ANGULAR

The Angular framework is written using the TypeScript language. It implements the core and optional functionality as a set of TypeScript libraries that we need to import into our applications in order to develop angular applications.
Angular is an open-source front-end web framework. It is one of the most popular JavaScript frameworks that is mainly maintained by Google. It provides a platform for easy development of Mobile and Desktop web applications using HTML and TypeScript.
It integrates powerful features like declarative templates, end to end tooling, dependency injection.
Angular is typically used for the development of SPA which stands for Single Page Applications.

What are Single Page Applications (SPA)?

A single-page application is defined as an application (web app or website) that loads only a single page and then rewrites the page with new content fetched from a web server as the user interacts with it instead of loading a new page for every interaction. A Single Page Application method is speedier, resulting in a more consistent user experience. Angular provides a set of ready-to-use modules that simplify the development of single page applications.

Advantages of using Angular framework are listed below

- It supports two-way data-binding
- It follows MVC pattern architecture
- It supports static template and Angular template
- You can add a custom directive
- It also supports RESTful services
- Validations are supported
- Client and server communication is facilitated
- Support for dependency injection
- Has strong features like Event Handlers, Animation, etc.

TypeScript is open-source and it is developed and maintained by Microsoft. The TypeScript language provides several benefits are as follows while developing angular applications
1- Intelligence support while writing code 2- Auto-completion Facility
3- Code navigation 4- Strong Typing
5- It also supports features like classes, interfaces, and inheritance as our traditional programming languages such as C#, Java, or C#.

For setting up Angular Application (node -v and npm –v)

Node.js is an open-source cross-platform javascript run-time environment)

NPM is node.js package manager for javascript programming language. It is automatically installed when we install node.js

Typescript - npm install –g typescript

Angular CLI (It is a tool that allows us to create a project, build and run the project in the development server directly using the command line command) - ng v

npm install -g @angular/cli
Create ang App- ng new project-name
Cmd-
Ng new appName --create-application=false ->create workspace but not application
Cd appName
Ng g application webAppName
Ng g library myLib
Ng g c home --skip-tests=true
Ng g c login
Ng g c login/logout
Ng g c signup -> create comp. At root level
ng update @angular/cli @angular/core

_____

node_modules Folder-This folder contains the packages (folders of the packages) which are installed into your application when you created a project in angular. If you installed any third-party packages then also their folders are going to be stored within this node_modules folder. npm install

Only private dependencies are required at production level.

Src folder- This is the source folder of our angular application. That means this is the place where we need to put all our application source code. So, every component, service class, modules, everything we need to put in this folder only.

Assets folder where you can store the static assets like images, icons, etc.

Environments folder is basically used to set up different environments. These two files are as follows:

1. environment.prod.ts. This file for the production environment
2. environment.ts. This file for the development environment.

Favicon.icon*: It is the icon file that displays on the browser.*

Index.html *file contains HTML code with the head and body section. It is the starting point of your application or you can say that this is where our angular app bootstraps.*

Polyfills.ts File is basically used for browser-related configuration. In angular, you write the code using typescript language. The Polyfills.ts file is used by the compiler to compile your Typescript code to a specific JavaScript method so that it can be parsed by different browsers.

Angular.json- It contains all the configuration of Angular Project. It contains the configurations such as what is the project name, what is the root directory as source folder (src) name which

contains all the components, services, directives, pipes, what is the starting point of your angular application (index.html file), What is the starting point of typescript file (main.ts), etc.

How does an Angular application work?

Every Angular app consists of a file named angular.json. This file will contain all the configurations of the app. While building the app, the builder looks at this file to find the entry point of the application. Inside the build section, the main property of the options object defines the entry point of the application which in this case is main.ts.
main.ts file creates a browser environment for the application to run, and, along with this, it also calls a function called bootstrapModule, which bootstraps the application. These two steps are performed in the following order inside the main.ts file:

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

In the above line of code, AppModule is getting bootstrapped. The AppModule is declared in the app.module.ts file. This module contains declarations of all the components. AppComponent is getting bootstrapped. This component is defined in app.component.ts file. This file interacts with the webpage and serves data to it. Each component is declared with three properties:

- Selector - used for accessing the component
- Template/TemplateURL - contains HTML of the component
- StylesURL - contains component-specific stylesheets

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

Browserslist file is currently used by autoprefixer to adjust CSS to support the specified browsers.

test.ts file is the configuration file of Karma which is used for setting the test environment. Within this file, the tester will write the unit test cases for testing the project.

karma.config.js file is used to store the setting of karma i.e. test cases. It has a configuration for writing unit tests. Karma is the test runner and it uses jasmine as a framework. Both tester and framework are developed by the angular team for writing unit tests.

package-lock.json is automatically generated for those operations where npm modifies either the node_modules tree or package.json. In other words, the package.lock.json is generated after an npm install.

Package.json: file is mandatory for every npm project. It contains basic information regarding the project (name, description, license, etc), commands which can be used, dependencies – these are packages required by the application to work correctly, and dependencies – again the list of packages which are required for application however only during the development phase.

README is the file that contains a description of the project which we would like to give to the end-users so that they can start using our application in a great manner. This file contains the basic documentation for your project, also contains some pre-filled CLI command information.

Tsconfig.app.json: This file is used during the compilation of your application and it contains the configuration information about how your application should be compiled.

Tsconfig.json: *This file contains the configurations for typescripts. If there is a tsconfig file in a directory, that means that the directory is a root directory of a typescript project, moreover, it is also used to define different typescript compilation-related options.*

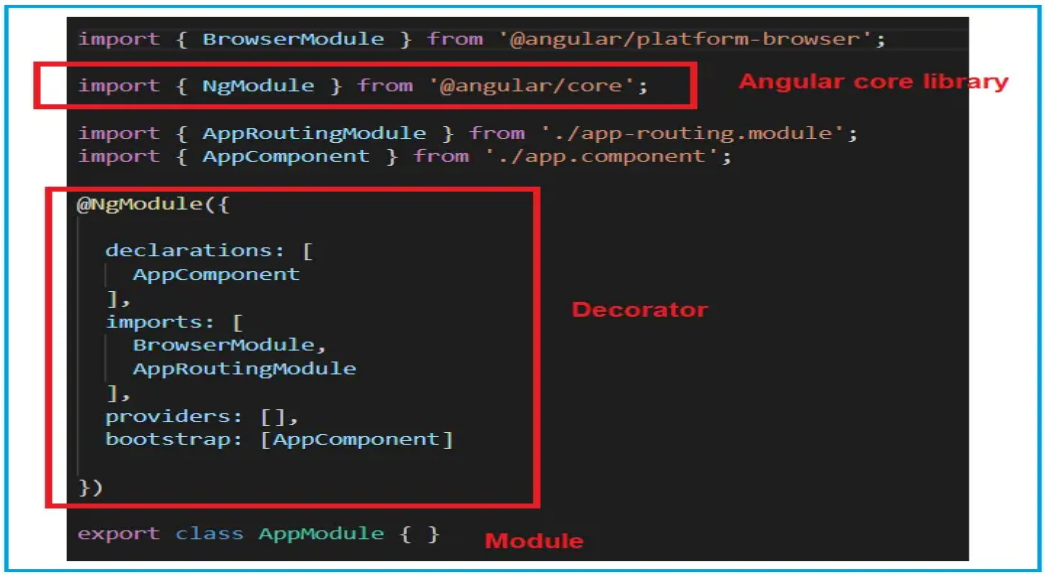
Tsconfig.spec.json*: This file is used for testing purposes in the node.js environment. It also helps in maintaining the details for testing.*

Tslint.json*: This is a tool useful for static analysis that checks our TypeScript code for readability, maintainability, and functionality errors.*

Module- is a mechanism to group components, services, directives, pipes. So, the grouping of related components, services, pipes, and directives is nothing but a module in angular. AppModule (app.module.ts file) is the default module or root module of our angular application.

By default, modules are of two types: 1- root module imports BrowserModule, whereas a 2- feature module imports CommonModule.

```
ng generate module modulename
ng g module modulename
```



Declarations array is an array of components, directives, and pipes. Whenever you add a new component, first the component needs to be imported and then a reference of that component must be included in the declarations array.

Imports section, we need to include other modules (Except @NgModule). By default, it includes two modules i.e. BrowserModule (Exports required infrastructure for all Angular apps) and AppRoutingModule.

Providers Array (providers): We need to include the services in the provider's section. Whenever you create any service for your application, first you need to include a reference of that service in the provider's section and then only you can use that service in your application.

Bootstrap Array (bootstrap): This section is basically used to bootstrap your angular application i.e. which component will execute first. At the moment we have only one component i.e. AppComponent and this is the component that is going to be executed when our application runs.

Decorators are the features of Typescript and are implemented as functions. The name of the decorator starts with @ symbol followed by brackets and arguments. The decorator provides metadata to [CPVM] classes, property, value, method, etc. and decorators are going to be invoked at runtime.
built-in decorators are available in angular. All built-in decorators are imported from @angular/core library. Some of them are as follows:

1. @NgModule to define a module.
2. @Component to define components.
3. @Injectable to define services.
4. @Input and @Output to define properties

Types of decorators: CPPM

1- Class Decorator: Class Decorators are the top-level decorator that define the purpose of the classes. They indicate that the class is a component or module. And the decorator enables us to declare this effect without having to write any code within the class.

@Component and @NgModule

2- Property Decorator: used to decorate specific properties inside classes.
such as @Input and @Output.

@Input and @Output (These two decorators are used inside a class)

3- Parameter Decorator: for parameters inside class constructors, such as @Inject

4- Method Decorator: are used to decorate the method defined inside class with functionality.

@HostListener (This decorator is used for methods inside a class like a click, mouse hover, etc.)

Components- are basic building blocks which control the UI part of an application. Component is defined using @Component decorator. Component is where you write your binding code. Component Binds the UI and Model.

- component file, <component-name>.component.ts
- A template file, <component-name>.component.html
- A CSS file, <component-name>.component.css

- A testing specification file, <component-name>.component.spec.ts

The most important advantage of this component-based development approach is that it provides support for code reusability. That means you can reuse the same component in multiple places. This Component-based development approach also provides great support for unit testing the Angular application.

Component consists of 3 parts-
Template-(loads view for component.) Templates are written with HTML that contains Angular-specific elements & attributes.These templates are combined with information coming from the model and controller which are further rendered to provide the dynamic view to the user.
There are two ways to create a template in an Angular component:

- Inline Template- The component decorator's template config is used to specify an inline HTML template for a component. The Template will be wrapped inside the single or double quotes.
- Linked Template- A component may include an HTML template in a separate HTML file. As illustrated below, the templateUrl option is used to indicate the path of the HTML template file.

Stylesheet- which defines look & feel for components.

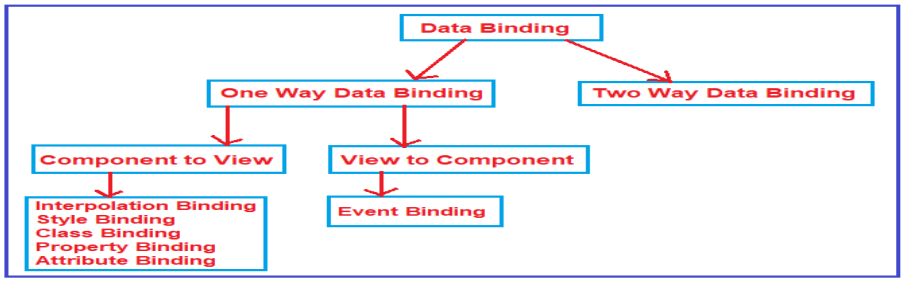
Class- contains business logic for components. Class is the most important part of a component in which we can write the code which is required for a template to render in the browser. Classes can also contain methods, variables and properties. Class properties and variables contain the data which will be used by a template to render on the view. Similarly, the method in class is used to implement the business logic like the method does in other programming languages.

Data Binding means to bind the data (Components filed) with the View (HTML Content). Data Binding is a process that creates a connection to communicate and synchronize between the user interface and the data.
Data-binding is all about binding to the DOM object properties and not the HTML element attributes. Binding works with the properties and events, and not with the attributes.

There are two types of Data binding
1- One-way Data Binding- where a change in the model affects the view (i.e. From Component to View Template) or change in the view affects the state (From View Template to Component).
2- Two-way Data Binding- where a change from the view can also change the model and similarly change in the model can also change in the view (From Component to View Template and also From View template to Component).

Difference b/w 1 way, 2way

In **One-Way data binding**, the View or the UI part does not update automatically whenever the data model changes. You need to manually write custom code in order to update it every time the view changes.
Whereas, in **Two-way data binding,** the View or the UI part is updated implicitly as soon as the data model changes. It is a synchronization process, unlike One-way data binding.



Component to View

1- **Attribute Binding** is used to bind the attribute of an element with the properties of a component dynamically. (such as colspan, area, etc). To tell the angular framework that we are setting an attribute value, we have to prefix the attribute name with the attr and a DOT as shown below.

```
<thead>
    <tr>
        <th [attr.colspan]="ColumnSpan">
            {{pageHeader}}
        </th>
    </tr>
</thead>
```

2- Property Binding is used to bind the values of model properties to the HTML element. Depending on the values, it will change the existing behavior of the HTML element. The syntax to use property is: [property] = 'expression'
Property binding example: `<button [disabled]='IsDisabled'>Click Me</button>`

```
<span [innerHTML] = 'Title' ></span>
<img [src] = 'ImagePath' />
```

Note: Interpolation and Property binding both flow the values from a component to the view template i.e. in one direction.

3- **Class Binding** is basically used to add or remove classes to and from the HTML elements. It is also possible in Angular to add CSS Classes conditionally to an element, which will create the dynamically styled elements and this is possible because of Angular Class Binding.

4- **Style Binding** is basically used to set the style in HTML elements. You can use both inline as well as Style Binding to set the style in the element in Angular Applications.
5- String interpolation is a special syntax that uses template expressions within double curly {{ }} braces for displaying the component data.
Interpolation allows you to place the component property name in the view template, enclosed in double curly braces i.e. {{propertyName}}. So, Interpolation is a technique that allows the user to bind a value to a UI element.

Interpolation example: <button disabled='{{IsDisabled}}'>Click Me</button>

DOM stands for Document Object Model. When a browser loads a web page, then the browser creates the Document Object Model (DOM) for that page.

View to Component-
Event binding It allows your component to react to user's actions such as button click,
Listens for an element change event. In event binding, parentheses () are used.
lets you listen for and respond to user actions such as keystrokes, mouse movements, clicks, and touches.
Syntax: (target event name) = "template statement"

Two-way binding gives components in your application a way to share data. Use two-way binding to listen for events and update values simultaneously between parent and child components.
Two-way binding combines property binding with event binding:
Event binding - Listens for an element change event
Property binding- Sets a specific element property.
Two-way binding syntax is a combination of square brackets and parentheses, [()]. The [()] syntax combines the brackets of property binding, [], with the parentheses of event binding, () way to achieve two-way data binding
1-**Using the input event of the input control**: export class AppComponent{Name:string = 'Anurag'; }
 Name : <input [value]='Name' (input) = 'Name = $event.target.value'>
 You entered : {{Name}}
$event: It is provided by angular event binding and it contains the event data. To retrieve the values from the input element, you need to use it as – $event.target.value.
2-**using ngModel Directive**
 Name : <input [(ngModel)]='Name'>
 You entered : {{Name}}
ngModel directive is available in FormsModule. Import the FormsModule in your root module that is AppModule.

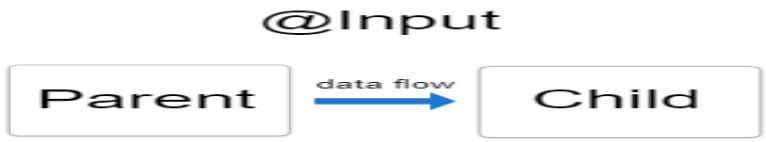
Angular Component Input Properties- Component Input Properties are used to pass the data from container component to the nested component. (parent to child)

How does one share data between components in Angular?

- Parent to Child: via @Input decorator
- Child to Parent: via Output() and EventEmitter
- Child to Parent: via ViewChild
- Unrelated Components: via a Service

Note- = Assign a value || == Compare two values || === Compare two values and their types

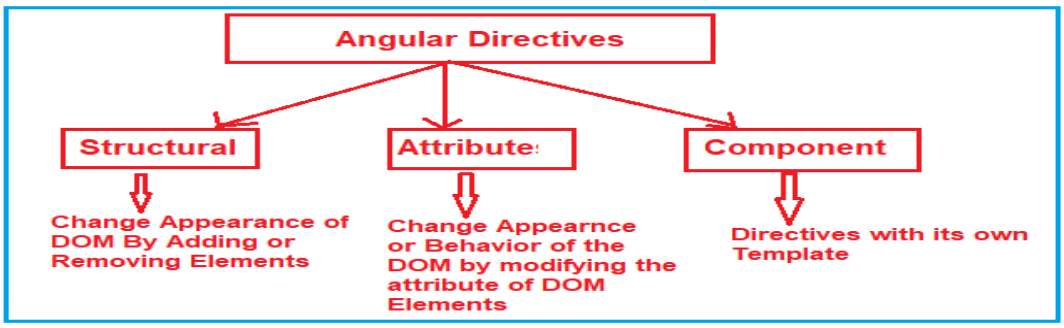
@Input() decorator in a child component or directive signifies that the property can receive its value from its parent component.



First, import Input

@Output() decorator in a child component or directive let's data flow from the child to the parent.

Directives are the elements which are basically used to change the appearance or layout of the DOM (Document Object Model) element.
Directives are attributes that allow you to write new HTML syntax, specific to your application. {SCA}
A directive is a class in Angular that is declared with a @Directive decorator.



Structural directives are directives which change the DOM layout by adding and removing DOM elements. Structural Directives are responsible for the HTML layout. Every structural directive has a ' * ' sign before them. We can apply these directives to any DOM element.
NgIf, NgForOf, NgSwitch
NgFor directive is used to iterate over a collection of data. The syntax to use ngFor directive is:
*ngFor="let <value> of <collection>"

ngIf directive works on the basis of a boolean true and false results of a given expression. If the condition is true, the elements will be added into the DOM layout otherwise they simply removed from the DOM layout.
<div *ngIf="isValid"> The Data is valid.

Attribute directives -Change the appearance or behavior of DOM elements and Angular components with attribute directives.

1- *NgStyle*: This NgStyle Attribute Directive is basically used to modify the element appearance. <button [ngStyle]="{'color':'red', 'font-weight': 'bold', 'font-size.px':20}">Click Me </button>

2- *NgClass:* This NgClass Attribute Directive is basically used to change the class attribute of the element in the DOM or in the Component to which it has been attached. ngClass directive is used to add or remove CSS classes dynamically (run-time) from a DOM Element.

ngStyle directive is used to set the DOM element style properties.

ngClass directive is used to add or remove CSS classes dynamically (run-time) from a DOM Element. <h3 [ngClass]="'one'">Angular ngClass with String</h3>

Component Directive requires a view along with its attached behavior and this type of directive adds DOM Elements. The Component Directive is a class with @Component decorator function.

Pipes (|)- Pipe takes the raw data as input and then transforms the raw data into some desired format. Pipes are simple functions to use in template expressions to accept an input value and return a transformed value. <td>{{student.Name | uppercase}}</td>

Example- lowercase, uppercase, titlecase, decimal, date, percent, currency etc.

two types i.e. Built-in Pipes and Custom Pipes

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

Syntax:

```
data | pipeName : parameter 1 : parameter 2 : parameter 3 ... parameter n
```

Examples:

Date:- {{DOB | date : "short"}}

Currency:- {{courseFee | currency : 'USD' : true : '1.3-3'}}

How to create a Custom Pipe? ng g pipe MyTitle –flat

In order to create a custom pipe in angular, you have to apply the @Pipe decorator to a class which you can import from the Angular Core Library. The @Pipe decorator allows you to define the pipe name that you will use within the template expressions

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
   name: 'pipeName'
})

export class pipeClass implements PipeTransform {
  transform(parameters): returntype {}
}
```

Pure pipes are the pipes that execute when it detects a pure change in the input value. A pure change is when the change detection cycle detects a change to either a primitive input value (such as String, Number, Boolean) or object reference (such as Date, Array, Function, or Object).
Impure pipes are the pipes that execute when it detects an impure change in the input value. An impure change is when the change detection cycle detects a change to composite objects, such as adding an element to the existing array.

What is Eager and Lazy Loading?

The application module i.e. AppModule is loaded eagerly before application starts. Whereas there is something called feature modules which holds all the features of each module which can be loaded in two ways namely eagerly or lazily or even preloaded in some cases.

Eager loading: To load a feature module eagerly we need to import it into an application module using imports metadata of @NgModule decorator. Eager loading is highly used in small size applications. In eager loading, all the feature modules will be loaded before the application starts.This is the reason the subsequent request to the application will always be faster.

Lazy loading: To load a feature module lazily we need to load it using loadChildren property in route configuration and that feature module must not be imported in the application module. Lazy loading generally is useful when the application is growing in size. In lazy loading, feature modules will be loaded on demand and hence application start will be faster.
Routing is a mechanism which is used for navigating between pages & displaying appropriate components or pages on the browser.

```
const routes: Routes = [
  {path:'studentLink', component:StudentComponent},
  {path:'studentdetailsLink',component:StudentdetailComponent}
  ];
@NgModule({
  imports: [RouterModule.forRoo
t(routes)],
```

```
  exports: [RouterModule]
})
```

```
<a [routerLink] = "['/studentLink']" >Student</a> <br/>
<a [routerLink] = "['/studentdetailsLink']" >student details</a>
```

Router Outlet is a dynamic component that the router uses to display views based on router navigation. RouterOutlet is one of the router directives <router-outlet>

With the help of routerLink directive, you can link to routes of your application right from the HTML Template.

RouterLink is an anchor tag directive that gives the router authority over those elements. Because the navigation routes are set. With the help of routerLink directive, you can link to routes of your application right from the HTML Template.
routerLink is the selector for the RouterLink directive that turns user clicks into router navigations. You can assign a string to the Router link. This directive generates the link based on the route path.
```
<a routerLink="/home" >Home Page of our website</a>
```
Server Side :
```
constructor(private router : Router){} GetStudent() { this.router.navigate(['/studentLink']); }
```

Redirecting Routes :

When the application starts, it navigates to the empty route by default. We can configure the router to redirect to a named route by default. So, a redirect route translates the initial relative URL (”) to the desired default path. For example, if you may want to redirect to the Login page or registration page by default when the application starts. Then you need to configure the redirectTo as shown below.

```
const routes: Routes = [
  {
    path:'', redirectTo:'Login',pathMatch:'full'
  },
  {
    path:'studentLink', component:StudentComponent
  },
  {
    path:'studentdetailsLink', component: StudentdetailComponent
  },
  {
    path:'Login', component:LoginComponent
  }
];
```

2

1

Wildcard route has a path consisting of two asterisks (**). It matches every URL, the router will select this route if it can’t match a route earlier in the configuration. A Wildcard Route can navigate to a custom component or can redirect to an existing route.

```
{
  path:'**', component:CustomerrorComponent
}
```

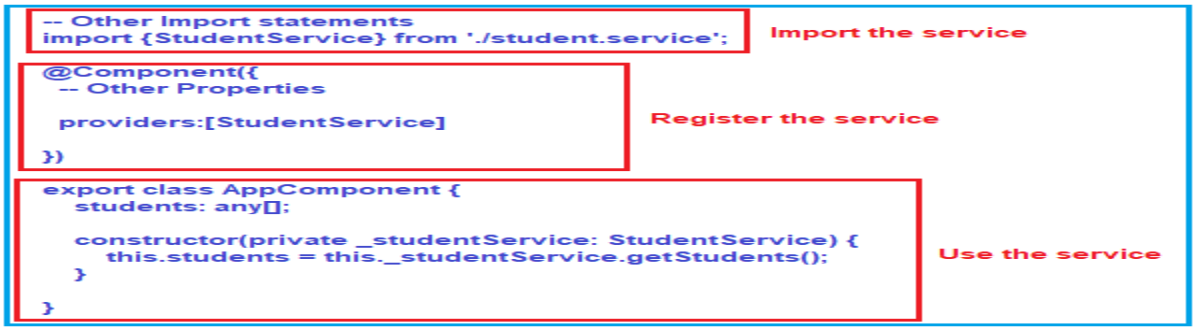
Order of Angular Routes- Routes with a static path should be placed first, followed by the empty path route that matches the default route. The wildcard route should be the last route in your router configuration.

factory() is a function which works similar to the service() but is much more powerful and flexible. factory() are design patterns which help in creating Objects.

Services are the pieces of code or logic that are used to perform some specific task. A service can contain a value or function or combination of both. The Services are injected into the application using the dependency injection mechanism. Whenever you need to reuse the same data and logic across multiple components of your application, then you need to go for angular service. ng generate service Student
Service is composed of three things. You need to create an export class and you need to decorate that class with @Injectable decorator and you need to import the Injectable decorator from the angular core library.

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

@Injectable()

export class ServiceName {

   //Method and Properties

}
```

ngOnInit is the right place to call a service method to fetch data from a remote server. We can also do the same using a class constructor, but the general rule of thumb is, tasks that are time consuming should use ngOnInit instead of the constructor. As fetching data from a remote server is time consuming, the better place for calling the service method is ngOnInit.



How to create a service in Angular?

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

1- Factory 2- Service 3- Provider

Dependency Injection, or DI, is a design pattern and mechanism for creating and delivering some parts of an application to other parts of an application that require them. Angular supports this design pattern and you can use it in your applications to increase flexibility and modularity. [angular.io]

Dependency Injection (DI) is a software design pattern where the objects are passed as dependencies rather than hard-coding them within the component. The concept of Dependency Injection comes in handy when you are trying to separate the logic of object creation to that of its consumption. The 'config' operation makes use of DI that must be configured beforehand while the module gets loaded to retrieve the elements of the application. With this feature, a user can change dependencies as per his requirements.

Dependency Injection in Angular is a combination of two terms i.e. Dependency and Injection.
Dependency: Dependency is an object or service that is going to be used by another object.
Injections: It is a process of passing the dependency object to the dependent object. It creates a new instance of the class along with its required dependencies.

```
//Register Dependency at App Level
providers: [StudentService],
```

Angular lifecycle hooks are nothing but callback functions, which angular invokes when a certain event occurs during the component's life cycle.

A component has a life-cycle, a number of different phases it goes through from birth to death. We can hook into those different phases to get some pretty fine grained control of our application. To do this we add some specific methods to our component class which get called during each of these life-cycle phases, we call those methods hooks.

ngOnChanges ngOnInit ngDoCheck
ngAfterContentInit ngAfterContentChecked ngAfterViewInit
ngAfterViewChecked ngOnDestroy

Constructor -Life Cycle of a component begins, when Angular creates the component class. The first method that gets invoked is class Constructor. Constructor is neither a life cycle hook nor it is specific to Angular. It is a Javascript feature. It is a method which is invoked when a class is created. Angular makes use of a constructor to inject dependencies

ngOnChanges()- Respond when Angular sets or resets data-bound input properties. The method receives a SimpleChanges object of current and previous property values.
Called before ngOnInit() (if the component has bound inputs) and whenever one or more data-bound input properties change.

ngOnInit() Initialize the directive or component after Angular first displays the data-bound properties and sets the directive or component's input properties.
Called once, after the first ngOnChanges(). ngOnInit() is still called even when ngOnChanges() is not (which is the case when there are no template-bound inputs).
ngOnInit is a life cycle hook used by Angular to signify that the component has finished being created.
The ngOnInit is defined in the following way:
To use OnInit, we must import it into the component class as follows:

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

It is not necessary to implement OnInit in every component.
ngDoCheck()- Detect and act upon changes that Angular can't or won't detect on its own. Called immediately after ngOnChanges() on every change detection run, and immediately after ngOnInit() on the first run.

ngAfterContentInit()- Respond after Angular projects external content into the component's view, or into the view that a directive is in.
Called once after the first ngDoCheck().

ngAfterContentChecked()- Respond after Angular checks the content projected into the directive or component. Called after ngAfterContentInit() and every subsequent ngDoCheck().

ngAfterViewInit()- Respond after Angular initializes the component's views and child views, or the view that contains the directive. Called once after the first ngAfterContentChecked().

ngAfterViewChecked()- Respond after Angular checks the component's views and child views, or the view that contains the directive. Called after the ngAfterViewInit() and every subsequent ngAfterContentChecked().

ngOnDestroy()- Cleanup just before Angular destroys the directive or component. Unsubscribe Observables and detach event handlers to avoid memory leaks.
Called immediately before Angular destroys the directive or component.

What are the lifecycle hooks for components and directives?

An Angular component has a discrete life-cycle which contains different phases as it transits through birth till death. In order to gain better control of these phases, we can hook into them using the following:

- constructor: It is invoked when a component or directive is created by calling new on the class.
- ngOnChanges: It is invoked whenever there is a change or update in any of the input properties of the component.
- ngOnInit: It is invoked every time a given component is initialized. This hook is only once called in its lifetime after the first ngOnChanges.
- ngDoCheck: It is invoked whenever the change detector of the given component is called. This allows you to implement your own change detection algorithm for the provided component.
- ngOnDestroy: It is invoked right before the component is destroyed by Angular. You can use this hook in order to unsubscribe observables and detach event handlers for avoiding any kind of memory leaks.

What do you understand by dirty checking in Angular?

In Angular, the digest process is known as dirty checking. It is called so as it scans the entire scope for changes. In other words, it compares all the new scope model values with the previous scope values. Since all the watched variables are contained in a single loop, any change/update in any of the variable leads to reassigning of rest of the watched variables present inside the DOM. A watched variable is in a single loop(digest cycle), any value change of any variable forces to reassign values of other watched variables in DOM
Provider is a configurable service in Angular. It is an instruction to the Dependency Injection system that provides information about the way to obtain a value for a dependency. It is an object that has a $get() method which is called to create a new instance of a service. A Provider can also contain additional methods and uses $provide in order to register new providers.

_____
Observables- A sequence of items that arrive asynchronously over time.
HTTP call - single item
Single item- HTTP response

observable updates in the latest version of Angular?

- Faster Builds
- Angular ESLint Updates
- Internet Explorer Updates
- Webpack 5 Support
- Improved Logging and Reporting
- Updated Language Service Preview

- Updated Hot Module Replacement (HMR) Support

RxJS - Reactive extensions for JS, External library to work with Observables.

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

A service() in Angular is a function that is used for the business layer of the application. It operates as a constructor function and is invoked once at the runtime using the 'new' keyword. Whereas factory() is a function which works similar to the service() but is much more powerful and flexible. factory() are design patterns which help in creating Objects.
$scope is used for implementing the concept of dependency injection (D.I) on the other hand scope is used for directive linking.
$scope is the service provided by $scopeProviderwhich can be injected into controllers, directives or other services whereas Scope can be anything such as a function parameter name, etc.
AOT stands for Ahead-of-Time compiler. It is used for pre-compiling the application components and along with their templates during the build process. Angular applications which are compiled with AOT have a smaller launching time. Also, components of these applications can execute immediately, without needing any client-side compilation. Templates in these applications are embedded as code within their components. It reduces the need for downloading the Angular compiler which saves you from a cumbersome task. AOT compiler can discard the unused directives which are further thrown out using a tree-shaking tool.
Pipes are functions that simplify the process of wiring up your JavaScript expressions and transforming them into their desired output.

What are the different types of filters in Angular?

Filters in Angular are used for formatting the value of an expression in order to display it to the user. These filters can be added to the templates, directives, controllers or services. Not just this, you can create your own custom filters. Using them, you can easily organize data in such a way that the data is displayed only if it fulfills certain criteria. Filters are added to the expressions by using the pipe character |, followed by a filter.
Below are the various filters supported by Angular:

- currency: Format a number to a currency format.
- date: Format a date to a specified format.
- filter: Select a subset of items from an array.
- json: Format an object to a JSON string.
- limit: To Limit an array/string, into a specified number of elements/characters.
- lowercase: Format a string to lowercase.
- number: Format a number to a string.
- orderBy: Orders an array by an expression.
- uppercase: Format a string to uppercase.

Angular expressions are code snippets that are usually placed in binding such as {{ expression }} similar to JavaScript. These expressions are used to bind application data to HTML. Syntax: {{ expression }}

Transpiling refers to the process of conversion of the source code from one programming language to another. In Angular, generally, this conversion is done from TypeScript to JavaScript. It is an implicit process and happens internally.

HTTP interceptors Using the HttpClient, interceptors allow us to intercept incoming and outgoing HTTP requests. They are capable of handling both HttpRequest and HttpResponse. We can edit or update the value of the request by intercepting the HTTP request, and we can perform some specified actions on a specific status code or message by intercepting the answer.

Events are specific directives that help in customizing the behavior of various DOM events. Few of the events supported by Angular are listed below: ng-click ng-copy ng-cut ng-dblclick ng-keydown ng-keypress ng-keyup ng-mousedown ng-mouseenter ng-mouseleave ng-mousemove ng-mouseover ng-mouseup ng-blur

List some tools for testing angular applications?

Karma, Angular Mocks, Mocha, Browserify, Sion

REST stands for REpresentational State Transfer. REST is an API (Application Programming Interface) style that works on the HTTP request. In this, the requested URL pinpoints the data that needs to be processed. Further ahead, an HTTP method then identifies the specific operation that needs to be performed on that requested data. Thus, the APIs which follows this approach are known as RESTful APIs.
Bootstrapping is nothing but initializing, or starting the Angular app. Angular supports automatic and manual bootstrapping.

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

Subscribing- in Angular, .subscribe() is basically a method on the Observable type. The Observable type is a utility that asynchronously or synchronously streams data to a variety of components or services that have subscribed to the observable.

Subscribe takes 3 methods as parameters each are functions:

- next: For each item being emitted by the observable perform this function
- error: If somewhere in the stream an error is found, do this method provider
- complete: Once all items are complete from the stream, do this method

Difference between a provider, a service and a factory

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

Describe how you will set, get and clear cookies?

For using cookies in Angular, you need to include a module called ngCookies angular-cookies.js.
To set Cookies – For setting the cookies in a key-value format 'put' method is used.

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

To get Cookies – For retrieving the cookies 'get' method is used.

```
1    cookie.get('nameOfCookie');
```

To clear Cookies – For removing cookies the 'remove' method is used.

```
1    cookie.delete('nameOfCookie');
```

Key differences between observables and promises are

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.

List the Differences Between Just-In-Time (JIT) Compilation and Ahead-Of-Time (AOT) Compilation?

Angular provides two types of compilation:

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

In JIT compilation, the application compiles inside the browser during runtime. Whereas in the AOT compilation, the application compiles during the build time.
With JIT, the compilation happens during run-time in the browser. It is the default way used by Angular. The commands used for JIT compilation are –> ng build ng serve
In AOT compilation, the compiler compiles the code during the build itself. The CLI command for aot compilation is –> ng build --aot ng server -aot
AOT is more suitable for the production environment whereas JIT is much suited for local development.

Angular Material is a user interface component package that enables professionals to create a uniform, appealing, and fully functioning websites, web pages, and web apps. It does this by adhering to contemporary web design concepts such as gentle degradation and browser probability.

What is the purpose of FormBuilder?

The FormBuilder is a syntactic sugar that speeds up the creation of FormControl, FormGroup, and FormArray objects. It cuts down on the amount of boilerplate code required to create complex forms.
When dealing with several forms, manually creating multiple form control instances can get tedious. The FormBuilder service provides easy-to-use control generation methods.
Follow the steps below to use the FormBuilder service:

- Import the FormBuilder class to your project.
- FormBuilder service should be injected.
- Create the contents of the form.

To import the FormBuilder into your project use the following command in the typescript file:

```
import { FormBuilder } from '@angular/forms';
```

BehaviorSubject holds one value (so we actually need to initialize a default value). When it is subscribed it emits that value immediately.
const subject = new Rx.BehaviorSubject(0); subject.next(1); subject.subscribe(x => console.log(x));
o/p- 1
A Subject on the other hand, does not hold a value.
const subject = new Rx.Subject(); subject.next(1); subject.subscribe(x => console.log(x));
o/p-empty

What it actually means is that in Subject, the subscribers will only receive the upcoming value whereas in BehaviorSubject the subscribers will receive the previous value and also upcoming value.

RxJS Observables	Promises
Observables are used to run asynchronously, and we get the return value multiple times.	Promises are used to run asynchronously, and we get the return value only once.
Observables are lazy.	Promises are not lazy.
Observables can be canceled.	Promises cannot be canceled.
Observables provide multiple future values.	Promises provide a single future value.

Guard

The guard is a special class that can be used to make sure that a secured page is not accessible for any user who is not authenticated with a valid JWT token. This is the entry point to achieve this tutorial's goal which is to Secure Angular Site using JWT Authentication. ng g guard helpers\auth

Interceptors

With interceptors, implementing the HttpInterceptor, you can inject some code into the http process pipeline to modify or process the outgoing/incoming http requests/responses.
ng g interceptor interceptors/auth

There are five ways to share data between components:

- Parent to child component
- Child to parent component
- Sharing data between sibling components

- Sharing data using ViewChild property
- Sharing data between not related components

1. **Parent to Child component:** When you define `@Input()` in the child component it will receive value from the master or parent component. Before sharing data between components first we define our data model using interfaces. Here we are defining Product interface with productID and productName as mandatory field and rest are optional.

Now we import the product interface in our parent component. Now construct template. Here template has a product id and product name. Now I want to share product name to child component. And once the child component updates the product name I want data back to the parent component.

Parent component code:

Child component code:

Now we define the following decorator in the child component. ***childToMaster*** is the property name that we want to share from the parent component. Input decorator clearly shows that the child component is expecting some value from parent components.

```
@Input('childToMaster') masterName: string;
```

In parent component, we will set ***childToMaster*** property.

```
<app-child [childToMaster]=product.productName
(childToParent)=”childToParent($event)”></app-child>
```

Now here we are sharing product.productName property to child component. Once you start writing productName it will automatically be shared to the child component.

2. **Child to Parent component:** For sharing data from child to parent we need output decorator. In child component we have to define output decorator like this:

```
@Output() childToParent = new EventEmitter<String>();
```

Now we want to share this property with a parent component. So sharing data from the child component requires to ***bubble the event with value to the parent components. We can bubble the event based on a trigger point***, here we are doing it by using a button click. sendToParent is called on button click.

```
sendToParent(name){
this.childToParent.emit(name);
}
```

Now we have to associate childtoParent to the property to the child tag in the parent component.

```
<app-child [childToMaster]=product.productName
(childToParent)=”childToParent($event)”></app-child>
```

Now once the value is received in parent component you have to set it.

```
childToParent(name){
this.product.productName=name;
}
```

Wow now we can see the product name changes in parent components.

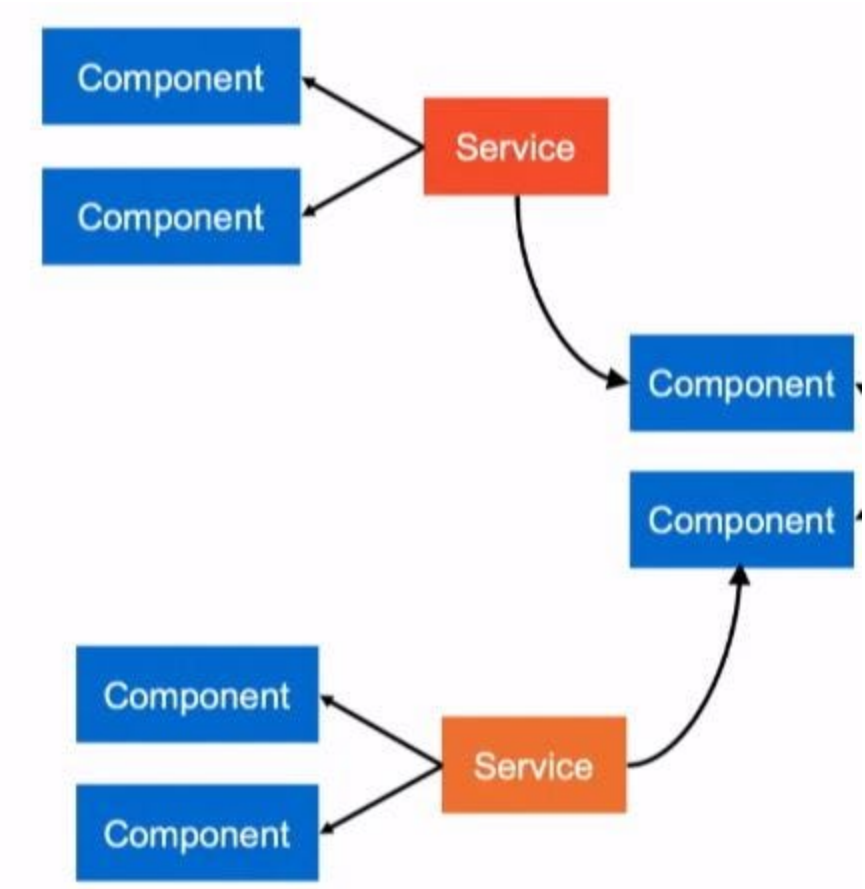
3. **Sharing data between sibling components:** Sharing data between siblings can be done by using points 1 and 2. First share data between the ***child to parent*** using output decorator and EventEmitter. Once received data in ***parent component share it to another child*** component using Input decorator. So siblings can talk each other via parent components.

4. **More Power: Sharing data using ViewChild decorator:** ViewChild allows child component to be ***injected in parent component. So this make ViewChild more powerful.*** It allows parents to controls the child's methods and properties. But a parent can get access to the properties after view init event. That means we have to implement ngAfterViewInit life cycle hook in order to get the properties from parent components.

```
@ViewChild(AppChildComponent) child;
constructor() { }
ngAfterViewInit() {
this.product.productName=child.masterName; //<= This will set data
}
```

Here **masterName** is defined in the child component. ViewChild gives a reference to the child component in parent components. And it sets the value of child masterName in parent productName.

5. **Sharing data between not related components:** When there is no relation between the component we can not pass the data using the above four methods. This happens when your components are in different modules. There are other scenarios when you have list of products and click on a particular product and then redirect to product details components. In these kinds of scenarios, we have to use data service to share data between components.



Complex Scenario- Sharing data using BehaviourSubjects

For creating data service. We have to define BehaviorSubject. BehaviorSubject holds the current value and the last value.

I always prefer to use BehaviorSubject because of following reasons:

- It automatically updates the lates value wherever subscribed.
- Always give last value when called via getValue() method.
- No need to call next, just create a set and get method in order to get value.

In the data service, I have created messageSource as BehaviorSubject. This accepts editDataDetails as any. We can create editDataDetails as the type of Product interface which is a much better practice. Create changeMessage method which will set the current value of observables.

```
export class SharedDataService { constructor(){} //Using any

public editDataDetails: any = [];
public subject = new Subject<any>();
private messageSource = new  BehaviorSubject(this.editDataDetails);
currentMessage = this.messageSource.asObservable();
changeMessage(message: string) { this.messageSource.next(message)} }
```

Now we have to share this service for the component where we want to set the values. Call changeMessage method with value as parameter to set the value. In components where we want to receive the value subscribe to it. Once you subscribe you will always get the latest value without any code changes.

```
   //Set value in component 1

this.sharedDataService.changeMessage("message here");
//Get value in component 2
selectedMessage:any;
ngOnInit() {
this.sharedDataService.currentMessage.subscribe(message =>
(this.selectedMessage= message)); //<= Always get current value!
}
```

Bonus 1: Creating Get and Set Property and BehaviourSubject:

In the above example, we can optimize more using Product interface and create a getter and setter method by following method.

```
import { Injectable } from '@angular/core';
import { Subject, BehaviorSubject, Observable,ReplaySubject } from
'rxjs';
```