Angular Interview Questions & Answers

By Manav Pandya

Technical Author and Front-end Engineer

DotNetTricks

Angular Interview Questions & Answers

Release History

- Initial Release 1.0.0 - 2nd Jan 2019
- Second Release 2.0 - 15th Apr 2020
- Third Release 3.0 - 17th Mar 2021
- Fourth Release 4.0 - 25th Jan 2022

About Dot Net Tricks

Dot Net Tricks were founded by Shailendra Chauhan in Jan 2010. Dot Net Tricks came into existence in 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 currently registered by Dot Net Tricks Innovation Pvt. Ltd. came into shape in 2015. Dot Net Tricks website has an average footfall to the tune of 300k+ per month. The site has become a cornerstone for 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 many trainees who have received training from our platforms and immediately got placement in some reputed firms testifying our claims of providing quality training. The website offers you a variety of free study material in the form of articles.

Unlimited Live Training Membership

Upgrade your skills set with hands-on real-time project-based training programs to build expertise on in-demand job skills and become industry competent. DotNetTricks Unlimited Live Training enables you to Become:

- **Full-stack JavaScript Developer** - JavaScript, Node.js, React, Angular
- **Full-stack .NET Developer** - .NET, MVC, ASP.NET Core, WebAPI
- **Cloud Engineer/Architect** - AWS, Microsoft Azure
- **Technical Architect** - Microservices, Design Patterns, and Clean Architecture
- **DevOps Engineer** - DevOps, Docker, and Kubernetes
- **Mobile Developer** - Xamarin, React Native

Learn more about Unlimited Live here: https://www.dotnettricks.com/membership

Self-Paced Courses Membership

The most effective way to gain job-ready expertise for your career. Learn how to build highly scalable modern web applications & transform your coding skills. Learn to build projects and build expertise on .NET, JavaScript, Database, Cloud, DevOps, Docker, Front-end technologies, and many more cutting-edge technologies which are in industry demand today.

- 5,00+ hours of unlimited access to our premium content
- Real Hands-on Labs with integrated IDE
- Full access to training, study mode quizzes, and assignments
- Step-by-step learning with exclusive Learning Paths
- Become job-ready with interview prep sessions

Learn more about Self-paced courses membership here: https://www.dotnettricks.com/plus-membership

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 that made this possible. I want 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 with vast experience working on different technologies like Angular, React, React-Native, VueJs, TypeScript, and 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 applications (SPA) and Mobile and Desktop based applications. This book will 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 those who will 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 and HTTP
- HTTP Client and REST API
- Pipes
- Dependency Injection
- Unit Testing
- Angular Deployment
- Ivy rendering Engine
- Angular 10
- Angular 11
- Angular 12
- Angular 13

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

About the Author

Manav Pandya - An Author and Front-end Engineer

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 possible, if something strikes you as odd or finds 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 are always happy to hear from our readers. Please provide your valuable feedback and comments!

You can follow us on YouTube, Facebook, Twitter, and LinkedIn or subscribe to our RSS feed.

Table of Contents

1

Introducing Angular

Q1. What is Angular?

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

Angular2+ is different from Angular1.x, and it is completely re-written from scratch using Typescript. When writing this book's second edition, the latest Angular version was 9, which was released on 6 Feb 2020.

Q2. Explain the evolution history of Angular?

Ans. The evolutionary history of Angular is given below:

- Developed in 2009 by Misko Hevery and Adam Abrons at Brat Tech
- Misko Hevery started to work for Google in 2009
- Angular version 1.0 (AngularJS) was released in 2012 by Google
- Angular version 2.0 was released in September 2016
- Angular 4.0 was released in Mar 2017
- Angular 5.0 was released in Nov 2017
- Angular 6.0 was released in May 2018
- Angular 7.0 was released in Oct 2018
- Angular 8.0 was released in May 2019
- Angular 9.0 was released in Feb 2020
- Angular 10.0 was released in Jun 2020
- Angular 11.0 was released in Nov 2020

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

Ans. There are the 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	It doesn't have CLI
The class is the only way to define services in Angular2	factory, service, provider, value, and constant are used for services
Runs on client-side & server-side	Run-on only client-side
bootstrapModule() function is used to initialize	ng-app and angular.bootstrap() function are used to initialize
Supports Pipe	Support filters
Supports camelCase and PascalCase syntaxes likengModel, ngForm and NgForm	Supports spinal-case and camelCase syntaxes like ng-model, ng-class, and ngModel
Use Html DOM elements properties and events	Uses its directives like ng-click, ng-show, and ng-src, etc.
Use () for events and [] for attributes	Doesn't support () and [] based syntax

Q4. Why use Angular?

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

- Angular is a 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
- Helps to enable testing using jasmine and karma
- Great community support

Q5. What are the building blocks of Angular?

Ans. Angular has a large number of features, making it more useful than other front-end frameworks; below is the list of essential building blocks.

- 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 and new features, which 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?

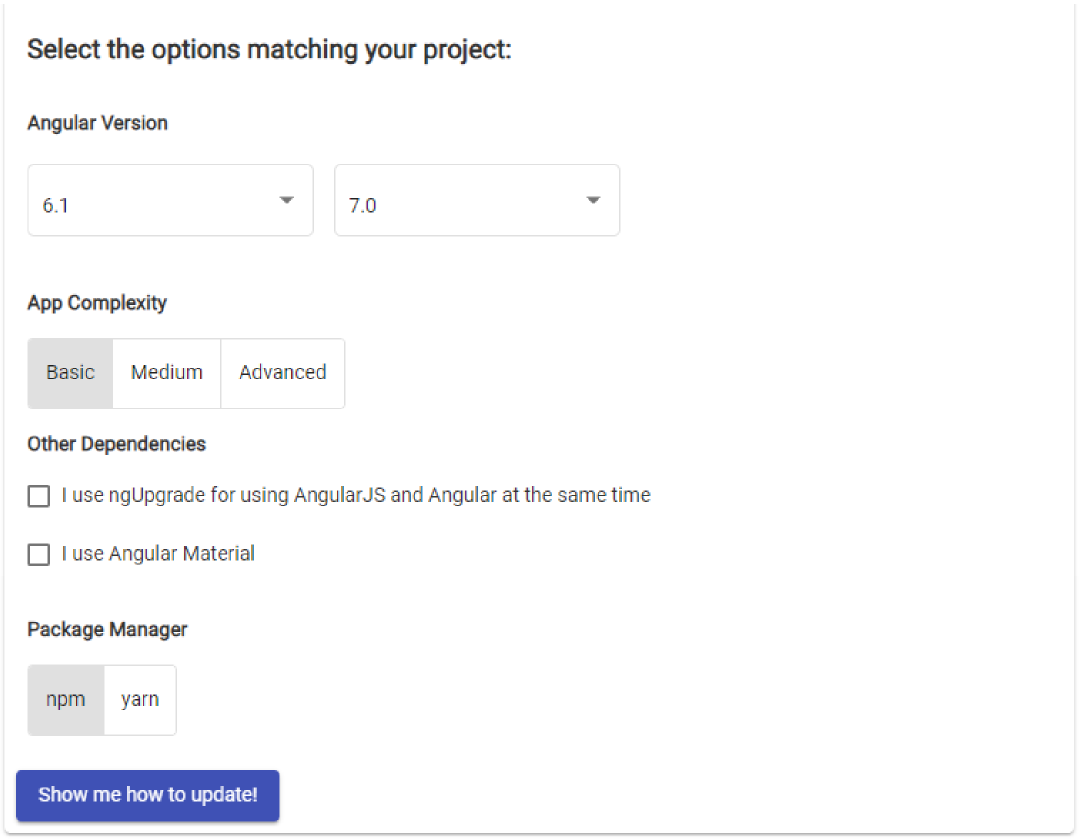
Ans. If you want to work on the latest Angular version, we need to update our Angular CLI and Core angular packages using the below command.

```
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 Angular 7 by following a 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

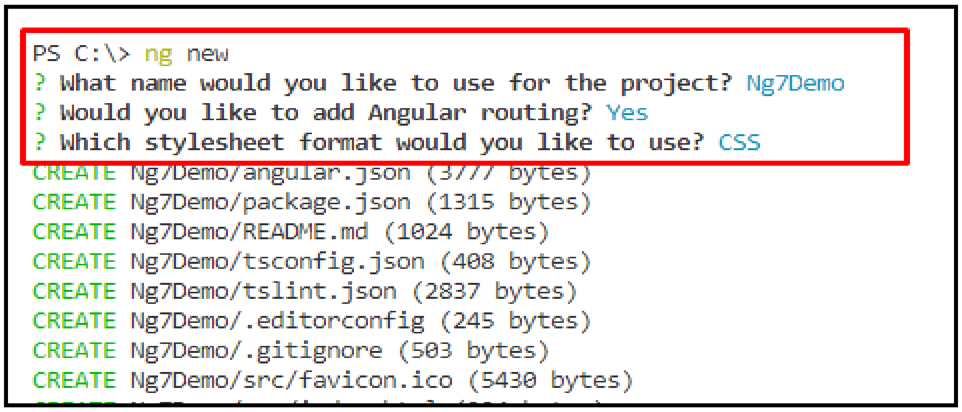
[✓ npm] yarn

Show me how to update!

Q9. What is a new feature called CLI prompt?

Ans. We know that it is challenging to keep in mind all those npm commands, but sometimes we may forget a few of them and get stuck, and in the end, we will find it with the help of different search engines.

Using CLI prompts, now we can use different commands very quickly, as 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, it will ask for three different names, 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. To use Angular Elements in our app, we need to add a new core element package with the appropriate name, as explained in the below command.

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

Here the core package for elements is @angular/elements, and to create new features, 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 they can be used by almost any framework which needs browser support.

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

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

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

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

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

If you observe the above code lines, we will inject the injector, and then we need to import elements and below.

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

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

Q12. What's new in Angular Material?

Ans. With the release of Angular 7, some highlighted changes 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 the latest Materialized components, we need to update our latest angular material package using the below command.

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

Q14. Do you explain Virtual Scrolling in Angular?

Ans. Virtual scrolling is one of the new and widely used features added with Angular 7, which allows us to scroll the long list of elements, images, and other elements.

A new module is added called ScrollingModule to enable virtual scrolling into our Angular application for virtual scrolling.

By using virtual scrolling, we can load and add different elements into the DOM, and in the same way, we can also remove them 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 added an inside import array.

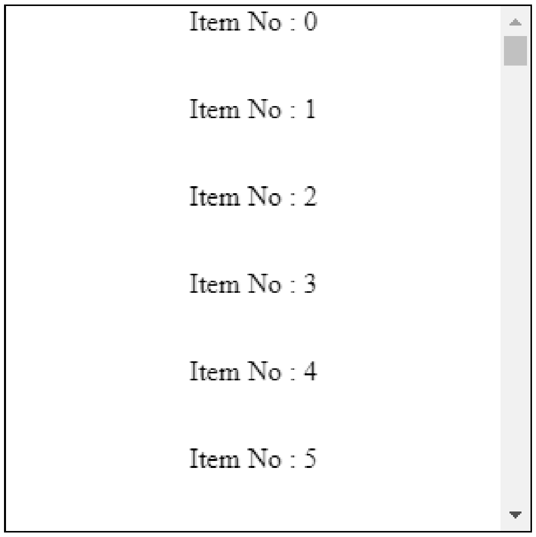
To render the list of items, we will use the following code snippets.

```
// 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 forget to add some stylesheets 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 get the output like this.



```
Item No : 0

Item No : 1

Item No : 2

Item No : 3

Item No : 4

Item No : 5
```

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

Not only drag and drop but different other functionality can be implemented, 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 orientations 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

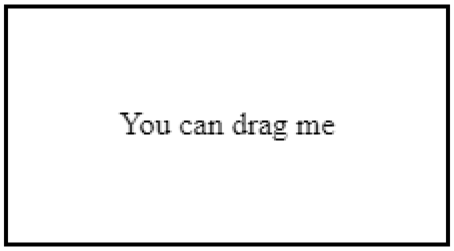
In this example, we will implement a simple draggable div using cdkDrag, following the steps explained below.

```
// app.module.ts

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

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

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

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

Now we are done with our simple example; let's run this see how it looks like.



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

Q17. What is the Budget bundle in Angular 7?

Ans. We know that the app's size is such a crucial thing, and for that, we need to optimize the total size of the application; using Angular 7, we can use the Budget bundler to be notified whenever our app reached 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"
}]
```

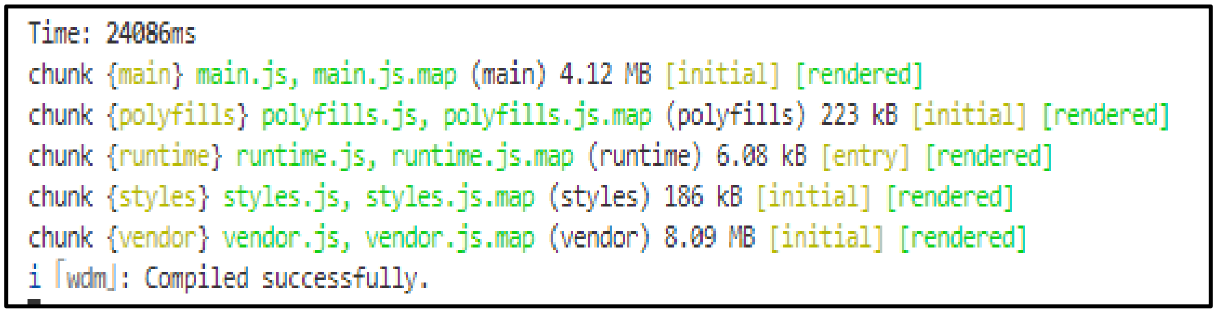
This is how we can do budget bundling, which is directly reflected in the application performance, so we should make sure that our application will be as small as possible.

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 the below command.

```
ng serve
```

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



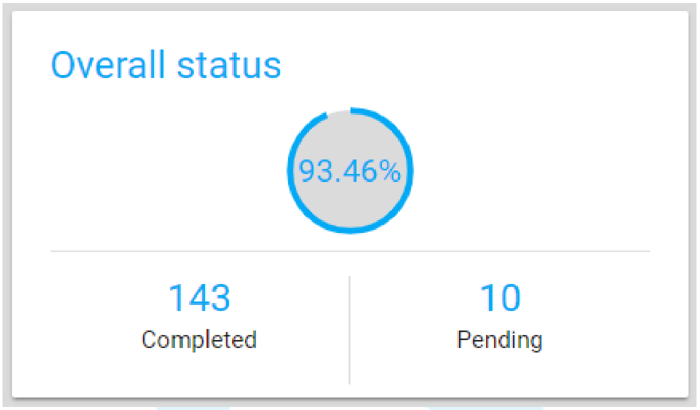
```
Time: 24086ms
chunk {main} main.js, main.js.map (main) 4.12 MB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 223 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.08 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles)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

Q19. What is the Ivy Renderer engine in Angular?

Ans. Ivy in Angular is a view engine. We can say view rendering pipeline, which is just like a compiler that takes component and template data, then converted into standard HTML and JavaScript, at last, reflected the DOM.

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

Thus, the Ivy renderer is not entirely ready for use, as suggested in this diagram.



We can get the complete setup and readme guide by following the link here.

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

Q20. What's new in Angular 9?

Ans. Recently, Angular 9 was released on 7th February 2020, and it comes with critical updates about the Ivy rendering engine; below are the few updates in Angular 9.

- Ivy rendering engine as the default engine
- Module along with the provider support
- Core dependency injection
- Service worker updates
- I18n improvements
- Angular diagnostic improvements
- Selector less directive

Q21. How to upgrade to Angular 9?

Ans. Upgrading Angular 8 to Angular 9 is pretty easy; we just need to update Angular core packages using the below command.

```
ng update @angular/cli@8 @angular/core@8
```

Then we have to upgrade Angular to version 9 using the below command.

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

After following the above command, we will get started with the latest version of Angular 9.

Q22. What is internationalization in Angular 9?

Ans. The Angular 9 version release brought us one feature called i18n in the template, which allows us to use a new function called $localize.

This new internationalization feature provides localization support using the latest package called @angular/localize. For that, we have to add a package into the application using the below command.

```
ng add @angular/localize
```

Q23. How to set up localization in the Angular 9 app?

Ans. The new package called @angular/localize is used to provide localization support to our Angular application. Still, to apply the different languages to convert the text, we have to configure something.

We have to configure the file called angular.json, where we need to setup i18n set up just like this.

```
"i18n": {
        "locales": {
          "en": "src/locale/messages.en.xlf",
          "es": "src/locale/messages.es.xlf",
        }
},
```

Before the above configuration, we have to extract the text messages into the file called **messages.xlf** file, and then we will be able to configure other locale files.

We should use the below command to extract the messages.

```
ng xi18n
```

Q24. What's new in Angular 8?

Ans. Angular 8 was released on 23rd May 2019, and it brought some changes that make it unique compared to Angular 7.

- Web worker
- Ivy Rendering engine
- Bazel support
- Differential loading
- Router backward compatibility
- Typescript support 3.4
- Lazy loading
- ngUpgrade improvements
- Workspace API

Q25. 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

Q26. 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

Q27. 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

Q28. 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

Q29. What are the prerequisites for learning Angular?

Ans. There are some prerequisites to learning Angular, which is listed below.

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

Q30. What can IDE 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 that we can use to develop the Angular application end to end.

Q31. How to set up the Angular Environment?

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

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

- **Nodejs**

  To get started with Angular, we need to have node.js installed; if you don't have node.js 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 uses this to install the various packages into our application.

  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**

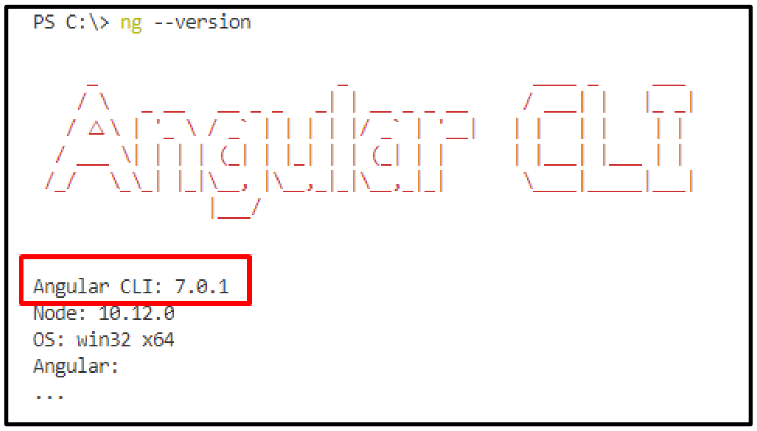
  CLI is one of the essential parts while getting started with Angular development, and it is used to scaffold and build an Angular app faster.

  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 the below npm command.

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

  After installing the CLI, we can find a release if you want to know the current version.



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

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

- **Any text editor tool as IDE**

  We can choose any of the code editors who are listed above in the IDE question. For Angular development, Visual Studio Code is an excellent 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 used to create, initialize, scaffold, and manage the whole angular application.

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

You can find the complete details of Angular CLI and various commands in the CLI introduction part later on in 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 to get started with Angular development within a few minutes.

- Packaging application and release for the deployment
- Testing the 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 build or deploy the angular application quickly.

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

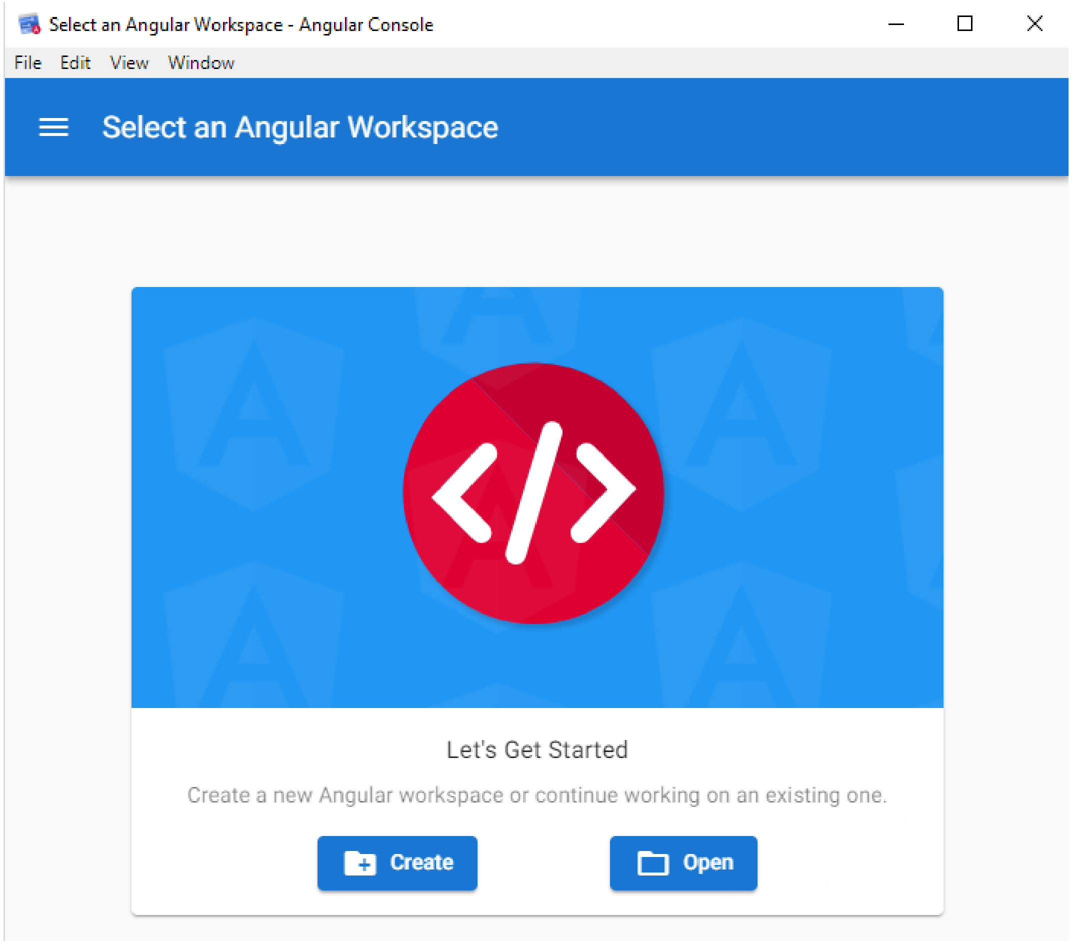
```
npm install -g @angular/cli
```

Q4. What is Angular Console?

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

Angular Console is a way to achieve different command-line features using a simple GUI, an entire alternative to know all 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.


Select an Angular Workspace - Angular Console
File
Edit
View
Window
Select an Angular Workspace
Let's Get Started
Create a new Angular workspace or continue working on an existing one.
Create
Open

Q5. What are the advantages of Angular Console?

Ans. There are a 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 the CLI command, then we can use the 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, it's time to run our app; we can use the below command to serve our app.

```
ng serve
```

After executing the above-given command, our application will start building, and it can also help rebuild the app after any changes occurred.

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

```
ng serve –o
```

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

Ans. If you have created a new Angular app and server the app using ng serve, our app will be executed on URL localhost:4200, but we can change it using an additional flag and ng serve –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 an 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 run our Angular app using HTTPS securely; by default –ssl value is false.

Q10. How to update your project's dependencies?

Ans. To update 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 a new file using just a single line of command. Using a command, we can generate multiple kinds 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 commands, now we will be able to generate a 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 dependencies and update all of the project's dependencies using the 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 the directory to the server.

It uses web pack build tools, and the data can be fetched from the angular.json file.

We can use the 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, we need to know the details about project configuration like project name, default output directory, routing configuration, testing, and other essential information.

Using the below command, we will get the complete data of the angular.json file, the global configuration file for the project.

```
ng config
```

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

Q16. How to test the Angular app?

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

```
ng test
```

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

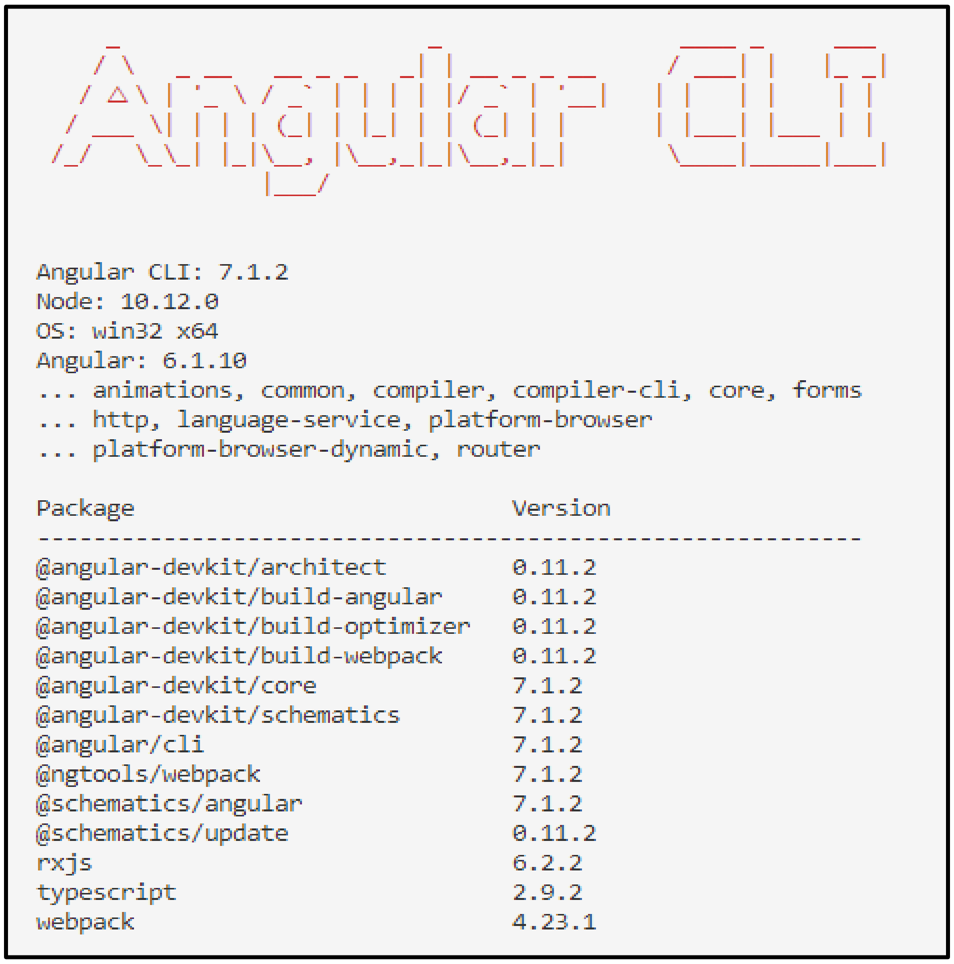
When we run this command, it will get the project's name from the angular.json file where our testing-related configuration is implemented.

Q17. How to get the version of Angular CLI?

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

```
ng -version
```

After running the 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 CSS was the default format for styling the pages for an older version.

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

```
ng new <project_name> --styles <type>
```

Example:

```
ng new ngDemo --styles scss
```

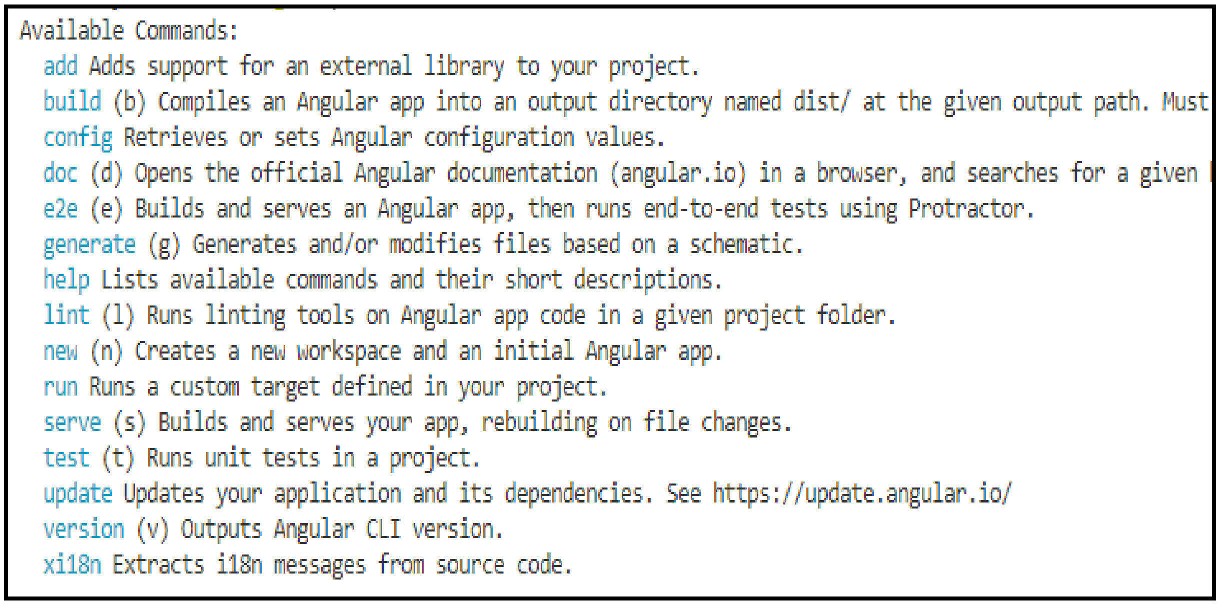
After running the above command, our default stylesheet format will be SCSS, not the CSS; this is how we can change the stylesheet type.

Q19. How to get help from the CLI?

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

```
ng help
```

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



```
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.
```

Q20. How to use Web workers with Angular CLI?

Ans. Web workers allow us to run the scripts in the background without affecting the actual execution process; from the Angular 8 version onwards, we will implement web workers in our Angular application.

We can add the web worker to the application by using the following command.

```
Ng generate web-worker webworker_name
```

After executing the above command, the new web worker file will be created, and the code snippet should look like this.

```
addEventListener('message', ({ data }) => {
  const response = `worker response to ${data}`;
  postMessage(response);
});
```

Q21. Testing the Angular app using Bazel

Ans. Bazel is one of the new features released with Angular 8, used to build our CLI application.

As per the official definition.

"The @angular/bazel package provides a builder that allows Angular CLI to use Bazel as the build tool."

The bazel allows our Angular application to build incremental builds of our application that create a separate task graph that allows building only necessary parts of the application that have sufficient changes that reflect the application's rebuild.

Q22. How to install the Bazel in our Angular app?

Ans. To use the **bazel** in the Angular application, there is one separate command to install the bazel into our app, which is given below.

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

Q23. How to install the Bazel in our existing Angular app?

Ans. If we don't want to upgrade our Angular app to the latest version but still want to use Bazel, we can use the below command to add Bazel support to our existing Angular app.

```
ng add @angular/bazel
```

Q24. How to integrate YouTube player in Angular 9?

Ans. Angular 9 comes with another update that allows us to integrate the YouTube player inline into our Angular application. Still, for that, we have to install one additional package using the following command.

```
npm install @angular/youtube-player
```

Now, let's import the newly added package into the root module file called the app.module.ts file.

```
import {YouTubePlayerModule} from '@angular/youtube-player';

@NgModule({
  imports:      [ BrowserModule, FormsModule, YouTubePlayerModule ],
  declarations: [ AppComponent, HelloComponent ],
  bootstrap:    [ AppComponent ]
})
```

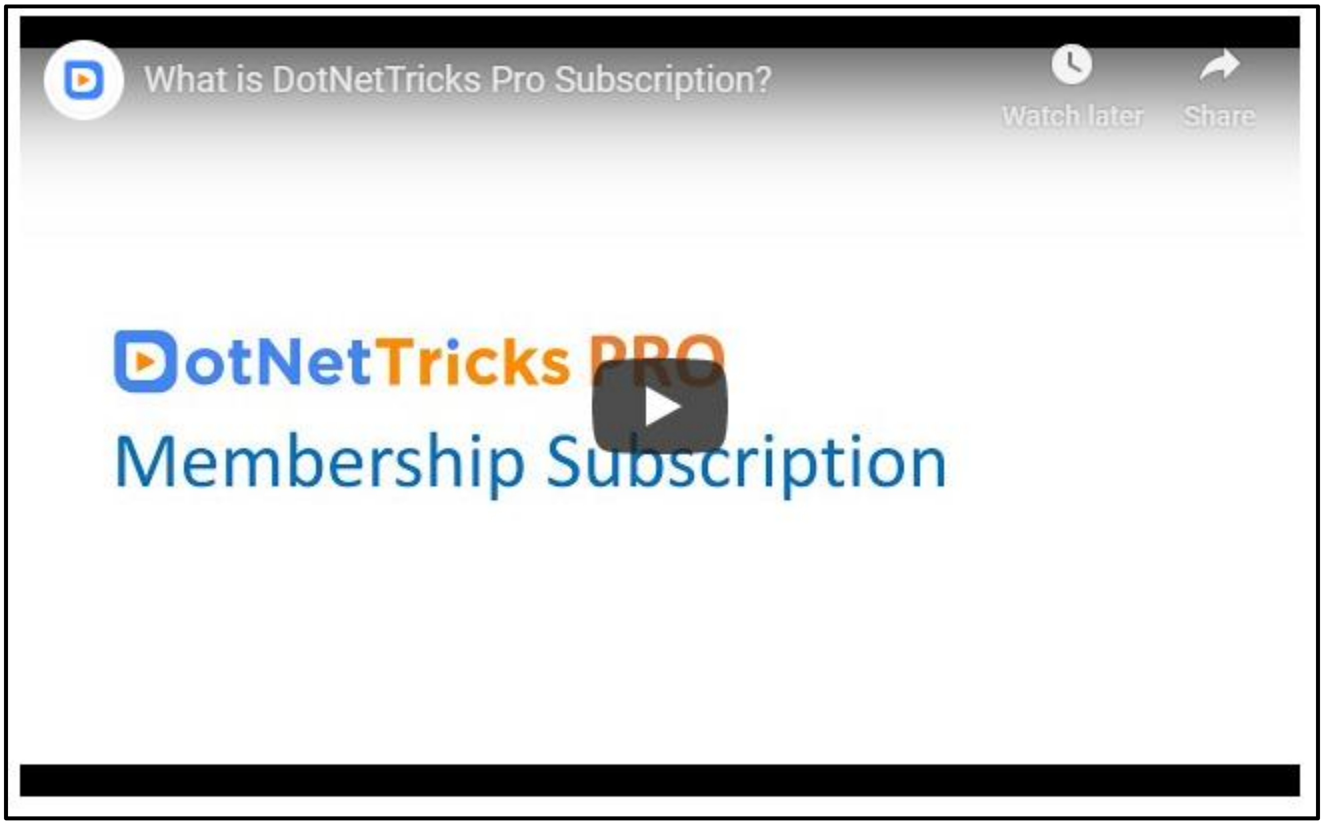
After importing the module, we have to provide the Component video URL to integrate the YouTube video into our Component.

```
import { Component, OnInit } from "@angular/core";
import { YouTubePlayerModule } from "@angular/youtube-player";

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

  ngOnInit() {
    const tag = document.createElement('script');
    tag.src = "https://www.youtube.com/iframe_api";
    document.body.appendChild(tag);
  }
}
```

Now let's jump to the template section where we are going to use the youtube player.

```
<youtube-player videoId="PRQCAL_RMVo"></youtube-player>
```

Once you reload the application, we can see the output like this.



Q25. How to integrate Google Maps in Angular 9?

Ans. Another package was released along with the Angular 9 version. We can also combine the Google map directly using the additional package; we can install it using the below npm command.

```
npm install @angular/google-maps
```

After installing the Google maps package, we have to get the API_KEY by following a few steps at the google maps official documentation, now open the app.module.ts file and import the package like this.

```
import {GoogleMapsModule} from '@angular/google-maps';

@NgModule({
  imports:      [ BrowserModule, FormsModule, GoogleMapsModule ],
  declarations: [ AppComponent, HelloComponent ],
  bootstrap:    [ AppComponent ]
})
```

Now, let’s configure the map, marker, and mouse moves the event into the app.component.ts file.

```
import { Component, ViewChild } from "@angular/core";
import { MapInfoWindow, MapMarker } from "@angular/google-maps";

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

  @ViewChild(MapInfoWindow, { static: false }) infoWindow: MapInfoWindow;

  center = { lat: 24, lng: 12 };
  markerOptions = { draggable: false };
  markerPositions: google.maps.LatLngLiteral[] = [];
  zoom = 4;
  display?: google.maps.LatLngLiteral;

  addMarker(event: google.maps.MouseEvent) {
    this.markerPositions.push(event.latLng.toJSON());
  }

  move(event: google.maps.MouseEvent) {
    this.display = event.latLng.toJSON();
  }

  openInfoWindow(marker: MapMarker) {
    this.infoWindow.open(marker);
  }

  removeLastMarker() {
    this.markerPositions.pop();
  }
}
```

Here in this component file, we have imported the required namespace for the google-maps that we are going to integrate with our angular 9 apps. Then we do have different events such as addMarker(), move(), openInfoWindow(), and the removeLastMarker(), which are useful to add the user interaction while using the map.

Our last step is to add the map markup into the template file called app.component.html which should look like this.

```
<google-map height="400px" width="750px" [center]="center" [zoom]="zoom"
(mapClick)="addMarker($event)"
       (mapMousemove)="move($event)" (mapRightclick)="removeLastMarker()">
       <map-marker #marker *ngFor="let markerPosition of markerPositions"
[position]="markerPosition"
              [options]="markerOptions" (mapClick)="openInfoWindow(marker)"></map-marker>
       <map-info-window>Info Window</map-info-window>
</google-map>
```

This is how we can easily integrate the Google maps in our Angular application by just using the @angular/google-maps package added in the Angular 9 version.

3

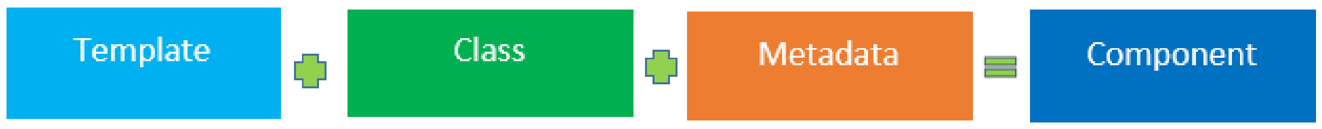
Components and Directives

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 of as a web page. An Angular component is exported as a custom HTML tag like as: <my-app></my-app>.



When we create an Angular app using CLI, it will generate the default App component as its entry point.

To declare a Component class, we can use the Decorators and decorators to declare a component that looks like the below example.

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

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

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

```
ng generate component democomponent
```

Q2. What is the Template Expression?

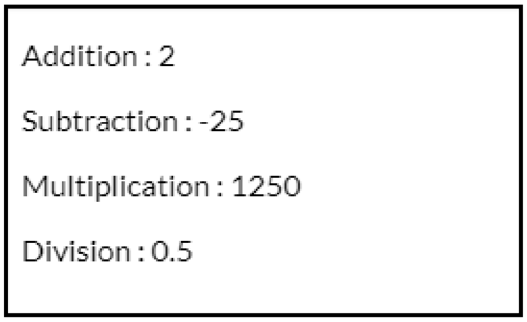
Ans. Template expression in Angular is the standard expression that we use in JavaScript. Still, one of the significant differences is that you can use JavaScript-based expressions within HTML template except for a few operators such as (new), (++), and (--), which are not supported.

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>
```

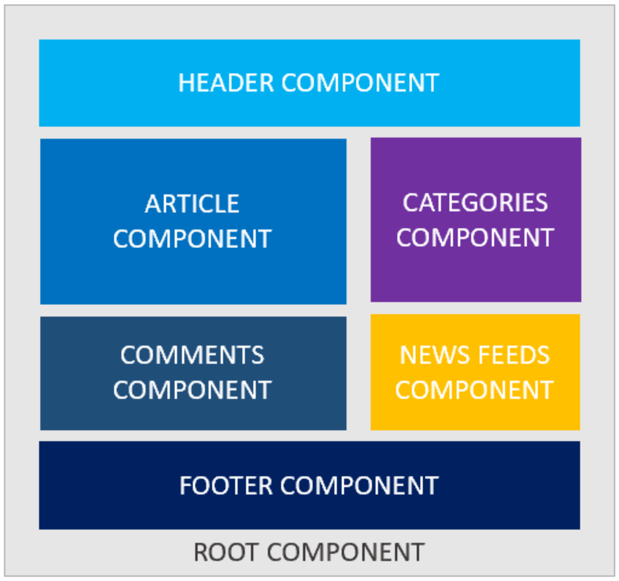
Here we have used {{ }} brackets to use an expression within our template, and you can see the output like this.



```
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, a news feed, and header, the footer can be managed using the angular components as given below:



Q4. What are the Template Statements?

Ans. Template statements will be triggered when an event will be raised using any binding target such as Component, directive, or pure element.

The expression will always be on 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 is supported in Template statements.

Q5. What is Data Binding in Angular?

Ans. Data binding is one of the essential core concepts used to communicate between the Component and DOM.

In other words, we can say that Data binding is a way to add/push elements into HTML Dom or pop the different elements based on the Component's instructions.

There are three types of data binding supported 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 DOM element component and perform various operations.

Q6. What is Interpolation?

Ans. Interpolation in Angular is used to show the property value from components to the template or execute it.

Generally, Interpolation can 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 by using curly braces {{ }}; let's see a simple example.

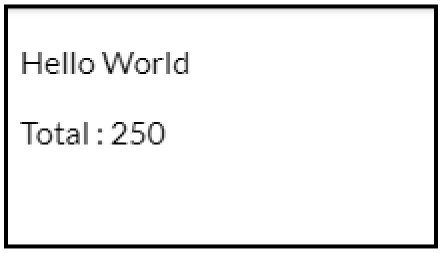
App.component.ts

```
myMessage: string = "Hello World";
```

App.component.html

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

Similarly, we can use Interpolation to the different places or elements like the source of the image, values for the classes, and other usages. The output will look like this.



```
Hello World

Total : 250
```

Q7. What is Property Binding?

Ans. Property binding is generally used to show or update the value of the component variables 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 the value will be reflected at the component level.

Let's see the simple example by taking simple string values from components 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, called Interpolation.

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 events will be used to fulfill the goal of the activity.

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

For event binding, the targeted event will be on the left side, and the event name will be on the right side. The punctuations like (surround the event name), [] 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 the () brackets surrounding it.

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 events 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 most potent 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 and generates the output based on the event, which was triggered.

We will use the punctuation **[( )]** to bind the two-way data for two-way data binding.

It can be happen using the directive called ngModel, which listens 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 corresponding 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. We are binding the textbox's value and returns to the same variable from our template, which holds the current value of fullName.

As we have discussed, it is a combination of two binding mechanisms equivalent to the below statement.

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

Q10. What is Directive in Angular?

Ans. The directive in Angular is a class with the (@) decorator. The directive has 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 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 the Component Directive in Angular?

Ans. A component is ultimately a Component Directive form, and it is a simple directive with its templates.

Q12. What is Structural Directive in Angular?

Ans. The structural directive is used to manipulate the DOM parts by adding or modifying DOM level elements.

For example, we are adding records whenever 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 another directive, but the difference is that we are changing the behavior of elements dynamically.

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

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

@Directive({
selector: '[appAttdirdemo]'
})
Export class AttdirdemoDirective {

    @Input() appAttdirdemo: string;
constructor(privateelref: ElementRef) { }

    @HostListener('mouseover') onMouseOver() {
       this.changefontsize(this.appAttdirdemo);
    }

    @HostListener('mouseleave') onMouseLeave() {
       this.changefontsize('15px');
    }

    changefontsize(size) {
       this.elref.nativeElement.style.fontSize = this.appAttdirdemo;
    }
}
```

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

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

As you can see, 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. There are three types of built-in attribute directives 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 the 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 tasks based on the directive values.

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

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 the 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 the total number of items into the array/list

Q16. What is the use of the 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 a simple example to show a specific div based on the requirement.

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

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

Ans. NgFor directive in Angular is used to iterate the array item; it will be emphasized until it reaches the list's total number.

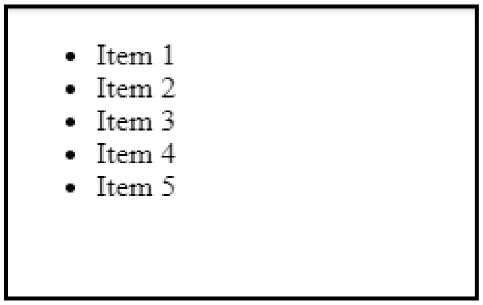
App.component.ts

```
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>
```

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



```
• Item 1
• Item 2
• Item 3
• Item 4
• Item 5
```

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

Ans. NgSwitch is entirely similar to the switch case used with JavaScript to select a single element amongst the different available features 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 is based on the value. A specific test case will be selected.

- **NgSwitchCase**

  NgSwitchCase acts as a specific case. If it matches the expression's value, it will add or remove an HTML Dom element.

- **NgSwitchDefault**

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

Let's see a simple example in which we will pass the static value to the NgSwitch directive and operate 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 an 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';
}
```

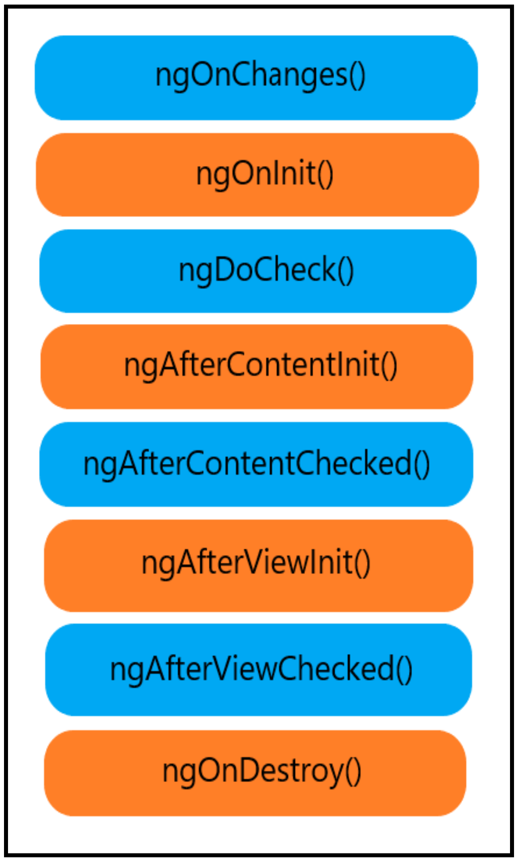
As you can see in the above example, we have used value for expression by default, an option2, when we pass it to the NgSwitch.

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

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

Q19. What are various Angular Lifecycle hooks?

Ans. Angular has its 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, the life cycle events 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 input-bound properties' value is changed.

❖ **ngOnInit()**

It will initialize the Component or a directive and set the Component's different input properties' values. And one important point is that this method is called once after ngOnChanges ().

❖ **ngDoCheck()**

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

- **ngAfterContentInit()**

  This method will be triggered whenever the 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 whether there are changes or not, but this method will still be invoked.

- **ngOnDestroy()**

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

Q20. What methods are used 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. Communication between components 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() decorator?

Ans. This is the standard way to share data from Parent component to child using @Input () decorator, allowing us to get the parent component's input value.

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 will get the counter value.

The first step is to take two different buttons for increment and decrement of 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;
}
```

Our child component has a @input () decorator, which accepts the parent component's input and displays 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() decorator?

Ans. This is another way to share the child's data with the parent component by emitting the event and listening to the parent component.

@Output is a decorator that becomes the output for the parent component and gets the child's message. 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 in the fullName, which is the model value.

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

Now in the 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 fullName using the event implemented into the child component, and 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 is changed at that time, the event will be emitted directly.

This is how we can use @Output and EventEmitter to communicate from the child to parent component.

Q23. How to share data using @ViewChild() decorator?

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

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

One thing to keep in mind about the ViewChild is that we need to implement the 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 of 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 will 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 the child component using ViewChild. Using different child increment methods () 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 the getValue() function, we can get the raw data as the 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 will inject the mydataservice file from where we will send the message.

Our next move is to create a new component called newmessage, from which we send a 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 to send the message. Still, here we will send a new message to the parent component using method changemessage ().

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

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

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

Q25. Do you explain Change Detection in Angular?

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

Using zode.js along with Angular, it detects the async events and allows Change Detection to the components.

It's always traversing from the UI tree from the root component to the child component and updates the DOM based on the updated state.

There are two different Change Detection strategies:

- **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, then their value has turned into heap memory or not.

We will see the simple example using the OnPush strategy. Then we will get to know about the mechanism.

Let's assume we have 3 different components called app, child, and subchild and 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 will 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 the child component, and the code snippet looks 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 the @angular/core package.

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

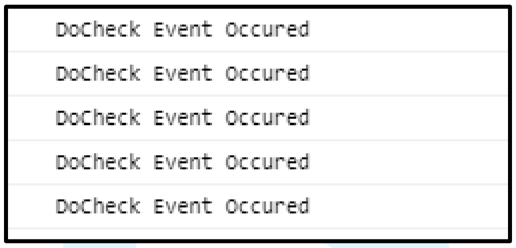
From the parent component, we are getting the list of the users, and at the same time, we are passing these records to the subchild component and it will look like this.

```
// 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 will render five different users, and you can see that we have used the ngDoCheck() lifecycle hook, which is used to detect every change in the child component.

Whenever we open the 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 the OnPush Change Detection strategy works by enabling the immutable state object, directly reflecting on the performance.

Q26. Do you explain Content Projection in Angular?

Ans. Content Projection in Angular is one of the essential 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.

Content Projection will work by providing markups as the element's content, and then we can project our content into the specific div or a particular position in the template.

We can implement Content Projection using the **<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 two components: 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 the <app-child> directive, 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 the <ng-content> directive to project the content. For this example, I have used a simple line of string inside the <app-child> directive, which will render inside the projected element.

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

We can also use <ng-content> along with the selector to project the content more precisely.

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

In our template file, we have used the tag <footer>; in the same way, we can also create another 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 the "select" attribute and ng-content, which is used to define the selector for our slot.

We can also use a selector with a CSS class; below is a simple example of that.

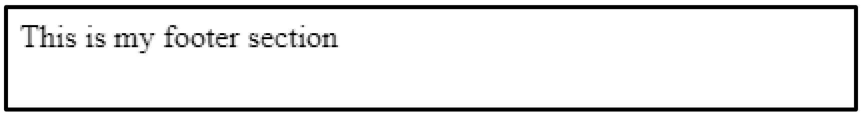
```
<!-- app.component.html -->
<app-child>
  <div class="my-css">
    This is my footer section
  </div>
  <div class="my-css1">
    This is my footer section 1
  </div>
</app-child>
```

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 content of the specific element.

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



```
This is my footer section
```

It means that only the 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?

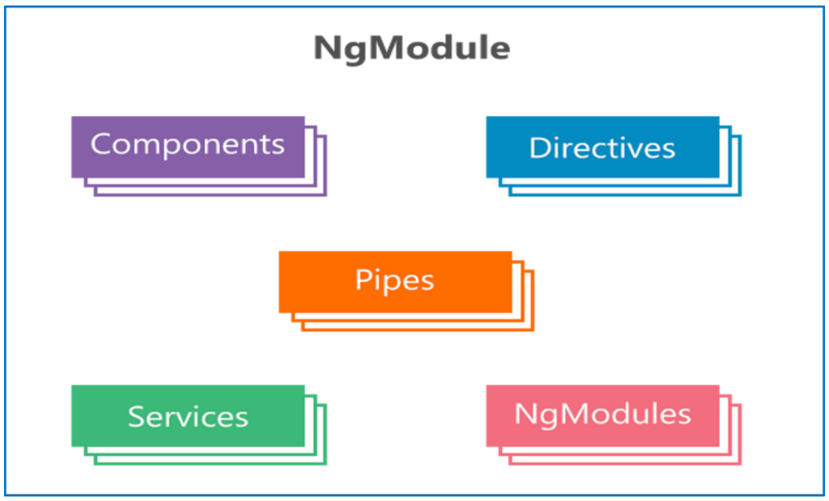
Ans. The Angular module is an essential part of the application, which is used to organize our angular app differently.

Or we can say that Modules play a vital role in structuring our Angular application.

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

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

Module files 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 using the below ng command.

```
ng generate module mycustommodule
```

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

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

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

Q4. Do you explain the Angular Module basic structure?

Ans. Every module file we create has a specific structure that 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 in 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

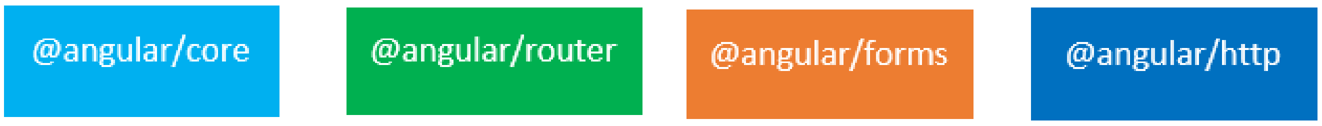
Q5. What are the multiple NgModule metadata properties?

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

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

Q6. What are the built-in Modules?

Ans. Angular has built-in library modules starting with the @angular as a prefix. Built-In Library & third-party modules can be installed using npm manager. Integrated modules, components, services, directives, etc., can be imported using integrated 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 the ng command along with the appropriate module name like this.

```
ng generate module myfeaturemodule
```

It will create the feature module file and contain 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 modules can be written into a 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 and the Modules 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 to create a Module for the routing and the standard module file.

After running the 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 the Entry Component?

Ans. The bootstrapped component is an entry component that loads the DOM during the bootstrapping process. In other words, we can say that entry components are the ones, 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 that 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 the constructor of the component, directives, and other classes.

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

- **Using providers into the Module file**

  We can add services to the provider's metadata like this.

```
providers: [
   MydataService
]
```

- **Using providers into the Component file**

  It 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. We need to work with data during the web application development by getting it from the database or another platform. We get the data into a specific format, and we need to transform it into a suitable form. 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. It should implement the PipeTransform interface, which accepts the input value and will return the transformed value.

To create a new pipe, we can use the below command.

```
ng generate pipe mycustompipe
```

Q2. How to create an Angular Pipe?

Ans. When we create a new pipe, a new file will be made, 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 a custom pipe, we should implement an interface named PipeTransform, which accepts input and returns the transformed value using the 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 pipes are available to use for data transformation.

Q5. How to use Pipes?

Ans. To use in-built pipes in angular, we can use a symbol pipe followed by the pipe name described below.

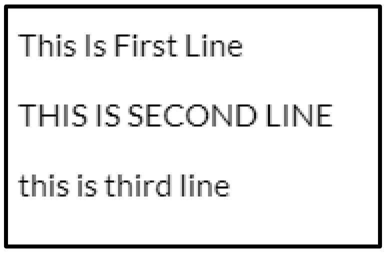
Syntax:

```
{{ value | pipe_name }}
```

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

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

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



```
This Is First Line

THIS IS SECOND LINE

this is third line
```

Q6. What are the different types of Pipes in Angular?

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

- Pure pipes
- Impure pipes

Q7. What is Pure Pipe?

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

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

Q8. Do you explain Pure Pipe in Angular?

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

Mycustompipe.pipe.ts

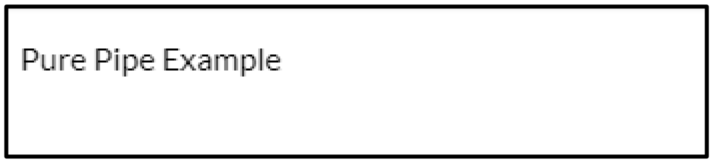
```
import{ Pipe, PipeTransform } from '@angular/core';

@Pipe({
   name:'mycustompipe',
   pure:true
})

Export class MycustompipePipe implements PipeTransform {

transform(value: any, args?: any): any {
   if(!value) {
    returnvalue = 'Value Not Found';
   }
   return value = value;
  }

}
```

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

App.component.html

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

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

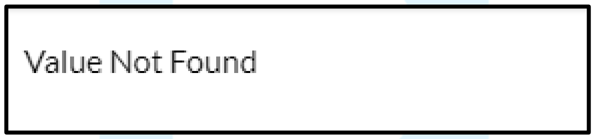


Pure Pipe Example

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

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

Our output now looks like this.



Value Not Found

Q9. Do you explain the Impure Pipe in Angular?

Ans. We know that every pipe we create is a pure pipe by default, but we can also make impure pipe by providing pure to false as described 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 we can set it to true or false using the pure option, so if we want to set a pipe to impure, then the value would be false, just like this example.

Q10. What is Pipes chaining?

Ans. Sometimes we need to use multiple transformations for the data. The pipes' chain is used; using the pipe chaining 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 will transform the data 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 the HTML template, we will 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 the (:) 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 pages from the current page. When we try to enter a URL in the browser, it navigates to a specific page and performs appropriate functions implemented into the pages.

Routing is a crucial task for any application because it may have many pages to work with.

Q2. How to compose a 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 an app directory, 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 correctly transforms the URL and also manages different URL transitions.

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

Q4. How to use Routing in Angular?

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

We need to import RouterModule like this into our root module file to use Routing in our Angular application.

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

Q5. What is the Route in Angular?

Ans. The route is the Array of different route configurations and contains the various properties listed below.

Property	Description
Path	String value which represents route, matcher
pathMatch	Matches the path against the path strategy
Component	The component name 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 features are supported like data, canActivate, resolve, canLoad, canActivateChild, canDeactivates.

Using these different properties, we can perform many operations like loading the child route, matching the different route paths, specifying the route path for a specific component, etc.

Q6. What are the different types of methods in RouterModule?

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

1. **forRoot():** It performs primary navigation and contains the list of configured router providers and directives. You can find a 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 router directives and provider registered routes.

Q7. What isRouterLink?

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 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. Several properties are available to use with RouterLink, which are listed below.

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

Q9. What are the various Router events?

Ans. When the navigation process started, at that time, different events will be triggered at a specific point in time using the property Router. Events.

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, 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 the one below in 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 standard 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 is clicked by the user and using this attribute. We can change the 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;
}
```

When we run our demo, we can see that the activated link's color is changed directly from blue to red.

Q13. What are the different Routing Strategies in Angular?

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

1. **Path Location Strategy**

   By default, in Angular, the 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 the Hash location strategy when Hash (#) will be appended to the URL.

Using these two routing strategies, we can implement routing in our Angular app because whenever a URL is changed, our application sends the request to the server and will render appropriate data based on the component loaded.

Q14. Do you explain the Path location strategy in Angular?

Ans. Whenever we create an Angular application, the default Path location strategy will be implemented, so we don't need to put the extra effort from our side.

It used an HTML5 history API called pushState, which allows us to push the different URLs 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 quickly and modify different URL strings based on the requirements.

And one of the most important things to keep in mind is that the browser would not request 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 the Path location strategy was implemented in our application.

Q15. Do you explain the Hash location strategy in Angular?

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

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

While implementing the 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 a URL like a **localhost: 4200/Employees/#/marketing/165,** then a request to the server will be only to **localhost: 4200/Employees**.

In short, the hash location strategy enables us to implement routing when we don't want to confuse the server by sending hash fragments, 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 standard 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. Still, we can automatically generate the routing module using CLI command ng generate; let's see how we can do this.

```
ng generate module <module_name> --routing
```

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

Q17. What are the different types of module loading?

Ans. There are the following types of loading:

- Pre-Loading
- Eager Loading
- Lazy Loading

Q18. What is Pre-Loading?

Ans. Preloading is an entirely different thing than lazy loading and eager loading; preloading means loading the module after any eager module is loaded. After an eager module is loaded at startup, the router looks for the unloaded modules and can be preloaded.

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

Now Home is the first page when our app will be executed, so the remaining two modules will continue unloaded. Still, at some point in time, the user may want to contact the business owner to contact them using the contact page to be ready to load after the Home page.

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

Q19. What is Eager Loading?

Ans. We have seen an example of how preloading works, but there is a difference between them. Is Eagerly loaded module will be executed at the 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 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 components: Home, About, and Contact, so all the modules will be loaded at the application startup whenever the application is loaded.

Q20. What is Lazy Loading?

Ans. We are loading every module at once, but do you know that the application's performance may decrease? For that, we need to load only the necessary modules 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 in the app bundle size.

Here in this question, we are talking about Lazy loading, in which the current screen will only be loaded, and the 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.

The 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 and performance by reducing initial loading modules.

Q21. How to use dynamic imports in Routing configuration?

Ans. Previously, in the older version of Angular, the syntax for importing the routed was like this.

```
{
        path: '/test',
        loadChildren: './test/test.module#TestModule'
}
```

But now we can import the routes from Angular 8 like this.

```
{
        path: `/test`,
        loadChildren: () => import(`./test/test.module`).then(m => m.TestModule)
}
```

7 Angular Forms

Q1. What is Angular Form?

Ans. We need to handle the different inputs and files like handling user login, registration, getting data for a different purpose, and other forms while developing the applications.

At that time, Angular forms are used to handle the user inputs. For that, there are two different approaches provided, which are listed below.

1. Reactive Forms
2. Template-driven Forms

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 use different form controls to get the user inputs; based on that, we have some basic building blocks.

- **FormGroup**

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

- **FormControl**

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

- **FormArray**

  Manages the status and value for Array of the form controls

- **ControlValueAccessor**

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

Q3. What is Reactive Form in Angular?

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

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

To use Reactive forms, we need to import modules as described below.

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

And then, we need to add this module into an import array like this.

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

Q4. What is Template-driven Form in Angular?

Ans. This is the most straightforward approach to use forms in Angular, and it will be useful if you have simple forms that can be managed within your template, but they cannot be scalable compared to Reactive forms.

In Template-driven forms, Angular will create the models, i.e., FormGroups and FormsControlsand, thenitusesngModel, to enable the template-driven forms approach.

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

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

And also need to add an inside import 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 the forms value property	It allows one-way and two-way data binding to pass the value of the form
Behaviour is Synchronous	Behaviour is Asynchronous

Unit testing will be much easier	It May take more effort for unit testing
We can achieve excellent code readability	Unable to get proper readability due to no small amount of code implemented in the template
It is challenging 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 other inputs and implement validations.

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 a component like this.

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

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

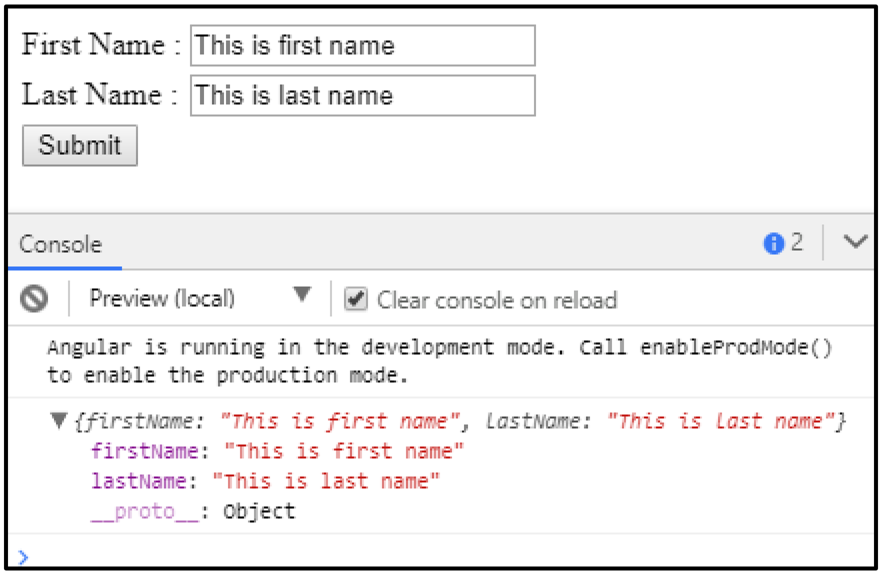
```
demoForm = new FormGroup({
   firstName :new FormControl('Manav', Validators.maxLength(10)),
   lastName:new FormControl('Pandya', Validators.maxLength(10)),
 });
```

Here in this example, we have two different form controls, firstName, and lastName, which are 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.


First Name : This is first name
Last Name : This is last name
Submit
Console
2
Preview (local)
Clear console on reload

```
Angular is running in the development mode. Call enableProdMode()
to enable the production mode.
▼{firstName: "This is first name", lastName: "This is last name"}
    firstName: "This is first name"
    lastName: "This is last name"
    __proto__: Object
```

Q8. Do you explain Form validations in Angular?

Ans. Form validation is a crucial part of any web application where we need to validate the inputs against malicious data. We can say security vulnerability and other scenarios like data are complete and accurate or not, and data is the 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 as follows:

- required
- min
- max
- required
- 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 a straightforward 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 be possible that these inbuilt validators would not help us accomplish our validation requirements, so we can also create our custom validations.

Custom validation is nothing more than a regular function; let’s see one simple example in which salary should be more than 10000 and implement a 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 that 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 ]),
});
```

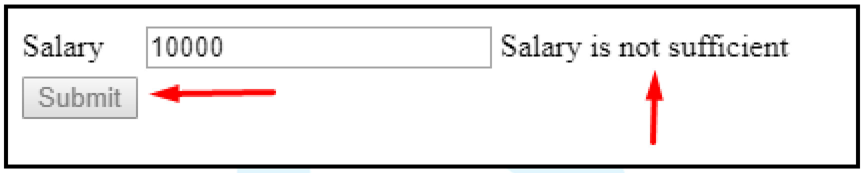
As you can see that we have used two different validators, first is required validator and another is our custom validator, find the template code below.

```
<form [formGroup]="demoForm" (ngSubmit)="submit()">
<table>
<tr>
<td>
<label for="salary">Salary</label>
</td>
<td>
```

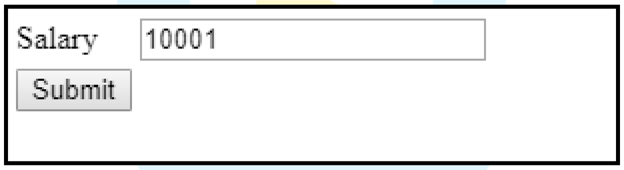
```
<input type="text" name="salary" id="salary" [formControlName]="'salary'"><br/>
</td>
<td>
<div [hidden]="!demoForm.get('salary').hasError('isSufficient')">
   Salary is not sufficient
</div>
</td>
</tr>
<tr>
<td><input [disabled]="!demoForm.valid" type="submit"></td>
</tr>
</table>
</form>
```

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

I will use a salary of 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 do get the submitted Reactive form values?

Ans. To get the user's values, 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 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 we use a reactive form approach in Angular. We can do this using the reset () method, as described below.

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

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

8
Dependency Injection

Q1. What is Dependency Injection in Angular?

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

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

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

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

Here in this constructor, we will inject MyDataService, so whenever an instance of the component is created, it determines which service and other dependencies are needed for an element by just looking at the constructor's parameters.

In short, it shows that the current component ultimately depends on MyDataService, and after instance of the component will be created. The 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 a few advantages to using Dependency Injection with Angular, which are listed below.

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

Q3. What is an Injector in Angular?

Ans. The injector in Angular is used to inject the service instance into the class. Using an injector, we can register our service to the module, component, and service used by all the application components.

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

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

Ans. To implement DI, we should have one provider of service, which we are going to use. It can be anything like registering the provider into modules or components.

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 the @Injectable () decorator; below is a 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 allow our service to be accessible to all of the components relevant to the current module In which we are registering our service.

You can get an idea of 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. We provide it to the array of providers to 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. Whenever a new instance of the component is created, we will get the service's latest instance.

```
@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 a Dependency Injection token?

Ans. When we work with Dependency Injection in Angular, every injector maintains the internal token 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 ideas by observing the 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){}
```

9

Observables and RxJS

Q1. What are Observables in Angular?

Ans. Observables are a way to populate the data from the different external resources asynchronously.

Observable replaces the promises in most cases like HTTP. The significant difference between the Promises and Observable is that Observable can observe the stream in different events.

Observable is declarative. It means that when we define a function for publishing, it won't be executed until any consumer subscribes.

Observable's asynchronous behavior is used extensively along with Angular because it is faster than Promise and can be canceled at any time.

Q2. What is RxJS?

Ans. RxJS stands for Reactive Extensions for JavaScript, and it is a popular library used to work with asynchronous data streams.

While working on large and complex applications, RxJS can help developers represent multiple asynchronous stream data and subscribe to the Observer's event streams.

When an event occurs, at that time, Observable notifies the subscribed observer instance.

To use RxJS in an Angular application, there is an npm package, and by installing a package, we can import a library like this.

```
import { interval } from 'rxjs';
```

Interval is just a function in the RxJS library, but it has a different set of functions that we can use in our application.

Q3. What different functions are provided by the RxJS?

Ans. RxJS has a collection of different functions, and a 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()

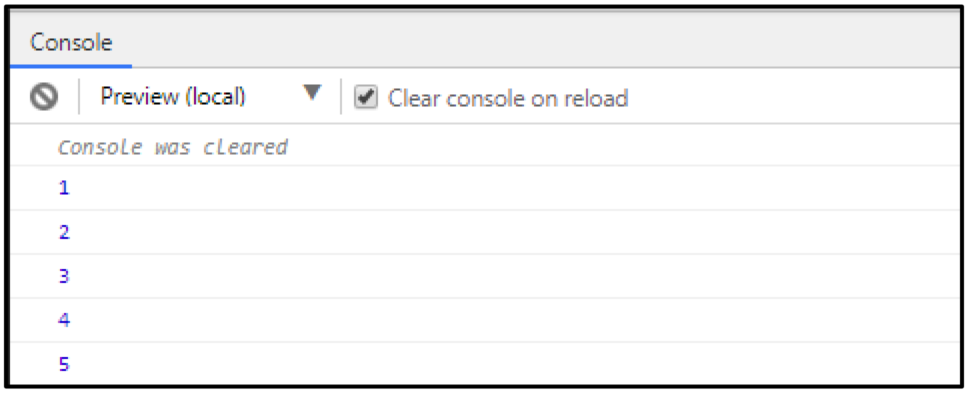
Q4. Do you explain RxJS operators in Angular?

Ans. We have seen different functions in RxJS, and now next question may arise how to use any functions or operators in the Angular app, below is the simple example using the 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 in the above example, I have imported map operators from the package rxjs/operators, and the map () operator iterate the array items.

Then we are subscribing to the same list using the subscribe () function. When we open the console, we can see the output like this.



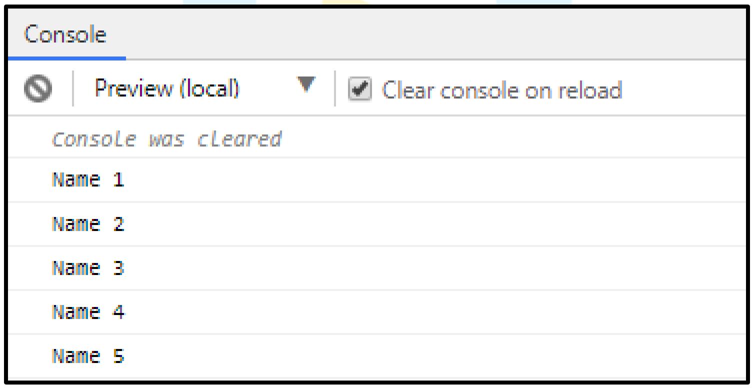
```
Console was cleared
1
2
3
4
5
```

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.

After mapping the data using the map operator, we will be able to iterate the array with a single property, and the output looks like this.



```
Console was cleared
Name 1
Name 2
Name 3
Name 4
Name 5
```

Q5. What is a Subscription?

Ans. Subscription is a kind of disposable resource, such as the execution of the Observables.

While working with the Observables, the subscription has one method called unsubscribe(), which is used to release the different resources held by the subscription, and it won't take arguments.

For example, we care calling the GET api for getting the employee details using the services.

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

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

    employee: string;
    subscription: Subscription;

    constructor(private service: MyDataService) { }

    ngOnInit() {
        this.subscription = this.service.getEmployees()
            .subscribe((employee: string) => this.employee = employee);
    }

    ngOnDestroy(): void {
        this.subscription.unsubscribe();
    }
}
```

In this example, we were getting the data using the service method and subscribed to it. But it is custom observable so that we can manually unsubscribe the Observable when our component gets destroyed.

The best place to unsubscribe from 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, we need to unsubscribe it using one of the ways, but we will do it using an async pipe like this in this example.

```
<!-- app.component.html -->
<div>
  {{ dataSubscription | async }}
</div>
```

As you can see that we have used a pipe symbol along with the 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. Do you explain Error handling in Observables?

Ans. Working with Observable is the right choice, but we need to manage unhandled errors when handling the error.

Using try/catch is not enough to process the Observables, for that we can use errors callback on the Observers like the 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 callback, which is used to get the complete error information, and in the same way, we can get the completion handler function to get to know about the success confirmation.

Q8. Do you explain catchError and Finalize operators?

Ans. The catchError and Finalize are the operators in RxJS, and it is used to handle the errors in an Observable sequence. Let's see the simple example using catchError.

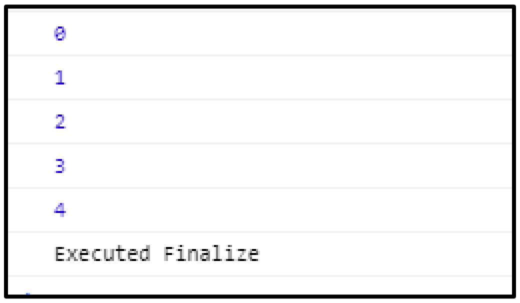
```
// Throw custom error
const error = throwError('I AM ERROR');

// 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 mistake will be handled by the catchError operator and will provide the error details.

Finalize operator is the same as the finally block that we have used along with the try/catch, which is used to execute the callback function when the observable is 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 to take and finalize the operator. The first five values will be printed and, in the end, using finalize. The completion message will be written into the console like this.



```
0
1
2
3
4
Executed Finalize
```

10
Angular Services and HTTP

Q1. What is Angular Service?

Ans. While developing the app, we need to change or manipulate data continuously, and to work with data, we need to use a service, which is used to call the API via HTTP protocol.

Angular Service is a simple function, which allows us to create properties and methods to organize the code.

We can use the below command to generate services, making a service file and the default service function structure.

```
ng generate service mydataservice
```

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

Ans. When we create a service using the ng command, it will create a service file like below.

Mydataservice.service.ts

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

@Injectable({
   providedIn:'root'
})

Export class MydataserviceService {
constructor() { }
}
```

Q3. Why use Angular Services?

Ans. Services in Angular are the primary concern for getting or manipulating data from the server. The services' immediate use is to invoke the function from the different files like Components, Directives, etc.

Usually, services can be placed separately 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 use different service functions into the integral component.

One of the main advantages is Code Reusability. When we write any function inside the service file, the different components can be used to don't need to write the same methods into multiple files.

Q4. How to use service in Angular Components?

Ans. After creating the service, the next step is to consume different service functions into the various components.

The first step is to import the service file from the respective folder location to use the component's services.

```
import{ EmployeeService } from './services/employee.service';
```

The next step is to inject the service using dependency injection into the Component's constructor like this.

```
constructor(private myService: EmployeeService) {
}
```

And using myService, we can access the different functions and properties implemented inside EmployeeService.

We can call the service function like this.

```
this.myService.GetEmployees();
```

We can call the services' different functions and communicate with the server by providing sufficient request data.

Q5. What is Singleton Service?

Ans. Singleton Service in Angular means is providing service reference to the 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.

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 is a total of 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 theHttpClientModule into the root module file, now we will be able to inject HttpClient into our service which looks like this.

```
import{ Injectable } from '@angular/core';
// Importing HttpClient
import{ HttpClient } from '@angular/common/http';

@Injectable()
Export class ConfigService {
// Injecting HttpClient
constructor(private http: HttpClient) { }
}
```

Using HTTP, we will use different HTTP request methods like Get, Post, Put, Delete, and other methods into our whole service file.

Q10. How to make error handling in HTTP clients?

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 whether the request was successful or failed to retrieve data. The reasons may be different, like internet connectivity, internal server error, insufficient request data, etc.

We should handle the error, and below is a simple example where we will 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 be possible that anything went wrong, then we will get the complete error details and the error code.

Q11. What is Interceptor?

Ans. The inception of the HTTP request and response is one of the significant parts of the HTTP package. Using an inceptor, we can transform the HTTP request from our angular application to the communication server. In the same way, we can also modify the response coming from the server.

If we don’t use the interceptors, we need to explicitly implement the tasks logic whenever we send the server's request.

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 implemented the interface HttpInterceptor which is an essential part of implementing the interceptor, you may be confused about HttpInterceptor. Still, 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. It is an interface used to use and implement the intercept method and manage the related techniques.

While working with the web application, we will send the request to the data for data communication, but before submitting a request to the server, we need to modify and manipulate each request.

To implement the HttpInterceptor interface, we can use simple syntax, as explained below.

```
Interface HttpInterceptor {
intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>>
}
```

Above code, the snippet is a simple example of interface HttpInterceptor which uses the intercept method to handle the requests.

Q14. What is the intercept method in HttpInterceptor?

Ans. When we send the request to the server and get the server's response, we can transform the response using the intercept () method.

We can use interceptors to attach authentication details to the request header automatically, or simply, we can 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 an authorization header using HttpInterceptor and intercept () to handle the request manipulation.

Q15. What is the next object in Interceptor?

Ans. We have seen the simple example of the HttpInterceptor in which we have used the next object, which is used to represent the next interceptor in a queue to be transformed.

The next object will be used whenever we have the interceptor's chain, and by using the 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 it into the provider's array, and by using the 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 HttpHeader in Angular?

Ans. While sending a request to the server, sometimes we need to pass some extra headers 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 an 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 the authorization header along with our request.

Q18. What are the 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. There is no direct way to modify the request header in Angular, but we can change it explicitly using the set () method described below.

```
httpOptions.headers = httpOptions.headers.set('Authorization', 'my_token');
```

Here I have modified authorization explicitly using the set () method of headers.

11
Angular Testing

Q1. What is 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. By using Unit testing, we can test different components 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 application's unit.

Q2. How to set up an Angular app for the Jasmine test?

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

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

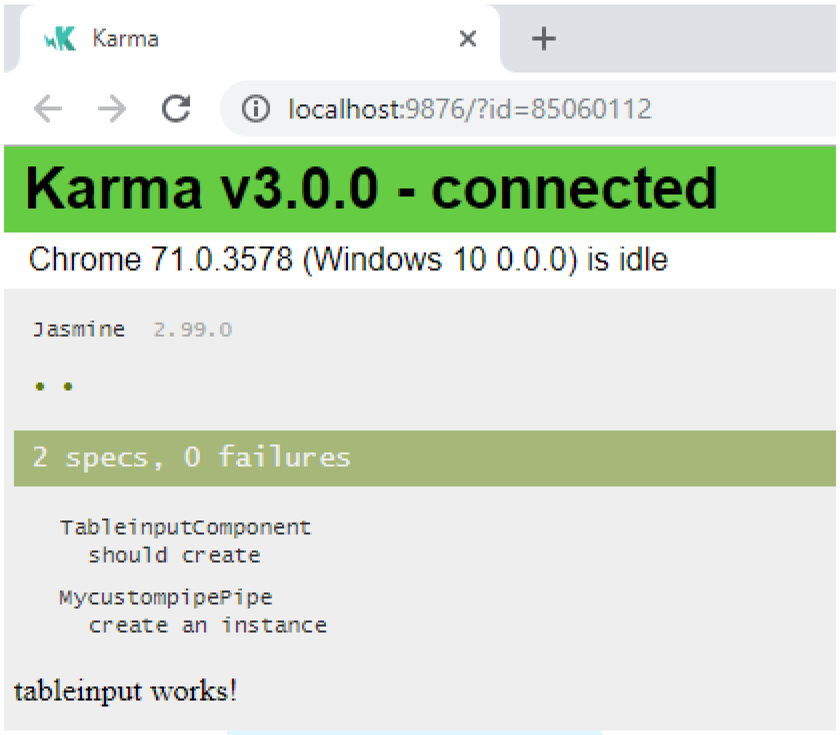
One of the advantages of using Jasmine is that it is dependency-free and doesn't require DOM.

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

We can run the below command to 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.



```
Karma v3.0.0 - connected
Chrome 71.0.3578 (Windows 10 0.0.0) is idle
Jasmine 2.99.0
• •
2 specs, 0 failures
  TableinputComponent
    should create
  MycustompipePipe
    create an instance
tableinput works!
```

I have two different testing specifications in this example project, which can reside inside the tableinput.component.spec.ts, and the code snippet looks 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 to run different test cases into the UI.

Q3. What is Karma?

Ans. Karma is a kind of tool 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 make changes to the files. It also allows the facility to run the tests into different devices like phones, tab, monitors, 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 the Test entry file in Angular?

Ans. Our application will be configured for testing using Jasmine and Karma in Angular, but we may question how Angular CLI configure the karma into our app.

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 the **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 in 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. How to test Angular Components?

Ans. The component is a fundamental building block in the Angular app, and it contains a Template and a stylesheet.

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 specifications 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 detail.

- **TestBed**

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

It is a high-level testing framework that allows us to test the different environments.

- **Expect()**

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

- **Matchers**

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

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.

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

Q6. What are 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(regexp).toBeRegExp();
- expect(string).toBeEmptyString();
- expect(string).toBeHtmlString();
- expect(string).toBeIso8601();
- expect(string).toBeJsonString();

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

Q7. How to test service in Angular?

Ans. Service in Angular is crucial for data communication; thus, testing a service file is essential.

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

For example, we have a service spec file called the **mydataservice.service.spec.ts** file, and the code snippet looks 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 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 the above test case, we are using the service method addNewUser () and trying to add a new user into the list, and at last, we are expecting that the 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 the Directive in Angular?

Ans. The directive in Angular is a good part, and we need to test the Directives for their behavior, so 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 will use HostListener, which is used to listen to elements or components' events.

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 implemented.

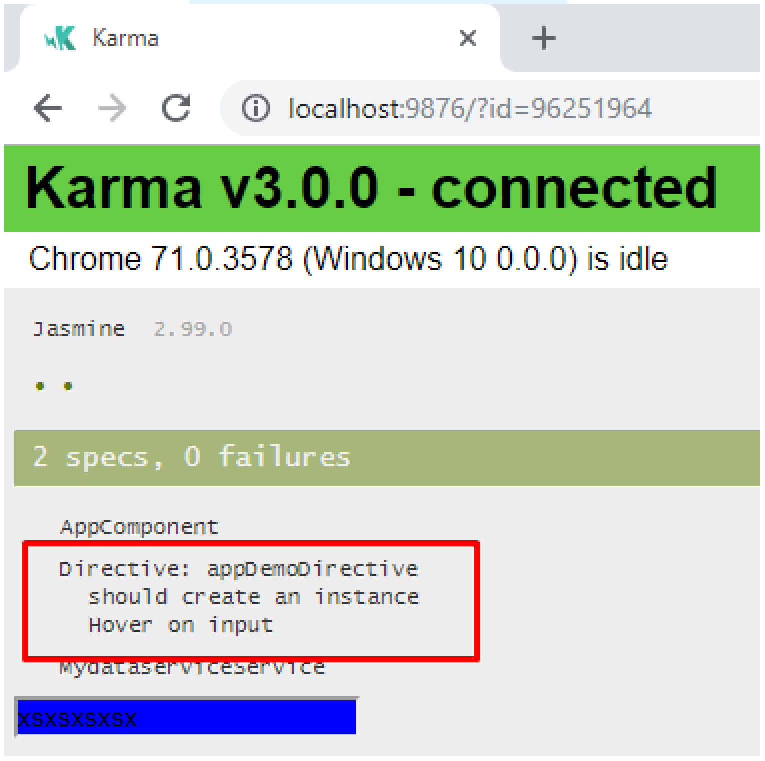
- **Should create the instance**

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

- **Hover on input**

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

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



```
Karma v3.0.0 - connected
Chrome 71.0.3578 (Windows 10 0.0.0) is idle

Jasmine 2.99.0

• •

2 specs, 0 failures

AppComponent
Directive: appDemoDirective
  should create an instance
  Hover on input
MydataserviceService
xsxsxsxsx
```

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 custom ones.

We can also test Pipes just like we do with Component and Service; below is a simple example of the pipe where we convert 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 custom pipe file, we are just transforming the text to lower case; now, let's use our custom pipe in our template like this.

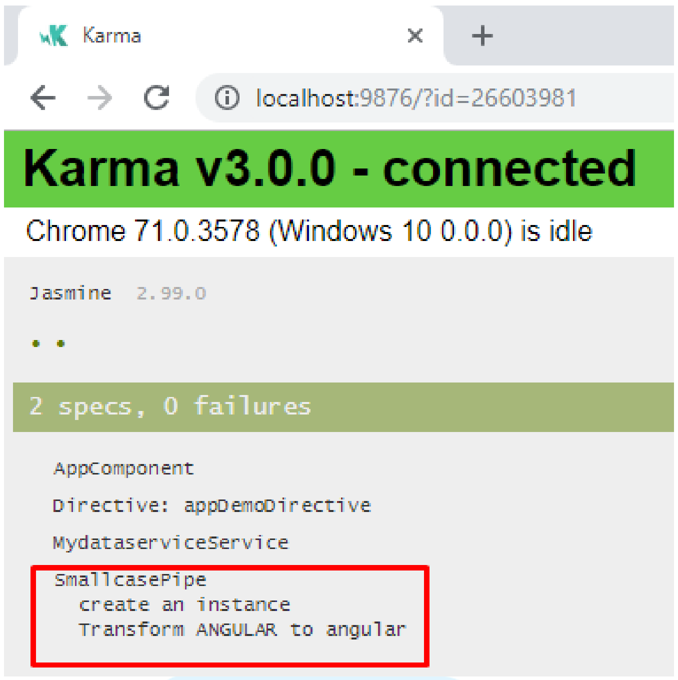
```
<!-- app.component.html -->

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

After running our app, we will get the expected output by converting text to lower case now. Let's write test specifications 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.



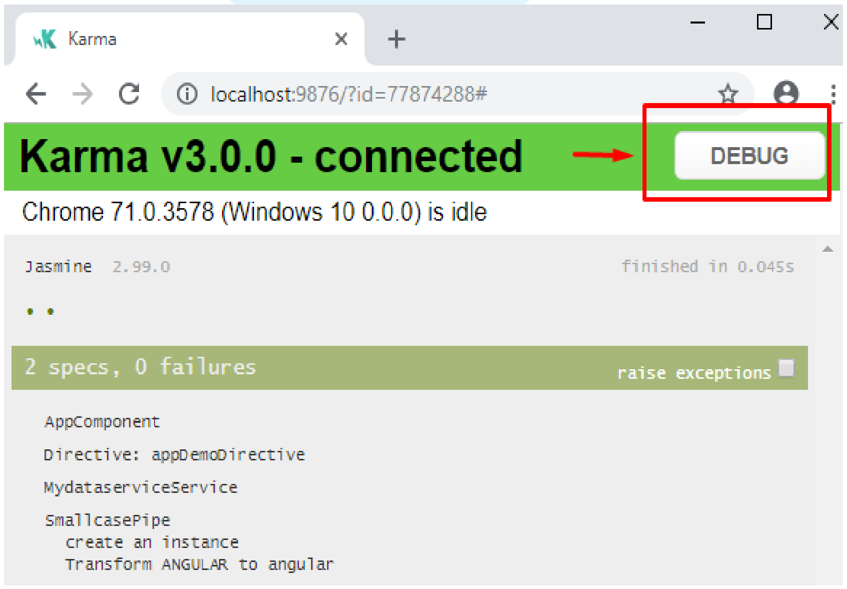
```
Karma v3.0.0 - connected
Chrome 71.0.3578 (Windows 10 0.0.0) is idle

Jasmine 2.99.0

• •

2 specs, 0 failures

  AppComponent
  Directive: appDemoDirective
  MydataserviceService
  SmallcasePipe
    create an instance
    Transform ANGULAR to angular
```

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.

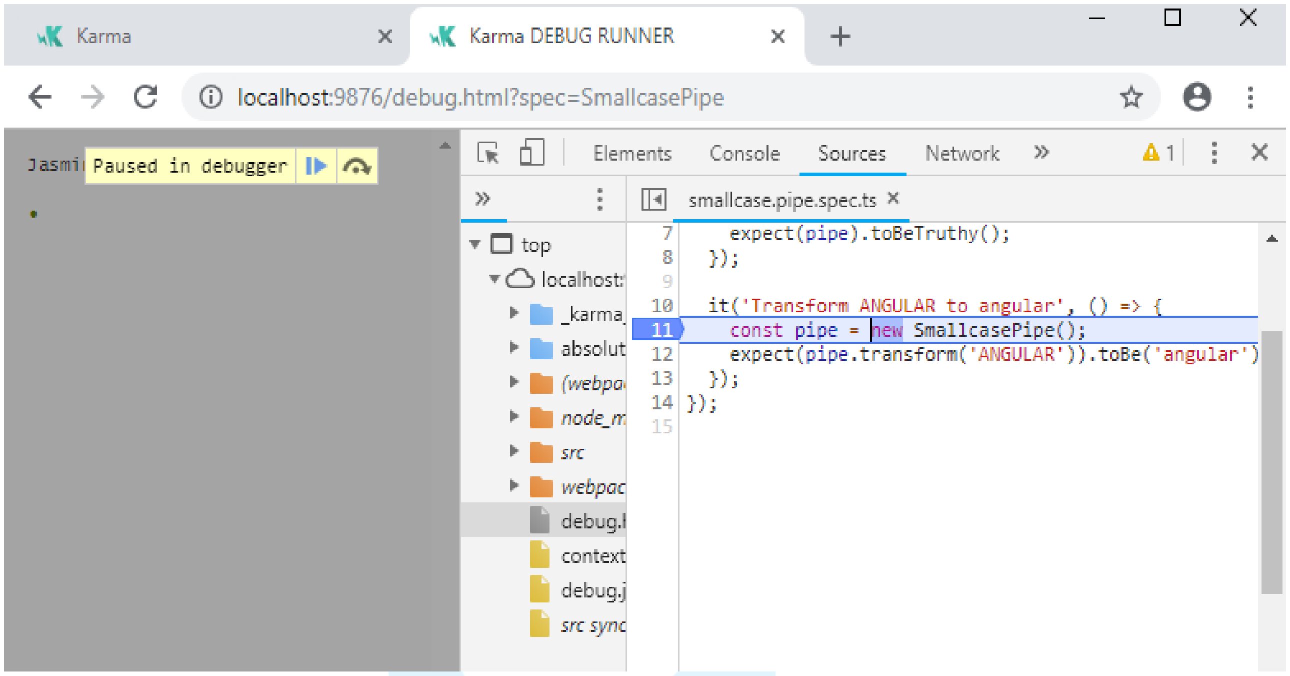
For that, we need to follow a few steps as described below.

- **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.



```
Karma v3.0.0 - connected
Chrome 71.0.3578 (Windows 10 0.0.0) is idle

Jasmine 2.99.0                                    finished in 0.045s

• •

2 specs, 0 failures                              raise exceptions

  AppComponent
  Directive: appDemoDirective
  MydataserviceService
  SmallcasePipe
    create an instance
    Transform ANGULAR to angular
```

- **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 updating the browser window, it will stop at a specific breakpoint.



```
    expect(pipe).toBeTruthy();
  });

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

12

Deployment

Q1. What is Deployment?

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

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

The deployment consists of several activities that 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 that 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 the below command.

```
ng build –prod
```

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

Q3. Do you explain the procedure for the Deployment in Angular?

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

- **Step 1:** Generate the production build for the application by using the 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, we are not working in runtime production mode, and 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 the production build of a project, and if you want to remove this info, you can go to the main.ts file and use the enableProdMode() method.

```
// 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.

We can also try another option as we have discussed already using the --prod flag while generating the production build using the 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 significant difference is that Build will process and generate the artifacts. It will reside in a new folder that 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 in the development mode.

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

Ans. We may have questioned why we should always use the **/dist** directory, but 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 the 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": []
},
```

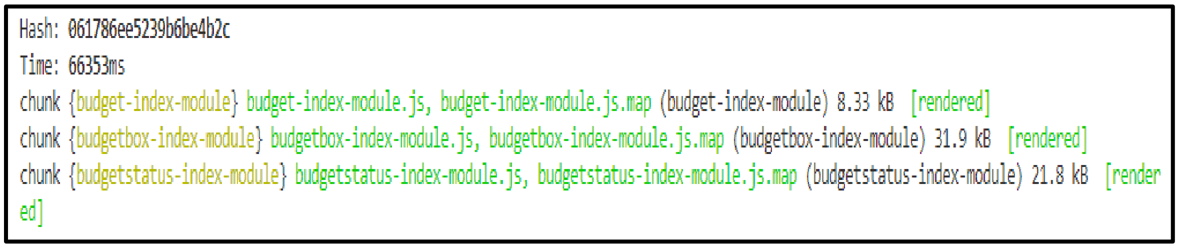
Just change the folder and project name into the options and build the angular app, now we can see that the output files will reside in our custom folder.

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 an output like this for our module files.



```
Hash: 061786ee5239b6be4b2c
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  [render
ed]
```

Q8. What is Builder API in Angular 8?

Ans. Application deployments are a crucial part of web development. Hence the Angular Firebase is one of the trusted platforms to host an application.

Angular CLI 8.3.0 supports the Firebase deploy command, which is used to deploy the app directly to the cloud. Below is the command for that.

```
ng run app-name:deploy
```

Using the above command will build our Angular app and also deploy the app to the hosting provider.

Q9. What is Firebase support in Angular 8?

Ans. With the release of Angular 8, the version supports the Firebase to deploy the Angular application with the production variant.

To deploy the Angular app using Firebase, we have to follow a few steps before the actual development. Thus you can follow the official link, which is given below.

https://angular.io/start/deployment

We can use the following command for the deployment.

```
ng run [PROJECT_NAME]: deploy
```

13

Ivy Rendering Engine

Q1. What is Ivy Rendering Engine?

Ans. Ivy Rendering Engine is one of the well-known updates released in Angular 9. It is a compiler, and a runtime engine used to run the Angular app with the smallest bundle size that directly improves the build performance.

It is one of the major updates from Google that focuses on reducing component file size that can help the small size application and the medium-scale application.

The angular Ivy rendering engine is now a default feature. It means we don't need to enable it just like we do in the Angular 8 version.

Q2. What are the features of Ivy?

Ans. There are numerous features introduced in Ivy rending engine, which are given below.

- Smaller bundle sizes
- Better debugging
- Faster testing
- Improvements in Internationalization
- Improved CSS class and style binding
- Improved type checking
- Improvements in build errors
- Improvements in build times, enabling AOT on by default

Q3. How Ivy provides a smaller bundle size?

Ans. Ivy is the all-new Angular renderer engine that promises smaller bundle sizes when building a project and faster compilation. Onwards, Angular apps will load faster for people, something we all strive for in a mobile-first world using Angular.

Bundle size is decreased because it uses Ivy official instruction, collecting tree-shakable runtime instruction for the rendering.

Q4. How Ivy provides a better debugging experience?

Ans. When we run the Angular applications in debug mode, The Ivy rendering engine exposes a set of new capabilities on the global 'ng object' that allows developers to manually call the different methods, update the state, and trigger change detection, and access instances of their components.

The different set of functions are list below.

- ng.applyChanges
- ng.getDirectives
- ng.getComponent
- ng.getListeners
- ng.getHostElement
- ng.getContext
- ng.getRootComponents

Q5. How to enable Ivy into the older versions of Angular i.e. < 9?

Ans. In the older version of Angular, like Angular 8, we have to enable the Ivy rendering engine manually, as explained below.

```
"angularCompilerOptions": {
    // To enable ivy
    "enableIvy": true
}
```

We have to enable the above option into the file called tsconfig.app.json, and after the configuration, we will be able to load the Ivy rendering engine for our Angular application.

Q6. What is tree shaking?

Ans. The tree shaking feature allows us to remove or ignore the unused code in the Angular application that directly reflects the smaller bundle size and overall application performance.

With the locality and tree-shaking, we can get enormous benefits like smaller bundle size, feasible web development, better app performance.

Ivy engine does not reflect the developers directly. Still, it helps us build a smaller bundle size for the application. The Ivy rendering engine does all the background operations without affecting the day-to-day app development.

Q7. How to improve the debugging experience using Ivy?

Ans. Ivy provides the different sets of tools by which we can have a better-debugging experience, and the Ivy APIs are now added to the global "ng" object.

The core global package @angular/core/global includes few functions which are listed below.

- ng.getComponent
- ng.applyChanges
- ng.getDirectives

- ng.getContext
- ng.getHostElement
- ng.getInjector
- ng.getRootComponents
- ng.getListeners
- ng.getOwningComponent
- ng.getHostElement

For example, we have one component called the test component, so if we want to get the instance associated with the child component, then we can use the "ng.getComponent()" global function.

Q8. How opt out of Ivy from Angular 9?

Ans. The Ivy renderer engine is now enabled by default in the Angular 9 version, but if we want to opt out of that, we can do the following configuration in the tsconfig.app.json file.

```
{
    "angularCompilerOptions": {
        "enableIvy": false
    }
}
```

This is by using the additional property called **enableIvy**. We can opt out from the Ivy rendered engine in Angular 9.

14
Angular 10

Q1. What's new in Angular 10?

Ans. Angular release its major release 10.0.0 on 25th Jun 2020 by putting the note here. It includes the changes in terms of framework, CLI, Angular material, and those summarized changes have been given below.

- TypeScript 3.9
- Common Message-ID for Localization
- Angular material new date range picker
- Strict project setup
- Default browser configuration
- The UrlMatcher can return null values
- MinLength and MaxLength validators with the numeric length property

Above are the standard and significant/minor changes that have been reflected in Angular 10.

Q2. What is the difference between Angular 9 & Angular 10?

Ans. Both Angular 10 and Angular 9 frameworks are excellent options for developing applications with high standard code reliability.

Angular 9 was launched for building mobile and web applications using JavaScript. While Angular 10 was released for a remarkable turnaround for Angular JavaScript experts. A few fundamental differences between Angular 9 and Angular 10 are as below:

Angular 9	**Angular 10**
Typescript 3.7	Typescript 3.9
Enhancement of language services	TSLib 2.9
Service worker updates	Compiler Update
API extractor updates	New default browser configuration
Welcome default Ivy	Converting pre-Ivy code
Component harness	Deprecation of FESN5 or ESM5
Reliable ng-updates	Optional stricter settings

Q3. How Angular 10 release ensures that the ecosystem stays up to date?

Ans. The JS ecosystem is ever-changing, and with the new version of Angular 10, the following dependencies were updated and synchronized:

- The TypeScript version updated is to 3.9: Unlike the previous Angular framework version, which supported TypeScript 3.6, 3.7, & 3.8, in Angular 10, the TypeScript is updated to TypeScript 3.9. TypeScript built is on JS by adding syntax for type declarations and annotations, which is later used by the TypeScript compiler to type-check the code. It gives the results in clean, readable JS that can run on different runtimes.
- TSLib updated to version 2.0: The runtime TypeScript library contains helper functions updated to TSLib 2.0.
- TSLint updated to version 6: Along with TSLib, the TSLint has also been updated to version 6.
- Also, the project layout is updated, so there is a new **tsconfig.base.json** file. It helps to build tool resolve type and package config better than the previous one.

Q4. How to update to Angular 10?

Ans. The release of Angular 10 as a major release can be updated using the below command.

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

Q5. What is an update for MinLength and MaxLength validators in Angular 10?

Ans. With the release of Angular 10, now the validator such as MinLength and MaxLength will work only if the numeric values appear.

Previously, the validator was triggered once the non-numeric values appeared, such as **false** or 0, but now onwards, only numeric values will only trigger the validators.

```
this.testForm = new FormGroup({
    name: new FormControl(this.test.name, [
        Validators.required,
        Validators.minLength(10), // only numeric works
    ]),
});
```

The validator function **minLength(numeric_val)** can only trigger when it gets a numeric value; otherwise, it won't activate.

Q6. Which TypeScript version can be used with Angular 10?

Ans. In Angular, the newer version of TypeScript **3.9** can be used. Hence few TypeScript versions such as 3.6, 3.7, and 3.8 are still supported.

The new TypeScript 3.9 version comes with the advantages of seamless editing performance in the text editor, performance, and polishing the functionalities.

Q7. What are Project stricter settings in Angular 10?

Ans. The new app created with Angular 10 comes with the option setting of stricter settings of the workspace that enhance CLI's performance, maintainability of the workspace, and optimize the app so that the app side effect will be reducer and to get the more tree-shakable developed app.

Q8. How to set Project stricter settings in Angular 10?

Ans. The below command can help apply the environment to use the stricter project settings while creating a new angular app.

```
ng new --strict
```

Q9. What is Angular material's new date-range picker?

Ans. Angular 10 comes with the new form element, which is a date-range picker and a combined element.

- mat-date-range input
- mat-date-range-picker

Using the new date-range picker, you will be able to select the dates, i.e., from the start date to the end date, and get the range as a date value.

Q10. How to use mat-date-range-picker in Angular 10?

Ans. To use the mat-date-range-picker, you can use below code snippet as a basic example.

```
<mat-form-field appearance="fill">
  <mat-label>Select a date range</mat-label>
  <mat-date-range-input [rangePicker]="picker">
    <input matStartDate placeholder="Start Date">
    <input matEndDate placeholder="End Date">
  </mat-date-range-input>
  <mat-datepicker-toggle matSuffix [for]="picker"></mat-datepicker-toggle>
  <mat-date-range-picker #picker></mat-date-range-picker>
</mat-form-field>
```

The element **<mat-date-range-picker>** is used to select the date range while **<mat-date-range-input>** is used to show the selected date range value.

Q11. What is the Default browser configuration in Angular 10?

Ans. Angular supports almost all browsers with specific versions, but with Angular 10 more specifically, now you can set the default browser configuration.

To get the list of all browsers supported to the current Angular application, the below command can be used to list all the supported browsers with their version.

```
npx browserslist
```

If you want to specify the specific list of browsers supported in the current angular app, then you can identify them in **package.json** as given below.

```
"browserslist": [
    "defaults",
    "not IE 10",
    "maintained node versions"
]
```

Q12. How will JSDoc comment removal work in Angular 10?

Ans. Angular NPM will no longer contain individual JSDoc comments to support the closure compiler's advanced optimizations.

Q13. How to handle unknown properties and elements error?

Ans. With the release of Angular 10, you will get unknown element errors rather than getting warnings.

It won't break an app, but it may be possible to cause issues with tools that expect nothing to be logged via **console.error()** specifically.

Q14. What is the return type of UrlMatcher?

Ans. The UrlMatcher is used to match the routes against the current app URL; hence it returns **UrlMatchResult** when any URL matches the route.

But, With the release of Angular 10, there is an update that the UrlMatcher can return as **null** if no matching routes are found, and if found, then **UrlMatchResult** will be returned.

Q15. What is the improvement in the input field for the numbers?

Ans. Previously, the **valueChanges** event gets fired twice, i.e., when the user starts typing the value and blurs the value. With the improved version of Angular, the valueChanges will be triggered only once as soon as the value change.

Q16. What is Improvement in Angular 10 with Ngcc?

Ans. The Ngcc is a standalone Angular Compatibility Compiler used to bundle the various dependencies of the Angular app; thus, if you opt-out from the option **bundleDependencies,** you need to enable Ngcc using the below option in **package.json**.

```
{
    "scripts": {
    ...
        "postinstall": "ngcc",
    ...
    }
}
```

Once you enable the option, it runs each time you run the app installation of node_modules, and it will resolve the Ivy version of the package.

15
Angular 11

Q1. What’s new in Angular 11?

Ans. Angular released its major release 11.0.0 on 11th Nov 2020 by putting the note here. It includes the changes in terms of framework, CLI, components, and summarized changes below.

- TypeScript 4.0
- Ngcc updates
- Webpack 5 support
- Improved logging and the Reporting
- Updates in Internet Explorer
- Updates in Hot Module Replacement (HMR)
- Auto Font Inlining
- Updates in Operation Byelog
- Component Test Harness

Q2. What’s the improvement in Component Test Harness in Angular 11?

Ans. The Component Test Harness was first introduced in Angular 9, which is used for the robust API surface, mainly used for the Angular material.

It gives an extensive way of managing components during the unit testing as end-users do the interaction. Based on the interaction, it tests the component themselves regarding the changes of DOM.

A function like the **manualChangeDetection()** will give us access to even better control of DOM element detection by disabling the auto change detections inside your unit tests.

Q3. What is the automatic inline of fonts in Angular 11?

Ans. With the release of Angular 11, the app's fonts are now converted into inline in file **index.html**.

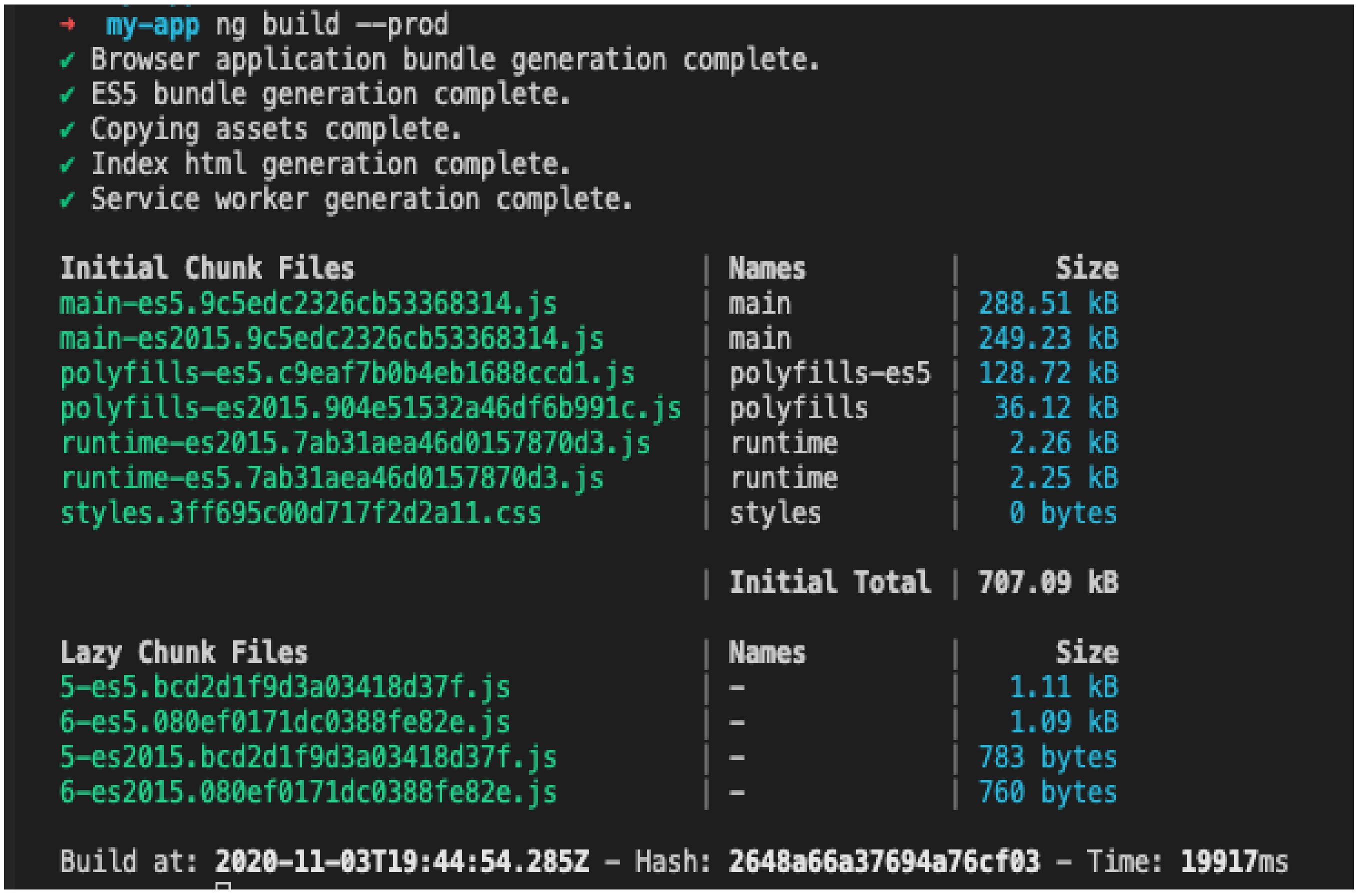
When the Angular app is compiled, the fonts used in the app will be downloaded by default when you update or create an Angular 11 app.

Q4. What are the improvements in Logging and Reporting?

Ans. While you build your Angular app, it's necessary to get all desired logs so that the files being generated can be identified.

There are some improvements in Logging and Reporting, which can be crucial while developing and building the Angular app; apart from that, the Angular CLI is also updated with the logging to be easier to read.

Below is the example that shows the summarized report while building the Angular app.



```
➜  my-app ng build --prod
✓ Browser application bundle generation complete.
✓ ES5 bundle generation complete.
✓ Copying assets complete.
✓ Index html generation complete.
✓ Service worker generation complete.

Initial Chunk Files                          | Names         |      Size
main-es5.9c5edc2326cb53368314.js             | main          | 288.51 kB
main-es2015.9c5edc2326cb53368314.js          | main          | 249.23 kB
polyfills-es5.c9eaf7b0b4eb1688ccd1.js        | polyfills-es5 | 128.72 kB
polyfills-es2015.904e51532a46df6b991c.js     | polyfills     |  36.12 kB
runtime-es2015.7ab31aea46d0157870d3.js       | runtime       |   2.26 kB
runtime-es5.7ab31aea46d0157870d3.js          | runtime       |   2.25 kB
styles.3ff695c00d717f2d2a11.css              | styles        |   0 bytes

                                             | Initial Total | 707.09 kB

Lazy Chunk Files                             | Names         |      Size
5-es5.bcd2d1f9d3a03418d37f.js                | -             |   1.11 kB
6-es5.080ef0171dc0388fe82e.js                | -             |   1.09 kB
5-es2015.bcd2d1f9d3a03418d37f.js             | -             | 783 bytes
6-es2015.080ef0171dc0388fe82e.js             | -             | 760 bytes

Build at: 2020-11-03T19:44:54.285Z - Hash: 2648a66a37694a76cf03 - Time: 19917ms
```

PC – angular.io

Q5. What is Hot Module Replacement (HMR) in Angular 11?

Ans. The Hot Module Replacement allows you to replace the module without refreshing the browser window; hence this feature is essential for app re-rendering.

With Angular 11, the CLI has been updated, so now you can use HMR while running the application with a given command.

```
ng serve --hmr
```

Once you run the above command, you can see the below message in the console.

Hot Module Replacement (HMR) is enabled for the dev server.

The advantage to using HMR is when you make some changes in the component, templates, or other util file, changes will be reflected without refreshing the browser.

Q6. Which version of TypeScript Angular 11 supports it?

Ans. Angular supports the latest version of TypeScript 4.0, so once you create a new Angular app, a recent version will be reflected, or you can update TypeScript into the existing app.

You can use the below command to install TypeScript.

```
npm install -g typescript
```

Q7. Which version of Webpack Angular 11 support?

Ans. With Angular 11, you will use the webpack 5.0 version to achieve a small app bundle size and a faster build process.

But keep in mind that the use of webpack in Angular 11 is still under development; hence it's not recommended to use it for production use.

You can opt for webpack 5.0 using the below solution in the **package.json** file.

```
"resolutions": {
    "webpack": "5.4.0"
}
```

Q8. How to use the generator for Resolvers in Angular 11?

Ans. With Angular 11, now you will be able to generate a resolver guard using the below command.

```
ng generate resolver resolver_name
```

Q9. What is the use of the i18n token from the library in Angular 11?

Ans. Angular 11 will enable you to fetch the token of i18n from the various libraries using the below-given command.

```
ng xi18n --ivy
```

Q10. How Angular 11 is faster than the previous versions?

Ans. The answer is NGCC; yes, the Angular 11 version gives faster app development and build process in terms of the compiler with Ngcc as now it gives around 4x speedier performance.

The update in Ngcc brings faster development; hence users don't need to wait for a longer cycle of build and re-build the app.

Q11. Does Angular 11 support Internet Explorer?

Ans. Angular 11 will only support IE 11, and other previous versions such as IE9/IE10 will no longer be supported.

Q12. What are the updates with Linting in Angular 11?

Ans. With the release of Angular 11, the support of TSLint is now deprecated, and now it's being discontinued, so now you can opt for a community-driven ESLint builder using **angular-eslint**.

You can add angular-eslint schematics to your app using the below command.

```
ng add @angular-eslint/schematics
```

Using the above command, it will install all the necessary packages required for the linting, which you can find inside the **devDependencies** of the package.json file.

Q13. What are the new improvements in the form with Angular 11?

Ans. Angular 11 has improved typing for validators and **asyncValidators** not available in the previous version of Angular.

Type of **AbstractFormControl.parent** in Angular 11 includes **null** included in a parent's types. The version 11 migration adds the non-null assertion operator wherever necessary.

Previously, the various async validator needs to be defined at initialization, but in the newer version of Angular, the status event is emitted into the **statusChanges** observable.

Q14. How to update to Angular 11?

Ans. If you want your application to be updated with Angular 11, you can use the below command to update the core dependencies.

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

16
Angular 12

Q1. What's new in Angular 12?

Ans. Angular was released on May 12, 2021, with the notable word "Ivy Everywhere" along with other updates as given below.

- Ivy, deprecation of view engines
- Styles improvement
- Nullish Coalescing
- Prevention of accidental production build while using ng build
- Strict mode with CLI
- Typescript V 4.2 support
- IE browser support deprecation
- Webpack 5.37 support

Q2. Is it feasible to upgrade to the latest Angular 12?

Ans. Since Angular 11 is going to end its support and get feasible updates from the newer version, it's always feasible to use the latest version of Angular 12.

Q3. What is the major difference between Angular 11 and Angular 12?

Ans. The major difference between them is given below.

- Typescript 4.2 support and Typescript 4.0 in Angular 12
- Deprecation the support of IE 11 in Angular 12 where Angular 11 deprecated support of IE 9 and 10
- New default strict mode in Angular 11 where it does not support previously in Angular 11
- Improvements in HTTP, language service provider, logging, and reporting

Q4. What is the role of Ivy everywhere in Angular 12?

Ans. With the release of Angular 12, the major update has been considered which is the deprecation of view engines. The Ivy everywhere works for the purpose to make Angular faster, simpler, and way easy to maintain.

Since Angular 9, the role of the Ivy is to improve compilation and it provides the runtime instruction, so it will continue to do so further.

With the release of Angular 12, developers can use either View engines or Ivy, so we need to start doing transitions to use Ivy in our application.

Q5. What is Nullish Coalescing?

Ans. The Nullish Coalescing is the new operator that has been released with Angular 12 which is used to help developers to manage complex conditional operations into the Typescript inline template.

Previously, we have been managing condition is such way.

```
{{ employee && employee.name ? employee.name : getEmployeeName() }}
```

But now it will become.

```
{{ employee.name ?? getEmployeeName() }}
```

Q6. How new transition of Legacy i18n Message-IDs works in Angular 12?

Ans. With the release of Angular 12, you now have tools available that will help you to migrate to latest legacy localization IDs to the latest IDs used by updated algorithms.

Q7. How to migrate legacy IDs using CLI in Angular 12?

Ans. There are some sets of CLI commands that will help you to migrate as given below.

- Run the new command for legacy migrate

```
ng extract-i18n --format=legacy-migrate
```

After running the above command, now you will have the file called "message.json" which contains the migrated IDs and new IDs that can be mapped to each other.

- Update IDs of your application

```
npx localize-migrate --files=*.xlf --mapFile=messages.json
```

Q8. How to enable inline SCSS styling Angular 12?

Ans. In the older version of Angular, SCSS was available in external resources because of Angular compiler configuration, but with Angular 12, now it's available to enable SCSS inline.

To enable inline SCSS styling, you need to specify the below property in the **angular.json** file

```
"inlineStyleLanguage": "scss"
```

Q9. Does IE 11 support Angular 12?

Ans. The answer is straight NO because previously in Angular 11, the support of IE 9 and 10 was deprecated and now with Angular 12, IE 11 will no longer be supported.

Q10. How strict mode will work with Angular 12?

Ans. Strict mode is now enabled by default with the use of CLI which can be helpful to developers to identify errors at the time of development.

You can enable it while creating a new Angular 12 app using the below command.

```
ng generate application [project-name] --strict
```

Q11. What is the default build mode in Angular 12?

Ans. Now, the command "ng build" will default to the production mode in Angular 12 which helps to avoid accidental development builds.

You can opt-in while creating a new Angular 12 application using the below command.

```
ng update @angular/cli — migrate-only production-by-default
```

Q12. Does Angular 12 support Webpack?

Ans. Till the Angular 11 version, the Webpack was introduced as experimental support but now it has been supporting a production-ready version with Angular 12.

17
Angular 13

Q1. What’s new in Angular 13?

Ans. The angular team launched the all-new version Angular 13 on Nov 3, 2021, with all sorts of updates and updated features to bring our development to the next level. It’s all-Ivy, no more View engine.

The new Angular13 version brought some essential updates which are given below.

- TypeScript 4.4
- Persistent build cache
- Angular Package Format (APF)
- No browser support of IE 11
- Accessibility enhancements in Angular Material

Q2. What’s the state of View Engine in Angular 13?

Ans. With the release of Angular 13, the View engines are not more supported from now onwards, instead, the Angular team will focus more on Ivy-based enhancement of its feature for the future release.

Q3. How will Angular 13 improve performance?

Ans. Angular 13 has been focused more on the critical updates and one of them is “Load Time” which is enhanced using ergonomic code-splitting at a component level.

To achieve the higher load time, the team has introduced “ESBuild” which is a faster JavaScript bundler and by using it, we can optimize scripts and source map CSS.

Q4. Does IE 11 support Angular 13?

Ans. Angular 13 version completely removed the support of IE 11 and it helps to improve overall bundle size, and will eventually improve the application performance.

Q5. How to create a dynamic component in Angular 13?

Ans. Before Angular 13, it was a bit difficult to create the dynamic component with boilerplate code, but now it gets easier with the latest updates using “ViewContainerRef.createComponent” without creating an associated factory.

For example, before were creating the component like below.

```
@Directive({ … })
export class MyDirective {
    constructor(private viewContainerRef: ViewContainerRef,
        private componentFactoryResolver:
            ComponentFactoryResolver) { }
    createMyComponent() {
        const componentFactory = this.componentFactoryResolver.
            resolveComponentFactory(MyComponent);

        this.viewContainerRef.createComponent(componentFactory);
    }
}
```

But now, it's easy to create it with less boilerplate code as given below.

```
@Directive({ … })
export class MyDirective {
    constructor(private viewContainerRef: ViewContainerRef) { }
    createMyComponent() {
        this.viewContainerRef.createComponent(MyComponent);
    }
}
```

Q6. How testing and debugging improved in Angular 13?

Ans. Angular uses "TestBed" as the primary unit testing framework, hence, it has been improved in the latest version so now it will learn the DOM and refreshes the tests faster after each test run.

Q7. What is Angular Material accessibility?

Ans. There is an Accessibility (A11y) improvement seen in Angular Material, hence all Material Design Components (MDC) have been checked for better Accessibility.

The form contro0ls such as checkboxes and radio buttons, for example, have now larger touch sizes, and other components have better high contrast modes.

Q8. What improvements are introduced in Angular CLI?

Ans. With the updates to Angular tooling, version Angular 13 now supports the use of persistent build-cache by default for new v13 projects.

Doing persistent build-cache by default led to this tooling update that results in approx. 68% improvements in build speed and more ergonomic options.

While using existing projects that have been upgrading to the Angular 13 app, we need to enable these features and that be done using the below configuration into the file "angular.json".

```
{
    "$schema": "...",
        "cli": {
        "cache": {
            "enabled": true,
            "path": ".cache",
            "environment": "all"
           }
        }
    ...
}
```

Q9. Which version of RxJs Angular 13 support?

Ans. Angular 13 update includes RXJS to all the versions up to version 7. New apps created with the CLI now will default to RxJS 7.4 version.

In case you are using RxJS 6 or an older version in your existing app, then you will have to manually run the below command for the latest updates.

```
npm install rxjs@7.4
```

Q10. What are the minor updates in Angular 13?

Ans. Along with the other major updates, a few minor updates that have been added are given below.

- Node.js 16 has been added in all the node engines by all the Angular packages
- The opt-in feature is now enabled by default
- The renderModuleFactory from @angular/platform-server is no more required with Ivy. Instead, renderModule can be used
- The Angular language supports auto-completion for union types in the templates
- Incremental builds performance don't contain any of the Angular traits such as Component, Directive, Injectable, Pipe, and NgModule decorators

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