
import{ NgModule } from '@angular/core';
import{ Routes, RouterModule } from '@angular/router'; import{ HomeComponent } from
'./home/home.component';
const routes: Routes = [
{
path:'', redirectTo: 'home', pathMatch: 'full',
},
{
path:'home',
component: HomeComponen
},
{
path:'about', loadChildren:'./about/index.module#AboutModule'
},
{
path:'contact', loadChildren:'./contact/index.module#ContactModule'
},
];
@NgModule({
imports: [RouterModule.forRoot(routes)], exports: [RouterModule]
})
Export class AppRoutingModule{ }
```

Consider the above example where we have three different components, but we
have one component
HomeComponent which is eagerly loaded when we start our app.

And rest of the components are lazy loaded and their respective module file will be
loaded whenever users click on their links, it will affect the size of the bundle as
well as a performance by reducing initial loading modules.

177. What is Router State?

After the end of each successful navigation life cycle, the router builds a tree of ActivatedRoute
objects that make up the current state of the router. With the help of 'routerState' property you can
access state of routes at anywhere in the application.

178. What is Wildcard route?

Add a wildcard route to intercept invalid URLs and handle them gracefully. A wildcard route has a
path consisting of two asterisks. It matches every URL. The router will select this route if it can't
match a route earlier in the configuration. A wildcard route can navigate to a custom "404 Not
Found" component or redirect to an existing route.The router selects the route with a first match
wins strategy. Wildcard routes are the least specific routes in the route configuration. Be sure it is
the last route in the configuration.

```
{ path: '**', component: PageNotFoundComponent }
```

179. What is pathMatch property in routing?

pathMatch is a property which tell the router that how to match a URL to the path of a route. You can set pathMatch property value as a "full" or as a "prefix".

The **pathMatch:'full'** results in a route hit when the remaining, unmatched segments of the URL match ''.

The **pathMatch:'prefix'** which tells the router to match the redirect route when the remaining URL begins with the redirect route's prefix path.

```
const appRoutes: Routes = [
  { path: 'home', component: HomeListComponent },
  { path: 'test', component: TestListComponent },
  { path: '',   redirectTo: '/test', pathMatch: 'full' },
  { path: '**', component: PageNotFoundComponent }
];
```

180. Difference between [routerLink] and routerLink?

Whenever you want to pass url as a dynamic then we use routeLink in square bracket as a property binding, like below.

```
<a [routerLink]="yourVariable"></a>
```

So this variable (yourVariable) could be defined inside your class and it should have a value like below:

```
export class myComponent {
  public myVariable = "/home";
  }
```

However, when we use without bracket you are passing string only and you can't change it, it's static (hard coded).

```
<a routerLink="/home"></a>
```

181. What is Angular Form?

During developing the applications, we need to handle the different inputs and files like handling user login, registration; getting data for a different purpose and other forms are as well.

At that time Angular forms are used to handle the user inputs. For that there are two

different approaches are provided which are listed below.

1. Reactive Forms
2. Template-driven Forms

By using these two approaches, we can create forms, implement various input validations, and track the input changes.

182. What are the Building blocks of an Angular Form?

While working with the Forms in Angular, we are using different form controls to get the user inputs, based on that we have some basic building blocks which are listed below.

- **FormGroup**

It has the collection of all FormControls used specifically for a single form.

- **FormControl**

FormControlis used to manage the form controls value and also maintain the status of the validation.

- **FormArray**

Manages the status and value for Array of the form controls

- **ControlValueAccessor**

Acts as a mediator between FormControl and our different native DOM elements.

183. What is Reactive Form in Angular?

One of the widely used forms approach which is called Reactive forms, in which structure of the form will be implemented in the code I mean to say inside the component itself.

Reactive forms approach is robust, scalable, and one of the best parts is that it is reusable and it is also called Model-driven forms.

In order to use Reactive forms, we need to import module as described below.

```
Import { ReactiveFormsModule } from '@angular/forms';
//And then we need to add this module into import array like this.
imports: [ BrowserModule, CommonModule, ReactiveFormsModule
],
```

184. What is Template-driven Form in Angular?

This is the simplest approach to use forms in Angular, and it will be useful if you have simple forms which can be managed within your template but it cannot be scalable as compared to Reactive forms.

In Template-driven forms, Angular will create the models i.e. FormGroups and FormsControl sand then it uses ngModel to enable template-driven forms approach.

To use template-driven approach, we need to import forms modle like this into module

```
import{ FormsModule } from '@angular/forms';
//And also need to add inside imports array like this.
```

185. What are the differences between Reactive form and Template-driven form in Angular?

The differences between Reactive Form and Template-driven Form are given below:

Reactive Forms	Template-driven Forms
Supports dynamic form and structure	Supports static form approach
Form structure will be implemented in Typescript or else you can say in the code itself	Form structure will be implemented in the HTML template
Form values passed to the code using forms value property	It allows one-way and two-way data binding to pass the forms value
Behavior is Synchronous	Behavior is Asynchronous

186.What is FormControl in Angular and how to use it?

To create a reactive form, we use FormGroup, FormControl, and FormArray with the ReactiveFormsModule.

FormControl is generally used to track the form values and also helps to validate the different input elements. To use FormControl, we need to import it into the module file like this.

```
Import { FormsModule, ReactiveFormsModule } from '@angular/forms';
```

Also, we need to add the same inside the imports array.

```
imports: [ BrowserModule, FormsModule, ReactiveFormsModule],
```

To use FormControl with the template, you can see the simple example like this.

```html
<table>
<tr>
<td>Enter Your Name :</td>
<td><input type="text" [formControl]="fullName"/></td>
</tr>
</table>
```

And inside the component file, we need to import FormControl and create new instance as described below.

```
import{ Component } from '@angular/core';
import{ FormControl, Validators } from '@angular/forms';
```

```
@Component({ selector: 'my-app',
templateUrl: './app.component.html', styleUrls: [ './app.component.css' ]
})
Export class AppComponent {
fullName = new FormControl('', Validators.required);
}
```

187.What is FormGroup and how to use it in Angular?

FormGroup in Angular is a collection of different FormControls and it is used to manage the value of different inputs and implement validations as well.

In other words, we can say that FormGroup is a group of different elements which are the instance of FormControl.

To use FormGroup, we need to import it into the component like

this. import{ FormGroup, FormControl, Validators } from

'@angular/forms';

And to use FormGroup with the different FormControls, here it is the simple structure which you can follow.

```
demoForm = new FormGroup({
firstName :new FormControl('Manav', Validators.maxLength(10)), lastName:new
FormControl('Pandya', Validators.maxLength(10)),
});
```

Here in this example, we have two different form controls firstName and lastName which in the textbox to get the values from the end user.

Now we will use formControlName which is used to sync the input value with the FormControl and vice versa.

```
<form [formGroup]="demoForm" (ngSubmit)="submit()">
<table>
<tr>
<td>First Name :</td>
<td><input type="text" formControlName="firstName"/></td>
</tr>
<tr>
<td>Last Name :</td>
<td><input type="text" formControlName="lastName"/></td>
</tr>
<tr>
<td><input type="submit"/></td>
</tr>
```

89

After running the above example, we will get the output like this.

188. Explain Form validations in Angular?

Form validation is one of the crucial parts of any web application where we need to validate the inputs against malicious data or we can say security vulnerability, and other scenarios like data is complete and accurate or not and also data is same as we have expected etc.

For all of these situations, we need to perform various form validations, in Angular, there are two ways to perform validation which are listed below.

- **In-built validation**

We can use in-built validations to validate different inputs into our form

- **Custom validation**

Sometimes we need to implement a validation specific to a condition where in-built validations cannot be used at that time, we can use custom validation functions.

189. What are various built-in validators in Angular?

Angular provides a set of inbuilt validators which we can use directly based on the situations and these validators are.

- required
- min
- max
- requiredTrue
- minLength

- maxLength
- email
- compose
- composeAsync
- nullValidator

190.How to use built-in form validators?

We have seen a different list of inbuilt validators provided in Angular, now next question may raise like how to use these validators in our application.

Let's see one simple example of minLength and maxLength validator. First, we need to import validators like this.

```
import{ Validators} from '@angular/forms';
Then we can use different validators using our form controls as explained below.
this.demoForm = new FormGroup({
'fullName':new FormControl('', Validators.minLength(4)),
});
```

Here in this example, as you can see that we have used Validators.minLength with the value 4 so less than 4 characters won't be allowed with the full name form control.

191.   How to create custom validation in Angular?

Sometimes it may possible that these inbuilt validators would not be helpful to accomplish our validation requirements, so for that, we can also create our custom validations.

Custom validation is as it's nothing more than regular function, let's see one simple example in which salary should be more than 10000 and for that implement function like this.

```
function isSufficientSalary (input: FormControl) {
const isSufficientSalary = input.value>10000;
return isSufficientSalary ?null : { isSufficient :true };
}
```

Here in this example, we have implemented an isSufficientSalary function which takes an input value and if the value is not more than expected value than it returns false, and to use this function inside component like this.

```
this.demoForm = this.builder.group({
salary:new FormControl('', [ Validators.required, isSufficientSalary ]),
});
```

As you can see that we have used two different validators, first is required validator and another is our custom validator, find the template code below.

```
<form [formGroup]="demoForm" (ngSubmit)="submit()">
<table>
<tr>
<td>
<label for="salary">Salary</label>
</td>
<td>
<input type="text" name="salary" id="salary" [formControlName]="'salary'"><br/>
</td>
<td>
<div [hidden]="!demoForm.get('salary').hasError('isSufficient')"> Salary is not sufficient
</div>
</td>
</tr>
<tr>
<td><input [disabled]="!demoForm.valid" type="submit"></td>
</tr>
</table>
</form>
```

And when we run this demo, you can see the output like this.

I am going to use salary 10000 which is not valid and see the error message.

When I'm going to provide the expected value then it will be accepted.

192.How get the submitted Reactive form values?

To get the values submitted by the user, we can use the value property of a FormGroup just like below. For that we need to define click event along with form like that.

```
<form [formGroup]="demoForm" (ngSubmit)="submit()">
</form>
```

And to get the all submitted values, we can use value property like this.

```
submit()
{
console.log(this.demoForm.value);
}
```

193. How to reset the form in Angular?

Sometimes we may have situations in which we need to reset the form, so for that in Angular using reactive form approach, we can do this using reset () method as described below.

```
submit()
{
console.log(this.demoForm.value);
// Reset the complete form this.demoForm.reset();
}
```

This is how we can simply reset the complete reactive form using the reset () method.

194. What is Dependency Injection in Angular?

You may have heard this term called Dependency Injection so many times, but many of them don't know the actual importance and meaning about that.

Dependency Injection is an application design pattern which is used to achieve efficiency and modularity by creating objects which depend on another different object.

In AngujarJs 1.x, we were using string tokens to get different dependencies, but in the Angular newer version, we can use type definition to assign providers like below example.

```
constructor(private service: MyDataService) { }
```

Here in this constructor, we are going to inject MyDataService, so a whenever instance of the component will be created at that time it determines that which service and other dependencies needed for a component by just looking the parameters of the constructor.

In short, it shows that current component completely depends on MyDataService, and after instance of the component will be created than requested service will be resolved and returned, and at last constructor with the service argument will be called.

195. What is Injector in Angular?

The injector in Angular is used to inject the service instance into the class, and by using injector, we can register our service to the module, component and will allow service to be used by all the components in the application.

The injector is just a simple class and it's also called singleton. Every angular application contains single injector, but we can create a tree of injectors as well.

196. What are many ways to implement Dependency Injection in Angular?

In order to implement DI, we should have one provider of service which we are going to use, it can be anything like register the provider into modules or component.

Below are the ways to implement DI in Angular.

- Using @Injectable() in the service
- Using @NgModules() in the module
- Using Providers in the component

197. How to use @Injectable () to implement DI in Services?

Whenever we create any service file, by default it will be registered with the root injector using @Injectable () decorator, below is the simple example.

```
@Injectable({ providedIn: 'root',})
```

By using this mechanism, we are allowing our service methods to be accessible throughout the application scope.

198. How to use @Injectable () to implement Dependency Injection in Modules?

This is another approach to register a provider to module level it means that we are allowing that our service will be accessible to all of the components relevant to the current module In which we are registering our service.

You can get the idea that how to do that by observing the below example.

```
@NgModule({ providers: [
...
MyDataService, // Registering service
RoutingModule,
...,
...
],
})
```

Here in the above example, we have imported our service file called MyDataService and we are providing it to the array of providers so that it can be used by any component using the current module file.

199. How to use providers to implement Dependency Injection in the component?

We can also register our service using @Component decorator into the component, so that every time, whenever a new instance of the component will be created at that time we will get the new instance of the service.

```
@Component({
selector: 'my-app',
templateUrl: './my-app.component.html',
providers: [MyDataService ]
})
```

In the component file, we can register our service by providing a service name to the array of providers as explained below.

200. What is Dependency Injection token?

When we work with Dependency Injection in Angular, every injector maintains the internal token which is used to map the dependency.

And the token will be used as a map, in other words, we can say that the instance is the dependency value and the class type will act as a lookup key, you will get more idea by observing below example.

```
dataService : MyDataService;
```

Here in this line, MyDataService is a type as token and dataService as a value, the same way we can use it inside the constructor as a parameter to initialize the instance like this.

```
constructor(myDataService : MyDataService){ }
```

201. What are Observables in Angular?

Observables is a way to populate the data from the different external resources asynchronously.

Observable replaces the promises in most of the cases like Http, and the major difference between the Promises and Observable is that Observable can be used to observe the stream is different events.

Observable is declarative it means that when we define a function for publishing but it won't be executed until any consumer subscribes it.

Due to asynchronous behavior of Observable, it is used extensively along with Angular because it is faster than Promise and also can be canceled at any time.

202. What is RxJS?

RxJS stands for Reactive Extensions for JavaScript, and it is a popular library which is used to work with the asynchronous data streams.

While working large and complex application, RxJS can help developers to represent multiple asynchronous stream data and can subscribe to the event streams using the Observer.

When an event occurs, at that time Observable notifies the subscribed observer instance.

In order to use RxJS into Angular application, there is an npm package and by installing package we can import a library like this.

import { interval } from 'rxjs';

Interval is just a function into RxJS library, but it has a different set of function which we can use into our application.

203. List down the different functions provided by the RxJS?

RxJS has a collection of different functions, and few of them are listed below.

1. Conditional
- iif()
- defaultIfEmpty()
- every()

2. Combinations
- concat()
- concatAll()
- startWith()
- race()
- merge()
- forkJoin()
- zip()

3. Filtering
- filter()
- first()
- last()
- skip()
- sample()
- debounce()
- debounceTime()

4. **Transformations**

- map()
- partition()
- buffer()
- bufferCount()
- window()
- windowCount()
- mapto()
- groupBy()

5. **Utility**

- let()
- delay()
- timeout()
- repeat()
- dematerialize()

6. **Error handling**

- retry()
- catch/catchError()
- retryWhen()

204.Example using RxJS operators with Angular

We have seen different functions in RxJS, now next question may arise how to use any functions or operators in Angular app, below is the simple example using RxJS map operator.

```
import { Component } from '@angular/core'; import { from } from 'rxjs';
import { map } from 'rxjs/operators';

@Component({ selector: 'my-app',
templateUrl: './app.component.html',
})
export class AppComponent  {
data = from([1, 2, 3, 4, 5]); constructor(){
// Iterate the items
const example = this.data.pipe(map(value => value));
// subscribe the source
const subscribe = example.subscribe(val => console.log(val));
}
}
```

As you can see into the above example that I have imported map operator from the package rxjs/operators, and the map () operator iterate the items of the array.

Then we are subscribing the same list using subscribe () function, when we open the

console, we can see the output like this.
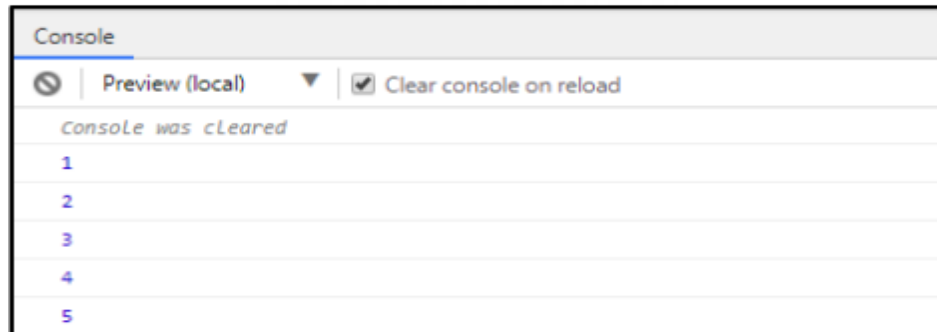
Same way, we can also map the single property from the array like below example.

Here in this example, we have two different properties for user details like Id and Name of the user.

After mapping the data using the map operator, we will be able to iterate the array with the single property, and output look like this.
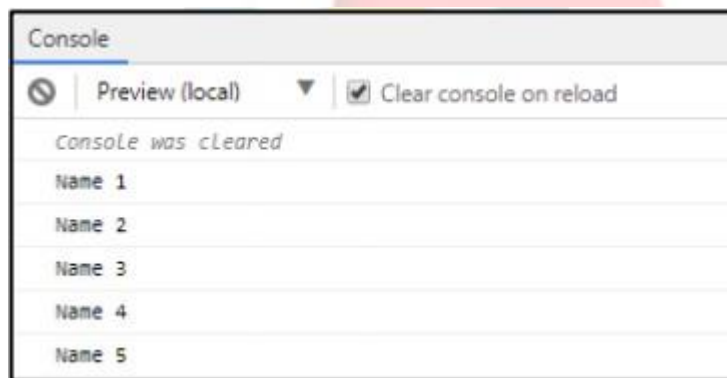
205. What is a Subscription?

Subscription is a kind of disposable resources such as execution of the Observables.

While working with the Observables, the subscription has one method called unsubscribe () which is used to release the different resources held by the subscription and it won't take arguments.

For example, we care calling the GET api for getting the employee details using the services.

```
import { Component } from '@angular/core'; import { Subscription } from 'rxjs';
import { MyDataService } from './mydataservice.service';

@Component({
selector: 'my-app',
templateUrl: './app.component.html', styleUrls: ['./app.component.css']
})
export class AppComponent {
```

```
employee: string; subscription: Subscription;
constructor(private service: MyDataService) {
 }
ngOnInit() {
this.subscription = this.service.getEmployees()
.subscribe((employee: string) => this.employee = employee);
}

ngOnDestroy(): void { this.subscription.unsubscribe();
}
}
```

In this example, we are getting the data using service method and subscribed it. But it is custom observable so that we can manually unsubscribe the Observable when our component gets destroyed.

The best place to unsubscribe the subscription is ngOnDestroy lifecycle hooks because we need to destroy the resources which are held by the subscribers.

206. How to unsubscribe from Observables using Async pipe?

We have seen one example that how to unsubscribe from Observable using life cycle hooks ngOnDestroy(), but we can also do that using Async pipe.

```
import { Component } from '@angular/core'; import { Subscription, Observable } from 'rxjs';
import { MyDataService } from './mydataservice.service';

@Component({
selector: 'my-app',
templateUrl: './app.component.html', styleUrls: ['./app.component.css']
})
export class AppComponent {

employee: string;
dataSubscription: Observable<string>;
constructor(private service: MyDataService) { } ngOnInit() {
this.dataSubscription = this.service.getEmployees()
.subscribe((employee: string) => this.employee = employee);
}

}
```

After subscribing, now we need to unsubscribe it using one of the ways, but in this example, we will do it using an async pipe like this.

```
<!-- app.component.html -->
<div>
{{ dataSubscription | async }}
</div>
```

As you can see that we have used pipe symbol along with pipe name async pipe which is used to unsubscribe from the RxJS Observables.

This is one of the prior choices to unsubscribe from the Observables because it runs everything in the background automatically.

207. Error handling using Observables

Working with Observable is a good choice, but when it comes to handling the error, we need to manage unhandled errors.

Using try/catch is not enough to process the Observables, for that we can use errors callback on the Observers like below example.

```typescript
import { Component } from '@angular/core'; import { Subscription, Observable } from 'rxjs';
import { MyDataService } from './mydataservice.service';

@Component({
selector: 'my-app',
templateUrl: './app.component.html', styleUrls: ['./app.component.css']
})
export class AppComponent {

employee: string;
dataSubscription: Observable<string>;
constructor(private service: MyDataService) {
 }
ngOnInit() {
this.dataSubscription = this.service.getEmployees()
.subscribe((employee: string) => this.employee = employee, err => console.log('Something
Went Wrong :', err),
() => console.log('Success')
);
}

}
```

In this example, we have used err callback which is used to get the complete error information and the same way we can get the completion handler function so that we get to know about the success confirmation.

208. Explain catchError and Finalize operators?

The catchError and Finalize are the operators in RxJS and it is used to handle the errors in Observable sequence. Let's see the simple example using catchError.

```
// Throw custom error
const error = throwError('I AM ERROR');

// handling error using catchError
const example = error.pipe(catchError(val => of(`Error : ${val}`)));

const subscribe = example.subscribe(value => console.log(value));
```
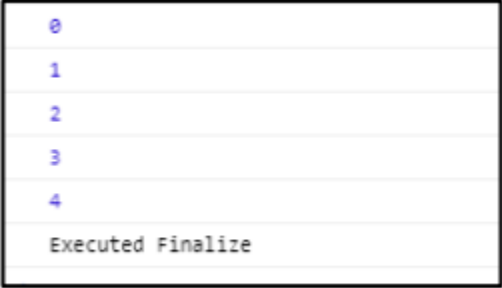
We have thrown a custom error using the throwError function and the same error will be handled by the catchError operator and will provide the error details.

Finalize operator is the same as finally block that we have used along with the try/catch which is used to execute the callback function when the observable

```
// print the value each 1 second const source = interval(1000);
const data = source.pipe( take(5), // First 5 values
finalize(() => console.log('Executed Finalize')) // when execution completed
)
const subscribe = data.subscribe(val => console.log(val));
```

executed.

In this example, we have used take and finalize operator, the first five values will be printed and at the end using finalize, the completion message will be printed into the console like this.

```
0
1
2
3
4
Executed Finalize
```

209. What is Angular Service?

While developing app we need to change or manipulate data constantly, and in order to work with data, we need to use service, which is used to call the API via HTTP protocol.

Angular Service is a simple function, which allows us to create properties and method to organize the code.To generate services, we can use below command which generates service file along with default service function structure.

ng generate service mydataservice

210. What is the default structure of an Angular Service?

When we create service using ng command, it will create service file like below.

Mydataservice.service.ts

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

@Injectable({ providedIn:'root'
})

Export class MydataserviceService { constructor() { }
}
```

211. Why we should use Angular Services?

Services in Angular are the primary concern for getting or manipulate data from the server. The primary use of the services is to invoke the function from the different files like Component, Directives and so on.

Normally services can be placed separately in order to achieve reusability as well as it will be easy to distribute the Angular application into small chunks. Using dependency injection, we can inject the service into the Component's constructor and can use different functions of service into the whole component.

One of the main advantages is Code Reusability because when we write any function inside the service file then it can be used by the different component so that we don't need to write same methods into multiple files.

212. How to use service in Angular Components?

After creating the service, the next step is to consume the different service function into the different components.To use the services in the component, the first step is to import the service file from the respective folder location like this.

```
import{ EmployeeService } from './services/employee.service';
```

Next step is to inject the service using dependency injection into the Component's constructor like this.

```
constructor(private myService: EmployeeService) {
}
```

And using the myService, we can access the different functions and properties implemented inside the EmployeeService. We can call the service function like this.

```
this.myService.GetEmployees();
```

This is how we can call the different function from the services and can communicate with the server by providing sufficient request data.

213. What is Singleton Service?

Singleton Service in Angular means to provide Service reference to application root, it can be created using two different ways.

· Declared service should be provided in the application root

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

@Injectable({
providedIn:'root'
})
```

```
Export class MydataserviceService { constructor() { }
}
```

Add inside AppModule or another module which indirectly imports into the AppModule

```
import{ NgModule } from '@angular/core'; import{ CommonModule } from
'@angular/common'; import{ AppComponent } from './app.component';
import{ Mydataservice } from './mydataservice.service';

@NgModule({ declarations: [
AppComponent
],
imports: [
CommonModule
],
exports: [], providers: [
ApiService, // Singleton Services
]
})

ExportclassAppModule{ }
```

214. What is HTTPClient in Angular?

While working with any web-application, we always need the data to work with and for that, we need to communicate with the database which resides on a server.

In order to communicate from the browser, it supports two types of API.

- Fetch() API
- XMLHttpRequest

Same way in Angular, we have HTTP API which is based on XMLHttpRequest which is called HttpClient to make a request and response.

Before Angular 5, HTTP Module was used but after that, it was replaced by the HttpClientModule.

To use HttpClientModule in our Angular app, we need to import it like this. import{ HttpClientModule } from '@angular/common/http';

215. How many types of HTTP Requests are supported?

There are total 8 types of methods are supported which are listed below.

- GET Request
- Post Request
- Put Request
- Patch Request
- Delete Request
- Head Request
- Jsonp Request
- Options Request

216. How to inject HttpClientModule in our component?

HttpClient is used to initiate a request to the server using different methods like GET, Post, Put and delete, but we for that we need to inject HttpClientModule into our application module file like this.

```
import{ NgModule } from '@angular/core';
import{ BrowserModule } from '@angular/platform-browser'; import{ HttpClientModule }
from '@angular/common/http';

@NgModule({ imports: [
BrowserModule,
// import HttpClientModule module HttpClientModule,
],
declarations: [ AppComponent,
],

bootstrap: [AppComponent]
})

Export class AppModule{ }
```

217.How to inject HttpClient in our Angular Service?

After importing theHttpClientModule into the root module file, now we will be able to inject HttpClient into our service which looks like this.

```
import{ Injectable } from '@angular/core';
// Importing HttpClient
import{ HttpClient } from '@angular/common/http';

@Injectable()
Export class ConfigService {
// Injecting HttpClient constructor(private http: HttpClient) { }
```

Now using HTTP, we will be able to use different Http request methods like Get, Post, Put, Delete

and other methods as well into our whole service file.

218. How to do error handling in HttpClient?

While sending a request to the server via service, there are two possibilities that either request will succeed or may fail to fulfill the request. Based on the request result, we should know about the response that request was successful or failed to retrieve data and reason may be different like internet connectivity, internal server error, insufficient request data etc. For that, we should handle the error, and below is a simple example where we are going to use an error object.

App.component.ts

```
getEmployees() { this.empService.getEmployeeList()
.subscribe(
(data: employee) =>this.employees = { ...data },
// Returns error information Error=>this.errorDetails = error
);
}
```

In our app component, we have one method to get the list of employees, but it may possible that anything went wrong, then we will get the complete error details along with the error code.

219. What is Interceptor?

The inception of HTTP request and response is one of the major parts of HTTP package and using inceptor we can transform the HTTP request from our angular application to the communication server, and the same way we can also transform the response coming from the server.

If we don't use the interceptors then we need to implement the tasks logic explicitly whenever we send the request to the server.

220. How to create Interceptor in Angular?

In order to use Interceptor, we need to implements HttpInterceptor interface, and then each request needs to be intercepted using intercept () method as described below.

```
import{ Injectable } from '@angular/core';
// Importing interceptor
import{ HttpEvent, HttpInterceptor, HttpHandler, HttpRequest } from
@angular/common/http';
import{ Observable } from 'rxjs';

@Injectable()
Export class NoopInterceptor implements HttpInterceptor {

// Intercept method
intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
return next.handle(req);
}
```

As you can see that we have implements the interface HttpInterceptor which is an

essential part of implementing interceptor, you may be confused about
HttpInterceptor but soon in this series, we are going to cover these topics with an
example.

221. What is HttpInterceptor in Angular?

HttpInterceptor is used to intercept the Http request, basically, it is an interface which is used to use
and implement the intercept method and also will be able to manage the related methods.

While working with the web application, we will send the request the data for data communication,
but before sending a request to the server we need to modify and manipulate each request.

To implement HttpInterceptor interface, we can use simple syntax as explained below.

```
interface HttpInterceptor {
intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>>
}
```

Above code snippet is a simple example of interface HttpInterceptor which uses intercept method in
order to handle the requests.

222. What is the intercept method in HttpInterceptor?

Basically, when we send the request to the server and getting the response from the
server then we can transform the response using intercept () method.

We can use interceptors to attach authentication details automatically to the request header or simple
we can say to attach authorization details with the request header.

Below is the complete example using
intercept () with HttpInterceptor interface.

```
Import { Injectable } from '@angular/core';
import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor
} from '@angular/common/http';
import{ DataService } from './dataService/data.service';
import{ Observable } from 'rxjs/Observable';

@Injectable()
Export class MyInterceptor implements HttpInterceptor { constructor(publicdata: DataService)
{ }
intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

request = request.clone({ setHeaders: {
Authorization:`Bearer${this.auth.getToken()}`}});
```

This is how we can attach authorization header using HttpInterceptor and intercept () to handle the
request manipulation.

223. What is next object in Interceptor?

We have seen the simple example about HttpInterceptor in which we have used next object, which is used to represent next interceptor in a queue to be transformed.

Next object will be used whenever we have the chain of the interceptor and by using next object, we will get the response as Observable.

By using below the line, we can handle the next interceptor in a chain of the interceptor. *next.handle()*

224. How to add an interceptor to the Providers?

Whenever we create the interceptor, at that time we need to add into the provider's array, and by using HTTP_INTERCEPTORS array, we can do this into our module file like this

```
// Importing interceptors
import{ HTTP_INTERCEPTORS } from '@angular/common/http';
// Created interceptor
import{ DataInterceptor } from './data/data.interceptor';

@NgModule({
bootstrap: [AppComponent], imports: [...],
providers: [
{
provide: HTTP_INTERCEPTORS, useClass: DataInterceptor,
}
]
})

Export class AppModule{ }
```

225. What is the use of HttpHeader in Angular?

While sending a request to the server, sometimes we need to pass some extra header for different uses like content type specific to JSON, authorization etc.

For that we can use HttpHeaders to provide header data with the requests, please find below example to understand its simple use.

Import the HttpHeaders like this. Import { HttpHeaders } from '@angular/common/http';

To provide the different header data, we can use HttpHeaders like this.

```
headers: newHttpHeaders({
'Content-Type':'application/json', 'Authorization':'authtoken'
})
```

Here in this example, we have used content-type as JSON and also, we are passing authorization header along with our request.

226. What are different parameters supported in the method request

header?There are different types of parameters supported to use with the HTTPHeaders which are listed below.

- headers
- params
- observe
- withCredentials
- reportProgress
- responseType

227. How to update request header details?

In Angular, there is no direct way to modify the request header but we can modify it explicitly using set () method as described below.

```
httpOptions.headers = httpOptions.headers.set('Authorization', 'my_token');
```

Here I have modified authorization explicitly using set () method of headers.

228. HTTP Services?

HTTP Services is the services for get /post/delete/update the external data using http module. In Angular we need to import the http module for use the http service.

229. What is subscribers?

We are using Subscribers for subscribe the observables.

230. How we can handle the error in Angular?

We can handle the error using try catch in the Angular application also we can handle the error using error section when subscribe the observables.

231. What is the difference between AOT and JIT?

In Angular we have AOT (Ahead of Time) and JIT (Just In Time) Compilation for compile the application. There are few differences between JIT and AOT:

- JIT stands for Just In Time so by the name you can easily understand its compile the while AOT stands for Ahead of Time so according to the name

- According to compilation, JIT loads the application slowly while application more quickly as compared to that of JIT.

- In case of JIT, Template Binding errors will show at the time of showing the application while in case of AOT Template binding errors will show at the

building time.

- In case of because in case of AOT the bundle

232. What is unit testing?

Unit testing is the great feature by which we can easily test the application feature which we have added within component, services etc. In Angular, we have a spec.ts file for each component where we can add the unit testing.

233. What is the need of Karma and jasmine in Angular?

Karma is the task runner and Jasmine is the framework which we are using for unit testing in Angular application.

234. How we can use Karma and Jasmine in Angular for testing?

When we create new application within Angular CLI, the Angular CLI will install the Karma and Jasmine for unit testing within Angular application which we can use using ng test command and execute the test cases.

235. What is the need of ng test?

ng test is the command which we are using for run the unit test cases with the help of karma and Jasmine. When we executed the ng test command in the Angular cli it will run all the test cases according to application and show the success or fail result.

236. What is test bed?

TestBed is the object and powerful unit testing tool which we can import from angular/core/testing within our Angular application as.

```
import {TestBed} from '@angular/core/testing';
```