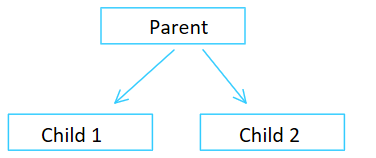
# \*ngFor – ‘\*’ indicates that it is structural directive => it is used to change the structure of DOM

* [(ngModel)] = "username" whatever comes in “ <this is the TypeScript expression> ”

All the custom components & directives have to be mentioned in *declarations[]* of app.module.ts

# @Input

Components can be nested as below. Generally properties of any component are specific to itself. It will not be accessible from any other components perspective.



DOM structural representation of above figure

*<app-root>*

*<app-child1></app-child1>*

*<app-child2></app-child2>*

*</app-root>*

For <app-root> to be able to send some data to <app-child1> or <app-child2> which are specific to *child1/2*, the properties have to be decorated with *@Input()* in the app-<child1-or-child2>.component.ts.

# @Output

To emit some custom events from child components to the parent components the specific properties are marked as *@Output()*

note

@Output() can be used for triggering events if the components are laid out very close to each other. If the level of nesting is too deep, @output() fails to serve the purpose rather we will have to write a lot of code just to listen the event triggered in another component.

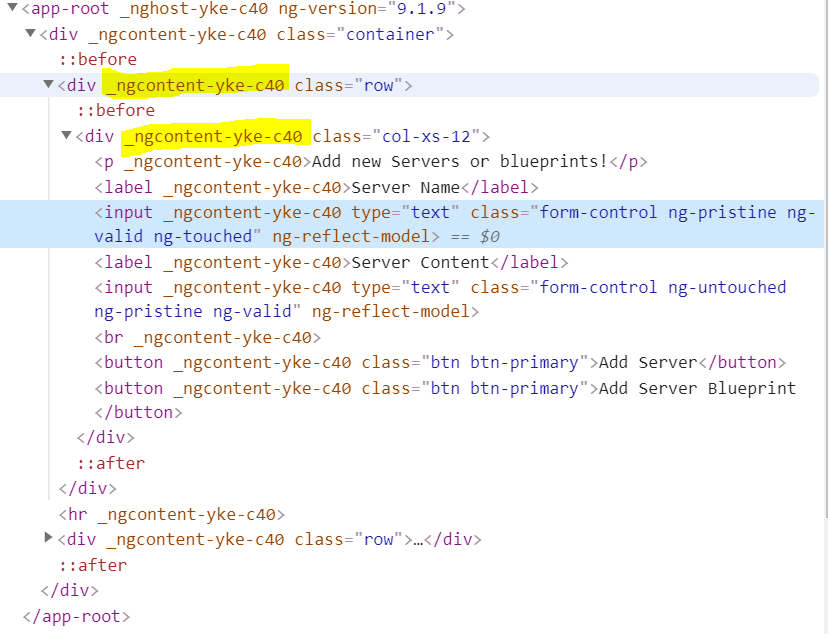
For event emission in deep nesting of components , services are of much help.

# @ViewChild()

After the DOM rendering is done, we can see various HTML element like <h1-h6>,<p>,<div> has different attributes. For example : component 1 has <p> and <h1>

component 2 has <p> and <h2> and comp2 is nested inside comp1

once after the DOM rendering is done, we can observe the DOM as below



All the HTML elements(p,h1-h6,input…..) inside one component will be having a specific attribute like the one highlighted in above snap i.e., \_ngcontent-yke-c40.

All the HTML elements in other component can look some thing like \_ngcontent-yke-ejo1. These attributes are completely manipulated by angular at run time. So we have no knowledge on how to select HTML elements based on attributes.

With @ViewChild, we can inject *any* component or directive (or HTML element) present on the template of a given component onto the component itself.

@ViewChild can only see elements inside the template of the component itself. To summarize: the @ViewChild decorator is a template querying mechanism that is *local* to the component

# LifeCycle

# 

We can use only one structure directive on any element => we wont be allowed to use ***\*ngIf*** and ***\*ngFor*** on the same element

# @NgClass

Adds a class attribute with dynamic values to the html elements or custom component elements.

<a [NgClass] = “ ’text-column’ ” 🡪 this removes all classes already available on the host element i.e, a and then add a new class ’text-column’

Other syntax of NgClass is <a [class.<name-of-clas> ] 🡪 this appends the ‘*name-of-class’* to the list of classes already existing.

Below is the object literal style of assigning classes . this is applicable for ngStyle as well. Using below way, we can assign multiple values at once

<h4>NgClass</h4>

<ul \*ngFor="let person of people">

<li [ngClass]="{

'text-success':person.country === 'UK',

'text-primary':person.country === 'USA',

'text-danger':person.country === 'HK'

}">{{ person.name }} ({{ person.country }})

</li>

</ul>

# @HostListener

* @HostListener is Angular’s decorator method that’s used for listening to DOM events on the host element of both component and attribute directives.
* @HostListener sets the listeners once the directive is initialized and removes them automatically once the directive gets destroyed.
* @HostListener catches and listens to events from children or nested elements when they bubble up to the host element. Moreover, you can listen to events on global elements, but you should do so sparingly for performance reasons.

# Services

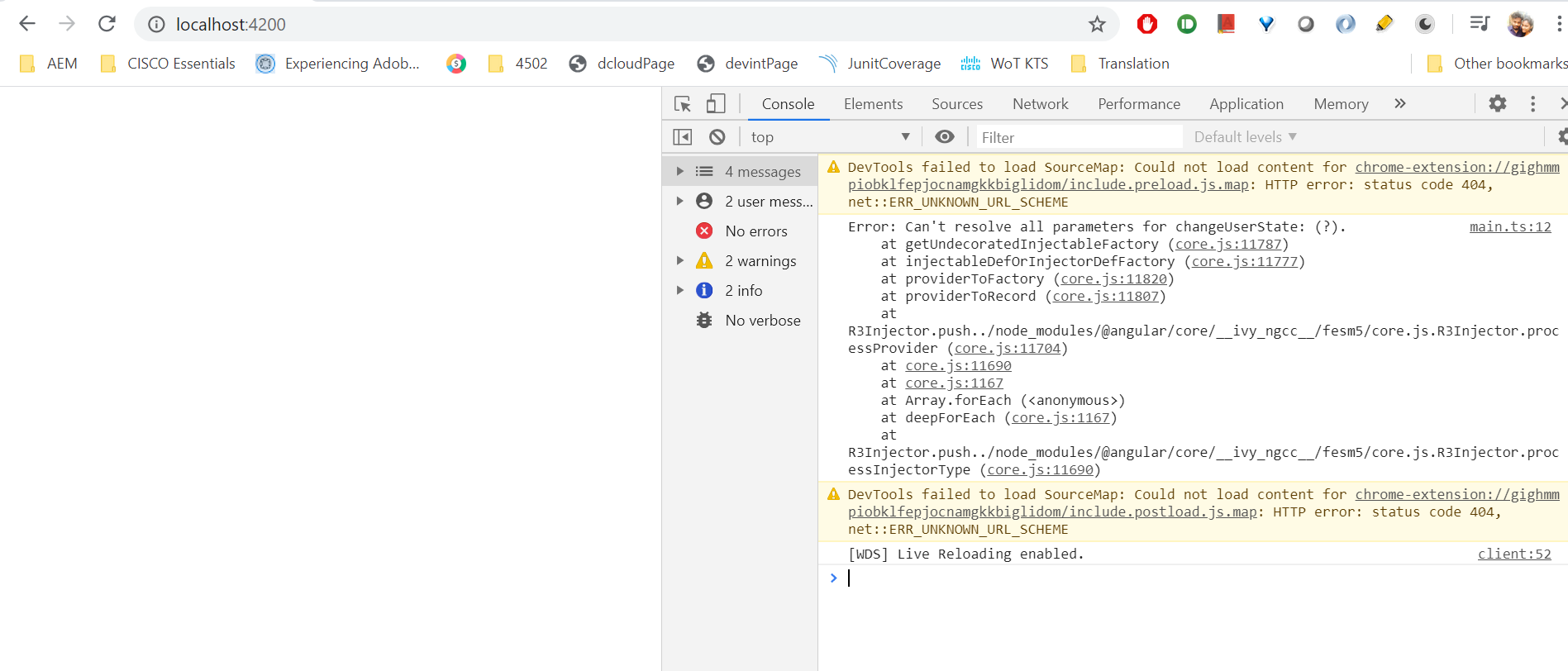
Services are just like helper classes which are to be injected into component where we need to use them.

To mark a class as service decorator ***@Injectable()*** has to be used on the TS class

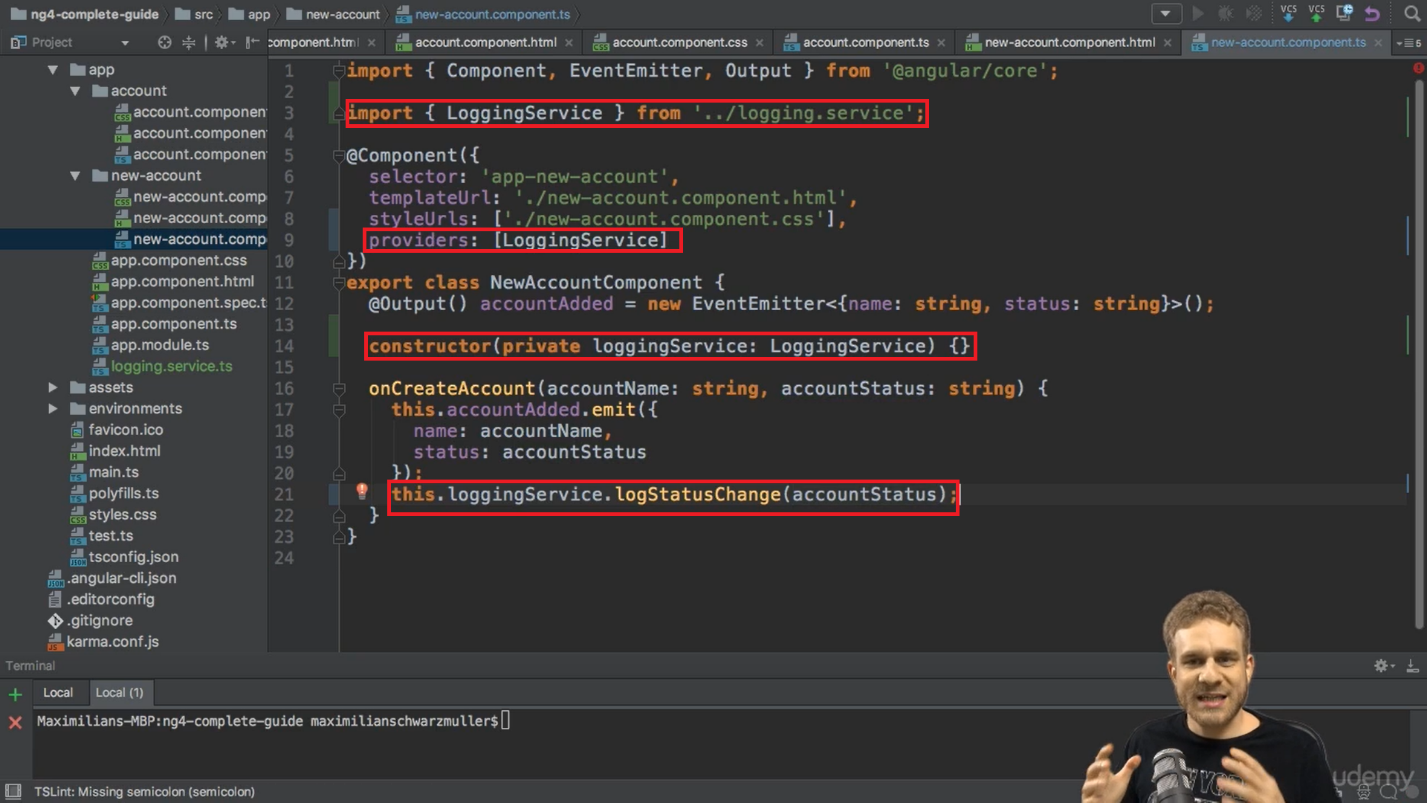
Note

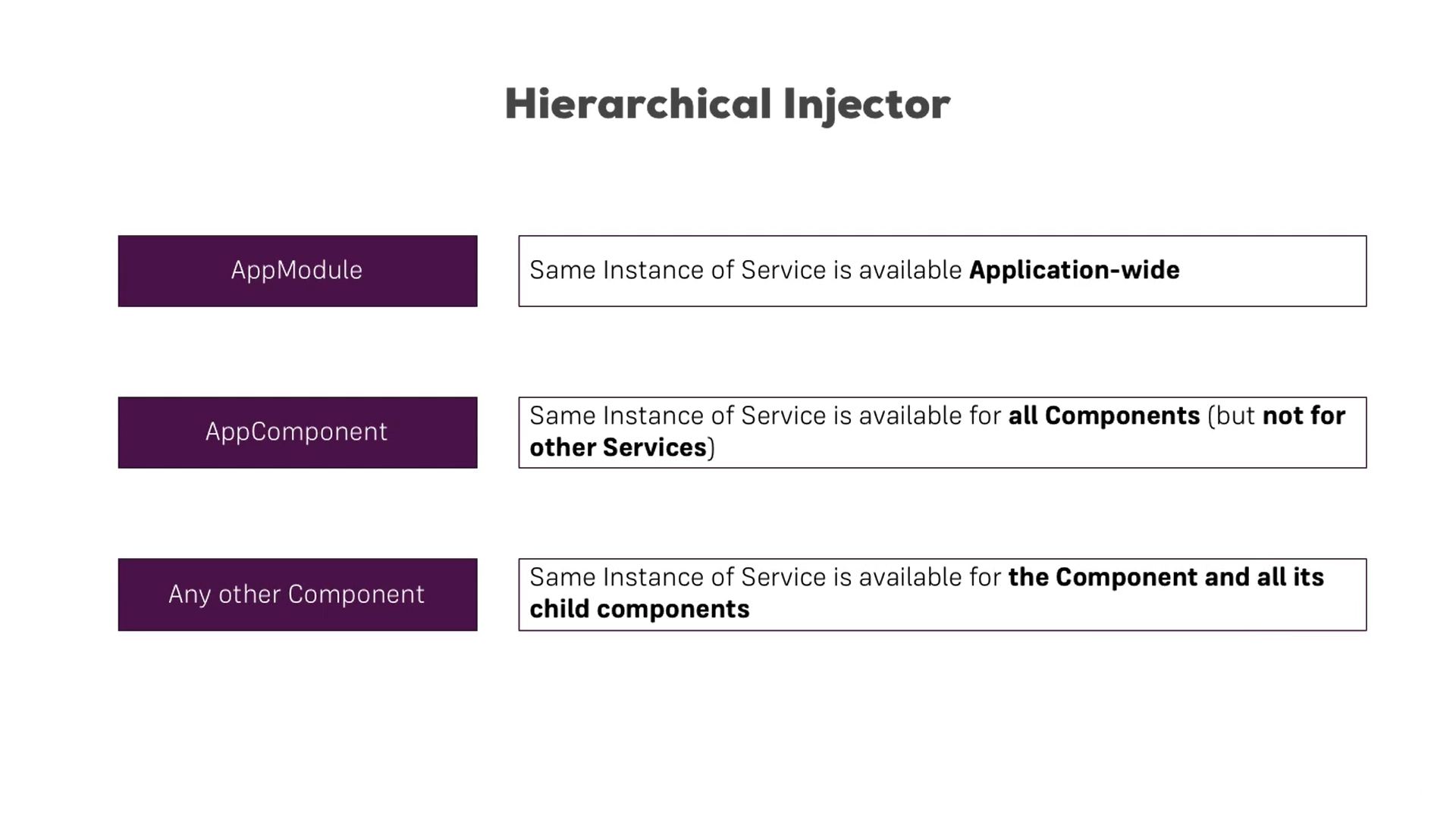
To inject a service A into service B, service A(service in which we are injecting) has to be marked with above decorator.

Below is the error we get if we DO NOT place **@Injectable()** to service A.



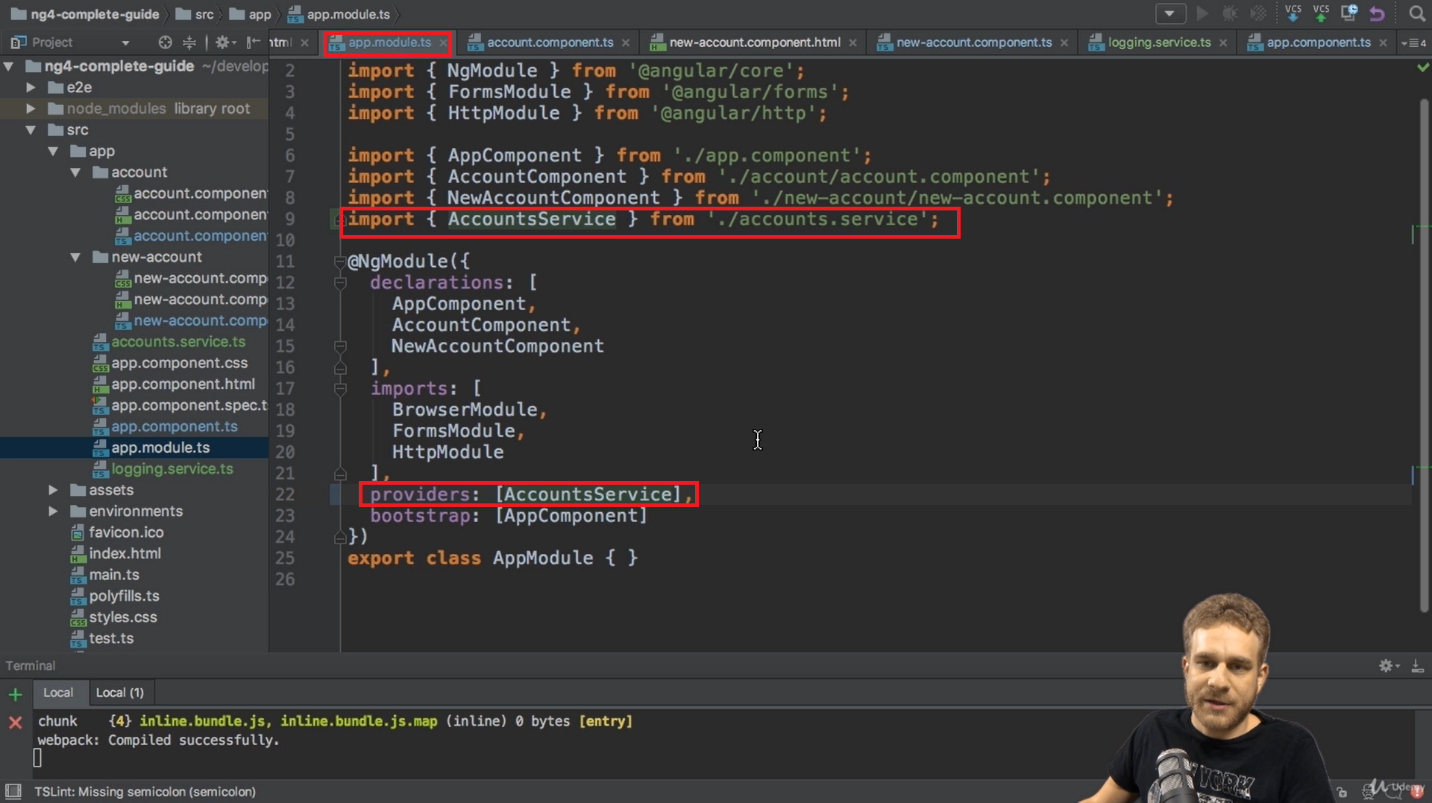
Rather than instantiating them using ***new*** keyword, below is the way to reference them and use in our component. Import🡪 providers 🡪 constructor 🡪 using reference





For Module level service provision

* If the services are to be declared application wide, they can be specified in the **providers []** of app.module.ts



# Routing

The router enables navigation by interpreting a browser URL as an instruction to change the view.

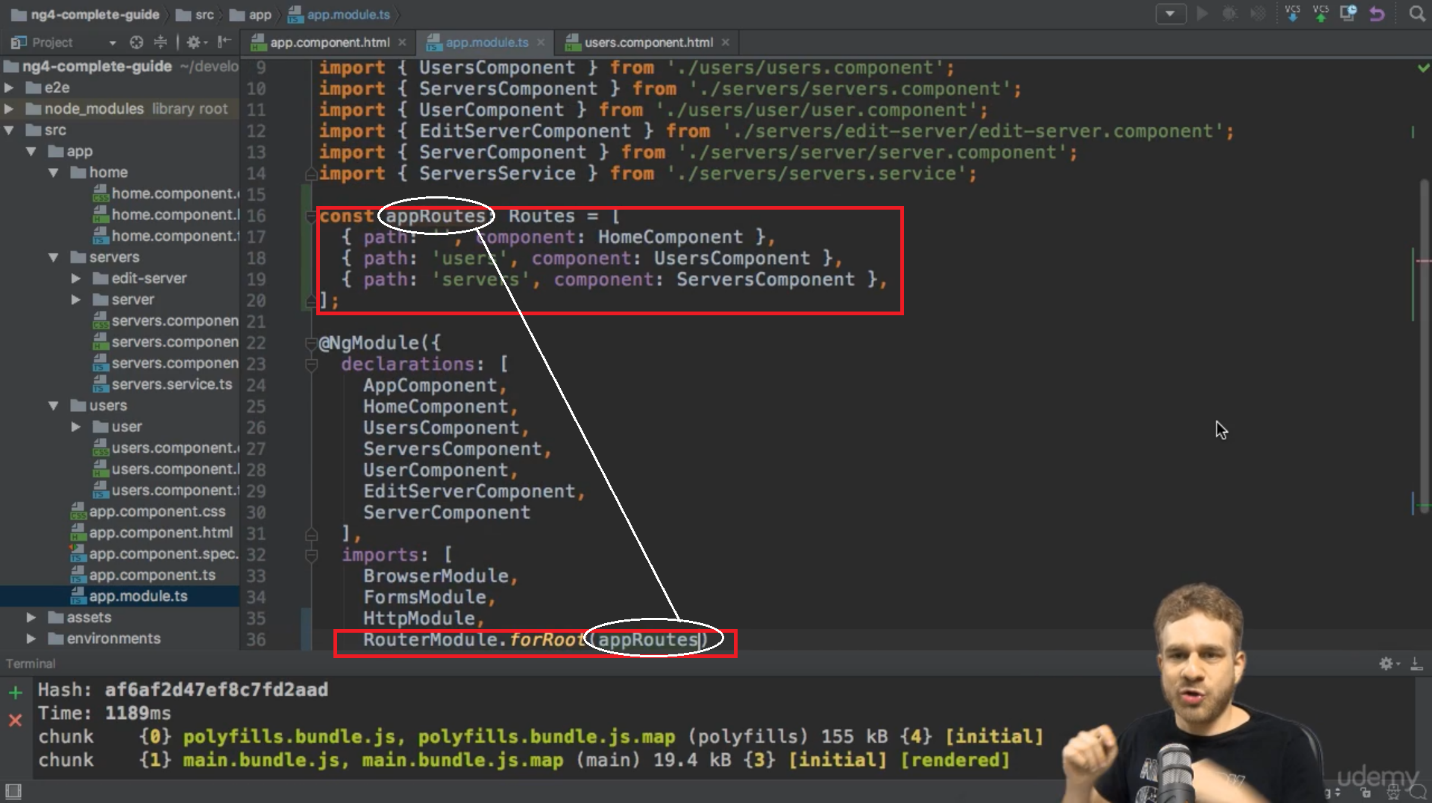
ng new routing-app –routing

--routing adds the AppRoutingModule to the imports[] in app.module.ts

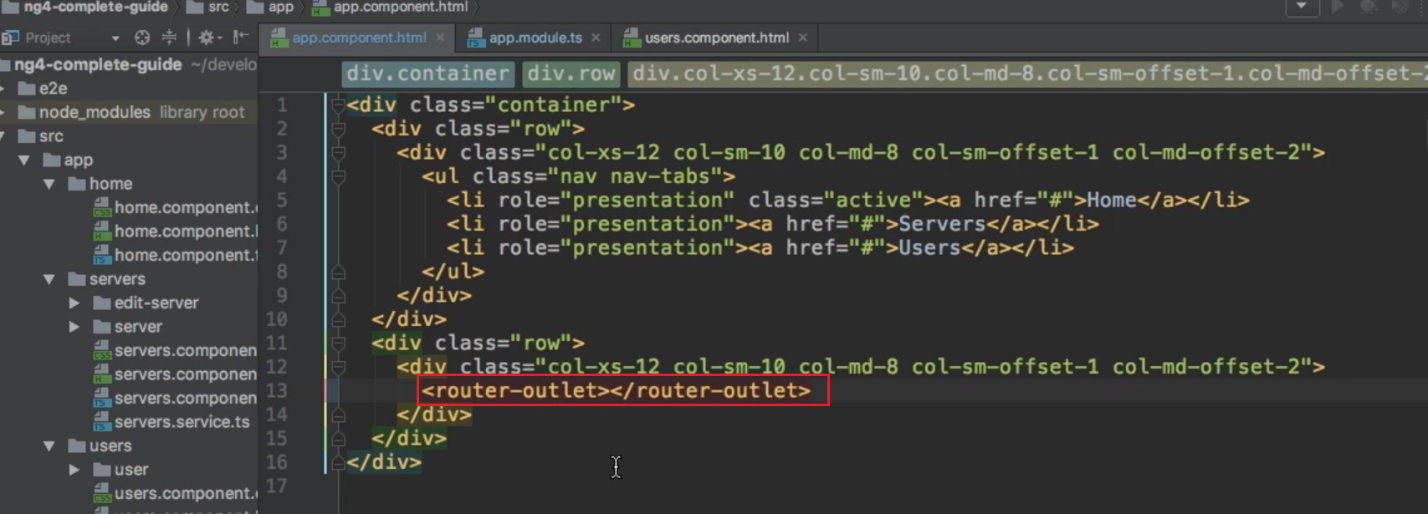
Routes are added as below. Each route is javascript object with keys path and component



First the routes are to be configured and then registered in the imports section of @NgModule as below



After the routes are registered, a place-holder in the HTML template to render components depending on user interaction is to be specified using <*router-outlet*></ *router-outlet* > as shown below



Above directive (<*router-outlet*></ *router-outlet* >) marks the place in DOM where we want to load the component of currently selected route.

Instead of changing the url manually, we can use *href* attribute of <a> tag to navigate from one component to another. But this in turn reloads the page owing to default behavior of href

Using ***routerLink*** attribute on <a> prevent the default behavior and specifies the route to navigate upon clicking <a>

routerLink can be configured using below

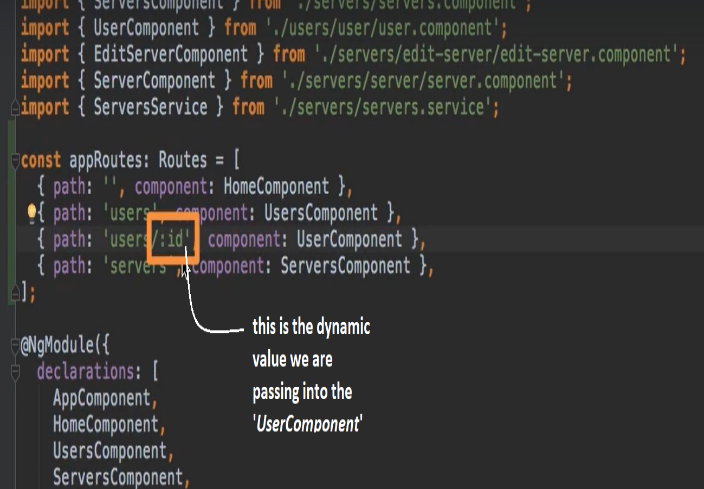
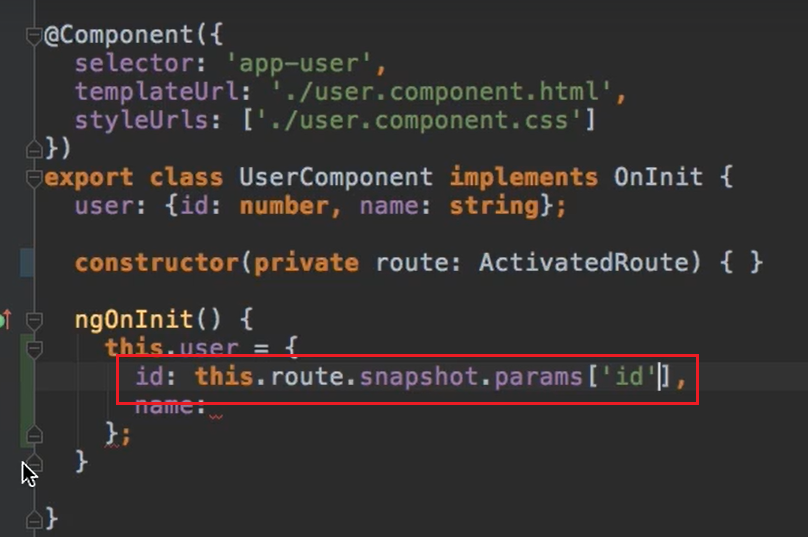
The routerLink name can be prepended with /,  ./, or ../:

* If the first segment begins with /, the router will look up the route from the root (the string after localhost:4200/<root of the app>) of the app.
* If the first segment begins with ./ (relative), or doesn't begin with a slash, the router will instead look in the children of the current activated route.
* And if the first segment begins with ../, the router will go up one level.

RouterLink is aware of active route whereas Router.navigate()[this uses configObject as a parameter for various configurations on activatedRoute] isnt

Route paths can be dynamic which injects some dynamic value into the components

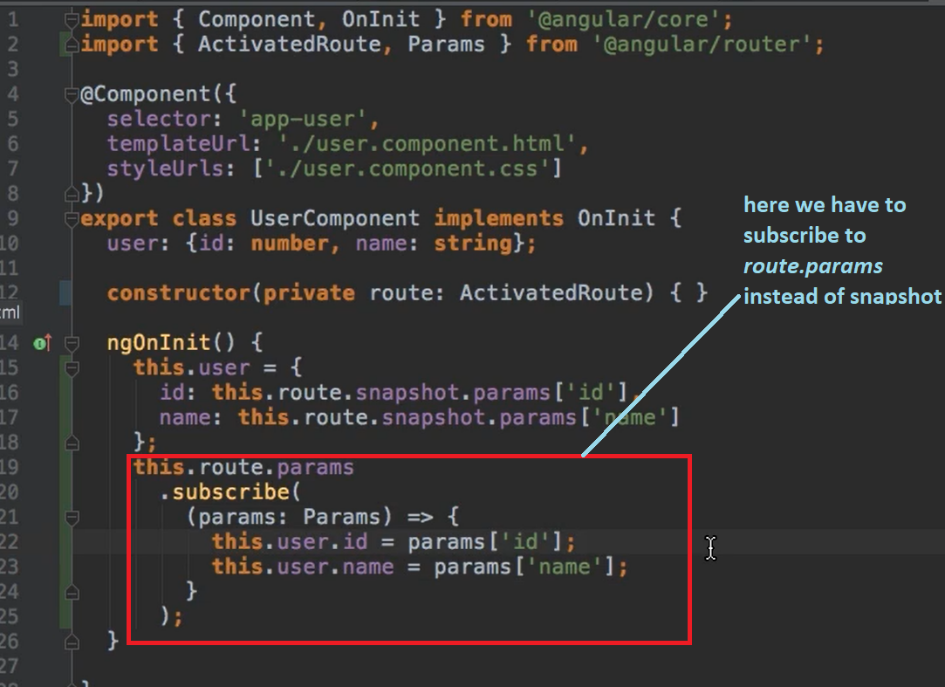
* :id indicates that this is gonna be a dynamic parameter which will be available to mapped component

A component is rendered with routes in place. If we change it from address bar, it works fine because entire component is re-rendered 🡪 ngOnInit() executes at the time of initializing the component

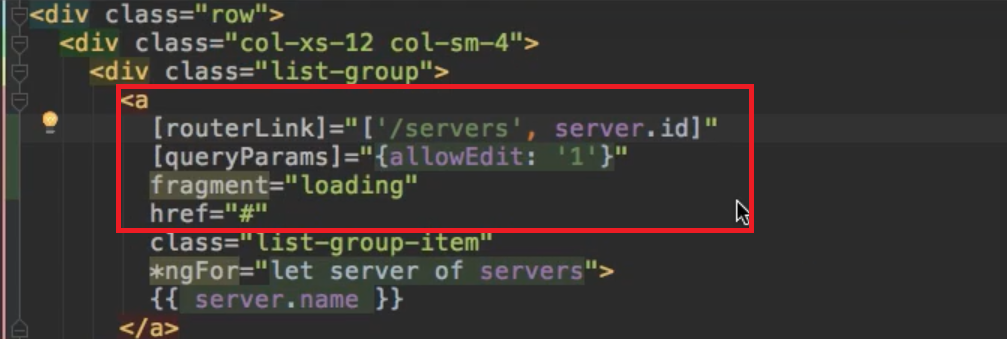
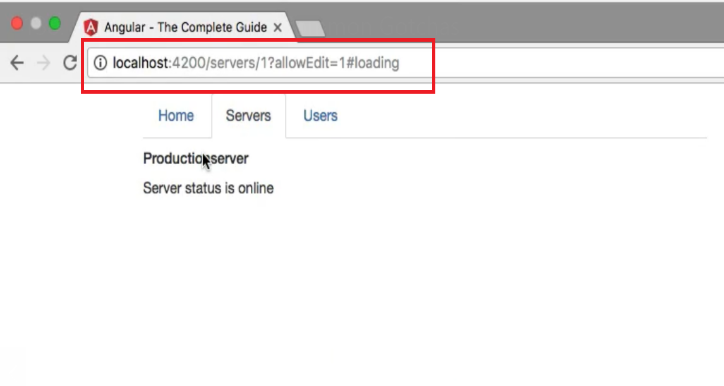
But when the route is manipulated from within the component (using a routerLink on <a>), the component won’t re-render the component because it costs us the performance. Only *OnInit(),* the dynamic values can be passed and the component will be re-rendered.

To handle such scenarios, we need to subscribe as below



The changes made to route from with in the component will be taken care by the code snipped above.

* Adding queryParams and fragments. The below gives the url as *localhost:4200/servers/5?allowEdit=1#loading*

Redirects in Routing

