# Angular:

# How to create component where you will import a created component ? How to create component in angular? What all created?Components and what will be in the components:? What are the files will be created while creating Components.?What is Components.

Ng generate component component\_folder/component\_name

Ng g c component\_folder/component\_name

## **Create and export component**

1. Create a new folder and call it components inside the src folder:
2. Create a new folder and call it hello-world inside the components folder:
3. Create a new file called hello-world.component.html inside the hello-world/ folder:
4. Open hello-world.component.html file and add the code:
5. create a new file called hello-world.component.ts inside the hello-world/ folder.
6. Add the following code to the hello-world.component.ts file:
7. import { Component, OnInit } from '@angular/core';
8. @Component({
9. selector: 'app-assign-audit',
10. templateUrl: './assign-audit.component.html',
11. styleUrls: ['./assign-audit.component.css']
12. })
13. export class AssignAuditComponent implements OnInit {
14. constructor() { }
15. ngOnInit(): void {
16. }
17. }

## 7. **Import a component into a module**

In order to import a component within our App component we have to:

* 1. Import a new component within the app.module.ts file:
     1. Add the following import line:

import { HelloWorld } from './components/hello-world/hello-world.component'

* + 1. Add HelloWorld into the declarations[] array.

## **Use a component**

Now we are ready to use our component. Open app.component.html and paste into it <hello-world></hello-world>. ---- >> Using component in another component

# What are directives and its types? What are the directives you have used in your projects?

The Angular 8 directives are used to manipulate the DOM. By using Angular directives, you can change the appearance, behavior or a layout of a DOM element. It also helps you to extend HTML.

**Angular 8 directives can be classified in 3 categories based on how they behave:**

* Component Directives
* Structural Directives
* Attribute Directives

**Component Directives:** Component directives are used in main class. They contain the detail of how the component should be processed, instantiated and used at runtime.

**Structural Directives:** Structural directives start with a \* sign. These directives are used to manipulate and change the structure of the DOM elements. For example, \*ngIf directive, \*ngSwitch directive, and \*ngFor directive.

* **\*ngIf Directive:** The ngIf allows us to Add/Remove DOM Element.
* **\*ngSwitch Directive:** The \*ngSwitch allows us to Add/Remove DOM Element. It is similar to switch statement of C#.
* **\*ngFor Directive:** The \*ngFor directive is used to repeat a portion of HTML template once per each item from an iterable list (Collection).

**Attribute Directives:** Attribute directives are used to change the look and behavior of the DOM elements. For example: ngClass directive, and ngStyle directive etc.

* **ngClass Directive:** The ngClass directive is used to add or remove CSS classes to an HTML element.
* **ngStyle Directive:** The ngStyle directive facilitates you to modify the style of an HTML element using the expression. You can also use ngStyle directive to dynamically change the style of your HTML element.

# What are pipes? Pipes and Custom pipes.

In Angular 1, filters are used which are later called Pipes onwards Angular2. In Angular 7, it is known as pipe and used to transform data. It is denoted by symbol |

### **Syntax:**

1. {{title | uppercase}}

Pipe takes integers, strings, arrays, and date as input separated with |. It transforms the data in the format as required and displays the same in the browser.

1. **<h1>**
2. {{ title | uppercase }} **<br/></h1>**
3. **<h1>**
4. {{ title | lowercase }} **<br/></h1>**

## **Angular 7 Built-in Pipes**

Angular 7 provides some built-in pipes:

* Lowercasepipe
* Uppercasepipe
* Datepipe
* Currencypipe
* Jsonpipe
* Percentpipe
* Decimalpipe
* Slicepipe
* **<div** style = "width:100%;"**>**
* **<div** style = "width:40%;float:left;border:solid 1px black;"**>**
* **<h1>**Uppercase Pipe**</h1>**
* **<b>**{{title | uppercase}}**</b><br/>**
* **<h1>**Lowercase Pipe**</h1>**
* **<b>**{{title | lowercase}}**</b>**
* **<h1>**Currency Pipe**</h1>**
* **<b>**{{6589.23 | currency:"USD"}}**</b><br/>**
* **<b>**{{6589.23 | currency:"USD":true}}**</b>** //Boolean true is used to get the sign of the currency.
* **<h1>**Date pipe**</h1>**
* **<b>**{{todaydate | date:'d/M/y'}}**</b><br/>**
* **<b>**{{todaydate | date:'shortTime'}}**</b>**
* **<h1>**Decimal Pipe**</h1>**
* **<b>**{{ 454.78787814 | number: '3.4-4' }}**</b>** // 3 is for main integer, 4 -4 are for integers to be displayed.
* **</div>**
* **<div** style = "width:40%;float:left;border:solid 1px black;"**>**
* **<h1>**Json Pipe**</h1>**
* **<b>**{{ jsonval | json }}**</b>**
* **<h1>**Percent Pipe**</h1>**
* **<b>**{{00.54565 | percent}}**</b>**
* **<h1>**Slice Pipe**</h1>**
* **<b>**{{months | slice:2:6}}**</b>**
* // here 2 and 6 refers to the start and the end index
* **</div>**
* **</div>**

## **How to create a custom pipe?**

To create a custom pipe, create a new ts file and use the code according to the work you have to do. You have to import Pipe, PipeTransform from Angular/Core. Let's create a sqrt custom pipe.

1. import {Pipe, PipeTransform} from '@angular/core';
2. @Pipe ({
3. name : 'sqrt'
4. })
5. export class SqrtPipe implements PipeTransform {
6. transform(val : number) : number {
7. return Math.sqrt(val);
8. }
9. }

In MODULE.ts file add the SqrtPipe.ts in the declarations of the ngModule and import the file

1. import { SqrtPipe } from './app.sqrt';
2. @NgModule({
3. declarations: [
4. SqrtPipe,
5. AppComponent,
6. NewCmpComponent,
7. ChangeTextDirective
8. ],

# How to component-to-component communication achieve? Explain parent and child relation in angular

<https://www.digitalocean.com/community/tutorials/angular-component-communication>

<https://www.digitalocean.com/community/tutorials/rxjs-subjects>

# How can we bootstrap module?

What is bootstrap module in Angular?

bootstrap is **a function component in the core ng module that is used for starting up the Angular application manually**, which gives you more control over how you initialize your application

Angular is flexible enough to run in a browser, server, web-worker, or mobile device. **When bootstrapModule(AppModule, options) is called, it compiles the AppModule in the first step**. You will notice that it is loading all the modules, directives, and pipes metadata. Then, it compiles all the components

The process of loading the index.html page, app-level module, and app-level component is called *bootstrapping*, or loading the app. In this guide, you will learn about the internals of the bootstrapping process.

Angular takes the following steps to bootstrap the application:

1. Load index.html
2. Load Angular, Other Libraries, and App Code
3. Execute main.ts File
4. Load App-Level Module
5. Load App-Level Component
6. Process Template

# Difference b/w Angular JS and Angular.

| **Category** | **Angular JS** | **Angular** |
| --- | --- | --- |
| **Architecture** | It supports the [Model-View-Controller design](https://www.geeksforgeeks.org/model-view-controllermvc-architecture-for-node-applications/). The view processes the information available in the model to generate the output. | It uses components and directives. Components are the directives with a template. |
| **Written Language** | Written in JavaScript. | Written in Microsoft’s TypeScript language, which is a superset of [ECMAScript 6 (ES6)](https://www.geeksforgeeks.org/introduction-to-es6/). |
| **Mobile support** | It does not support mobile browsers. | Angular is supported by all the popular mobile browsers. |
| **Expression Syntax** | [**ng-bind**](https://www.geeksforgeeks.org/angularjs-ng-bind-directive/) is used to bind data from view to model and vice versa. | Properties enclosed in “()” and “[]” are used to bind data between view and model. |
| **Dependency Injection** | It does not use Dependency Injection. | It uses Dependency Injection. . |
| **Supported Languages** | It only supports JavaScript. | It provides support for TypeScript and JavaScript. |
| **Routing** | AngularJS uses $routeprovider.when() for routing configuration. | Angular uses @Route Config{(…)} for routing configuration. |
| **Structure** | It is less manageable in comparison to Angular. | It has a better structure compared to AngularJS, easier to create and maintain for large applications but behind AngularJS in the case of small applications. |
| **CLI** | It does not come with a CLI tool. | It comes with the Angular CLI tool. |
| **Examples Application** | iStock, Netflix, and Angular JS official website. | Upwork, Gmail, and Wikiwand. |

# Difference b/w JIT and AOT.

An angular application mainly consists of **HTML templates,** their components which include various TypeScript files. There are some unit testing and configuration file. Whenever we run over an application, the browser cannot understand the code directly hence we have to compile our code.

**What is Ahead of Time (AOT) compiler ?**

All technologies Ahead of Time is a process **of compiling higher-level language or intermediate language** into a **native machine code,** which is system dependent.

In simple words, when you serve/build your angular application, the **Ahead** **of** **Time** **compiler converts your code during the build time** **before your browser downloads and runs that code. From Angular 9, by default compiling option is set to true for** ahead of time compiler.

**Why should you use the Ahead of Time compiler ?**

* When you are using Ahead of Time Compiler**, compilation only happens once, while you build your project**.
* **We don’t have to** ship the **HTML templates** and **the Angular compiler** whenever we enter a new component.
* It can **minimize the size of your application**.
* **The browser does not need to compile** the code in run time, it can **directly render** the application immediately, without waiting to compile the app first so, it provides **quicker component rendering.**
* The Ahead of time **compiler detects template error earlier**. It detects and reports template binding errors during the build steps before users can see them.
* AOT **provides better security**. It compiles HTML components and templates into JavaScript files long before they are served into the client display. So, there are no templates to read and **no risky client-side HTML or JavaScript evaluation**. This will **reduce the chances of injections attacks.**

**How Ahead of Time works ?**

We use Typescript, HTML, style-sheets to develop our Angular project **and ng build –prod or ng build to build** our source code into bundles which **include JS files, index.html, style-sheets, and assets files.**

Now Angular uses the **angular compiler (whichever** you have selected) to build source code, and they do it in three phases, which are code analysis, code generation and template type checking. At the end of this process, **bundle size will be much smaller than the JIT compiler’s bundle size.**

After that AOT builds this into a war file to deploy directly by using Heroku or by JBoss or by any other hosting that supports Node. And then we map this host to the domain by using a CNAME.

Now the clients can access your web application via the domain. The browser will download all necessary files like HTML, style-sheets, and JavaScript which is needed for the default view.  At last, your application will get bootstrap and get rendered.

**How to compile your app in ahead of time compiler:**For compiling your app in Ahead of time, you don’t have to do much, because from angular 9 default compiling option is set to Ahead of time. Just add –AoT at the end **ng serve –aot.**

**What is the Just in Time (JIT) compiler ?**

[Just in time compiler](https://www.geeksforgeeks.org/just-in-time-compiler/) provides compilation during the **execution of the program at a run time before execution**. In simple words, code get compiles **when it’s needed, not at the build time.**

**Why and When Should you use Just In Time Compiler ?**

* Just in time **compiler compiles each file separately** and it’s mostly **compiled in the browser**. You don’t have to build your project again after changing your code.
* Most **compiling** is done on the **browser side,** so it **will take less compiling time**.
* If you have a big project or a situation where some of your components don’t come in use most of the time then you should use the Just in time compiler.
* Just in Time compiler is best **when your application is in local development**.

**How Just in Time compiler Works?**

Initially, compiler was responsible for converting a high-level language into machine language, which would then be converted into executable code.

Just in time compiler, compiles code at runtime which means instead of interpreting byte code at build time, **it will compile byte code** **when that component is called**.

**A few important points:**

* In case of Just in time, not all code is compiled at the initial time. **Only necessary componen**t which are going to be needed at the starting of your application will be **compiled**. Then if the functionality is need in your project and it’s not in compiled code, that function or component will be compiled.
* This process will help to **reduce the burden on the CPU and** make **your app render fast.**
* One more interesting thing is, you can see and link to **your source code in inspect mode** because Just in Time, compiles your code with JIT mode and a map file.

**Comparison between Ahead of Time (AOT) and Just in Time (JIT) –**

| **JIT** | **AOT** |
| --- | --- |
| JIT downloads the **compiler** and compiles code **exactly** **before** **Displaying** in the browser. | AOT has already complied with the code while building your application, so it doesn’t have **to compile at runtime.** |
| Loading in JIT is slower than the AOT because it needs to compile your application at runtime. | Loading in AOT is much quicker than the JIT because it already has compiled your code at build time. |
| JIT is more suitable for **development mode**. | AOT is much suitable in the case **of Production mode**. |
| **Bundle** **size** is **higher** compare to AOT. | **Bundle size optimized in AOT**, in results AOT bundle size is half the size of JIT bundles. |
| You can run your app in JIT with this command:  **ng build OR ng serve** | To run your app in AOT you have to provide –aot at the end like:  **ng build --aot OR ng serve --aot** |
| You can catch **template binding error at display time**. | You can catch the **template error at building your application**. |

**Conclusion:**You can compile your angular application in two ways: JIT and AOT. Both are suitable for a different scenario like you can use **JIT for development mode** and **AOT is better in production mode.  Implementing features and debugging is easy in JIT mode** since you have to map files while AOT does not have it. However, that AOT provides a big benefit to angular developers for production **mode by reducing bundle size** and **making your app render faster.**

# What is Single Page Application?

A single-page application is a web application or website that **interacts** **with the user by dynamically rewriting the current web page with** **new data from the web server, instead of the default method of a web browser loading entire new pages**

**Why is it called single-page application?**

Single Page Applications are web applications that load a single HTML page and only a part of the page instead of the entire page gets updated with every click of the mouse. The page does not reload or transfer control to another page during the process

A Multi-page Application is a web application consisting of a large number of pages completely refreshed every time when data changes on them. Any data change or data transfer to the server leads to a new page displayed in the browser

# What is Angular? Why we use that?\*\*

Angular 2+ has been a very popular front-end platform for modern web applications since 2016. It introduced a component-based architecture with great benefits like modularity, which allows developers to split the web application into different modules and load them with one of the three module-loading strategies, including Eager Loading, Lazy Loading, and Pre-Loading.

# Lazy Loading and Eager Loading

<https://medium.com/@lifei.8886196/eager-loading-lazy-loading-and-pre-loading-in-angular-2-what-when-and-how-798bd107090c>

What is {Eager Loading, Lazy Loading, Pre-Loading} in Angular?

1. What is Eager Loading?

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

2. What is Lazy Loading?

Feature modules under Lazy Loading would **be loaded on demand after the application starts**. It helps to start application faster.

3. What is Pre-Loading?

Feature Modules under **Pre-Loading would be loaded automatically after the application starts.**

When to use {Eager Loading, Lazy Loading, Pre-Loading} in Angular?

Understanding when to use the right module-loading strategy in the web application is far more important than just having a purely intellectual concept of these strategies. I would like to share my personal opinion as follows:

When to use Eager Loading?

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.

When to use Lazy Loading?

The scenario of applying Lazy Loading is relatively simple and straightforward. In a big-size web application, we can lazily load all other modules that are not required when the application starts.

When to use Pre-Loading?

Compared with Eager Loading and Lazy Loading, Pre-Loading is not so much frequently used in web application development. Based on my understanding of this loading strategy, Pre-Loading would be favorable for two cases though.

Case 1: Medium size application. In this scenario, we can make the application start faster since it will load all other modules later that are not required to run the application. And the application would be more responsive to process users’ requests than applying Lazy Loading strategy since the application will load all these modules after the application started.

Case 2: Some specific modules that users are very likely to use after the application started. In this scenario, we can pre-load these feature modules and still lazy load other modules.

for an enterprise-level web application, the ideal solution would be a combination of these three strategies:

Eager Loading: used to load core modules and feature modules that are required to start the application.

Pre-Loading: used to load specific feature modules that are very likely to be used soon after the application started.

Lazy Loading: all other modules could be lazily loaded on demand after the application started.

Lazy loading example

|  |
| --- |
|  |
| const routes: Routes = [ |
|  | {path: '', redirectTo: 'eager-loading', pathMatch: 'full'}, |
|  | {path: 'eager-loading', component: EagerHomeComponent, children: [ |
|  | {path: '', redirectTo: 'child1', pathMatch: 'full'}, |
|  | {path: 'child1', component: EagerChild1Component}, |
|  | {path: 'child2', component: EagerChild2Component}, |
|  | {path: '\*\*', redirectTo: 'child1'} |
|  | ]}, |
|  | { |
|  | path: 'lazy-loading', |
|  | **loadChildren**: './features/lazy-loading-module/lazy-loading.module#LazyLoadingModule' |
|  | }, |
|  | {path: '\*\*', redirectTo: 'eager-loading'} |
|  | ]; |
|  |  |

Pre loading example

|  |
| --- |
| { |
|  | path: 'pre-loading', |
|  | loadChildren: './features/pre-loading-module/pre-loading.module#PreLoadingModule', |
|  | data: { applyPreload: true } |
|  | }, As you can see from above sample code in **line 29** and **line 36**: I added an extra property named “data” with “applyPreload” set to “true”, and I also specified that I will use my own CustomPreloadingStrategy. So now we need to complete two things:  Implement the **CustomPreloadingStrategy** class using **PreloadingStrategy** interface.  Provide the CustomPreloadingStrategy class in app.module.ts  Following is my sample code to implement the CustomPreloadingStrategy class:   |  |  | | --- | --- | |  | import { PreloadingStrategy, Route } from '@angular/router'; | |  | import { Observable, of } from 'rxjs'; | |  |  | |  | export class CustomPreloadingStrategy implements PreloadingStrategy{ | |  |  | |  | // loadModule could be any function name here | |  | preload(route: Route, loadModule: Function): Observable<any> { | |  | return route.data && route.data.applyPreload ? loadModule() : of(null); | |  | } | |  | } | |

# Data binding and its types. One way and two way binding?\*\*

<https://www.javatpoint.com/data-binding-in-angular-8#:~:text=Data%20binding%20is%20the%20core,your%20template%20which%20user%20sees>.

Data binding is the core concept of Angular 8 and **used to define the communication between a component and the DOM**. It is a technique **to link your data to your view layer**. In simple words, you can say that data binding is a communication between **your typescript code** of your component **and your template which user sees.**  It makes easy to define interactive applications without worrying about pushing and pulling data.

Data binding can be either one-way data binding or two-way data binding.

**One-way databinding**

One way databinding is a simple one way communication where HTML template is changed when we make changes in TypeScript code.

In one-way databinding, the value of the Model is used in the View (HTML page) but you can't update Model from the View. Angular Interpolation / String Interpolation, Property Binding, and Event Binding are the example of one-way databinding.

**Two-way databinding**

In two-way databinding, automatic synchronization of data happens between the Model and the View. Here, change is reflected in both components. Whenever you make changes in the Model, it will be reflected in the View and when you make changes in View, it will be reflected in Model.

This happens immediately and automatically, ensures that the HTML template and the TypeScript code are updated at all times.

Angular provides four types of data binding and they are different on the way of data flowing.

* String Interpolation
* Property Binding
* Event Binding
* Two-way binding

**String interpolation**

String Interpolation is a one-way databinding technique which is used to output the data from a TypeScript code to HTML template (view). It uses the template expression in double curly braces to display the data from the component to the view.

For example:

{{ data }}

String interpolation adds the value of a property from the component:

<li>Name: {{ user.name }}</li>

<li>Email: {{ user.email }}</li>

**Property Binding**

Property Binding is also a **one-way data binding technique**. In property binding, we **bind a property of a DOM element** to a field which is a defined property in **our component TypeScript code**.

For example:

<img [src]="imgUrl"/>

**<input** type="email" [value]="user.email"**>**

**Event Binding**

In Angular 8, event binding is used to handle the events raised from the DOM like button click, mouse move etc. When the DOM event happens (eg. click, change, keyup), it calls the specified method in the component. In the following example, the cookBacon() method from the component is called when the button is clicked:

<button **(click)="cookBacon()"></**button>

**Two-way Data Binding**

We have seen that in one-way data binding any change in the template (view) were not be reflected in the component TypeScript code. To resolve this problem, Angular provides two-way data binding. The two-way binding has a feature to update data from component to view and vice-versa.

In two way data binding, property binding and event binding are combined together.

**[(ngModel)] = "[property of your component]"**

# Types of forms? What is Forms?\*

In Angular 8, there are 2 approaches to **handle user's input** through forms:

* Reactive forms
* Template-driven forms

Both approaches are used **to collect user input events from the view**, **validate the user input**, **create a form model** and **data model** to upd Both Reactive forms and Template-driven forms manage and process data **differently**. Each offers different advantages.ate, and provide a way to track changes.

**Reactive Forms**

* Reactive forms are more **robust**.
* Reactive forms are more **scalable**, **reusable**, and testable.
* They are most preferred to use if **forms** are **a key part of your application**, or your application is already built using reactive patterns. In both cases, reactive forms are best to use.

**Template-driven Forms**

* Template-driven forms are best if you want to **add a simple form to your application**. **For example:** **email list signup form**.
* Template-driven forms are **easy to use in the application** but they **are not as scalable as Reactive forms**.
* Template-driven forms are mainly used if your application's **requires a very basic form and logic**. It can **easily** be **managed** in a template.

Register the Reactive Forms Module

Use the reactive forms by importing ReactiveFormsModule from the @angular/forms package and add it to your app.module.ts file's imports array.

1. **// app.module.ts**

**import { ReactiveFormsModule } from '@angular/forms';**

**@NgModule({**

**declarations: [**

**AppComponent**

**],**

**imports: [**

**BrowserModule,**

**AppRoutingModule,**

**ReactiveFormsModule**

**],**

* Add **FormControl** class register the control into the template and update the **FormControl** value

The **FormControl** class is the **fundamental building block** when using the reactive forms. So if you want to register the single form control, you need to **import the FormControl** class into your component and **create the new instance of a form control to save as the class property.**

// app.component.ts

import { FormControl } from '@angular/forms';

export class AppComponent {

email = new FormControl('');

password = new FormControl('');

updateEmail() {

this.email.setValue('sonoojaiswal@javatpoint.com');

}

}

* Also, update the view app.component.html file.

<div class="container">

<div class="form-group">

<label>

Email:

</label>

<input type="text" [formControl]="email" />

</div>

<div class="form-group">

<label>

Password:

</label>

<input type="text" [formControl]="password" />

</div>

<div class="form-group">

<button (click)="updateEmail()" class="btn btn-dark">Update Email</button>

</div>

<p>

Value: {{ email.value }}

Value: {{ password.value }}

</p>

</div>

Now, save your code and start the server.

# Create a reactive Form?\*\* they asked me to write a reactive form and whole flow **with validation and all** \*\*Design Registration Form in notepad (Html, Ts file, service file, modal file)

# Fonts in angular?

You need to put the font files in **assets** folder (may be a fonts sub-folder within assets) and refer to it in the styles:

@font-face {

font-family: lato;

src: url(assets/font/Lato.otf) format("opentype");

}

Once done, you can apply this font any where like:

\* {

box-sizing: border-box;

margin: 0;

padding: 0;

font-family: 'lato', 'arial', sans-serif;

}

You can put the @font-face definition in your global styles.css or styles.scss and you would be able to refer to the font anywhere - even in your component specific CSS/SCSS. styles.css or styles.scss is already defined in angular-cli.json. Or, if you want you can create a separate CSS/SCSS file and declare it in angular-cli.json along with the styles.css or styles.scss like:

"styles": [

"styles.css",

"fonts.css"

],

<https://stackoverflow.com/questions/49878988/how-to-import-a-new-font-into-a-project-angular-5>

# No of screens developed?

5 – login page , user registration , car dashboard , profile information , Contract information

# Life cycle Hooks. Life cycle hooks in usage?

# What kind of routing in UI? How do you implement router?

# How services are called from angular? How to create service.

Dependency injection (DI) is the part of the Angular framework that provides components **with access to services and other resources**. Angular provides the ability for you to inject a service into a component to give that component access to the service.

**The @Injectable() decorator defines a class as a service in Angular** and allows Angular to inject it into a component as a dependency. Likewise, the @Injectable() decorator indicates that a component, class, pipe, or NgModule has a dependency on a service.

@Injectable({

  providedIn: 'root'

})

export class AuditorsService {

  constructor(private http:HttpClient) { }

constructor(private service: HeroService) { } in component

# What is observables \* and promises?

<https://www.infragistics.com/community/blogs/b/infragistics/posts/angular-observable-vs-angular-promise>

Observable in Angular is a feature that **provides** **support for delivering messages** between **different parts of your single-page application.** This feature is frequently used in Angular because it is **responsible for handling multiple values**, **asynchronous programming in Javascript**, and also **event handling processes.**

However, an observer or observable is a software paradigm constituting the design of an object which is called a subject that maintains a list of various dependencies that are called observers. These dependencies are automatically notified whenever you try to change states.

 The subscriber is termed as a **consumer** who **receives the notifications until the function completes itself** or until they **manually unsubscribe**. Since the setup and breakdown are handled by the observable, you don't need to worry about your application code being subscribed to consumer values or unsubscribed. Be it an HTTP response or a timer of intervals, the interface for listening to events and stopping them will be the same. This adds an advantage of making use of observables frequently for development purposes

To make use of the observable, all you need to do is to begin by creating notifications using subscribe() method, and this is done by passing observer as discussed previously. The notifications are generally Javascript objects that handle all the received notifications. Also, the unsubscribe() method comes along with subscribing () method so that you can stop receiving notifications at any point in time.

Consider the below example constituting geolocation updates whenever a subscriber subscribes.

const locationslocationsSubscription = locations.**subscribe**({

**next**(position) {

console.log('Current Position: ', position);

},

error(msg) {

console.log('Error Getting Location: ', msg);

}

});

**Types of Notifications and Description**

1. **Next**: It is called **after the execution starts** for zero times or more than that. It is a mandatory notification for catching each value delivered.
2. **Error**: This is **an optional handler** for chasing error notifications. This notification halts execution while running instances of observable.
3. **Complete**: It is an optional handler **that notifies completion of executions**. The currently delayed values continue to be delivered even when the execution is marked as complete

Promises in Angular **provide an easy way to execute asynchronous functions that use callbacks, while emitting and completing (resolving or rejecting) one value at a time**.

# What is @viewchild?

Property decorator that configures a view query. The change detector looks for the first element or the directive matching the selector in the view DOM. If the view DOM changes, and a new child matches the selector, the property is updated.

The @ViewChild decorator allows us to **inject into a component class references** to e**lement**s used inside its **template**, that's what we should use it for.

Using @ViewChild we can **easily inject components**, **directives or plain DOM elements**. We can even override the defaults of @ViewChild and specify exactly we **need to inject, in case that** multiple options are available.

@ViewChild is a **local component template** querying mechanism, that cannot see the **internals of its child components.**

By injecting references directly into our component class, we can easily **write any coordination logic** that involves multiple elements of the template

<https://angular.io/api/core/ViewChild>

# Implementing Onint is need or not?

Technically we don’t need to implement OnInit interface on the top as long as we have a method called ngOnInit() defined in our class. Angular will automatically call this when it initializes our component but we use implement OnInt interface to add compile time checking, so when we define implement OnInit interface typescript ensures that we have a method called ngOnInit.

So the lesson is, o not call the http services inside constructors. If you want initialization then do that in ngOnInit method.

A lifecycle hook that is called after Angular has initialized all data-bound properties of a directive. Define an ngOnInit() method to handle any additional initialization tasks.

interface [OnInit](https://angular.io/api/core/OnInit) { [**ngOnInit**(): void](https://angular.io/api/core/OnInit#ngOnInit) }

**Methods**

**ngOnInit()**

**A callback method that is invoked immediately after the default change detector has checked the directive's data-bound properties for the first time, and before any of the view or content children have been checked. It is invoked only once when the directive is instantiated.**

**ngOnInit(): void**

###### **Parameters**

There are no parameters.

###### **Returns**

Void

Usage notes

The following snippet shows how a component can implement this interface to define its own initialization method.

@[Component](https://angular.io/api/core/Component)({selector: 'my-cmp', template: `...`}) class MyComponent implements [OnInit](https://angular.io/api/core/OnInit) { ngOnInit() { // ... } }

<https://stackoverflow.com/questions/52511811/what-is-the-purpose-of-implementing-oninit-class-removing-ngoninit-works-fine>

# Authorization

<https://www.tutorialspoint.com/angular8/angular8_authentication_and_authorization.htm>

# Karma and jasmine.

# Base Framework for angular?

# Flow of execution in Angular

# How to connect Angular to SpringBoot? How to connect front and backend in angular.