Angular 16 : Online Class

Routing

By: Sahosoft Solutions

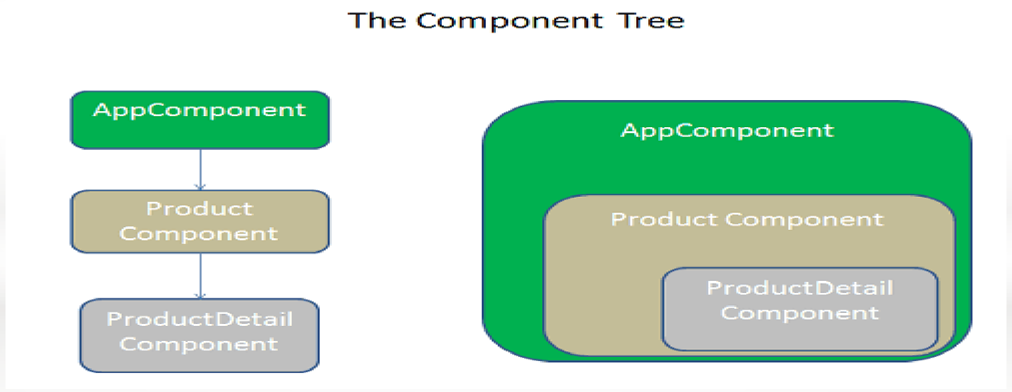
Presented by : Chandan Kumar

Child Routes / Nested Routes

The Angular 2 and above applications are based on the idea of Components. The Components follows a Tree structure, where we have a root component at the top. We can then add child components forming loosely coupled components resembling a Tree.

The Routes in Angular also follows the component tree structure and allows us to define the nested or child routes.

Example
Consider the following Component Tree



Child Routes / Nested Routes

Example

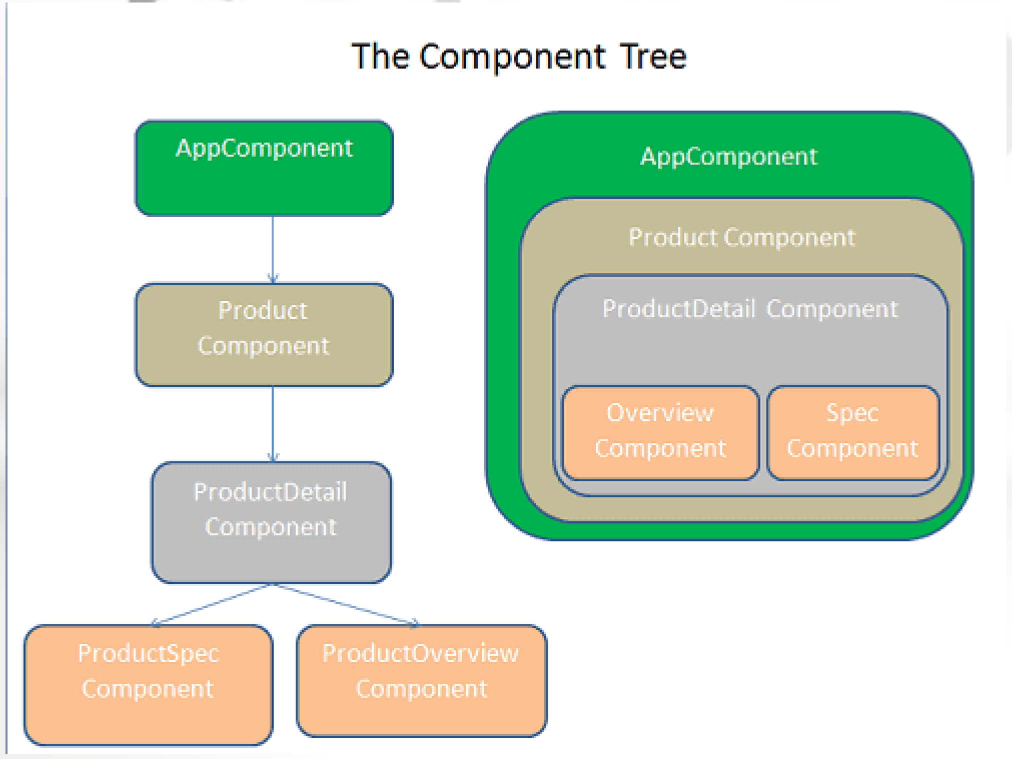
```
const routes: Routes = [
{ path: '', redirectTo: 'dashboard', pathMatch: 'full' },
{ path: 'dashboard', component: DashboardComponent },
{ path: 'about', component: AboutComponent },
{ path: 'contact', component: ContactComponent },

{
path: 'student',
children: [
{ path: '', component: StudentComponent , pathMatch: 'full' },
{ path: 'studentdetails', component: StudentdetailsComponent },
{ path: 'studentregistration', component: StudentregistrationComponent },
]
},

{ path: '**', component: PagenotfoundComponent },
];
```

Nested Routes (Nesting Children's under a child)

Nesting Children's under a child

We can add child routes to a child route.

For Example, What if we want to show Product Overview & Specification under the Product Details Page. Our Component Tree is as shown below



Parameterize Route

A route can receive the parameters either using a **snapshot property** of the ActivatedRoute or by **subscribing to its property param**.

The Angular Router provides two different methods to get route parameters:

1. **Using the route snapshot,**
2. **Using Router Observables**

Example:

- route.snapshot.param
- **route.snapshot.paramMap**

- route.param.subscribe
- **route.paramMap.subscribe**

Parameterize Route

The Snapshot Way:

The Snapshot Way: The router provides us with a snapshot of the current route.
The Angular Router provides an **ActivatedRoute** that we can inject into our classes. This will be the main tool we'll use to grab route parameters. We inject the ActivatedRoute into this component via the constructor.

The snapshot approach fails if we are navigating from one component back to the same component. The reason is that Angular needs to figure out whether it needs to initialize the new component or if it can reuse the same component. If we are navigating back to the same component. Angular reuses the same component. This is why our view is not getting updated.

The Observable/Stream Way:

The Observable/Stream Way: By subscribing to its property param. Now there is a property paramMap that allows you to get a particular parameter by using the method get() .

We are getting the updated value even on navigation on the same component. You can see the URL value is getting changed and also making the view updated. Clicking on any record will give you the updated view.

Parameterize Route

1. **When to use route.snapshot.paramMap ?**

If you intend not to update your URL parameter within the same component you are accessing it, then you can use the snapshot.
As the name suggests, the parameter would only be accessed once, when the component loads. Hence, it won't be updated, even if you change its value from within the same component.

```
ngOnInit() {
  this.obj = this.route.snapshot.paramMap.get("anyValue")
}
```

2. **When to use route.paramMap.subscribe ?**

If you intend to update the URL parameter within the same component, then you have to use a subscription.
The good news is that it works just like any other subscription in your Angular app.
This technique is only useful if you plan to change the URL parameters within the current route.

```
ngOnInit() {
  this.route.paramMap.subscribe(params => {
    this.obj = params.get("anyValue")
  })
}
```

Angular 16 : Online Class

Route Guards

By: Sahosoft Solutions

Presented by : Chandan Kumar

Route Guards

Being web application developers, we are aware how a Server checks the permissions for the user on the navigation and returns the result Telling whether a user can access the resource or not.

In the same way, we need to achieve this on client-side. Route Guards are the solution for that. We can use the route guards to control whether the user can navigate the route or not.

Angular’s route guards are interfaces which can tell the router whether or not it should allow navigation to a requested route. They make this decision by looking for a true or false return value from a class which implements the given guard interface.

In the Angular application in which authentication and authorization is required to navigate a route, the role of Angular route guard comes into the picture.

There are five different types of guards and each of them is called in a particular sequence.

The router’s behavior is modified differently depending on which guard is used.

Route Guards

Guard Types

There are 5 types of Route Guards we use in Angular.

CanActivate
This helps to decide whether the route can be activated or not.

CanActivateChild
This helps to decide whether child routes can be activated or not.

CanDeactivate
This helps to decide whether the route can be deactivated or not. Generally, this is used to warn a user if they are navigating from the component.

Resolve
This performs route data retrieval before any activation of the route.

CanLoad
Checks to see if the user can navigate to the module which is lazily loaded.

Route Guards

Guard Types

In Short:

✓ CanActivate guard (e.g. it checks route access).
✓ CanActivateChild guard (checks child route access).
✓ CanDeactivate guard (prompt for unsaved changes).
✓ Resolve guard (pre-fetching route data).
✓ CanLoad guard (check before loading feature module).

Before starting to implement the Guards, let's have some basic information about route guards.

❑ Guards are always implemented as service that needs to be provided so we always make them *@Injectable* while creation.
❑ Guards always return true or false telling you if access is allowed or not
❑ Guards can also return the Observables or Promises which can resolve into the Boolean value finally and will be returned.
❑ Every Route guard must be imported to the application or Root Module
❑ As the Route guards are services they must be registered in the Providers section of the Application module.

Route Guards

To sum up we can add the Route Guards in Following ways

- ✓ To Restrict Access or to warn before the navigation we can use the Route Guards depending on our need.
- ✓ We can use the Interface canActivate,canDeactivate,canActivateChild for that purpose.
- ✓ Route Guards are injectable classes which can accept service dependency which can further be called and determine if we want to load the Route or not.

We can use the Angular CLI command
ng g g [guard name]

```
? Which interfaces would you like to implement?
>( ) CanActivate
 ( ) CanActivateChild
 ( ) CanLoad
```

Route Guards Parameters
ActivatedRouteSnapshot
This is the future route which will be activated when the guard condition is passed.
RouterStateSnapshot
This the future router state which will be activated if the guard is passed.

Route Guards

CanActivate Route Guard

Generally, in our application, we have a need where we want to check if the user is logged in or not. In that case, we can decide if we want to activate that route for that user or not.

If the CanActivate guard returns false then the user is not able to navigate to route further where want to navigate. Angular provides CanActivate Route for that.

CanActivate decides whether we can navigate to a route or not. It is used to redirect to login page to require authentication.

CanActivate is interface and have method **canActivate()** that is used for authentication.

CanActivate is an Angular interface. It is used to force user to login into application before navigating to the route.

Route Guards

CanActivate Route Guard

CanActivate interface has a method named as canActivate() which has following arguments.

ActivatedRouteSnapshot:
This is the future route which will be activated when the guard condition is passed.
RouterStateSnapshot:
This the future router state which will be activated if the guard is passed.

canActivate() returns boolean value, UrlTree or Observable or Promise of Boolean or UrlTree value.

Before Angular 7.1, route guards can only return a **boolean**, **Promise<boolean>** or **Observable<boolean>** (asynchronous boolean objects) to tell the router if the route can be activated or not.

But now, you can also return an **UrlTree** variable which provides the new router state (route) that should be activated.
According to the Angular docs an UrlTree is a data structure that represents a parsed URL.
Note: You can create an UrlTree by calling the parseUrl() or createUrlTree() method of the Router object.

Route Guards

CanActivateChild Route Guard

This guard checks the access on child routes.

CanActivateChild decides whether we can navigate to child routes or not. It is used to decide link access on the basis of authorization. It is possible that those links accessible to ADMIN role, will be not be allowed to USER role.

CanActivateChild is interface and have method **canActivateChild()** that is used for authorization.

CanActivateChild is an Angular interface to guard child routes. Suppose a user has been authenticated but not authorized to visit the child routes, so child routes can be guarded using CanActivateChild.

Route Guards

CanDeactivate Route Guard

Sometimes, we have a condition where we want the user to confirm if he really wants to navigate from the component. We can use CanDeactivate route guard for this.

This guard is used when a user makes some changes in a form and by mistake clicks on a menu to navigate to some other route. In this case, it can be implemented to prompt a user that there are some unsaved changes, do you want to navigate away from this page or not.