**Why were client-side frameworks like Angular introduced?**

Back in the day, web developers used VanillaJS and jQuery to develop dynamic websites but, as the logic of one's website grew, the code became more and more tedious to maintain. For applications that use complex logic, developers had to put in extra effort to maintain separation of concerns for the app. Also, jQuery did not provide facilities for data handling across views. For tackling the above problems, client-side frameworks like Angular came into the picture, which made life easier for the developers by handling separation of concerns and dividing code into smaller bits of information (In the case of Angular, called Components). Client-side frameworks allow one to develop advanced web applications like Single-Page-Application. Not that we cannot develop SPAs using VanillaJS, but by doing so, the development process becomes slower.

**What is SPA** **(Single Page Application) in Angular? Contrast SPA technology with traditional web technology?**

In the SPA technology, only a single page, which is index.HTML, is maintained although the URL keeps on changing. Unlike traditional web technology, SPA technology is faster and easy to develop as well.

In conventional web technology, as soon as a client requests a webpage, the server sends the resource. However, when again the client requests for another page, the server responds again with sending the requested resource. The problem with this technology is that it requires a lot of time.

#### ****Please explain the various features of Angular.****

* **Accessibility Applications:** Angular allows creating accessible applications using ARIA-enabled components, built-in a11y test infrastructure, and developer guides.
* **Angular CLI:** Angular provides support for command-line interface tools. These tools can be used for adding components, testing, instant deploying, etc.
* **Animation Support**: Angular’s intuitive API allows the creation of high-performance, complex animation timelines with very little code.
* **Cross-Platform App Development**: Angular can be used for building an efficient and powerful desktop, native, and progressive web apps. Angular provides support for building native mobile applications using Cordova, Ionic, or NativeScript. Angular allows creating high performance, offline, and zero-step installation progressive web apps using modern web platform capabilities. The popular JS framework can also be used for building desktop apps for Linux, macOS, and Windows.
* **Code Generation**: Angular is able to convert templates into highly-optimized code for modern JavaScript virtual machines.
* **Code Splitting**: With the new Component Router, Angular apps load quickly. The Component Router offers automatic code-splitting so that only the code required to render the view that is requested by a user is loaded.
* **Synergy with Popular Code Editors and IDEs**: Angular offers code completion, instant errors, etc. with popular source code editors and IDEs.
* **Templates**: Allows creating UI views with a simple and powerful template syntax.
* **Testing**: Angular lets you carry out frequent unit tests using Karma. The Protractor allows running faster scenario tests in a stable way.

#### ****Can you give us an overview of Angular architecture?****



**How does an Angular application work?**

Every Angular app consists of a file named **angular.json**. This file will contain all the configurations of the app. While building the app, the builder looks at this file to find the entry point of the application. Following is an image 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"

]

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Inside the build section, the main property of the options object defines the entry point of the application which in this case is **main.ts**. The main.ts file creates a browser environment for the application to run, and, along with this, it also 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 { }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

As one can see in the above file, **AppComponent** is getting bootstrapped. This component 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** - used for accessing the component  
2. **Template/TemplateURL** - contains HTML of the component  
3. **StylesURL** - contains component-specific stylesheets  
  
After this, 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**. This is how the index.html file looks:

<!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 how every angular application works.

**What are some of the advantages of Angular over other frameworks?**

* **Features that are provided out of the box -** Angular provides a number of built-in features like,routing, state management, rxjs library and http servicesstraight out of the box. This means that one does not need tolook for the above stated features separately. They are allprovided with angular.
* **Declarative UI -**Angular uses HTML to render the UI of an application. HTML isa declarative language and is much easier to use than JavaScript.
* **Long-term Google support -**Google announced Long-term support for Angular. This means that Google plans to stick with Angular and further scale up its ecosystem.

**List out differences between AngularJS and Angular**

|  |  |  |
| --- | --- | --- |
| **Parameters** | **AngularJS** | **Angular** |
| **Architecture** | MVC or  Model-View-Controller architecture facilitates the AngularJS framework, where the Model contains the business logic and Controllers processes information, lastly, View shows the information present in the Model. | Angular replaces controllers with Components that are directives with a predefined template. |
| **Language** | AngularJS uses JavaScript language, which is a dynamically typed language. | Angular uses TypeScript language, a statically typed language, and a superset of JavaScript. Angular provides better performance while developing larger applications. |
| **Mobile Support** | Does not support mobile support. | Supported by all popular mobile browsers. |
| **Structure** | The process of maintaining code becomes tedious in the case of larger applications. | It is easier to maintain code for larger applications as it provides a better structure. |
| **Expression Syntax** | A developer needs to remember the correct ng-directive for binding an event or a property. | Property binding is done using "[ ]" attribute and event binding is done using "( )" attribute. |
| **Routing** | AngularJS uses $routeprovider.when() | Angular uses @RouteConfig{(…)} |
| **Speed** | The development effort and time are reduced significantly because of the two-way data binding | Angular is faster due to upgraded features. |
| **Dependency** **Injection** | AngularJS doesn’t support DI. | Angular supports a hierarchical Dependency Injection with unidirectional tree-based change detection. |
| **Support** | No official support or updates are available for AngularJS. | Angular has active support with updates rolling out every now and then. |

**What are the building blocks of Angular?**

1. **Components –** A component controls one or more views. Each view is some specific section of the screen. Every Angular application has at least one component, known as the root component. It is bootstrapped inside the main module, known as the root module. A component contains application logic defined inside a class. This class is responsible for interacting with the view via an API of properties and methods.
2. **Data Binding –** The mechanism by which parts of a template coordinates with parts of a component is known as data binding. In order to let Angular know how to connect both sides (template and its component), the binding markup is added to the template HTML.
3. **Dependency Injection (DI) –** Angular makes use of DI to provide required dependencies to new components. Typically, dependencies required by a component are services. A component’s constructor parameters tell Angular about the services that a component requires. So, a dependency injection offers a way to supply fully-formed dependencies required by a new instance of a class.
4. **Directives –** The templates used by Angular are dynamic in nature. Directives are responsible for instructing Angular about how to transform the DOM when rendering a template. Actually, components are directives with a template. Other [types of directives](https://angular.io/guide/attribute-directives) are attribute and structural directives.
5. **Metadata –** In order to let Angular know how to process a class, metadata is attached to the class. For doing so decorators are used.
6. **Modules –** Also known as NgModules, a module is an organized block of code with a specific set of capabilities. It has a specific application domain or a workflow. Like components, any Angular application has at least one module. This is known as the root module. Typically, an Angular application has several modules.
7. **Routing –** An Angular router is responsible for interpreting a browser URL as an instruction to navigate to a client-generated view. The router is bound to links on a page to tell Angular to navigate the application view when a user clicks on it.
8. **Services –** A very broad category, a service can be anything ranging from a value and function to a feature that is required by an Angular app. Technically, a service is a class with a well-defined purpose.
9. **Template –** Written with HTML, templates in Angular contains Angular-specific attributes and elements. Combined with information coming from the controller and model, templates are then further rendered to cater the user with the dynamic view.

**Explain Components, Modules and Services in Angular**

For better understanding, I would like you to create an Angular application by running the following inside the command terminal: **ng new angularApp**

The above command will create an angular application in the directory.  
Next, let's move on to understand Components, Modules, and Services.  
  
**Components**  
In Angular, components are the basic building blocks, which control a part of the UI for any application.  
A component is defined using the **@Component** decorator. Every component consists of three parts, the template which loads the view for the component, a stylesheet which defines the look and feel for the component, and a class that contains the business logic for the component.  
For creating a component, inside the command terminal, navigate to the directory of the application created, and run the following command:

**ng generate component test**

Or

**ng g c test**

One can see the generated component inside src/app/test folder. Our component is defined with **@Component** decorator. The component will be defined inside test.component.ts and this is how it looks:

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

@Component({

selector: 'app-test',

templateUrl: './test.component.html',

styleUrls: ['./test.component.css']

})

export lass TestComponent implements OnInit {

constructor() {}

ngOnInit() {

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Modules**  
A module is a place where we can group components, directives, services, and pipes. Module decides whether the components, directives, etc can be used by other modules, by exporting or hiding these elements. Every module is defined with a @NgModule decorator.  
By default, modules are of two types:

* Root Module
* Feature Module

Every application can have only one root module whereas, it can have one or more feature modules.  
A root module imports **BrowserModule**, whereas a feature module imports **CommonModule**. In the application that we created before, one can see that the root module is defined inside **app.module.ts** and this is how it looks:

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

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

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

import { TestComponent } from './test/text.component';

@NgModule({

declarations: [

AppComponent,

TestComponent

],

imports: [

BrowserModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

We can see in the above image that the component we created earlier is already imported in the declarations array. To create a feature module, run the following command:

**ng g m test-module**

The module is created inside the src/app/test-module/test-module.module.ts file:

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

import { CommonModule } from '@angular/common';

@NgModule({

declarations: [],

imports: [

CommonModule

]

})

export class TestModuleModule { }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

As one can see, **CommonModule** is imported since this is a feature module.

**Services** Services are objects which get instantiated only once during the lifetime of an application. The main objective of a service is to share data, functions with different components of an Angular application.  
A service is defined using a **@Injectable** decorator. A function defined inside a service can be invoked from any component or directive. To create a service, run the following command:

**ng g s test-service**

The service will be created inside src/app/test-service.service.ts:

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

@Injectable({

providedIn: 'root'

})

export class TestServiceService {

constructor() { }

}

Any method/function defined inside the TestServiceService class can be directly used inside any component by just importing the service.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**What are lifecycle hooks in Angular? Explain a few lifecycle hooks.**

Every component in Angular has a lifecycle, different phases it goes through from the time of creation to the time it's destroyed. Angular provides **hooks** to tap into these phases and trigger changes at specific phases in a lifecycle.

1. **ngOnChanges( )** This hook/method is called before **ngOnInit** and whenever one or more input properties of the component changes.  
   This method/hook receives a SimpleChanges object which contains the previous and current values of the property.
2. **ngOnInit( )** This hook gets called once, after the **ngOnChanges** hook.  
   It initializes the component and sets the input properties of the component.
3. **ngDoCheck( )** It gets called after **ngOnChanges** and **ngOnInit** and is used to detect and act on changes that cannot be detected by Angular.  
   We can implement our change detection algorithm in this hook. **ngAfterContentInit( )** It gets called after the first **ngDoCheck** hook. This hook responds after the content gets projected inside the component.
4. **ngAfterContentChecked( )** It gets called after **ngAfterContentInit** and every subsequent **ngDoCheck**. It responds after the projected content is checked.
5. **ngAfterViewInit( )** It responds after a component's view, or a child component's view is initialized.
6. **ngAfterViewChecked( )** It gets called after **ngAfterViewInit**, and it responds after the component's view, or the child component's view is checked.
7. **ngOnDestroy( )** It gets called just before Angular destroys the component. This hook can be used to clean up the code and detach event handlers.

Let’s understand how to use **ngOnInit** hook, since it’s the most oftenly used hook. If one has to process lot of data during component creation, it’s better to do it inside **ngOnInit** hook rather than the constructor:

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..

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

As you can see we have imported OnInit but we have used **ngOnInit** function. This principle should be used with the rest of the hooks as well.

**Explain string interpolation and property binding in Angular.**

String interpolation and property binding are parts of **data-binding** in Angular. Data-binding is a feature in angular, which provides a way to communicate between the component(Model) and its view(HTML template).Data-binding can be done in two ways, **one-way** binding and **two-way** binding.  
In Angular, data from the component can be inserted inside the HTML template. In one-way binding, any changes in the component will directly reflect inside the HTML template but, vice-versa is not possible. Whereas, it is possible in two-way binding. String interpolation and property binding allow only one-way data binding.  
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, {{ 2 + 2 }} will be evaluated by Angular and the output 4, will be displayed inside the HTML template. Using property binding, we can bind the DOM properties of an HTML element to a component's property. Property binding uses the square brackets **[ ]** syntax.

They can be categorized based on the direction of the data flow.

|  |  |  |
| --- | --- | --- |
| **Data flow Direction** | **Type** | **Description** |
| From source to view (one-way) | Interpolation - Attribute, style, class, property | Interpolates values calculated from application data into HTML |
| From lightview to the source (one-way) | Event | Enables applications to respond to users in the target environment |
| View-source-view (two-way) | Two-way | Changes in the application state automatically get reflected in the view and vice-versa. For this type of binding, ngModel directive is used. |

### How can you bind data to templates?

### •Property binding: Property binding in Angular helps you set values for properties of HTML elements or directives

### <img [src]="itemImageUrl">

### •Event binding: Event binding allows you to listen for and respond to user actions such as keystrokes, mouse movements, clicks, and touches.

### <button (click)="onSave()">Save</button>

### •Two-way binding: Two-way binding gives components in your application a way to share data. Use two-way binding binding to listen for events and update values simultaneously between parent and child components.

### <app-sizer [(size)]="fontSizePx"></app-sizer>

### What is the difference between ViewChild and ContentChild?

### ViewChild and ContentChild are used for component communication in Angular, for example, if a parent component wants access to one or multiple child components.

### •A ViewChild is any component, directive, or element which is part of a template.

### •A ContentChild is any component or element which is projected in the template.

### In Angular exist two different DOMs:

### •Content DOM which has only knowledge of the template provided by the component at hand or content injected via <ng-content>.

### Tell Us Some Differences Between Component and Directive

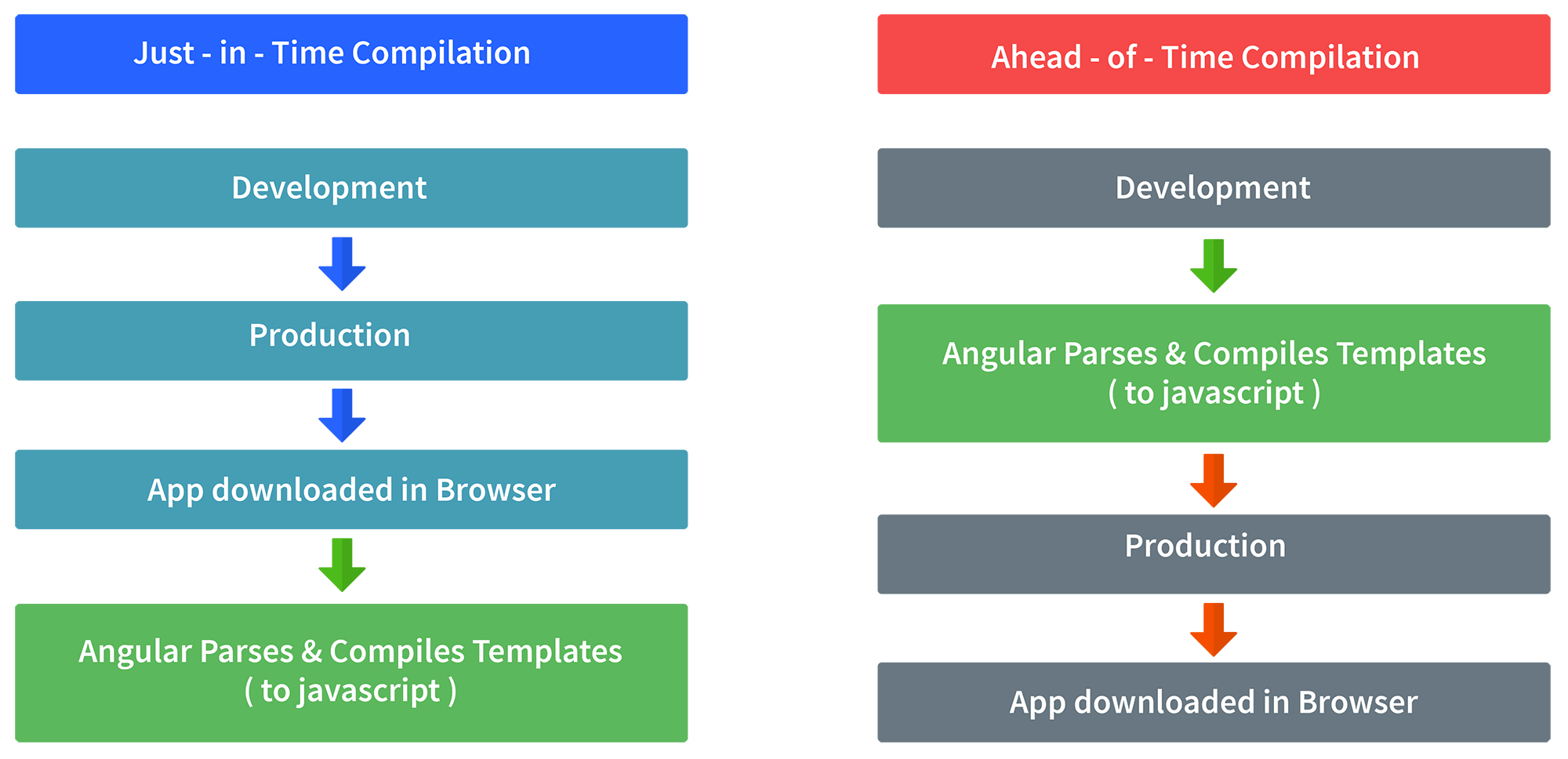
**Answer:** The component is a specific type of directive, that has a view.

|  |  |
| --- | --- |
| **Component** | **Directive** |
| To register component, Annotation used is @Component | @Directive is used to register a directive |
| The primary purpose of ingredients is to break the complex application into smaller, more manageable parts (components) | Purpose of the `directive is to create new custom components that are reusable |
| Each DOM element can have only one component | Any number of directives can be used in one DOM element |
| Component mandatorily requires @View decorator, template, or template URL to specify the view. | A directive has nothing to do with views |

**What is AOT compilation? What are the advantages of AOT?**

The Angular components and templates cannot be understood by the browser directly. Due to that Angular applications require a compilation process before they can run in a browser. For example, In AOT compilation, both Angular HTML and TypeScript code converted into efficient JavaScript code during the build phase before browser runs it. Angular provides two types of compilation:

* JIT(Just-in-Time) compilation
* AOT(Ahead-of-Time) compilation



In JIT compilation, the application compiles inside the browser during runtime.  
Whereas in the AOT compilation, the application compiles during the build time.  
The advantages of using AOT compilation are:

* Since the application compiles before running inside the browser, the browser loads the executable code and renders the application immediately, which leads to **faster rendering**.
* In AOT compilation, the compiler sends the external HTML and CSS files along with the application, eliminating separate AJAX requests for those source files, which leads to **fewer ajax requests**.
* Developers can detect and handle errors during the building phase, which helps in **minimizing errors**.
* The AOT compiler adds HTML and templates into the JS files before they run inside the browser. Due to this, there are no extra HTML files to be read, which provide **better security** to the application.

By default, angular builds and serves the application using JIT compiler:

**ng build  
ng serve**

For using AOT compiler following changes should be made:

**ng build --aot  
ng serve –aot**

### What Is the Primary Language Used in Angular?

Angular is based on TypeScript and HTML. HTML is used for the template, and TypeScript (a superset of JavaScript) is used for components.

**How are Angular expressions different from JavaScript expressions?**

|  |  |
| --- | --- |
| **Angular expressions** | **JavaScript expressions** |
| Conditions, exceptions, and loops (control statements) cannot be used | All the control statements can be used |
| Regular expressions cannot be used | Regex is widely used |
| Filters can be used within the expression itself so that data is formatted before being displayed | Such a concept doesn’t exist |
| Expressions are evaluated against a scope object | Expressions are evaluated against the global window |
| Expression evaluation forgives to undefined or null | If there are issues in evaluating an expression or property, JS generates ReferenceError or TypeError. |
| Functions cannot be declared | Any number of functions can be declared |
| New, comma, bitwise, void operators cannot be used. | These are possible |

The first and perhaps, the biggest difference is that Angular expressions allow us to write JavaScript in HTML which is not the case when it comes to JavaScript expressions. Next, Angular expressions are evaluated against a **local** scope object whereas JavaScript expressions against **global** window object. Let's understand that better with an example :

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() {

}

}

As one can see that Angular expression is used to display **message** property of a component. Since we are using Angular expressions, in the present template, we cannot access a property outside of its local scope, which in this case is **TestComponent**. This proves that Angular expressions are always evaluated based on **scope** object rather than the global object. Next difference is how Angular expressions handle **null** and **undefined**.

<!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>

If you run the above code, you will 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 consider 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() {

}

}

If you render the above component, you will **not** see undefined being displayed on the screen. Next, in Angular expressions one **cannot** use loops, conditionals and exceptions. The difference which makes Angular expressions quite beneficial is the use of **pipes**. Angular uses pipes(called filters in AngularJS), which can be used to format data before displaying it. Let’s see one predefined pipe in action:

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 WORLD";

constructor() { }

ngOnInit() {

}

}

In the above code we have used a predefined pipe called **lowercase**, which transforms all the letters in lowercase. Therefore, if you render the above component, you will see “hello world” being displayed. In contrast, JavaScript does not have the concept of **pipes**.

**How are observables different from promises?**

|  |  |
| --- | --- |
| Promise | Observable |
| Emits a single value | Emits multiple values over a period of time |
| Not Lazy | Lazy. An observable is not called until we subscribe to the observable |
| Cannot be cancelled | Can be cancelled by using the unsubscribe() method |
| It does not provide | Observable provides operators like map, forEach, filter, reduce, retry, retryWhen etc. |

Consider the following Observable:

const observable = rxjs.Observable.create(observer => {

console.log('Text inside an observable');

observer.next('Hello world!');

observer.complete();

});

console.log('Before subscribing an Observable');

observable.subscribe((message)=> console.log(message));

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

**Before subscribing an Observable  
Text inside an observable  
Hello world!**

As you can see, observables are lazy. Observable runs only when someone subscribes to them hence, the message “Before subscribing…” is displayed ahead of the message inside the observable. Now let’s consider a Promise:

const promise = new Promise((resolve, reject) => {

console.log('Text inside promise');

resolve('Hello world!');

});

console.log('Before calling then method on Promise');

greetingPoster.then(message => console.log(message));

Running the above promise, the messages will be displayed in the following order:

**Text inside promise  
Before calling then method on Promise  
Hello world!**

As you can see the message inside Promise is displayed first. This means that a promise runs before the **then** method is called. Therefore, promises are **eager**. The next difference is that Promises are always **asynchronous**. Even when the promise is immediately resolved. Whereas an Observable, can be both **synchronous** and **asynchronous**. The above example of an observable is the case to show that an observable is synchronous. Let’s see the case where an observable can be asynchronous:

const observable = rxjs.Observable.create(observer => {

setTimeout(()=>{

observer.next('Hello world');

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');

The messages will be displayed in the following order:

**Before calling subscribe on an Observable  
After calling subscribe on an Observable  
Hello world!**

You can see in this case, observable runs asynchronously.

**Angular by default, uses client-side rendering for its applications.**

Can one make an angular application to render on the server-side?  
Yes, angular provides a technology called **Angular Universal**, which can be used to render applications on the server-side. The advantages of using Angular Universal are :

* First time users can instantly see a view of the application. This benefits in providing **better user experience**.
* Many search engines expect pages in plain HTML, thus, Universal can make sure that your content is available on every search engine, which leads to **better SEO**.
* Any server-side rendered application **loads faster** since rendered pages are available to the browser sooner.

**What are directives in Angular?**

A directive is a class in Angular that is declared with a **@Directive** decorator.  
Every directive has its own behaviour and can be imported into various components of an application.

**When to use a directive?**  
Consider an application, where multiple components need to have similar functionalities. The norm thing to do is by adding this functionality individually to every component but, this task is tedious to perform. In such a situation, one can create a **directive** having the required functionality and then, import the directive to components which require this functionality. Types of directives:

* **Component directives**  
  These form the main class in directives. **Instead** of @Directive decorator we use **@Component** decorator to declare these directives. These directives have a view, a stylesheet and a selector property.
* **Structural directives**  
  These directives are generally used to manipulate DOM elements.  
  Every structural directive has a ‘ \* ’ sign before them.  
  We can apply these directives to any DOM element.  
    
  Let’s see some built-in structural directives in action:

<div \*ngIf="isReady" class="display\_name">

{{name}}

</div>

<div class="details" \*ngFor="let x of details" >

<p>{{x.name}}</p>

<p> {{x.address}}</p>

<p>{{x.age}}</p>

</div>

In the above example, we can \*ngIf and \*ngFor directives being used.  
\*ngIf is used to check a boolean value and if it’s truthy,the div element will be displayed.  
\*ngFor is used to iterate over a list and display each item of the list.

* **Attribute Directives**  
  These directives are used to change the look and behaviour of a DOM element. Let’s understand attribute directives by creating one: How to create a custom directive? We’re going to create an attribute directive:  
  In the command terminal, navigate to the directory of the angular app and type the following command to generate a directive.

ng g directive blueBackground

The following directive will be generated. Manipulate the directive to look like this:

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

@Directive({

selector: '[appBlueBackground]'

})

export class BlueBackgroundDirective {

constructor(el:ElementRef) {

el.nativeElement.style.backgroundColor = "blue";

}

}

Now we can apply the above directive to any DOM element:

<p appBlueBackground>Hello World!</p>

**How does one share data between components in Angular?**

* **Parent to child using @Input decorator**: Consider the following parent component:

@Component({

selector: 'app-parent',

template: `

<app-child [data]=data></app-child>

` ,

styleUrls: ['./parent.component.css']

})

export class ParentComponent{

data:string = "Message from parent";

constructor() { }

}

In the above parent component, we are passing “data” property to the following child component:

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

@Component({

selector: 'app-child',

template:`

<p>{{data}}</p>

`,

styleUrls: ['./child.component.css']

})

export class ChildComponent {

**@Input()** data:string

constructor() { }

}

In the child component, we are using @Input decorator to capture data coming from a parent component and using it inside the child component’s template.

* **Child to parent using @ViewChild decorator:** Child component:

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

@Component({

selector: 'app-child',

template:`

<p>{{data}}</p>

`,

styleUrls: ['./child.component.css']

})

export class ChildComponent {

data:string = "Message from child to parent";

constructor() { }

}

Parent Component

import { Component,ViewChild, AfterViewInit} from '@angular/core';

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

@Component({

selector: 'app-parent',

template: `

<p>{{dataFromChild}}</p>

` ,

styleUrls: ['./parent.component.css']

})

export class ParentComponent implements AfterViewInit {

dataFromChild: string;

**@ViewChild(ChildComponent,{static:false}) child;**

ngAfterViewInit(){

this.dataFromChild = this.child.data;

}

constructor() { }

}

In the above example, a property named “data” is passed from the child component to the parent component.  
**@ViewChild** decorator is used to reference the child component as “child” property.  
Using the **ngAfterViewInit** hook, we assign the child’s data property to the messageFromChild property and use it in the parent component’s template.

* **Child to parent using @Output and EventEmitter**: In this method, we bind a DOM element inside the child component, to an event ( **click** event for example ) and using this event we emit data that will captured by the parent component: Child Component:

import {Component, Output, EventEmitter} from '@angular/core';

@Component({

selector: 'app-child',

template:`

<button (click)="emitData()">Click to emit data</button>

`,

styleUrls: ['./child.component.css']

})

export class ChildComponent {

data:string = "Message from child to parent";

**@Output() dataEvent = new EventEmitter<string>();**

constructor() { }

emitData(){

this.dataEvent.emit(this.data);

}

}

As you can see in the child component, we have used **@Output** property to bind an **EventEmitter**. This event emitter emits data when the button in the template is clicked.  
  
In the parent component’s template we can capture the emitted data like this:

<app-child (dataEvent)="receiveData($event)"></app-child>

Then inside the receiveData function we can handle the emitted data:

receiveData($event){

this.dataFromChild = $event;

}

**Explain the concept of Dependency Injection?**

Dependency in angular are nothing but services which have a functionality. Functionality of a service, can be needed by various components and directives in an application. Angular provides a smooth mechanism by which we can inject these dependencies in our components and directives. So basically, we are just making dependencies which are injectable across all components of an application. Consider the following service, which can be generated using: **ng g service test**

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

**@Injectable**({

providedIn: 'root'

})

export class TestService {

importantValue:number = 42;

constructor() { }

returnImportantValue(){

return this.importantValue;

}

}

As one can notice, we can create injectable dependencies by adding the **@Injectable** decorator to a class.  
We inject the above dependency inside the following component:

import { TestService } from './../test.service';

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

@Component({

selector: 'app-test',

templateUrl: './test.component.html',

styleUrls: ['./test.component.css']

})

export class TestComponent implements OnInit {

value:number;

**constructor(private testService:TestService) { }**

ngOnInit() {

this.value = this.testService.returnImportantValue();

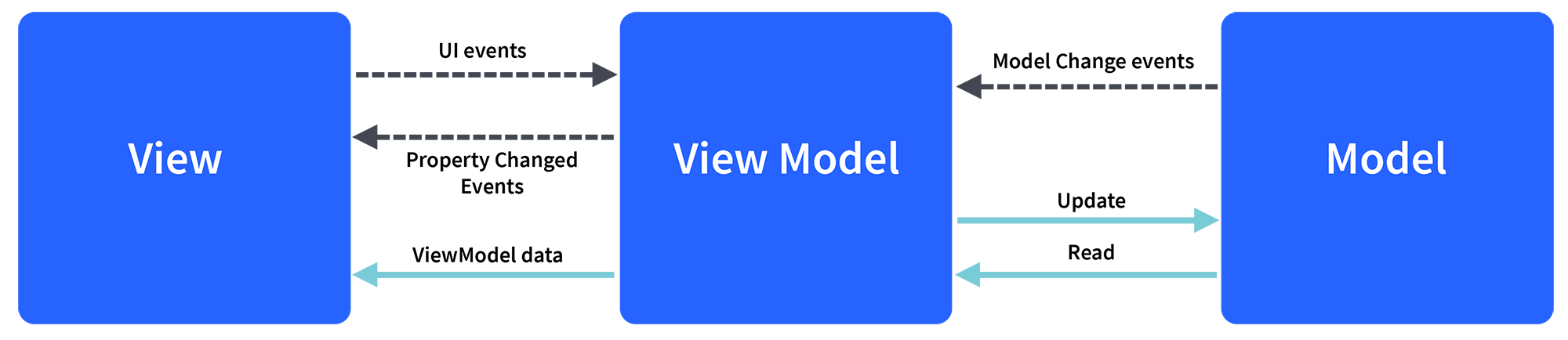
}

}

One can see we have imported our TestService at the top of the page. Then, we have created an instance inside the constructor of the component and implemented the **returnImportantValue** function of the service.

**Explain MVVM architecture**

MVVM architecture consists of three parts:  
1. Model  
2. View  
3. ViewModel



* **Model** contains the structure of an entity. In simple terms it contains data of an object.
* **View** is the visual layer of the application. It displays the data contained inside the Model. In angular terms, this will be the HTML template of a component.
* **ViewModel** is an abstract layer of the application. A viewmodel handles the logic of the application. It manages the data of a model and displays it in the view.  
  View and ViewModel are connected with data-binding (two-way data-binding in this case). Any change in the view, the viewmodel takes a note and changes the appropriate data inside the model.

**Demonstrate navigating between different routes in an Angular application.**

Following code demonstrates how to navigate between different routes in an Angular app dubbed “Some Search App”:

import from "@angular/router";

**@Component({**

selector: 'app-header',

template: `

<nav class="navbar navbar-light bg-faded">

<a class="navbar-brand" (click)="goHome()">Some Search 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']);

}

}

**Discuss the advantages and disadvantages of using Angular?**

* **Out of box Features:**Several built-in features like routing, state management, rxjs library, and HTTP services are straight out of the box services provided by Angular. So, one does not need to look for the above-stated features separately.
* **Declarative UI:**Angular uses HTML to render the UI of an application as it is a declarative language and is much easier to use than JavaScript.
* **Long-term Google Support:**Google plans to stick with Angular and further scale up its ecosystem as it has offered its long term support to Angular.
* Ability to add a custom directive
* Exceptional community support
* Facilitates client and server communication
* Features strong features, such as Animation and Event Handlers
* Follows the MVC pattern architecture
* Offers support for static template and Angular template
* Support for two-way data-binding
* Supports dependency injection, RESTful services, and validations

Disadvantages of using Angular are enumerated as follows:

* Complex SPAs can be inconvenient and laggy to use due to their size
* Dynamic applications do not always perform well
* Learning Angular requires a decent effort and time

**Can you explain the concept of scope hierarchy in Angular?**

Angular organizes the $scope objects into a hierarchy that is typically used by views. This is known as 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 a typical representation of a Scope Hierarchy:

* Root $scope
  + $scope for Controller 1
  + $scope for Controller 2
  + …
  + ..
  + .
  + $scope for Controller n

#### ****How to generate a class in Angular 7 using CLI?**** ****ng generate class Dummy [options]****

#### ****Explain the difference between an Annotation and a Decorator in Angular?****

In Angular, annotations are used for creating an annotation array. They are only metadata set of the class using the Reflect Metadata library.

Decorators in Angular are design patterns used for separating decoration or modification of some class without changing the original source code.

**Please explain the differences between Angular and jQuery?**

The single biggest difference between Angular and jQuery is that while the former is a JS frontend framework, the latter is a JS library. Despite this, there are some similarities between the two, such as both features DOM manipulation and provides support for animation.

Nonetheless, notable differences between Angular and jQuery are:

* Angular has two-way data binding, jQuery does not
* Angular provides support for RESTful API while jQuery doesn’t
* jQuery doesn’t offer deep linking routing though Angular supports it
* There is no form validation in jQuery whereas it is present in Angular

#### ****What is Angular Material?****

It is a UI component library. [Angular Material](https://material.angular.io/) helps in creating attractive, consistent, and fully functional web pages as well as web applications. It does so while following modern web design principles, including browser portability and graceful degradation.

**Can you draw a comparison between the service() and the factory() functions?**

Used for the business layer of the application, the service() function operates as a constructor function. The function is invoked at runtime using the new keyword. Although the factory() function works in pretty much the same way as the service() function does, the former is more flexible and powerful. In actual, the factory() function are design patterns that help in creating objects.

**Please explain the digest cycle in Angular?**

The process of monitoring the watchlist in order to track changes in the value of the watch variable is termed the digest cycle in Angular. The previous and present versions of the scope model values are compared in each digest cycle. Although the digest cycle process gets triggered implicitly, it is possible to start it manually by using the $apply() function.

**Could you explain the various types of filters in AngularJS.**

In order to format the value of expression so that it can be displayed to the user, AngularJS has filters. It is possible to add these filters to the controllers, directives, services, or templates. Angular also provides support for creating custom filters. Organizing data in such a way so that it is displayed only when certain criteria are fulfilled is made possible using filters. Filters are added to the expressions using the pipe ‘|’ character.

|  |  |
| --- | --- |
| Filter name | Description |
| Uppercase | Convert string to uppercase |
| Lowercase | Convert string to lowercase |
| Date | Convert date to the specified format |
| Currency | Convert the number to currency format |
| Number | Format number into a string |
| Orderby | Orders an array by specific expression |
| limitTo | Limits array into the specified number of elements; string to specified number of characters |
| JSON | Format object to JSON string |
| Filter | A select a subset of items from the array |

You can mention few of them and show an example as well –

**<p>Amount: {{ amount | currency }}</p>**

**What is the code for creating a decorator?**

We create a decorator called Dummy:

function Dummy(target) {

dummy.log('This decorator is Dummy', target);

}

**What is the process called by which TypeScript code is converted into JavaScript code?**

It is called **Transpiling**. Even though TypeScript is used for writing code in Angular applications, it gets internally transpiled into equivalent JavaScript.

**What is ViewEncapsulation and how many ways are there do to do it in Angular?**

Component CSS styles are encapsulated into the component’s view to avoid styling side effects in the rest of the Angular application. To put simply, ViewEncapsulation determines whether the styles defined in a particular component will affect the entire application or not. Angular supports 3 types of ViewEncapsulation:

* **Emulated –** Styles used in other HTML spread to the component. It is the default mode and emulates the shadow DOM behavior. It renames and preprocesses the CSS code to effectively scope the CSS to the component’s view. Each DOM element gets attached some additional attributes like \_**nghost or \_ngcontent**. An element that would be a shadow DOM host in native encapsulation has a generated **\_nghost** attribute. This is typically the case for component host elements. An element within a component’s view has a **\_ngcontent** attribute that identifies to which host’s emulated shadow DOM this element belongs.
* **Native –** Styles used in other HTML doesn’t spread to the component
* **None –** Styles defined in a component are visible to all components of the application. It tells Angular to not use view encapsulation and adds CSS to the global styles. Essentially, this is the same behavior as pasing the component’s styles into the HTML.
* **ShadowDom** which uses the browser’s native shadow DOM implementation. It attaches a shadow DOM to the component’s host element and then puts the component view inside that shadow DOM. The component’s styles are included within the shadow DOM.

**Why prioritize TypeScript over JavaScript in Angular?**

TypeScript is a superset of Javascript as it is Javascript + Types or extra features like typecasting for variables, annotations, variable scope and much more. The typescript is designed in a way to overcome Javascript shortcomings like typecasting of variables, classes, decorators, variable scope and many more. Moreover, Typescript is purely object-oriented programming that offers a "Compiler" that can convert to Javascript-equivalent code.

### What Is Angular CLI? How Do You Use It?

Angular CLI automates the end-to-end development process. The app's initialization, configuration, and development process become straightforward and easy. With a CLI (Command Line Interface), we can create a new project, add new features, and run tests (unit tests and end-to-end tests) by just typing a few simple commands. This way, development and testing processes both become faster. For example,

To create a new application,-- **ng new <appname> [options]**

To create a class using CLI (in Angular 7), we have to type –**ng generate class MySampleClass [options]**

To generate a component-- **ng g c <componentname>**

1

### What Are Ngmodules? Differentiate Between Javascript Modules and Ngmodules

NgModule was introduced after Angular 2, to enable developers to declare all the relationships in one place with metadata. Thus, in short, NgModules are built from metadata that describes components, services, directives, pipes etc… Angular then creates a component factory, a class that creates components.

Difference between JS modules and NgModules –

|  |  |
| --- | --- |
| JS modules | NgModules |
| Bounds all the classes | Bounds only declarable classes |
| All the member classes are defined in a single file | The module’s classes are listed in the @NgModule.declarations list |
| Cant extend the entire application with services | The entire application can be extended with services using @NgModules.providers list to add providers |
| Can import or export any kind of classes | It can import or export only those declarable classes that it owns or imports from other modules. |

### What Are ngIf and ngFor? Can You Show a Small Example to Use Them?

Just like if and for in other languages, ngIf and ngFor are used as control statements. Example –

**<p \*ngIf="display">Show this only if the Boolean "display" is true</p>**

Where the display is a boolean with the value true or false. [Learn more](https://malcoded.com/posts/angular-ngif-else/) about ngIf. ngFor is used to loop through and display elements of an array (set of data).

**<tr \*ngFor="let student of students; let i = index"> <td>{{student.name}}**

**</td> <td>{{i}}</td> </tr>**

The second part (i=index) is optional and only needed if you want to display the index.

**What Is the Digest Cycle?**

Digest cycle is the process of monitoring watchlist to track the changes in the value of the watch variable. The digest cycle is implicitly triggered, but we can also trigger it manually using $apply() function.

**What Is a Pipe? Write a Simple Code to Demonstrate.**

Pipe (|) is used to transform input data into desired format. For example,

<p>Price is {{ price | currency }}</p>

**Can You Create a Parameterized Pipe in the Above Example?**

Yes🡪 <p>Price is {{ price | currency : “USD$” : 0.00 }}</p>

**Explain How You Can Chain Pipes**

We can add any number of filters using pipes -

<p>Average is {{ average | uppercase | number}}</p>

**Is It Possible to Create a Custom Pipe? How?**

Yes, we can create custom pipes.

* Pipe metadata @Pipe decorator can be imported from core Angular library
* Pipe is a class that is decorated with the above metadata (@Pipe({name: 'myCustomPipe'}))
* The next step is to define the transformation. For this, the pipe class should implement the method transform() of the PipeTransform class.
* Specify the pipe name in the main code

**<p>Size: {{number | myCustomPipe: 'Error'}}</p>**

**What Is the Purpose of an Async Pipe?**

Async pipe subscribes to a promise or an observable, and returns the latest value. If a new value is emitted, the pipe marks the component that needs to be checked for any changes.

**<code>observable|async</code>**

**What Is the Difference Between Pure and Impure Pipe?**

|  |  |
| --- | --- |
| **Pure pipe** | **Impure pipe** |
| Doesn’t get affected by internal state | Can produce different output for the same input based on the internal state |
| Can be shared with many different instances | It cannot be shared because the internal state can be affected by any factors. |

**Pure pipes** are stateless that flow input date without remembering anything or causing detectable side-effects. Pipes are pure by default, hence most pipes are pure. As well as, we can make a pipe impure by setting its pure flag to false. Secondly, Angular executes a pure pipe only when it detects a pure change to the input value. Also, a pure change is either a change to a primitive input value or a changed object reference.

**Impure pipes** are those which can manage the state of the data they transform. Also, a pipe that creates an HTTP request, stores the response and displays the output, is an impure or stateful pipe. Secondly, Stateful Pipes should be used cautiously. Angular provides AsyncPipe, which is stateful. In the following code, the pipe only calls the server when the request URL changes and it caches the server response.

**Explain the Importance of HttpClient.**

HttpClient is a simplified Http API for Angular applications. It gives better observable APIs, better error handling mechanisms, testability, request and response interception, typed request and response objects. The HttpClientAPI rests on the XMLHttpRequest interface exposed by the browsers.

**How Does Angular Router Work?**

Angular router interprets a browser URL as commands to navigate to a client-generated view. The router is bound to the links on a page. This way Angular knows to navigate the application view to the required page when a user clicks on it.

**What Are the Router Navigation Events?**

Router navigation events help track the lifecycle of a route. These are –

* NavigationStart,
* RouteConfigLoadStart,
* RouteConfigLoadEnd,
* RoutesRecognized,
* GuardsCheckStart,
* ChildActivationStart,
* ActivationStart,
* GuardsCheckEnd,
* ResolveStart,
* ResolveEnd,
* ActivationEnd
* ChildActivationEnd
* NavigationEnd,
* NavigationCancel,
* NavigationError
* Scroll

**Is the Routing Module Mandatory for an Application?**

No, routing module can be totally skipped if there are simple configurations.

**What Is a Wildcard Route?**

Wildcard route has the path that consists of two asterisks (\*\*) that can match any URL. It is helpful when a URL doesn’t match any of the predefined routes. Instead of throwing error, we can use a wildcard route and defining a component for the same.

**How Are Animations Done in Angular?**

To use the animation module, it has to be enabled. For this, the BrowserAnimationModule has to be imported.

import { BrowserAnimationsModule } from '@angular/platform-browser/animations'; After this, import the required animation functions into the component files. Example, import { state, animate, transition, // ... } from '@angular/animations';

Next, add the animation metadata property within the @Component() decorator in the component file.

@Component({

selector: 'app-root',

templateUrl: 'app.component.html',

animations: [ // animation triggers go here ]

})

**What are the Special Transition States?**

Special transition states are wildcard (\*) and void. Wildcard matches any animation state. The void state is used to configure transitions for elements entering or leaving a page.

**How Can You Disable All the Animations in Angular?**

To disable all the animations, place the @.disabled host binding on the topmost Angular component.

**Explain the steps to create a reusable animation.**

To create an animation that can be reused, use the animation() method and define the animation in a separate .ts file. Declare this animation as a const export variable. This can be then imported and reused in any app components that use the useAnimation() API. Check an example on the [Angular website](https://angular.io/guide/reusable-animations).

**Mention Some of the Functions That Help Control Complex Animation Sequences**

|  |  |
| --- | --- |
| query() | finds one or more inner HTML elements within the current element being animated in the sequence |
| stagger() | applies a cascading delay (timing gap) after each animation |
| group() | runs multiple animation steps in parallel. |
| sequence() | runs animation steps one after another (sequentially) |

**the Features of Forms in Angular.**

There are two approaches to handle form data (user inputs) – reactive and template-driven.

Reactive forms can be used when you are using reactive patterns in your application and forms are a key part of your application. These forms are scalable, robust and testable.

Template-driven forms are used to add simple forms, for example, a sign-up page. These are not as scalable as reactive forms and should be used only if your form requirements are simple and minimal.

**How Is Metadata Represented in Angular?**

Metadata is represented using decorators like class decorators, property decorators, method decorators, property decorators. Example, @Component, @NgModule etc…

**What Are Class Decorators in Angular?**

Class decorator contains the metadata of the suitable class type. It appears just before the class definition and declares the class to be of a certain type. Some class decorators are — @Component, @NgModule, @Pipe, @Directive, @Injectable.

**Explain the Difference Between Annotations and Decorators in Angular**

Annotations are hardcoded features of Angular and store array in it. The compiler creates am attribute of the annotated class and instantiates an object of the same name, passing the metadata to the constructor.

Decorators, on the other hand, are functions that receive the object to be decorated. After receiving, they are free to modify the object in the way it likes. Decorators are implemented by the TypeScript compiler.

**What Is the Difference Between Class Decorators and Class Field Decorators?**

Class decorators appear just before class definition, whereas class field decorators appear just before a field in the class definition. Examples of class decorators are @Component, @NgModule etc… Examples of a class field decorator are @Input, @Output etc…

**What Is Package.json? Explain its Purpose**

With json package, it becomes easy to manage the project dependencies. We can mention details like the version, language etc… in package.json. For example, if typescript is used in our project, we can mention typescript and its version in package.json. Examples are metadata.json, tsconfig.json etc…

**What Does the Representation [()] Mean?**

This is a representation for ngModel used for two-way data binding. It is written as [(ngModel)] = “propertyvalue”.

**What Is a Bootstrapping Module in Angular?**

The root module that you bootstrap to launch the application is called as a bootstrapping module. Every Angular application has a bootstrapping module. It is also called as the AppModule. The bootstrapping module is mentioned in the AppModule class.

@NgModule({

declarations: [ AppComponent ],

imports: [

BrowserModule,

FormsModule,

HttpClientModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

Angular bootstrapping, in simple words, allows professionals to initialize or start the Angular application. Angular supports both manual and automatic bootstrapping. Let’s briefly understand the two.

* **Manual bootstrapping:** It gives more control to professionals with regards to how and when they need to initialize the Angular app. It is extremely useful in places where professionals wish to perform other tasks and operations before the Angular compiles the page.
* **Automatic bootstrapping:** Automatic bootstrapping can be used to add the ng-app directive to the application’s root, often on the tag if professionals need Angular to automatically bootstrap the application. Angular loads the associated module once it finds the ng-app directive and, further, compiles the DOM.

## **Explain ng-app directive in Angular.**

ng-app directive is used to define Angular applications which let us use the auto-bootstrap in an Angular application. It represents the root element of an Angular application and is generally declared near <html> or <body> tag. Any number of ng-app directives can be defined within an HTML document but just a single Angular application can be officially bootstrapped implicitly. Rest of the applications must be manually bootstrapped.

**Example**

<div ng-app=“myApp” ng-controller=“myCtrl”>  
First Name :  
<input type=“text” ng-model=“firstName”>  
<br />  
Last Name :  
<input type=“text” ng-model=“lastName”>  
<br>  
Full Name: {{firstName + ” ” + lastName }}  
</div>

## **49. What is the process of inserting an embedded view from a prepared TemplateRef?**

@Component({

selector: 'app-root',

template: `

<ng-template #template let-name='fromContext'><div>{{name}}</ng-template>

`

})

export class AppComponent implements AfterViewChecked {

@ViewChild('template', { read: TemplateRef }) \_template: TemplateRef<any>;

constructor() { }

ngAfterViewChecked() {

this.vc.createEmbeddedView(this.\_template, {fromContext: 'John'});

}

}

## **50. How can you hide an HTML element just by a button click in angular?**

An HTML element can be easily hidden using the ng-hide directive in conjunction along with a controller to hide an HTML element on button click.

**View**

<div ng-controller="MyController">

<button ng-click="hide()">Hide element</button>

<p ng-hide="isHide">Hello World!</p>

</div>

**Controller**

controller: function() {

this.isHide = false;

this.hide = function(){

this.isHide = true; }; }

### ****What is server-side rendering in Angular?****

In a normal Angular application, the browser executes our application, and JavaScript handles all the user interactions. However, because of this, sometimes, if we have a large application with a big bundle size, our page’s load speed is slowed down quite a bit as it needs to download all the files, parse JavaScript, and then execute it. To overcome this slowness, we can use server-side rendering, which allows us to send a fully rendered page from the server that the browser can display and then let the JavaScript code take over any subsequent interactions from the user.

### ****What is Angular Universal?****

Angular Universal is a package for enabling server-side rendering in Angular applications. We can easily make our application ready for server-side rendering using the Angular CLI. To do this, we need to type the following command:

ng add @nguniversal/express-engine

This allows our Angular application to work well with an ExpressJS web server that compiles HTML pages with Angular Universal based on client requests. This also creates the server-side app module, app.server.module.ts, in our application directory.

### ****What is the difference between interpolated content and the content assigned to the innerHTML property of a DOM element?****

Angular interpolation happens when in our template we type some JavaScript expression inside double curly braces ‘{{ someExpression() }}’. This is used to add dynamic content to a web page. However, we can do the same by assigning some dynamic content to the innerHTML property of a DOM element. The difference between the two is that, in Angular, the compiler always escapes the interpolated content, i.e., HTML is not interpreted, and the browser displays the code as it is with brackets and symbols, rather than displaying the output of the interpreted HTML. However, in innerHTML, if the content is HTML, then it is interpreted as the HTML code.

### ****What are HttpInterceptors in Angular?****

HttpInterceptors are part of the @angular/common/http module and are used to inspect and transform HTTP requests and HTTP responses as well. These interceptors are created to perform checks on a request, manipulate the response, and perform cross-cutting concerns, such as logging requests, authenticating a user using a request, using gzip to compress the response, etc.

### ****What is the purpose of the common module in Angular?****

In Angular, the common module that is available in the package @angualr/common is a module that encapsulates all the commonly needed features of Angular, such as services, pipes, directives, etc. It contains some sub-modules as well such as the HttpClientModule, which is available in the @angular/common/http package. Because of the modular nature of Angular, its functionalities are stored in small self-contained modules, which can be imported and included in our projects if we need these functionalities.

### ****What does subscribing mean in RxJS?****

In RxJS, when using observables, we need to subscribe to an observable to use the data that flows through that observable. This data is generated from a publisher and is consumed by a subscriber. When we subscribe to an observable, we pass in a function for the data and another function for errors so that, in case there is some error, we can show some message or process the message in some way.

### ****What is Angular Router?****

Routing in a single-page frontend application is the task of responding to the changes in the URL made by adding and removing content from the application. This is a complicated task as we first need to intercept a request that changes the browser’s URL as we do not wish for the browser to reload. Then, we need to determine which content to remove and which content to add, and finally, we have to change the browser’s URL as well to show the user the current page they are on.

As we can see, this can be very difficult to implement, especially in multiple applications. That is why Angular comes with a full routing solution for a single-page application. In this, we can define routes with matching components and let Angular handle the routing process.

### ****What is the scope?****

A scope is an object in Angular referring to the application model. It is a context for executing expressions. These scopes are organized in a hierarchical form that is similar to the application’s DOM structure. A scope helps in propagating various events and watching expressions.

### ****Explain Angular CLI.****

Angular CLI is otherwise known as Angular command-line interface. Angular supports CLI tools that give professionals the ability to use them to add components, deploy them instantly, and perform testing and many such functions.

### ****What is HttpClient, and what are its benefits?****

HttpClient is an Angular module used for communicating with a backend service via the HTTP protocol. Usually, in frontend applications, for sending requests, we use the fetch API. However, the fetch API uses promises. Promises are useful, but they do not offer the rich functionalities that observables offer. This is why we use HttpClient in Angular as it returns the data as an observable, which we can subscribe to, unsubscribe to, and perform several operations on using operators. Observables can be converted to promises, and an observable can be created from a promise as well.

### ****What is multicasting in Angular?****

In Angular, when we are using the HttpClient module to communicate with a backend service and fetch some data, after fetching the data, we can broadcast it to multiple subscribers, all in one execution. This task of responding with data to multiple subscribers is called multicasting. It is specifically useful when we have multiple parts of our applications waiting for some data. To use multicasting, we need to use an RxJS subject. As observables are unicast, they do not allow multiple subscribers. However, subjects do allow multiple subscribers and are multicast.

### What Is an Angular Library? Can You Create Your Own Library in Angular?

Angular library is a set of generic solutions that other developers have put together to be re-used. We can create own library using Angular. These libraries can be published and shared as npm packages. A library should be imported in the app.

### What Do You Know About the NPM Package?

The components, framework and CLI used by Angular applications are packaged as npm packages. Npm packages can be downloaded using the npm CLI client.

### Write a Sample Code to Create a Library

Answer: You can use the Angular CLI for this. The following set of commands generates a new library skeleton –

ng new my-workspace --create-application=false cd my-workspace

ng generate library my-lib

**What are Template and Reactive forms?**

Template-driven approach

* In this method, the conventional form tag is used to create forms. Angular automatically interprets and creates a form object representation for the tag.
* Controls can be added to the form using the NGModel tag. Multiple controls can be grouped using the NGControlGroup module.
* A form value can be generated using the “form.value” object. Form data is exported as JSON values when the submit method is called.
* Basic HTML validations can be used to validate the form fields. In the case of custom validations, directives can be used.
* Arguably, this method is the simplest way to create an Angular App.

Reactive Form Approach

* This approach is the programming paradigm oriented around data flows and propagation of change.
* With Reactive forms, the component directly manages the data flows between the form controls and the data models.
* Reactive forms are code-driven, unlike the template-driven approach.
* Reactive forms break from the traditional declarative approach.
* Reactive forms eliminate the anti-pattern of updating the data model via two-way data binding.
* Typically, Reactive form control creation is synchronous and can be unit tested with synchronous programming techniques.

**What is Bootstrap? How is it embedded into Angular?**

Bootstrap is a powerful toolkit. It is a collection of HTML, CSS, and JavaScript tools for creating and building responsive web pages and web applications.

There are two ways to embed the bootstrap library into your application.

1. Angular Bootstrap via CDN - Bootstrap CDN is a public Content Delivery Network. It enables you to load the CSS and JavaScript files remotely from its servers.
2. Angular Bootstrap via NPM - Another way to add Bootstrap to your Angular project is to install it into your project folder by using NPM (Node Package Manager).

**npm install bootstrap**

**npm install jquery**

**What is Eager and Lazy loading?**

Eager loading is the default module-loading strategy. Feature modules under Eager loading are loaded before the application starts. This is typically used for small size applications.

Lazy loading dynamically loads the feature modules when there's a demand. This makes the application faster. It is used for bigger applications where all the modules are not required at the start of the application.

**What type of DOM does Angular implement?**

DOM (Document Object Model) treats an XML or HTML document as a tree structure in which each node is an object representing a part of the document. Angular uses the regular DOM. This updates the entire tree structure of HTML tags until it reaches the data to be updated. However, to ensure that the speed and performance are not affected, Angular implements Change Detection. With this, you have reached the end of the article. We highly recommend brushing up on the core concepts for an interview. It’s always an added advantage to write the code in places necessary.

## **What is a provider in Angular?**

A provider is a configurable service in Angular. It is an instruction to the Dependency Injection system that provides information about the way to obtain a value for a dependency. It is an object that has a $get() method which is called to create a new instance of a service. A Provider can also contain additional methods and uses $provide in order to register new providers.

## **Does Angular support nested controllers?**

Yes, Angular does support the concept of nested controllers. The nested controllers are needed to be defined in a hierarchical manner for using it in the View.  **List at down the ways in which you can communicate between applications modules using core Angular functionality.** Below are the most general ways for communicating between application modules using core Angular functionality :

* Using events
* Using services
* By assigning models on **$rootScope**
* Directly between controllers [**$parent**, **$$childHead**, **$$nextSibling**, etc.]
* Directly between controllers [**ControllerAs**, or other forms of inheritance]

## **What is the difference between a service() and a factory()?**

A service() in Angular is a function that is used for the business layer of the application. It operates as a constructor function and is invoked once at the runtime using the ‘new’ keyword. Whereas factory() is a function which works similar to the service() but is much more powerful and flexible. factory() are design patterns which help in creating Objects.

## **What is the difference between $scope and scope in Angular?**

* $**scope** in Angular is used for implementing the concept of dependency injection (D.I) on the other hand **scope** is used for directive linking.
* $**scope** is the service provided by $scopeProviderwhich can be injected into controllers, directives or other services whereas **Scope** can be anything such as a function parameter name, etc.

## **How to create a service in Angular?**

In Angular, a service is a substitutable object that is wired together using dependency injection. A service is created by registering it in the module it is going to be executed within. There are basically three ways in which you can create an angular service. They are basically three ways in which a service can be created in Angular:

* Factory
* Service
* Provider

|  |  |  |
| --- | --- | --- |
| **Provider** | **Service** | **Factory** |
| A provider is a method using which you can pass a portion of your application into app.config | A service is a method that is used to create a service instantiated with the ‘new’ keyword. | It is a method that is used for creating and configuring services. Here you create an object, add properties to it and then return the same object and pass the factory method into your controller. |

## **What do you understand by dirty checking in Angular?**

In Angular, the digest process is known as **dirty checking**. It is called so as it scans the entire scope for changes. In other words, it compares all the new scope model values with the previous scope values. Since all the watched variables are contained in a single loop, any change/update in any of the variable leads to reassigning of rest of the watched variables present inside the DOM. A watched variable is in a single loop(digest cycle), any value change of any variable forces to reassign values of other watched variables in DOM

## **Differentiate between DOM and BOM.**

|  |  |
| --- | --- |
| **DOM** | **BOM** |
| 1. Stands for Document Object Model | 1. Stands for Browser Object Model |
| 2. Represents the contents of a web page | 2. Works a level above web page and includes browser attributes |
| 3. All the Objects are arranged in a tree structure and the document can be manipulated & accessed via provided APIs only | 3. All global JavaScript objects, variables & functions become members of the window object implicitly |
| 4. Manipulates HTML documents | 4. Access and manipulate the browser window |
| 5. W3C Recommended standard specifications | 5. Each browser has its own implementation |

## **What is transclusion in Angular?**

The transclusion in Angular allows you to shift the original children of a directive into a specific location within a new template. The ng directive indicates the insertion point for a transcluded DOM of the nearest parent directive that is using transclusion. Attribute directives like **ng-transclude** or **ng-transclude-slot** are mainly used for transclusion.

## **List some tools for testing angular applications?**

1. Karma
2. Angular Mocks
3. Mocha
4. Browserify
5. Sion

## **What is a singleton pattern and where we can find it in Angular?**

Singleton pattern in Angular is a great pattern which restricts a class from being used more than once. Singleton pattern in Angular is majorly implemented on dependency injection and in the services. Thus, if you use ‘new Object()’ without making it a singleton, then two different memory locations will be allocated for the same object. Whereas, if the object is declared as a singleton, in case it already exists in the memory then simply it will be reused.

**What is bootstrapping in Angular?**

Bootstrapping in Angular is nothing but initializing, or starting the Angular app. Angular supports automatic and manual bootstrapping.

* ***Automatic Bootstrapping:*** this is done by adding the ng-app directive to the root of the application, typically on the tag or tag if you want angular to bootstrap your application automatically. When Angular finds ng-app directive, it loads the module associated with it and then compiles the DOM.
* **Manual Bootstrapping:**Manual bootstrapping provides you more control on how and when to initialize your Angular application. It is useful where you want to perform any other operation before Angular wakes up and compile the page.

## **What is the difference between a link and compile in Angular?**

* Compile function is used for template DOM Manipulation and to collect all the directives.
* Link function is used for registering DOM listeners as well as instance DOM manipulation and is executed once the template has been cloned.

**What do you understand by constants in Angular?**

In Angular, constants are similar to the services which are used to define the global data. Constants are declared using the keyword “constant”. They are created using constant dependency and can be injected anywhere in controller or services.

## **What is the difference between a provider, a service and a factory in Angular? 45.** **What are Angular Global APIs?**

Angular Global API is a combination of global JavaScript functions for performing various common tasks like:

* Comparing objects
* Iterating objects
* Converting data

There are some common Angular Global API functions like:

* **angular. lowercase:** Converts a string to lowercase string.
* **angular. uppercase:** Converts a string to uppercase string.
* **angular. isString:**Returns true if the current reference is a string.
* **angular. isNumber:** Returns true if the current reference is a number.

## **In Angular, describe how will you set, get and clear cookies?**

For using cookies in Angular, you need to include a  module called ngCookies angular-cookies.js.

**To set Cookies** – For setting the cookies in a key-value format ‘put’ method is used.

cookie.set('nameOfCookie',"cookieValue");

**To get Cookies –** For retrieving the cookies ‘get’ method is used.

cookie.get(‘nameOfCookie’);

**To clear Cookies –** For removing cookies ‘remove’ method is used.

cookie.delete(‘nameOfCookie’);

## **If your data model is updated outside the ‘Zone’, explain the process how will you the view?**

You can update your view using any of the following:

1. **ApplicationRef.prototype.tick()**: It will perform change detection on the complete component tree.
2. **NgZone.prototype.run():** It will perform the change detection on the entire component tree. Here, the run() under the hood will call the tick itself and then parameter will take the function before tick and executes it.
3. **ChangeDetectorRef.prototype.detectChanges():**It will launch the change detection on the current component and its children.

### ****What is RxJS?****

RxJS is a library, and the term stands for Reactive Extensions for JavaScript. It is used so that we can use observables in our JavaScript project, which enables us to perform reactive programming. RxJS is used in many popular frameworks such as Angular because it allows us to compose our asynchronous operations or callback-based code into a series of operations performed on a stream of data that emits values from a publisher to a subscriber. Other languages such as Java, Python, etc. also have libraries that allow them to write reactive code using observables.

**What are @HostBinding and @HostListener?**

•@HostListener() function decorator allows you to handle events of the host element in the directive class. For example, it can be used to change the color of the host element if you hover over the host element with the mouse.

•@HostBinding() function decorator allows you to set the properties of the host element from the directive class. In this directive class, we can change any style property like height, width, color, margin, border, etc.

**What is the difference between OnPush and default change detection?**

#### ****Define the ng-content Directive?****

#### ****Explain Angular Authentication and Authorization.****

| **Authentication** | **Authorization** |
| --- | --- |
| Process of verifying the user | Process of verifying that you have relevant access to any procedure |
| Methods: Login form, HTTP Authentication, HTTP digest, X 509 Certificates, and Custom Authentication method. | Methods: Access controls for URL, secure objects and methods and Access Control Lists (ACL) |