

[Home](#)[Interview Questions](#)[Java](#)[SQL](#)[Python](#)[JavaScript](#)[Angular](#)

Angular Interview Questions and Answers

1) What is Angular? / What do you know about Angular?

Angular is one of the most popular JavaScript frameworks developed and maintained by Google. It is an open-source front-end web framework based on TypeScript. It is most suited for developing enterprise web applications because its code is reusable and maintainable.

2) What are some powerful features integrated into Angular?

Angular integrates some powerful features like declarative templates, end to end tooling, dependency injection and various other best practices that smoothen the development path.

3) What is the main purpose of Angular?

The main purpose of using Angular is to create fast, dynamic and scalable web applications. We can create these applications very easily with Angular using components and directives.

Angular was started as a SPA (Single-Page-Application) framework, and now it supports dynamic content based on different users through dependency injection. It provides a platform for easy development of web-based applications and empowers the front end developers in curating cross-platform applications. YouTubeTV is the most popular example that uses Angular.



4) What is the difference between AngularJS and Angular?

Let's compare the features of AngularJS and Angular in a tabular form:

A list of differences between AngularJS and Angular-

| Feature | AngularJS | Angular |
|-----------------------------|--|---|
| Version | AngularJS was the very first version initially released in 2010. It was a browser-side JavaScript used within HTML code and created a revolution in web application development. It is popularly known as AngularJS. | The later Angular versions were a complete rewrite of AngularJS. For example, Angular 2 was initially released in 2016. There is nothing common between Angular2 and AngularJS except the core developer's team. After that, Angular 6, Angular 7, Angular 8, Angular 9, and Angular 10 were released that are very similar to each other. These later versions are known as Angular. |
| Architecture | AngularJS supports the MVC design model. | Angular uses components and directives. |
| Supported Language | The recommended and supported language of AngularJS is JavaScript. | The recommended and supported language of Angular is TypeScript. |
| Expression Syntax | In AngularJS, a specific ng directive is required for the image/property and an event. | Angular uses () for event binding and [] for property binding. |
| Mobile Support | AngularJS doesn't provide any mobile support. | Angular provides mobile support. |
| Dependency Injection | There is no concept of Dependency Injection in AngularJS. | Angular supports hierarchical Dependency Injection with uni-directional tree-based change detection. |
| Routing | In AngularJS, \$routeProvider.when() is used for routing configs. | In Angular, @RouteConfig{({})} is used for the routing config. |
| Structure | It is the first and basic version, so it is very easy to manage. | It has a very simplified structure that makes the development and maintenance of large applications very easy. |

| | | |
|----------------|--|--|
| Speed | It is slower because of its limited features. | It is faster than AngularJS because of its upgraded features. |
| Support | It doesn't provide support or new updates anymore. | It provides active support, and frequent new updates are made. |

5) What are the biggest advantages of using Angular?

Following is the list of the biggest advantages of using the Angular framework:

- Angular supports two-way data-binding.
- It follows MVC pattern architecture.
- It supports static templates and Angular template.
- It facilitates you to add a custom directive.
- It also supports RESTfull services.
- Validations are supported in Angular.
- Angular provides client and server communication.
- It provides support for dependency injection.
- It provides powerful features like Event Handlers, Animation, etc.

6) What do you understand by Angular expressions? How are Angular expressions different from JavaScript expressions?

Angular expressions are code snippets that are used to bind application data to HTML. Angular resolves the expressions, and the result is returned to where the expression is written. Angular expressions are usually written in double braces: {{ expression }} similar to JavaScript.

Syntax:

{{ expression }}



Following is a list of some differences between Angular expressions and JavaScript expressions:

1. The most crucial difference between Angular expressions and JavaScript expressions is that the Angular expressions allow us to write JavaScript in HTML. On the other hand, the JavaScript expressions don't allow.
2. The Angular expressions are evaluated against a local scope object. On the other hand, the JavaScript expressions are evaluated against the global window object. We can understand it better with an example. Suppose we have a component named test:

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-test',
  template: `
    <h4>{{message}}</h4>,
  styleUrls: ['./test.component.css']
})
export class TestComponent implements OnInit {
  message:string = ?Hello world?;
  constructor() {}
  ngOnInit() {
  }
}
```

In the above example, we can see that the Angular expression is used to display the message property. In the present template, we are using Angular expressions, so we cannot access a property outside its local scope (in this case, TestComponent). This proves that Angular expressions are always evaluated based on the scope object rather than the global object.

3. The Angular expressions can handle null and undefined, whereas JavaScript expressions cannot.

See the following JavaScript example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript Test</title>
</head>
<body>
  <div id="foo"><div>
</body>
<script>
  'use strict';
  let bar = {};
  document.getElementById('foo').innerHTML = bar.x;
</script>
</html>
```

After running the above code, you see undefined displayed on the screen. Although it's not ideal to leave any property undefined, the user does not need to see this.

Now see the following Angular example:

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-new',
  template: `
    <h4>{{message}}</h4>    `,
  styleUrls: ['./new.component.css']
})
export class NewComponent implements OnInit {
  message:object = {};
  constructor() {}
  ngOnInit() {
  }
}
```

In the above example, you will not see undefined being displayed on the screen.

4. In Angular expressions, we cannot use loops, conditionals, and exceptions. The difference which makes Angular expressions quite beneficial is the use of pipes. Angular uses pipes (known as filters in AngularJS) to format data before displaying it.

See this example:

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-new',
  template: `
```

```
    <h4>{{message | lowercase}}</h4>,
    styleUrls: ['./new.component.css']
  })
  export class NewComponent implements OnInit {
    message:string = "HELLO JAVATPOINT";
    constructor() {}
    ngOnInit() {
    }
  }
}
```

In the above example, we have used a predefined pipe called lowercase, which transforms all the letters in lowercase. If you run the above example, you will see the output displayed as "hello javatpoint".

On the other hand, JavaScript does not have the concept of pipes.

7) What are templates in Angular?

In Angular, templates contain Angular-specific elements and attributes. These are written with HTML and combined with information coming from the model and controller, which are further rendered to provide the user's dynamic view.

8) What is the difference between an Annotation and a Decorator in Angular?

In Angular, annotations are the "only" metadata set of the class using the Reflect Metadata library. They are used to create an "annotation" array. On the other hand, decorators are the design patterns used for separating decoration or modification of a class without actually altering the original source code.

9) Why was Angular introduced as a client-side framework?

Before the introduction of Angular, web developers used VanillaJS and jQuery to develop dynamic websites. Later, when the websites became more complex with added features and functionality, it was hard for them to maintain the code. Along with this, there were no provisions of data handling facilities across the views by jQuery. The need for a client-side framework like Angular was obvious that can make life easier for the developers by handling separation of concerns and dividing code into smaller bits of information (components).

Client-side frameworks like Angular facilitate developers to develop advanced web applications like Single-Page-Application. These applications can also be developed using VanillaJS, but the development process becomes slower by doing so.

10) How does an Angular application work?

Every Angular app contains a file named `angular.json`. This file contains all the configurations of the app. While building the app, the builder looks at this file to find the application's entry point. See the structure of the `angular.json` file:

```
"build": {
  "builder": "@angular-devkit/build-angular:browser",
  "options": {
    "outputPath": "dist/angular-starter",
    "index": "src/index.html",
    "main": "src/main.ts",
    "polyfills": "src/polyfills.ts",
    "tsConfig": "tsconfig.app.json",
    "aot": false,
    "assets": [
      "src/favicon.ico",
      "src/assets"
    ],
    "styles": [
      "./node_modules/@angular/material/prebuilt-themes/deeppurple-amber.css",
      "src/style.css"
    ]
  }
}
```

When the application enters the build section, the options object's main property defines the entry point of the application. The application's entry point is main.ts, which creates a browser environment for the application to run and calls a function called bootstrapModule, which bootstraps the application.

These two steps are performed in the following order inside the main.ts file:

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';  
platformBrowserDynamic().bootstrapModule(AppModule)
```

In the above line of code, AppModule is getting bootstrapped.

The AppModule is declared in the app.module.ts file. This module contains declarations of all the components.

Below is an example of app.module.ts file:

```
import { BrowserModule } from '@angular/platform-browser';  
import { NgModule } from '@angular/core';  
import { AppComponent } from './app.component';  
@NgModule({  
  declarations: [  
    AppComponent  
  ],  
  imports: [  
    BrowserModule  
  ],  
  providers: [],  
  entryComponents: [],  
  bootstrap: [AppComponent]
```

```
})  
export class AppModule { }
```

In the above file, you can see that AppComponent is getting bootstrapped. It is defined in app.component.ts file. This file interacts with the webpage and serves data to it.

Below is an example of app.component.ts file:

```
import { Component } from '@angular/core';  
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
})  
export class AppComponent {  
  title = 'angular';  
}
```

Each component is declared with three properties:

1. **Selector** - It is used to access the component.
2. **Template/TemplateURL** - It contains HTML of the component.
3. **StylesURL** - It contains component-specific stylesheets.

Now, Angular calls the index.html file. This file consequently calls the root component that is app-root. The root component is defined in app.component.ts.

See how the index.html file looks like:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Angular</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
  <app-root> </app-root>
</body>
</html>
```

The HTML template of the root component is displayed inside the `<app-root>` tags. This is the way how every angular application works.

11) Why is Angular preferred over other frameworks? / What are some advantages of Angular over other frameworks?

Due to the following features, Angular is preferred over other frameworks:

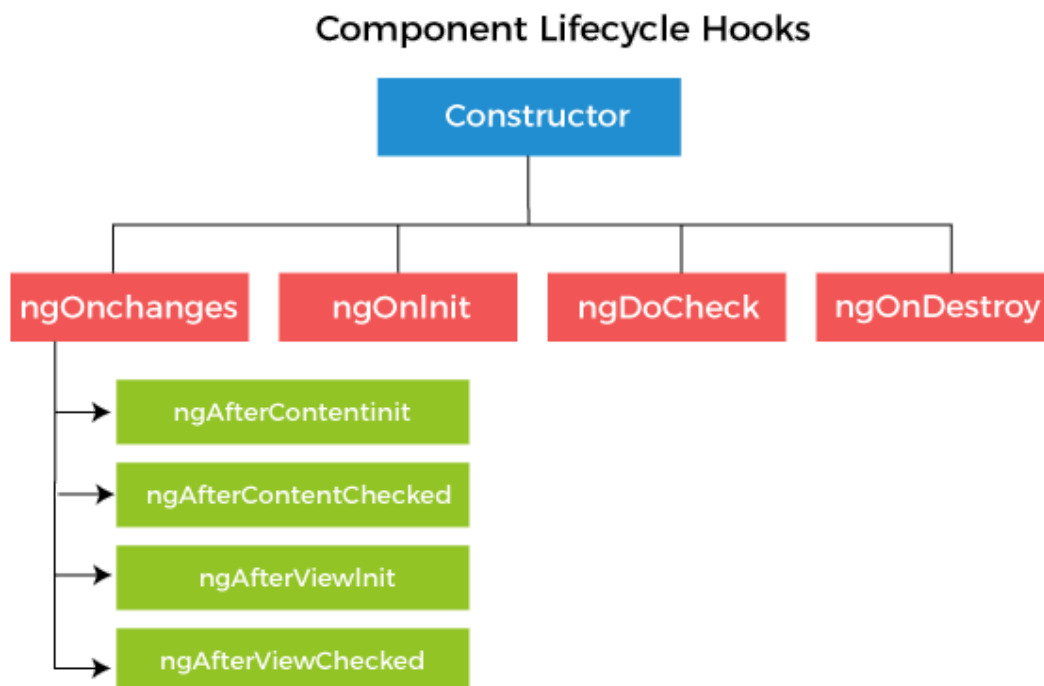
Extraordinary Built-in Features: Angular provides several out of the box built-in features like routing, state management, RxJS library, Dependency Injection, HTTP services, etc. That's why the developers do not need to look for the above-stated features separately.

Declarative UI: Angular has declarative UI. It uses HTML to render the UI of an application as it is a declarative language. It is much easier to use than JavaScript.

Long-term Google Support: Angular is developed and maintained by Google. Google has a long term plan to stick with Angular and provide support.

12) What are the different Lifecycle hooks of Angular? Explain them in short.

When the Angular components are created, they enter their lifecycle and remain when they are destroyed. Angular Lifecycle hooks are used to check the phases and trigger changes at specific phases during the entire duration.



ngOnChanges(): This method is called when one or more input properties of the component are changed. The hook receives a SimpleChanges object containing the previous and current values of the property.

ngOnInit(): This is the second lifecycle hook. It is called once, after the ngOnChanges hook. It is used to initialize the component and sets the input properties of the component.

ngDoCheck(): This hook is called after ngOnChanges and ngOnInit and is used to detect and act on changes that Angular cannot detect. In this hook, we can implement our change detection algorithm.

ngAfterContentInit(): This hook is called after the first ngDoCheck hook. This hook responds after the content gets projected inside the component.

ngAfterContentChecked(): This hook is called after ngAfterContentInit and every subsequent ngDoCheck. It responds after the projected content is checked.

ngAfterViewInit(): This hook is called after a component's view or initializing a child component's view.

ngAfterViewChecked(): This hook is called after ngAfterViewInit. It responds after the component's view or when the child component's view is checked.

ngOnDestroy(): This hook is called just before Angular destroys the component. This is used to clean up the code and detach event handlers.

In the above hooks we have described, the ngOnInit hook is the most often used hook. Let's see how to use the ngOnInit hook. If you have to process a lot of data during component creation, it's better to do it inside the ngOnInit hook rather than the constructor:

See the example:

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-test',
  templateUrl: './test.component.html',
  styleUrls: ['./test.component.css']
})
export class TestComponent implements OnInit {
  constructor() {}
  ngOnInit() {
    this.processData();
  }
  processData(){
    // Do something..
  }
}
```

In the above code, you can see that we have imported OnInit, but we have used the ngOnInit function. This is how we can use the rest of the hooks as well.

13) What is AOT in Angular?

In Angular, AOT stands for Ahead-Of-Time compiler. It is used to convert your Angular HTML and TypeScript code into efficient JavaScript code during the build phase before the browser downloads and runs that code. By compiling the application during the build process provides a faster rendering in the browser.

14) What is the reason for using the AOT compiler in Angular?

An Angular application is made of several components and their HTML templates. Because of these Angular components and templates, the browsers are not able to understand them directly. So, Angular applications require a compilation process before they run in a browser. That's why AOT compilers are required.

15) What are the biggest advantages of AOT in Angular?

Following are the advantages of using the AOT compiler in Angular:

The rendering is faster: When we use the AOT compiler, the browser gets a pre-compiled version of the application to download. Here, the browser loads executable code to render the application immediately, without waiting to compile the app first.

The Angular framework's download size is smaller: AOT facilitates you not to download the Angular compiler if the app is already compiled. The compiler is roughly half of Angular itself, so omitting it dramatically reduces the application payload.

Fewer asynchronous requests: The compiler is used to inline external HTML templates and CSS style sheets within the application JavaScript so, it eliminates separate AJAX requests for those source files.

Detect template errors earlier: While using the AOT compiler, developers can easily detect and report template binding errors during the build step before users can see them.

Better security: AOT provides better security because it compiles HTML templates and components into JavaScript files before they are served to the client. Because there are no templates to read and no risky client-side HTML or JavaScript evaluation, so the chances for injection attacks are very rare.

16) What is JIT in Angular?

In Angular, JIT stands for Just-in-Time compiler. The JIT compiler provides a dynamic translation or run-time compilation, which provides a way of executing computer code that involves compilation during the execution of a program at run time rather than before execution.

17) What is the main difference between JIT and AOT in Angular?

Following are the main differences between JIT and AOT compiler in Angular:

- Just-in-Time (JIT) compiler compiles our app in the browser at run-time while Ahead-of-Time (AOT) compiler is used to compile your app at build time on the server.
- The JIT compilation runs by default when you run the `ng build` (build only), or `ng serve` (build and serve locally) CLI commands. This is used for development. On the other hand, we have to include the `--aot` option with the `ng build` or `ng serve` command for AOT compilation.
- JIT and AOT are both two ways used to compile code in an Angular project. JIT compiler is used in development mode while AOT is used for production mode.
- JIT is easy to use. We can easily implement features and debug in JIT mode because here we have a map file while AOT does not. On the other hand, the biggest advantage of using AOT for production is that it reduces the bundle size for faster rendering.

18) What is the concept of scope hierarchy in Angular?

Angular provides the `$scope` objects into a hierarchy that is typically used by views. This is called the scope hierarchy in Angular. It has a root scope that can further contain one or several scopes called child scopes.

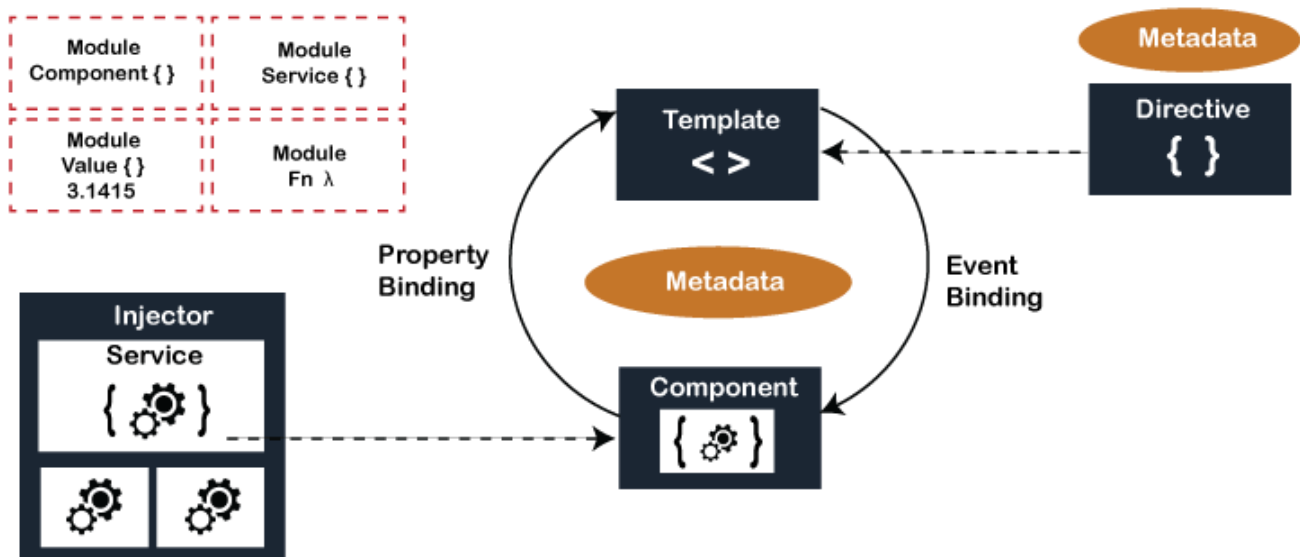
In a scope hierarchy, each view has its own `$scope`. Hence, the variables set by a view's view controller will remain hidden to other view controllers.

Following is the typical representation of a Scope Hierarchy:

```
Root $scope
$scope for Controller 1
$scope for Controller 2
...
..
.
$scope for Controller n
```


19) What are the main building blocks of an Angular application? Explain with the pictorial diagram of Angular architecture.

Following are the main building blocks of an Angular application. You can see them in the following picture:



20) What is the difference between Observables and Promises in Angular?

In Angular, as soon as we make a promise, the execution takes place, but this is not the case with observables because they are lazy. It means nothing happens until a subscription is made.

| Promise | Observable |
|--------------------------|--|
| It emits a single value. | It emits multiple values over a period of time. |
| Not Lazy | Lazy. An observable is not called until we subscribe to the observable. |
| We can not cancel it. | We can cancel it by using the unsubscribe() method. |
| | Observable provides operators like map, forEach, filter, reduce, retry, retryWhen etc. |

Let's understand it by an example:

```
const observable = rxjs.Observable.create(observer => {  
  console.log('This is what inside an observable');  
  observer.next('Hello JavaTpoint');
```

```
observer.complete();  
});  
console.log('Before subscribing an Observable');  
observable.subscribe((message)=> console.log(message));
```

When you run the above Observable, you can see the following messages displayed in the following order:

```
Before subscribing an Observable  
This is what inside an observable  
Hello JavaTpoint
```

Here, you can see that observables are lazy. Observable runs only when someone subscribes to them. That's why the message "Before subscribing an Observable" is displayed ahead of the message inside the observable.

Now see the example of a Promise:

```
const promise = new Promise((resolve, reject) => {  
  console.log('This is what written inside promise');  
  resolve('Hello JavaTpoint');  
});  
console.log('Before calling then method on Promise');  
greetingPoster.then(message => console.log(message));
```

When you run the above Promise, you will see the messages displayed in the following order:

```
This is what written inside Promise  
Before calling then method on Promise  
Hello JavaTpoint
```

Here, you can see that the message inside Promise is displayed first. This means that the Promise runs first, and then the method is called.

The next difference between them is that Promises are always asynchronous; even when the Promise is immediately resolved. On the other hand, an Observable can be both synchronous and asynchronous.

In the case of the above example, observable is synchronous. Let's see the case where an observable can be asynchronous:

```
const observable = rxjs.Observable.create(observer => {
  setTimeout(()=>{
    observer.next('Hello JavaTpoint');
    observer.complete();
  },3000)
});
console.log('Before calling subscribe on an Observable');
observable.subscribe((data)=> console.log(data));
console.log('After calling subscribe on an Observable');
```

When you run the above observable, you will see the messages in the following order:

```
Before calling subscribe on an Observable
After calling subscribe on an Observable
Hello JavaTpoint
```

22) What are directives in Angular?

A directive is a class in Angular that is declared with a `@Directive` decorator. Every directive has its own behavior, and you can import them into various components of an application.

23) What were the main reasons behind introducing client-side frameworks like Angular?

Before Angular was introduced, the web developers used VanillaJS and jQuery to develop dynamic websites, but the biggest drawback of these technologies is that as the logic of the website grew, the code became more and more complex to maintain. For websites and applications that use complex logic, developers had to put in extra effort to maintain the separation of concerns for the app. Also, jQuery did not provide facilities for data handling across views.

The client-side frameworks like Angular were introduced to overcome the above problems. It provides developers many benefits over VanillaJS and jQuery by providing a new feature called components for handling separation of concerns and dividing code into smaller bits of information.

Client-side frameworks such as Angular facilitate developers to develop advanced web applications like Single-Page-Applications. So, the main reasons behind introducing Angular were to create fast, dynamic, and scalable web applications easily.

Note: We can also develop dynamic websites and SPAs (Single Page Applications) using VanillaJS, and jQuery but by doing so, the development process becomes slower.

24) What is Angular CLI?

Angular CLI is a short form for Angular Command Line Interface. It is a command-line interface to scaffold and build angular apps using node.js style modules.

To use Angular CLI, we have to install it by using the following **npm** command:

```
npm install @angular/cli@latest
```

Following is a list of some useful commands which would be very helpful while creating angular projects:

- **Creating New Project:** ng new
- **Generating Components, Directives & Services:** ng generate/g
- **Running the Project:** ng serve

25) What is lazy loading in Angular?

Lazy loading is one of the most powerful and useful concepts of Angular Routing. It makes the web pages easy to download by downloading them in chunks instead of downloading everything in a big bundle. Lazy loading facilitates asynchronously loading the feature module for routing whenever required using the property `loadChildren`.

See the following example where we are going to load both Employee and Order feature modules lazily.

See the example:

```
const routes: Routes = [
  {
    path: 'employees',
    loadChildren: () => import('./employees/employees.module').then(module => module.Employees),
  },
  {
    path: 'orders',
    loadChildren: () => import('./orders/orders.module').then(module => module.OrdersModule),
  },
  {
    path: '',
    redirectTo: '',
    pathMatch: 'full'
  }
];
```

26) What is Angular Router?

Angular Router is a mechanism that facilitates users to navigate from one view to the next as users perform application tasks. It follows the concept model of browser's application navigation.

27) What do you understand by the router imports?

The Angular Router, representing a particular component view for a given URL, is not part of Angular Core. It is available in a library named `@angular/router`, and we have to import the required router components. This process is called router imports.

See the following example of how we can import them in the app module:

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

28) What do you understand by RouterOutlet and RouterLink?

A RouterOutlet is a directive from the router library that acts as a placeholder. It marks the spot in the template where the Router should display the components for that outlet. Router outlet is used as a component.

Syntax:

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

On the other hand, a RouterLink is a directive on the anchor tags that gives the router control over those elements. Since the navigation paths are fixed, you can assign string values to router-link directive as below,

Syntax:

```
<h1>Angular Router</h1>
<nav>
  <a routerLink="/todosList" >List of todos</a>
  <a routerLink="/completed" >Completed todos</a>
</nav>
<router-outlet> </router-outlet>
```

29) What are the different router events used in Angular Router?

During each navigation, the Router emits navigation events through the Router.events property. It allows us to track the lifecycle of the route.

Following is the list of different router events in sequence:

- NavigationStart
- RouteConfigLoadStart
- RouteConfigLoadEnd
- RoutesRecognized

- GuardsCheckStart
- ChildActivationStart
- ActivationStart
- GuardsCheckEnd
- ResolveStart
- ResolveEnd
- ActivationEnd
- ChildActivationEnd
- NavigationEnd
- NavigationCancel
- NavigationError

30) What do you understand by the RouterLinkActive?

The RouterLinkActive is a directive used to toggle CSS classes for active RouterLink bindings based on the current RouterState. i.e., the Router will add CSS classes when this link is active and remove them when the link is inactive.

For example, you can add them to RouterLinks as follows:

```
<h1>Angular Router</h1>
<nav>
  <a routerLink="/todosList" routerLinkActive="active">List of todos</a>
  <a routerLink="/completed" routerLinkActive="active">Completed todos</a>
</nav>
<router-outlet> </router-outlet>
```

31) What do you understand by the RouterState?

The RouterState is a tree of activated routes. Every node in this tree knows about the "consumed" URL segments, the extracted parameters, and the resolved data. We can access the current RouterState from anywhere in the application by using the Router service and the routerState property.

```
@Component({templateUrl:'template.html'})
class MyComponent {
```

```
constructor(router: Router) {  
  const state: RouterState = router.routerState;  
  const root: ActivatedRoute = state.root;  
  const child = root.firstChild;  
  const id: Observable<string> = child.params.map(p => p.id);  
  //...  
}  
}
```

32) What is HttpClient, and what are the advantages of it?

Most front-end applications use either XMLHttpRequest interface or the fetch() API to communicate with backend services over HTTP protocol. For the same purpose, Angular provides a simplified client HTTP API known as HttpClient. This is based on top of XMLHttpRequest interface. This HttpClient is available in the @angular/common/http package, which you can import in your root module as follows:

```
import { HttpClientModule } from '@angular/common/http';
```

Following are some of the crucial advantages of HttpClient:

- HttpClient contains testability features.
- It provides typed request and response objects.
- It can intercept requests and responses.
- It supports Observable APIs.
- HttpClient also supports streamlined error handling.

33) By default, Angular uses client-side rendering for its applications. Is it possible to make an Angular application to render on the server-side?

Yes, it is possible to make an Angular application to render on the server-side. Angular provides a technology called Angular Universal that can be used to render applications on the server-side.

The crucial advantages of using Angular Universal are as follows:

- Making an Angular application render on the server-side can provide a better user experience. By using this, first-time users can instantly see a view of the application. So, it can be used to provide better UI.

- It can lead to a better SEO for your application. The reason is that many search engines expect pages in plain HTML. So, Angular Universal can ensure that your content is available on every search engine, and it is good for better SEO.
- The server-side rendered applications load faster than normal pages. It is because the rendered pages are available to the browser sooner.

34) What is the best way to perform Error handling in Angular?

Error is when the request fails on the server or fails to reach the server due to network issues. In this condition, HttpClient returns an error object instead of a successful response. To resolve this issue, we must handle the component by passing the error object as a second callback to the subscribe() method.

See the following example to understand how we handle in the component:

```
fetchUser() {  
  this.userService.getProfile()  
    .subscribe(  
      (data: User) => this.userProfile = { ...data }, // success path  
      error => this.error = error // error path  
    );  
}
```

You can write an error message to give the user some meaningful feedback instead of displaying the raw error object returned from HttpClient.

35) What do you understand by Angular bootstrapping?

Angular bootstrapping is nothing but to allow developers to initialize or start the Angular application. Angular supports two types of bootstrapping:

- Manual bootstrapping
- Automatic bootstrapping

Manual bootstrapping: Manual bootstrapping provides more control to developers and facilitates them regarding how and when they need to initialize the Angular app. It is useful when professionals wish to perform other tasks and operations before Angular compiles the page.

Automatic bootstrapping: As the name specifies, automatic bootstrapping is started automatically to start the Angular app. The developers need to add the ng-app directive to the application's root if they want Angular to bootstrap the application automatically. Angular loads the associated module once it finds the ng-app directive and, further, compiles the DOM.

36) What is the digest cycle process in Angular?

The digest cycle process in Angular is the process that is used to monitor the watchlist to track changes in the watch variable value. There is a comparison between the present and the previous versions of the scope model values in each digest cycle.

37) What are the key differences between a Component and a Directive in Angular?

A Component is a directive that uses shadow DOM to create encapsulated visual behavior. Usually, components are used to create UI widgets by breaking up the application into smaller parts. In short, we can say that a component (@component) is a directive-with-a-template.

A list of the major differences between a Component and a Directive in Angular:

| Component | Directive |
|--|---|
| Components are generally used for creating UI widgets. | Directives are generally used for adding behavior to an existing DOM element. |
| We use @Component meta-data annotation attributes to register a component. | We use @Directive meta-data annotation attributes to register directives. |
| It is used to break up the application into smaller parts called components. | It is used to design re-usable components. |
| Only one component is allowed to be used per DOM element. | Multiple directives are allowed to be used per DOM element. |
| @View decorator or templateUrl/template is mandatory in a component. | A Directive doesn't use View. |
| A component is used to define pipes. | In a directive, it is not possible to define Pipes. |

38) What do you understand by Angular MVVM architecture?

The MVVM architecture or **Model-View-ViewModel** architecture is a software architectural pattern that provides a facility to developers to separate the development of the graphical user interface (the View) from the development of the business logic or back-end logic (the Model). By using this architecture, the view is not dependent on any specific model platform.

The Angular MVVM architecture consists of the following three parts:

- Model
- View
- ViewModel



Model: The Model consists of the structure of an entity and specifies the approach. In simple words, we can say that the model contains data of an object.

View: The View is the visual layer of the application. It specifies the structure, layout, and appearance of what a user sees on the screen. It displays the data inside the Model, represents the model, and receives the user's interaction with the view in the form of mouse clicks, keyboard input, screen tap gestures, etc., and forwards these to the ViewModel via the data binding properties. In Angular terms, the View contains the HTML template of a component.

ViewModel: The ViewModel is an abstract layer of the application. It is used to handle the logic of the application. It also manages the data of a model and displays it in the view. View and ViewModel are connected with two-way data-binding. If you make any changes in the view, the ViewModel takes a note and changes the appropriate data inside the model.

39) What is the purpose of AsyncPipe in Angular?

The AsyncPipe is used to subscribe to an observable or promise and return the latest value it has emitted. When a new value is emitted, the pipe marks the component that has been checked for changes.

See the following example where a time observable continuously updates the view for every 2 seconds with the current time.

Example:

```
@Component({
  selector: 'async-observable-pipe',
  template: `<div> <code>observable|async</code>:
    Time: {{ time | async }}</div>`
})
export class AsyncObservablePipeComponent {
  time = new Observable(observer =>
    setInterval(() => observer.next(new Date().toString()), 2000)
  );
}
```

40) What do you understand by services in Angular?

In Angular, services are singleton objects that get instantiated only once during the lifetime of an application. An Angular service contains methods that are used to maintain the data throughout the life of an application. Angular services are used to organize as well as share business logic, models, or data and functions with various components of an Angular application.

Angular services offer some functions that can be invoked from an Angular component, such as a controller or directive.

41) What is the key difference between a constructor and ngOnInit?

Constructor is a default method in TypeScript classes that are normally used for the initialization purpose. On the other hand, the ngOnInit is specifically an Angular method and is used to define Angular bindings. Even though constructors are getting called first, it is always preferred to move all of your Angular bindings to the ngOnInit method.

See the following example how we can use ngOnInit by implementing OnInit interface as follows:

```
export class App implements OnInit{
  constructor(){
    //called first time before the ngOnInit()
  }
}
```

```
ngOnInit(){  
    //called after the constructor and called after the first ngOnChanges()  
}  
}
```

42) What do you understand by observable and observer in Angular?

Observable: An observable is a unique object just like a promise that is used to manage async code. Observables are not part of the JavaScript language so the developers have to rely on a popular Observable library called RxJS. The observables are created using the new keyword.

See a simple example of observable to understand it better:

```
import { Observable } from 'rxjs';  
const observable = new Observable(observer => {  
    setTimeout(() => {  
        observer.next('This is a message from Observable!');  
    }, 1000);  
});
```

Observer: Any object that has to be notified when the state of another object changes is called an observer. An observer is an interface for push-based notifications delivered by an Observable.

See the structure of an observer:

```
interface Observer<T> {  
    closed?: boolean;  
    next: (value: T) => void;  
    error: (err: any) => void;  
    complete: () => void;  
}
```

The handler that implements the observer interface for receiving observable notifications is passed as a parameter for observable as follows:

```
myObservable.subscribe(myObserver);
```

Note: If you don't use a handler for a notification type, the observer ignores notifications of that type.

43) How do you categorize data binding types in Angular?

In Angular, we can categorize data binding types in three categories distinguished by the direction of data flow. These data binding categories are:

- From the source-to-view
- From view-to-source
- View-to-source-to-view

Let's see their possible binding syntax:

| Data direction | Syntax | Type |
|---|--|--|
| From the source-to-view(One-way data binding) | 1. {{expression}} 2. [target]="expression" 3. bind-target="expression" | Interpolation, Property, Attribute, Class, Style |
| From view-to-source(One-way data binding) | 1. (target)="statement" 2. on-target="statement" | Event |
| View-to-source-to-view(Two-way data binding) | 1. [(target))="expression" 2. bindon-target="expression" | Two-way data binding |

44) What is multicasting in Angular?

Multicasting or Multi-casting is the practice of broadcasting to a list of multiple subscribers in a single execution.

Let's take a simple example to demonstrate the multi-casting feature:

```
var source = Rx.Observable.from([1, 2, 3]);  
var subject = new Rx.Subject();  
var multicasted = source.multicast(subject);  
// These are, under the hood, `subject.subscribe({...})`:
```

```
multicasted.subscribe({
  next: (v) => console.log('observerA: ' + v)
});
multicasted.subscribe({
  next: (v) => console.log('observerB: ' + v)
});
```

45) What do you understand by Angular Material?

Angular Material is a UI component library that is used by professionals to develop consistent, attractive, and completely functional websites, web pages, and web applications. It follows the modern principles of web designing, such as graceful degradation and browser probability, and is capable of doing a lot of fascinating things in website and application development.

46) What is lazy loading in Angular? Why is it used?

In Angular, the by default tendency of NgModules is eagerly loaded. It means that as soon as the app loads, all the NgModules are loaded, whether or not they are immediately necessary. That's why lazy loading is required. Lazy loading is mandatory for large apps with lots of routes. This design pattern makes the app load NgModules when they are only required. Lazy loading helps keep initial bundle sizes smaller, which in turn helps decrease load times.

47) What is the use of Angular filters? What are its distinct types?

Filters are an essential part of Angular that helps in formatting the expression value to show it to the users. We can easily add filters to services, directives, templates, or controllers. We can also create personalized filters as per requirements. These filters allow us to organize the data in such a way that only the data that meets the respective criteria are displayed. Filters are placed after the pipe symbol (|) while used in expressions.

A list of various types of filters used in Angular:

- **currency:** It is used to convert numbers to the currency format.
- **filter:** It is used to select a subset containing items from the given array.
- **date:** It is used to convert a date into a necessary format.
- **lowercase:** It is used to convert the given string into lowercase.
- **uppercase:** It is used to convert the given string into uppercase.

- **orderBy:** It is used to arrange an array by the given expression.
- **json:** It is used to format any object into a JSON string.
- **number:** It is used to convert a numeric value into a string.
- **limitTo:** It is used to restrict the limit of a given string or array to a particular number of elements or strings.

48) When do we use a directive in Angular?

If you create an Angular application where multiple components need to have similar functionalities, you have to do it by adding this functionality individually to every component. This is not a very easy task. Directives are used to cope up with this situation. Here, we can create a directive with the required functionality and then import the directive to components that require this functionality.

49) What are the different types of directives in Angular?

There are mainly three types of directives in Angular:

Component Directives: The component directives are used to form the main class in directives. To declare these directives, we have to use the @Component decorator instead of @Directive decorator. These directives have a view, a stylesheet and a selector property.

Structural directives: These directives are generally used to manipulate DOM elements. The structural directive has a ' * ' sign before them. We can apply these directives to any DOM element.

Following are some example of built-in structural directives:

***ngIf Structural Directive:** *ngIf is used to check a Boolean value and if it's truthy, the div element will be displayed.

```
<div *ngIf="isReady" class="display_name">
  {{name}}
</div>
```

***ngFor Structural Directive:** *ngFor is used to iterate over a list and display each item of the list.

```
<div class="details" *ngFor="let x of details" >
  <p>{{x.name}}</p>
  <p> {{x.address}}</p>
  <p>{{x.age}}</p>
```


</div>

Attribute Directives: The attribute directives are used to change the look and behavior of a DOM element. Let's create an attribute directive to understand it well:

This is how we can create a custom directive:

Go to the command terminal, navigate to the directory of the angular app and type the following command to generate a directive:

```
ng g directive yellowBackground
```

This will generate the following directive. Manipulate the directive to look like this:

```
import { Directive, ElementRef } from '@angular/core';
@Directive({
  selector: '[appYellowBackground]'
})
export class YellowBackgroundDirective {
  constructor(el:ElementRef) {
    el.nativeElement.style.backgroundColor = "yellow";
  }
}
```

Now, you can easily apply the above directive to any DOM element:

```
<p appYellowBackground>Hello JavaTpoint</p>
```

50) What are string interpolation and property binding in Angular?

String interpolation and property binding are parts of data-binding in Angular. Data-binding is a feature of Angular, which is used to provide a way to communicate between the component (Model) and its view (HTML template). There are two ways of data-binding, one-way data binding and two-way data binding. In Angular, data from the component can be inserted inside the HTML template. Any changes in the component will directly reflect inside the HTML template in one-way binding, but vice-versa is not possible. On the other hand, it is possible in two-way binding.

String interpolation and property binding both are examples of one-way data binding. They allow only one-way data binding.

String Interpolation: String interpolation uses the double curly braces `{{ }}` to display data from the component. Angular automatically runs the expression written inside the curly braces. For example, `{{ 5+5 }}` will be evaluated by Angular, and the output will be 10. This output will be displayed inside the HTML template.

Property Binding: Property binding is used to bind the DOM properties of an HTML element to a component's property. In property binding, we use the square brackets `[]` syntax.

51) Is it possible to make an angular application to render on the server-side?

Yes, we can make an angular application to render on the server-side. Angular provides a technology Angular Universal that makes you able to render applications on the server-side.

Following are the benefits of using Angular Universal:

Better User Experience: It enables users to see the view of the application instantly.

Better SEO: Angular Universal ensures that the content is available on every search engine leading to better SEO.

Load Faster: Angular Universal ensures that the render pages available to the browsers sooner to make the loading faster server-side application loads faster.

52) What is Dependency Injection in Angular?

Dependency injection is an application design pattern that is implemented by Angular. It is used to form the core concepts of Angular. Dependencies are services in Angular which have some specific functionality. Various components and directives in an application can need these functionalities of the service. Angular provides a smooth mechanism by which these dependencies are injected into components and directives.

53) Can you demonstrate navigation between different routes in an Angular application?

You can demonstrate the navigation between different routes in an Angular app in the following way. See the following code to demonstrate navigation in an Angular app named "My First App."

```
import from "@angular/router";
```

```

.
.
.
@Component({
  selector: 'app-header',
  template: `
<nav class="navbar navbar-light bg-faded">
  <a class="navbar-brand" (click)="goHome()">My First App</a>
  <ul class="nav navbar-nav">
    <li class="nav-item">
      <a class="nav-link" (click)="goHome()">Home</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" (click)="goSearch()">Search</a>
    </li>
  </ul>
</nav>
  `
})
class HeaderComponent {
  constructor(private router: Router) {}
  goHome() {
    this.router.navigate([""]);
  }
  goSearch() {
    this.router.navigate(['search']);
  }
}

```

54) What is the difference between Angular and Backbone.js?

Following are the various notable differences between Angular and Backbone.js:

| Comparison Parameter | Angular | Backbone.js |
|-------------------------|---------|-------------|
|-------------------------|---------|-------------|

| | | |
|-------------------------|--|--|
| Architecture | Angular works on the MVC architecture and makes use of two-way data binding for driving application activity. | Backbone.js makes use of the MVP architecture and doesn't provide any data binding process. |
| Type | Angular is an open-source JavaScript-based front-end web application framework that extends HTML with new attributes. | Backbone.js is a lightweight JavaScript library that uses a RESTful JSON interface and MVP framework. |
| Data Binding | Angular is a little bit complex because it uses a two-way data binding process. | On the other hand, Backbone.js has a simple API because it doesn't have any data binding process. |
| DOM | Angular's main focus is on valid HTML and dynamic elements that imitate the underlying data for rebuilding the DOM as per the specified rules and then work on the updated data records. | Backbone.js follows the direct DOM manipulation approach for representing data and application architecture changes. |
| Performance | Because of its two-way data binding functionality, Angular provides powerful performance for both small and large projects. | Backbone.js is quite a significant upper hand in performance over Angular in small data sets or small web pages. It is not recommended for larger web pages or large data sets due to the absence of any data binding process. |
| Templating | Angular supports templating via dynamic HTML attributes. You can add them to the document to develop an easy to understand application at a functional level. | Backbone.js uses Underscore.js templates that aren't fully-featured as Angular templates. |
| Testing Approach | The testing approach is lengthy for Angular because it is preferred for building large applications. It uses unit testing. | The testing approach is completely different for Backbone.js because it is ideal for developing smaller webpages or applications. |













| | | |
|--------------------------|--|---|
| Community Support | The angular framework is developed and maintained by Google, so it receives great community support. Here, extensive documentation is available. | Backbone.js also receives a good level of community support, but it only documents on Underscore.js templates, not much else. |
|--------------------------|--|---|

You may also like:

- [Java Interview Questions](#)
- [SQL Interview Questions](#)
- [Python Interview Questions](#)
- [JavaScript Interview Questions](#)
- [Angular Interview Questions](#)
- [Selenium Interview Questions](#)
- [Spring Boot Interview Questions](#)
- [HR Interview Questions](#)
- [C Programming Interview Questions](#)
- [C++ Interview Questions](#)
- [Data Structure Interview Questions](#)
- [DBMS Interview Questions](#)
- [HTML Interview Questions](#)
- [IAS Interview Questions](#)
- [Manual Testing Interview Questions](#)


- OOPs Interview Questions
- .Net Interview Questions
- C# Interview Questions
- ReactJS Interview Questions
- Networking Interview Questions
- PHP Interview Questions
- CSS Interview Questions
- Node.js Interview Questions
- Spring Interview Questions
- Hibernate Interview Questions
- AWS Interview Questions
- Accounting Interview Questions


Learn Latest Tutorials

| | | | |
|---|---|---|---|
|  Splunk tutorial Splunk |  SPSS tutorial SPSS |  Swagger tutorial Swagger |  T-SQL tutorial Transact-SQL |
|  Tumblr tutorial Tumblr |  React tutorial ReactJS |  Regex tutorial Regex |  Reinforcement learning tutorial Reinforcement Learning |
|  R Programming tutorial R Programming |  RxJS tutorial RxJS |  React Native tutorial React Native |  Python Design Patterns Python Design Patterns |
|  Python Pillow tutorial Python Pillow |  Python Turtle tutorial Python Turtle |  Keras tutorial Keras | |

Preparation

 Aptitude
Aptitude

 Logical
Reasoning
Reasoning

 Verbal Ability
Verbal Ability


 Interview
Questions
Interview Questions

 Company
Interview
Questions
Company Questions


Trending Technologies

 Artificial
Intelligence
Tutorial
Artificial
Intelligence


 AWS Tutorial
AWS

 Selenium
tutorial
Selenium

 Cloud
Computing
tutorial
Cloud Computing

 Hadoop tutorial
Hadoop


 ReactJS
Tutorial
ReactJS

 Data Science
Tutorial
Data Science

 Angular 7
Tutorial
Angular 7

 Blockchain
Tutorial
Blockchain


 Git Tutorial
Git


 Machine
Learning Tutorial
Machine Learning


 DevOps
Tutorial
DevOps


B.Tech / MCA

 DBMS tutorial
DBMS

 Data Structures
tutorial
Data Structures


 DAA tutorial
DAA

 Operating
System tutorial
Operating System

 Computer
Network tutorial
Computer Network

 Compiler
Design tutorial
Compiler Design

 Computer
Organization and
Architecture

 Discrete
Mathematics
Tutorial



Ethical Hacking
Tutorial

Ethical Hacking



Computer
Graphics Tutorial

Computer Graphics



Computer
Organization
Software
Engineering
Tutorial

Software
Engineering



Discrete
Mathematics
html tutorial
Web Technology



Cyber Security
tutorial

Cyber Security



Automata
Tutorial

Automata



C Language
tutorial

C Programming



C++ tutorial
C++



Java tutorial
Java



.Net
Framework
tutorial
.Net



Python tutorial
Python



List of
Programs
Programs



Control
Systems tutorial
Control System



Data Mining
Tutorial
Data Mining



Data
Warehouse
Tutorial
Data Warehouse