**Angular (JavaScript framework)**

**Why?**

* Gives application a clear structure
* Easy to test
* Includes a lot of re-usable code
* Allows to create single page reactive application

**Commands:**

npm install –g @angular/cli@latest

cd location

ng new my-frst-app

ng serve

ng generate component comp\_name

npm install --save bootstrap@3 (file. In angular. json file, we need to add Bootstrap to the **styles []** array as **node\_modules/bootstrap/dist/css/bootstrap.min.css**)

**Loading angular Project: what is the flow?**

1. Index.html -<app-root/> -at the end of index.html, angular creates .js bundles those invokes main.ts
2. main.ts: platformBrowserDynamic (). bootstrapModule(**AppModule**) –imported from @angular-platform-browser-dynamic
3. app.module.ts:

@NgModule ({

declarations: [],

imports: [],

providers: [],

bootstrap: [AppComponent]

})

**app.component.ts :**

@Component ({

selector: ‘’,

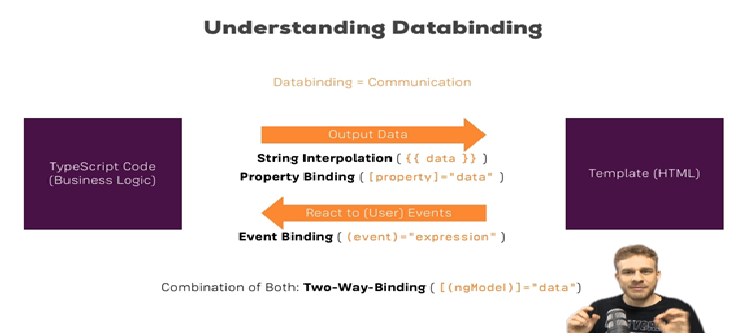
template/templateUrl: ‘’/``

styles/styleUrls: [‘’]

})

selector:’[app-servers]’ (as a tag property)/’app-servers’ (as a tag)/’. app-servers’(as a css class)

**Data Binding:**



1. String Interpolation: {{variable\_from\_ts\_file/ method calls ()/ or a String value}}

Here final value should be able to converted as string

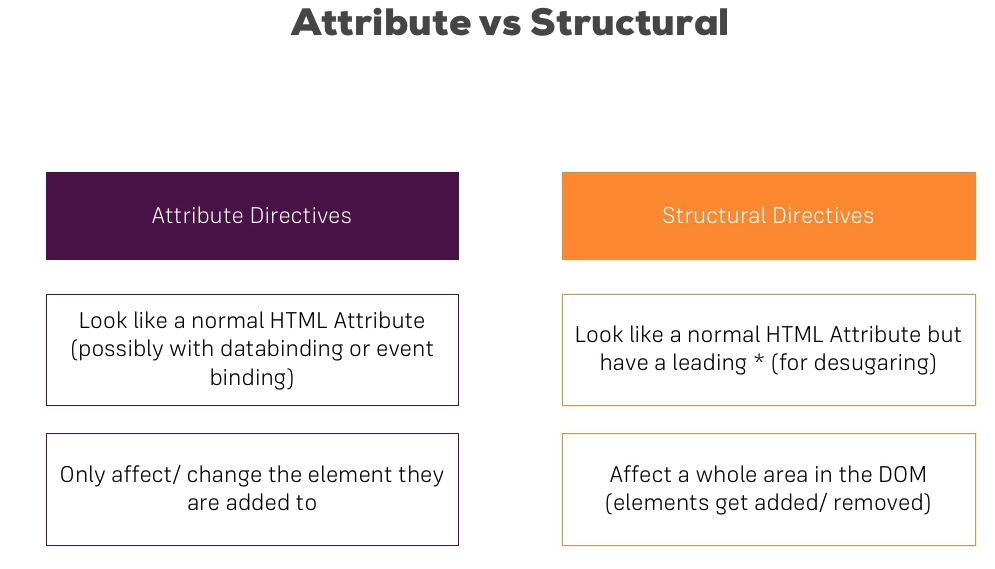
1. Property Binding:
2. Event Binding:

Ex: (click)=” onClick ()” or (input)=” onClick($event)” so we can get value in .ts file with event.target.value

1. Two wat Binding: [(**ngModel**)] (directive)–need to add below import and add to imports in AppModule

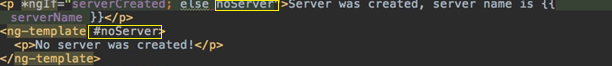
**import { FormsModule } from '@angular/forms';**

**Directives:**

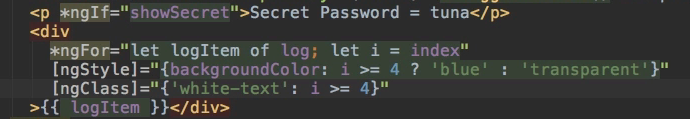
****

Structural Directives: changes the structure of the DOM

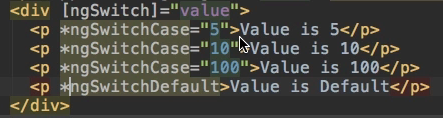
Ex: \*ngIf



Ex: \*ngFor



ngSwitch



Attribute Directives: we can only change the property of the DOM element like background color

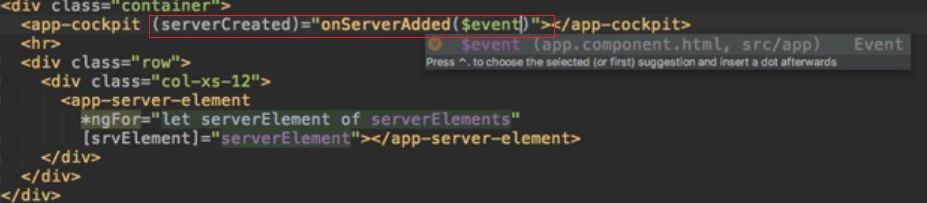
ngStyle, ngClass,

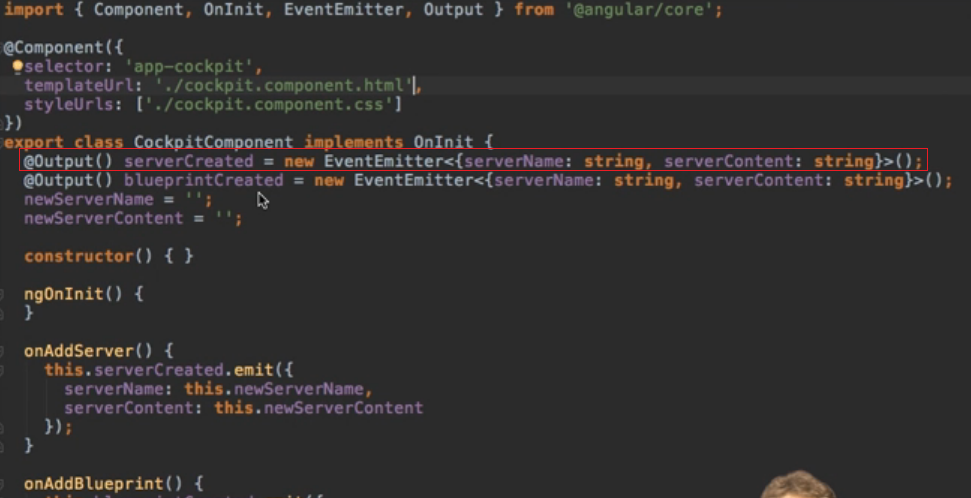


**@Input()-(Decorator**): to bind parent component values to the child component properties

-can alias by @Input(‘srvElement’)

**@Output()**-(Decorator): to make listenable from outside (to patent)





**EventEmitter<> ():** to create custom events

**View Encapsulation:** default nature of css file is, it will apply to the all the components in application irrespective of which component css file it is.

Using **encapsulation** property of @Component decorator we can change it behavior

Encapsulation: ViewEncapsulation.None/Native/Emulated

Emulated: id default- it includes extra (unique property ex(\_ngcontent-wwg-1) per component tags) property to the tags of component

None: will not add extra unique property to the component tags

Native: acts same emulated but it depends on browser. Some browsers may not support this.

**Local References:** #name

We can pass elements with local reference to the ts file and can use the value (here from while passing html to ts file it passes as Element itself)

**@ViewChild:**

@ViewChild ('serverContentInput', {static: true}) serverContentInput: ElementRef;

{static: true} as a second argument needs to be applied to ALL usages of @ViewChild () (and also @ContentChild ()) IF you plan on accessing the selected element inside of ngOnInit ().

If you DON'T access the selected element in ngOnInit (but anywhere else in your component), set static: false instead!

If you're using Angular 9, you only need to add {static: true} (if needed) but not {static: false}.

Using local references, we can access DOM element reference using this decorator

So while using to get value this.serverContentInput.nativeElement.value

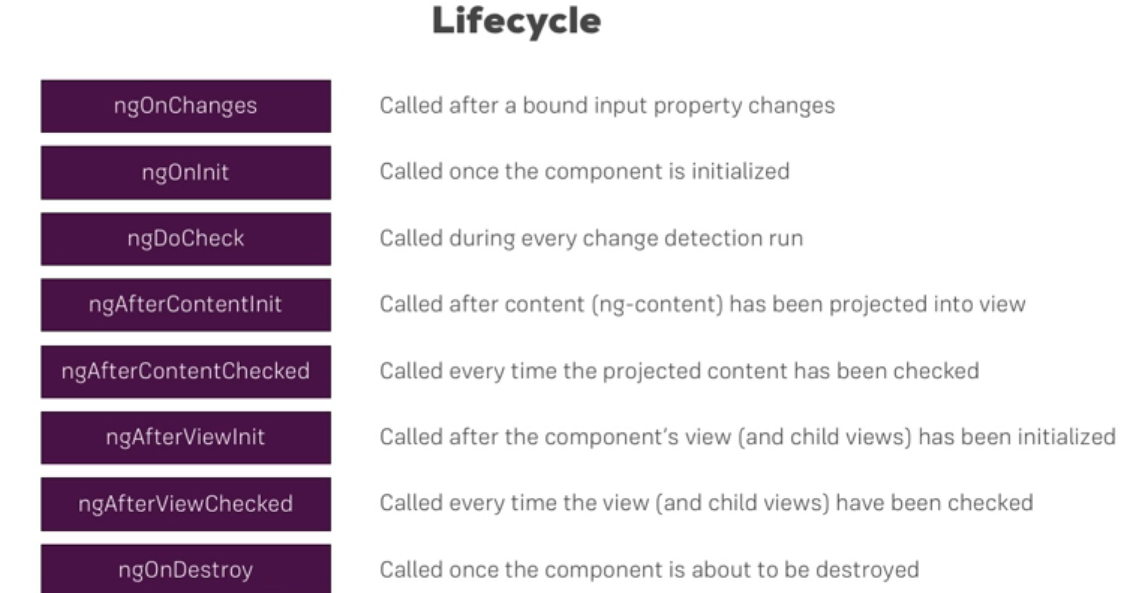
**ng-content:** directive

code between opening and closing tags of our own component will be lost, it is default behavior of DOM.

So we can add code in between opening and closing tags of our own component and add

**< ng-content > </ ng-content >** in custom component html code

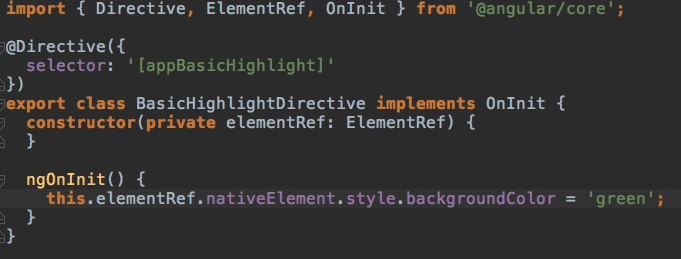
**Lifecycle hooks:**



**@ContentChild ():** to get access to ng-content of parent in child

This data is available after ngAfterContentInit

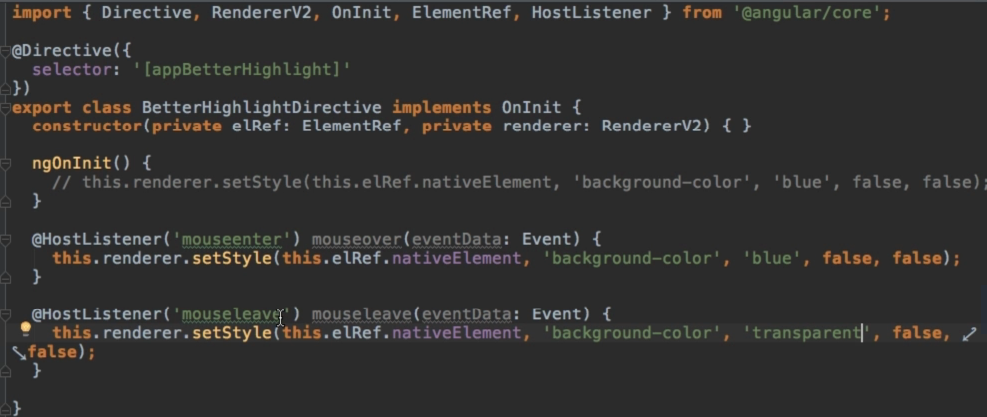
**Custom Attribute Directive:**

****

It is not a good practice to access DOM elements directly using ElementRef Bcoz direct access may give errors in some cases where we don’t have access to DOM elements like angular application with service workers, so use renderer to access and change the element.



**@HostListener:** when we need to respond to some events when a directive sits on the that element

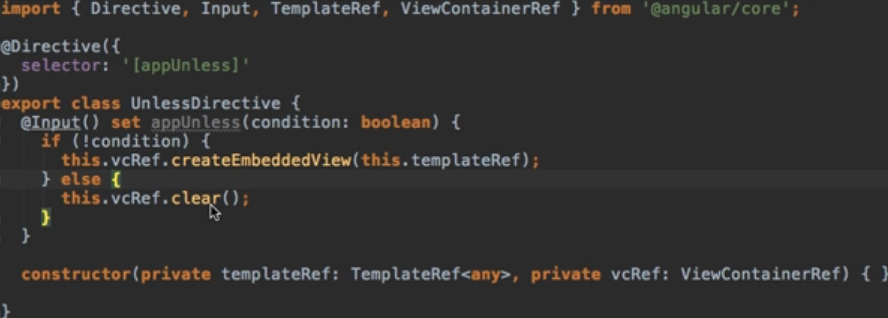


**@HostBinding:**

Using the renderer is not a bad approach, but we can do same thing with this decorator more easily.

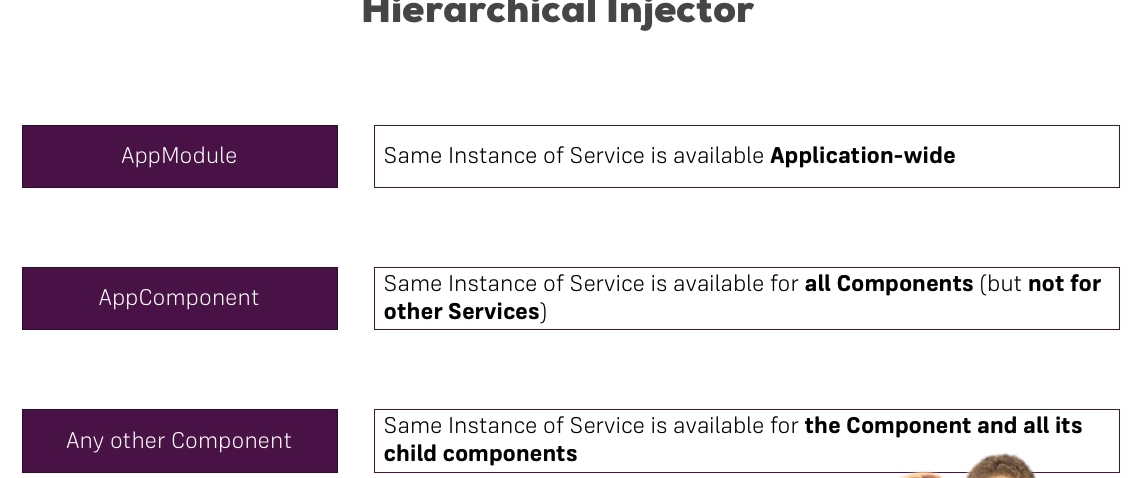


Custom Structural directives:



**Services:**

Need to add service class in Providers.

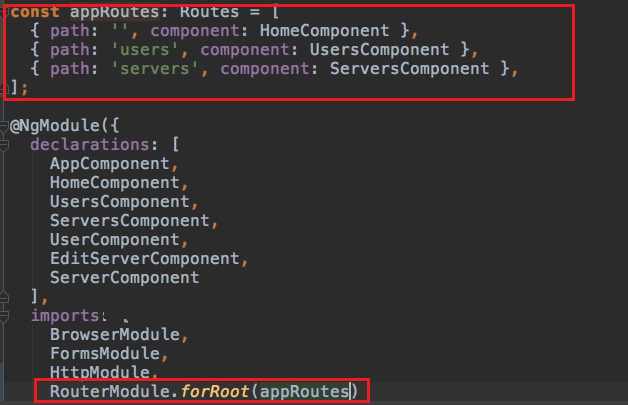
****

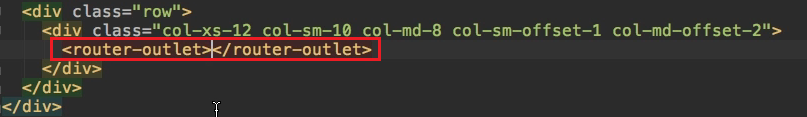
While injecting service to other service the receiving service need to add with some metadata using **@Injectable**

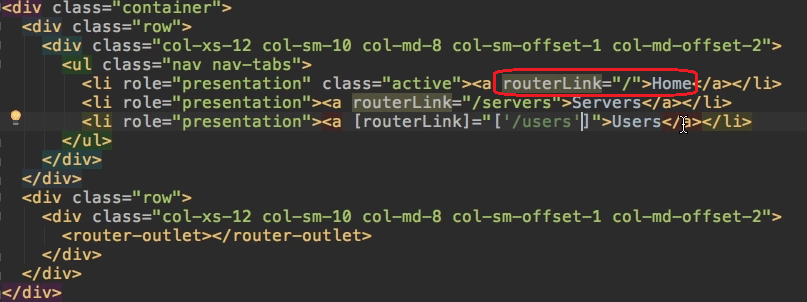
But for god practice add **@Injectable** to all services

In Angular 6+: we can use @Injectable ({providedIn: 'root'}), so no need to add in AppModule providers array

**Routing:**

****

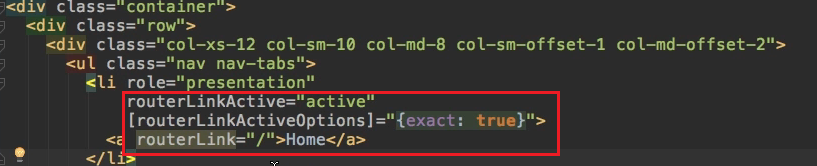
****

****

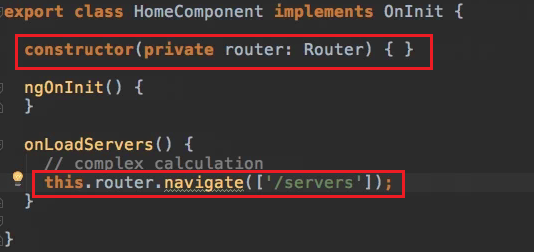
**-routerLink**

**-routerLinkActive**

**-rouetrLinkActiveOptions**

****

**Dynamically loading link through ts file:**

****

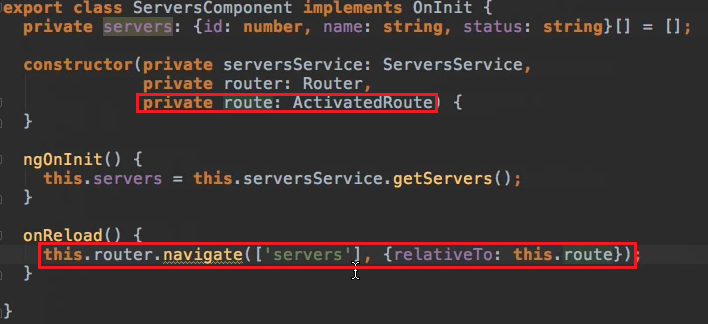
By mentioning “/users”(absolute path) on routerLink, it loads localhost:4200/users

But by mentioning “/users”(relative path) on routerLink, it loads users on current\_path/users

By mentioning absolute/relative path on router.navigate() method it loads localhost:4200/path mentioned in that method

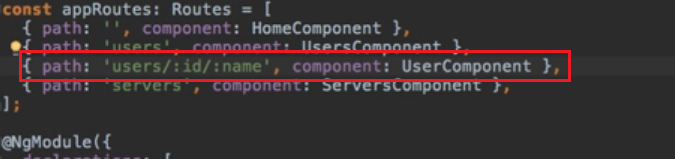
But we can mention using **relativeTo** to which route it is relative to in navigate () method

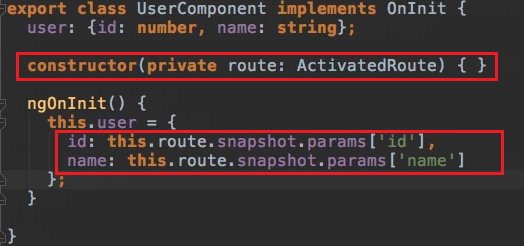
-**ActivatedRoute –** gives access to the currently active path/route

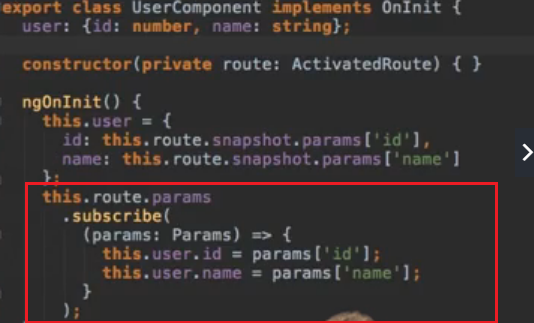


-Use **:variable\_name** for dynamic paths

-to access path variable inject ActivatedRoute





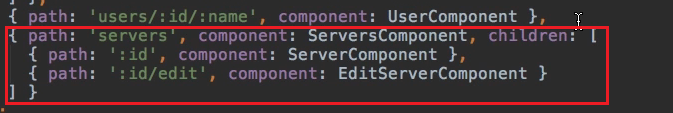


**Query Parameters:**

****

**-fragment:** adds extra text at the end of url proceeding #

**Child Routes:**

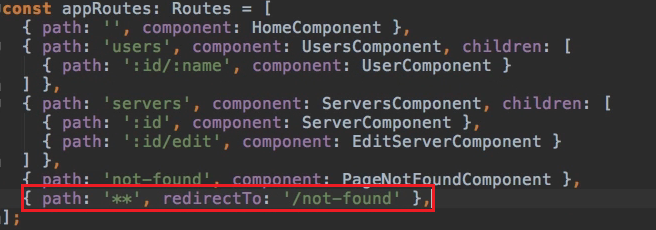


To preserve query parameters to the next component load on url we can use **quesryParameterHandling**

To this we can pass **preserve (**for preserving previous ones**)/merge (**for adding new parameters**)**



To access any other text with url, ‘\*\*’ path should be the last in array



By default, Angular matches paths by prefix. That means, that the following route will match both /recipes and just /

{ path: '', redirectTo: '/somewhere-else' }

Actually, Angular will give you an error here, because that's a common gotcha: This route will now ALWAYS redirect you! Why?

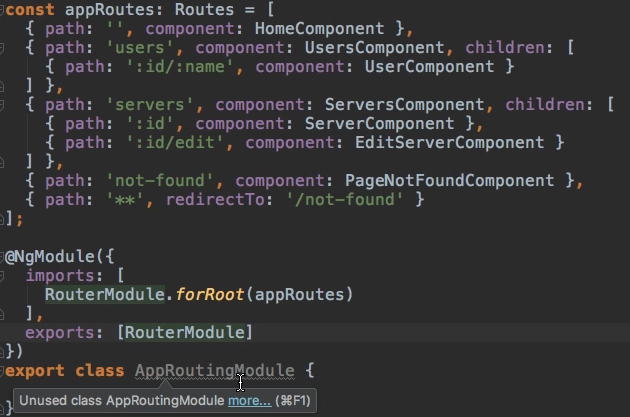
Since the default matching strategy is "prefix”, Angular checks if the path you entered in the URL does start with the path specified in the route. Of course every path starts with '' (Important: That's no whitespace, it's simply "nothing").

To fix this behavior, you need to change the matching strategy to "full”:

{path: '', redirectTo: '/somewhere-else', pathMatch: 'full’}

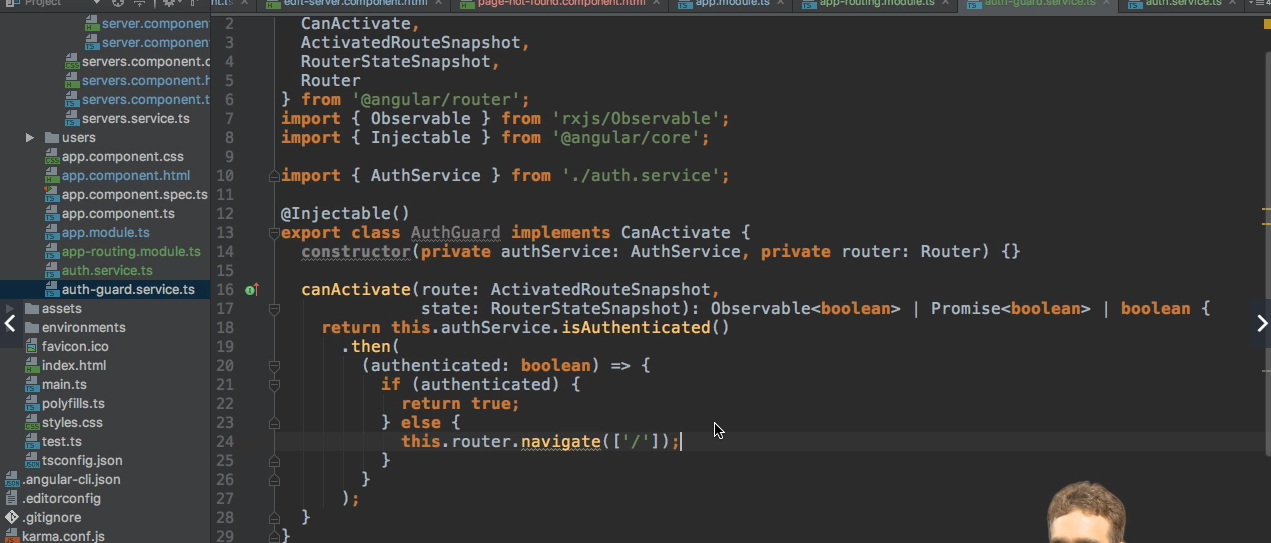
Now, you only get redirected, if the full path is '' (so only if you got NO other content in your path in this example).

**RouteModule:**



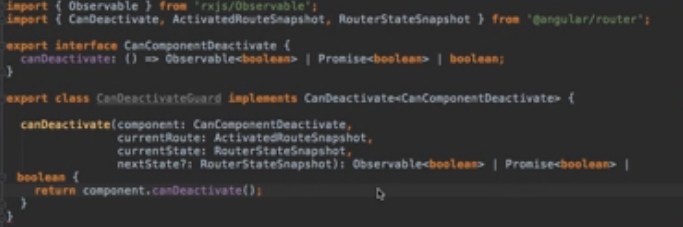
**Router Guards:**

1. **CanActivate/CanActivateChild:** To prevent accessing routes from outside we can use this

****

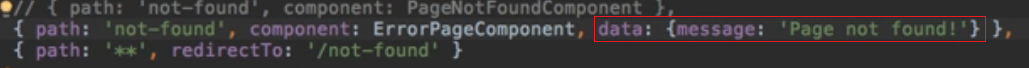
****

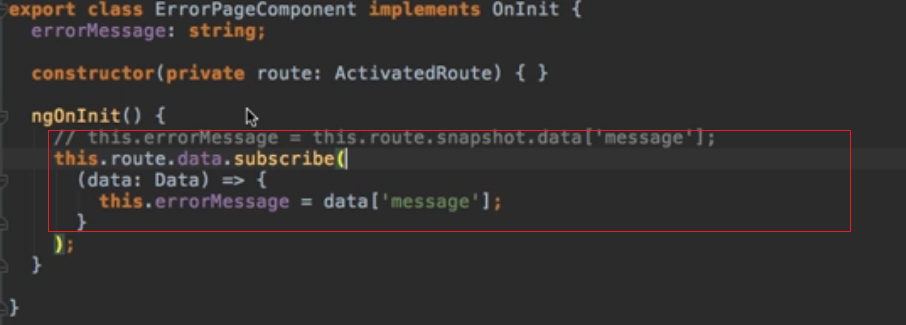
1. **CanDeactivate:** to control leaving the route



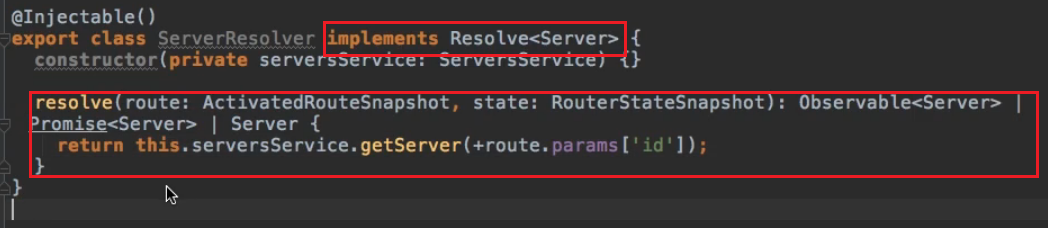
* After the above code, need to implement “**CanComponentDeactivate**” interface in component and implement **canDeactivate**() method with our own logic.

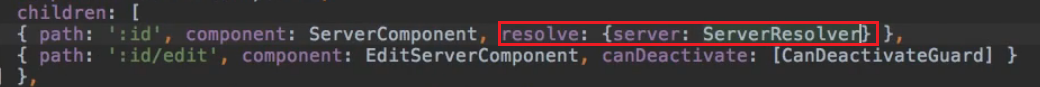
**Passing static data to routes:**

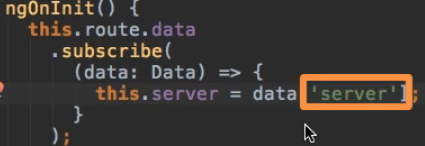
****

****

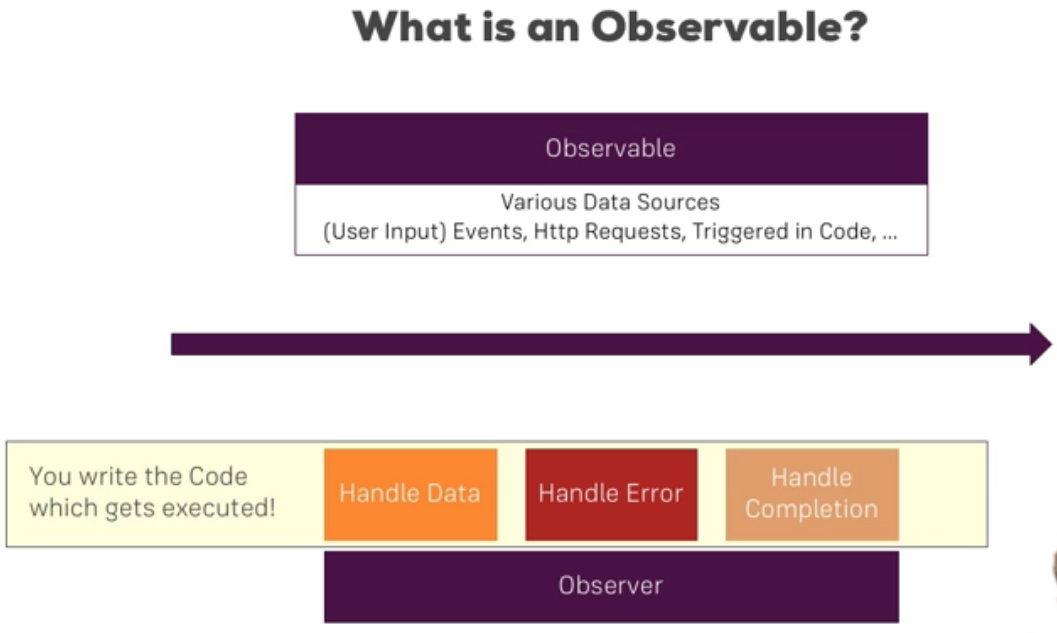
1. **Dynamic data with the Resolve guard:** Runs before the guard is rendered, unlike canActivate/CanDeactivate it will not decide whether the route should render or not.Which will always render the component in the end.







**Observables:** For asynchronous data access



**AngularJS**

**Controller and Module:** 