Crack Your Angular Interview

By-Sahosoft Solution

ABOUT THE BOOK

This book is based on Angular Interview Questions which are mainly asked in most of the interviews. We are pleased to say that we got the idea of this book while looking for the perfect solutions for our Angular-related queries. Unfortunately, in most cases, we couldn't find enough information or satisfactory solutions. Often, we had to search through multiple websites, tutorials, and links to collect enough information to resolve our queries and after that we used to compile for further convenience. After facing this type of situation, we planned to write a book which covers almost every Angular-based interview question, as well as covering almost all the topics of Angular.

We believe this book will help you to prepare in a better way towards cracking your interviews. This is the reason behind the book's title, **"Crack Your Angular Interview"**.

We believe that you also believe that reading multiple questions related to a subject is a great way to learn and revise the concept. It helps to enhance the knowledge in a very useful way. In this book, we tried to cover most of the important concepts of basic and advance of Angular in a descriptive way of each version in the form of questions and answers with examples.

Although we have tried to make this book as accurate as possible, but if there is something that is not necessary or you find an error in the book, please let us know.

ABOUT THE AUTHORS

Chandan Kumar Singh

Chandan Kumar Singh has an experience of 10+ years in developing enterprise-level applications in technologies like : C#, ASP.NET4.0 with MVC, Angular 2, Angular 4+, React JS, Node JS, Mongo DB, Express JS , LINQ, SQL Server, AJAX, XML, JavaScript, and jQuery and also analyzing the business requirements and translating it to technical specifications.

He holds a vast experience in all the phases of software development life cycle (SDLC), including requirements gathering, analysis, design, reviews, coding, testing, debugging, documenting, unit and integration testing.

He is also a good communicator with many interpersonal skills that help him in leveraging technical, business knowledge to communicate effectively with client executives and their respective teams and recognize the project business needs and present solutions.

Ajeet Kumar Singh

Ajeet Kumar singh has an experience of 9 years in the field of software development. He has designed and developed Various applications in technologies like: C#, ASP.NET4.0 with MVC, LINQ, SQL Server, AngularJS, Angular (version 2+), React JS, Node JS, Mongo DB, Express JS etc.

He also has experience in all the phases of software development life cycle (SDLC): including requirements gathering, analysis, design, reviews, coding, testing, debugging, documenting, unit and integration testing.

He is interested in doing work with client-side framework like Angular, Angular (ver. 2+), React, Vue, etc. and learning all the new technologies also with the expertise to grasp new concepts quickly and utilize the same in a productive manner.

Index

Q1. What is Angular JS?

Ans. Angular JS was introduced in 2010 as a JavaScript library for creating client-side applications. It helps us to create single-page applications (SPA) or one-page web applications that only require HTML, CSS, and JavaScript on the client side.

Q2. What is Angular?

Ans.

The Microsoft's TypeScript team and Google's Angular team together have rewritten the AngularJS and announced Angular 2 in 2016. We can say that it is a reborn version of AngularJS entirely based on TypeScript.

It is a framework (not library) for creating client-side application in HTML, CSS and TypeScript.

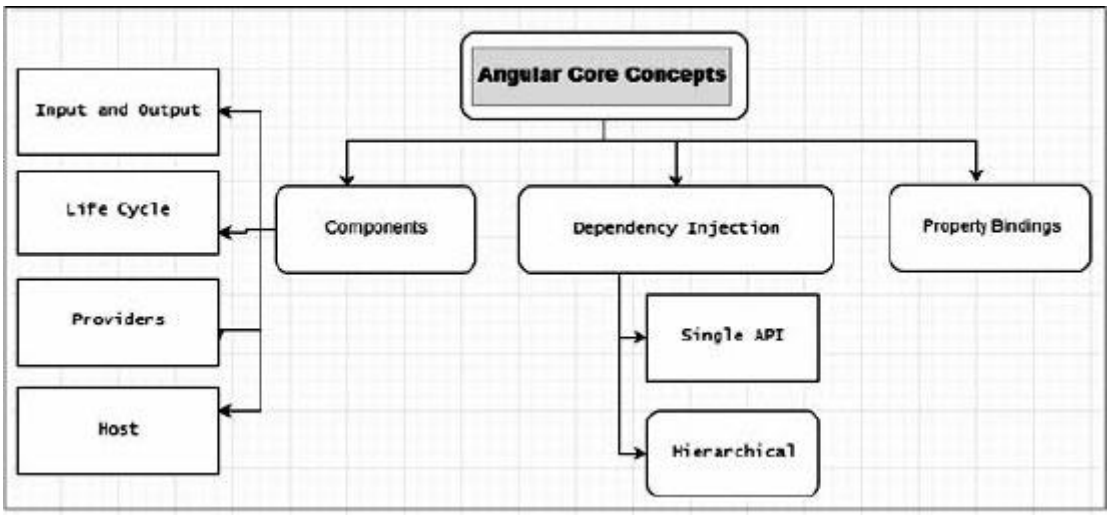
- **Angular** is the most popular web development framework for developing **mobile** and **desktop applications**.

- **Angular framework** is also used in **cross-platform mobile development** called **IONIC, which** means it is not limited to web applications only.

- **Angular** is an open source framework written and maintained by Google's Angular team and the **Father** of Angular is **Misko Hevery**.

Misko Hevery: - Agile Coach at Google, Attended Santa Clara University and Lives in Saratoga, California.

Angular is written in TypeScript and it comes with all the functionality offered by the typescript. The core concepts of Angular is:



Don't worry about the TypeScript versions. The compiler manages to the version related issues.

Q3. What are the various versions of Angular?

Ans.

Q4. Why was angular version 3 skipped?

Ans. In short, Angular router was in version 3.x before Angular 3 was released. In an attempt to avoid confusion, they skipped Angular 3 and programmed Angular 4 to match with Angular router 4.

Means to avoid the confusion due to the misalignment of the router's package version which is already distributed as version 3.3.0 with Angular 2, i.e. Angular 2 contains the version of the router file 3.3.0.

Q5. Why we need Angular?

Ans. As explained above, Angular has been completely rewritten, now you need to know why we are using Angular while we have AngularJS. So, the answer is that Angular is designed for all the devices.

In AngularJS, developer face a problem when using the application on mobile or any other device except the web. Thus, the team has decided that they will develop Angular, which can be used for all the devices, such as Mobile, iPhones, Tablets etc.

It is the product of Google, so today all developers wants to learn Angular. There are a few main things you must know.

- **Easier:** Angular is easier to learn and within a limited time, we can learn it and create the applications with simple CRUD features.

- **Performance:** Its performance is much better than AngularJS because execution is very fast when we perform any operation in the application.

- **TypeScript**: Nowadays, most of the developers are using TypeScript because it is easier to learn since it is based on OOPS concepts and is a superset of JavaScript. As you know, most of the developers have good knowledge of OOPS, so they can learn it in a minimal time.
- **Easy Testing**: When we work on an Angular project, we can easily test the application by creating test cases and it is easier than others.

- **Increased Developer Productivity**: Angular increases the developer productivity because a developer can easily learn and complete any task or feature quickly.

- **Work in Coding pattern**: Using Angular, you can work on a good consistent coding model because it provides codelyzer with which you can write consistent code and discover potential errors.

- **Angular uses full featured Routing**: Routing is yet another great feature of Angular. Angular performs navigation from one view to another and works very quickly. It supports lazy loading that allows you to load the code fragments on demand which means entire code will not be executed at one time.

- **Easy to build and use only required file**: When we build the Angular v2 - v6

application, the application will build and required file will store in a dist file. In the dist folder, only necessary files will be saved.

- **Change detection:** It is a very important feature of Angular. It updates the running application whenever changes are made to the code.

Q6. What is the problem in Angular JS for which we are using Angular?

Ans. With AngularJS, developer face problem when using the application on mobile or any other device except the web. Thus, the team decided that they will develop Angular, which can be used for all the devices, such as Mobile, iPhones, Tablets etc.

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

- It is entirely based on component.
- Better change detection.
- Ahead of Time compilation (AOT) improves rendering speed.
- TypeScript is primarily used for developing Angular 2 applications.
- Angular 2 or higher has better performance over Angular JS.
- Angular 2 or higher has a more powerful templating system than Angular JS.
- Angular 2 or higher has simpler APIs, lazy loading, easier debugging.
- Angular 2 or higher is much more testable than Angular JS.
- Angular 2 or higher provides nested components.
- Angular 2 or higher provides a way to execute more than two systems together.

Q7. Which language we can use with Angular for Component creation?

Ans. In Angular JS, we generally write JavaScript program. Primarily, we use TypeScript in Angular. However, we can also use several other languages like - TypeScript, JavaScript ES5, ES6, and Dart. With Angular 2 or higher the official site provides code exapmles in several languages; like JavaScript, TypeScript and Dart.

JavaScript is an implementation of the ECMAScript standard. . There are several versions, including ES5 and ES6. Today, most browsers fully support JavaScript ES5. This means that code compliant with ES5 will run on most browsers without modification. ES6 on the other hand, may not be supported and is generally converted to ES5 before it reaches the browser. The latest version of JavaScript (ES6) adds functionality and simplifies certain tasks. It is unclear when browsers will be fully compatible with ES6.

TypeScript and Dart are super sets of JavaScript. This means that they include all the

JavaScript functionality, but also add many helpful features as well. Unlike the ES6 version of JavaScript they don't work on current browsers. To avoid this limitation, there are utilities to convert code written in their non-browser-compliant syntax back to JavaScript ES5. In this way, they provide the developer with advanced functionality and still allow code execution in existing browsers. The process of converting code from one language to another is known as **'transpilation'**. Technically, this conversion is performed by a language pre-processor that, although does not compile anything, is commonly known as compiler.

TypeScript is an open source language developed and managed by Microsoft. It provides type safety, advanced object-oriented features, and simplified module loading. Angular 2 itself is built using TypeScript. Documentation support is better on the Angular 2 site for TypeScript than for other options. So, while you could write your Angular 2 applications in pure ES5 JavaScript, you may find it more difficult and find less support if you get into trouble than biting the bullet and spending time on learning TypeScript.

Dart is another alternative for programming Angular 2 applications. It is similar in features to TypeScript and has its own loyal following. If your team is already versed in Dart and is looking to start developing in Angular 2, it could be a good choice. .Having said that, in the rest of this post, we will focus on TypeScript.

Q8. What is the need of TypeScript in Angular?

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

Q9. Can I use another language except TypeScript for Angular?

Ans. Yes, JavaScript ES5, ES6, ES7, ES8, ES9… and DART can also be used for angular.

Q10. What is Angular CLI?

Ans. Angular CLI is a command line interface which we use to build the Angular applications using node.js. It provides us with the functionality by which we can easily create all the required files as component, service, pipes etc. with the help of only command line in a good structure. It is based on webpack, so you can easily bundle the application automatically when it is created.

OR

The Angular CLI (Command Line Interface) is a tool for initializing, developing, scaffolding and maintaining Angular applications.

You can use CLI commands to generate an app, the default AppModule is as follows –

Ng new <projectName>

The above CLI command is used to create a new angular project and this CLI command will

automatically create many folders and files that are necessary for project development, testing, configuration and so on.
To use the Angular CLI, we must first install it (globally on your computer).

npm install -g @angular/cli
Here g stands for global

Some Additional CLI Commands:

1. ng new
2. ng serve
3. ng generate
4. ng lint
5. ng test
6. ng e2e
7. ng build
8. ng get
9. ng set
10. ng doc
11. ng eject
12. ng xi18n
13. and many more

Q11. Why do we use Angular CLI while we have other methods to create Angular application?

Ans. Yes, we can also create the application with other methods, such as using Visual Studio (any version - 2015, 2017 and 2019) but with Angular CLI, it is much easier to create a robust application.

Q12. What are the differences between Angular Js and Angular 2?

Ans. Angular 2 is a TypeScript based open source front-end web application platform managed by
the Angular Team of Google and community of individuals and corporations. Angular 2 is a complete rewritten from the same team that built AngularJs.

Angular 2 is a platform that makes easy to build applications with the web. Angular 2 combines declarative templates, dependency injection, an end to end tooling, and integrated best practices to solve development challenges. Angular empowers developers to build applications that live on the web, mobile, or the desktop.

The first version of AngularJs was released in 2010 while Angular 2 was released in 2016.

Both AngularJs and Angular 2 are completely different. Actually, Angular 2 is completely component based and AngularJs is both scope and controller based.

Angular 2 is a platform and framework for building client applications in HTML and TypeScript. Angular 2 is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that can be imported in your application.

Angular 2 is used on Cross-platform, modern browsers only. The core differences and many

more advantages on Angular 2 vs. AngularJs are:

- AngularJS is the first version of Angular released in 2010. It is also known as Angular 1. Angular 2 is the second version of Angular released in 2016.
- JavaScript is used in AngularJS whereas from Angular 2, TypeScript version 1.8 is used.
- Angular 2 is entirely component based, while AngularJs is controllers and scope based.
- Angular 2 has better change detection as compare to AngularJs.
- Angular 2 has better performance as compare to AngularJs.
- Angular 2 has a more powerful template system.
- Angular 2 provide simpler APIs, lazy loading and easier application debugging.
- Angular 2 is much more testable.
- Angular 2 provides nested level components. Ahead of Time compilation (AOT) improves rendering speed.
- Angular 2 execute run more than two programs simultaneously. The syntax of Angular 2 structural directives is changed like **ng-repeat** is replaced with ***ngFor** etc.
- In Angular 2, local variables are defined using the hash prefix (#). You can see the following *ngFor loop Example. TypeScript can be used to develop Angular 2 applications.
- Better syntax and application structure. There are more advantages over performance, template system, application debugging, testing, components, and nested level components.

Q13. What are the differences between Angular 2 and Angular 4?

Ans. Obviously! Angular 4 is smaller, faster, easier to use and make developer's life simpler than Angular 2.

Angular 2 was released in 2016 whereas Angular 4 was released in 2017.

Angular 2 and Angular 4 use the same concept and patterns.

Angular 4 contains some additional enhancement and improvement. Consider the following enhancements:

- Smaller & Faster Apps.
- View Engine Size Reduce.
- Animation Package.
- NgIf and ngFor Improvement.
- Template.
- NgIf with Else.
- Use of AS keyword.
- Pipes.
- HTTP Request Simplified.
- Apps Testing Simplified.

- Introduce Meta Tags.
- Added some Forms Validators Attributes.
- Added Compare Select Options.
- Enhancement in Router.
- Added Optional Parameter.
- Improvement Internationalization.

- Smaller & Faster Apps - Angular 4 applications is smaller and faster in comparison to Angular 2.
- View Engine Size Reduce - Some changes under to hood to what AOT generated code compilation that means in Angular 4, compilation time was improved. In most of the cases these changes reduce around 60% size.
- Animation Package- Now, animations have their own package i.e. @angular/platform-browser/animations
- Improvement - Some Improvement on ∗ngIf and ∗ngFor.
- Template - Now Angular has its own template tag that is called “ng-template”. You should use the “ng-template” tag instead of “template”.
- NgIf with Else- From Angular 4, it is possible to use an else syntax in template.

Q14. What are the differences between Angular 4 and Angular 5?

Ans. Angular 5 being smaller, faster, easier to use and it make developer’s
life easier as compare to Angular 2.
Angular 4 is released in 2017 while Angular 5 is released on 1st November 2017.
Angular 5 is a much better version and you are able to take advantage of it in simple way.
Angular 5 contains a bunch of new features, performance improvements and a lot of bug fixes and also some surprises to Angular lovers:

- Smaller and Faster Apps.
- Build optimizer - It helps to remove unnecessary code from your application.
- Angular Universal State Transfer API and DOM Support.
- Supports TypeScript 2.3+ version.
- Compiler Improvements - It makes AOT the default and ng serves aot.
- Increased the standardization across all browsers.
- Watch mode.
- Type checking in templates.
- More flexible metadata.
- Remove ∗.ngfactory.ts files.
- Better error messages.
- Smooth upgrades.

- Tree-Shakeable components.
- Hybrid Upgrade Application.
- Include Representation of Placeholders to xliff and xmb in the compiler.
- Include an Options Arg to Abstract Controls in the controls of the form.
- Include add default updateOn values for groups and arrays to form controls.
- Include updateOn blur option to form controls.
- Include updateOn submit option to form controls.
- Include an Events Tracking Activation of Individual Routes.
- Include NgTemplateOutlet API as stable in the common controls.
- Create StaticInjector which does not depend on Reflect polyfill.

Q15. What are the differences between Angular 5 and Angular 6?

Ans. Angular 6 being smaller, faster, easier to use and it makes developer's life easier.
The Angular Team have worked on lots of bug fixes, new features to be added/ update/remove/ re-introduce/ and many more things.
Let's start to explore all the changes of Angular 6 step by step!

Added ng update - This CLI commands will update your angular project dependencies to their latest versions. The ng update is normal package manager tools to identify and update other dependencies.

CLI Command- Ng update

Angular 6 uses RxJS 6 – It is the third-party library (RxJS) and introduces two important changes as compared to RxJS 5.
1. RxJS 6 introduces a new internal package structure.
2. Operator concept.

Both require you to update your existing code.
To update to RxJS 6, you have to simply run the following command:
npm install --save rxjs@6

Simply run the below command and update your existing Angular project:

```
npm install --save rxjs-compat
```

Alternatively, you can also use the command - ng update rxjs to update RxJS and install the rxjs-compat package automatically.

RxJS 6 Related import paths:

Instead of:

```
import { Observable } from 'rxjs/Observable';
import { Subject } from 'rxjs/Subject';
```

You can also use a single import as shown below:

```
import { Observable, Subject } from 'rxjs';
```

So, all from **rxjs/Something** imports become from one 'rxjs'.

Operator imports have to change:

Instead of:

```
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/throttle';
```

You can also use a single import as shown below:

```
import { map,throttle } from 'rxjs/operators';
```

And
Instead of:

```
import 'rxjs/add/observable/of';
```

Now you can use:

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

RxJS 6 Changes - Changed Operator Usage -

Instead of:

```
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/throttle';

yourObservableData.map (data => data * 2)
.throttle(...)
.subscribe(...);
```

You can also use the new pipe () method as shown below:

```
import { map, throttle } from 'rxjs/operators';
yourObservableData.pipe(map(data => data * 2), throttle(...))
.subscribe(...);
```

CLI update and added a new project config file –

Instead of ".angular-cli.json" use "angular.json". Now in Angular 6 new projects use the "angular.json" file instead of ".angular-cli.json" file.

```
ng update @angular/cli --from=1 --migrate-only
```

The above command will help you to update your existing ".angular-cli.json" file in the new "angular.json" file.

The "angular.json" file contains the following properties:

1. Version: This is integer file format version and it is currently 1.

2. newProjectRoot: This is string path where new projects are created.
3. defaultProject: This is default project name used in commands.
4. CLI: This is workspace configuration options for Angular CLI and it contains:
   - ❖ defaultCollection.
   - ❖ packageManager.
   - ❖ Warnings.
   - ❖ And so on.

5. Schematics: This is configuration options for Schematics.
6. Projects: This is configuration options for each project in the workspace and it contains:
   - ❖ root.
   - ❖ sourceRoot.
   - ❖ projectType.
   - ❖ prefix.
   - ❖ Schematics.
   - ❖ Architect: This is the project configuration for Architect targets.

The <template> deprecated, Now Angular 6 introduce <ng-template>:

From Angular 6, you should use <ng-template> instead of <template>
For example, previously you are using

app.component.html

src > app > app.component.html > ...

```
1  <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3  <template [ngIf]="IsStudent">
4    <p>This template renders only if IsStudent is true.</p>
5  </template>
6
```

Now, in Angular 6, you must use <ng-template> instead of <template>.

app.component.html

src > app > app.component.html > ...

```
1  <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3  <ng-template [ngIf]="IsStudent">
4    <p>This template renders only if IsStudent is true.</p>
5  </ng-template>
6
```

Service level changes (the way of making a service global):

In the previous versions, if you want to provide a service to the entire application you have to add it to the providers [] in the AppModule, but from Angular 6 you don't have to add in the providers [] in the AppModule.

Example for marking a service as global:

Instead of

my.service.ts

my.service.ts

src > app > my.service.ts > ...

```
1  import { Injectable } from '@angular/core';
2
3  @Injectable()
4  export class MyService {
5
6    constructor() { }
7  }
```

app.module.ts

my.service.ts | app.module.ts

src > app > app.module.ts > ...

```
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6  import { MyService } from './my.service';
7
8  @NgModule({
9    declarations: [
10     AppComponent
11   ],
12   imports: [
13     BrowserModule,
14     AppRoutingModule
15   ],
16   providers: [MyService], //My services instances are now available across the entire app.
17   bootstrap: [AppComponent]
18 })
19 export class AppModule { }
```

Use this code in Angular 6-

my.service.ts

my.service.ts × app.module.ts

src > app > my.service.ts > ...

```
1   import { Injectable } from '@angular/core';
2
3   @Injectable({
4     providedIn: 'root'
5   })
6   export class MyService {
7
8     constructor() { }
9   }
10
```

app.module.ts

my.service.ts app.module.ts ×

src > app > app.module.ts > ...

```
1   import { BrowserModule } from '@angular/platform-browser';
2   import { NgModule } from '@angular/core';
3
4   import { AppRoutingModule } from './app-routing.module';
5   import { AppComponent } from './app.component';
6
7   @NgModule({
8     declarations: [
9       AppComponent
10    ],
11    imports: [
12      BrowserModule,
13      AppRoutingModule
14    ],
15    providers: [], //My services does not need to add here.
16    bootstrap: [AppComponent]
17  })
18  export class AppModule { }
```

Q16. What are the differences between Angular 6 and Angular 7?

Ans. Angular 7 is smaller, faster and easier to use and make developers life easier.
Angular 7 is a major release and expands across the platform including core framework, Angular Material, and Angular CLI (Command Line Interface).
Let's introduce some new features of Angular 7 –

- Added a new compiler –Angular Compatibility Compiler (ngcc)

- ❖ Added a new interface - UrlSegment interface
- ❖ Added a new interface - DoBootstrap interface
- ❖ Introduce a new Pipe called - KeyValuePipe
- ❖ Now supporting to TypeScript 2.9 and higher.
- ❖ Added a new elements features - enable Shadow DOM version1 and slots.
- ❖ Added a new router features - warn if navigation triggered outside Angular zone.
- ❖ Added a new mapping for ngfactory and ngsummary files to their module names in AOT summary resolver.
- ❖ Added a new "original" placeholder value on extracted XMB.
- ❖ Added a new ability to recover from malformed URLs.
- ❖ A new Drag & Drop feature added.

Q17. What are the differences between Angular 7 and Angular 8?

Ans. Let's introduce some new features added in Angular 8 -

- Small bundle size, CLI APIs, and alignment with the ecosystem.
- AngularJs Migration Improvements.
- Route Configurations use Dynamic Imports.
- By default Differential Loading
- Web Worker Support
- New Deprecation Guide
- Builder APIs in the CLI
- Workspace APIs in the CLI
- Ivy & Bazel
- Ivy aims to change this. Compared with the current Angular View Engine, Ivy provides the following benefits.
    1. Generated code that is easier to read and debug at runtime.
    2. Faster re-build time.
    3. Improved payload - applications size improvements
    4. Improved template type checking
    5. Great backward compatibility

Q18. What are the differences between Angular 8 and Angular 9?

Ans. Let's introduce some new features added in Angular 9 -

- In the core, undecorated classes migration schematic is added.

- The formControlName also accepts number in the form.
- Now, allow selector-less directives as base classes in View Engine in the compiler.
- Added support selector-less directive as base classes in Ivy and also make the Ivy compiler the default for ngc.
- Convert all ngtsc diagnostics to ts.Diagnostics
- bazel: support ts _library targets as entry-points for ng _package.
- core: add dynamic queries schematic.
- core: Mark TestBed.get as deprecated.
- ivy: expose window.ng.getDebugNode helper and also support ng-add in localize package.
- ivy: i18n – add syntax support for $localize metadata block.
- ivy: i18n – reorganize entry-points for better reuse.
- language-service: enable logging on TypeScriptHost.
- language-service: provide diagnostic for invalid templateUrls.
- language-service: provide diagnostics for invalid styleUrls.

Q19. What are the prerequisites for Angular?

Ans. 1.First, we need to install latest version of "node".

Basically, it is a runtime environment for executing JavaScript code into the browser. Node provides some tools that are needed to build Angular project.

- Install node from nodejs.org (https://nodejs.org/en/)
- Minimum version required for building Angular application is 6.9

2. Then, we need a tool called NPM (Node Package Manager) to install any library.

3. Angular CLI (Command Line Interface) is a tool that we use to create a new Angular project.

Q20. How can we check the version of Angular?

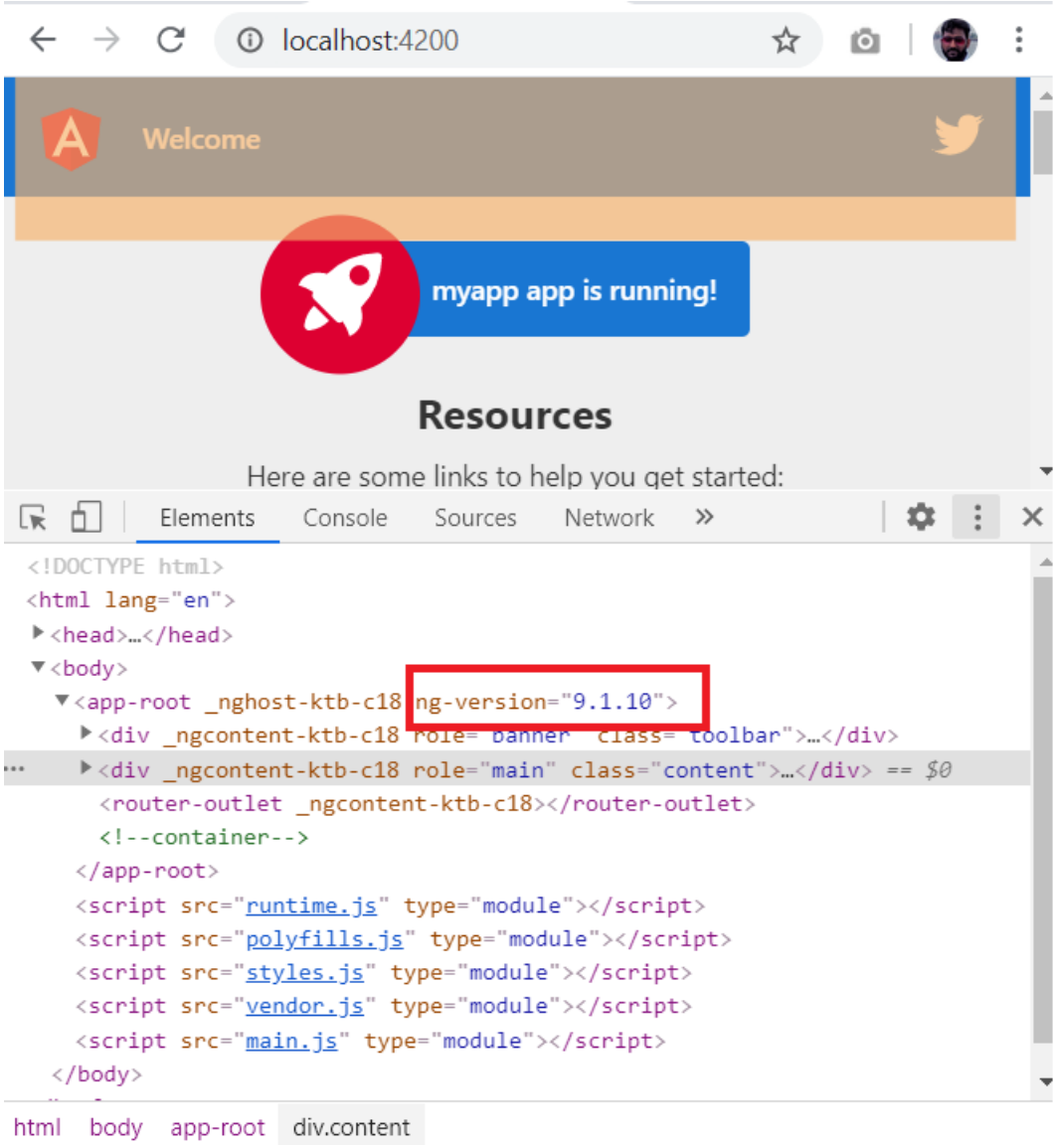
Ans. We can check the version of Angular in many ways -

1. By using cmd
ng –v or ng –version or ng v

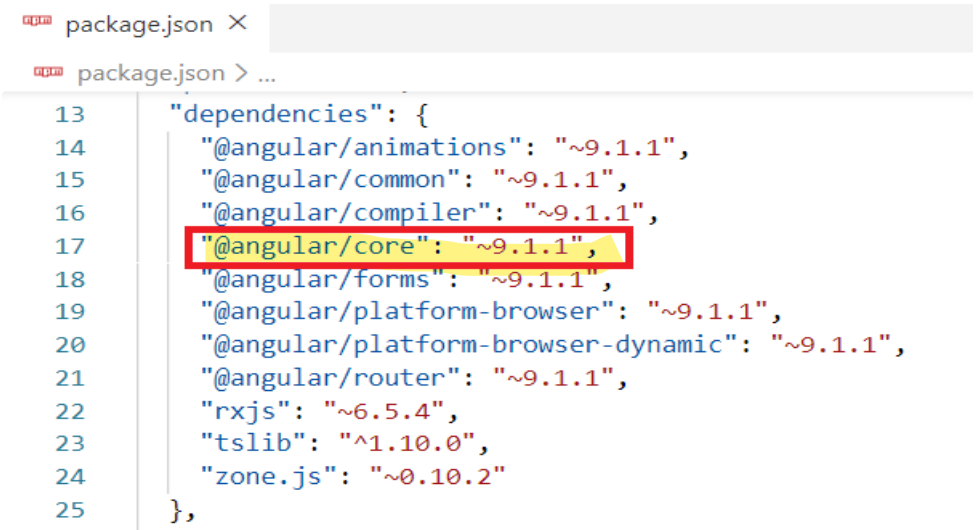
```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
For more detailed help run "ng [command name] --help"
PS C:\Users\Ajeet\sahosoftapp> ng v
```

```
Angular CLI: 9.1.7
Node: 12.13.0
OS: win32 x64

Angular: 9.1.10
... animations, common, compiler, compiler-cli, core, forms
... language-service, platform-browser, platform-browser-dynamic
... router
Ivy Workspace: Yes

Package                           Version
```

2. By using browser's developer tool (Press F12).



3. By using the package.json file.



Q21. What are the best IDE for Angular?

Ans. The best IDEs for Angular are the following.

Q22. How can we run two Angular projects simultaneously?

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

ng serve

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

ng serve --port 4210

In place of 4210 you can any port number.

Q23. What is Bootstrapping (bootstrap) in Angular?

Ans. The Bootstrap is the root AppComponent that Angular creates and inserts into the host web page i.e. “index.html”.

index.html

src > index.html > ...

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8">
5    <title>Myapp</title>
6    <base href="/">
7    <meta name="viewport" content="width=device-width, initial-scale=1">
8    <link rel="icon" type="image/x-icon" href="favicon.ico">
9  </head>
10 <body>
11   <app-root></app-root>
12 </body>
13 </html>
14
```

You can also put more than one component tree on the web page of host, that's not typical. Most of the applications have only one component tree and they bootstrap a single root component and you can call the one root component you want but most developers call the AppComponent.

In bootstrapping process, components list is created in the bootstrap array and inserts each into the browser (DOM).

The Angular default Module i.e. NgModules helps us to organize an application into connected blocks of functionality.

The NgModule properties for the minimum "AppModule" generated by the CLI which follows as:

- Declarations — It is use to declare the application components.
- Imports — Every application must import BrowserModule to run the app in the browser.
- Providers — There are none to start.
- Bootstrap — This is the root AppComponent that Angular creates and inserts into the index.html host web page.

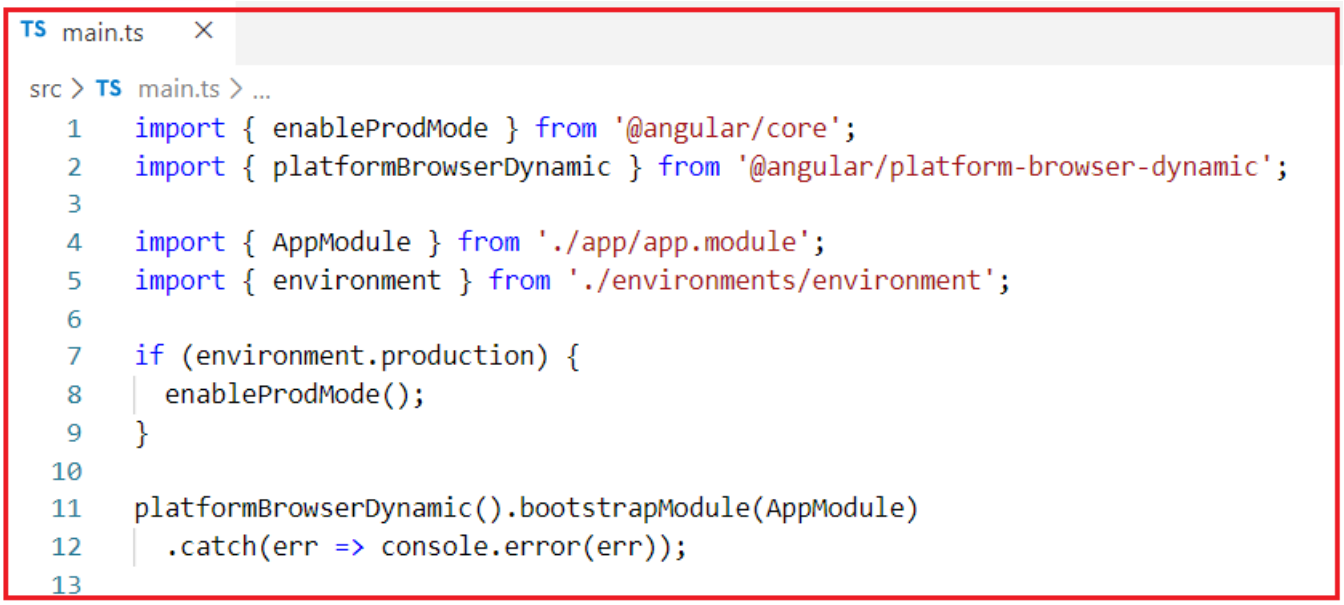
app.module.ts –

app.module.ts

src > app > app.module.ts > ...

```
1   import { BrowserModule } from '@angular/platform-browser';
2   import { NgModule } from '@angular/core';
3
4   import { AppRoutingModule } from './app-routing.module';
5   import { AppComponent } from './app.component';
6   import { LoginComponent } from './login/login.component';
7   import { SignupComponent } from './signup/signup.component';
8
9   @NgModule({
10    declarations: [
11      AppComponent,
12      LoginComponent,
13      SignupComponent
14    ],
15    imports: [
16      BrowserModule,
17      AppRoutingModule
18    ],
19    providers: [],
20    bootstrap: [AppComponent]
21  })
22  export class AppModule { }
```

By default, Bootstrap file is created in the folder "src/main.ts" and "main.ts" file is very stable. Once you have set it, you never change it again and its looks like –



```
TS main.ts
src > TS main.ts > ...
1   import { enableProdMode } from '@angular/core';
2   import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
3
4   import { AppModule } from './app/app.module';
5   import { environment } from './environments/environment';
6
7   if (environment.production) {
8     enableProdMode();
9   }
10
11  platformBrowserDynamic().bootstrapModule(AppModule)
12    .catch(err => console.error(err));
13
```

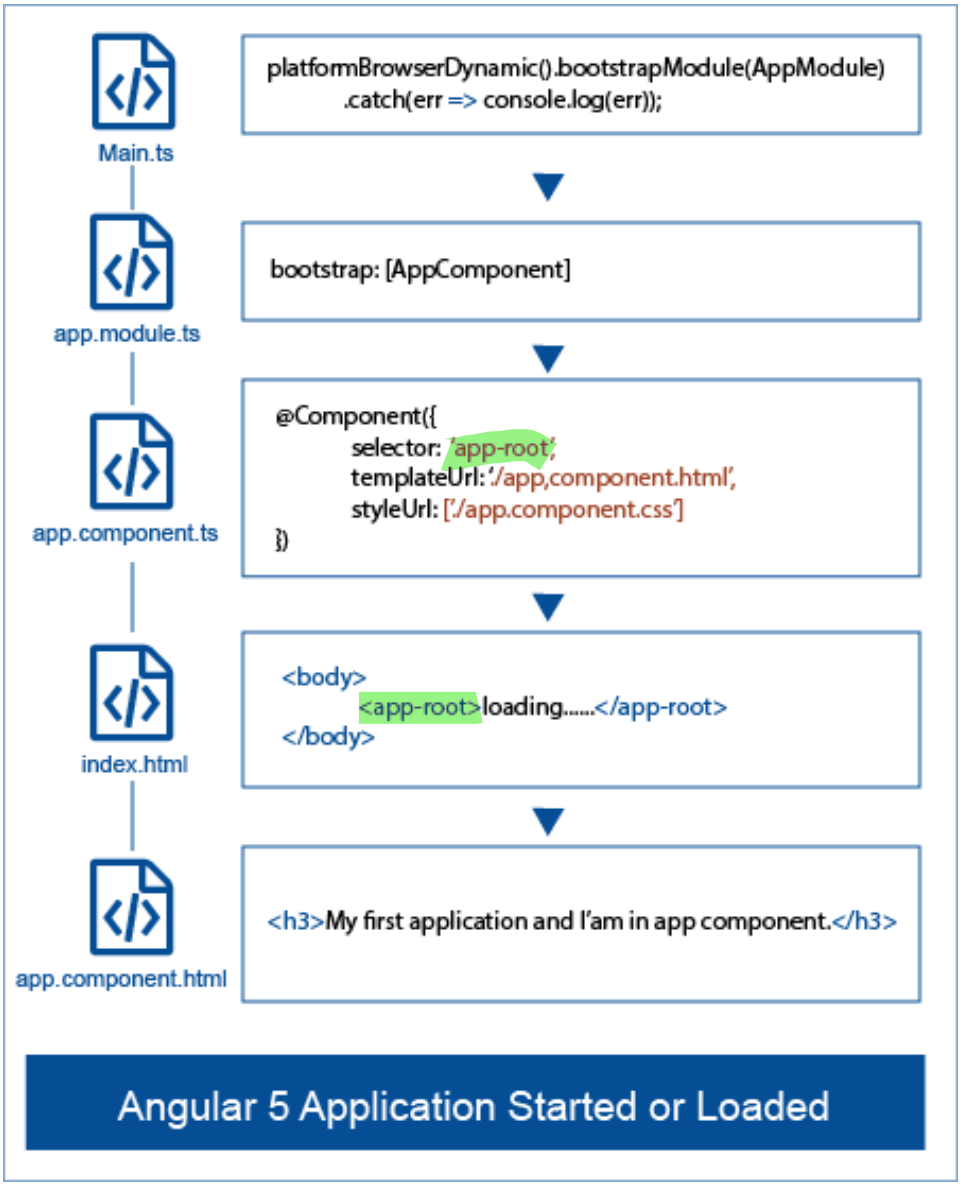
Q24. What is bootstrapping in Angular? Is it possible to start Angular in any other way rather than app.module? If yes, then how?

Ans. Bootstrapping means starting an Angular application. Yes, it is possible to start angular in any other rather than app.module. We can start Angular using our defined component instead of "app.module". For this, we must follow the following steps.

1. Make a module file, say "my.module.ts", in app folder.
2. Make a component using command "*ng g c my*" in app folder. Instead of c you can also write component in command.
3. Go to "my.module" file and register your component in this file and use "MyComponent" in declaration and bootstrap.
4. Go to the "main.ts" file, put your module name "MyModule" into "bootStrapModule" method.
5. Go to "index.html" file and put the my component tag inside body tag.

Q25. How does an Angular application get start?

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


Main.ts
platformBrowserDynamic().bootstrapModule(AppModule)
.catch(err => console.log(err));
app.module.ts
bootstrap: [AppComponent]
app.component.ts
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrl: ['./app.component.css']
})
index.html
<body>
<app-root>loading.....</app-root>
</body>
app.component.html
<h3>My first application and I'am in app component.</h3>
Angular 5 Application Started or Loaded

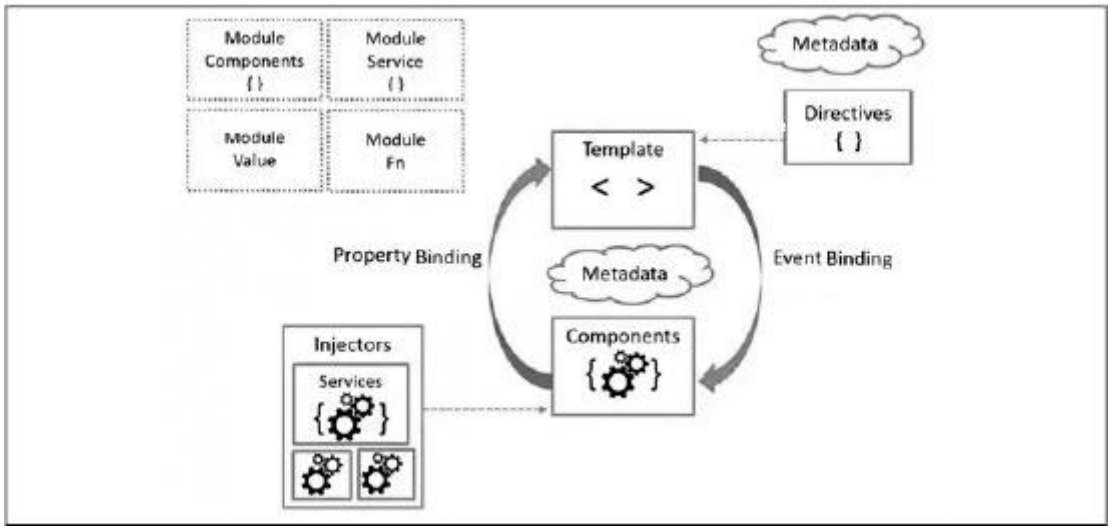
Q26. What is the Architecture Overview of Angular?

Ans. Angular is the most popular web development framework for developing mobile and desktop applications.

The Angular framework is also used in cross-platform mobile development called IONIC so it is not limited to web applications only.

Angular is an open source framework written and managed by the Angular team of Google and the
Father of Angular is Misko Hevery.

The bootstrapping process creates the components listed in the bootstrap array and inserts each into the browser (DOM).



With the help of above image, you can identify the seven main building blocks of an Angular Application, which are:

1. Component
2. Templates
3. Metadata
4. Data Binding
5. Directives
6. Services
7. Dependency Injection

The basic building blocks of an Angular application are NgModules, which provide a compilation context for components.
Angular application is defined by a set of NgModules and always has at least a root module that enables bootstrapping, and many more feature modules.

8. Components define views

9. Components use services

Introduction to Modules

The NgModule is a class and work with the @NgModule decorator function and also accepts a metadata object that tells Angular how to compile and run the module code.

The purpose of the module is to declare everything that is created in Angular and group them together.
The NgModule is used to simplify the ways of defining and managing the dependencies in applications and you can also consolidate different components and services in associative blocks of functionality.

Introduction to Components

Components are the basic building block of a user interface (UI) in Angular applications and it controls views (HTML/CSS). They also communicate with other components and services to provide functionality to your applications.
Introduction to Templates
The Template is only a subset of HTML, which tells Angular how to display the HTML view. Templates are created using the normal HTML tags and also use the Angular-specific markup such as interpolation or Property binding.
Interpolation – {{}} double-curly braces
Property binding – []
Most of all HTML syntax is a valid template syntax.

A template expression produces a value. Angular runs the expression and assigns it to a property of a binding target. The target can be HTML elements, components, or directives.

The <script> element is a notable exception. It is prohobited, eliminating the risk of scripting attacks.

In practice, <script> is ignored and a warning is displayed in the browser console.

The <html>, <body>, and <base> elements do not have a useful role.

```
Interpolation double-curly braces - {{...}}
```

You have seen the double curly braces of interpolation, {{and}}, early in your Angular education.

```
<p>Hello, {{student.name}} </p>
```

1. **Template**

A template is a piece HTML code that tells Angular how to render the component in angular application.

The template is immediately associated with the component that defines the view of that component.

Types of Templates

There are 2 ways of defining template in an angular component.

Inline Template

The inline template is defined by placing the HTML code in back ticks “and is linked to the component metadata using the template property of @Component decorator.

Template File

The template is defined in a separate HTML file and is linked to the component metadata using the templateUrl property of the @Component decorator.

Introduction to metadata

It tells Angular how to compile and run the application.

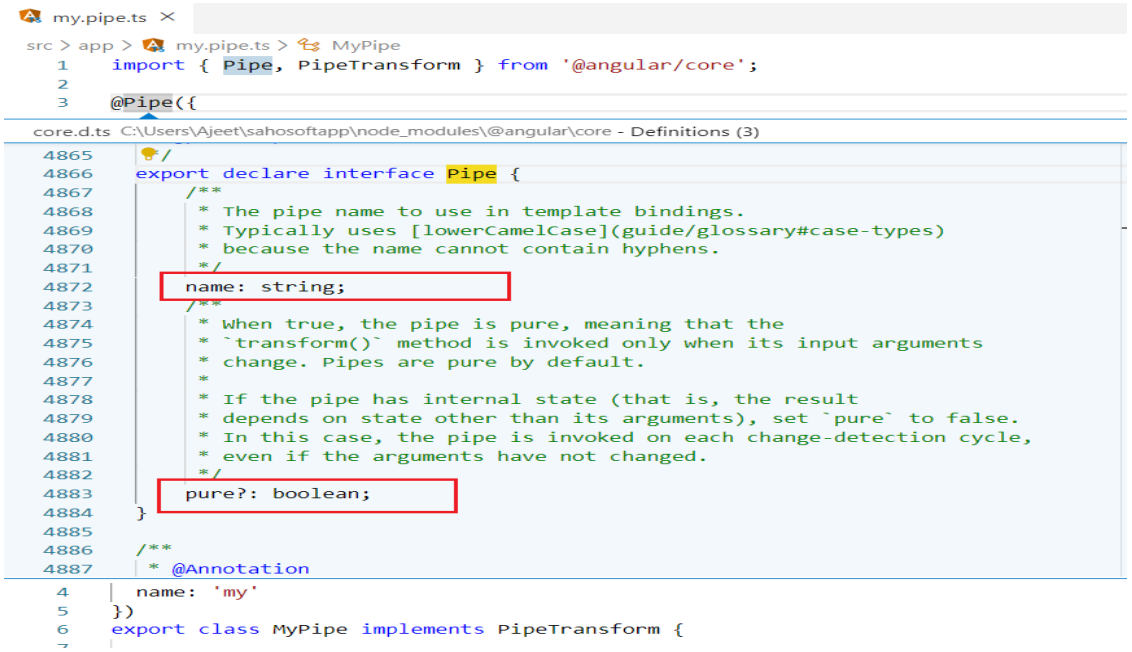
Data binding

Angular uses Data Binding to get the data from the Component to Template View using Template Syntax.

Angular supports four types of Data binding

Interpolation: It is used to bind the data from component to View.
Property Binding: It is used to bind the data from component to the property of an HTML control in the view.
Event Binding: The DOM Events are bind from View to the Component method.
Pipes – Pipes transform displayed values within a template.

Use the @Pipe annotation to declare that a given class is a pipe. A pipe class must also implement a PipeTransform interface.
The @Pipe decorator allows you to define the pipe name that is globally available for use in any template across Angular application.
Pipe decorator and metadata –



Directives

Angular Directive is a TypeScript class which is declared as a @directive decorator.

The directives allow you to attach behavior with the DOM elements and the @directive decorator

provides you an additional metadata that determine how directives should be processed, instantiated, and used at run-time.

Services

Services are commonly used to store data and make HTTP calls. The main idea behind a service is to provide a simple way to share data between components and with the help of dependency injection (DI) you can control how the instances of service are shared.

Dependency injection (DI)

Dependency Injection is a powerful pattern for managing code dependencies. DI is a way to create objects that depend upon other objects.

Angular has its own DI framework pattern, and you can't really build an Angular application without Dependency injection (DI).

Q27. What are the differences between Interpolations vs. Property Binding?

Data Binding: The Data binding helps you to communicate between the component and template for rendering the views.

There are two types of Data Binding:

1. One-way data binding
2. Two-way data binding

- One-Way Data-Binding: In one-way data binding, value flows in one direction from a component's data property into a target element property.
- Two-Way Data-Binding: Two-way bindings synchronize the data between the model
- and the view.

Two-way data binding means that any data-related changes affecting the model are immediately propagated to the matching views, and any changes made in the views are immediately reflected in the corresponding template.

Example of **Interpolation**

app.component.ts

src > app > app.component.ts > ...

```
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-root',
5     template: `<h1l>{{ empName }}</hi>
6   <img src='{{empPic}}' style="height :40px”/>`,
7     styleUrls: ['./app.component.css']
8   })
9   export class AppComponent {
10    title = 'myapp';
11
12    empName: string = 'My Name is Ajeet Singh';
13    empPic: string = 'http://www.sahosoft.com/Angular/Expert_Ajeet.jpg';
14
15  }
16
17
```

- Property Binding: The target property in the following code is the image src property of the element.

app.component.ts

src > app > app.component.ts > ...

```
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-root',
5     template: `<h1l [innerHtml]="empName" ></hi>
6
7   <img [src]="empPic" style="height :40px”/>`,
8     styleUrls: ['./app.component.css']
9   })
10  export class AppComponent {
11    title = 'myapp';
12
13    empName: string = 'My Name is Ajeet Singh';
14    empPic: string = 'http://www.sahosoft.com/Angular/Expert_Ajeet.jpg';
15
16  }
17
```

You can also use alternative of bind- prefix, knows as the canonical form.

app.component.ts

```
src > app > app.component.ts > ...
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-root',
5     template: `<h1l [innerHtml]="empName" ></hi>
6
7   <img bind-src="empPic" style="height :40px”/>`,
8     styleUrls: ['./app.component.css']
9   })
10  export class AppComponent {
11    title = 'myapp';
12
13    empName: string = 'My Name is Ajeet Singh';
14    empPic: string = 'http://www.sahosoft.com/Angular/Expert_Ajeet.jpg';
15
16  }
17
```

The target name is always the name of a property, even when it appears to be the name of something else.
Angular data binding sanitizes the values before displaying on DOM. It never allows HTML or script tags to leak the security into the browsers. It will never warn for dangerous HTML, script tags and render the content harmlessly.
Both Interpolation and Property Binding sanitize the malicious content.

Attribute, class, and style bindings

- **Attribute binding:** This binding starts with the attr prefix, followed by a dot (.), and the name of the attribute.

Example–

app.component.html

```
src > app > app.component.html > ...
1   <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3   <table border="1">
4       <!– The below expression calculates colspan=2 –>
5           <tr>
6               <td [attr.colspan]="1 + 1">Gender</td>
7           </tr>
8           <!-- GOT ERROR, If TRY BELOW LINE OF CODE AND ERROR IS - There is no ‘colspan‘ property to set!
9           <tr>
10              <td colspan="{{1 + 1})}">Gender</td>
11          </tr> -->
12  </table>
13
```

- Class binding: You can add and remove element’s CSS classes with the help of class binding.

Example-

<div class=" row" [class.row] ="! IsActive">This one is not active row!

- **Style binding:** This binding starts with style prefix, followed by a dot (.), and the name of the CSS style property

Example-

```
<button [style.color]="IsActive ? 'green': 'red' ">This is a green!</button>
```

And

```
<button [style.font-size.%J="!IsActive ? 50 : 100" >This is not active!</button>
```

Note: To set an element property to a non-string data value, you must use the property binding. Otherwise, Angular perform the process of converting your interpolation to property binding. Therefore, it will not be good for your application.

Q28. What Is the class Decorator?

Ans. Without using class decorator, AppComponent is just a class. There is nothing Angular in without it. It is the decorator, which tells angular how to treat the class.

Angular currently has 7 class decorators -

@Component
@Directive
@Injectible
@NgModule
@Pipe

Q29. What is webpack?

Ans. Webpack is a tool used by Angular CLI to bundles JavaScript files and stylesheets. Webpack injects these bundles into the index.html file at runtime.

Q30. Which files are bundled and injected in index.html at runtime by webpack?

Ans. At runtime following files are bundled and injected in index.html.

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

Q31. Can we create the Angular application without using Angular CLI?

Ans. Yes, with the help of Visual Studio we can create Angular application. Any version of VS can do; like - Visual studio 2015 or Visual Studio 2017 or Visual Studio 2019 or later versions.

Q32. What is npm and what is the need for Angular application?

NPM (Node Package manager) is the package manager which we use to install the dependencies required for the application. In Angular, we have package.json file in which all the dependencies have added which we have installed. When we need any new dependencies, we can simply install using the following command.

npm install <package name>
It adds the installed package in the node_modules folder and also, adds the dependencies in package.json file.

Q33. How Webpack is different to SystemJS?

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

Q34. What is the difference between AOT and JIT?

Ans. In Angular we have AOT (Ahead of Time) and JIT (Just In Time) Compilation to compile the application. There are some differences between JIT and AOT:

1- JIT stands for Just In Time, so with the name you can easily understand that it compiles the code just in time means in the browser, while AOT stands for Ahead of Time, so according to the name it compiles the code at the time of building.

2- According to compilation, JIT loads the application slowly while AOT loads the application faster than JIT.

3- In case of JIT, Template Binding errors are shown at the time of showing the application, while in case of AOT Template binding errors will show at the time of building.

4- In case of JIT, bundle size is more than AOT because in case of AOT the bundle size is half of the JIT.

Q35. What is the concept of transpilation in Angular?

Ans. Transpiling is the process of converting the code from one high level language to another high level language.

All modern browsers only understand JavaScript and in Angular we write all the codes in TypeScript. So, in Angular, transpilation means that what you wrote in TypeScript is converted to another high-level language, which is JavaScript.

Q36. In @NgModule can we add more than one component in bootstrap?

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

Q37. How we can run Angular project? Which is the default port used by Angular?

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

Q38. What is the difference between ng serve and ng serve --open?

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

Q39. What are the commands for the following?

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

Q40. What is the purpose of main.ts file?

Ans. The main.ts file is the main file which is the starting point of application. As you have read before about the main method, the same concepts are here in the Angular application. It is the file for the bootstrapping the application by the main module as .bootstrapModule (AppModule). It means that based on the main.ts file, Angular loads this module first.

Q41. What is the role of package.json file?

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

Q42. How package.json file is different from package-lock.json file?

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

- ✓ ✓This file must be confirmed in the source repositories and has several purposes:
- ✓ Describe a unique representation of a dependency tree so that teammates,
- ✓ deployments and continuous integration are guaranteed to install exactly the same dependencies.
- ✓ Provide users with the facility to "time-travel" to previous states of node_modules without having to commit the directory itself.
- ✓ To facilitate greater visibility of tree changes through readable source control differences.
- ✓ And simplify the installation process by allowing npm to skip repeated metadata resolutions for previously-installed packages.
- ✓ And simplify the installation process by allowing npm to skip repeated metadata resolutions for previously installed packages.

Q43. What is the role of tsconfig.json file?

Ans. The tsconfig.json is the configuration file and there are many setting for TypeScript compiler. According to these settings the TypeScript code is compiled into Javascript so that the browser can understand it.

Q44. What is the role of angular.json file?

Ans.The angular.json is the configuration file of Angular application where there are many configuration settings necessary for the Angular application.

Q45. What is the use of “assets” folder in Angular?

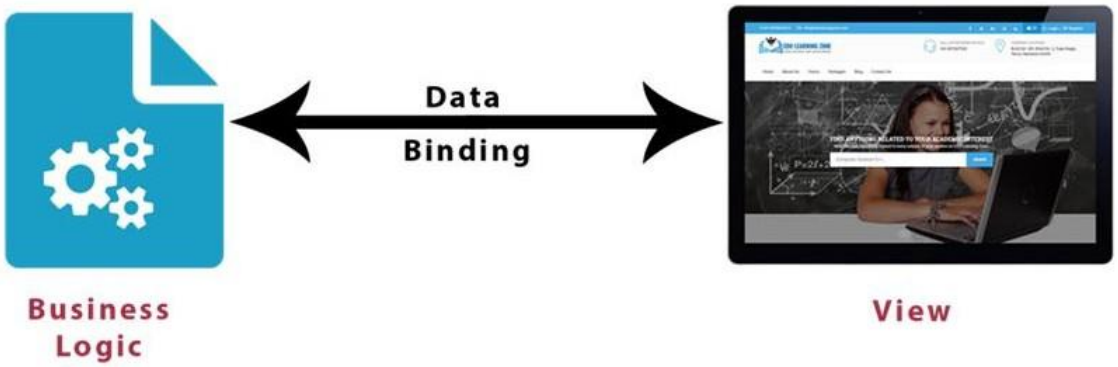
Ans. Whenever we build our Angular application with "npm run build" or "ng build --prod" commands, then Angular CLI moves all the assets into the dist folder. It will do the same when it

sees that there are images in the assets folder.

So, we can say that assets folder is used in Angular to maintain assets like image etc. hat should not be changed during compilation.

Q46. What is data binding in Angular?

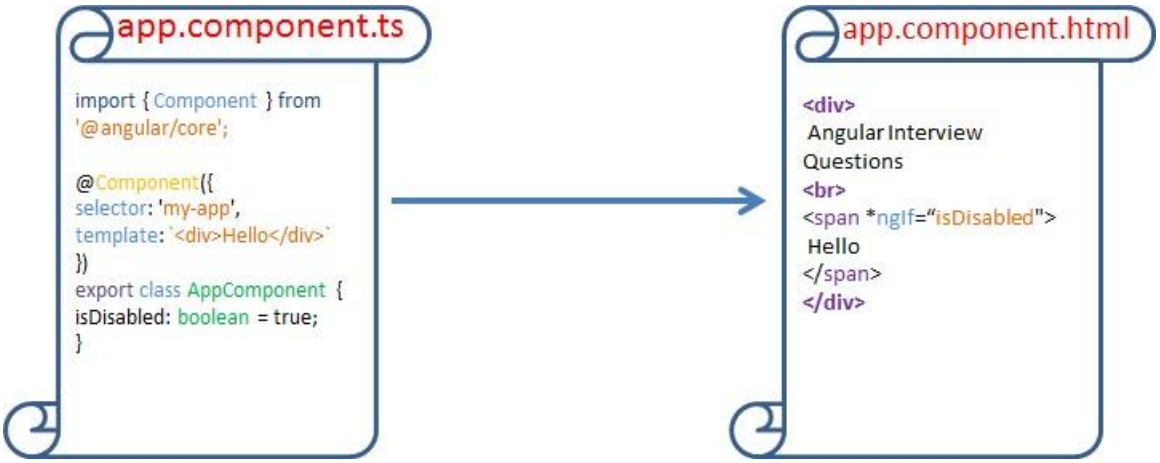
Ans. Data binding is the process of establishing a connection between the application UI and business logic. Basically, it acts as a bridge between UI (View) and the business logic (View Model) for the application.



Q47. How many types of binding are supported by Angular?

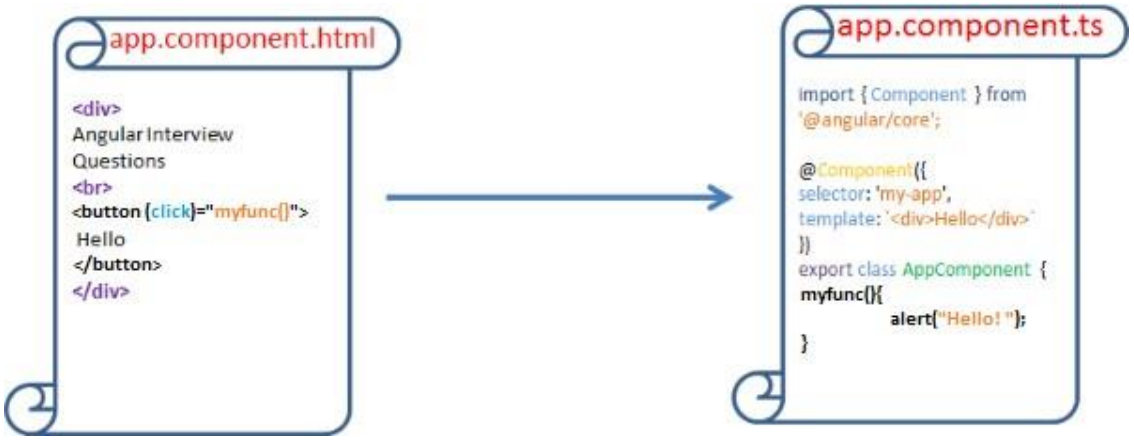
Ans. On the basis of data direction (i.e. movement of data), binding is divided into 3 parts.

1. Source to View



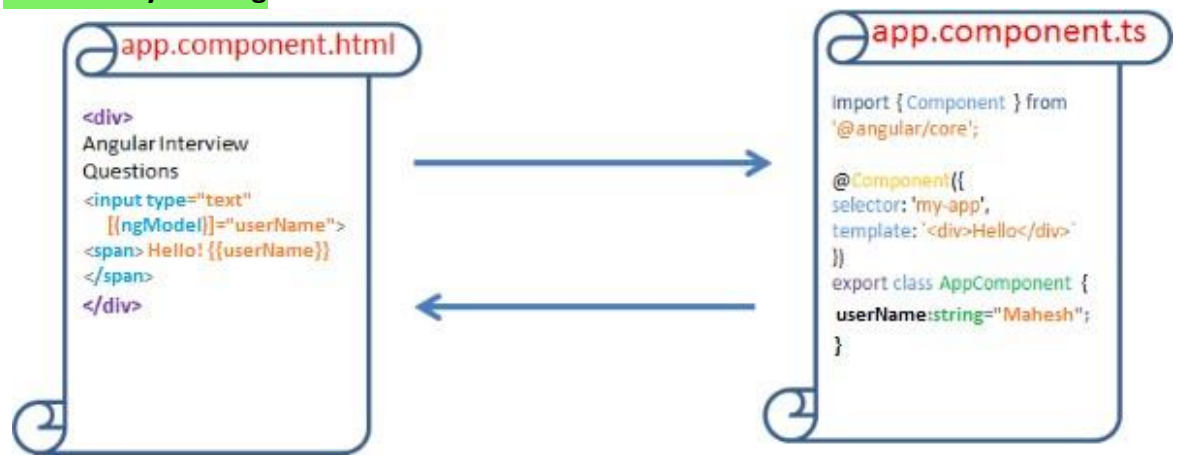
- **INTERPOLATION**
- **PROPERTY BINDING**
- **CLASS BINDING**
- **ATTRIBUTE BINDING**
- **STYLE BINDING**

2. View to Source



EVENT BINDING

3. Two Way Binding



Q48. What is Interpolation? Give an example.

Ans. Interpolation allows you to define the properties in a component class, and communicate these properties to and from the template. This is the simplest form of data binding. It also helps to solve expression.

For example –

app.component.html

src > app > app.component.html > ...

```
1  <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3  <div>
4      Employee Name :- {{empName}}
5      <!-- example of string interpolation> -->
6      Sum of 1 and 2 is :- {{1+2}}
7  </div>
8
```

Q49. What is Property binding? Where you can implement property binding?

Ans. In this binding, we pass the data from the Component (.ts file) to View (HTML file) to set the value of given element.

Property binding can be implemented on the following targets.

Element Property - Implement property binding on DOM elements like

app.component.html

src > app > app.component.html > ...

```
1  <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3  <div>
4      <img [src]="empImage"> <!-- myPath property declare in component -->
5  </div>
6
7
```

Component Property - Implement property binding on input elements (want to pass property from Parent to Child Component) as

app.component.html

src > app > app.component.html > ...

```
1  <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3  <div>
4      <app-employee [childValue]="myValue"></app-employee>
5  </div>
6  <!-- app-employee is a child selector which we render into app.component.html
7  file, and using property binding ([childValue]), pass the value -->
8
9
```

Directive Property - Implement property binding on directive, like

app.component.html

src > app > app.component.html > ...

```
1  <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3  <div>
4      <p [ngClass]="'one two'">Angular Interview Questions by sahosoft</p>
5  </div>
6  <!-- ngClass is a directive and we implement a class using property binding. -->
7
```

Q50. In how many ways you can achieve property binding? What is the benefit of using property binding?

Ans. You can achieve property binding by 3 ways.

app.component.html

src > app > app.component.html > ...

```
1  <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3  <!-- Achieve Property Binding by the following 3 ways -->
4  <img src="{{myImagePath}}">
5  <img [src]="myImagePath">
6  <img bind-src="myImagePath">
7
```

You can use the first method, that is, property binding using interpolation if and only if the value (myImagePath) that you are defining is a string, otherwise use other 2 methods.
The advantage of property binding is that you can easily control element property value of a View from the Component (.ts file) and change as per your requirement.

Q51. When do we use the Property binding and when we use Interpolation?

Ans. When rendering data values as strings, we can choose any, there is no difference. However, when we want to set an element property to a non-string data value, you must use the property binding.

app.component.ts

src > app > app.component.ts > ...

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

@Component({
  selector: 'app-root',
  template: `<div>
  <button [disabled]='isDisabled'> Sahosoft Solutions </button>
  </div>`,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'myapp';
  isDisabled: boolean = true;
}
```

Q52. What is Class binding? Give an example.

Ans. Class binding is used when we want to add or remove CSS class names from an element's class attribute based on some condition.

Syntax of class binding is very similar to property binding except that it starts with the prefix 'class', optionally followed by a dot(.) and the name of a CSS class. For example-

app.component.html

src > app > app.component.html > ...

```html
<h2>Sahosoft Online Classes (sahosoft.com)</h2>

<button class="btn" [class.btn-primary]="isApplied" >Hello</button>

```

Here, isApplied is a boolean property defined within the Component.

Q53. What is Style binding? Give an example.

Ans. Whenever we bind a component field to our inline HTML styles, is called style binding. Suppose, if we want to apply an inline style based on some condition, then we use Style binding.

Syntax of style binding is very similar to class binding, except that it starts with the prefix "style", optionally followed by a dot(.) and the name of a style. For Example-

app.component.html

src > app > app.component.html > ...

```
1   <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3   <button class="btn" [style.backgroundColor]="isApplied?'green':'red'">
4       Sahosoft Solutions
5   </button>
6
```

Here, isApplied is a boolean property defined within the Component.

Q54. What is Attribute binding? What is benefit of using it?

Ans. Attribute binding is used to bind the attribute of an element with the component field. It is useful when we need to bind attribute, that is, attribute cannot be bind with property or interpolation method. For example –

app.component.html

src > app > app.component.html > ...

```
1   <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3   <table>
4       <tr>
5           <td colspan="{{1 + 1}}">Employee's Record</td>
6
7       </tr>
8       <!-- Above line generates an error, because "colspan" is an attribute
9   not a property, so, we bind it like -->
10      <tr>
11          <td [attr.colspan]="1+1">Employee's Record</td>
12      </tr>
13  </table>
14
```

Q55. What is Event binding? Give an example.

Ans. Event binding is used to manage events generated from the DOM like keystrokes, mouse movements, clicks, etc. Or we can say that whenever you want to pass data from View (HTML) to Component event binding is used.

Syntax of event binding consist of a target event name within parentheses on the left of an equal sign, and a quoted template statement on the right.

For Example-

The following event binding listens for the button click events, calling the component's callMyMethod( ) method each time a click occurs.

app.component.html

```
src > app > app.component.html > ...
1   <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3   <button class="btn btn-primary" (click)="callMyMethod()">
4       Click Me
5   </button>
6
```

Q56. Where you can implement event binding?

Ans. You can implement event binding on the following targets.

Element Event - Implement event binding on DOM elements,Ex-

app.component.html

```
src > app > app.component.html > ...
1   <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3   <button (click)="callMyMethod()"> Click Me </button>
4
```

Component Event - Implement event binding on output elements (want to pass property from Child to Parent Component) Ex –

app.component.html

```
src > app > app.component.html > ...
1   <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3   <app-employee (deleteEmployee)="deleteRecord()"></app-employee>
4
```

Directive Event - Implement event binding on directive, Ex

app.component.html

```
src > app > app.component.html > ...
1   <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3   <div (myClick)="clicked=$event" clickable>click me</div>
4
```

Q57. Name some common events that you can use for event binding?

Ans. Some common events are:

focus, blur, submit, scroll, cut, copy, paste, keydown, keypress, keyup, mouseenter, mousedown, mouseup, click, dbclick, etc.

Q58. What is Two-way binding? Give an example.

Ans. Two-way binding means changes in the View (UI) can automatically change in the template (Component's field) and vice-versa. The syntax of two way binding is different from other binding. Angular use brackets enclosed in square brackets to achieve this functionality, such as [()].

To achieve two-way binding, we need NgModel directive.
Example:

app.component.html

```
src > app > app.component.html > ...
1   <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3   <input type="text" [(ngModel)]="empName" placeholder="Enter your name" />
4   <p>{{empName}}</p>
5
```

Q59. What is template reference variable?

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

A template reference variable is a way to acquire a reference to a specific element, component, directive, and pipe so that it can be used in the same template HTML.
A reference variable must be declared using the hash symbol (#).
The Angular components and directives only match selectors for classes that are declared in the Angular module.

Q60. In how many ways we can create template reference variable?

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

Syntax of Template Reference Variable –

You can use a template reference variable by two ways.

- Using hash symbol (#)
- Using reference symbol (ref-)

The following examples of specifying a template reference variable using Input Text Box –
I have declared a reference variable "cellnumber" using the hash symbol (#) and the reference symbol
(ref-).

app.component.html

```
src > app > app.component.html > ...
1   <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3   <input type="text" ref-cellnumber> //cellnumber will be a template reference variable.
4
```

And

app.component.html

```
src > app > app.component.html > ...
1   <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3   <input #cellnumber placeholder="Cell number"> //cellnumber will be a template reference variable.
4
```

A reference to the input element is created that can be used later in template and the scope of the "cellnumber" variable is the entire HTML template in which the reference is defined.
To use the reference to get the input value, for example –

app.component.html

```
src > app > app.component.html > ...
1   <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3   //cellnumber refers to the input element
4   <button (click)="show(cellnumber)">click to see</button>
5
```

In the following code, the variable "cellnumber" refers to the instance of the HTMLElement object for the input –

app.component.ts

src > app > app.component.ts > ...

```
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-root',
5     templateUrl: './app.component.html',
6     styleUrls: ['./app.component.css']
7   })
8   export class AppComponent {
9     title = 'myapp';
10
11    show(cellnumber: HTMLInputElement) {
12      console.log(cellnumber.value);
13    }
14  }
```

You can use the ViewChild decorator for reference, inside your component.

app.component.ts

src > app > app.component.ts > ...

```
1   import { Component } from '@angular/core';
2
3   import { ViewChild, ElementRef } from '@angular/core';
4
5   @Component({
6     selector: 'app-root',
7     templateUrl: './app.component.html',
8     styleUrls: ['./app.component.css']
9   })
10  export class AppComponent {
11    title = 'myapp';
12
13    // Reference cellnumber variable inside Component
14    @ViewChild('cellnumber') celliInputRef: ElementRef;
15
16
17  }
```

And finally, you can use this.nameInputRef anywhere in your component class.

app.component.ts

src > app > app.component.ts > ...

```
1  import { Component } from '@angular/core';
2
3  import { ViewChild, ElementRef } from '@angular/core';
4
5  @Component({
6    selector: 'app-root',
7    templateUrl: './app.component.html',
8    styleUrls: ['./app.component.css']
9  })
10 export class AppComponent {
11   title = 'myapp';
12
13   contactNumber: number;
14   // Reference cellnumber variable inside Component
15   @ViewChild('cellnumber') celliInputRef: ElementRef;
16
17   show() {
18     this.contactNumber = this.celliInputRef.nativeElement.value
19   }
20
21 }
```

Q61. How to bind to user input events to Component event Handlers?

Most of the DOM events are triggered by user input, and binding to these events provides a way to get inputs from the user.

The following example shows the click event binding – [app.component.ts]

app.component.ts

src > app > app.component.ts > ...

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

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

  onclick() {
    this.welcomeMsg = 'Welcome you, Ajeet Kumar Singh!';
  }

}
```

In app.component.html –

app.component.html | app.component.ts

src > app > app.component.html > ...

```html
<h2>Sahosoft Online Classes (sahosoft.com)</h2>

<div class="msg">
    <button (click)="onClick()">Click Me!</button>
    <p>
        {{welcomeMsg}}
    </p>
</div>
```

OR

app.component.html × app.component.ts

src > app > app.component.html > ?

```
1  <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3  <!-- Canonical form, the (on-) prefix alternative -->
4      <div class="msg">
5          <button on-click="onClick(Sevent)">Click Me!</button>
6          <p>
7              {{welcomeMsg}}
8          </p>
9      </div>
```

When user clicks the button, Angular calls the onClick method from OnClickComponent.

Q62. How to get user input from the $event object?

The DOM events contain all useful information that are needed in the component.
The following example shows how to get user input from the $event – keyup.
In component.ts

app.component.ts ×

src > app > app.component.ts > ...

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    values = '';
10
11   //KeyUp events.
12   onKeyUp(event: any) {
13     this.values += event.target.value + ' : ';
14   }
15
16 }
```

In component.html –

app.component.ts × app.component.html ×

src > app > app.component.html > ...

```
1   <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3   <div>
4       <button (click)="onKeyUp($event)">KeyUp Event !</button>
5       <p>
6           {{values}}
7       </p>
8   </div>
9
```

Q63. How to get user input from a Template Reference Variable?

This is the other way to get the data from user. It is also called #var.
"A template reference variable is primarily a reference to a DOM element within a template. It can also be reference to Angular components or directives and others."
For Ex-

app.component.html ×

src > app > app.component.html > ...

```
1   <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3   <input #name placeholder="Enter Name">
4
```

The following example shows how to get user input from a template reference variable –

In app.component.ts

app.component.ts ×

src > app > app.component.ts > ...

```
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-root',
5     templateUrl: './app.component.html',
6     styleUrls: ['./app.component.css']
7   })
8   export class AppComponent {
9
10
11  }
```

In app.component.html –

app.component.ts app.component.html

src > app > app.component.html > ...

```
1   <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3   <div>
4       <button #keydownVal (keydown)="0"></button>
5       <p>
6           {{keydownVal.value}}
7       </p>
8   </div>
9
```

Q64. How to Create a Custom Validator for both Model Driven and template driven forms?

There are 2 types of Validators –

1) Built-in Validators
2) Custom Model Form Validators

Ex:
a. Email Validator
b. Password Validator
c. Secure Site Validator
d. Credit card validator

Built-in Validators -
Validators .required: Requires a form control to have a non-empty value.
Validators .minlength: Requires a form control to have a value of a minimum length.
Validators .maxlength: Requires a form control to have a value of a maximum length.
Validators .pattern: Requires a form control's value to match a given regex.

Q65. Can we use template reference variable using select (combo box)? If yes, then how?

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

app.component.html

src > app > app.component.html > ...

```
1   <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3   <select #myValue (change)="setStudentRecord(myValue. value)"> </select>
4
```

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

Q66. How Angular ensure content security at the time of binding?

Ans. Angular sanitizes the values before displaying them. It will not allow HTML with script tags to leak into the browser, neither with interpolation nor with property binding.
For example –
myTitle= 'Template <script>alert("evil never sleeps")</script>Syntax'; Then, after sanitization it removes the script tag.

Q67. What is ngModel? What is its role?

Ans. Angular is no longer provided with built-in two-way data binding, but with APIs that allow to implement this type of binding using property and event bindings. ngModel is provided as a built-in directive as part of the FormsModule to implement two-way data binding and should be preferred when building components that serve as custom form controls.

Q68. Which binding we will use for dynamic css?

Ans. NgStyle and NgClass we are using for dynamic css. For example:
NgStyle:

app.component.html

src > app > app.component.html > div > div

```
1  <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3  <div [ngStyle]="{'background-color':person.country === 'UK' ? 'green' : 'red' }">
4      Hello
5      <div>
6
```

In the above code, we are checking the country value that is of person object and shows the style based on condition.

NgClass:

app.component.html

src > app > app.component.html > div > div

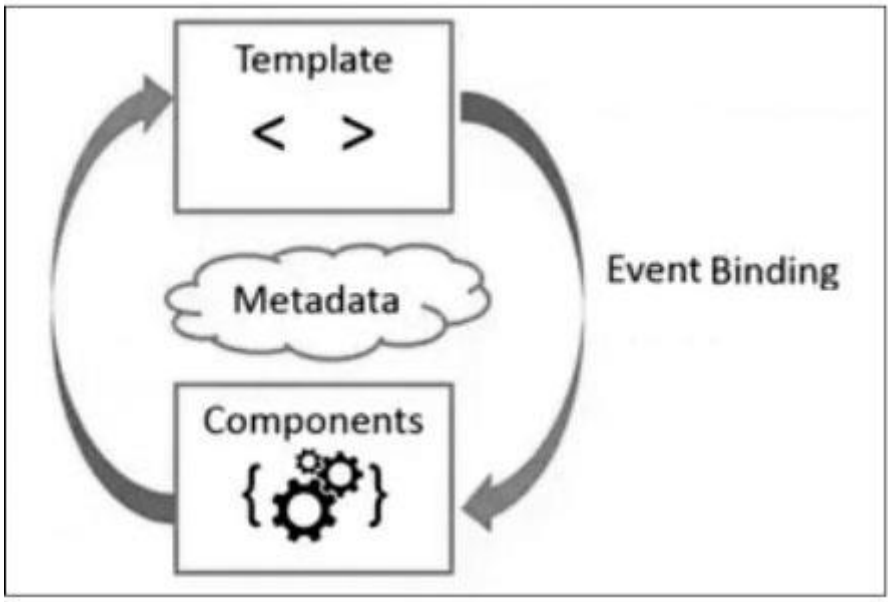
```
1  <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3  <div [ngClass]="{'text-success':person.country === 'UK' }">
4      Hello
5      <div>
6
```

As you have seen in the above code, we are verifying the country value and according to value applying the class using Ngclass.

Q69. What is a component in Angular?

Ans. Components are the basic building block of user interface (UI) in Angular applications and

it controls views (HTML/CSS). They also communicate with other components and services to provide functionality to your applications. Components are basically, Typescript classes that interact with the component's HTML files, which are displayed on the browsers. The component is the core functionality of Angular applications, but you need to know how to pass the data to the components to configure them. Angular applications must have a root component that contains all other components.



Components are created using @Component decorator which is part of the @angular/core module.
You can also create your own project using Angular CLI, this command will allow you to quickly create an Angular application such as - generate components,
services, pipes, directive, classes, and modules, etc. as per your requirements.
Create your own component (login) using the following command line –

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Ajeet\sahosoftapp> ng g c login
```

After executing the above Angular CLI command in the project directory, the output will look like this–

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Ajeet\sahosoftapp> ng g c login
CREATE src/app/login/login.component.html (20 bytes)
CREATE src/app/login/login.component.spec.ts (621 bytes)
CREATE src/app/login/login.component.ts (271 bytes)
CREATE src/app/login/login.component.css (0 bytes)
UPDATE src/app/app.module.ts (608 bytes)
PS C:\Users\Ajeet\sahosoftapp>
```

And by default the application login files are created and it looks like this –

- login.component.html
- login.component.spec.ts
- login.component.ts
- login.component.css
- app.module.ts

In addition, Angular CLI commands also import the Login component in the Angular module. See the example in detail:

login.component.ts:

login.component.ts

src > app > login > login.component.ts > ...

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

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

  constructor() { }

  ngOnInit(): void {
  }

}
```

Above component class shows some of the most useful @Component configuration options:

- Selector
- TemplateUrl
- Style URLs

The selector – It is a CSS selector that allows Angular to create an instance of this component wherever it finds the corresponding tag in template HTML. For example, it is - <app-login></app-login>.

The templateUrl – It is the module-relative address of this component's HTML template and you can also provide inline HTML template.

The styleUrls - It can be used for CSS rules and will affect the style of the template elements and also provide an inline style CSS.

Summary:
Components are the fundamental building blocks of UI in Angular applications and communicates with other components and services to provide functionality to applications.

- It is the core component of Angular applications.
- An angular application must have a root component that contains all other components.
- It contains well-defined selector.
- It contains well-defined styles and styleUrls.
- It contains well-defined template and templateUrl.
- It have well-defined inputs and outputs.
- It have well-defined encapsulation and animations.
- It have a well-defined lifecycle.
- It is a self-describing property.

Q70. How can we create a component?

Ans. To create a component, we have to follow these 3 steps.

1. Create a component.

We can create a component with the help of @Component decorator function. For creating a component, firstly create any Typescript file and write the following code.

myfirst.component.html | myfirst.component.ts

src > app > myfirst > myfirst.component.ts > ...

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

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

  constructor() { }

  ngOnInit(): void {
  }

}

```

2. Register a component in a module.

After creating your component as mention in step 1, just register your component into app.module.ts file. For Ex-

myfirst.component.ts | app.module.ts

src > app > app.module.ts > ...

```ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { MyfirstComponent } from './myfirst/myfirst.component';

@NgModule({
  declarations: [
    AppComponent,
    MyfirstComponent // register your component here
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

3. Add an element in an HTML markup.

After creating and registering the component just call the component selector where you want to display it. For Ex-

myfirst.component.ts | app.component.html

src > app > app.component.html > ...

```
1  <h2>Sahosoft Online Classes (sahosoft.com)</h2>
2
3  <app-myfirst></app-myfirst>
4
```

Q71. Component is directive or not?

Ans. Yes, Component is Directive in which we have template.

Q72. What is component decorator?

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

Component decorator Example-:

myfirst.component.ts | app.component.html

src > app > myfirst > myfirst.component.ts > ...

```
1  import { Component, OnInit } from '@angular/core';
2
3  @Component({
4    selector: 'app-myfirst',
5    templateUrl: './myfirst.component.html',
6    styleUrls: ['./myfirst.component.css']
7  })
8  export class MyfirstComponent implements OnInit {
9
10   constructor() { }
11
12   ngOnInit(): void {
13   }
14
15 }
16
```

Q73. What are the different metadata of a component?

Ans. The components provides some additional metadata configurations which are shown below in figure–

```
@Component ({
    changeDetection?: ChangeDetectionStrategy
    viewProviders?: Provider[]
    moduleId?: string
    templateUrl?: string
    template?: string
    styleUrls?: string[]
    styles?: string[]
    animations?: any[]
    encapsulation?: ViewEncapsulation
    interpolation?: [string, string]
    entryComponents?: Array<Type<any> | any[]>
    preserveWhitespaces?: boolean
    // inherited from core/Directive
    selector?: string
    inputs?: string[]
    outputs?: string[]
    host?: {...}
    providers?: Provider []
    exportAs?: string
    queries?: {...}

    })
```

The detail list of Component's Metadata Properties are:

- Selector Property: The CSS selector is that which identifies the component in a template.
- StyleUrls Property: It contains list of URLs of style sheets to be applied on the component's view.
- Styles Property: It contains the inline styles for the component's view.
- Template Property: It is used for the inline template for the component's view.
- TemplateUrl Property: It conatins the URLs of external files containing a template for the view.
- Animations Property: Applied the list of animations of the component.
- ChangeDetection Property: The change detection strategy is used by the component.
- Encapsulation Property: The style encapsulation strategy is used by the component.
- EntryComponents Property: It use the list of components that are dynamically inserted into the view of the component.
- ExportAs Property: The name under which component instance is exported in a template.
- Host Property: Used to map the class property to host element bindings for events, properties, and attributes.
- Inputs Property: The list of class property names to data-bind as component inputs.
- Interpolation Property: The custom interpolation markers used in this component's template.
- ModuleId Property: This is the CommonJS module id of the file in which this component is defined.
- Outputs Property: The list of class property names that expose output events that others can subscribe too.
- Providers Property: It consist list of providers available to the component and its children.

- ❖ Queries Property: To configure queries that can be injected into the components.
- ❖ ViewProviders Property: It consist list of providers available to the component and its view children.

Q74. What is the mandatory property of @Component() decorator function?

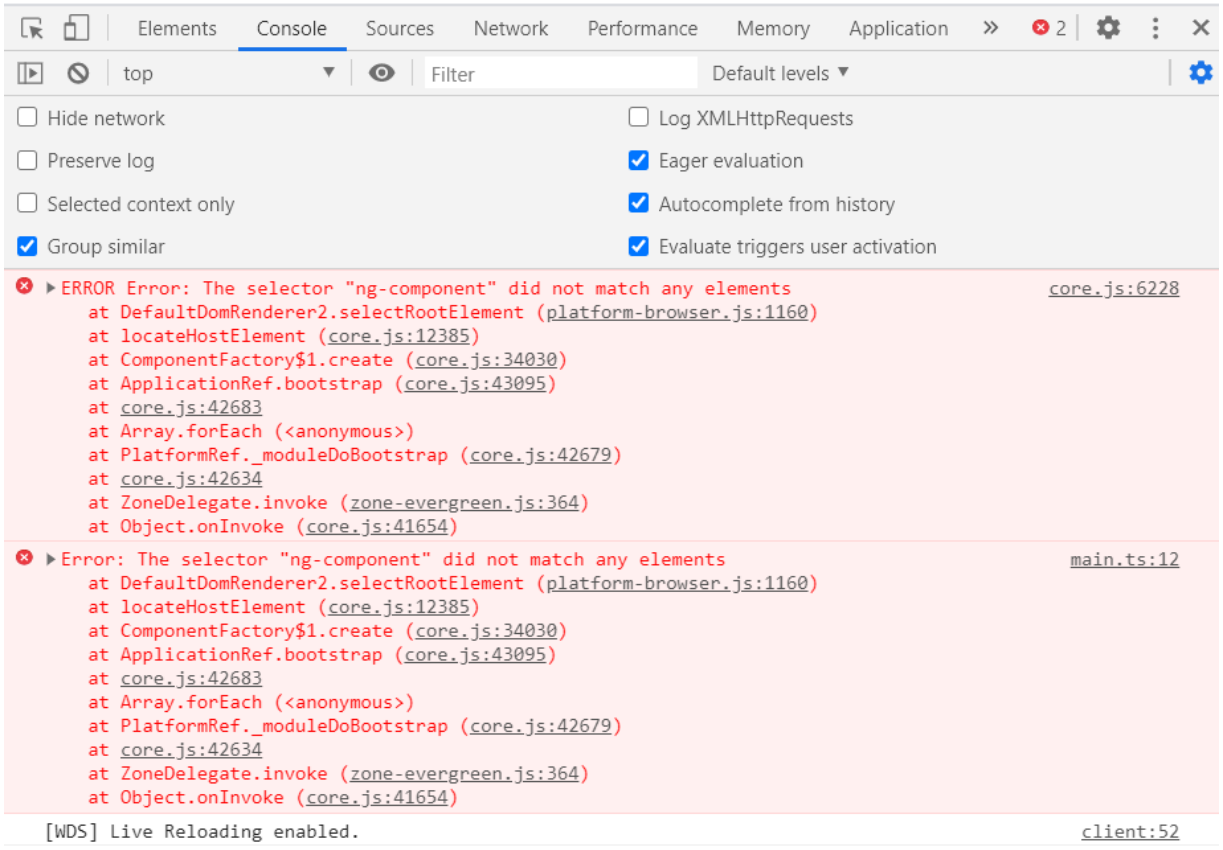
Ans. “selector” and “template or templateUrl” are the mandatory properties for @Component() decorator function. If you lose the selector property , how can you use it, because selector is used to display the template, and, if you lose the template property then compile time error is generates such as:

1. When you lose the selector property

app.component.ts

src > app > app.component.ts > ...

```
1   import { Component } from '@angular/core';
2
3   @Component({
4     // selector: 'app-root',
5     templateUrl: './app.component.html',
6     styleUrls: ['./app.component.css']
7   })
8   export class AppComponent {
9
10  }
11
```

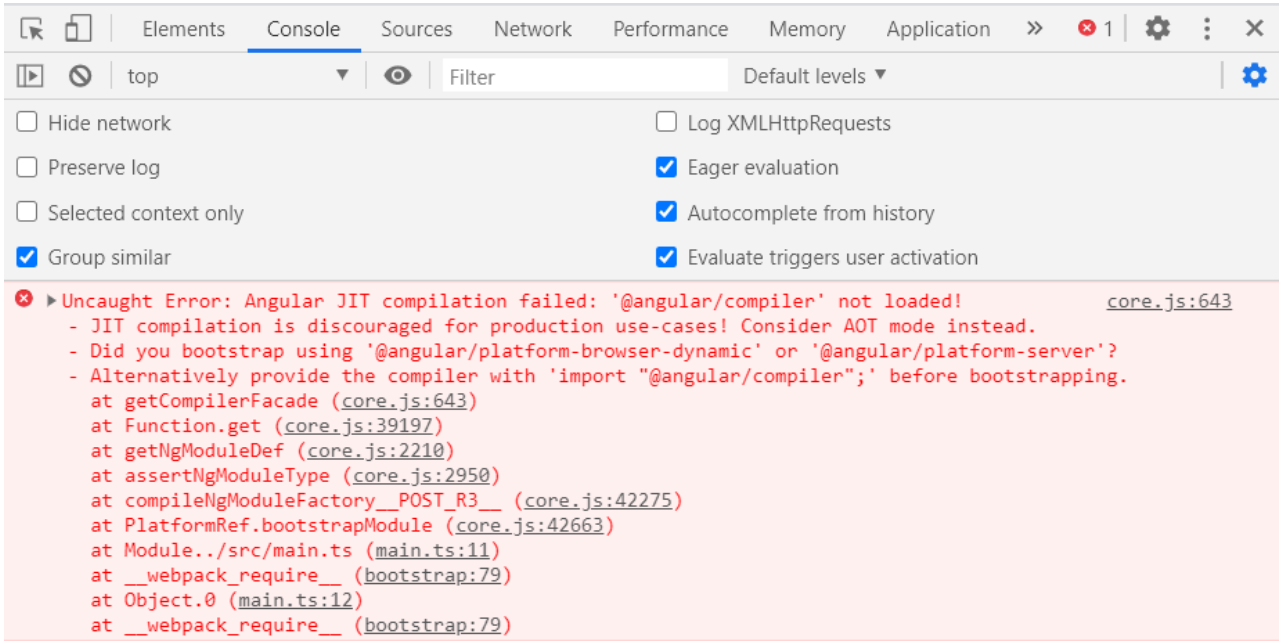


2. When you lose the templateUrl property

app.component.ts

src > app > app.component.ts > ...

```
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-root',
5     //templateUrl: './app.component.html',
6     styleUrls: ['./app.component.css']
7   })
8   export class AppComponent {
9
10
11  }
12
```



Q75. What is ViewEncapsulation?

Ans. ViewEncapsulation decides whether the styles defined in the component can affect the whole application or not. There are 3 ways to do it in Angular:

Emulated: When we set the ViewEncapsulation.Emulated in the @Component decorator, Angular will not create a Shadow DOM for the component and the style will be applied to the component. One thing you need to know that it is mandatory to set the default value for encapsulation.

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

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

Q76. What is the difference between templeteUrl and template?

Ans. The templateUrl and template are the parts of @Component decorator. We use templateUrl when we have to add the path of any template file means html file, and use template when we have to add html code of entire html page needed in the component.

Q77. What is the difference between styleUrl and style?

Ans. The styleUrl and style, are the parts of @Component decorator. We use styleUrl when we have to add the path of any style file, and style is used when we have to add only required style for the page required for the component.

Q78. What is the difference between providers present in component and present

in app.module.ts file?

Ans. Basically, the provider is used for dependency injection (DI). It accepts list of services name. So, when we inject the service name in Provider inside the app.module.ts file then it behaves like a global service that is presentend in the whole application and we can use any component, however, when we inject service name into the Provider within component file then it behave like local service that can be only used in the current component.

Q79. What is Dynamic Component?

Ans. Always component templates are not fixed. An application may need to load new components at runtime. Therefore, when we create component so that they can load runtime, this type of component is called a Dynamic Component.

Q80. What is the use of ComponentFactoryResolver Service?

Ans. Whenever, we create a Dynamic Component, we have to use ComponentFactoryResolver service. This service can be used to display an instance of the component in another component's template. However, this is similar to the $compile method in AngularJS, but this is much better than old $compile method since it encapsulates each template in the child component.

Q81. Give an example of Dynamic Component?

Ans. We have one component as app.component.ts with template of html in which we have two buttons as the first component and the value of Second component. Now, we have to render html of the desired component on the page by clicking on button
That is, when we click on first button, the "First component" html will be displayed and if you click on the "Second component" button, the html of second component will be displayed so we will achieve this requirement using Dynamic component with component factory resolve.

Q82. What is nested component?

Ans. When we create 2 components as myfirst component and mysecond component. Now, if we place second component selector on first component, it is called nested component:

For example:

If we have one component as, mysecond.component.ts:

myfirst.component.ts mysecond.component.ts

src > app > mysecond > mysecond.component.ts > ...

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

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

  styleUrls: ['./mysecond.component.css']
})
export class MysecondComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {
  }

}
```

Another Component as, myfirst.component.ts:

myfirst.component.ts mysecond.component.ts

src > app > myfirst > myfirst.component.ts > ...

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

@Component({
  selector: 'app-myfirst',

  template: `<h2>My first Component </h2>
  <app-mysecond></app-mysecond>`,

  styleUrls: ['./myfirst.component.css']
})
export class MyfirstComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {
  }

}
```

As you can see in the above example, we are using the selector of second component in first component so it is the example of nested component. We can say one component as Parent as in above myfirst component where we have added the selector of mysecond component as a child component.

Q83. What is the role of selector?

Ans. We use selector of any component for identifying each component uniquely in component tree.

Q84. What is entry Component?

Ans. The "entry Component" is a property that contains the list of components that will be generated at run time. Adding the component to entryComponents instructs the offline template compiler to compile the components so that they can be used at run time.

The entry component is used to define components and is created dynamically using the ComponentFactoryResolver. Initially, Angular creates a component factory for every bootstrap components using ComponentFactoryResolver. And then, at runtime, it use the factories to instantiate the components. You can also specify an entry component by bootstrapping in the Angular module or specify an entry component by routing the definition. Entry components are not connected to routes. It is loaded dynamically and are not referenced in component templates (HTML code).

All other root components must be listed in the declarations array.

app-routing.module.ts

src > app > app-routing.module.ts > ...

```
1   import { NgModule } from '@angular/core';
2   import { Routes, RouterModule } from '@angular/router';
3   import { DashboardComponent } from './dashboard/dashboard.component';
4   import { LoginComponent } from './login/login.component';
5
6   const routes: Routes = [
7     { path: '', redirectTo: 'home', pathMatch: 'full' },
8     { path: 'dashboard', component: DashboardComponent },
9     { path: 'login', component: LoginComponent },
10    { path: '**', redirectTo: 'home' }
11  ];
12
13
14  @NgModule({
15    imports: [RouterModule.forRoot(routes)],
16    exports: [RouterModule]
17  })
18  export class AppRoutingModule { }
19
```

There are mainly 2 kinds of entry components:

- The bootstrapped root component
- A component you specify in a route

The bootstrapped entry component

A bootstrapped component is an entry component that Angular loads into the DOM (Data Object Model) when application is launched and the other root components are loaded dynamically in the entry components.
The angular loads a root dynamically because it is bootstrapped in the Angular Module. In the below example, AppComponent is a root component so that angular loads dynamically.

Example of specifying a bootstrapped component is shown below: –

app.module.ts

src > app > app.module.ts > ...

```
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6  import { DashboardComponent } from './dashboard/dashboard.component';
7  import { LoginComponent } from './login/login.component';
8
9  @NgModule({
10   declarations: [
11     AppComponent,
12     DashboardComponent,
13     LoginComponent
14   ],
15   imports: [
16     BrowserModule,
17     AppRoutingModule
18   ],
19   providers: [],
20   bootstrap: [AppComponent] //bootstrapped entry component
21 })
22 export class AppModule { }
```

A Routed entry component

All the components of the router must be entry components because the component would require you to add in 2 places.

- Router and
- EntryComponents

The Angular compiler is so smart and recognizes that it is a router component and automatically adds the router components in entry components.
A routes definition refers to components. For Ex-

- LoginComponent
- DasboardComponent

There are 2 components, that is Login component and another one is Dashboard. If these passed the authentication and authorization then these components have the access to navigate between the login and dashboard views.

Example –

app-routing.module.ts

src > app > app-routing.module.ts > ...

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { DashboardComponent } from './dashboard/dashboard.component';
import { LoginComponent } from './login/login.component';

const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'dashboard', component: DashboardComponent },
  { path: 'login', component: LoginComponent },
  { path: '**', redirectTo: 'home' }
];


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

```

Q85. Why does Angular need entry components?

The entry components improve the performance, and it is smallest, fastest and reusable code of your production application.

For example, if you want to load the code which is smallest, fastest and reusable in the application. These codes contain only those classes which you actually need and it must exclude the components that are never used, regardless of those components are declared in application or not.

As you know, many libraries declare and export components that are never used in your application. If not referenced, the tree shaker removes these libraries and components from the final code package.

The following is an example of specifying a bootstrapped component:

app.module.ts

src > app > app.module.ts > ...

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { DashboardComponent } from './dashboard/dashboard.component';
import { LoginComponent } from './login/login.component';

@NgModule({
  declarations: [
    AppComponent,
    DashboardComponent,
    LoginComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent] //bootstrapped entry component
})
export class AppModule { }
```

If the component is not in an entry component, the compiler skips compiliation of this

component.

Q86. What's the difference between a Bootstrap Component and an Entry Component?

A bootstrapped component is an entry component that Angular loads into the DOM when the application starts and the other root components are loaded dynamically into entry components.
Following is an example of specifying a bootstrapped component –

app.module.ts

src > app > app.module.ts > ...

```
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6  import { DashboardComponent } from './dashboard/dashboard.component';
7  import { LoginComponent } from './login/login.component';
8
9  @NgModule({
10   declarations: [
11     AppComponent,
12     DashboardComponent,
13     LoginComponent
14   ],
15   imports: [
16     BrowserModule,
17     AppRoutingModule
18   ],
19   providers: [],
20   bootstrap: [AppComponent] //bootstrapped entry component
21 })
22 export class AppModule { }
```

The entry component is used to define components and is created dynamically using ComponentFactoryResolver.
The @NgModule.bootstrap property informs the compiler that it is an entry component and it must generate code to start the application with this component.

Q87. When do I add components to entryComponents?

Most large application developers wouldn't need to add components to the entry components, and Angular automatically adds few components to entry components.
Entry components are connected to routes. They are dynamically loaded and are not referenced in component templates.

Q88. What is component hook lifecycle?

Ans. A component goes through different phases after creation until it terminates. Angular is responsible for maintaining and taking care of all these stages. In every phase, you can write your own code or logic, which helps a lot.

Angular offers 8 life cycle hooks that allow you to tap into the lifecycle of directives and components as they are created, updated, and destroyed. Each has a single hook method with the prefixed name ng. Angular call these as hook methods in the following order:

ANGULAR COMPONENT HOOK LIFECYCLE:

These are the phases of component lifecycle hook:

- ngOnChanges( ) – It is called before ngOnInit( ) when the one or more data-bound input

properties changes then this hook is invoked in the child component.

- ngOnInit( ) – It is called once the components is initialized and by default it is added to every component.
- ngDoCheck( ) – It is called during every change detected on the template of the component.
- ngAfterContentInit –It is called only once after component content is initialized.
- ngAfterContentChecked – It is called after every change of component content.
- ngAfterViewInit – It is called after the component and child views has been initialized.
- ngAfterViewChecked – It is called every time the component's and child's views has been checked.
- ngOnDestroy – It is called just before the component or directive is destroyed.

The ngOnInit() and ngOnDestroy() methods play important roles in the real-time applications.

Use of ngOnInit()
There are 2 main reasons to Use ngOnInit method that are:-

- It performs the complex initializations shortly after construction.
- It is used to set up the component after Angular sets the input properties.

Use of ngOnDestroy( )
The ngOnDestroy( ) method is used to clean-up logic and it must run before Angular destroys the directive or components.
If you miss to call this method than memory leaks may occur.

Q89. What is the difference between @ViewChild and @ViewChildren?

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

Q90. What is the difference between @ContentChild and @ContentChildren?

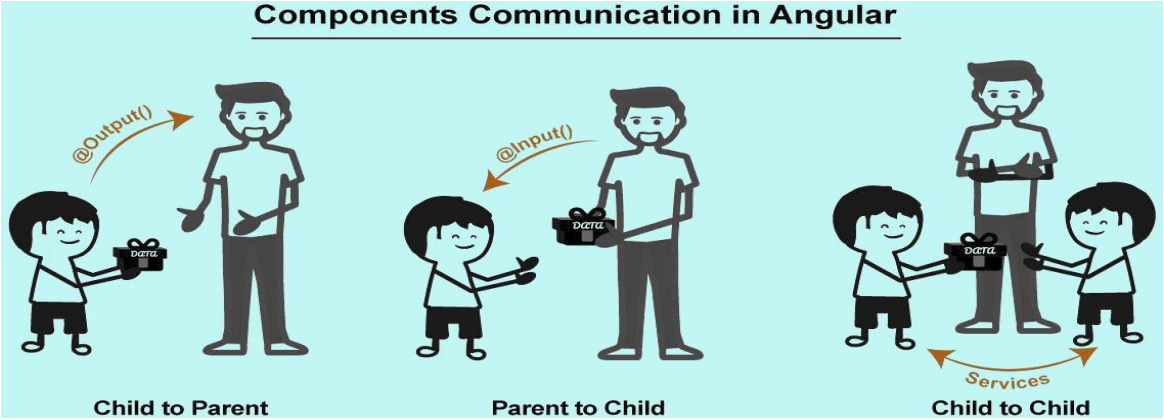
Ans. Both @ContentChild and @ContentChildren are decorators and we use these when we want to get single child element or all the child elements of the content DOM which means inside the <ng-content></ng-content>.

Q91. What is the difference between ngDoCheck and ngOnChange?

Ans. ngDoCheck is similar to the ngOnChanges lifecycle hook, the main difference is that ngOnChanges does not detect all the changes made to the input properties. It detects changes only for those properties that are passed by value not by reference. Whereas, ngDoCheck detects changes also for those properties that are passed by reference, like arrays.

Q92. How we can pass data from one component to another component?

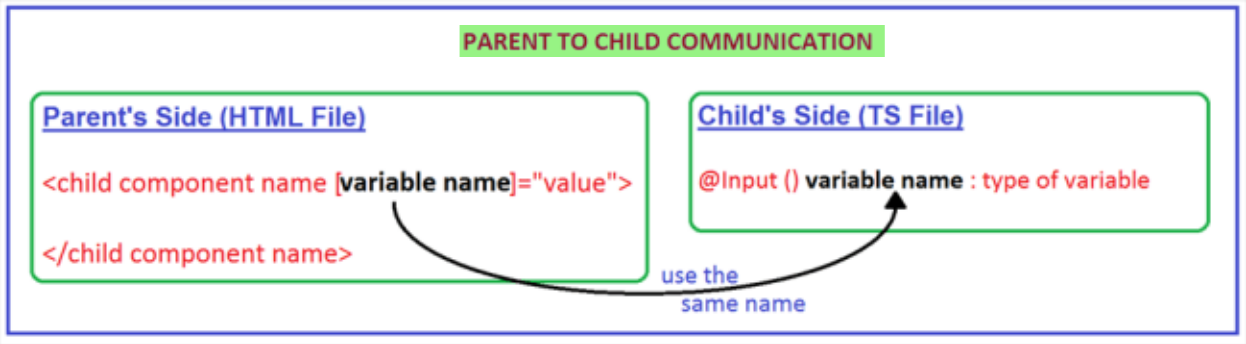
Ans.



We can also pass the data from one component even if parent and child relationships exists or not between them. If the components are separate and we want to pass data than we can use services using get and set properties. Set property is used for setting the values which you want and get for getting the values which is added by a component.

Q93. How can we pass data from parent component to child component using @Input.

Ans.

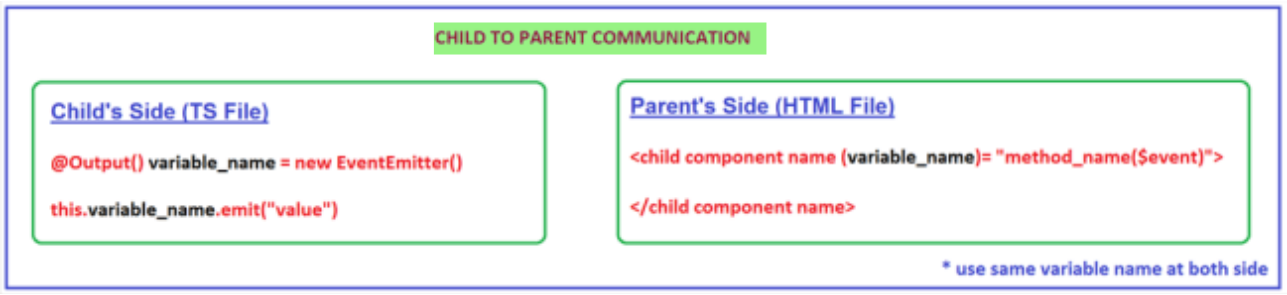


According to the above image, it is very easy to pass data from Parent to Child, you need to remember the
following steps,

- At the parent side, Declare a variable and use it as a Property Binding and assign a value, when parent component is called, like <child component [variable name]=”value”> </child component>.
- In the child component, fetch this variable in component (.ts file) using the @Input decorator function, like @Input () variable name: type of variable
- The use it into child HTML page.

Q94. How can we pass data from child component to parent component using @Output.

Ans.



According to the image above, it is simple to pass data from Child to Parent component, you need to remember the following steps.

On Child component

- Go to the HTML page, use event binding and call a function to set the value.
- Go to the component page which means .ts and declare a property with @Output(), like @Output () variable name=new EventEmitter();

And emit data using this variable like this.variablename.emit(passing-data)

On Parent component

Go to the HTML page, use the same variable name as defined in the child component with @Output( ), as an event binding <child component selector (variable name)= ”method name ($event)”>
Go to the component page and inside the method set the value and use it in the parent HTML.

Q95. What is Event Emitter?

Ans. As components is used in Angular and there are many events that we are using so, component emits the event using @output and event emitter. When we pass the value from child to parent component, parent can emit the value anytime using event emitter. Basically, directives are used to extend the power of HTML attributes and to model and reshape the DOM structures.

For example:

app.component.ts

src > app > app.component.ts > AppComponent > constructor

```
1  import { Component, OnInit, Output, EventEmitter } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent implements OnInit {
9
10   @Output() change: EventEmitter<number> = new EventEmitter<number>();
11
12   constructor() {
13
14   }
15
16   ngOnInit() {
17
18   }
19
20 }
```

Q96. What is angular directive?

Ans. Basically, directives are used to extend the power of the HTML attributes and to change the appearance or behavior of the DOM structure.

It is a TypeScript class which is declared as a @directive decorator. The directives allow you to attach behavior to DOM elements and the @directive decorator provides additional metadata that determine how directives are to be processed, instantiated, and used at run-time.

Q97. How many types of directives are supported by Angular?

Ans. Angular supports three types of directives

The 3 types of Directives in Angular are:

- Component Directive
- Attribute Directive
- Structural Directive

Components - The component is a directive with its own templates and is responsible for how a component must be processed, instantiated and used at runtime.
Structural Directives - The structural directive is a directive which is responsible for the modification in DOM layout by adding, removing, and manipulating elements. The most of the common built-in structural directives are NgIf, NgFor, and NgSwitch.

Attribute Directives - The Attribute directive is a directive which is responsible for change in the behavior of a particular element or component.

Q98. What is @Directive decorator?

In Angular we use @Directive when we create the custom directive based on the application's requirements. In which we will add the selector property for the directive.

```
@Directive({
    selector?: string
    inputs?: string[]
    outputs?: string[]
    host?: {...}
    providers?: Provider[]
    exportAs?: string
    queries?: {...}
})
```

Selector – It is selector of CSS that tells Angular to instantiate this component wherever it finds the corresponding tag in the template HTML.

```
For example, it is - <app-login></app-login>
```

The CSS selector also triggers the instantiation of a directive.
The selector can be declared by element name, class name, attribute name, and attribute value.
Suppose we have a directive with an <input type="checkbox"> selector and the HTML looks like as shown below.

```
For example, it is - <app-login></app-login>
```

The directive will be instantiated only in the <input type="checkbox"> element.

- Inputs– The list of class property names to data-bind as component inputs.
- Outputs - The list of class property names that expose output events which others can subscribe.
- Host– This property is use to map the class property to host element bindings for properties, events, actions, and attributes.

The host looks like as shown in below code.

```
@Directive({
  selector: 'button',
  host:{'(click)':'onClick=($event.target)'}
})
export class MyDirective {

  constructor() { }
}
```

- Providers – It contains list of providers that are available to this component and its children.
- Queries– It is used to configure queries that can be injected into the component.

Q99. Can we create constructor in our directive? If yes then how.

Ans. Yes, we can create constructor in our directive, as shown in code below.

my.directive.ts

src > app > my.directive.ts > ...

```
@Directive({
  selector: '[appMy]'
})
export class MyDirective {

  constructor(private elref: ElementRef) { }

  @HostListener('click') onClick() {
    this.elref.nativeElement.style.color = "red";
  }

}
```

Here, when the directive is created, Angular can inject an instance of called ElementRef into this constructor.

Q100. What is Component directive? Give an example

Ans. Component directive is a type of directive that always come has a template (view) and is present in an Angular application. To create a component directive, we use the @Component ( ) decorator function.

For Ex –

app.component.ts

src > app > app.component.ts > ...

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

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

  constructor() { }

  ngOnInit() {

  }
}
```

Now, to use this directive, you should code as shown above

```
<app-root></app-root>
```

This is what we said, directive with template.

Q101. What is structural directive?

Ans. The Structural directive modifies the DOM structure by adding or removing DOM elements. Basically, you can say that it works on DOM. These directives always use * as a prefix.

Q102. Name some structural directives provided by Angular.

Ans. Some structural directives are:-

Q103. What are the differences between @Component and @Directive?

Components are used to create new elements in the DOM with their own HTML template.
The attribute directives are used, when you want to modify or update the existing elements in the DOM.

Q104. What is the difference between ngIf and hidden?

Ans. Both are different due to their rendering methods, for example *ngIf can render and add element into DOM if the condition is true, but the hidden property had already rendered and added elements into DOM, it only displays and hides these elements according to the condition.

Q105. How can we use "then" and "else" keywords with *ngIf directive? Explain with an example.

Ans. Using “ng-template” and “template reference variables” we can use the keywords “else” and “then” with ngIf, because if condition is true, then it will execute the “then” part, otherwise it will execute “else” part. For ex-

App.component.html

app.component.html

src > app > app.component.html > ng-template

```
<button type="button" (click)="changeValue()">click me</button>

<div *ngIf="isValid; then thenBlock else elseBlock">

</div>
<ng-template #thenBlock>
  <div>Then Block: Show</div>
</ng-template>
<ng-template #elseBlock>
  <div>Else Block: Show</div>
```

App.component.ts

app.component.ts

src > app > app.component.ts > ...

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  isValid: boolean = true;
  constructor() { }

  ngOnInit() {

  }
  changeValue(valid: boolean) {
    this.isValid = valid;
  }
}
```

Q106. What are the keywords that we use at the time of ngSwitchCase and what is the role of that keywords?

Ans. Mainly we use three keywords to implement ngSwitch

- ngSwitch : It use binding property and takes an expression that returns the value of switch.
- ngSwitchCase : It contains the value matched element.
- ngSwitchDefault : If there is no match in ngSwitchCase then it executes.

Q107. What is *ngFor and what are the exported values of ngFor directive?

Ans. The *ngFor directive is used to render objects list. For example (create and render a name list). The exported values of ngFor directives are –

- index - It gives the index number of the current item.
- first - It returns boolean value, i.e. returns true if the item in the list is the first otherwise false.
- last - It returns boolean value, i.e. returns true if the item in the list is the last otherwise false.
- even - It returns boolean value, i.e. returns true if the item has an even index in the list otherwise false.
- odd - it returns boolean value, i.e. returns true if the item has an odd index in the list otherwise false.
- $implicit :T :- The value of an individual item in the iterable (used with ngForOf)

Q108. What is the difference between ngFor and ngForOf?

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

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

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

During compilation of *ngFor it is converted like this,

```
<ng-template [ngFor]="let record of item">
<div></div>
</ng-template>
```

After that “let”, which is present in “let record of items”, it again compiles and converted into

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

Q109. How to Create Custom Directives?

Ans. Let's start to create a simple custom directive.
Firstly, installed the Angular CLI and all the necessary configurations are running in your app.
Now, go to your project directory and execute the below CLI command to create custom directive –

```
ng g directive myCustom
```

After executing the above CLI command, create two files in the project - src/app folder

- src/app/my-custom.directive.spec.ts
- src/app/my-custom.directive.ts

And update files reference automatically in your project module – "src/app/app.-module.ts"
Let's see in the code below, how it works-my-custom.directive.ts –

my-custom.directive.ts

src > app > my-custom.directive.ts > ...

```
1   import { Directive } from '@angular/core';
2
3   @Directive({
4     selector: '[appMyCustom]'
5   })
6   export class MyCustomDirective {
7
8     constructor() { }
9
10  }
11
```

app.module.ts –

app.module.ts

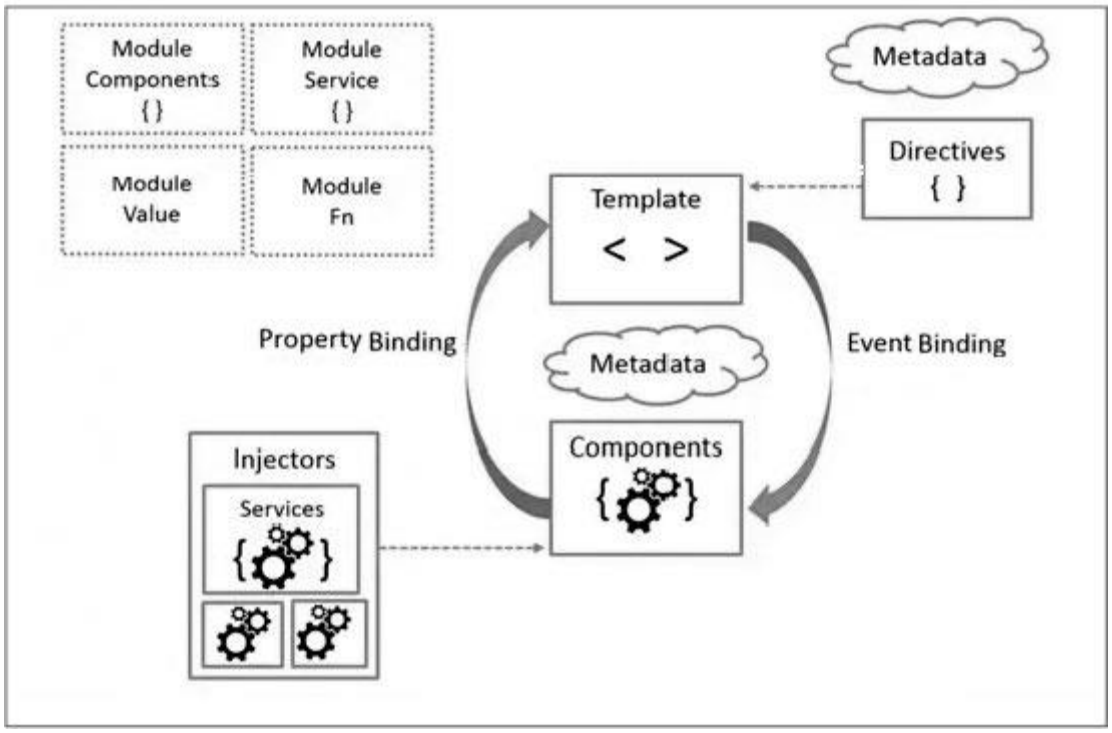
src > app > app.module.ts > ...

```
1   import { BrowserModule } from '@angular/platform-browser';
2   import { NgModule } from '@angular/core';
3
4   import { AppRoutingModule } from './app-routing.module';
5   import { AppComponent } from './app.component';
6   import { MyCustomDirective } from './my-custom.directive';
7
8   @NgModule({
9     declarations: [
10      AppComponent,
11      MyCustomDirective
12    ],
13    imports: [
14      BrowserModule,
15      AppRoutingModule
16    ],
17    providers: [],
18    bootstrap: [AppComponent] //bootstrapped entry component
19  })
20  export class AppModule { }
21
```

Q110. What Is Modules (NgModules)?

Ans. NgModule is a TypeScript class marked by the @NgModule decorator.
The NgModule class works with the @NgModule decorator function and also accepts a metadata object that tells Angular how to compile and excute the module code.
The Angular module allows to organize an application in the associative blocks of functionality.
It also represents a core concept and plays a fundamental role in structuring Angular applications.

The NgModule is used to simplify the ways of defining and managing the dependencies in the applications and can also consolidate different components and services into associative blocks of functionality.



Each Angular application must have atleast one module and it contains the components, service providers, pipes and other code files whose scope is defined by the NgModule that contains it.
The purpose of the module is to declare everything that is created in Angular and group them together.
All angular application has atleast one module, the root module that you bootstrap to launch the application. The Angular root module is called AppModule.

app.module.ts

src > app > app.module.ts > ...

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { LoginComponent } from './login/login.component';

@NgModule({
  declarations: [
    AppComponent,
    LoginComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent] //bootstrapped entry component
})
export class AppModule { }

```

Q111. What are the @NgModule Metadata Properties?

The @NgModule decorator identifies AppModule as NgModule class.
The @NgModule takes a metadata object that tells Angular how to compile and start the application.
The important metadata properties of NgModule are the following:-

- providers
- declarations
- imports
- exports
- entryComponents
- bootstrap
- schemas
- id

The @NgModule class with the decorator and metadata properties -

```
@NgModule({
 providers?: Provider[ ]
 declarations?: Array<Type<any> | any[]>
 imports?: Array<Type<any> | ModuleWithProviders | any[]>
 exports?: Array<Type<any> | any[]>
 entryComponents?: Array<Type<any> | any[]>
 bootstrap?: Array<Type<any> | any[]>
```

```
  schemas?: Array<SchemaMetadata | any[]>
  id?: string
})
```

Let understand in detail about NgModule metadata is as follows-

- Providers: It contains the list of dependency injection (DI) and it also defines the set of injectable objects that are available in the injector of this module.
- Declarations: It contains list of declarable classes, components, directives, and pipes that belong to this module. The compiler throws an error when you try to declare the same class in multiple modules.
- Imports: It contains the list of modules and it is used to import the supporting modules like FormsModule, RouterModule, CommonModule, or any other custom functionality module.
- Exports: It contains the list of declarable components, directives, pipes, and modules that an import module can use within a template of any component.
- EntryComponents: It contains list of components that must be compiled when this module is defined. By default, an Angular application always has atleast one entry component, the root component i.e. AppComponent.
- A bootstrapped component is an entry component that Angular loads into the DOM during application launch and other root components dynamically loaded into the entry components.
- Bootstrap: A list of components that automatically start and the listed components will be added automatically to entryComponents.
- Schemas: Defines a schema that will allow non-angular elements and properties.
- Id: The Id is used to identify the modules in getModuleFactory. If not defined, the NgModule is not registered with getModuleFactory.

Q112. Why use multiple NgModules?

Ans. Multiple NgModules provides some important benefits.
In fact, the modules help you to organize an application into associative blocks of functionality.
Firstly, we have to organize the application code. If you put many resource files in the default app module and see the happing.
And the second is - It opens the possibility of lazy loading through the router.

Q113. What Are the Purpose of @NgModule?

NgModule is used to simplify the ways of defining and managing the dependencies in the applications

- and you can also consolidate different components and services in cohesive blocks of functionality.
- The @NgModule metadata is divided into 3 categories as follows:

- Static
- Runtime
- Composability/Grouping

Static: It is a compiler configuration and configured through the declarations array.
Runtime: It is injector configuration and configured through the provider's array.
Composability/Grouping: Introducing NgModules together and configured via the imports and exports arrays.

Following is an example of specifying a NgModule metadata –

```
@NgModule({
  // Static, This is the compiler configuration
  declarations: [], //declarations is used for configure the
selectors.
  entryComponents: [], //entryComponents is used to generate
the host factory.

  //Runtime or injector configuration
  providers: [], // providers is used for runtime injector
configuration.

  //Composability and Grouping
  imports: [], // imports used for composing NgModules together.
  exports: [] //A list of declarations components, directives,
and pipes classes that an importing module can use.
})
```

Q114. Types of NgModules?

There are 5 types of NgModules:

- Feature Module
- Routing Module
- Service Module
- Widget Module
- Shared Module

Feature Module: The feature module are NgModules to organize an application code.
Routing Module: Routing is used to manage routes and also allows navigation from one view to another view while users perform application tasks.
Service Module: Modules that contain only services and providers. It provides utility services such as data access and messaging. The root AppModule is the only module that needs to import service modules. The HttpClientModule is a good example of a service module.
Widget Module: It is a third party UI component libraries .
Shared Module: It allows us to organize the application code. You can put frequently used components, directives, and pipes in a module and use whenever required.

Q115. What are the different types of Feature Modules?

There are 5 types of feature modules:

- Domain Feature Modules
- Routed Feature Modules
- Routing Modules
- Service Feature Modules
- Widget Feature Modules

Routed Feature Module: Routed feature modules are domain feature modules whose components that are present on the top are the targets of router navigation routes. A lazy-loaded should not be imported by any of the module and Routed feature modules also do not export anything because their components never displayed in the template of an external component.
Routing Module: A routing module provides the routing configuration for another module and focuses on:

1. Defines Routes
2. Adds Router Configuration to the module imports.
3. Add service providers to the module providers.
4. A routing module does not have its own declarations. The components, directives, and pipes are the responsibility of the feature module and not the routing module. A routing module only needs to be imported via its companion module.

Service Feature Module: Service module provides utility services and are used to communicate with the server. The HttpClientModule is an excellent example of a service module. The root AppModule is the only module that should import service modules.
Domain Feature Module: Domain feature modules offer a dedicated user experience to a special application domain, such as editing a client,etc.
Widget Feature Module: A widget module makes components, directives, and pipes available for external modules. Libraries and third-party user interface components are widget modules.Widget modules needs to be imported in any module whose component templates required the widgets.

Q116. Why you use BrowserModule, CommonModule, FormsModule, RouterModule, and HttpClientModule?

- BrowserModule: The browser module is imported from @angular/platform-browser and is used when you want to run the application in a browser.
- CommonModule: The common module is imported from @angular/common and is used when you want to use directives like NgIf, NgFor,etc.
- FormsModule: The forms module is imported from @angular/forms and is used while building template driven forms.
- RouterModule: The router module is imported from @angular/router and is used to route RouterLink, forRoot, and forChild.
- HttpClientModule: The HttpClientModule is imported from @angular/common/http and is used to initiate HTTP requests and responses in angular application. The HttpClient is latest and easier to use than the http.

Q117. What is the difference in NgModules and JavaScript Modules?

The NgModule is a TypeScript class decorated with @NgModule Decorator.It is a fundamental

feature of Angular.
JavaScript also has its own module system for managing collections of JavaScript objects and is completely different from NgModule system.
In JavaScript, each file is a module and all the objects are defined in the file of that module.
By marking objects with the export keyword the module declare as public.
Other JavaScript modules use import declarations to access public objects from other modules.

Following is an example of specifying an export and import declarations:–

app.component.ts

```
src > app > app.component.ts > ...
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-root',
5     templateUrl: './app.component.html',
6     styleUrls: ['./app.component.css']
7   })
8   export class AppComponent  {
9    //...
10  }
11
```

After export class, you can import that file code in other file.

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

Both the JavaScript and Angular use modules for organizing applications code.

Q118. What classes should you not add to Module Declarations?

Modules, Services, objects, and non-angular helper classes don't need to be declared in the module declarations.
The Syntax for the array of NgModule declaration:–

```
declarations: [
  AppComponent,
  LoginComponent,
  MyPipe,
  MyDirective
]
```

The non-Angular classes and objects as following as -

- Strings

- Numbers
- Functions
- Entity Models
- Configurations
- and other helper classes

Note - Directives, components, and pipes classes can be used in a module declaration.

Q119. Should you import BrowserModule or CommonModule?

BrowserModule – Most browser applications need to import BrowserModule from @angular/platform-browser and the BrowserModule provides necessary services for launching and running browser applications.
Do not import BrowserModule in another module.
BrowserModule also re-exports CommonModule from @angular/common and CommonModule is used when we have to use directives like NgIf, NgFor and many more.
BrowserModule exports a pair of NgModules as -

```
exports: [
  CommonModule,
  ApplicationModule
]
```

And

```
//AppModule class with the @NgModule decorator
@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    MyPipe,
    MyDirective
  ],
  imports: [//DOM rendering, sanitization, and location
    BrowserModule
  ],
  providers: [//service providers
    MySericeService,
    MyModuleModule
  ],
 bootstrap: [AppComponent],  // bootstrapped entry component
  exports: [
    CommonModule,
    ApplicationModule
  ]
})
export class AppModule {
  //exporting app module
}
```

Q120. What happens if you Import the same module twice?

We can import the same module twice, but Angular doesn't like modules with circular references and raise the circular dependency warnings on builds.
In fact, the module helps in organizing the application in associative blocks of functionality.
For example, Class A and Class B can be in the same file if required.

Q121. Why is it bad if a shared module provides a service to a lazy-loaded module?

The lazy loading cause the application to create a new instance every time, instead of using the singleton.
Lazy loading is the best practice for loading required resources on-demand.
This can significantly reduce the initial startup time for single page web applications (SPA). Instead of downloading all the application code and resources before the application starts, you can get it just-in-time (JIT), as needed.
In case of eager loading application create a singleton, instead of creating a new instance every time.

Q122. What are the Validator functions?

There are 2 types of validator functions-

- Async validators
- Sync validators

Async validator functions take a control instance and return an observable which then emits a series of validation errors or null whereas sync validator functions accept a control instance and return a series of validation errors or null.

Angular only runs Async validators due to some performance issues.

Q123. Why "*" is prefix with structural directive? Can we use structural directive without using *?

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

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

Angular convert this code into below code during compilation

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

Yes, we can use structural directive without using *, but if so, the line of code will increase as

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

Q124. How many structural directives can we implement on a single element?

Ans. We can implement only one structural directive on a single element but if you want to use more than one structural directive then we have to use <ng-container> as in the next line, there are 2 structural directives on one element i.e. *ngFor and *ngIf, which is not possible.

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

So, we should use ng-container as shown below: -

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

Q125. How can list items be tracked by default?

Ans. Angular has no information about the object, so it cannot be specified which property it shall use for tracking, so by default, Angular tracks objects based on their identity.

Q126. Can we provide our own mechanism for tracking the elements? If yes, then how?

Ans. Yes, we can provide our own mechanism for keeping track of items in a list using trackBy. We need to pass a function to trackBy, and this function takes 2 arguments, an "index" and the other is "current item". trackBy is used for increasing the performance.
For Ex-

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

In component, function is defined which is used in trackBy.

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

Q127. What are attribute directives? Give an example.

Ans. Attribute directives are used as element attributes, mainly it change the appearance or behavior of an element, component or directive. By default, Angular provides 2 attribute directives. They are: -

NgClass :
Using NgClass we can add or remove CSS class dynamically, that is, based on certain conditions.
NgStyle:
Using NgStyle we can add or remove styles dynamically, that is, based on certain conditions.
Example of ngClass and ngStyle is shown below:

```
<div [ngStyle]="{'width':isEven? '400px':'300px'}">
  <p [ngClass]="{'bg-dark text-white p-2':isEven,'bg-primary text-white p-2':!isEven}">Welcome to sahosoft</p>
  <div>
    <p>Hello! How are you ?</p>
  </div>
</div>
```

Now, if the value of the "isEven" property (defined in the component) is true, it will be implemented
"bg-dark text-white p-2" class, otherwise it will implement "bg-primary text-white p-2".

Similarly, if value of the "isEven" property (define in the component) is true, it will take width as 400px, otherwise, it will take 300px.

Q128. What is the difference between attribute directive and component directive?

Attribute Directives

Attribute Directives add behavior to an existing DOM element or to an existing component instance.

They help us create reusable component.
The @Directive decorator function is used to create an attribute or structural directive.

Component Directives

Instead of adding or changing behavior, a component actually creates its own view (hierarchy of DOM elements) with attached behavior.

It helps us to divide an Angular application into smaller components. The @Component decorator function is used to create component directives.

Q129. What is ng-template?

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

Q130. What is Host Listner?

Ans. @HostListener is a decorator in Angular that is used during the creation of custom directives so you can say that we can use it for events on the host element or component.

Q131. What is ElementRef?

Ans. ElementRef is a class that contains the reference to a DOM element. In Angular we use it when we have to use custom directive. We will follow these steps when we have to use ElementRef:

Import the ElementRef from @Angular/core:

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

Inject it within constructor of the component:

```
constructor(private hostElememt: ElementRef) {

}
```

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

Q132. What is the purpose of @HostBinding?

Ans. @HostBinding is also a decorator that is used at the time of custom directive creation. In Angular, we use it to set the properties on the element or component that hosts the directive.

Q133. What are pipes and how we can use pipe in Angular?

Ans. In each application, we get the data from database and display it on our View, but more often, we have to format or transform the data before displaying it. For this, we use PIPES. For example, database provides the date in the format "Fri Apr 15 1988 00:00:00 GMT-0700" but we want to show it in the View as April 15, 1988, for this we use pipes.

OR
Pipes transform the displayed values in a template.
Use the @Pipe annotation to declare that a particular class is a pipe. A pipe class must also implement a PipeTransform interface.
The @Pipe decorator allows you to define the name of the pipe that is globally available for use in any template across Angular application.
The Pipe class implements the transformation method of "PipeTransform" interfaces which accepts an input value and returns the transformed result.
There will be one additional argument on the transform method for each parameter passed to the pipe.

The CLI command to generate Pipe:

```
ng g p pipename
//or
ng generate pipe pipename
```

Pipe decorator and metadata are:

```
@pipe({
  name:strig,
  pure:boolean
})
```

The pipe name is used for template binding.
To use the pipe, you must set a reference to this pipe class in the module.

Q134. Why we use Pipes?

Sometimes, the data is not displayed in the well format in the HTML templates. We can run a function in the HTML template to get its returned value.
For example - If you want to see a credit card number on your web apps you can not display the whole number on your web app , you should write a custom logic to display card number as like *****-21475 using the custom pipe.

Q135. Name some built-in pipe provided by the Angular.

Ans. In Angular we have various built-in pipes:-

- Currency
- Date
- Decimal
- LowerCase
- UpperCase
- Percent 8-Slice
- Async

Q136. Can I create custom pipe in Angular?

Ans. Yes, in Angular we have some built-in pipes, but you can also build your own pipes. A pipe accepts the values and then returns a value after transformation based on the filter criteria that will be added in the TypeScript (.ts) file.

Q137. What is @Pipe Decorator?

Ans. The @Pipe decorator allows you to define the name of the pipe that will be used within the template expressions. It must be a valid JavaScript identifier.

Q138. How many types of pipes are support by Angular?

Ans. There are two types of pipes which angular support:-

- Pure Pipes: Angular only executes pure pipe when it detects a pure change in the input value. A pure change is either a change to a primitive input value like String, Number, Boolean, Symbol or a changed object reference like Date, Array, Function, Object.
- Impure Pipes: Angular executes an impure pipe when change is detected in every component. An impure pipe is often called, whenever a key or mouse movement occurs.

Q139. How can you define or set your pipe as pure or impure?

Ans. You make a pipe impure by setting its pure property false.

```
@Pipe({
  name: 'order',
  pure:false
})
```

Q140. Can I create any pipe in Angular if yes then How?

Ans. Yes, we can create pipe as shown below in code:

order.pipe.ts

src > app > order.pipe.ts > ...

```
1  import { Pipe, PipeTransform } from '@angular/core';
2
3  @Pipe({
4    name: 'order',
5    pure: false
6  })
7
8  export class OrderPipe implements PipeTransform {
9
10   transform(value: string, ...args: unknown[]): string {
11     return "Hello" + value;
12   }
13 }
14
```

Q141. What Is PipeTransform interface?

The Pipe class implements the PipeTransform interface that takes input value (It is optional parameters) and returns the transformed value.
The transform method is an important method of a pipe.
To create a Pipe, you need to implement this interface.
Angular calls the transformation method with the value of a binding as the first, and second argument in list form.
The PipeTransform interface will look like as shown below:

```
export declare interface PipeTransform {
    transform(value: any, ...args: any[]): any;
}
```

And it is imported from '@Angular/core':

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

Two Categories of Pipes in Angular are:

- pure
- impure

By default, every pipe is pure. If you want to make a pipe impure then you have to Set the pure flag to false.

Q142. What Is Impure Pipe?

Angular runs an impure pipe during after every change detection cycle in the component. It is called often, as every keystroke or mouse-move occurs.
When you have to make pipe impure that time you will allow the setting pure flag to false.

```
@Pipe({
  name: 'currency',
  pure: false
})
```

The example of impure pipe is shown below:

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

@Pipe({
  name: 'currency',
  pure: false
})

export class CurrencyPipe implements PipeTransform {

  transform(value: any, ...args: any): any {
    if (!value) {
      return '1.00';
    }
    return value;
  }
}
```

Q143. What Is Pure Pipe?

Angular runs a pure pipe only when it detects a pure change in the input value. A pure change can be primitive or non-primitive.
Primitive data are single values, they don't have special capabilities whereas the non-primitive data types are used to store the group of values.

```
@Pipe({
  name: 'currency'
})
```

OR

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

Another example for a pure pipe is following:

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

@Pipe({
  name: 'currency',
  pure: false
})

export class CurrencyPipe implements PipeTransform {

  transform(value: any, ...args: any): any {
    if (!value) {
      return '1.00';
    }
    return value;
  }
}
```

The Pipe operator is denoted by symbol (|)

The pipe operator is used to specify the value transformation in HTML template or view.

Q144. What Is Parameterizing Pipe?

A pipe can accept any number of optional parameters to get its output. The parameter value can be any valid template expressions. If you want to add optional parameters, follow the pipe name with a colon (:). Its looks like- currency: 'INR' .For example:-

```
<h2>
    My Birthday is - {{birthday | date:'MM/dd/yy'}}
</h2>

<!-- Output : My Birthday is - 08/10/91 -->
```

Q145. What Is Chaining Pipe?

The chaining Pipe is used to perform the many operations in the single expression. This chaining operation will be chained using the pipe (I).
In the example below, to show the date of birth in the uppercase we need to use the inbuilt date pipe and uppercase pipe.
For example: -

```
<h2>
    Today is - {{date | date:uppercase}}
</h2>

<!-- Output : Today is - Monday, July 27, 2020 -->
```

Q146. What Is DatePipe?

The Date Pipe is used to format a date using the locale rules.

```
{{value_expression | date [: format : [: timezone [: locale]] ]}}
```

The Example for date pipe:
The full date gives the full date for the given date. The short date converts the date to the short date format and the long date gives you long date for the given date.

```
<h3>{{todaydate}}</h3>
<h3>{{todaydate | date:'shortDate'}}</h3>
<h3>{{todaydate | date:'longDate'}}</h3>
<h3>{{todaydate | date:'fullDate'}}</h3>
```

Q147. What Is CurrencyPipe?

The Currency Pipe is used to format the currency using locale rules.

```
{{value_expression | currency [: currencyCode : [: display [: digitsInfo [:locale]]] ]}}
```

Currency Pipe formats the number as the currency of a particular country. It takes country currency type as a parameter.
The example for the currency pipe is shown below:

```
<tr>
    <td>{{employee.salary | currency}}</td>
    <td>{{employee.salary | currency : 'INR'}}</td>
    <td>{{employee.salary | currency : 'INR' : true : '6.2'}}</td>
</tr>
```

Q148. What Is Percent Pipe?

Angular provides a Percent Pipe and it is used to format the number as percentage according to following rules.
The expression rule with percent pipe:

```
{{ value_expression | percent [ : digitsInfo [ : locale ] ] }}
```

The input value will be formatted as a percentage and it can be of any type.
The digitsInfo is an optional string parameter and is undefined by default.
The locale is optional string parameter and by default it is undefined.
For Ex-

```
<h2>Result- {{marks | pexcent}}</h2>
<!-- output result is - '98%' -->
```

Q149. What Is Lowercase Pipe?

Angular provides a Lowercase Pipe and it is used to transform the given text to lowercase.
The expression with lowercase –

```
{{ value_expression | lowercase }}
```

Consider the following example:

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

@Component({
  selector: 'lowercase-pipe',
  template: `<div>
    <input type="text" (keyup)="changeLowerCase(name.value)">
    <p>LowerCase - <h2>{(value | lowercase}}</h2>
</div>` })

export class LowerCasePipeComponent {
  value: string;

  changeLowerCase(value: string) {
    this.value = value;
  }
}
```

Q150. What Is Uppercase Pipe?

Angular provides an Uppercase Pipe and it is used to transforms the given text to uppercase.
The expression with uppercase is shown below: -

```
{{ value_expression | uppercase }}
```

Consider the following example:

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

@Component({
  selector: 'uppercase-pipe',
  template: `<div>
    <input type="text" (keyup)="changeUpperCase(name.value)">
    <p>UpperCase - <h2>{(value I uppercase}}</h2>
</div>` })

export class UpperCasePipeComponent {
  value: string;

  changeUpperCase(value: string) {
    this.value = value;
  }
}
```

Q151. What Is Titlecase Pipe?

The Titlecase Pipe is used to convert the text i.e. string type data, where the first alphabet of each word is in uppercase and the rest will be in the lowercase.
The expression with titlecase is shown below:

```
{{ value_expression | titlecase }}
```

For Ex-

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

@Component({
  selector: 'titlecase-pipe',
  template: `<div>
    <input type="text" (keyup)="changeTitleCase(name.value)">
    <p>TitleCase - <h2>{(value | titlecase}}</h2>
</div>` })

export class TitleCasePipeComponent {
  value: string;

  changeTitleCase(value: string) {
    this.value = value;
  }
}
```

Q152. What Is Async Pipe?

Angular provide a special type of pipe called Async Pipe, and the Async Pipe subscribes to an observable and returns the last value it has passed.
The AsyncPipe allows you to bind the HTML templates directly to values which arrives in an asynchronous manner that has a great ability for the promises and observables.
The expression with Async pipe is shown below:

```
{{ obj_expression | async }}
```

OR

```
<ul>
    <li *ngFor="let account of account | async">{{account.ACNo }}</li>
</ul>
```

The expression of the object can be observable, promise, null, or undefined.

For Ex-

```
import { Component, OnInit } from '@angular/core';
import { AccountService } from './account-service.service';

@Component({
  selector: 'app-async-pipe',
  template: `<u1><li *ngFor="let account of accounts I async">
            A/C No- f{account.ACNo }} </li></ul>`,
  styleUrls: ['./async-pipe.component.css']
})
export class AsyncPipeComponent implements OnInit {
  //accounts declarations
  accounts = [];
  //fetching json data from Rest API
  apiURL: string = 'http://www.sahosoftweb.com/api/ProductMaster/GetProductList';
  //AsyncPipe Component constructor
  constructor(private accountService: AccountService) { }
  //Load the account list
  ngOnInit() {
    this.accountService.getAccount(this.apiURL).subscribe(data => this.accounts = data);
  }
}
```

Q153. What Are Service Workers?

A Service Worker is a script that runs in the web browsers and handles the caching for web applications. This script runs in the separates background and requires no user interactions.
They can query a local cache and provide a cached response if it is available in the cache. This makes more it reliable and increases performance.
A Service Worker is a programmable network proxy and intercepts all outgoing HTTP requests and controls how network requests for your page are handled.
The Service Worker is a method that allows applications to take advantage of persistent data in the background processing, including hooks to allow bootstrapping of web applications while offline.

Q154. What Are Service Workers in Angular?

Angular 5+ begins to use service workers that increase reliability and performance of the application without coding against this.
These are the great advantages of angular, and the Angular service worker is designed for:

- Improving the performance regarding the unreliable network connection.
- Minimizing the risks of serving out-dated content.
- Optimizing the end user experience.

- The main design goal of Angular Service Worker are:
- Caching an application.
- When users refresh applications, firstly they see latest version of cached file.
- The Updates happen in the background process. Do not interrupt other processes.
- When Updates happens, earlier version of the application is served until an update is ready to use.

Prerequisites to Supports Service Workers:
For supporting of service workers we must have the following Angular and Angular CLI versions and also our web application must run in a web browser.

- Angular 5 or later
- Angular CLI 1.6 or later

Q155. What are angular services?

Primarily, we use services in Angular for reusability. If we need the same code in many components, then we can create the services.

Services are commonly used to store data and make HTTP calls.
The main idea behind a service is to provide an easy way to share the data between the components, and with the help of dependency injection (DI) ,you can also control the sharing of service instances.

Services are used to fetch the data from the RESTful API.

There are various uses of services, like –

- Communicate between two components when data is passed and both components are separated.
- Use the HTTP services to get/post/update required data from an external data source.

Q156. What Is Singleton Service?

In Angular, there are 2 ways to make a service singleton:

Using providedIn

Declaration of the service should be providedIn the application root.
It is the preferred way to create a singleton service.

customers.service.ts

src > app > customers.service.ts > ...

```
1   import { Injectable } from '@angular/core';
2
3   @Injectable({
4     providedIn: 'root'
5   })
6   export class CustomersService {
7
8     constructor() { }
9   }
10
```

Another way to create a singleton service - Include service in the AppModule
Customer.service.ts

customers.service.ts

src > app > customers.service.ts > ...

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

@Injectable()
export class CustomersService {

  constructor() { }
}

```

app.module.ts

app.module.ts

src > app > app.module.ts > ...

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import {FormsModule} from '@angular/forms';
import { CustomersService } from './customers.service';


@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
  providers: [CustomersService],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Q157. What is HTTP Services?

Ans. HTTP Services are the services for get, post, delete, update the external data using http module. In Angular, to use the http service we need to import the http module.

Q158. What are the different HTTP methods supported by Angular?

Ans. GET, POST, PUT, PATCH, and DELETE.

Q159. What are the difference between Patch () and Put ()?

Ans. Patch and Put methods are used to update the data, but the main difference is that Put is used to update all the data fields, however, Patch is used for updating partial few data fields.

For example-

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

In short, we can define these methods as-

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

Q160. Why we use services for transfer of the data while we have @Input and @Output decorator?

Ans. We use services for using http services and communication between data within the component when the component does not have the parent and child relationship.

Q161. How we can send the value from one component to another component using service and there is no relation between a parent and child?

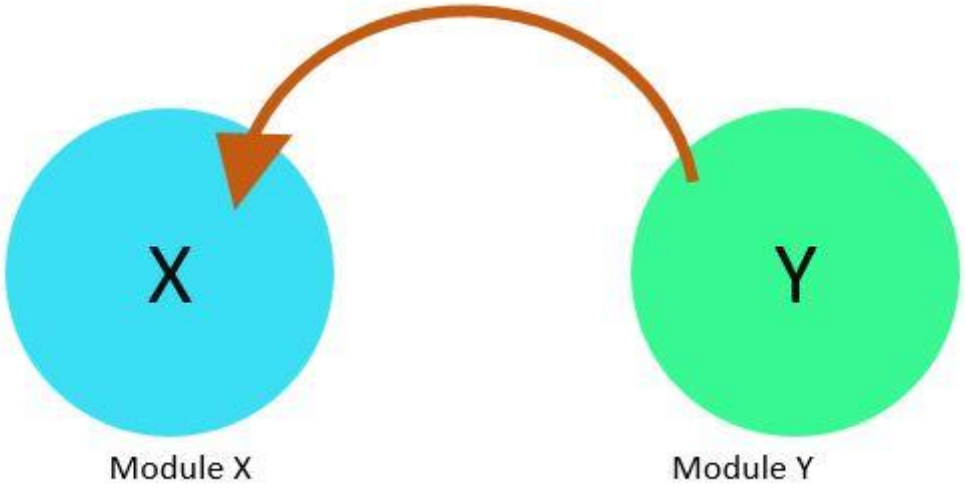
Ans. In this case we can pass the value using services with the help of get and set method.

Q162. What are providers?

Ans. Providers mainly provides the instruction to get the value for dependency injection. In Angular, it is a method by which we can use the singleton pattern. We can add dependency within providers like services. in the providers section of app.module.ts and all the component can use it.

Q163. What Is a Dependency?

When module X of an application requires module Y for execution, then module Y has the dependency on module X.



Q164. What is Dependency Injection (DI)?

Ans. DI is connected to the Angular framework and used everywhere to provide new components with the services or other things it require. Components consume services, that is, you can inject a service into a component by giving the access to component of that service class.

Dependency injection is a design pattern. Angular has its own DI framework, which is used in the design of Angular applications for increasing the efficiency and modularity.

Dependencies are services or objects that a class needs to perform its function. DI is a coding pattern in which a class requires dependencies from external sources rather than creating by itself.

Dependency Injection is a powerful model for managing code dependencies.
DI is a way to create objects that depend on other objects.
Angular has its own DI framework pattern, and an Angular application cannot be created without Dependency injection (DI).
A DI system provides dependent objects when it instantiates an object.

Q165. What Is Dependency Injection Pattern?

DI is an application design model and you cannot create an Angular application without dependency injection (DI).

Q166. What is @Injectable?

Ans. @Injectable is the decorator which is used in Angular application so that Angular knows that the class can be used in the dependency injector.

For example, when we create the service and want to use it in the application, we have to use @Injectable within services when we create. However, a component can use the service if service don't use @Injectable but the service cannot consume other service without this

decorator.

Q167. How many services we can add into Providers?

Ans. Provider is an array. We can add multiple dependencies in the providers so we can add many services in the providers.

Q168. How many decorators are there in Angular?

Ans. We have following decorator in angular:

- **Class decorators**

Class decorator is high level decorator and it allows us to tell Angular that a particular class is either a component or module.
Examples of class decorator are- @Component and @NgModule

- **Property decorator**

These are second most common decorators which is used in Angular application. We can create specific properties in class using property decorator and use it with the class while communicating between parent and child or child and parent components.
Example of property decorator are- @Input and @Output

- **Parameter decorator**

Parameter decorators are also important and plays an important role in creation of Angular application. We can use when we need to inject the parameter.
Example: @Inject

- **Method decorator.**

Method decorator are also the most common decorators that is used in the Angular application. We are able to create specific methods within our class with required functionality. Example- @HostListner.

Q169. What Are Injectors?

In Angular, service is just a class until it is registered with an Angular dependency injector.
The injector is responsible for creating angular service instances and injecting them into classes.
You can rarely create an injector by own but Angular creates it automatically during the booting process.
Angular doesn't know automatically either you want to create instances of your services or injector. You need to configure it by specifying providers for each service. Actually, providers tell the injector how to create the service and without provider we cannot create the service.
Bootstrap defines the components that should be start when this module is bootstrapped. The components listed here will be automatically added to the entryComponents.

Q70. Why @Inject()?

The @Inject is a special kind of technique through which Angular knows that a parameter must be injected.
Syntax of Inject decorator and metadata:

```
@Inject({
  token: any
})
```

When @Inject () is not present then Injector is used as the type annotation of the parameter.

```
import { Component, Inject } from '@angular/core';
import { HttpClient } from '@angular/common/http';

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

  constructor(@Inject(HttpClient) private http) {
    // Use this.http which is the Http Provider
  }

}
```

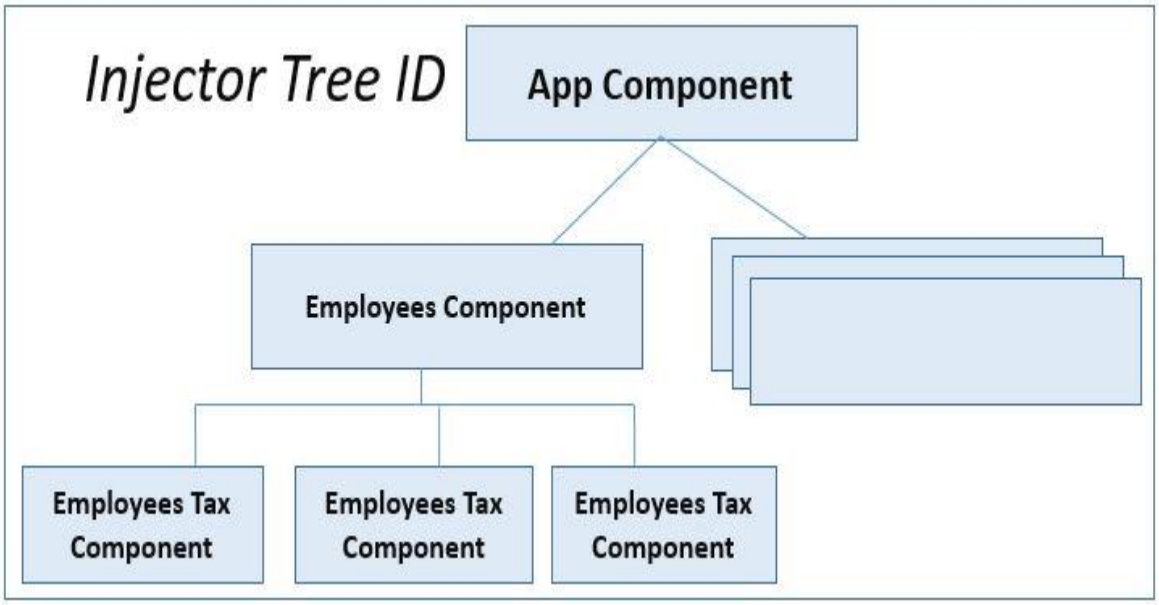
At this point, @Inject is a manual way to specify this lookup token, followed by the lowercase HTTP argument to tell Angular what to assign it.

Q171. What Are Hierarchical Dependency Injectors?

Angular has a Hierarchical Dependency Injection system. It is a tree of injectors that are parallel of application component tree. You can again configure the injectors at any level of the component tree.

Q172. What Is Injector Tree?

In the Dependency Injection guide, you learned how to set up a dependency injector and how to get dependencies if you need them.
An application can have multiple injectors. An Angular application is a component tree where each component instance has its own injector. The tree of components parallels the tree of injectors.
Figure of 3 level component tree is shown below:
In Angular 2 or later, we use Backtick for multiline HTML. For example, if we want the long data in the HTML file within separate lines, we can use the Backtick.



Q173. What is backtick and how we use it in Angular?

Ans. In Angular 2 or later version, Backtick is used for writing multiline HTML code. For example, if you want the long HTML file data in separate line, then you can use the Backtick (` `).

Q174. What is DOM Shadowing?

Dom Shadowing allows us to hide DOM logic behind other elements. We can say that it allows us to apply the style based on scope and requirements of the application.

Q175. If we send any number value from child to parent component then the emitter will send number or string?

Ans. In this case we will send number as below.

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

Q176. What is RXJS?

Ans. RxJS stands for Reactive Extensions for Java Script. It is a Library for reactive programming using observables.

Q177. What are Observables?

Ans. Observables helps in managing asynchronous data and asynchronous event handling. Observables are slow and it can have multiple values over time. Observable is an interface and is used to manage asynchronous operations.
The HTTP Client module uses observables to handle AJAX requests and responses.

Angular HttpClient returns observables from HTTP method calls.

- EventEmitter class extends Observable.
- The Router modules use observables to request and respond of user events.
- The Forms modules use observables to request and respond to user events. To use observables, Angular uses a third-party library called Reactive Extensions (RxJS).

Q178. What is the difference between promises and observable?

Ans. Following are the differences between Promises and Observable:

- Promises handles a single event when the asynchronous operation is completed or fails whereas Observable is like a stream that allows to pass zero or more event.
- Promises are less preferable, while Observable is more preferable because its providers have the feature of promises or more.
- Promises cannot be canceled, while observables can be canceled when any asynchronous operation is no longer needed.

Q179. What are subscribers?

Ans. Subscribers is used to subscribe the observables.

Q180. How we can handle the error in Angular?

Ans. We can handle the error using try catch in the Angular application and we can also handle the error using error section when subscribing the observables.

Q181. What is Routing in Angular?

Ans. Each application contains many pages, so using routing we can navigate from one page to another. Or you can say that routing helps a user in navigating to different pages using links.

An Angular Router is a tool, a library that configures the navigations between states and views in the Angular application.
The Angular Core Team writes and maintains the Routing library.
Angular router is imported from the @angular/router.

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

The basic concept of Angular Router, enables you to:

- Redirect a URL to another URL.
- Resolve data before a page is displayed.
- Run scripts when a page is activated or deactivated.
- Lazy load parts of our application.

The router supports styles with two LocationStrategy providers:

- PathLocationStrategy— this is the default style.
- HashLocationStrategy—it adds the route path to the hash (#) in the browser URL.

Q182. What are Routes?

Routes is an array of route configurations. The "RouterModule.forRoot" method in the module is imported to configure the router.

```
type Routes = Route[];
```

Each Route has the following properties:

```
interface Route {
  path?: string
  pathMatch?: string
  matcher?: UrlMatcher
  component?: Type<any>
  redirectTo?: string
  outlet?: string
  canActivate?: any[]
  canActivateChild?: any[]
  canDeactivate?: any[]
  canLoad?: any[]
  data?: Data
  resolve?: ResolveData
  children?: Routes
  loadChildren?: LoadChildren
  runGuardsAndResolvers?: RunGuardsAndResolvers
}
```

List of properties and has the following order:

- path: It is a string and is used for route matcher DSL.
- pathMatch: It is a string and used to specify the matching strategy.

- matcher: It is used to define the custom strategy for path matching.
- component: It is a component type.
- redirectTo: It is the URL fragment and it will replace the current matched segment.
- outlet: It is the name component that is to be loaded.
- canActivate: It is an array of DI tokens and used to handle the CanActivate handlers.
- canActivateChild: It is an array of DI tokens and used to handle the CanActivateChild handlers.
- canDeactivate: It is an array of DI tokens and used to handle the CanDeactivate handlers.
- canLoad: It is an array of DI tokens and used to handle the CanLoad handlers.
- data: It is an additional data provided to the component using the Activated Route.
- resolve: It is a map of DI tokens used to look up data resolvers.
- runGuardsAndResolvers: It is defined when guards and resolvers will run and by default, they run only when the matrix parameters of the route change.
- children: it is an array of child route definitions.
- loadChildren: It is a reference to lazy loaded child routes.

Q183. How Angular Router Works?

Angular Router performs the following actions:

- The router reads the browser URL that user wants to navigate.
- The router applies a URL redirect (if one is defined otherwise page not found the error).
- It finds out which router state corresponds to the URL.
- It runs the guards that are defined in the router state.
- It resolves the required data for the router state.
- It activates the Angular components to display the page.
- It manages navigations and repeats the above steps whenever a new page is requested.

Angular Router introduces the following terms:

- <base href>
- Router imports
- Configuration
- Router outlet
- Router links
- Router state
- Activated route
- Router events

Q184. What Is <base href>?

Most Angular routing applications should have the <base> element in the index.html or layout page in the <head> tag.
When using the PathLocationStrategy, browsers need to be told what must be prefixed to the requested path to generate the URL.
You can specify a base URL as shown below:

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

OR

index.html

```
src > index.html > ...
1   <!doctype html>
2   <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <title>Routingapp</title>
6     <base href="/">
7     <meta name="viewport" content="width=device-width, initial-scale=1">
8     <link rel="icon" type="image/x-icon" href="favicon.ico">
9   </head>
10  <body>
11    <app-root></app-root>
12  </body>
13  </html>
14
```

Angular route is defined like this –

```
{ path: 'users', component: UserComponent, data:{ title:'User List'} }
```

If <base href="/" >, the route become "/users".
If <base href="/angular" >, the route become "/angular/users".

Q185. How to Append Base URL to HTTP requests?

We can append base URL to HTTP requests using these two:

- Dependency Injection
- Using HttpInterceptors

The following example is of append base URL by using DI:

Firstly, we register a base URL in provider in the NgModule and after registering this BASE_URL, it is available globally in the Application.

```
//Runtime or injector configuration
//providers is used for runtime injector configuration.
providers: [{ provide: 'BASE_URL', useFactory: getBaseUrl }]
```

Now, provide factory method which gets the base URL from <base> element.

```
export function getBaseUrl() {
  return document.getElementsByTagName('base')[0].href;
}
```

Finally, we inject the baseUrl and add it to the URL-

```
export class GetUserComponent {
  users: any;
  constructor(http: HttpClient, @Inject('BASE_URL') baseUrl: string) {
    http.get(baseUrl + 'api/users').subscribe(data =>
      {
        this.users = data.json()
      },
      error => console.error(error));
  }

}
```

The following example to add the base URL using HttpInterceptors:
To create an interceptor, we need to create an Injectable class that implements HttpInterceptor.

First, register the interceptor in the module provider:

```
// providers: [CustomersService],
//Runtime or injector configuration
//providers is used for runtime injector configuration.
providers: [{ provide: HTTP_INTERCEPTORS, useClass: ApiInterceptor, multi: true }]
```

And after register interceptor:

```
import { Injectable } from '@angular/core';
import { HttpInterceptor, HttpRequest, HttpHandler, HttpEvent } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class ApiInterceptor implements HttpInterceptor {
  //Intercepts HttpRequest and handles them.
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    const baseUrl = document.getElementsByTagName('base')[0].href;
    const apiReq = req.clone({ url: `${baseUrl}${req.url}` });

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

Now, we can access the base URL globally in the application.

Q186. What Is PathLocationStrategy?

A LocationStrategy is used to configure the Location service which represents its state in the browser URL path and PathLocationStrategy is a default routing strategy.
While using the PathLocationStrategy, we need to provide APP_BASE_HREF in the module or base element in the application document.

Q187. What Is HashLocationStrategy?

To enable HashLocationStrategy {useHash: true} in an Angular application, you need to provide routes with the router module.

For Example-

```
  //Composability and Grouping
  //imports used for composing modules together.
  imports: [
    BrowserModule,
    //enableTracing enables debugging purposes only
     //useHash enables the location strategy that uses the
URL fragment instead of the history API.
     RouterModule.forRoot(appRoots, { enableTracing: true,
useHash:true })
  ],
```

The HashLocationStrategy adds the route path to the hash (#) in the browser URL.
The hash (#) part of the URL is called the hash fragment.
While using HashLocationStrategy for routing and providing a base Href, it is always placed after the hash (#).For ex-
http://localhost:8080/#/UserDetail/1
The Hash style routing withthe anchor tag technique is used to get client side routing and the URL is never sent to the server.

The anchor tag, when used with the hash (#), allows us to jump to a point, within application.

The URL will look like this:

- http://localhost:8080.
- http://localhost:8080/#/Users.
- http://localhost:8080/#/UserDetail/1.

In the above URLs "#/Users" and "#/UserDetail/1" are never sent to the server.

Q188. How do you change the base URL Dynamically?

Instead of setting the base element href value, you can set the base URL programmatically, by providing APP_BASE_HREF with your custom operation.

Q189. What Are Router Imports?

It is an optional service that presents a special component view for a given URL.

It has its own @angular/router library package and is not part of the Angular core.
The Angular package will look like this:

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

Q190. What is Router module?

The Router module is a module that provides the required service providers and directives to navigate from one view to another in the application.

Q191. How can you define routing in Angular?

Ans. You can define routing in Angular by the following steps.

First, import the Routes and RouterModule from @angular/router in app.module.ts, like

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

Create a const of Routes type and define your routes array here, like

```
const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'dashboard', component: DashboardComponent },
  { path: 'login', component: LoginComponent },
  { path: '**', redirectTo: 'home' }
];
```

Import these 2 components (Home, Student).

To add the routes in the application use RouterModule.forRoot inside the @NgModule, like

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

Q192. What is a RouterOutlet?

Ans. RouterOutlet is a substitution for templates that renders the components. In other words, it

represents or renders the components in a template at a particular location.

The Router-Link, RouterLink-Active and Router outlet are directives provided by the Angular RouterModule package.
It also provides the navigation and URL manipulation capabilities. It also renders the components for specific location of your applications. Both the template and templateUrl renders the component where this directive is used.

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

Q193. Is it possible to have a multiple router-outlet in the same template?

Ans. Yes, it is possible. We can use multiple router-outlets in the same template by configuring our routers and by simply adding the router-outlet name.

app.component.html

src > app > app.component.html > div.row > div

```
<div class="row">
    <div class="user">
        <router-outlet name="users"></router-outlet>
    </div>
    <div class="detail">
        <router-outlet name="userDetail"></router-outlet>
    </div>
<div>

```

Q194. What is Router links?

Ans. RouterLink is a directive that is used in view (html) to tell the Angular which route to be navigated. It is used to link the specific part of the application.

For example-

```
<a routerLink="/user/id"> See the User detail</a>
```

Q195. What Is RouterLinkActive?

The RouterLinkActive is a directive. It is used to add the active CSS class to the element when the associated RouterLink becomes active.

```
@Directive({
    selector: '[routerLinkActive]',
    exportAs: 'routerLinkActive'
} )
```

Consider the following example to active a link:

```
<a routerLink="/user/detail" routerLinkActive="active-link">User Detail</a>
```

You can also configure more than one class and it will look like as below.

```
<a routerLink="/user/detail" routerLinkActive="active-class1 active-class2">User detail</a>
<a routerLink="/user/detail" [routerLinkActive]="['active-class1', 'active-class2']">User detail</a>
```

Q196. What is Router State?

Ans. After the end of each successful navigation lifecycle, the router creates a tree of ActivatedRoute objects that make the current state of the router. Using 'routerState' property you can access state of routes anywhere in the application.

RouterState is an interface and represents the state of the router.
It will look like as below:

```
interface RouterState extends Tree {
snapshot: RouterStateSnapshot
toString(): string

}
```

It is also a tree of activated routes.
We can access the current RouterState from anywhere in the Angular application using the Router service and routerState property.

Q197. What Is ActivatedRoute?

ActivatedRoute is an interface and contains the information about the route associated with a component and loaded into an outlet and it can also be used to traverse the router state tree.
And it contains the following list of Properties:

- Snapshot: It is the current snapshot of this route.
- URL: It is an observable of the URL segments and it matched by this route.
- Params: It is an observable of the matrix parameters scoped to this route.
- QueryParams: It is an observable of the query parameters shared by all the routes.
- Fragment: It is an observable of the URL fragment shared by all the routes.

- Data: It is an observable of the static and resolved data of this route.
- Outlet: It is a constant and outlet name of the route.
- Component: It is a constant and a component of the route.
- RouteConfig: This configuration is used to match the route.
- Root: This is the root of the router state.
- Parent: The parent of this route in the router state tree.
- FirstChild: The first child of this route is in the router state tree.
- Children: The children of this route are in the router state tree.
- pathFromRoot: The path from the root of the router state tree to this route.
- paramMap: It is read-only.
- queryParamMap: It is read-only.

Q198. What are Router events?

Ans. During each navigation, the Router emits navigation events through the Router.events property. These events range from the beginning of navigation to the end of too many intermediate points.

Whenever you browse the root, the router emits navigation events using Router.events property.
The sequence of router events is given below:

- NavigationStart
- RouteConfigLoadStart
- RouteConfigLoadEnd
- RoutesRecognized
- GuardsCheckStart
- ChildActivationStart
- ActivationStart
- GuardsCheckEnd
- ResolveStart
- ResolveEnd
- ActivationEnd
- ChildActivationEnd
- NavigationEnd
- NavigationCancel
- NavigationError

The Router events are also logged in the console when the enableTracing option is enabled.
The NavigationStart event is triggered when the navigation begins.
The RoutesRecognized event triggered when the routes are recognized.
The RouteConfigLoadStart event triggered before the Router is lazily loaded.
The RouteConfigLoadEnd event triggered after the route has been lazily loaded.
The NavigationEnd event triggered when navigation ends successfully.
The NavigationCancel event triggered when navigation is cancelled.
The NavigationError event triggered when router navigation fails due to an error.

Q199. What is Wildcard route?

Ans. Wildcard route is added to intercept invalid URLs. A wildcard route has a path consisting of two asterisks (**). It matches all the URLs. The router will select this route if it cannot match the route earlier in the configuration. A wildcard route can go to the custom "404 Not Found" component or redirect to an existing route.

The router selects the route with the first match wins strategy. Wildcard routes are the least specific routes in the route configuration. Make sure it is the last route in the configuration otherwise those links which are defined after the wildcard route cannot be navigated.

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

Q200. What is pathMatch property in routing?

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

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

The property pathMatch:'prefix' tells the router to match the redirect route when the remaining URL start with the redirect route prefix path.

app-routing.module.ts

src > app > app-routing.module.ts > ...

```
1  import { NgModule } from '@angular/core';
2
3  import { Routes, RouterModule } from '@angular/router';
4
5
6  import { DashboardComponent } from './dashboard/dashboard.component';
7  import { LoginComponent } from './login/login.component';
8  import { PagenotfoundComponent } from './pagenotfound/pagenotfound.component';
9
10 const routes: Routes = [
11   { path: '', redirectTo: 'home', pathMatch: 'full' },
12   { path: 'dashboard', component: DashboardComponent },
13   { path: 'login', component: LoginComponent },
14   { path: '**', component:PagenotfoundComponent }
15 ];
16
17 @NgModule({
18   imports: [RouterModule.forRoot(routes)],
19   exports: [RouterModule]
20 })
21 export class AppRoutingModule { }
22
```

Q201. How can we pass parameter in Routing?

Ans. 1. On html page, code will be written like this:

```
<a [routerLink]="['/Student', student.Id]">Student Record </a>
```

2. After that, in component.ts file

Inject ActivatedRoute in constructor

```
constructor(private route : ActivatedRoute){}
```

After that, use as shown below:

```
constructor(private route: ActivatedRoute) {
  this.route.params.subscribe((params: Params) => { this.id = params["id"] });
}
```

Q202. What are the guard interfaces supported by router?

Ans. The router supports multiple guard interfaces, like:

CanActivate checks whether the current user is allowed or not to activate the requested route.
CanActivateChild checks whether the current user is allowed to activate the requested child route or not.
CanDeactivate checks whether the current user has permission to deactivate the requested route.
Resolve is used to perform route data retrieval before route activation.
CanLoad decide if a module can be loaded configured with loadChildren property.

Q203. Difference between [routerLink] and routerLink?

Ans. Whenever you want to pass the URL as dynamic then we have to use routeLink in square parentheses as a property binding, as shown below.

app.component.html

src > app > app.component.html > ...

```
1
2   <a routerLink="path" ></a>
3
4
```

So, the variable (yourVariable) could be defined within the class and it should have a value like:

app.component.ts

src > app > app.component.ts > ...

```
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-root',
5     templateUrl: './app.component.html',
6     styleUrls: ['./app.component.css']
7   })
8   export class AppComponent {
9
10    path = "/home";
11
12  }
13
```

However, when we use without parentheses, you just need to pass a string and it is static can't be changed.

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

Q204. What is Form and how many strategies we have for developing the forms?

Ans. Forms are required to access the application, view the request on the page and submit the data and perform the operation within the requested page in the application.

We have mainly 2 strategies which is used to develop the form.

Template Driven Form
Reactive Form

Q205. What is Template driven form and reactive form and why we use in Angular?

Ans. In Angular, when we work with form to get, post, and update the data, etc., we have 2 approaches – (a) either we can use Template driven form (b) the Reactive form approach.

Template Driven Form
Template driven forms are where we write all the logic, validations, controls etc. within the template part of the code i.e. html code.

Reactive Form
Reactive form is basically a model driven where you can add your logic to the component class and its templates.

Q206. When we use the Template-driven form, which module we have to add and

where?

We will import the FormsModule from angular/forms and add within imports array in the app.module.ts as following:

Q207. In how many ways we can add form validation?

Ans. We can add form validation in 2 ways as Template driven and Reactive.

Q208. How we will use Template-Driven Form Validation?

Ans. When we use the template-driven form validation, we have to add the html validation with each form control like required, max length, etc.

We will also add the directive like *ngIf, ngModel, etc. that match these attributes with the validator function.

In this validation form, it is valid only if all validation results are true. We will use the button which is enabled only if form is valid as you can see below.

```
<form (ngSubmit)="onSubmit()" #testform="ngForm">
    <!-- ...all html control in which we will use validation... -->
    <input type="submit" [disabled]="!testform.valid" value="Submit">
</form>
```

Q209. Why we are checking dirty and touched?

Ans. If you want to display errors in your application, if the user does below 2 things, before these we will prevent the errors:

Dirty: If user changes the value then it is considered as dirty.
Touched: If the user move to control and then move to another control means on blurs the form control then it is considered as Touched.

Q210. What are Reactive Forms?

Ans. Reactive forms provides us the model-driven approach when we build the form using form controls. In this approach, we can use multiples controls in a group and validation form controls, so we can say that with this approach, we can easily implement more advanced forms.

Q211. When we will use the form using Reactive approach, which module we have to add and where?

Ans. When we implement as Template driven the difference only within template driven that we use FormsModule but here we are using ReactiveForms Module within app.module.ts as shown below.

app.module.ts

src > app > app.module.ts > ...

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import {ReactiveFormsModule} from '@angular/forms';


@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    ReactiveFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Q212. What is Dynamic Form and how we can implement it in Angular application?

In Angular we can build the dynamic form as per our requirements. We can create the form dynamically using the metadata and object model.

For more details, we can check this link.
https://angular.io/guide/dynamic form

Q213. Which classes we use for show in form according to status of form:

We use following class:
1- ng-valid
2-ng-invalid 3-ng-pending 4-ng-pristine 5-ng-dirty
6-ng-untouched 7-ng-touched

Q214. What is HttpClient in Angular? What is the role and responsibility of HttpClient?

HttpClient makes HTTP requests and responses.
Most of the web applications communicate with backend services via the HTTP protocol and all modern browsers support 2 different APIs for making HTTP requests, that are-

- XMLHttpRequest interface
- fetch() APIs

The HttpClient is easy to use, as an alternative to HTTP.
It is an improved substitution of HTTP. They expect to deprecate http in Angular 5 and remove it in the later version.

The new HttpClient service is included in the HTTP Client Module that is used to start HTTP request and responses in angular application. The HttpClientModule is a replacement of HttpModule.
HttpClient also offers us advanced features such as the ability to listen for progress events and interceptors to modify requests or responses.
Before using HttpClient, you need to import the HttpClient Module and the

And is imported from @angular/common/http.
You must import HttpClientModule after BrowserModule in the application.
HttpClient supports request mutation, i.e. sending data to the server with HTTP methods such as PUT, POST, and DELETE.
HttpClient uses the XMLHttpRequest browser API to perform HTTP request.
HttpClient Perform following HTTP requests:

- GET method – get()
- POST method – post()
- PUT method – put()
- DELETE method – delete()
- HEAD method – head()
- JSONP method – jsonp()
- OPTIONS method – options()
- PATCH method – patch()

Q215. What Is HttpInterceptor?

HttpInterceptors is an interface used to implement the intercept method.
Intercepts makes HttpRequest and handles them.
It is an advanced feature that allows us to intercept each request and response and modify it before sending or receiving.
Interceptors capture every request and manipulate it before sending it, and also capture every response and process it before completing the call.
First of all, we can implement our own interceptor service and this service will "catch" each request and add an Authorization header.
You can see it in the following example:

mylnterceptor.service.ts

src > app > mylnterceptor.service.ts > ...

```
1   import { Injectable } from '@angular/core';
2   import { HttpEvent, HttpInterceptor, HttpHandler, HttpRequest } from '@angular/common/http';
3   import { Observable } from 'rxjs';
4
5
6   @Injectable({
7     providedIn: 'root'
8   })
9   export class MyInterceptor implements HttpInterceptor {
10    //Intercepts HttpRequest and handles them.
11
12    intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
13
14      const reqHeader = req.clone({ headers: req.headers.set('Authorization', 'MyAuthroken1') });
15
16      return next.handle(reqHeader);
17    }
18  }
```

After that you can configure your own interceptor service (MyInterceptor) and HTTP_INTERCEPTORS in the app Module.

app.module.ts

src > app > app.module.ts > ...

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

import { HTTP_INTERCEPTORS } from '@angular/common/http';
import { MyInterceptor } from './myinterceptor.service';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [{ provide: HTTP_INTERCEPTORS, useClass: MyInterceptor, multi: true }],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

By using this logic, Authorization token will be added to each request. It is also possible to overwrite request headers by using set() method.

Q216. What is the difference between HttpModule and HttpClientModule?

In HttpClientModule -
The HttpClientModule imported from '@angular/common/http' as shown below–

```
import { HttpClientModule } from '@angular/common/http';
```

NgModule which provides the HttpClient and is associated with the components services and the interceptors can be added to the chain behind HttpClient by binding them to the multi-provider for HTTP_INTERCEPTORS.

In HttpModule –
Http deprecate @angular/http in support of @angular/common/http.
HttpModule is imported from '@angular/http' –

```
import { HttpModule } from '@angular/http';
```

They both support HTTP calls but HTTP is the older API and is deprecated now.
The new HttpClient service is included in the HttpClientModule that is used to initiate HTTP request and responses in angular application. The HttpClientModule is the replacement of

HttpModule.

Q217. What's the difference between HTTP and HttpClient?

The HttpClient is used to perform HTTP requests and it is imported form '@angular/common/http' and is easy to use as an alternative of HTTP.
HttpClient is an improved replacement of Http. They expect to deprecate Http in Angular 5 and remove it in a later version.
It is an upgraded version of http from '@angular/http' module with the following improvements:

- Immutable request and response objects .
- Interceptors allow middleware logic to be inserted into the pipeline.
- Progress events for both request and response.
- Typed.
- Event firing.
- Synchronous response body access.
- Support JSON body types and JSON by default and now, no need to be explicitly parsed.
- Automatic conversion from JSON to an object.
- Post request verification.
- A flush based testing framework.
- Simplified syntax for headers.

Example of HttpClient-

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

@Injectable() export class CustomerService {
//Inject HttpClient into your components or services
constructor(private http: HttpClient) {}
}
```

Example of Http-

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

@Injectable() export class CustomerService {
//Inject HttpClient into your components or services
constructor(private http: Http) {}
}
```

Q218. What Are HttpHeaders?

The Http Headers is immutable Map and every set( ) returns a new instance and applies the changes with lazy parsing.

An immutable set of Http headers, with lazy parsing.
In HttpHeaders Constructor –

```
constructor(headers?: string | { [name: string]: string | string[];});
```

Imports HttpHeaders from '@angular/common/http':

```
import {HttpHeaders ) from '@angular/common/http';
```

HttpHeaders class contains the following list of methods:

- has() - Checks for existence of header by given name.
- get() - Returns the first header that matches given name.
- keys() - Returns the names of the headers.
- getAll() - Returns list of header values for a given name.
- append() - Append headers by chaining.
- set() - To set a custom header on the request for a given name.
- delete() - To delete the header on the request for a given name.

Q219. How to set a custom header on the request?

To set a custom header on the request, first of all we have to instantiate HttpHeaders() object and pass ('header', 'value') into the function.

```
let requestHeaders = new HttpHeaders().set('Content-Type', `text');
```

In the above example we set "Content-Type" header value to be "text" and the default header "Content-Type" is "application/json".
It is a type of immutable Map so if you assign a new value it will initialize the object again.

```
let requestHeaders = new HttpHeaders()
.set('Content-Type', `application/json');

requestHeaders = requestHeaders.set('authorization', 'Bearer + token);
```

We can also add headers by chaining HttpHeaders() constructor and code will look like this-

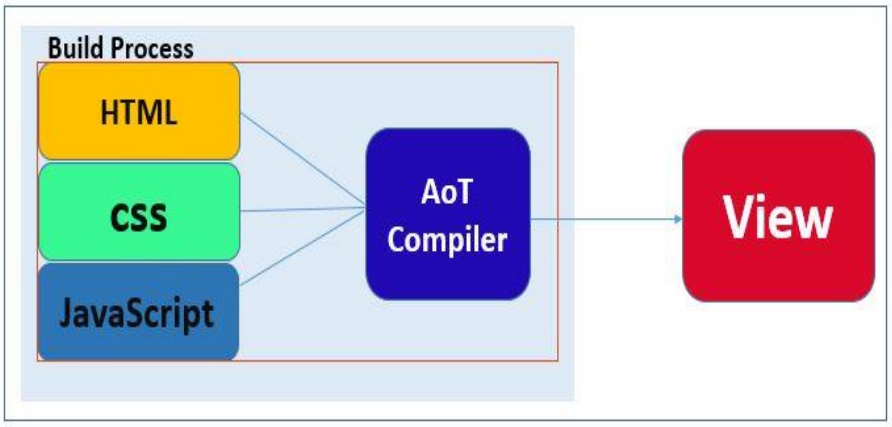
```
let requestheaders = new Httpileaders()
.set('Content-Type', 'application/j son')
.set('authorization', 'Bearer ' + token);
```

Q220. What Happens If the Request fails on the Server Due to Poor Network Connection?

If the Request fails on the server HttpClient will return an error instead of a successful response.

Q221. What Is the Angular Compiler?

The Angular compiler converts the applications code i.e. HTML and TypeScript, into JavaScript code before the browser downloads and executes that code.
The @NgModule metadata plays an important role in guiding the compilation process and also tells the compiler which components to be compiled for this module and how to link this module with other modules.



The Angular offers 2 ways to compile the code of application:

(1)Just-in-Time (JIT) - JIT compiles the application in the browser at runtime (compiles before execution).

(2)Ahead-of-Time (AOT) - AOT compiles the application at build-time. (compiles while running).
The JIT compilation is the default when we run the build or serve CLI commands:

```
ng build
ng serve
```

The AOT compilation, we add the —aot flags to build or serve the CLI commands. For ex-

```
ng build --aot
ng serve --aot
```

Q222. Why we need Compilation in Angular?

We need compilation for greater efficiency, performance improvements, faster rendering, and sometimes to detect template errors.

Q223. Why Compile with AOT?

Ans. We compile due to following reasons:-

- Faster Rendering
- Asynchronous Requests
- Detect template errors earlier
- Smaller Angular frameworks download size
- Better Security

Q224. What Is the difference between JIT compiler and AOT compiler?

JIT (Just-in-Time) -

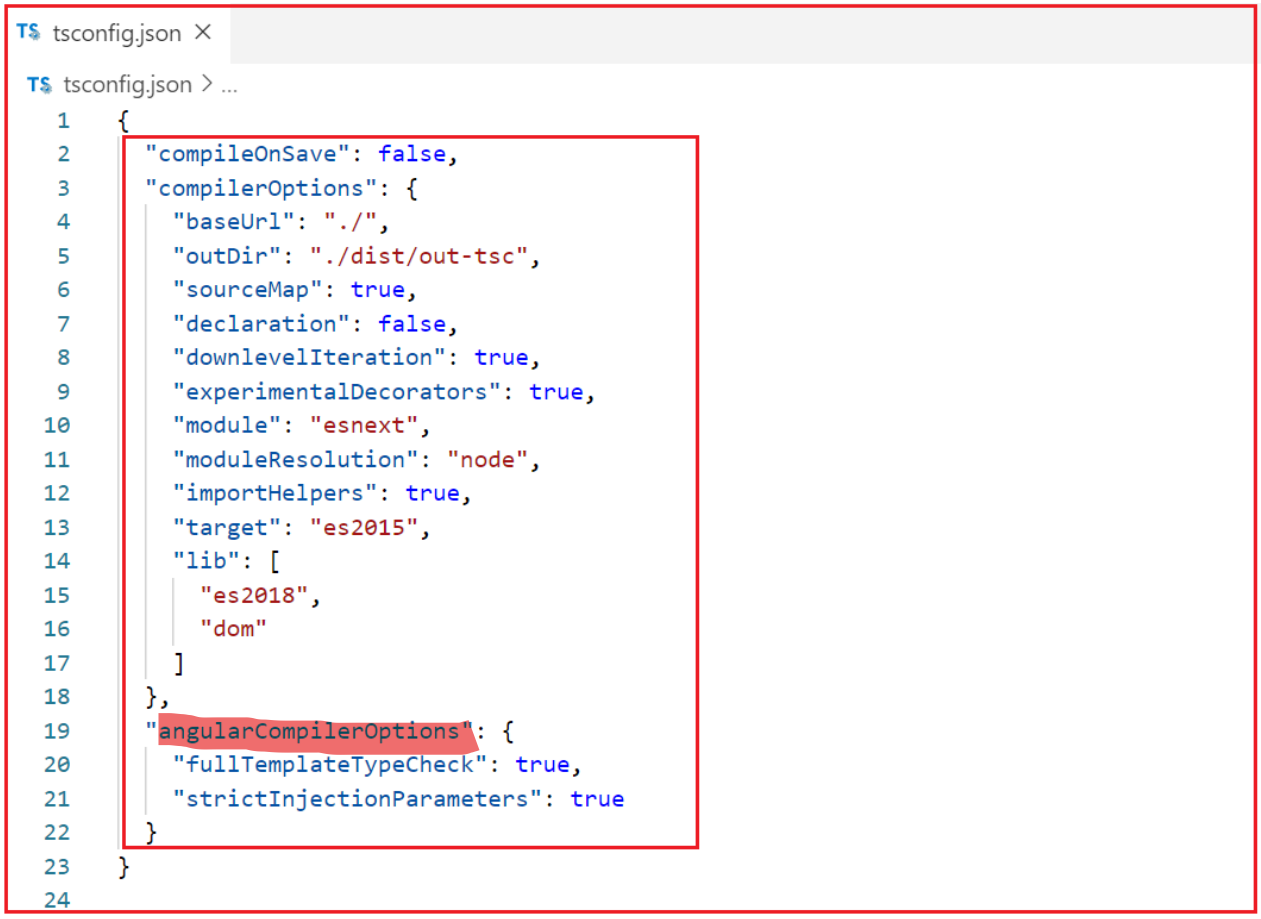
- JIT compiles our app in the browser at runtime.
- Compiles before execution.
- Each file is compiled separately.
- No need to build after changing the application code and it automatically reflects the changes in your browser page.
- Highly secure.
- Very suitable for local development.

AOT (Ahead-of-Time) -

- AOT compiles our app code at build time.
- Compiles while execution.
- Compiled by the machine itself, through the command line (Faster).
- All code compiled together, in lining HTML/CSS in the scripts.
- Highly secure.
- Very suitable for production builds.

Q225. What Are Angular Compiler Options?

We can control compilations of application using the compilerOptions in the tsconfig.json file. The tsconfig.json file will look like as shown below:



Q226. What Is Ivy Renderer?

The new Ivy renders and it is not stable for now and is only in beta version.
Ivy Renderer is new rendering engine designed to be backward compatible with existing rendering and focused to improve rendering speed and optimizes the size of the final package.
The main purpose of Ivy rendering is to speed up its load time and reduce the bundle size of application packages.

Features of New Ivy engine are:-

- ❖ Smaller builds.
- ❖ Faster rebuild times.
- ❖ Faster development.
- ❖ A simpler, more hack-able pipeline.
- ❖ Human readable code.

Q227. What Is Bazel Compiler? What does Angular doing with Bazel Compiler?

The Bazel Complier is built-in system used for almost all software created by Google. Starting with Angular 6, it will start having support for the Bazel compiler and when you compile the code with Bazel Compiler, you will recompile the entire base code, but compile only with the necessary code.
Bazel Complier uses advanced, local and distributed caching, optimized dependency analysis and parallel execution.

Bazel allows us to divide an application into different build units. In Angular, build units are defined at the NgModule level.

This means the the scope of a build can be as granular as a single NgModule. If a change is internal to an NgModule, only that module needs to be rebuilt.
Angular, Angular Universal, NgRx, and Tsickle all switched to Bazel as the build tool, and ship Bazel-built artefacts to npm.

Q228. What Is Angular Universal?

Angular Universal is the fastest technology that runs angular application on the server. It generates static pages of the application on the server through a procedure that is called server-side rendering (SSR). It also generates and serve those pages in response to browsers requests.

If you want to make a Universal application, you need to install the "platform-server " package.
This package has server implementations of following:
DOM
XMLHttpRequest
And other features

The platform- server application does not run in browsers because it compiles the application using the platform-server module instead of the platform-browser module.

Q229. How to Install Universal?

To get started, first you need to install these packages.

- @angular/platform-server - It is a universal server-side component and the platform-server application doesn't run in the browsers.
- @nguniversal/module-map-ngfactory-loader – It is used for handling lazy-loading in the context of a server-render.
- @nguniversal/express-engine – It is an express engine for the universal application.
- ts-loader - To transpile the server application. You can also install these packages using the following command:
- npm install —save @angular/platform-server @nguniversal/module-map-ngfactory-loader
- ts-loader @nguniversal/express-engine

Q230. Why Angular Universal?

There are 3 main reasons to create an universal application.

- Improve performance on mobile and other devices.
- Facilitate web crawlers. The crawling is a SEO part of your app.
- Render the first page quickly.

Google, Bing, Twitter, Facebook, and other social media sites depend on web crawlers to index the content of the application and make that content searchable on the web.

Q231. What are the key points to keep in mind when you are developing Angular apps?

There are 4 key points to keep in mind when developing an Angular application.

- The application level securities like authentication and authorization.
- Coding with Best Practices.
- Preventing cross-site scripting (XSS).
- Reporting vulnerabilities and HTTP Level vulnerabilities.

Q232. How to write Best Practices Applications?

As per experts, be careful when developing Angular application:

- ❖ We keep and watching the latest version of Angular.
- ❖ Don't try to add hacks or modification in Angular generates files.
- ❖ Avoid Angular Security Risk.
- ❖ Avoid direct use of the DOM APIs.
- ❖ Try to use offline template compiler.
- ❖ Try to prevent CSRF or XSRF attacks in your web application.
- ❖ Try to prevent JSON data in your web application.

Q233. What Is Cross Site Scripting (XSS) Attack?

The Cross Site Scripting (XSS) attack is a type of injection and attackers inject their web applications using the client side scripts and malicious code on web pages.
An attacker can inject vulnerability scripts and malicious code in your web applications.
Cross Site Scripting (XSS) attacks are common on web browsers and run on approximately 84% websites. Angular treats all values as untrusted by default. These are the great advantages of Angular. The value is the vulnerability inserted in the DOM since:

Q234. How to Preventing Cross Site Scripting (XSS) in Angular?

How Angular Protects Us from XSS Attacks?

Ans :By default, the Angular treats all values as untrusted. This is the great advantages of Angular.
The value is inserted Vulnerability into the DOM from:

- A Template
- Property
- Attribute
- Style
- Class Binding
- Interpolation
- And so on.

Angular recognizes the value as unsafe and automatically sanitizes and removes the script tag and other security vulnerabilities.
Angular provides built-in, by default values as untrusted, anti XSS, and CSRF/XSRF protection.
The CookieXSRFStrategy class takes care from preventing XSS and CSRF/XSRF attacks.
The DomSanitizationService takes care of removing the dangerous bits in order to avoid XSS attacks.

Angular applications must follow the same security principles as regular web applications follows:

- You should avoid direct use of the DOM APIs.
- You should enable Content Security Policy (CSP) and configure your web server to return appropriate CSP HTTP headers.
- You should use the offline template compiler.
- You should use Server-Side XSS protection.
- You should use DOM Sanitizer.
- You should prevent from CSRF or XSRF attacks.

Angular defines the following security -
HTML is used when interpreting a value as HTML. For ex-

```
<div [innerHTML]="UNTRUSTED"></div>
OR
<input value="UNTRUSTED">
```

(i) Style is used when binding CSS into the style property. For ex-

```
<div [style]="height:UNTRUSTED"></div>
```

(ii) URL is used for URL properties. For ex-

```
<a [href]="UNTRUSTED-URL"></a>
OR
<script [src]="UNTRUSTED-URL"></script>
OR
<iframe src="UNTRUSTED-URL" />
```

(iii) Resource URL is a URL that will be loaded and executed. For Ex-

```
<script>var value = 'UNTRUSTED'</script>
<p class="e2e-inner-html-interpolated">{{html3nippet}}</p>
<p class="e2e-inner-html-bound" [innerHTML]="htmlSnippet"></P>
```

Q235. How does Angular handle with XSS or CSRF? How Angular prevents this attack?

Angular applications must follow the same security principles as regular web applications follows:

- You should avoid direct use of the DOM APIs.
- You should enable Content Security Policy (CSP) and configure your web server to return appropriate CSP HTTP headers.
- You should use the offline template compiler.
- You should use Server-Side XSS protection.
- You should use DOM Sanitizer.
- You should prevent from CSRF or XSRF attacks.

Q236. What is a Cookie?

A cookie is a small piece of data sent from the website and stored on the user machine through user web browsers while browsing.

OR
Cookies are small packages of information that are normally stored in the browsers and websites tend to use cookies for various things.

Q237. How to install a cookie in Angular?

For Installing cookie in angular following command is used

```
npm install ngx-cookie-service --save
```

If you do not want to install this through npm, you can run npm run compile and use the ∗.d.ts and ∗.js files in the dist-lib folder
After successfully installation, add the cookie service in the Angular module i.e. app.module.ts

app.module.ts

```
src > app > app.module.ts > ...
1   import { BrowserModule } from '@angular/platform-browser';
2   import { NgModule } from '@angular/core';
3
4   import { AppRoutingModule } from './app-routing.module';
5   import { AppComponent } from './app.component';
6   import { CookieService } from 'ngx-cookie-service';
7   @NgModule({
8     declarations: [
9       AppComponent
10    ],
11    imports: [
12      BrowserModule,
13      AppRoutingModule
14    ],
15    providers: [CookieService],
16    bootstrap: [AppComponent] //bootstrapped entry component
17  })
18  export class AppModule { }
19
```

Then, import and inject it into the component:

app.component.ts

```
src > app > app.component.ts > ...
2   import { CookieService } from 'ngx-cookie-service';
3
4   @Component({
5     selector: 'app-root',
6     templateUrl: './app.component.html',
7     styleUrls: ['./app.component.css']
8   })
9   export class AppComponent implements OnInit {
10
11    constructor(private _cookieService: CookieService) {
12
13    }
14
15    ngOnInit() {
16      this._cookieService.set("Name", "Ajeet Kumar Singh");
17      console.log(this._cookieService.get("Name"));
18    }
19
20  }
```

Q238. What are the cookies methods?

The concept of Angular cookies is similar to the Angular 1.x, but Angular only adds an additional method to delete all cookies.
All cookie methods are following:

- Check: This method is used to check either the cookie is existing or not.
- Get: This method returns the value of given cookie name.
- GetAll: This method returns the value object of all the cookies.
- Set: This method is used to set the cookies with a name.
- Delete: This method is used to delete the cookie with the given name.
- deleteAll: This method is used to delete all the cookies.

Q239. What are the Cookies Limitations?

We can only store around 20 cookies per web server and no more than 4KB of information in each cookie and they can last indefinitely.

Token Based Authentication

The Token-based authentication has expanded in recent years due to RESTful Web APIs, SPA, etc.
The Token-based authentication is stateless.

Q240. What is Stateless?

Every transaction is executed as if it was done for the very first time and there is no previously stored information used for the current transaction.
Token Based Authentication steps are following-
A user enters their login credentials and the server verifies the entered credentials. Validation of the credentials entered either it is correct or not. If the credentials are correct, it returns a signed token.
This token is stored in local storage on the client side. We can also store in session storage or cookie.

Advantages of Token-Based Authentication are-

- Stateless.
- Scalable.
- Decoupled.
- JWT is placed in the browser's local storage.
- Protect Cross Domain and CORS.
- Store Data in the JWT.
- Protect XSS and XSRF Protection.

Q241. Where to Store Tokens?

It depends on the user, where user wants to store the JWT. The JWT is placed in local storage of user browser.

Q242. What is TypeScript?

- TypeScript is a strongly typed, object-oriented compiled programming language. This language was developed and maintained by Microsoft.
- It was designed by "Anders Hejlsberg" at Microsoft.

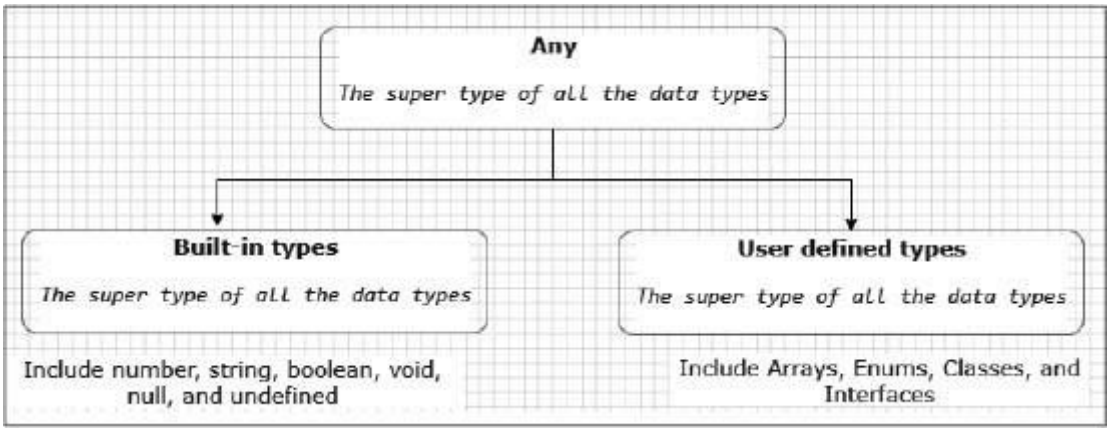
- It is the superset of JavaScript.
- The TypeScript is JavaScript and also has some additional features like static typing and class-based object-oriented programming, automatic assignment of constructor parameters, assigned null values and so on.
- The entire JavaScript program is valid for TypeScript as the entire TypeScript (.ts) file is converted to the JavaScript file after source code is compiled and this process is automatic.

Q243. Why you should use TypeScript? What are the Benefits of Using TypeScript?

- Typescript supports Object Oriented Programming.
- It adds static typing to JavaScript. Having static typing makes easier to develop and maintain complex applications.
- Angular2 uses typescript to simplify relations between various components and how the framework is built in general.
- Provide an optional type system for JavaScript.
- Provide planned features from future JavaScript editions to current JavaScript engines.
- Supports type definitions.

Q244. What are Types in TypeScript?

The Type represents the different types of values used in programming languages and checks the validity of the values provided before they are manipulated by programs.
The TypeScript provides data types as part of its optional type and also provides us some primitive types as well as a dynamic type “any”, and this “any” work as “dynamic”.



In TypeScript, we define a variable with a “type” and add the “variable name” with “colon” followed by the type name.

- Number - the “number” is a primitive number type in TypeScript. There is a no different type of float or double in TypeScript.
- Boolean - The “boolean” type represents true or false condition.
- String -The “string” represents a sequence of characters similar to C#.
- Null - The “null” is a special type which assigns a null value to the variable.
- Undefined - The “undefined” is also a special type of data and can be assigned to any variable.
- Any - This data type is the super-type of all types in TypeScript. It is also known as a

dynamic type and using “any” type is equivalent to opting out of type checking for a variable.

Q245. What Is Testing?

The testing is a tool and technique for a unit and integration testing Angular applications.

Q246. Why Test?

Tests are the best ways to prevent from software bugs and defects.

Q247. How to Setup Test in Angular Project?

Angular CLI install everything needed to test an Angular application.
This CLI command takes care of configuring Jasmine and karma configuration. Run this CLI command-

```
ng test
```

The test file extension must be “.spec.ts” so that tools can identify the test file.
You can also perform unit testing of your application using other testing libraries and test runners.

Q248. What is unit testing?

Ans. Unit testing is the great feature with which we can easily test the application feature which we have added in the component, services etc. In Angular, we have a spec.ts file for each component where we can add the unit testing.

Q249. What is the need of Karma and jasmine in Angular?

Ans. Karma is the task runner and Jasmine is the framework used for unit testing in Angular application.

Q250. How we can use Karma and Jasmine in Angular for testing?

Ans. When we create new application using Angular CLI, the Angular CLI will install the Karma and Jasmine for unit testing in the application which we can use using ng test command and execute the test cases.

Q251. What is the need of ng test?

Ans. ng test is the command which is used to run the unit test cases with the help of karma and Jasmine. When we run the ng test command in the Angular cli, it will perform all the application based test cases and display the result of success or failure.

Q252. What is the need of test.ts in Angular?

Ans. As you know, we use Karma in Angular application for unit testing. In the karma.config file, we have setting in which we add the test.ts file path that is used by the Angular application.

It's the entry point of the tests for the application.

Q253. What is test bed?

Ans. TestBed is the object and powerful unit testing tool which we can import from '@angular/core/testing' in the Angular application as.

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

Q254. We have extension .spec.ts in Angular application what is use if it?

Ans. Spec.ts denote that it is a test file in which we will add the code for unit testing of the component. In Angular, we have test.ts file where we have setting through which Angular application identifies the test file as spec.ts as shown below.

```
const context = require.context('./',true,/\.spec\.ts$/);
```

Thank You

You can register on www.sahosoft.com for Angular online class with Live Project Training...

You can join our online class and Live project Training

Website:

http://www.sahosoft.com/

All Plan Link:

http://www.sahosoft.com/SahosoftPaidPlans.aspx

Course syllabus:
http://www.sahosoft.com/AngularMasterClasses.aspx

Tutorial Website:

http://www.sahosofttutorials.com/

Facebook page:

https://www.facebook.com/Sahosoft-Solutions-247509462467379/