

Angular Interview Questions & Answers



By Manav Pandya

Technical Author and Front-end Engineer

 **DotNetTricks**

Angular Interview Questions & Answers

All rights reserved. No part of this book can be reproduced or stored in any retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, uploading on server and scanning without the prior written permission of the Dot Net Tricks Innovation Pvt. Ltd.

The author of this book has tried their best to ensure the accuracy of the information described in this book. However, the author cannot guarantee the accuracy of the information contained in this book. The author or Dot Net Tricks Innovation Pvt. Ltd. will not be liable for any damages, incidental or consequential caused directly or indirectly by this book.

Further, readers should be aware that the websites or reference links listed in this book may have changed or disappeared between when this book was written and when it is read.

All other trademarks referred to in this book are the property of their respective owners.

Release History

- Initial Release 1.0.0 - 2nd Jan 2019



About Dot Net Tricks

Dot Net Tricks is founded by Shailendra Chauhan (Microsoft MVP), in Jan 2010. Dot Net Tricks came into existence in form of a blog post over various technologies including .NET, C#, SQL Server, ASP.NET, ASP.NET MVC, JavaScript, Angular, Node.js and Visual Studio etc.

The company which is currently registered by a name of Dot Net Tricks Innovation Pvt. Ltd. came into the shape in 2015. Dot Net Tricks website has an average footfall on the tune of 300k+ per month. The site has become a cornerstone when it comes to getting skilled-up on .NET technologies and we want to gain the same level of trust in other technologies. This is what we are striving for.

We have a very large number of trainees who have received training from our platforms and immediately got placement in some of the reputed firms testifying our claims of providing quality training. The website offers you a variety of free study material in form of articles.

Dot Net Tricks Training Solutions

Dot Net Tricks provide you training in traditional as well as new age technologies, via various formats.

Master Courses (Instructor-led)

For a beginner who needs regular guidance, we have a fully packed Master Courses. They are almost equal to semester courses taught in engineering colleges when it comes to length, breadth of content delivery, the only difference instead of 5-6 months, they take approx. 16-weekend classes (2 months).

The detail about Master courses can be found here: <https://www.dotnettricks.com/instructor-led-courses>

Hands-On Learning (Learn to code)

Hands-On Learning courses give you the confidence to code and equally helpful to work in real-life scenarios. This course is composed of hands-on exercise using IDE or cloud labs so that you can practice each and everything by yourself. You can learn to code at your own pace, time and place.

The detail about Hands-On Learning courses can be found here: <https://www.scholarhat.com>

Skill Bootcamps (Instructor-led)

Professionals who don't have two months' time and want to get skilled up in least possible time due to some new project that their company has to take in very near future, we have designed Skill Bootcamps Concept, where you will get trained on consecutive days in a fast-paced manner, where our full focus is going to be on hands-on delivery of technological exercises.

The detail about Skill Bootcamps can be found here: <https://www.dotnettricks.com/skill-bootcamp>

Self-paced Courses

Self-paced courses give you the liberty to study at your own pace, time and place. We understand everyone has their own comfort zone, some of you can afford to dedicate 2 hours a day, some of you not. Keeping this thing in mind, we created these self-paced courses. While creating these courses we have ensured that quality of courses doesn't get compromise at any parameter, and they also will be able to produce the same results as our other course formats, given the fact you will be able to put your own honest effort.

The detail about Self-paced courses can be found here: <https://www.dotnettricks.com/self-paced-courses>

Corporate Training (Online and Classroom)

Dot Net Tricks having a pool of mentors who help the corporate to enhance their employment skills as per changing technology landscape. Dot Net Tricks offers customized training programs for new hires and experienced employees through online and classroom mode. As a trusted and resourceful training partner, Dot Net Tricks helps the corporate to achieve success with its industry-leading instructional design and customer training initiatives.

The detail about Corporate Training can be found here: <https://www.dotnettricks.com/corporate-training>

Learning Platform

We have a very robust technology platform to answer the needs of all our trainees, no matter which program they enrolled in. We have a very self-intuitive Learning Management System (LMS), which help you in remain focused and keeping an eye over your progress.

We offer two Learning Platforms as given below:

1. Dot Net Tricks: <https://www.dotnettricks.com>
2. Scholar Hat: <https://www.scholarhat.com>

Apart from these, we also provide on-demand Skill bootcamps and personalized project consultation.

Dedication

My mom Kiranben Pandya and my father Prakashbhai Pandya deserve to have their name on the cover as much as I do for all their support made this possible. I would like to say thanks to all my family members, friends, the mentors who supported me either directly or indirectly no matter how tough and easy the situations to achieve my goals.

-Manav Pandya



Introduction

What Where Author Qualification to Write This Book

Manav Pandya is an enthusiast Front-end developer having vast experience, working on different technologies like Angular, React, React-Native, VueJs, TypeScript, NodeJs, MongoDB and other technologies such as Redux, Angular Material, Material-UI, Bootstrap, HTML, CSS, JQuery etc.

What This Book Is

Angular is the most popular framework to create single page application (SPA) as well as Mobile and Desktop based applications. In this book, we are going to learn about the fundamentals of Angular like Component, Modules, Services, Routing, Dependency injection, Deployment, and many other topics.

What You'll Learn

This book is for those who want to learn Angular and also for those who are going to appear for the Angular interview to have a bright future in front-end technologies.

- Angular Version History
- Angular Core concepts
- Angular CLI
- Angular Components
- Angular Built-in Module
- Routing
- Angular Forms and Validations
- RXJS and Observables
- Services
- HTTP Client and REST API
- How to use and create custom pipes
- Dependency Injection in Angular
- Unit Testing
- Angular Deployment

Our best wishes always with you for your learning and growth!

About the Author

Manav Pandya - An Author and ILETS Trainer



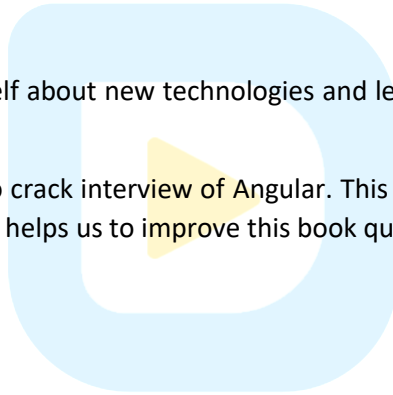
Manav Pandya is working as a Front-end engineer with a leading organization and having vast experience in Front-end technologies. He is very passionate about the JavaScript, Angular, React and Vue.

He is Author, Blogger, Community contributor, Learner by following self-learning approach.

He is keen to learn new technologies. His contributions gave him various recognition including Member of the month September 2018, Member of the month October 2018.

He always tries to keep updated himself about new technologies and learning new skills and share with other in simple manner.

He hopes that this e-book helps you to crack interview of Angular. This is the first edition of this book but not last. Please provide your feedback that helps us to improve this book quality.



How to Contact Us

Although the author of this book has tried to make this book as accurate as it possible but if there is something strikes you as odd, or you find an error in the book please drop a line via e-mail.

The e-mail addresses are listed as follows:

- mentor@dotnettricks.com
- info@dotnettricks.com

We always happy to hear from our readers. Please provide your valuable feedback and comments!

You can follow us on [YouTube](#), [Facebook](#), [Twitter](#), [LinkedIn](#), and [Google Plus](#) or subscribe to [RSS feed](#).



Table of Contents

Angular Interview Questions & Answers	1
Release History	1
About Dot Net Tricks	2
Dot Net Tricks Training Solutions	2
Dedication	4
Introduction	5
About the Author	6
How to Contact Us.....	7
Introducing Angular	15
Q1. What is Angular?	15
Q2. Explain evolution history of Angular	15
Q3. What are the differences between Angular2+ and Angular 1.x?	16
Q4. Why uses Angular?	16
Q5. Important fundamentals of Angular.....	16
Q6. What are the new features of Angular 7?	17
Q7. How to update your app to Angular 7?	17
Q8. How to update your older Angular app to Angular?	17
Q9. What is a new feature called CLI prompt?	18
Q10. How to create Angular Elements?	19
Q11. Explain Angular Elements in Detail?	19
Q12. What's new in Angular Material?	20
Q13. How to update Angular Material to the latest version?.....	20
Q14. Explain Virtual Scrolling in Angular	20
Q15. What is the Drag and Drop feature in Angular 7?	21
Q16. How to implement simple Drag and Drop?	22
Q17. What is Budget bundle in Angular 7?	23
Q18. How to check the current bundle size of an Angular app?	23
Q19. What is Ivy Renderer engine in Angular?	24

Q20.	What version of TypeScript, Angular 7 support?	24
Q21.	What's new in Angular 6?.....	24
Q22.	What's new in Angular 5?.....	25
Q23.	What's new in Angular 4?.....	25
Q24.	What's new in Angular 2?.....	25
Q25.	What are the prerequisites for learning Angular?	26
Q26.	What IDE can be used to develop an Angular application?	26
Q27.	How to set up the Angular Environment?	26
Angular CLI.....		28
Q1.	What is Angular CLI?.....	28
Q2.	What are the features of Angular CLI?	28
Q3.	How to use different Angular CLI Commands?	29
Q4.	What is Angular Console?.....	29
Q5.	What are the advantages of Angular Console?	30
Q6.	How to create a new Angular app?	30
Q7.	How to serve the Angular app?	30
Q8.	How to change default port number other than 4200?.....	30
Q9.	How to change the default protocol other than HTTP?	30
Q10.	How to update your project's dependencies?	31
Q11.	How to generate various types of files using Angular CLI?	31
Q12.	How to update all dependencies of the Angular app?.....	31
Q13.	How to add a new dependency to the project?.....	32
Q14.	How to build an Angular application using CLI?	32
Q15.	How to get project configuration details?.....	32
Q16.	How to test the Angular app?	32
Q17.	How to get the version of Angular CLI?.....	33
Q18.	How to specify the type of the stylesheet using the command?.....	33
Q19.	How to get help from the CLI?	34
Components and Directives		35
Q1.	What is Component in Angular?	35
Q2.	What is the Template Expression?	36

Q3.	How to manage a web page using Angular Component?	36
Q4.	What are the Template Statements?	37
Q5.	What is Data Binding in Angular?	37
Q6.	What is Interpolation?	37
Q7.	What is Property Binding?	38
Q8.	How Does Event binding work in Angular?	38
Q9.	How to implement Two-way data binding in Angular?	39
Q10.	What is Directive in Angular?	40
Q11.	What is Component Directive in Angular?	40
Q12.	What is Structural Directive in Angular?	40
Q13.	What is Attribute Directive in Angular?	41
Q14.	How many types of built-in attribute directives are available?	42
Q15.	What are various Structural directives in Angular?	42
Q16.	What is the use of ngIf directive?	42
Q17.	What is the use of ngFor directive in Angular?	43
Q18.	What is the use of ngSwitch directive in Angular?	44
Q19.	What are various Angular Lifecycle hooks?	45
Q20.	Methods to Share Data between Angular Components	47
Q21.	How to share data using @Input ()	47
Q22.	How to share data using @Output and EventEmitter ()	49
Q23.	How to share data using @ViewChild ()	50
Q24.	How to share data using the Service	51
Q25.	Explain Change Detection in Angular	53
Q26.	Explain Content Projection in Angular	56
Modules.....		58
Q1.	What is Module in Angular?	58
Q2.	How to create a new Module using Angular CLI?	59
Q3.	What different types of Modules are supported by Angular?	59
Q4.	Explain Angular Module basic structure?	59
Q5.	What are the various NgModule metadata properties?	60
Q6.	What are built-in Modules?	60

Q7.	What are the different Feature Modules?	60
Q8.	How to create a Feature Module?	60
Q9.	What is the difference between the JavaScript module and NgModule?	61
Q10.	How to create a Module with routing using CLI?	61
Q11.	What is Entry Component?	62
Q12.	What are the Providers?	62
Pipes		64
Q1.	What is Pipe in Angular?	64
Q2.	The defaults file structure of Angular Pipes	64
Q3.	What is the use of PipeTransform Interface?	65
Q4.	What are built-In Pipes in Angular?	65
Q5.	How to use Pipes?	65
Q6.	What are different types of Pipes in Angular?	66
Q7.	What is a Pure Pipe?	66
Q8.	Explain Pure Pipe using an Example?	66
Q9.	Explain Impure Pipe using an example?	67
Q10.	What is Pipes chaining?	68
Q11.	What is Parameterized Pipe?	68
Routing and Navigation		69
Q1.	What is Routing in Angular?	69
Q2.	How to compose navigation URL?	69
Q3.	What is RouterModule?	69
Q4.	How to use Routing in Angular?	69
Q5.	What is the Route in Angular?	70
Q6.	What are different types of methods in RouterModule?	70
Q7.	What is RouterLink?	70
Q8.	What are the different properties of a RouterLink?	71
Q9.	What are various Router events?	71
Q10.	What is RouterOutlet?	71
Q11.	How to use routerLink?	72
Q12.	How to use RouterLinkActive?	72

Q13.	What are the different Routing Strategies in Angular?	73
Q14.	Explain Path location routing strategy in Angular?	73
Q15.	Explain Hash location routing strategy in Angular?	74
Q16.	How to generate a routing module along with the normal module?	74
Q17.	Explain the types of Route loading	74
Q18.	What is Pre-Loading?	74
Q19.	What is Eager Loading?	75
Q20.	What is Lazy Loading?	76
Angular Forms.....		77
Q1.	What is Angular Form?	77
Q2.	What are the Building blocks of an Angular Form?	77
Q3.	What is Reactive Form in Angular?	77
Q4.	What is Template-driven Form in Angular?	78
Q5.	What are the differences between Reactive form and Template-driven form in Angular?	78
Q6.	What is FormControl in Angular and how to use it?	79
Q7.	What is FormGroup and how to use it in Angular?	80
Q8.	Explain Form validations in Angular?	81
Q9.	What are various built-in validators in Angular?	81
Q10.	How to use built-in form validators?	82
Q11.	How to create custom validation in Angular?	82
Q12.	How get the submitted Reactive form values?	83
Q13.	How to reset the form in Angular?	84
Dependency Injection		85
Q1.	What is Dependency Injection in Angular?	85
Q2.	What are the advantages of using Dependency Injection?	85
Q3.	What is Injector in Angular?	85
Q4.	What are many ways to implement Dependency Injection in Angular?	86
Q5.	How to use @Injectable () to implement DI in Services?	86
Q6.	How to use @Injectable () to implement Dependency Injection in Modules?	86
Q7.	How to use providers to implement Dependency Injection in the component?	87
Q8.	What is Dependency Injection token?	87

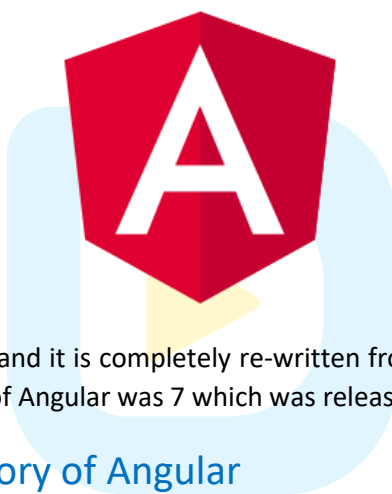
Observables and RxJS.....	88
Q1. What are Observables in Angular?	88
Q2. What is RxJS?	88
Q3. List down the different functions provided by the RxJS?	88
Q4. Example using RxJS operators with Angular.....	90
Q5. What is a Subscription?	91
Q6. How to unsubscribe from Observables using Async pipe?.....	92
Q7. Error handling using Observables.....	93
Q8. Explain catchError and Finalize operators?	94
Angular Services and HTTP.....	95
Q1. What is Angular Service?	95
Q2. What is the default structure of an Angular Service?	95
Q3. Why we should use Angular Services?	95
Q4. How to use service in Angular Components?.....	96
Q5. What is Singleton Service?	96
Q6. What is HttpClient in Angular?	97
Q7. How many types of HTTP Requests are supported?	97
Q8. How to inject HttpClientModule in our component?	98
Q9. How to inject HttpClient in our Angular Service?.....	98
Q10. How to do error handling inHttpClient?.....	99
Q11. What is Interceptor?.....	99
Q12. How to create Interceptor in Angular?	99
Q13. What is HttpInterceptor in Angular?	100
Q14. What is the intercept method in HttpInterceptor?.....	100
Q15. What is next object in Interceptor?.....	101
Q16. How to add an interceptor to the Providers?	101
Q17. What is the use of HttpHeaders in Angular?	101
Q18. What are different parameters supported in the method request header?	102
Q19. How to update request header details?.....	102
Angular Testing	103
Q1. Why we need Testing?	103

Q2.	Setup to test Angular app using Jasmine test	103
Q3.	What is Karma?.....	105
Q4.	What is Test entry file in Angular?	105
Q5.	Testing of Angular Components	106
Q6.	List down different in-built Matchers in Angular	107
Q7.	How to test service in Angular?.....	108
Q8.	How to test Directive in Angular?	108
Q9.	How to test a Pipe in Angular?	111
Q10.	How to do Test Debugging in Angular?	112
Deployment		114
Q1.	What is Deployment?	114
Q2.	How to build an Angular app for Deployment?.....	114
Q3.	Explain the procedure for the Deployment in Angular	114
Q4.	How to enable runtime production mode?.....	115
Q5.	What is the difference between Build and Deployment?	116
Q6.	How to set a different folder for the output of build other than /dist?.....	116
Q7.	How to watch the changes while building an Angular app?	116
References		117

Introducing Angular

Q1. What is Angular?

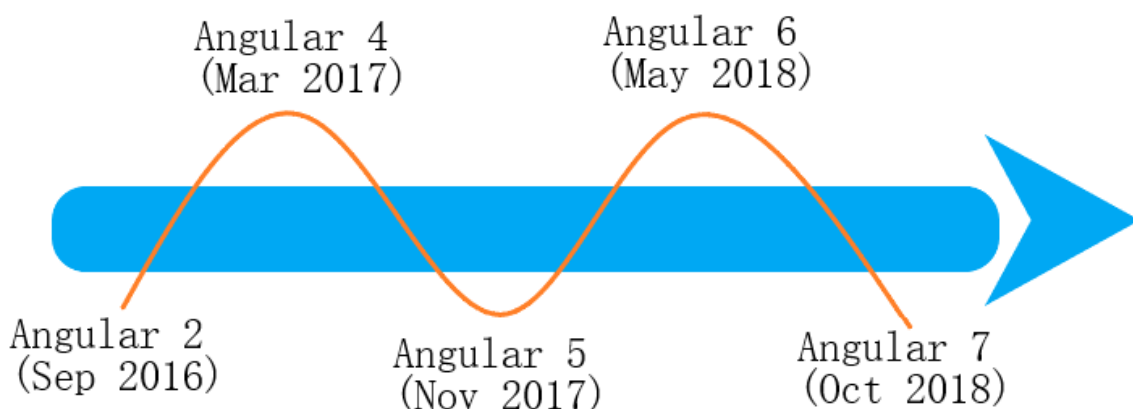
Ans. Angular is a JavaScript-based framework developed by Google. Angular allows developers to build applications for browsers, mobile, and desktop using web technologies like HTML, CSS, and JavaScript.



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.

Q2. Explain evolution history of Angular

Ans. The evolutionary history of Angular is given below –



Q3. What are the differences between Angular2+ and Angular 1.x?

Ans. 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 like ngModel, 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

Q4. Why uses Angular?

Ans. 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

Q5. Important fundamentals of Angular

Ans. 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

Q6. What are the new features of Angular 7?

Ans. 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

Q7. How to update your app to Angular 7?

Ans. 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.

```
ng update @angular/core @angular/cli
```

Q8. How to update your older Angular app to Angular?

Ans. 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 [this](#) official guide to update your old angular app to a newer version and it will look like this.

Select the options matching your project:

Angular Version

6.1 7.0

App Complexity

Basic Medium Advanced

Other Dependencies

☐ I use ngUpgrade for using AngularJS and Angular at the same time

☐ I use Angular Material

Package Manager

npm yarn

Show me how to update!

Q9. What is a new feature called CLI prompt?

Ans. We know that it is difficult to keep in mind all those npm commands, but sometimes we may forget 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 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.

Q10. How to create Angular Elements?

Ans. 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.

Q11. Explain Angular Elements in Detail?

Ans. 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) {
    const el = 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 customElements.define() method by providing element name.

Q12. What's new in Angular Material?

Ans. 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

Q13. How to update Angular Material to the latest version?

Ans. To work with latest Materialized components, we need to update our latest angular material package using below command.

```
ng update @angular/material
```

Q14. Explain Virtual Scrolling in Angular

Ans. 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}`);
}
```

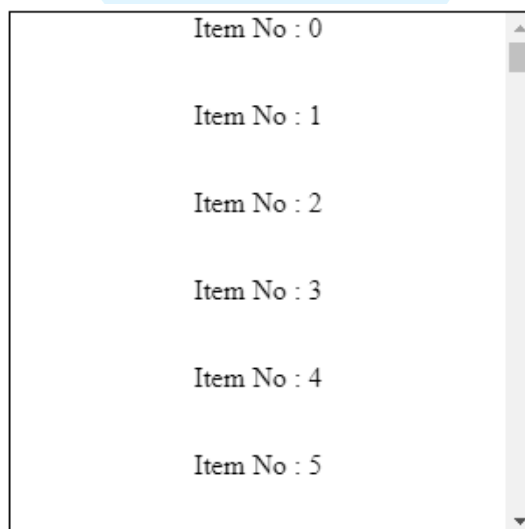
Do not forgot to add some stylesheet into app.component.css.

```
/* 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.



Q15. What is the Drag and Drop feature in Angular 7?

Ans. 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

Q16. How to implement simple Drag and Drop?

Ans. 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

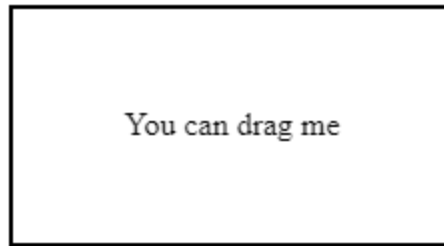
import{ NgModule } from '@angular/core';
import{ BrowserModule } from '@angular/platform-browser';
import{ FormsModule } from '@angular/forms';
import{ AppComponent } from './app.component';
import{ BrowserAnimationsModule } from '@angular/platform-browser/animations';
// Imported DragDropModule from 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 use cdkDrag 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.



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

Q17. What is Budget bundle in Angular 7?

Ans. 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.

Q18. How to check the current bundle size of an Angular app?

Ans. 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.

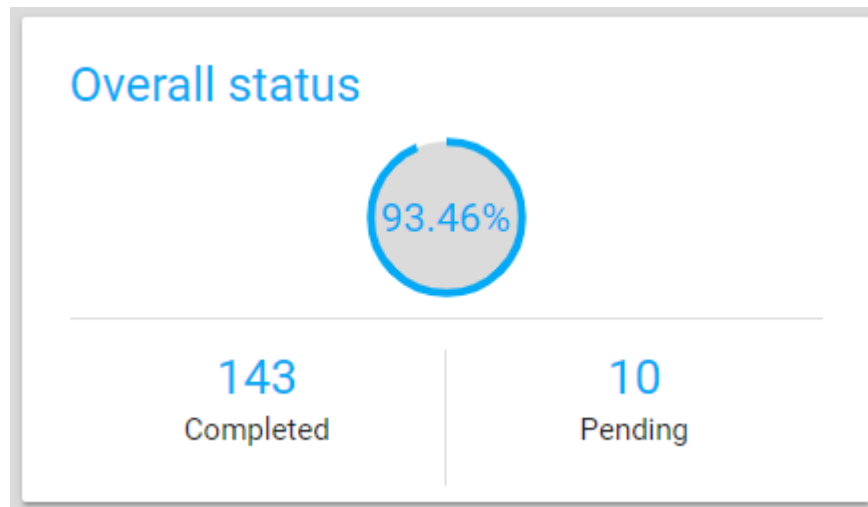
```
Time: 24086ms  
chunk {main} main.js, main.js.map (main) 4.12 MB [initial] [rendered]  
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 223 kB [initial] [rendered]  
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.08 kB [entry] [rendered]  
chunk {styles} styles.js, styles.js.map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


Q19. What is Ivy Renderer engine in Angular?

Ans. 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.



We can get the complete setup and readme guide by following the link [here](#).

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.

Q20. What version of TypeScript, Angular 7 support?

Ans. 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.

Q21. What's new in Angular 6?

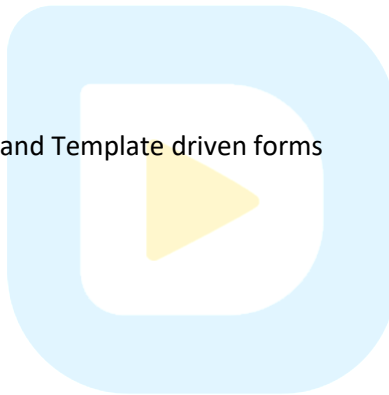
Ans. Below are the new features as well as breaking changes.

- 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

Q22. What's new in Angular 5?

Ans. 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



Q23. What's new in Angular 4?

Ans. 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

Q24. What's new in Angular 2?

Ans. Below are the new features as well as breaking changes.

- Component based architecture

- Views
- Directives
- Modules
- Templates
- Metadata
- supports Typescript
- Services
- Router

Q25. What are the prerequisites for learning Angular?

Ans. 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

Q26. What IDE can be used to develop an Angular application?

Ans. 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.

Q27. How to set up the Angular Environment?

Ans. 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 from [here](#).

- **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. You can find the registry from [here](#).

- **Angular CLI**

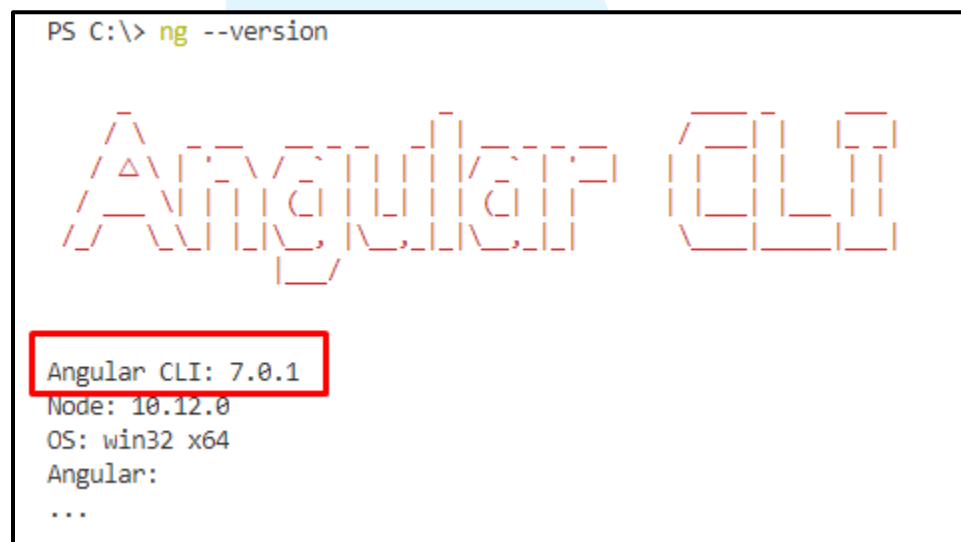
CLI is one of the important parts while getting started with Angular development and it is used to scaffold and build Angular app faster.

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
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.

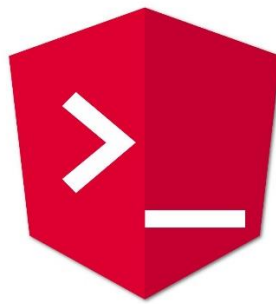
2

Angular CLI

Q1. What is Angular CLI?

Ans. 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.

Q2. What are the features of Angular CLI?

Ans. 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

Q3. How to use different Angular CLI Commands?

Ans. 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.

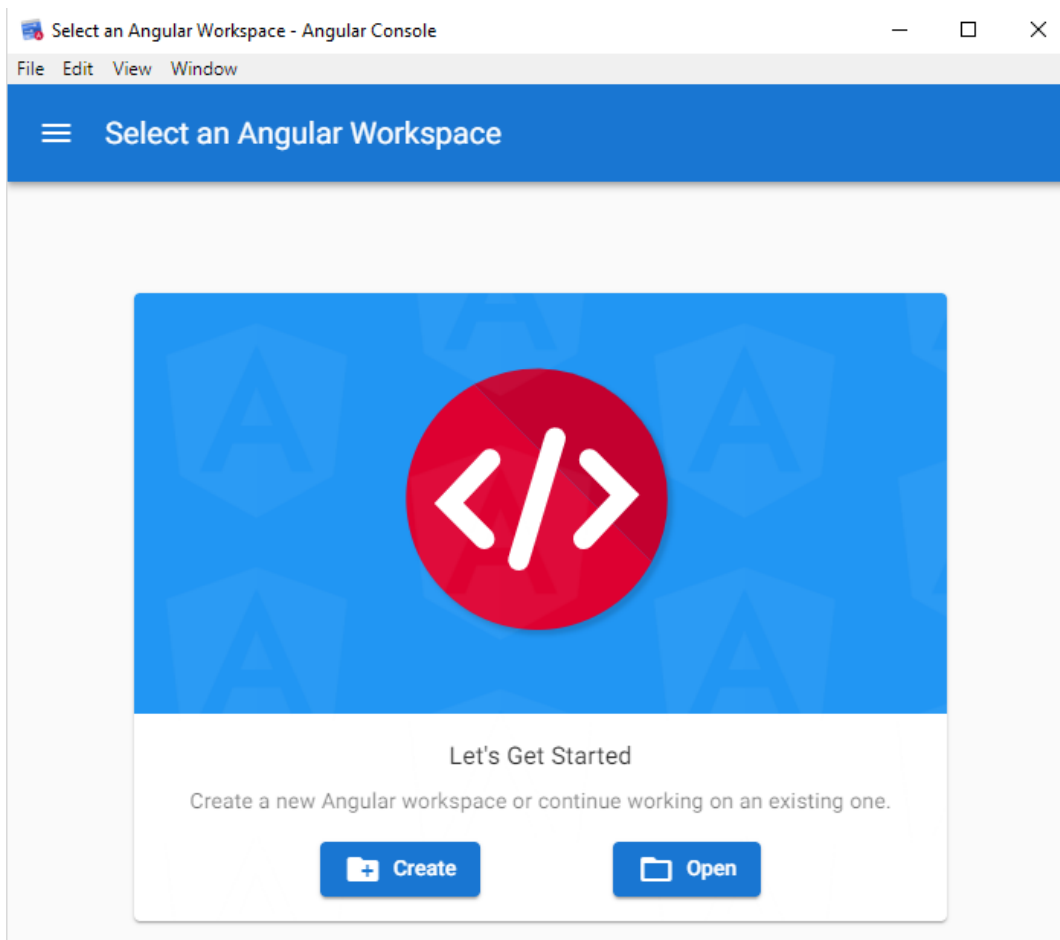
```
npm install -g @angular/cli
```

Q4. What is Angular Console?

Ans. 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 [here](#). And after downloading the tool, it will look like this.



Q5. What are the advantages of Angular Console?

Ans. 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

Q6. How to create a new Angular app?

Ans. 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.

Q7. How to serve the Angular app?

Ans. 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
```

Q8. How to change default port number other than 4200?

Ans. 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
```

Q9. How to change the default protocol other than HTTP?

Ans. 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.

Q10. How to update your project's dependencies?

Ans. 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.

Q11. How to generate various types of files using Angular CLI?

Ans. 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.

Q12. How to update all dependencies of the Angular app?

Ans. 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.

Q13. How to add a new dependency to the project?

Ans. 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
```

Q14. How to build an Angular application using CLI?

Ans. 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.

Q15. How to get project configuration details?

Ans. 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.

Q16. How to test the Angular app?

Ans. 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.

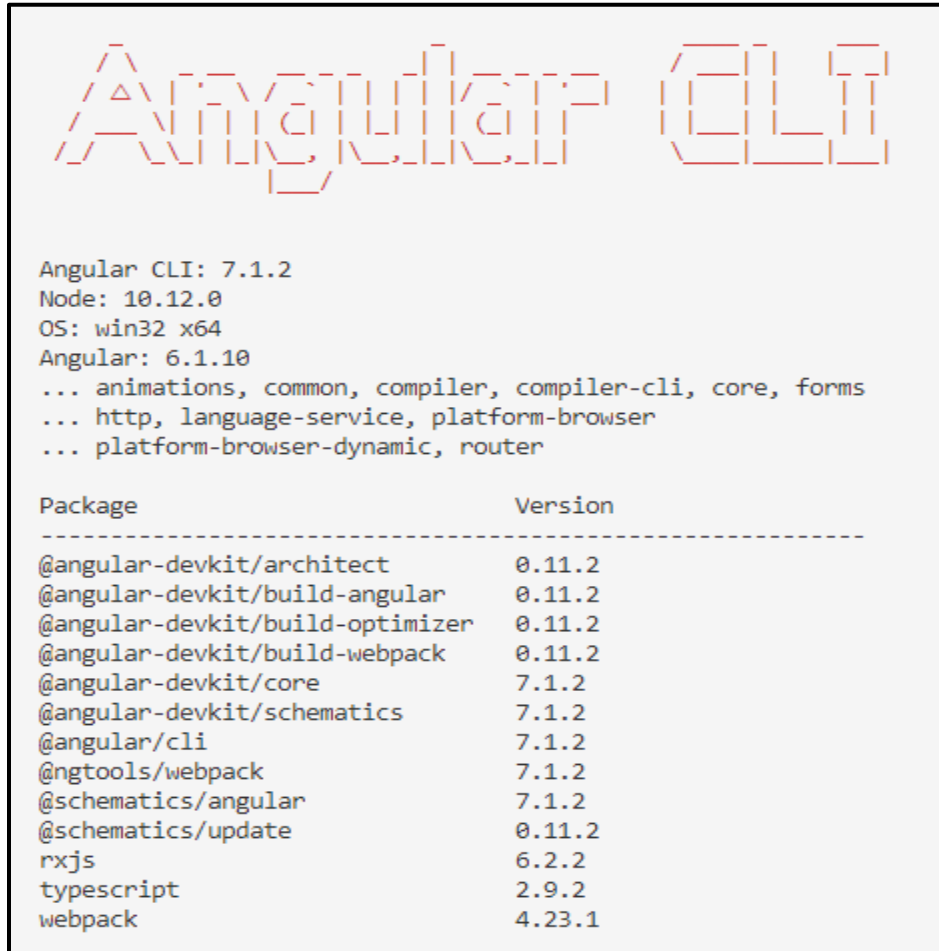
When we run this command, it will get the name of the project from the angular.json file where our testing related configuration is implemented.

Q17. How to get the version of Angular CLI?

Ans. 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
```

Q18. How to specify the type of the stylesheet using the command?

Ans. 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> --style <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.

Q19. How to get help from the CLI?

Ans. 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.
```

3

Components and Directives

Q1. What is Component in Angular?

Ans. Components are the basic building block of an Angular application.

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']
})
export class AppComponent {
  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 democomponent
```

Q2. What is the Template Expression?

Ans. 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

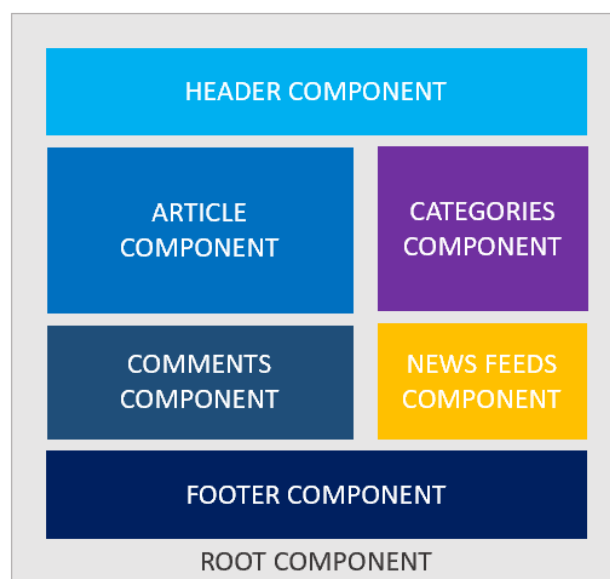
```
<p>Addition : {{ 1 + 1 }}</p>
<p>Subtraction : {{ 25 - 50 }}</p>
<p>Multiplication : {{ 25 * 50 }}</p>
<p>Division : {{ 25 / 50 }}</p>
```

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
```

Q3. How to manage a web page using Angular Component?

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



Q4. What are the Template Statements?

Ans. 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.

Q5. What is Data Binding in Angular?

Ans. 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.

Q6. What is Interpolation?

Ans. 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.

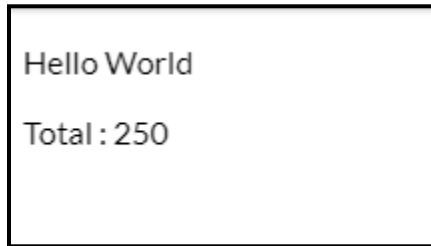
App.component.ts

```
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 the 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.



Q7. What is Property Binding?

Ans. 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.

Q8. How Does Event binding work in Angular?

Ans. 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 () brackets.

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.

Q9. How to implement Two-way data binding in Angular?

Ans. 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">
```

Q10. What is Directive in Angular?

Ans. 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

Q11. What is Component Directive in Angular?

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

Q12. What is Structural Directive in Angular?

Ans. 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() { }
}
```

Q13. What is Attribute Directive in Angular?

Ans. 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(private elref: 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.

Q14. How many types of built-in attribute directives are available?

Ans. 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.

Q15. What are various Structural directives in Angular?

Ans. 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

- **NgForOf**

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

Q16. What is the use of ngIf directive?

Ans. 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.

Q17. What is the use of `ngFor` directive in Angular?

Ans. `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();
  }

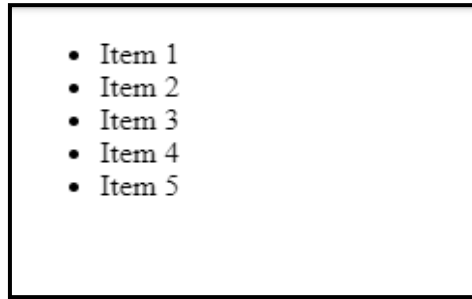
  ngOnInit() {
    this.items.push('Item 1');
    this.items.push('Item 2');
    this.items.push('Item 3');
    this.items.push('Item 4');
    this.items.push('Item 5');
  }

}
```

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.



Q18. What is the use of ngSwitch directive in Angular?

Ans. 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

```
<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

```
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.

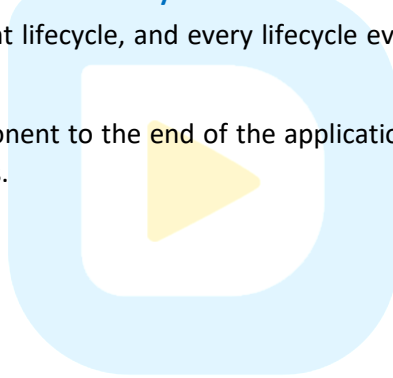
It will find the matching expression value from the available cases, and if one of them is the same as expression, then it will manipulate the Dom element.

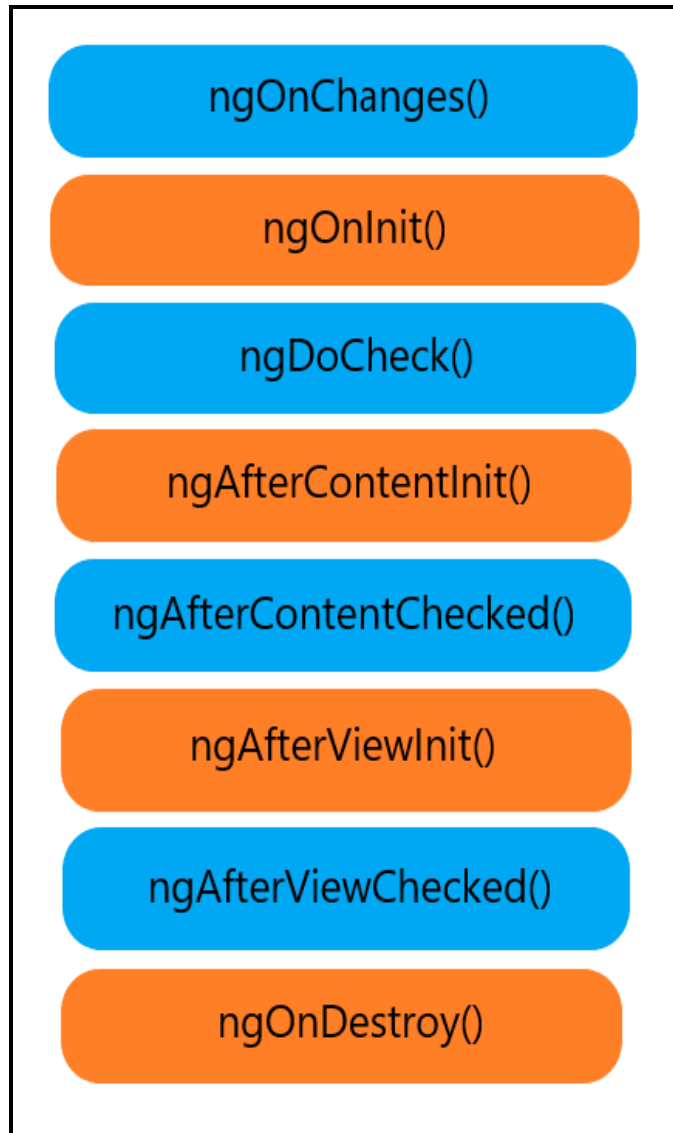
And if there are no matched cases found then it will render the default case element into the HTMLDom.

Q19. What are various Angular Lifecycle hooks?

Ans. 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.

Q20. Methods to Share Data between Angular Components

Ans. 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

Q21. How to share data using `@Input ()`

Ans. 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 component.

```
<!-- 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>
```

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

```
// 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.

```
// 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.

Q22. How to share data using @Output and EventEmitter ()

Ans. 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.

Q23. How to share data using @ViewChild ()

Ans. 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.

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

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

  counter: number = 0;

  increment() {
    this.counter++;
  }
  decrement() {
    this.counter--;
  }
}
```

```
}
```

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 ().

Q24. How to share data using the Service

Ans. 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.

Q25. Explain Change Detection in Angular

Ans. Change Detection in Angular detects any changes into and based on the changes it will update the HTML DOM.

Using zone.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.

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

```
// 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.

```

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

@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.

Q26. Explain Content Projection in Angular

Ans. 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.

```
<!-- 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.

```
// 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.

```
<!-- 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.

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

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

export class ChildComponent {}
```

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.

```
<!-- 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>
```

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.

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

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

export class ChildComponent {}
```

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.

This is my footer section

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.

4

Modules

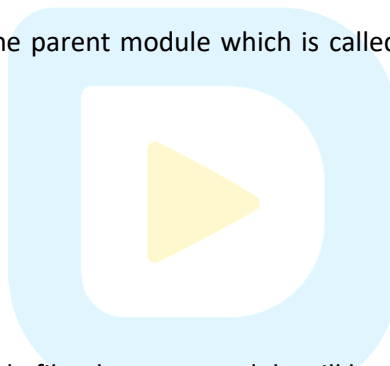
Q1. What is Module in Angular?

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

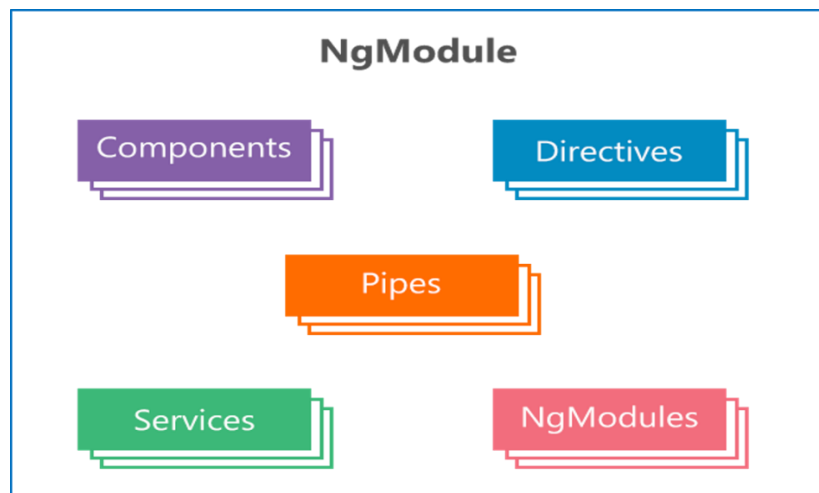
Or we can say that Modules plays an important role to structure our Angular application.

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.



Q2. How to create a new Module using Angular CLI?

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

```
ng generate module mycustommodule
```

Q3. What different types of Modules are supported by Angular?

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

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

Q4. Explain Angular Module basic structure?

Ans. 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

Q5. What are the various NgModule metadata properties?

Ans. 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

Q6. What are built-in Modules?

Ans. 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.



Q7. What are the different Feature Modules?

Ans. 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

Q8. How to create a Feature Module?

Ans. 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.

```
// myfeaturemodule.module.ts
```

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

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

Q9. What is the difference between the JavaScript module and NgModule?

Ans. 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.

Q10. How to create a Module with routing using CLI?

Ans. 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

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

const routes: Routes = [];

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

2. Mymodule.module.ts

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

@NgModule({
  declarations: [],
```

```
imports: [
    CommonModule,
    MymoduleRoutingModule
]
})
Export class MymoduleModule{ }
```

Q11. What is Entry Component?

Ans. 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

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

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

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

2. Using route definition

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

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

Q12. What are the Providers?

Ans. 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';  
}
```



5 Pipes

Q1. What is Pipe in Angular?

Ans. 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.

```
ng generate pipe mycustompipe
```

Q2. The defaults file structure of Angular Pipes

Ans. 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.

Q3. What is the use of PipeTransform Interface?

Ans. 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.

Q4. What are built-In Pipes in Angular?

Ans. 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.

Q5. How to use Pipes?

Ans. 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, and lower 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.

This Is First Line
THIS IS SECOND LINE
this is third line

Q6. What are different types of Pipes in Angular?

Ans. Mainly there are two types of Pipes are available.

- Pure pipes
- Impure pipes

Q7. What is a Pure Pipe?

Ans. 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.

Q8. Explain Pure Pipe using an Example?

Ans. 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) {
      return value = '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

Q9. Explain Impure Pipe using an example?

Ans. 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.

Q10. What is Pipes chaining?

Ans. 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>
```

Q11. What is Parameterized Pipe?

Ans. 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>
```

6

Routing and Navigation

Q1. What is Routing in Angular?

Ans. 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.

Q2. How to compose navigation URL?

Ans. 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.

```
<base href="/">
```

Q3. What is RouterModule?

Ans. 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.

Q4. How to use Routing in Angular?

Ans. 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';
```

Q5. What is the Route in Angular?

Ans. 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.

Q6. What are different types of methods in RouterModule?

Ans. 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.

Q7. What is RouterLink?

Ans. 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>
```

Q8. What are the different properties of a RouterLink?

Ans. 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

Q9. What are various Router events?

Ans. 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

Q10. What is RouterOutlet?

Ans. 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>
```

Q11. How to use routerLink?

Ans. 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
```

Q12. How to use RouterLinkActive?

Ans. 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.

```
<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.



Q13. What are the different Routing Strategies in Angular?

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

1. Path Location Strategy

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

2. 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.

Q14. Explain Path location routing strategy in Angular?

Ans. 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.

Q15. Explain Hash location routing strategy in Angular?

Ans. 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 then 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.

Q16. How to generate a routing module along with the normal module?

Ans. 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.

Q17. Explain the types of Route loading

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

- Pre-Loading
- Eager Loading
- Lazy Loading

Q18. What is Pre-Loading?

Ans. 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.

Q19. What is Eager Loading?

Ans. 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.

Q20. What is Lazy Loading?

Ans. 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.

```
// app-routing.module.ts
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: HomeComponent
  },
  {
    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.

7

Angular Forms

Q1. What is Angular Form?

Ans. 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.

Q2. What are the Building blocks of an Angular Form?

Ans. 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**

FormControls 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.

Q3. What is Reactive Form in Angular?

Ans. 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  
],
```

Q4. What is Template-driven Form in Angular?

Ans. 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 FormsControls and then it uses ngModel to enable template-driven forms approach.

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

```
import { FormsModule } from '@angular/forms';
```

And also need to add inside imports array like this.

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

Q5. What are the differences between Reactive form and Template-driven form in Angular?

Ans. 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

Unit testing will be much easier	May take more effort for unit testing
We can achieve great code readability	Unable to get proper readability due to a large amount of code implemented in the template
It is difficult to implement and understand	Easy to understand

Q6. What is FormControl in Angular and how to use it?

Ans. 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.

```
<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);
}
```


Q7. What is FormGroup and how to use it in Angular?

Ans. 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.

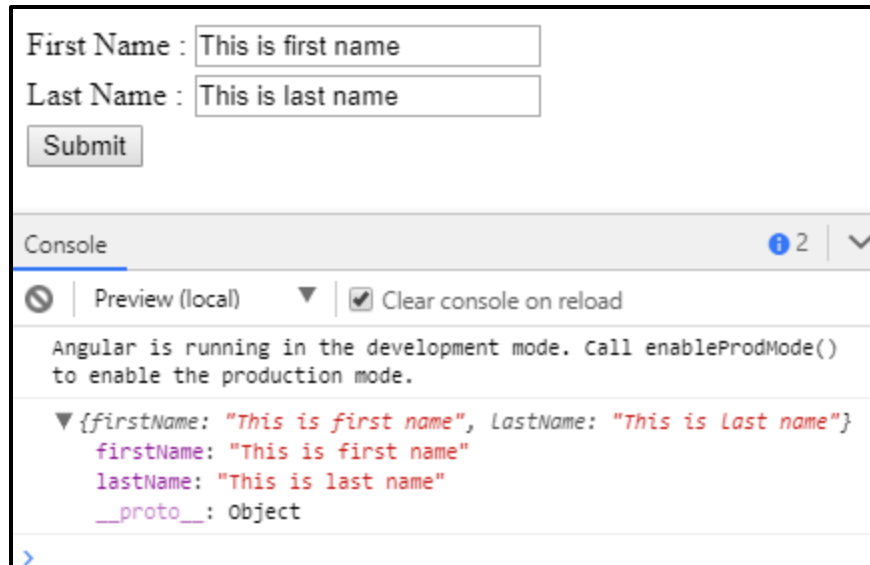
```
demoForm = new FormGroup({  
  firstName :new FormControl('Manav', Validators.maxLength(10)),  
  lastName:new FormControl('Pandya', Validators.maxLength(10)),  
});
```

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>  
</table>  
</form>
```

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



Q8. Explain Form validations in Angular?

Ans. 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.

Q9. What are various built-in validators in Angular?

Ans. 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

Q10. How to use built-in form validators?

Ans. 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.

Q11. How to create custom validation in Angular?

Ans. 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.

Q12. How get the submitted Reactive form values?

Ans. 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);
}

```

Q13. How to reset the form in Angular?

Ans. 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.



Dependency Injection

Q1. What is Dependency Injection in Angular?

Ans. 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 AngularJs 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.

Q2. What are the advantages of using Dependency Injection?

Ans. There are few advantages by using Dependency Injection with Angular which is listed below.

- It reduces dependencies
- Code reusability
- Better and understandable code
- Testable
- Loosely-coupled code
- It prevents circular dependencies

Q3. What is Injector in Angular?

Ans. 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.

Q4. What are many ways to implement Dependency Injection in Angular?

Ans. 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

Q5. How to use @Injectable () to implement DI in Services?

Ans. 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.

Q6. How to use @Injectable () to implement Dependency Injection in Modules?

Ans. 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.

Q7. How to use providers to implement Dependency Injection in the component?

Ans. 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.

Q8. What is Dependency Injection token?

Ans. 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){}
```


Observables and RxJS

Q1. What are Observables in Angular?

Ans. 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 of 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.

Q2. What is RxJS?

Ans. 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.

Q3. List down the different functions provided by the RxJS?

Ans. 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()



Q4. Example using RxJS operators with Angular

Ans. 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';

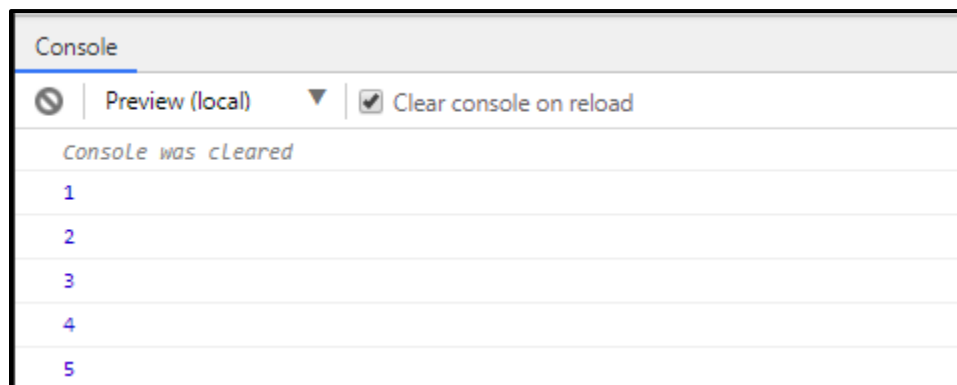
@Component({
  selector: 'my-app',
  templateUrl: './app.component.html',
})
export class AppComponent {

  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.

```
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 {

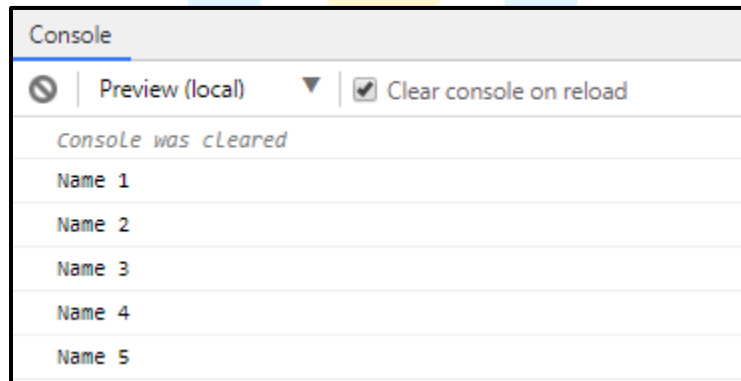
    userDetails = from([
      { id: 1, Name: 'Name 1' },
      { id: 2, Name: 'Name 2' },
      { id: 3, Name: 'Name 3' },
      { id: 4, Name: 'Name 4' },
      { id: 5, Name: 'Name 5' }
    ]);

    constructor()
    {
      // To iterate single property
      const user = this.userDetails.pipe(map( ({Name}) => Name));
      // subscribe the source
      const subscribeUser = user.subscribe(val => console.log(val));
    }
  }
}

```

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.



Q5. What is a Subscription?

Ans. 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 are 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.

Q6. How to unsubscribe from Observables using Async pipe?

Ans. 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.

Q7. Error handling using Observables

Ans. 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.

```
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.

Q8. Explain catchError and Finalize operators?

Ans. 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 executed.

```
// 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));
```

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

Angular Services and HTTP

Q1. What is Angular Service?

Ans. 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
```

Q2. What is the default structure of an Angular Service?

Ans. 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() { }
}
```

Q3. Why we should use Angular Services?

Ans. 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.

Q4. How to use service in Angular Components?

Ans. 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 steps 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.

Q5. What is Singleton Service?

Ans. 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{ }
```

Q6. What is HttpClient in Angular?

Ans. 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';
```

Q7. How many types of HTTP Requests are supported?

Ans. 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

Q8. How to inject HttpClientModule in our component?

Ans. HttpClient is used to initiate a request to the server using different methods like GET, Post, Put and delete, but 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{ }
```

Q9. How to inject HttpClient in our Angular Service?

Ans. After importing the HttpClientModule 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.

Q10. How to do error handling in HttpClient?

Ans. 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.

Q11. What is Interceptor?

Ans. 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.

Q12. How to create Interceptor in Angular?

Ans. 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.

Q13. What is `HttpInterceptor` in Angular?

Ans. `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.

Q14. What is the `intercept` method in `HttpInterceptor`?

Ans. 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()}`  
      }  
    });  
  }  
}
```

```
    return next.handle(request);  
  }  
}
```

This is how we can attach authorization header using `HttpInterceptor` and `intercept ()` to handle the request manipulation.

Q15. What is next object in Interceptor?

Ans. 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()
```

Q16. How to add an interceptor to the Providers?

Ans. 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{ }
```

Q17. What is the use of HttpHeaders in Angular?

Ans. 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.

Q18. What are different parameters supported in the method request header?

Ans. There are different types of parameters supported to use with the HTTPHeaders which are listed below.

- headers
- params
- observe
- withCredentials
- reportProgress
- responseType

Q19. How to update request header details?

Ans. 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.

11

Angular Testing

Q1. Why we need Testing?

Ans. Testing is a mechanism to prevent various software defects, and one of the key concepts is Unit testing which is a level of software testing and by using Unit testing we can test different component or a unit in the application.

Unit testing in angular is an automated process and it will take different inputs and process the test result to verify the application before presenting it to the customers.

Unit testing is also called Isolated testing because it is used to test an individual unit of an application.

Q2. Setup to test Angular app using Jasmine test

Ans. Jasmine is one of the most used and popular JavaScript-based testing framework used with Angular.

It is also called a behavior-driven development framework for the testing of any JavaScript based frameworks.

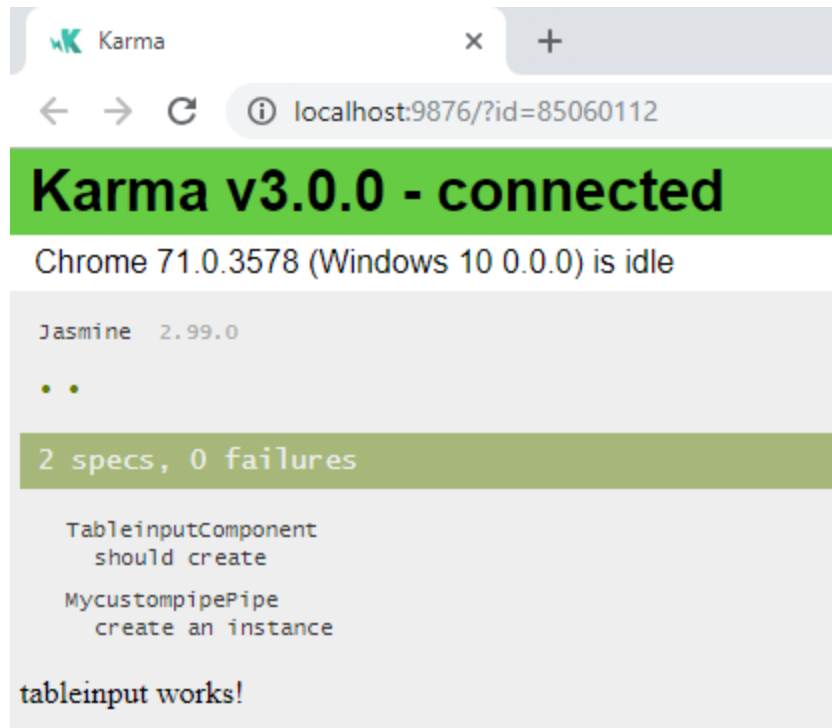
One of the advantages using Jasmine is that it is dependency free as well as don't required DOM.

When we create a new Angular app using Angular CLI than our application will be ready for testing using Jasmin framework.

We can run below command to run test cases using command prompt or PowerShell.

```
ng test
```

After executing the above command, a new browser window will be opened and you can see the test results like this.



In this example project, I have two different specifications for testing which can reside inside the file called `tableinput.component.spec.ts` and the code snippet look like this.

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { TableinputComponent } from './tableinput.component';

describe('TableinputComponent', () => {
  let component: TableinputComponent;
  let fixture: ComponentFixture<TableinputComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ TableinputComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(TableinputComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

This specification file is used to store the list of unit test specs so that we can run different test cases into the UI.

Q3. What is Karma?

Ans. Karma is a kind of tool which is used to run the Jasmine tests repeatedly by the refreshing browser window.

Whenever we update the code snippet, at that time we need to test the specifications, for that Karma comes into the picture.

When we create a new Angular app using Angular CLI, at that time Karma will be configured internally so that we don't need to configure it separately.

One of the best parts about Karma is that it runs the Jasmin tests automatically when we do changes into the files and it also provides the facility to run the tests into different devices like phone, tab, monitor, and PS3 devices.

Karma can be used with any testing framework like Jasmine, Mocha etc. which is the best feature of Karma.

Q4. What is Test entry file in Angular?

Ans. In Angular, our application will be configured for testing using Jasmine and Karma, but we may have a question like how Angular CLI configure the karma into our application.

For that karma uses a file called **karma.conf.js** as an entry component for the different test cases into our Angular app.

And the structure of **karma.conf.js** file looks like this.

```
module.exports = function (config) {
  config.set({
    basePath: '',
    frameworks: ['jasmine', '@angular-devkit/build-angular'],
    plugins: [
      require('karma-jasmine'),
      require('karma-chrome-launcher'),
      require('karma-jasmine-html-reporter'),
      require('karma-coverage-istanbul-reporter'),
      require('@angular-devkit/build-angular/plugins/karma')
    ],
    client: {
      clearContext: false // leave Jasmine Spec Runner output visible in browser
    },
    coverageIstanbulReporter: {
      dir: require('path').join(__dirname, '../coverage'),
      reports: ['html', 'lcovonly'],
      fixWebpackSourcePaths: true
    },
    reporters: ['progress', 'kjhtml'],
    port: 9876,
    colors: true,
    logLevel: config.LOG_INFO,
    autoWatch: true,
    browsers: ['Chrome'],
```

```
    singleRun: false
  });
};
```

As you can see into the above file's code snippet, we have different configuration options.

- **framework:** Name of the testing framework
- **plugins:** Different npm packages are used for testing
- **port:** The default port number to run testing into the browser
- **browsers:** The default browser name

Q5. Testing of Angular Components

Ans. The component is a basic building block in Angular app, and it contains Template as well as a stylesheet.

Basically, a Component is a combination of Class and Template, and it will generate native DOM elements based on the configuration.

Whenever we create a new Angular app, at that time one specification file will be created called **app.component.spec.ts**, where we can write different jasmine test specification like this.

Test case for App component :

- Should create the app
- Should have title Ng7Demo
- Should render title in an h1 tag

Test specification :

```
it('should create the app', () => {
  const fixture = TestBed.createComponent(AppComponent);
  const app = fixture.debugElement.componentInstance;
  expect(app).toBeTruthy();
});

it('should have as title 'Ng7Demo'', () => {
  const fixture = TestBed.createComponent(AppComponent);
  const app = fixture.debugElement.componentInstance;
  expect(app.title).toEqual('Ng7Demo');
});

it('should render title in a h1 tag', () => {
  const fixture = TestBed.createComponent(AppComponent);
  fixture.detectChanges();
  const compiled = fixture.debugElement.nativeElement;
  expect(compiled.querySelector('h1').textContent).toContain('Welcome to Ng7Demo!');
});
```

Let's understand these test specifications in details.

- **TestBed**

It provides the complete environment for the unit testing along with functionality to create the component and services into different unit tests.

Basically, it is a high-level testing framework which allows us to test the different environment.

- **Expect()**

Expect is a function which is also called expectation which takes the different expressions and will match with the matchers.

- **Matchers**

Matchers will compare the value of expression via expect () function.

There are different types of matchers are supported so that we can test our expressions based on the different situations

For example:

toEqual(), toBeGreaterThan(), toContain(), toBeNaN() and so on.

By using the combination of matchers and expressions, we can write different test specification based on our business requirements, and if any in-built matchers are not enough then we can write custom matchers also.

Q6. List down different in-built Matchers in Angular

Ans. Jasmine provides different types of in-built matchers which are listed below.

- expect(array).toBeEmptyArray();
- expect(array).toBeNonEmptyArray();
- expect(boolean).toBeBoolean();
- expect(boolean).toBeFalse();
- expect(object).toHaveArrayOfBooleans(memberName);
- expect(object).toHaveArrayOfNumbers(memberName);
- expect(object).toHaveArrayOfObjects(memberName);
- expect(object).toHaveMember(memberName);
- expect(object).toHaveMethod(memberName);
- expect(object).toHaveUndefined(memberName);
- expect(object).toHaveWhitespaceString(memberName);
- expect(object).toHaveWholeNumber(memberName);
- expect(regex).toBeRegExp();
- expect(string).toBeEmptyString();
- expect(string).toBeHtmlString();
- expect(string).toBeIso8601();
- expect(string).toBeJsonString();

These are the few in-built matchers supported but jasmine has vast collection of matcher which helps us to write different test specification for the all possible situations.

Q7. How to test service in Angular?

Ans. Service in Angular is one of the crucial parts for data communication, thus testing a service file is important.

When we create any service file using Angular CLI then additional service file will be created for the test specification where our service test cases are implemented.

For example, we have service spec file called **mydataservice.service.spec.ts** file and code snippet look like this.

```
import { TestBed } from '@angular/core/testing';

import { MydataserviceService } from './mydataservice.service';

describe('MydataserviceService', () => {
  beforeEach(() => TestBed.configureTestingModule({}));

  it('should be created', () => {
    const service: MydataserviceService = TestBed.get(MydataserviceService);
    expect(service).toBeTruthy();
  });
});
```

In this test specification file, we have one default case that service should be created, now let's assume we have a list of users and we need to add a new user into an existing list.

Then we can write a test case for that like this.

```
it('should add new user', () => {
  const user = "User 1";
  const service: MydataserviceService = TestBed.get(MydataserviceService);
  service.addNewUser(user);
  expect(service.users.length).toBeGreaterThanOrEqual(0);
});
```

In an above test case, we are using service method `addNewUser()` and trying to add a new user into the list, and at last, we are expecting that length of the user list should be greater than zero.

This is how we can write different test cases based on different scenarios.

Q8. How to test Directive in Angular?

Ans. The directive in Angular is a useful part and we need to test the Directives for its behavior, so for that, we can also write a test specification for Directives.

Let's see one simple example where we have a directive file named `demo-directive.directive.ts` and for the test specification, we have a `demo-directive.directive.spec.ts` file like this.

```
// demo-directive.directive.ts
import { Directive, HostListener, HostBinding } from '@angular/core';

@Directive({
  selector: '[appDemoDirective]'
})
export class DemoDirectiveDirective {

  constructor() { }

  @HostBinding("style.background-color") bgColor: string;

  @HostListener('mouseover') onHover() {
    this.bgColor = 'purple';
  }
  @HostListener('mouseout') onLeave() {
    this.bgColor = 'blue';
  }
}
```

In the above directive file, we are going to use HostListener which is used to listen to the events of elements or components.

In our example, we are going to use two different events onHover() and onLeave().

```
import { Component, DebugElement } from "@angular/core";
import { TestBed, ComponentFixture } from '@angular/core/testing';
import { DemoDirectiveDirective } from './demo-directive.directive';
import { By } from "@angular/platform-browser";

@Component({
  template: `<input type="text" appDemoDirective>`
})
class MyComponent {
}

describe('Directive: appDemoDirective', () => {

  let fixture: ComponentFixture<MyComponent>;
  let element: DebugElement;
  let component: MyComponent;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [MyComponent, DemoDirectiveDirective]
    });
    fixture = TestBed.createComponent(MyComponent);
    component = fixture.componentInstance;
    element = fixture.debugElement.query(By.css('input'));
  });

  it('should create an instance', () => {
    const directive = new DemoDirectiveDirective();
    expect(directive).toBeTruthy();
  });
});
```

```

});

// Custom specification
it('Hover on input', () => {
  // Mouse over event
  element.triggerEventHandler('mouseover', null);
  fixture.detectChanges();
  expect(element.nativeElement.style.backgroundColor).toBe('purple');

  // Mouse out event
  element.triggerEventHandler('mouseout', null);
  fixture.detectChanges();
  expect(element.nativeElement.style.backgroundColor).toBe('blue');
});
});

```

In our directive's specification file, there are two specifications are implemented.

- **Should create the instance**

In this specification, the directive's instance should be created and tested

- **Hover on input**

In this specification, we are going to assure that the background style should be changed according to the mouse and the color should be changed from blue to purple based on hover effect.

After running above test specifications, we can get output like this.



Q9. How to test a Pipe in Angular?

Ans. Pipe in Angular is used to transform the data into a different format based on the requirement, we can use in-built pipes or can create the custom one.

We can also test Pipes just like we do with Component and Service, below is the simple example of the pipe where we are converting text to lower case.

```
// smallcase.pipe.ts
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'smallcase'
})
export class SmallcasePipe implements PipeTransform {

  transform(value: string, args?: any): any {
    return value.toLowerCase();
  }

}
```

In this our custom pipe file, we are just transforming the text to lower case, now let's use our custom pipe in our template like this.

```
<!-- app.component.html -->

<h1>{{ userName | smallcase }}</h1>
```

After running our app, we will get expected output by converting text to lower case, now let's write test specification for that.

```
import { SmallcasePipe } from './smallcase.pipe';

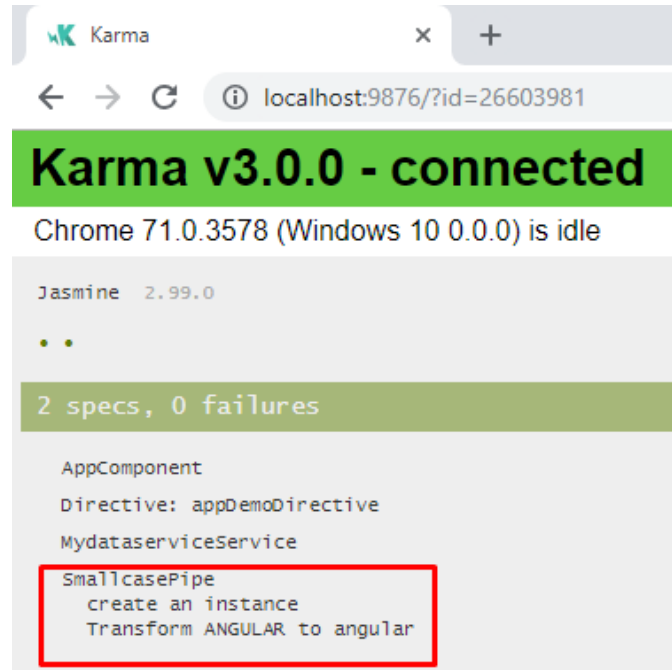
describe('SmallcasePipe', () => {

  it('create an instance', () => {
    const pipe = new SmallcasePipe();
    expect(pipe).toBeTruthy();
  });

  it('Transform ANGULAR to angular', () => {
    const pipe = new SmallcasePipe();
    expect(pipe.transform('ANGULAR')).toBe('angular');
  });
});
```

Here in this pipe specification file, we have one default specification to create the instance and another one is to transform uppercase text to lowercase.

When we run this specification, we can get the output like this.

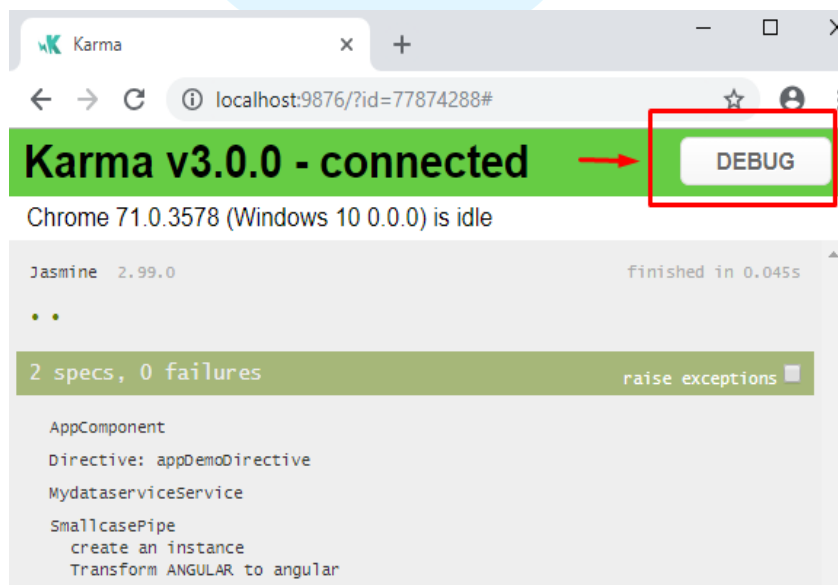


Q10. How to do Test Debugging in Angular?

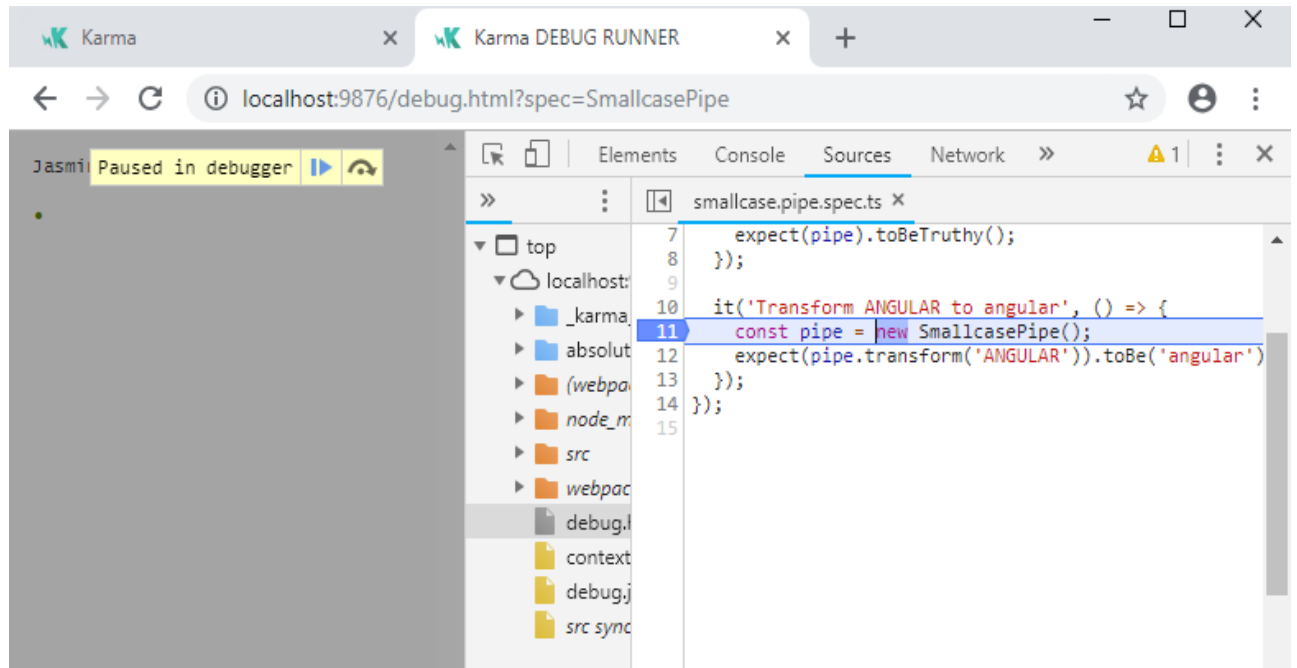
Ans. We can debug the different test specifications just like we do for JavaScript files using debug points.

For that, we need to follow few steps as described below.

- **Step 1:** Our first step is to run the test for the application by using the command **ng test** and it will open a new browser window
- **Step 2:** From the window, click on the DEBUG button like this.



- **Step 3:** After clicking on the debug button, a new tab window will be opened, then we need to open Developer tools
- **Step 4:** Now go to the Source tab and open any specification file (.spec.ts)
- **Step 5:** When we open any specification file, then put a breakpoint whenever you want in the file and refresh the browser window
- **Step 6:** After refreshing the browser window, and it will stop at the specific breakpoint like this.



12

Deployment

Q1. What is Deployment?

Ans. When our project is ready and tested against all the possibility, the next step is to deploy the project container into a specific environment.

Deployment of a project contains the several activities, and after all of the activities will be completed then we can deploy our project container, but sometimes it may possible that pre-deployment and other factors may cost you high but nowadays there are so many choices are there to deploy our application using cloud computing.

Deployment consists several activities which are listed below.

- Release product
- Installation to the specific environment and activation
- Product update/maintenance
- Adaption
- Product version tracking
- Deactivation/ Un-installation

These are the primary activities can be followed by the deployment, thus other factors are also concerned like pre-production environment, administration, database migration etc.

Q2. How to build an Angular app for Deployment?

Ans. To deploy the Angular app for a production environment, we can use below command.

```
ng build -prod
```

By executing the above command, it will generate a new folder called dist inside the current working directory which we can use to deploy to the server.

Q3. Explain the procedure for the Deployment in Angular

Ans. For deployment of an Angular app, we can follow these three steps.

- **Step 1:** Generate the production build for the application by using below command.

```
ng build -prod
```

- **Step 2:** After generating the production build using the above command, it will create a new folder called /dist, copy the same folder and upload it to the server.
- **Step 3:** After uploading the build folder, the next step is to configure the redirects using index.html if we want to set any default page or error page if something went wrong.

By following these three steps, now our product is production ready and can be used by the end-users.

Q4. How to enable runtime production mode?

Ans. When we work with the Angular application, at that time we are not working in runtime production mode, we should enable it manually.

I hope you have noticed something by opening developer tools into the browser and it shows the information like this.

Angular is running in the development mode. Call enableProdMode() to enable the production mode

We can get rid of this using -prod flag by generating production build of a project, and if you want to remove this info then you can go to the main.ts file and use enableProdMode() method like this.

```
// main.ts file

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

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

// For production environment
if (environment.production) {
  enableProdMode();
}

// To remove info from the developer tools
enableProdMode();

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err=>console.error(err));
```

And after applying the above changes, you can see into the developer tools that the message was removed. And enableProdMode() is just a function without any parameter which enables the production environment.

And also we can try another option as we have discussed already using --prod flag while generating the production build using below command.

```
ng build -prod
```

Q5. What is the difference between Build and Deployment?

Ans. Build and deployment are two different things, there is a connection between them directly, but the major difference between them is that Build will process and generates the build artifacts and it will reside on a new folder which can be ready to deploy.

Deployment is the process where our production ready build will be executed and at last, it will publish the output the same as we have used into the development mode.

Q6. How to set a different folder for the output of build other than /dist?

Ans. We may have questioned why we should always use /dist directory, but for that, we can change the default output folder to another folder using the angular.json file as described below.

We can change the default directory by using outputPath in build array which is the option and by using the same, we can set a custom folder for the output directory.

```
// angular.json
"options": {
  "outputPath": "mydirectory/project_name",
  "index": "src/index.html",
  "main": "src/main.ts",
  "polyfills": "src/polyfills.ts",
  "tsConfig": "src/tsconfig.app.json",
  "assets": [
    "src/favicon.ico",
    "src/assets"
  ],
  "styles": [
    "src/styles.css"
  ],
  "scripts": []
},
```

Just change the folder and project name into the options and build the angular app, now we can see that the output files will reside in our custom folder.

Q7. How to watch the changes while building an Angular app?

Ans. Sometimes we need to see the changes for the deployed version of our Angular app into the server, for that, we can use the flag --watch along with the ng build command like this.

```
ng build --watch
```

After executing the above command, we will get output like this for our module files.

```
Hash: 061786ee5239b6b64b2c
Time: 66353ms
chunk {budget-index-module} budget-index-module.js, budget-index-module.js.map (budget-index-module) 8.33 kB [rendered]
chunk {budgetbox-index-module} budgetbox-index-module.js, budgetbox-index-module.js.map (budgetbox-index-module) 31.9 kB [rendered]
chunk {budgetstatus-index-module} budgetstatus-index-module.js, budgetstatus-index-module.js.map (budgetstatus-index-module) 21.8 kB [rendered]
```

References

This book has been written by referring to the following sites:

1. <https://angular.io/> - Angular official site
2. <https://stackoverflow.com/questions/tagged/angularjs> - Stack Overflow - AngularJS
3. <https://www.dotnettricks.com/learn/angular> - Dot Net Tricks - Angular

