**What is Angular Framework?**

Angular is a **TypeScript-based open-source** front-end platform that makes it easy to build applications with in web/mobile/desktop.

**What are directives in Angular ?**

Directives in Angular is a js class, which is declared as @directive. Directives add behaviour to an existing DOM element or an existing component instance. We have 3 directives in Angular. The directives are listed below −

Component Directives

These form the main class having details of how the component should be processed, instantiated and used at runtime. e.g. ngModel,ngStyle

Structural Directives

A structure directive basically deals with manipulating the dom elements. Structural directives have a \* sign before the directive. For example, \*ngIf and \*ngFor.

Attribute Directives

Attribute directives deal with changing the look and behavior of the dom element. You can create your own directives as shown below.

**Explain the importance of NPM and Node\_Modules folder ?**

You can think of the node\_modules folder like a cache for the external modules that your project depends upon. When you npm install them, they are downloaded from the web and copied into the node\_modules folder and Node. js is trained to look for them there when you import them (without a specific path).

**Explain the importance of Package.json file in Angular ?**

It is the first file your browser looks for, to find all the information of your web app. package.json provides a simple way to keep track of packages that are being used in application.

**What is typescript and why do we need it ?**

[TypeScript](http://www.typescriptlang.org/) is a superset of JavaScript which primarily provides optional static typing, classes and interfaces.

**Explain importance of Angular CLI ?**

Angular CLI stands for Angular Command Line Interface. As the name implies, it is a command line tool for creating angular apps.

**Explain the importance of Component and Modules ?**

In Angular, Modules are the collection of the Components, Service, directives, and Pipes which are related such that they can be combined to form a module. - @NgModule

The component which controls the View or the template that we use. - @Component

### What is a module?

Modules are logical boundaries in your application and the application is divided into separate modules to separate the functionality of your application. Lets take an example of **app.module.ts** root module declared with **@NgModule** decorator as below,

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

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

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

@NgModule ({

imports: [ BrowserModule ],

declarations: [ AppComponent ],

bootstrap: [ AppComponent ],

providers: []

})

export class AppModule { }

The NgModule decorator has five important(among all) options

* 1. The **imports** option is used to **import other dependent modules**. The BrowserModule is required by default for any web based angular application
  2. The **declarations** option is used to define **components** in the respective module
  3. The **bootstrap** option tells Angular which **Component** to **bootstrap** in the application
  4. The **providers** option is used to configure set of **injectable** objects that are available in the injector of this module.
  5. The entryComponents option is a set of components dynamically loaded into the view.

**What is a decorator in Angular ?**

Decorators are functions that allow a **service, directive,** or **filter** to be modified before it is used.

Class decorators, such as @Component and @NgModule.

Property decorators for properties inside classes, such as @Input and @Output

**Explain the four types of Data bindings in Angular ?**

One-way Data Binding

In one-way data binding, data flows only in one direction i.e from the models to the views. As mentioned earlier, one-way data binding in Angular can be of three types i.e Interpolation, Property binding, and Event binding.

**Interpolation Binding**

Interpolation binding is used to return HTML output from TypeScript code i.e. from the components to the views.

<h1>{{title}}</h1>

Property binding

Property binding is used to bind the data from property of a component to DOM elements. It is denoted by [].

<input type="text" [value]="userName">

**Event Binding**

Events are actions like mouse click, double click, hover or any keyboard and mouse actions. If a user interacts with an application and performs some actions, then event will be raised. It is denoted by either parenthesis () or on-.

<button (click)="showData($event)">Click here</button>

Class binding

It is used to bind the data from component to HTML class property.

<h2 [class]="successClass">Shivam Bagi</h2>

**Two-way Binding**

Angular allows two-way data binding that will allow your application to share data in two directions. For Two-Way binding to work, you need to enable the ngModel directive. This is done by adding the FormsModule to imports array in the AppModule.

<input [(ngModel)]="name" type="text">

{{name}}

export class TwowaybindingComponent implements OnInit {

public name = "";

}

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

**Explain architecture of Angular ?**

Both of MVC and MVVM.

**How to implement routing in Angular ?**

It can be included by using the CLI command ***ng new routing-app –routing.***

Provide paths and components in routing module. Can use id. Also child routes can be done.

Routing modules can also be placed inside a components but have to specify ***forChild*** in ***@NgModule*** in that components routing module.

router-outlet

routerLink instead of href

**Explain Lazy Loading ?**

Lazy loading is a technique in Angular that allows you to load JavaScript components asynchronously when a specific route is activated. It improves the speed of the application load time by splitting the application into several bundles. When the user navigates through the app, the bundles are loaded as required.

When implementing Lazy Loading, Angular calls a function inside the loadChildren parameter when the route is defined.

const routes: Routes = [

{

path: 'home',

component: HomeComponent

},

{

path:'auth',

loadChildren: () => import('./auth/auth.module').then(m => m.default)

}

];

**When would you use eager module loading?**

Feature modules under Eager Loading would be loaded before the application starts. This is the default module-loading strategy.

Case 1: Small size applications. In this case, it’s not expensive to load all modules before the application starts, and the application will be faster and more responsive to process requests.

Case 2: Core modules and feature modules that are required to start the application. These modules could contain components of the initial page, interceptors (for authentication, authorization, and error handling, etc.), error response components, top-level routing, and localization, etc. We just have to eagerly load these modules to make the application function properly despite the application size.

**Define Services ?**

Service is a broad category encompassing any value, function, or feature that an application needs. A service is typically a class with a narrow, well-defined purpose. It should do something specific and do it well. Angular distinguishes components from services to increase modularity and reusability. It is a class.

@Injectable({ **@Injectable() decorator** is a must if you

  providedIn: 'root' want to inject a service into another service

})

**Differentiate between ng serve and ng build ?**

The ng serve command is intentionally for fast, local and iterative developments and also for builds, watches and serves the application from a local CLI development server. The ng serve builds artifacts from memory instead for a faster development experience.

The ng build command is intentionally for building the apps and deploying the build artifacts. The ng build command generates output files just once and doesn’t serve them.

**Explain the --prod parameter in ng build ?**

Set the build configuration to the production target.

**Explain ViewChild and ViewChildren?**

Both ViewChild and ViewChildren are used to communicate between the components to access the data.

***@ViewChild*** and ***@ViewChildren*** are the types of decorators used to access the child component class and its different properties into the parent component. It's similar to the inheritance.

**ViewChild**

Suppose we have a ChildComponent having 2 methods increment and decrement.

We can use the ViewChild in the parent component to get the reference to the ChildComponent.

Now in parent component

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

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

export **class** AppComponent {

  title = 'Parent calls an @ViewChild()';

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

  increment() {

**this**.child.increment();

  }

  decrement() {

**this**.child.decrement();

  }

}

**ViewChildren**

If we want to retrieve values from multiple components then we use ViewChildren.

@ViewChildren(ClockComponent) myValue:QueryList<ClockComponent>;

**Why do we need Template reference variables?**

Template variables help you use data from one part of template to another part of the same template. It can be used to get value from textbox or any other such inputs and use them or bind them to other views or components. You use the hash symbol, #, to declare a template variable. It can be then used or passed in HTML actions to functions.

<input #inputelement type="text">

<button (click)="onClick(inputelement.value)">Log</button>

**What is ContentProjection?**

We know how to use [@Input](https://www.tektutorialshub.com/angular/angular-input-output-eventemitter/) decorator to pass data from the parent component to the child component. But it is only limited to data. We cannot use that technique to pass the content which includes the HTML elements, CSS, etc to the child component. To do that we have to make use of content projection.

Single Slot

@Component({

selector: 'app-fancybtn',

template: `

     <button>

       <ng-content></ng-content>

     </button> `

})

<app-fancybtn>Click Me</app-fancybtn>

<app-fancybtn><b>Submit</b></app-fancybtn>

Multi Slot

@Component({

  selector: 'app-card',

  template: `

    <div **class**="card">

    <div **class**="header">

      <ng-content select="header" ></ng-content>

    </div>

    <div **class**="content">

      <ng-content select="content" ></ng-content>

    </div>

    <div **class**="footer">

      <ng-content select="footer" ></ng-content>

    </div>

    </div>

  `

})

<app-card>

<header><h1>Angular</h1></header>

<content>One framework. Mobile & desktop.</content>

<footer><b>Super-powered by Google </b></footer>

</app-card>

**Explain Content projection Slot?**

We use the ng-content element to designate a spot in the template of the child component.

**What is ContentChild and ContentChildren?**

The ContentChild & ContentChildren is very similar to the [ViewChild & ViewChildren](https://www.tektutorialshub.com/angular/understanding-viewchild-viewchildren-querylist-in-angular/). We use the [ViewChild](https://www.tektutorialshub.com/angular/understanding-viewchild-viewchildren-querylist-in-angular/)  or  [ViewChildren](https://www.tektutorialshub.com/angular/understanding-viewchild-viewchildren-querylist-in-angular/) to Query and get the reference of any DOM element in the Component. The DOM element can be an HTML element, Child Component or directive, etc. But, We cannot use the [ViewChild](https://www.tektutorialshub.com/angular/understanding-viewchild-viewchildren-querylist-in-angular/)  or  [ViewChildren](https://www.tektutorialshub.com/angular/understanding-viewchild-viewchildren-querylist-in-angular/) to get the reference to the template inserted using the Content projection.

**ViewChild vs ViewChildren vs ContentChild vs ContentrChildren?**

As the name suggests, @ContentChild and @ContentChildren queries will return directives existing inside the <ng-content></ng-content> element of your view, whereas @ViewChild and @ViewChildren only look at elements that are on your view template directly.

**Explain events and sequence of component life cycle ?**

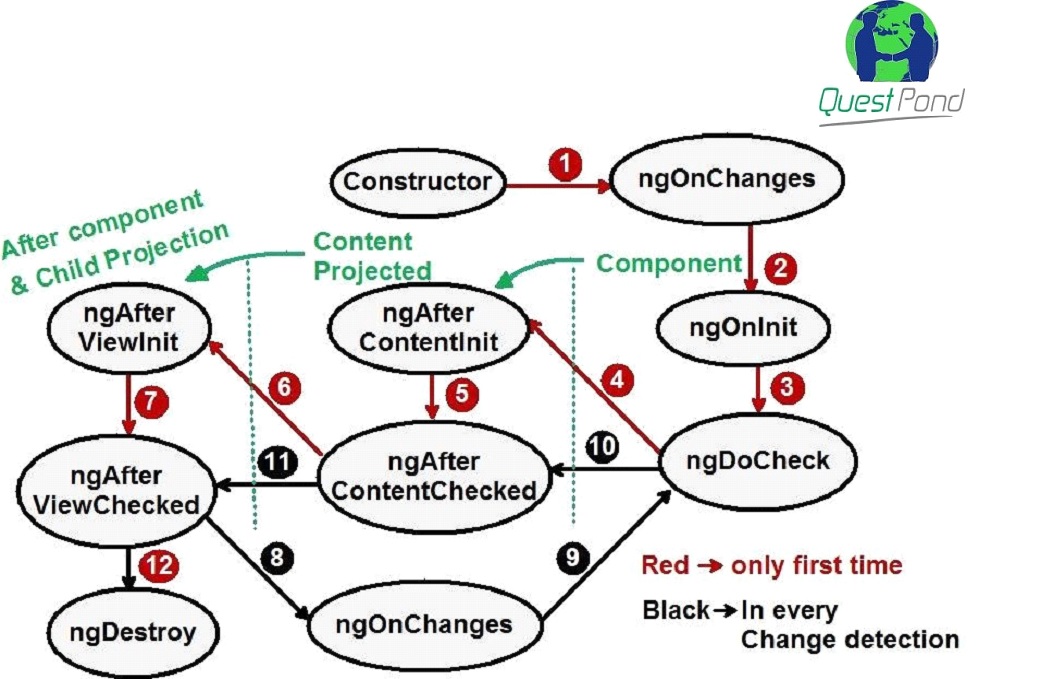
|  |  |
| --- | --- |
| **OnInit** | **Called on initialization** |
| **OnChanges** | **Called when the input properties have changed** |
| **DoCheck** | **Developer’s custom change detection. This is for the detection and to act on changes that Angular can't or won't detect on its own.** |
| **Destroy** | **Called before the component is destroyed** |
| **AfterContentInit** | **Called when the component’s content ngContent is initialized** |
| **AfterContentChecked** | **Called when the component’s content is updated or checked for updates** |
| **AfterViewInit** | **Called when the component’s projected view has been initialized** |
| **AfterViewChecked** | **Called after the projected view has been checked** |
| **OnDestroy** | **This is the cleanup phase just before Angular destroys the directive/component.** |

These are the hooks for components or directives, in call order:

1. constructor() - simple values are initialized
2. OnInit - called after Angular has initialized all data-bound properties of a directive. Define an ngOnInit() method to handle any additional initialization tasks. Complex values are initialized using API calls,etc.
3. DoCheck
4. OnChanges
5. OnDestroy

**What is the difference between constructor and ngOnInit?**

TypeScript classes has a default method called constructor which is normally used for the initialization purpose. Whereas ngOnInit method is specific to Angular, especially used to define Angular bindings. Even though constructor getting called first, it is preferred to move all of your Angular bindings to ngOnInit method. In order to use ngOnInit, you need to implement OnInit interface as below,



export class App implements OnInit{

constructor(){

//called first time before the ngOnInit()

}

ngOnInit(){

//called after the constructor and called after the first ngOnChanges()

}

}

**What is HttpClient and its benefits?**

Most of the Front-end applications communicate with backend services over HTTP protocol using either XMLHttpRequest interface or the fetch() API. Angular provides a simplified client HTTP API known as **HttpClient** which is based on top of XMLHttpRequest interface. This client is avaialble from @angular/common/http package. You can import in your root module as below,

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

**How to make HTTP calls using Angular ?**

**HTTPClientModule** is used to request data from servers.

* First you have to import the HTTPClientModule first in AppModule and add in the imports array.

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

* Now to use HTTP Service in the service class we declare it as dependency in the constructor of service class. Also you have to import the **HTTPClient** and pass an object of **HTTPClient** in the constructor.

import {HttpClient} from '@angular/common/http'

export class EmployeeService {

private url: string = "/assets/data/employees.json"

  getEmployees() {

    return this.http.get(this.url);

  }

  constructor(private http: HttpClient) { }

}

* Finally use this service by subscribing

**What are observables and observers and subscribe?**

The HTTP module uses observables to handle AJAX requests and responses. Angular’s HttpClient returns observables from HTTP method calls. For instance, http.get(‘/api’) returns an observable.

Trying to explain with a really simple example:-

1. **Observable** is like a youtube channel of someone else. (( It uploads new videos(data) from time to time, **so it is a data source** for you))

*An observable is a function that creates an observer and attaches it to the source where values are expected, for example, clicks, mouse events from a dom element or an Http request, etc.*

1. Your youtube account is an **Observer**

*It is an object with next(), error() and complete() methods, that will get called when there is interaction to the with the observable i.e. the source interacts for an example button click, Http request, etc.*

1. Your youtube account **(Observer)** can only get notifications about whether someone else's youtube channel **(Observable)** has uploaded a new video **(has new data)** or made a livestream **(new event)** only if you have **subscribed** to that channel

*When the observable is created, to execute the observable we need to subscribe to it. It can also be used to cancel the execution.*

**(Observer subscribes Observable to listen for new data/any event)**

where observable is a data source, subscribe is like a method/function , Observer is generally on your side

**How to handle errors when HTTP fails ?**

Whenever the error occurs in an HTTP operation, the Angular wraps it in an httpErrorResponse Object before throwing it back. We catch the httpErrorResponse either in our component class or in the data service class or globally. The Global HTTP error handling is done using the Angular HTTP Interceptor.

In component-

**this**.githubService.getReposCatchError(**this**.userName)

      .subscribe(

        (response) => {                           *//Next callback*

          console.log('response received')

**this**.repos = response;

        },

        (error) => {                              *//Error callback*

          console.error('error caught in component')

**this**.errorMessage = error;

**this**.loading = **false**;

*//throw error;   //You can also throw the error to a global error handler*

        }

      )

The subscribe method has three callback arguments.

.subscribe(success, error, completed);

In Service-

We can also catch errors in the service, which makes the HTTP Request using the catchError Operator as shown below. Once you handle the error, you can re-throw it back to the component for further handling.

getRepos(userName: **string**): Observable<repos[]> {

**return** **this**.http.**get**<repos[]>(**this**.baseURL + 'usersY/' + userName + '/repos')

      .pipe(

        catchError((err) => {

          console.log('error caught in service')

          console.error(err);

*//Handle the error here*

**return** throwError(err);    *//Rethrow it back to component*

        })

      )

  }

Other

1) You can use try catch blocks to catch errors in classes, this will not allow it give the error In console.

2) Create a GlobalErrorHandlerService  which implements the ErrorHandler  
Then, override the handleError(error) method and handle the error.

Next, register the GlobalErrorHandlerService in the Application root module using the token ErrorHandler.

Catch error globally using HTTP Interceptor

**How to pass data between components ?**

Refer notes component interaction @Input, @Output, EventEmitter, ViewChild, ContentChild

**How to pass during routing ?**

The Angular can pass data to Route in several ways.

1. Using URL or [Route Parameter](https://www.tektutorialshub.com/angular/angular-passing-parameters-to-route/)

{ path: 'static', component: StaticComponent, data :{ id:'1', name:"Angular"}},

Access using

ngOnInit() {

**this**.activatedroute.data.subscribe(data => {

**this**.product=data;

      })

}

1. The [Optional Parameter or Query Strings](https://www.tektutorialshub.com/angular/angular-pass-url-parameters-query-strings/)  
   <a [routerLink]="['dynamic']" [state]="{ id:1 , name:'Angular'}">**Dynamic** Data</a>

Access using

**this**.router.getCurrentNavigation().extras.state

1. Using URL Fragment
2. Static data using the data property
3. Dynamic data using state object

**What is the need of Angular Pipes? Types? How to create Custom Pipes?**

Refer notes Pipes. A pipe takes in data as input and transforms it to a desired output.

**Whats the full form of RxJs?**

RxJs stands for Reactive extensions for JavaScript. RxJs helps to handle asynchronous data stream with ease.

Assume you have an entity which is streaming async data, now this entity which is streaming async data can be a HTTP response , Port which is streaming data , Timer emitting data and so on. This Async stream is coming in undecided intervals like stream of data. As a developer you would like to listen to this stream , run some logic on these stream and so on.RxJs makes this task easy. RxJS is a library for composing asynchronous and callback-based code in a functional, reactive style using Observables. Many APIs such as HttpClient produce and consume RxJS Observables and also uses operators for processing observables.

**Explain concept of operators with sample code.**

An operator is simply a method that acts on an Observable and changes the stream in some way. Operators are by nature immutable. This immutability makes the code easier to test and reason about.

Of,from,map examples of operators.

**Pipe**

The pipe method accepts operators such as filter, map, as arguments. Each argument must be separated by a comma.

**Name some rxJs Operators?**

There are types like Creation operators, Joining Operators(concat,merger), filtering(filter), transformation(map)

* **Map :-** Transforms data in a observable in to a different format. .pipe(map((user) => { } ))
* **Filter :-** Allows data which meets conditions. http.get(“ ---- ”).pipe(filter(x => x % 2 == 0))
* **Merge :-** This operator will combine multiple Observables into one. So if one of the observables emit a value the combined one will emit as well.
* **Concat :-** only when observable completes, it will start with the next observable.
* **From :-** This operator will turn array, promise or iterable into an observable.

**Differentiate between RxJs and Promises?**

|  |  |
| --- | --- |
| RxJs | Promise |
| Observable return stream of data. | Promise return single value. |
| You can subscribe and unsubscribe stream. | You cannot cancel a promise. |

**What is the difference between promise and observable?**

| **Observable** | **Promise** |
| --- | --- |
| Declarative: Computation does not start until subscription so that they can be run whenever you need the result | Execute immediately on creation |
| Provide multiple values over time | Provide only one |
| Subscribe method is used for error handling which makes centralized and predictable error handling | Push errors to the child promises |
| Provides chaining and subscription to handle complex applications | Uses only .then() clause |

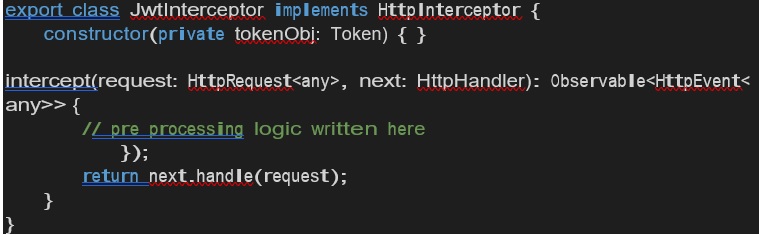
**What are interceptors in Angular?**

It Intercepts and handles an HttpRequest or HttpResponse. The Interceptor can be useful for adding custom headers to the outgoing request, logging the incoming response, etc.

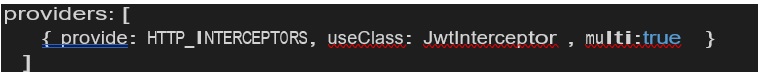
**How to implement interceptors?**

Implementing interceptor is a two-step process: -

Step 1 :- First create the class which has the pre-processing logic and this class should implement "HttpInterceptor" interface. In the "intercept" method we need to write pre-processing logic which will fire before HTTP call is executed.



Step 2 :- Dependency inject the pre-processing class at module level using providers as shown in the below code.



**What are Annotations or MetaData ?**

Metadata is used to decorate a class so that it can configure the expected behavior of the class. The metadata is represented by decorators

1. **Class decorators**, e.g. @Component and @NgModule
2. **Property decorators** Used for properties inside classes, e.g. @Input and @Output
3. **Method decorators** Used for methods inside classes, e.g. @HostListener
4. **Parameter decorators** Used for parameters inside class constructors, e.g. @Inject, Optional

**What is a bootstrapping module?**

Every application has at least one Angular module, the root module that you bootstrap to launch the application is called as bootstrapping module. It is commonly known as AppModule.

**What are custom elements?**

Custom elements (or Web Components) are a Web Platform feature which extends HTML by allowing you to define a tag whose content is created and controlled by JavaScript code. Custom elements bootstrap themselves - they start automatically when they are added to the DOM, and are automatically destroyed when removed from the DOM. Once a custom element is added to the DOM for any page, it looks and behaves like any other HTML element, and does not require any special knowledge of Angular terms or usage conventions.

<my-popup message="Use Angular!"></my-popup>

**Explain how custom elements works internally?**

* App registers custom element with browser: Use the createCustomElement() function to convert a component into a class that can be registered with the browser as a custom element.
* App adds custom element to DOM: Add custom element just like a built-in HTML element directly into the DOM.
* Browser instantiate component based class: Browser creates an instance of the registered class and adds it to the DOM.
* Instance provides content with data binding and change detection: The content with in template is rendered using the component and DOM data. The flow chart of the custom elements functionality would be as follows,

**What is router outlet?**

The RouterOutlet is a directive from the router library and it acts as a placeholder that marks the spot in the template where the router should display the components for that outlet. Router outlet is used like a component,

<router-outlet></router-outlet>

**What are router links?**

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

<h1>Angular Router</h1>

<nav>

<a routerLink="/todosList" >List of todos</a>

<a routerLink="/completed" >Completed todos</a>

</nav>

<router-outlet></router-outlet>

**What is router state?**

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

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

class MyComponent {

constructor(router: Router) {

const state: RouterState = router.routerState;

const root: ActivatedRoute = state.root;

const child = root.firstChild;

const id: Observable<string> = child.params.map(p => p.id);

//...

}

}

**What is Activated route?**

The ActivatedRoute is the API that holds observable data that you can use to react to events with. You can subscribe to specific observable properties to get data from it asynchronously. The ActivatedRoute also contains ActivatedRouteSnapshot. The ActivatedRouteSnapshot holds the same activated route data but is a static property.

**What is the purpose of Wildcard route?**

If the URL doesn't match any predefined routes then it causes the router to throw an error and crash the app. In this case, you can use wildcard route. A wildcard route has a path consisting of two asterisks to match every URL.

For example, you can define PageNotFoundComponent for wildcard route as below

{ path: '\*\*', component: PageNotFoundComponent }

**What are different types of compilation in Angular?**

Angular offers two ways to compile your application,

1. Just-in-Time (JIT)
2. Ahead-of-Time (AOT)

**What is JIT?**

Just-in-Time (JIT) is a type of compilation that compiles your app in the browser at runtime. JIT compilation is the default when you run the ng build (build only) or ng serve (build and serve locally) CLI commands. i.e, the below commands used for JIT compilation,

ng build

ng serve

**What is AOT?**

Ahead-of-Time (AOT) is a type of compilation that compiles your app at build time. For AOT compilation, include the --aot option with the ng build or ng serve command as below,

Fast Rendering as precompiled code,Detects template errors earlier since precompiled

ng build --aot

ng serve --aot

**Note:** The ng build command with the --prod meta-flag (ng build --prod) compiles with AOT by default.

**Why do we need compilation process?**

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.

**What is the purpose of any type cast function?**

You can disable binding expression type checking using $any() type cast function(by surrounding the expression). In the following example, the error Property contacts does not exist is suppressed by casting user to the any type.

template:

'{{ $any(user).contacts.email }}'

**How do you pass headers for HTTP client?**

You can directly pass object map for http client or create HttpHeaders class to supply the headers.

constructor(private \_http: HttpClient) {}

this.\_http.get('someUrl',{

headers: {'header1':'value1','header2':'value2'}

});

(or)

let headers = new HttpHeaders().set('header1', headerValue1); // create header object

headers = headers.append('header2', headerValue2); // add a new header, creating a new object

headers = headers.append('header3', headerValue3); // add another header

let params = new HttpParams().set('param1', value1); // create params object

params = params.append('param2', value2); // add a new param, creating a new object

params = params.append('param3', value3); // add another param

return this.\_http.get<any[]>('someUrl', { headers: headers, params: params })

**Is Angular supports dynamic imports?**

Yes, Angular 8 supports dynamic imports in router configuration. i.e, You can use the import statement for lazy loading the module using loadChildren method and it will be understood by the IDEs(VSCode and WebStorm), webpack, etc. Previously, you have been written as below to lazily load the feature module. By mistake, if you have typo in the module name it still accepts the string and throws an error during build time.

{path: ‘user’, loadChildren: ‘./users/user.module#UserModulee’},

This problem is resolved by using dynamic imports and IDEs are able to find it during compile time itself.

{path: ‘user’, loadChildren: () => import(‘./users/user.module’).then(m => m.UserModule)};

**What is Sanitization? Is angular supports it?**

**Sanitization** is the inspection of an untrusted value, turning it into a value that's safe to insert into the DOM. Yes, Angular suppports sanitization. It sanitizes untrusted values for HTML, styles, and URLs but sanitizing resource URLs isn't possible because they contain arbitrary code.

**What is the purpose of innerHTML? You have an HTML response I want to display. How do I do that?**

The innerHtml is a property of HTML-Elements, which allows you to set it's html-content programmatically. Let's display the below html code snippet in a <div> tag as below using innerHTML binding,

<div [innerHTML]="htmlSnippet"></div>

and define the htmlSnippet property from any component

export class myComponent {

htmlSnippet: string = '<b>Hello World</b>, Angular';

}

Unfortunately this property could cause Cross Site Scripting (XSS) security bugs when improperly handled.

**What is the difference between interpolated content and innerHTML?**

The main difference between interpolated and innerHTML code is the behavior of code interpreted. Interpolated content is always escaped i.e, HTML isn't interpreted and the browser displays angle brackets in the element's text content. Where as in innerHTML binding, the content is interpreted i.e, the browser will convert < and > characters as HTMLEntities.

**What is the difference between \*ngIf vs [hidden]?**

\*ngIf effectively removes its content from the DOM while [hidden] modifies the display property and only instructs the browser to not show the content but the DOM still contains it.

**How would you protect a component being activated through the router?**

The Angular router ships with a feature called guards. These provide us with ways to control the flow of our application. We can stop a user from visitng certain routes, stop a user from leaving routes, and more. The overall process for protecting Angular routes:

* Create a guard service: ng g guard auth
* Create canActivate() or canActivateChild() methods
* Use the guard when defining routes

// import the newly created AuthGuard

const routes: Routes = [

{

path: 'account',

canActivate: [AuthGuard]

}

];

Some other available guards:

* CanActivate: Check if a user has access
* CanActivateChild: Check if a user has access to any of the child routes
* CanDeactivate: Can a user leave a page? For example, they haven't finished editing a post
* Resolve: Grab data before the route is instantiated
* CanLoad: Check to see if we can load the routes assets

**What is difference between declarations, providers and import in NgModule?**

  @NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

FormsModule,

HttpClientModule

],

providers: [],

bootstrap: [AppComponent]

})

* Declarations are used to **declare** components, directives, pipes that belongs to the **current module.**
* Providers are used to make services and values known to dependency injection.
* Imports makes the exported declarations of other modules available in the current module.

**What is Multicasting?**

An Observable by default is unicast. Unicasting means that each subscribed observer owns an independent execution of the Observable. ... Multicasting basically means that one Observable execution is shared among multiple subscribers. Subjects are like EventEmitters, they maintain a registry of many listeners.

**Add Elements on Button Click**

add(){

    let row = document.createElement('div');

      row.className = 'row';

      row.innerHTML = `

      <br>

      <input type="text">`;

      document.querySelector('.showInputField').appendChild(row);

  }

**Template Driven Forms Validation**

<form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)" novalidate>

    <p>

      <label for="firstname">First Name </label>

      <input type="text" id="firstname" name="firstname" required minlength="10" #firstname="ngModel"

        [(ngModel)]="contact.firstname">

    </p>

    <div \*ngIf="!firstname?.valid && (firstname?.dirty || firstname?.touched)" class="error">

      <div \*ngIf="firstname.errors.required">

        First Name is required

      </div>

      <div \*ngIf="firstname.errors.minlength">

        First Name Minimum Length is {{firstname.errors.minlength?.requiredLength}}

      </div>

    </div>

    <p>

      <button type="submit" [disabled]="!contactForm.valid">Submit</button>

    </p>

    <p>{{contactForm.valid}} </p>

</form>

**Passing Multiple Files to API**

In HTML-

<input type="file" id="file" multiple

            (change)="getFileDetails($event)">

<button (click)="uploadFiles()">Upload</button>

In Component-

myFiles:string [] = [];

sMsg:string = '';

getFileDetails (e) {

    //console.log (e.target.files);

    for (var i = 0; i < e.target.files.length; i++) {

      this.myFiles.push(e.target.files[i]);

    }

}

uploadFiles () {

    const frmData = new FormData();

    for (var i = 0; i < this.myFiles.length; i++) {

      frmData.append("fileUpload", this.myFiles[i]);

    }

    this.httpService.post('http://localhost:60505/api/fileupload/', frmData).subscribe(

      data => {

        // SHOW A MESSAGE RECEIVED FROM THE WEB API.

        this.sMsg = data as string;

        console.log (this.sMsg);

      },

      (err: HttpErrorResponse) => {

        console.log (err.message);    // Show error, if any.

      }

    );

}

[HttpPost()]

public string UploadFiles()

{

int iUploadedCnt = 0;

// DEFINE THE PATH WHERE WE WANT TO SAVE THE FILES.

string sPath = "";

sPath = System.Web.Hosting.HostingEnvironment.MapPath("~/locker/");

System.Web.HttpFileCollection hfc = System.Web.HttpContext.Current.Request.Files;

// CHECK THE FILE COUNT.

for (int iCnt = 0; iCnt <= hfc.Count - 1; iCnt++)

{

System.Web.HttpPostedFile hpf = hfc[iCnt];

if (hpf.ContentLength > 0)

{

// CHECK IF THE SELECTED FILE(S) ALREADY EXISTS IN FOLDER. (AVOID DUPLICATE)

if (!File.Exists(sPath + Path.GetFileName(hpf.FileName)))

{

// SAVE THE FILES IN THE FOLDER.

hpf.SaveAs(sPath + Path.GetFileName(hpf.FileName));

iUploadedCnt = iUploadedCnt + 1;

}

}

}

// RETURN A MESSAGE.

if (iUploadedCnt > 0) {

return iUploadedCnt + " Files Uploaded Successfully";

}

else {

return "Upload Failed";

}

}

**Generate Components Dynamically**

<button (click)="render()">Render dynamic component</button>

    <ng-template vc></ng-template>

export class AppComponent  {

  constructor(private componentFactoryResolver: ComponentFactoryResolver) { }

@ViewChild('vc', { read: ViewContainerRef }) vc!: ViewContainerRef;

  numberi = 0;

  public render(): void {

    this.numberi++;

    let componentFactory =

      this.componentFactoryResolver.resolveComponentFactory(DynacompComponent);

    let viewContainerRef = this.vc;

    viewContainerRef.clear();

    let componentRef = viewContainerRef.createComponent(componentFactory).instance;

    componentRef.index = this.numberi;

  }

}

{{ index }}

<ul>

    <li \*ngFor="let item of array">

        <input type="text" >

    </li>

</ul>

export class DynacompComponent implements OnInit {

  @Input('index') index: any;

  array : any[] = []; //created this array bcz cannot loop using a number in angular

  constructor() { }

  ngOnInit(): void {

    this.array = Array(this.index).fill(1); //unable to initialize array before ngOninit

  }

}

Angular can infer the reference type by the type of the DOM element. For example, if it’s a simple html element like span, Angular returns ElementRef. If it’s a template element, it returns TemplateRef. Some references, like ViewContainerRef cannot be inferred and have to be asked for specifically in the read parameter. Others, like ViewRef cannot be returned from the DOM and have to be constructed manually.

**ElementRef** – It only holds the native element it’s associated with. It’s useful for accessing native DOM element

<span #tref>I am span</span>

@ViewChild("tref", {read: ElementRef}) tref: ElementRef;

**TemplateRef** –

<ng-template #tpl>

<span>I am span in template</span>

</ng-template>

@ViewChild("tpl") tpl: TemplateRef<any>;

**ViewContainerRef** - Represents a container where one or more views can be attached to a component. The first thing to mention here is that any DOM element can be used as a view container. What’s interesting is that Angular doesn’t insert views inside the element, but appends them after the element bound to ViewContainer. This is similar to how the router-outlet inserts components.

Usually, a good candidate to mark a place where a ViewContainer should be created is the ng-container element. It’s rendered as a comment and so it doesn’t introduce redundant HTML elements into the DOM. Here is the example of creating a ViewContainerat a specific place in a component template:

<ng-container #vc></ng-container>

@ViewChild("vc", {read: ViewContainerRef}) vc: ViewContainerRef;