ANGULAR

What is a angular:

Angular is a typescript framework for developing single page application

Key	AngularJS	Angular
Architecture	AngularJS works on MVC, Model View Controller Design. Here View shows the information present in the model and controller processes the information.	Angular uses components and directives. Here component is directive with a template.
Language	AngularJS code is written in javascript.	Angular code is written in typescript.
Mobile	AngularJS code is not mobile friendly.	Angular develped applications are mobile browser friendly.
Expression syntax	{{}} are used to bind data between view and model. Special methods, ng-bind can also be used to do the same.	() and [] attributes are used to bind data between view and model.
Dependency Injection	DI is not used.	Hiearchical DI system is used in Angular.
Routing	@routeProvider.when, then are used to provide routing information.	@Route configuration is used to define routing information.
Management	AngularJS project is difficult to manage with increasing size of the source code.	Angular code is better structured, is easy to create and manage bigger applications.

Features of angular 7:

added canLoad interface (lazy loading)

What Is Module"

Module is a collection of components, pipes, guards, services and modules, it is available in @angular/core package.

It provides @ngModule decorator to decorate module class

```
@NgModule ({
  imports: [],
 declarations: [],
 providers: [],
 bootstrap: [AppComponent]
})
```

Why use multiple NgModules?

To organize an application, we are using multiple NgModules.

What is the Purpose of @NgModule?

To register the components and pipes, guards, services we are using @NgModule decorator.

Components:

Component is collection of class, template ang style sheets

1. Class
   Class is a collection of typescript functions and data (variables)
2. Template: template is a HTML file
3. Metadata

```
@Component ({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrls: ['./app.component.scss']
})
```

Data Bindings:

Data binding is a core concept in Angular and allows to define communication between a component and the DOM (view)

4 types of data bindings are there

1. String Interpolation. {{}} data floats from component to view
2. Property Binding. [] data floats from component to view, to bind the angular properties
3. Event Binding. () data floats from view to component, to bind the angular events
4. Two-Way Data Binding. [()] data floats from component to view and view to component simultaneously

template reference variable in angular?

A template reference variable is often a reference to a DOM element within a template. It can also be a reference to an Angular component or directive or a web component. That means you can easily access the variable anywhere in the template.

You declare a reference variable by using the hash symbol (#). Ex: “#firstName”

PIPES:

Pipes are useful for transform data into one form to another form.

Built in pipes:

Date, UpperCase, LowerCase, Currency, Decimal, Percent, number etc.

CustomPipes:

CustomPipe is our own pipe to transform the data

Using **ng g pipe pipe-name** cli command to create custom pipe

In this custom pipe @pipe decorator is provided like this

@Pipe ({

Name:" pipe-name",

Pure: True/False ->True is for pure pipe

False is for impure pipe

})

Pure pipe: pure pipe is only called when Angular detects a change in the value or the parameters passed to a pipe.

Impure pipe: An impure pipe is called for every change detection cycle no matter whether the value or parameter(s) changes.

PipeTransform interface that is implemented by pipes in order to perform a transformation. Angular invokes the **transform method** with the value of a binding as the first argument, and any parameters as the second argument in list form.

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

What Is Parameterizing Pipe?

we can pass any number of parameters to the pipe using a colon (:) and when we do so, it is called Angular Parameterized Pipes

DIRECTIVES:

Directives are classes that add additional behavior to elements in your Angular applications (or)

Directives are used for adding and removing and change the performance of dom elements

1. Components is for creating DOM element
2. Attribute directive directives that change the appearance or behavior of an element, ngStyle, ngClass
3. Structural directive directives that change the DOM layout by adding and removing DOM elements.

   *ngFor, *ngIf, *ngSwitch, *ngIfElse, *ngIfthenElse.

DECORATORS:

Decorator is used to decorate a class with additional features, every decorator contains metadata information, that information we are providing in the form of properties.

1. **Class Decorator**
   @Component, @NgModule
2. **Property Decorator**
   @Input, @Output
3. **Method decorator**
   @HostListner, @HostBinding
4. **Parameter decorator**
   @Inject

@hostListner:

@hostListner is a decorator

@HostListener - will listen to the event emitted by the host element

@HostBinding - will bind the property to the host element, if a binding (property) change, Host Binding will update the host element.

Angular lifecycle hooks?

Life cycle are used for tune our component

In Angular application Every component is providing different phases in life cycle and every page is proving interface with abstract function to execute some statement in that particular page.

Angular calls the lifecycle hook methods in the following sequence those are:

1. **ngOnChanges ()**
   this lifecycle Hock is emitting when any changes are detected in angular application
2. **ngOnInit ()**
   Used to initialize the directive/component after Angular first displays the data-bound properties while setting the directive/component's input properties.
3. **ngDoCheck ()**
   Called during every change detection run, immediately after ngOnChanges () and ngOnInit () to detect and act upon changes that Angular won't do on its own
4. **ngAfterContentInit ()**
   Called once after the first ngDoCheck () to respond after Angular projects external content into the component's/directive's view
5. **ngAfterContentChecked ()**
   Called after the ngAfterContentInit () and every subsequent ngDoCheck () to respond after Angular checks the projected content.

   ngOnChanges ()

This responds when Angular re-sets data-bound input properties.

6. **ngAfterViewInit ()**
   Called once after the first ngAfterContentChecked () to respond after Angular validates the component and its child views.
7. **ngAfterViewChecked ()**
   Called after the ngAfterViewInit () and after every subsequent ngAfterContentChecked ().
8. **ngOnDestroy ()**
   Called just before Angular destroys the directive/component to avoid memory leaks by effective clean up just before destroying the directives

Compiler:

Compiler is for compile the coad into machine understandable language

Types of compilers:

1. **JIT (just in Time)**
   JIT compiles your app in the browser at runtime

2. **AOT (Ahead-of-time)**
   AOT compiles your app and libraries at build time

   The Angular ahead-of-time (AOT) compiler converts your Angular HTML and TypeScript code into efficient JavaScript code during the build phase before the browser downloads and runs that code. Compiling your application during the build process provides a faster rendering in the browser.

JiT	AoT
Loads the application slower than AoT since it needs to compile the application when running for the first time	Loads the page more quickly than the JiT compilation
It download the compiler and compiles the application before displaying.	It doesn't want to download the compiler, since AoT already compiles the code when building the application
Since the code include the compiler code also the bundle size will be higher.	Since it created fully compiled code and its optimized so it bundle size will be half the bundle size compiled by JiT
Suitable in development mode	Suitable in the case of production
Following command use JiT ng build, ng serve	Following command use AoT ng build --aot, ng serve --aot, ng build --prod
Template binding errors can be viewed at the time of displaying the application.	Template binding errors are shown at the time of building.

ANGULAR ELEMENTS:

Angular elements are the regular angular components packed as a web component for use in the DOM

This angular element we can use in outside of our angular application

Services:

1.What Are Angular Services?

Service is used to share the common functionalities throughout components in application.

2.How to Setup and Create Angular Services?

We can create the services by injectable decorator, and this decorator is available in @angular/core package.

3.What Is Angular Singleton Service?

A singleton is a class that allows only a single instance of itself to be created and gives access to that created instance.

RESTAPI:

Representational state transfer

Using http clint module we can create rest api's in angular project

Using rest, we can perform CRUD operations on server

Restapi methods:

Get: get method is to read the resource

Post: is to create a source in server

Put: is to update existing fields in server

Delete: to delete source on server

Patch: to update existing fields and also, we can add new fields in server

HTTP Module:

Most front-end applications need to communicate with a server over the HTTP protocol, in order to download or upload data and access other back-end services. Angular provides a simplified client HTTP API for Angular applications, the HttpClient service class in @angular/common/http.

The HTTP client service offers the following major features.

- The ability to request typed response objects.
- Streamlined error handling.
- Testability features.
- Request and response interception.

HTTP Interceptors:

interceptors are used for inspect and transform HTTP requests, and HTTP response object at a time

Interceptors can perform a variety of implicit tasks, from authentication to logging, in a routine, standard way, for every HTTP request/response.

It will give the intercept method to write our coad.

```
intercept (req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
   return next. handle(req);
 }
```

Uses of http interceptor:

- Using http interceptor, we can modify the request object (http Headers)
- Handle http errors in a single place
- Trace the request

ROUTINGS:

Routes are useful for render the pages inside the angular application.

Routes are used to navigate from one page to another page in angular application

Wildcard routs: { path: '**', component: }

when users attempt to navigate to a part of your application that does not exist.

routerLink: it is an attribute. Set the value of the attribute to the component to show when a user clicks on each link.

<router-outlet>: it is an element. this element informs Angular to update the application view with the component for the selected route.

RouterLinkActive: it is a directive;

Router link status:

Tracks whether the linked route of an element is currently active, and allows you to specify one or more CSS classes to add to the element when the linked route is active.

Five types of outer guards in angular:

1. CanActivate
2. CanActivateChild
3. CanLoad (Lazy loading)
4. CanDeactivate
5. Resolve

1. **CanActivate:** checks the navigation before the component loaded
2. **CanActivateChild:** Check the children navigation before the component loads
3. **CanDeactivate:** checks the navigation status from the current route
   E.g. leaving the partially form field
4. **Resolve:** Resolve loads/retrieves Data before the router is activated
5. **CanLoad:** to perform lazy loading
   Check to see if a user can route to a module the lazy loaded
6.

Router Events:

Router events is to track the status of a route.

several router events are there un angular there are

1. **NavigationStart**: Navigation starts.
2. **RouteConfigLoadStart:** Before the router lazy loads a route configuration.
3. **RouteConfigLoadEnd**: After a route has been lazy loaded.
4. **RoutesRecognized**: When the router parses the URL and the routes are recognized.
5. **GuardsCheckStart**: When the router begins the guards phase of routing.
6. **ChildActivationStart**: When the router begins activating a route's children.
7. **ActivationStart**: When the router begins activating a route.
8. **GuardsCheckEnd**: When the router finishes the guards phase of routing successfully.
9. **ResolveStart**: When the router begins the resolve phase of routing.
10. **ResolveEnd**: When the router finishes the resolve phase of routing successfuly.
11. **ChildActivationEnd**: When the router finishes activating a route's children.
12. **ActivationEnd**: When the router finishes activating a route.
13. **NavigationEnd**: When navigation ends successfully.
14. **NavigationCancel**: When navigation is canceled.
15. **NavigationError**: When navigation fails due to an unexpected error.
16. **Scroll**: When the user scrolls.
17.

RXJS Library:

Stands for **Reactive Extensions for JavaScript**, it's a library that gives us an implementation of Observables for JavaScript.

Observables:

Observables can emit number of values asynchronously over a period of time

The observable can emit/ provide the data only how subscribe to this

Observable is nothing but a HTTP response from server, by default the HTTP response (observable) is not ready to use in application

After receiving the observable, we have to convert that observable into array format, this array we use in our application

RXJS Operators?

AREA	OPERATORS
Creation	: from, fromEvent, of
Combination	: combineLatest, concat, merge, startWith, withLatestFrom, zip
Filtering	: debounceTime, distinctUntilChanged, filter, take, takeUntil.
Transformation	: bufferTime, concatMap, map, mergeMap, scan, switchMap.
Utility	: tap
Multicasting	: share

Map operator:

Map operator provides a transformation method for every observable source transformation method is execute and transform the observable value into new value, then this new value will emit to the observer.

```
srcArray <observable> = from ([1, 2, 3, 4]);
```

filter operator:

RxJS filter is used to filter values emitted by source Observable on the basis of given predicate. If the condition returns true, filter will emit value obtained from source Observable otherwise not. Find some usability of RxJS filter operator.

```
const source = from ([
{name: 'Joe', age: 31},
{name: 'Bob', age: 25}
]);
//filter out people with age under 30
const example = source. pipe (filter (person => person.age >= 30));
```

pipe ():

You can use pipes to link operators together. Pipes let you combine multiple functions into a single function.

```
const squareOdd = of (1, 2, 3, 4, 5)
  . pipe (
    filter (n => n % 2 !== 0),
    map (n => n * n)
```

);

Subject in RXJS:

A Subject is a Special type of Observable that allows value to be multicasted to many Observers.
Subject are like event emitters.
No Initial Value

Multicasting?
It will provide an updated value to all observers

Behaviour Subject?

Behaviour Subject is similar to subject but only difference is that we can set the initial value in behaviour subject.

Promises:
Promises are most commonly used to handle HTTP requests. In this model, you make a request and then wait for a single response. You can be sure that there won't be multiple responses to the same request

Observables	Promises
Emit multiple values over a period of time.	Emit a single value at a time.
Are lazy: they're not executed until we subscribe to them using the subscribe() method.	Are not lazy: execute immediately after creation.
Have subscriptions that are cancellable using the unsubscribe() method, which stops the listener from receiving further values.	Are not cancellable.
Provide the map for forEach, filter, reduce, retry, and retryWhen operators.	Don't provide any operations.
Deliver errors to the subscribers.	Push errors to the child promises.

Dependency injection:

Dependency Injection (DI) is a core concept of Angular. DI allows a class receive dependencies from another class. Most of the time in Angular, dependency injection is done by injecting a service class into a component or module class.

```
@Injectable ({
        providedIn: 'root',
})
```

FORMS:

Angular forms are used to handle the user input data, and display the error massages according to the input values.

Two types of forms are there

1. Template-driven forms
2. Reactive forms

LocalStorage VS session storage:

sessionStorage is similar to localStorage to storage the data on browser, the difference is that while data in localStorage doesn't expire until we manually delete data on browser, but in sessionStorage data is cleared when the page session ends (or) closing the tab.

JavaScript:

JavaScript is a very powerful client-side scripting language. It is lightweight and most commonly used as a part of web pages, for dynamically perform client-side validations in browser.

1. JavaScript is a lightweight, interpreted programming language.
2. Designed for creating network-centric applications.
3. Complementary to and integrated with Java.
4. Complementary to and integrated with HTML.
5. Open and cross-platform

Advantages of JavaScript

The merits of using JavaScript are −

1. **Less server interaction** − You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
2. **Immediate feedback to the visitors** − They don't have to wait for a page reload to see if they have forgotten to enter something.
3. **Increased interactivity** − You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
4. **Richer interfaces** − You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

JavaScript	TypeScript

Java script is not fully object-oriented programming	Type script is fully object-oriented programming
It is loosely typed script. it does not support types	It is tightly typed script. it does support types
It is a interpreted scripting language	It is a compiled scripting language
Javascript coad is heavy	Typescript coad is less
Javascript does not prompt all types of errors	typescript does not prompt all types of errors

Array methods:

1. The **push ()** method adds one or more elements to the end of an array and returns the new length of the array.
2. The **pop ()** method removes the **last** element from an array and returns that element. This method changes the length of the array.
3. The **unshift ()** method adds one or more elements to the beginning of an array and returns the new length of the array.
4. The **shift ()** method removes the **first** element from an array and returns that removed element. This method changes the length of the array.
5. The **slice ()** method returns a shallow copy of a portion of an array into a new array object selected from start to end (end not included) where start and end represent the index of items in that array. The original array will not be modified.
6. The **splice ()** method changes the contents of an array by removing or replacing existing elements and/or adding new elements
7. The **find ()** method returns the value of the first element in the provided array that satisfies the provided testing function. If no values satisfy the testing function, undefined is returned.
8. The **filter ()** method **creates a new array** with all elements that pass the test implemented by the provided function.
9. The **forEach ()** method executes a provided function once for each array element.
10. The **map ()** method **creates a new array** populated with the results of calling a provided function on every element in the calling array.
11. The **reduce ()** method executes a **reducer** function (that you provide) on each element of the array, resulting in single output value.

Difference between callback, promises and Asyc/await:

1. Callback: a callback function is a function that is to be executed after another function has finished execution
2. Promise: a promise is an object that may produce single value some times in the feature
3. Asyc/await: Asyc and await are the keywords
   Asyc => when we place a asyc keyword before a function then that function returns a promise

await => it must be used with in the asyc block only it makes that coad wait until promise return a result

Arrow Functions in ES6.

Arrow function is an anonymous function. It is passed as an argument for another function it will get executed after execute main function

() => {} arrow function

Promises in ES6.

Var, Let and Const.

Var:

Var is a function scope

var variables can be updated and re-declared within its scope

let:

let is a block we can use this variable with in the blank only

let variables can be updated but not re-declared.

Const:

const variables can neither be updated nor re-declared.

They are all hoisted to the top of their scope. But while var variables are initialized with undefined, let and const variables are not initialized.

Hoisting:

Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution.

Inevitably, this means that no matter where functions and variables are declared, they are moved to the top of their scope regardless of whether their scope is global or local.

Event bubbling and event capturing:

Event Capturing is opposite to event bubbling, where in event capturing, an event moves from the outermost element to the target. Otherwise, in case of event bubbling, the event movement begins from the target to the outermost element in the file.

Event Bubbling: event flows in this order from bottom to top.

Event Capturing: event flows in the order from top to bottom.

Difference between call and apply method in javascript

call () /apply () to invoke the function immediately.

Note that when using the apply() function the parameter must be placed in an array.

Call () accepts both an array of parameters and a list of parameters itself.

Both are great tools for borrowing functions in JavaScript.

Ex:

```
function theFunction (name, profession) {
   console.log ("My name is " + name + " and I am a " + profession +".");
}
theFunction ("John", "fireman");
theFunction.apply(undefined, ["Susan", "school teacher"]);
theFunction.call(undefined, "Claude", "mathematician");
theFunction.call(undefined, ... ["Matthew", "physicist"]); // used with the spread operator
```

Bind method:

The Bind method returns a new function, allowing you to pass in this array and any number of arguments. Use it when you want that function to later be called with a certain context like events.

BOOTSTRAP

@ How many grid structures we have?

Bootstrap Uses a 12 Column **Grid**

The **Bootstrap grid** system is based on a 12-column **grid**.

@ Why we have 12 grid structure?

1. **Introduction of ANGULAR**
2. **Typescript**
3. **Node Modules**
4. **Angular Framework Architecture**
5. **Components, Modules**
6. **Decorators**
7. **Organize Routes using Modules**
8. **Injectable Services 9. Directives & Custom Directives.**
9. **Data Bindings.**
10. **Events**
11. **JSON with ANGULAR**
12. **Reactive form and Template Driven Forms**
13. **Bind form controls to a model**
14. **Custom Pipe**
15. **Routing (Single Page Application)**
16. **Integration with Server Side Script**
17. **Input and Output Data.**
18. **Express JS API Services**
19. **Mongo DB integration.**
20. **NPM Commands**
21. **Observable**
22. **Event Emitter**
23. **Microsoft Visual Studio Code IDE**
24. **Angular CLI**
25. **ANGULAR Communication with NODE JS**
26. **MEAN Stack Programming**
27. **(DELETE, INSERT, SELECT, UPDATE with Mongo DB)**
28. **Angular with Bootstrap**
29. **Angular with AJAX**
30. **Angular with jQuery**
31. **Search Plugin**
32. **Pagination with Data Grid**
33. **Sorting Data Using Angular SOFT.**
34. **Angular Lifecycle Hooks**
35. **Lazy loading in Angular**
36. **forRoot and forChild**

37. **Local Storage**
38. **@ViewChild**
39. **Http and HttpClient**
40. **Authentication Guards**
41. **AOT and JIT**
42. **Material Design**
43. **Mini Applications with Angular**
44. **Interview Questions.**
45. **NGX Library**
46. **AG grid**
47. **Promises**
48. **Role based Router guards**
49. **JSON web tokens**
50. **Http Interceptors**
51. **Deployment of Angular Application in live servers**
52. **Angular Testing - (using Karma, Jasmine, Protractor)**

Fundamentals:

Angular CLI?

Ng Module?

Components?

4. Class
5. Template
6. metadata

Template syntax?

1. Interpolation
2. Property Binding
3. Event Binding
4. Two-way Binding

5. Pipes
6. Template Reference Variable

Directives?

1. Component Directive
2. Structural Directive
    - ngIf
    - ngFor
    - ngIfElse
    - ngSwitch
3. Attribute Directive
    - ngClass
    - ngStyle

Decorators?

1. Class Decorator
    - @Component
    - @NgModule
2. Property Decorator
    - @Input
    - @Output
3. Method decorator
    - @HostListner
    - @HostBinding
4. Parameter decorator
    - @Inject

Lifecycle Methods?

ADVANCED:

Services?

1. Dependency Injection
2. Providing services

Observables?

1. Creating Observable
2. Subscribing to observable
3. Executing observable
4. Disposing observable
5. operators

HTTP client?

1. Http Client
2. Get, post, put, delete and patch
3. Typed response
4. Error Handling
5. interceptors

Forms?

1. Template Driven Forms
    - Forms Module
    - NgForm
    - NgModel
    - NgModelGroup
    - Tracing State & Validate
    - ngSubmit
2. Reactive Forms
    - ReactiveFormsModule
    - FormControl
    - FormGroup
    - setValue & patchValue
    - FormBuilder Service
    - Validator Function
    - FormArray
    - ngSubmit
    -

Routing?

1. Router Module
2. Configuring Routers
3. RouterOutlet
4. routerLink
5. Wildcard Router
6. Redirecting Routes
7. Router parameters
8. Nested Routes
9. Relative Paths
10. Lazy Loading Routes
11. Route Guards

Animation?

ECOSYSTEM:

State management?

NgRx

UI library?

Angular materials

Server side rendering?

Angular universe

Testing?

Jasmin, karma, protractor

Miscellaneous?

I18n, accessibility