

# ANGULAR NOTES

By: Deepa Chaurasia

<https://www.linkedin.com/in/deepa-chaurasia-3704351a8>

# What is Data Binding?

→ Communication between typescript code of your component and the template (html) that the user see.

Data Binding == Communication

We get different ways of communication.

For eg → we want to output data from our typescript code in html code.  
we can use String Interpolation for this

Syntax for String Interpolation

{} {} data {} {}

you put your data that will come from ts in double curly braces

## Eg of String Interpolation

Eg → In your ts file.

```
export class MyComponent {
```

```
  name: string = 'deepa';
```

3

In your html file

```
<p> My name is {{name}} </p>
```

It will print My name is deepa.

⇒ Now Suppose you have a button to submit.  
It must be enabled only if you're admin.

In this case you'll use property binding.

★ <sup>native</sup>  
The property [disabled] will bind to boolean  
which we'll define in ts.

## Eg of Property binding.

In ts File:

```
export class MyComponent {
```

```
  isAdmin: boolean = true;
```

3

If not admin then disable

In html

```
<button [disabled] = "!isAdmin"> Add </button>
```

Event Binding → It allows your component to react to user's actions such as button click, key strokes, etc.

Like property Binding use [ ] square brackets

Event Binding use () parenthesis

Let's say on click of button I want to show something.

In ts File

```
export class MyComponent {
```

```
    name: String;
```

```
    onClick(): void {
```

```
        this.name = 'deepa';
```

```
}
```

In html

```
<p> My name is ${name} </p>
```

```
<button (click) = "onClick()"> Add </button>
```

So before clicking Add, name will be blank

After clicking Add, name will be shown i.e.

(deepa)

# What are Directives?

Directives are instruction in the DOM!

Basically these are custom html attributes which tell angular to change the style or behaviour of Dom elements.

Components are building blocks of Angular applications, Similarly directives are also building block of Angular Applications.

## Types of Directives

Component      Structural Directives      Attribute Directives

Directives  
with a  
template

Change layout  
of the elements

Change appearance  
or behaviour  
of a particular  
element

## 1. Component Directives

- \* Directives must be declared in Angular Modules in the same manner as components.
- \* Directive in Angular is a re-usable component.
- \* The other 2 directives such as structural and attribute do not have templates.

If is decorated with @Component decorator

Example:- change-text directive.

change-text.directive.ts

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

@Directive({

selector: '[changeText]'

)

export class ChangeTextDirective {

constructor (Element: ElementRef) {

~~Element~~

Element.nativeElement.innerText =

"Text is changed";

} }

app.component.html

<span changeText> Welcome </span>

In above eg, there is class called ChangeText Directive and a constructor, which takes the element of type ElementRef, which is mandatory.

The element has all the details to which Change Directive is applied.

2.

## Structural Directives :

- \* Structural Directives are responsible for changing structure of DOM
- \* They work by adding or removing element from DOM.
- \* The Structural directive always starts with '\*'.

Three most popular Structural Directive

- 1) ngIf
- 2) ngFor
- 3) ngSwitch

All of them work same as if  
for or switch respectively.

Except that you use it for DOM Tree  
in html template

Eg →

```
<div *ngIf = "info" class = "name">
```

```
{&gt; info.name &lt;/div>
```

```
<div *ngFor = "let movie of movies">  
  &lt;div>{{movie.name}} &lt;/div>
```

### 3) Attribute Directives

Attribute Directives manipulate Dom by changing its behaviour and appearance.

- \* It is used to show or hide elements or dynamically change behaviour of component.
- \* Built in Attributes are `ngStyle`, `ngClass` etc.

# Pipes In Angular

## What are Pipes?

Pipes are a feature built into Angular 2 which allows you to transform output in your template

Eg. You have a property `username = 'deepa'`  
you want to output `username` in your template  
using string interpolation

like `<p> {{username}} </p>`

Now you want your name to be in Upper Case  
only in output on template

You don't wanna change property in your ts  
you want to transform the way it is displayed  
on screen

For this you could simply use uppercase pipe  
which is a built-in pipe

Like this `<p> {{username | uppercase}} </p>`

Now Output → DEEPA

# Parametrizing Pipes

Suppose you have date in a format  
8 April, 2022

You don't want this format, you want to customize it

You can do it using parametrized pipes

ts file

```
students = [  
  { name: 'deepa',  
    class: '10',  
    dob: new Date(14, 12, 1999) }]
```

Now In html

```
<p> {{ students | date }} </p>
```

```
<p> {{ students | date : 'fullDate' }} </p>
```

You can refer angular.io to know more about built in pipes

## Chaining Multiple Pipes

```
<p> {{ students | dob | date : 'FullDate' | Uppercase }}</p>
```

Here order in which we write pipe matters because it iterates from left to right

- \* first FullDate pipe will apply
- \* on the result Uppercase will apply

## Creating a Custom Pipe.

- \* create ts file shortenPipe.ts (you can use any name instead of shorten)
- \* your pipe.ts should look like this

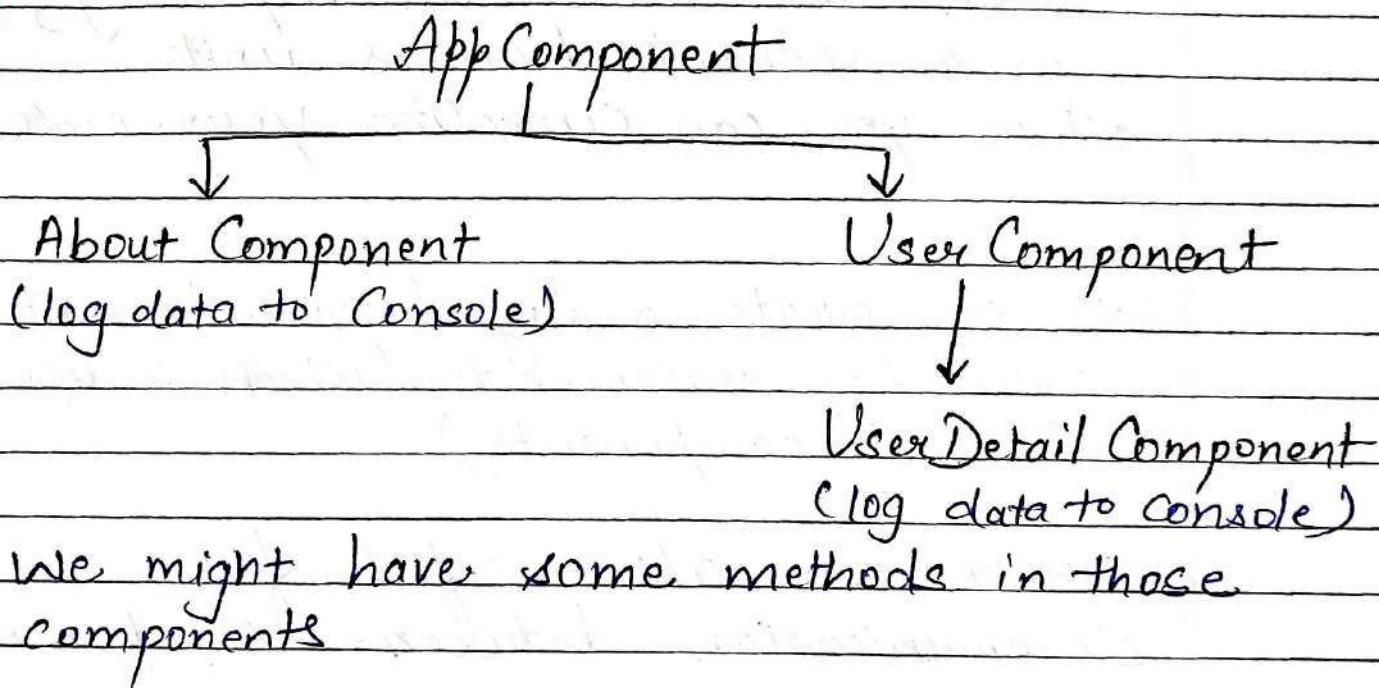
```
Imp. import { Pipe, PipeTransform } from 'angular/core';
R
@Pipe({ name: "myPipe" })
* compulsory
  export class MyPipe implements PipeTransform {
    method
    in which
    all operation
    happens
    ↪ transform (value: any) {
      return value;
    }
  }

  In html → <p> {{ students.name | myPipe }}</p>
```

# Services and Dependency Injection

Angular services are ~~single~~ singleton object that gets instantiated only once in a lifetime of an application

Consider an application having:



We might have some methods in those components

Let's say we want to log some data in About Component and User Detail Component

Both logs are same, so basically, we are duplicating code in 2 different component

Suppose in User Component, we also want to access some data, some array of users and we don't know if we will later use that in other part of application

## Both User Cases

- \* Duplication of Code
- \* Data Storage

In both cases we typically use Service

A Service is just another piece in your Angular App, Another class which acts as a central repository, as a central business unit where you can centralize your code

We can create a log Service to centralize our log statement (which is used in 2 components)

Service are also useful for communication between components

## How to create a Service

- \* Simply run command  
ng g s service-name

Your Service will be generated

# How do you Access a Service In Your Component?

Let's say you have made service named Logging Service

you can simply instantiate your Logging Service in component

account.component.ts

- \* ~~You can~~ const service = new LoggingService();
- \* Import the Logging Service
- \* service.logStatus(status);  
    ↓

Your method you wanted to call from service in component.

But this is not suggested, Angular provides us more better way to Inject Service in Our Component.

# Angular Dependency Injector

Angular provides us more better way to inject service in component. It is Angular dependency injector.

So, we have dependency on logging Service from our component and dependency injector simply injects this dependency, instance of our class automatically into our component.

We need to only inform Angular, that we require such an instance.

How do we inform that?

- \* We add constructor to the component you'll see constructor present if you have made service through component
  - \* Inside constructor, I can bind it to property
- Constructor. (private LoggingService: LoggingService)  
    we use this access specifies  
    must  
    import type and that  
    service should be
- (make sure to import LoggingService at top)

Now Angular know we want logging Service.  
But it doesn't know how to give instance.

- \* We need to provide a service.  
Add one providers property in @Component decorator.

@ Component ({  
  selector :  
  templateURL :  
  styleURL :  
  providers : [Logging Service] } ) @ {  
  [Here we  
  have provided]

## Hierarchical Injector

Angular dependency injector is actually a hierarchical injector.

It means that if we provide a service in one component, then Angular framework knows how to create instance of service for this component and important, all it's child components.

All of them will receive same instance of service

## Hierarchical Injector

App Module  $\rightarrow$  Same Instance of Service  
is available Application-wide

App Component  $\rightarrow$  Same Instance of Service  
is available for all components  
(but not for other service)

Any Other Component  $\rightarrow$  Same Instance is available  
for component and all its child  
components.

It works Top to down not down to Top

Suppose If you want to use same  
Instance of Service in AppComponent  
to child component

Then you don't use providers in  
child component

Everything else stays same.

# Injecting Services into Services

The highest level of providing service.  
is provide your service in providers array of App.Module.TS

If we are providing service in module, we make sure we are using some instance of service everywhere, in every component throughout app Unless we are overriding our service.

# Route Guards

What is Route Guards?

They are interface provided by angular which when implemented allows us to control the accessibility of route based on condition provided in class implementation of that Interface

Five types of Route Guards

- \* Can Activate
- \* CanActivateChild
- \* Can Load
- \* Can Deactivate
- \* Resolve

1. Can Activate → Can Activate

↓ implement

AuthGuard Service

(define class AuthGuard  
Implement canActivate Method)

will call  
everytime user navigate  
to this field

{ path: 'user/:id', component: HomeComponent  
canActivate: [AuthGuard] }

## // Auth Guard Service

canActivate ( route: ActivatedRouteSnapshot,  
state: RouterStateSnapshot ) :

Observable<boolean> | Promise<boolean> }

{

return true;      (If navigation happen)

}

Now Use this Service

2. canActivate Child - Similar to canActivate  
but also prevent access to child  
routes of given route.

3. canLoad - In angular we implement  
lazy loading to download only required  
modules using loadChildren.

If we want to prevent unauthorized  
user we can use canActivate Guard.

that will do job but also download Module

Now to control navigation as well as  
prevent downloading of that module  
we can use canLoad Guard

canLoad ( Route: Route, Segments: URLSegment[] )  
: Observable<boolean> | Promise<boolean>

{

}

return true; (whether want to  
download module or not.)

4. canDeactivate - This guard is useful  
when you want to prevent user  
from accidentally navigating  
away without saving or  
some undone task.

canDeactivate ( component: T, currentRoute:  
ActivatedRouterSnapshot, currentState:  
RouterStateSnapshot, nextState?: ):  
Observable<boolean> | UrlTree | Promise  
<boolean> | UrlTree | boolean | UrlTree,

Here T is interface, that we will be using  
in component as well as guard

## 11 CanDeactivate Service

→ export interface canComponentDeactivate {

canDeactivate : () => Observable<boolean>

| Promise<boolean> | boolean;

}

→ export class CanDeactivate implements

CanDeactivate & (canDeactivateComponentDeactivate)

{

CanDeactivate.Component : CanComponentDeactivate  
> currentRoute, currentState, nextState)

{

return component.canDeactivate();

}

}

Now Implement canComponentDeactivate  
interface in component.

→ export class EditComponent implements OnInit,  
CanComponentDeactivate {

canDeactivate () : Observable<boolean>  
| Promise<boolean> {

{

{

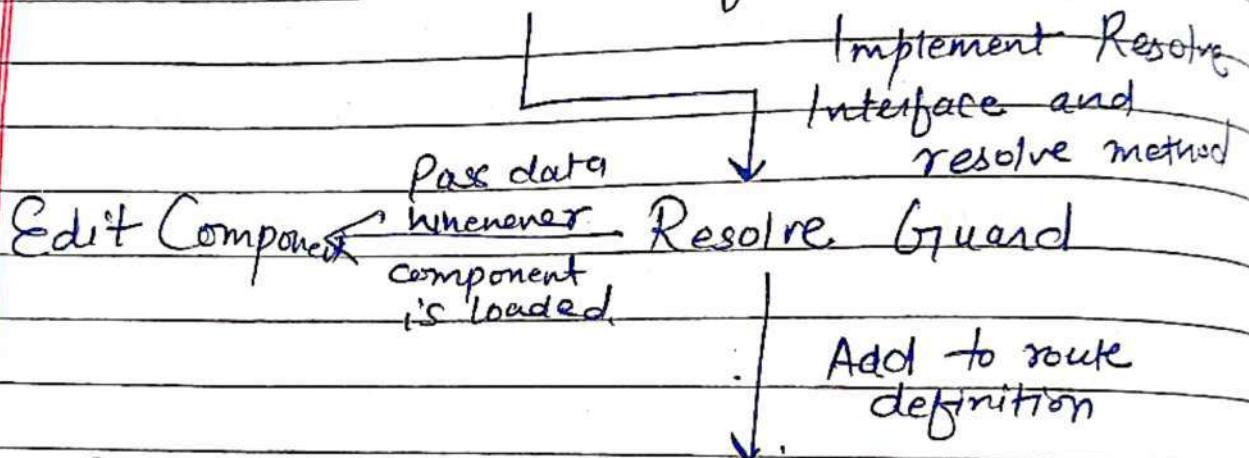
## 5. Resolve Guard

While data communication b/w components

Sometimes data is so heavy that is not possible to pass data through query params.

To handle this we have Resolve Guard

### Resolve Interface



```
{ path: 'id', component: EditComponent,  
  resolve: { data: ResolveGuard } }
```

### // Resolve Guard Service.

```
resolve ( route, state )
```

```
{  
  return this.fetchService.get(+route.params['id'])
```



- By Deepa Chawasia

# What are Modules In Angular?

Angular doesn't scan automatically all your components, directives etc.

You have to tell Angular what component, directives, service your project have. Or you are using.

Modules does that Job for you!

Module tell Angular list of all components, directives or services we are using.

It is basically a place where you group together all your components, directives, services.

@NgModule ({

  declaration: [ AppComponent,  
    XComponent,  
    YComponent ],

  imports: [ AppRoutingModule,  
    BrowserModule, ... ],

  providers: [ ],  
  services goes here)

  bootstrap: [ AppComponent ],

  entryComponents: [ ]

})

export class AppModule { }

1. `@NgModule` - Angular analyzes `ngModule`s to understand application and its features.

You can't use a feature without including it in `@NgModule`.

2. Imports - Imports are used for importing other modules like `FormsModule`, `RoutingModule` etc. and their features into our main Module.

3. Providers - It defines all services we are providing in our angular app.

All services declare in your app must be inside the Providers of APPMODULE

OR

You have one other method to import your services throughout root of application

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

If you use `@Injectable`, you don't have to declare it in providers of Module.

Remember `@Injectable ( S )`  
providedIn : 'root'

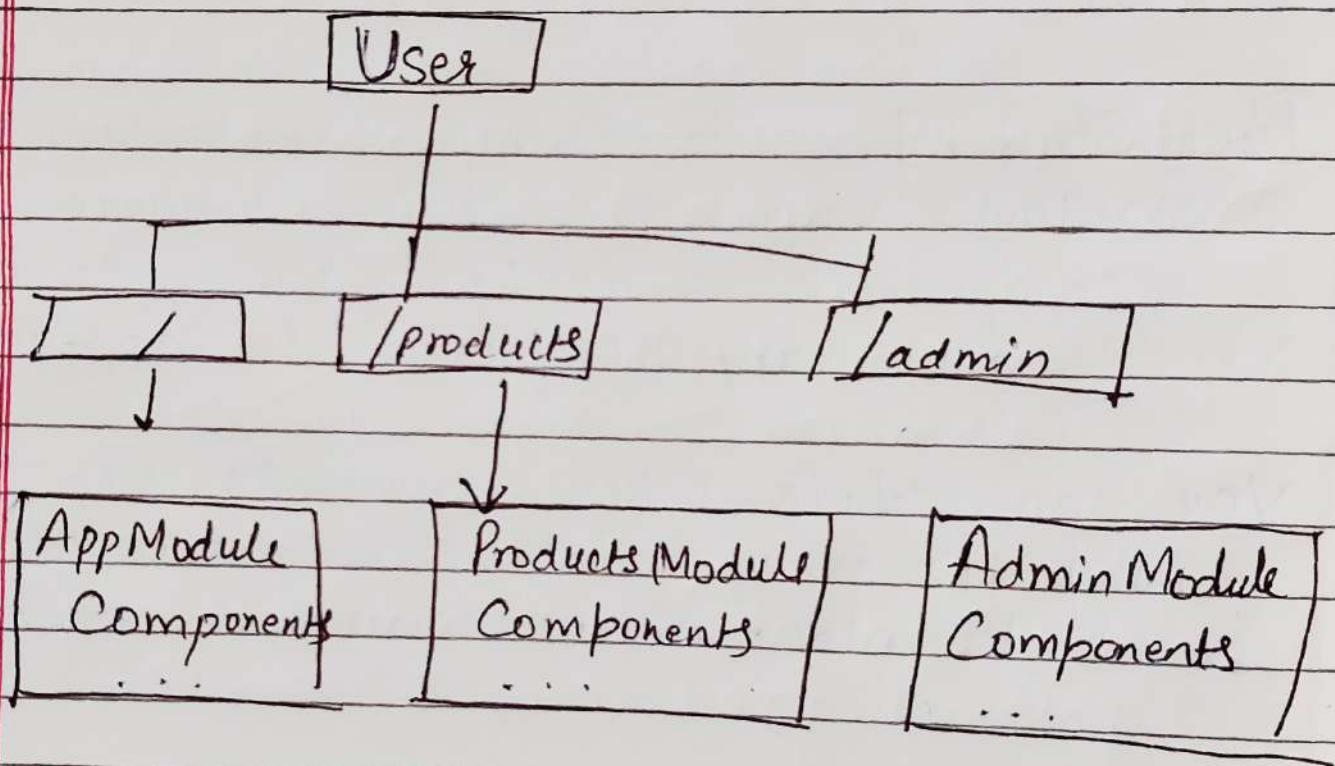
Will come above every service in Service Component.

`Bootstrap[]` - Bootstrap is important for starting your app. It defines which component is available straight in your app at first. (i.e app-root)

You can include other components also that's why it's array. But it is not likely to add other components generally

# Lazy Loading In Angular

Consider Our application has 3 routes,  
Every routes has a module  
associated with it that contains  
their respective component, services  
and directives.



Here you can see we have 3 routes

- Root
- Products
- Admin

Now When user visit root route then  
we must be loading the respective  
modules, and then we load  
other modules when required.

But does it really happen??

The answer is NO.

Angular actually load all the modules whenever we visit any page.

It's like you ordered starter, maincourse and desert

And chef is saying No I will make all starter, maincourse, desert first then start serving.

You definitely want atleast your starter, first, and ask chef to make rest when it's required.

So Lazy Loading does the same for you.

In Lazy Loading the modules are loaded only if the user hit the route of that particular Module.

# Advantages of Lazy Loading In Angular

- \* Initially we load a smaller bundle and module for our root route.
- \* We load more modules / bundle as we hit route for following.
- \* So our first page loads faster, and it doesn't take much time becoz bundles are comparatively less to download now.
- \* Basically your application size shrinks and it becomes faster.

Good Optimization Can be achieved with Lazy Loading .

→ Notes to add

- \* you can read about Preloading Strategy
- \* dynamic importing of Components and render using Component Factory for better Angular Build Optimization

Thankyou  
For  
Reading

Like, Share and Comment

<https://www.linkedin.com/in/deepa-chaurasia-notes>