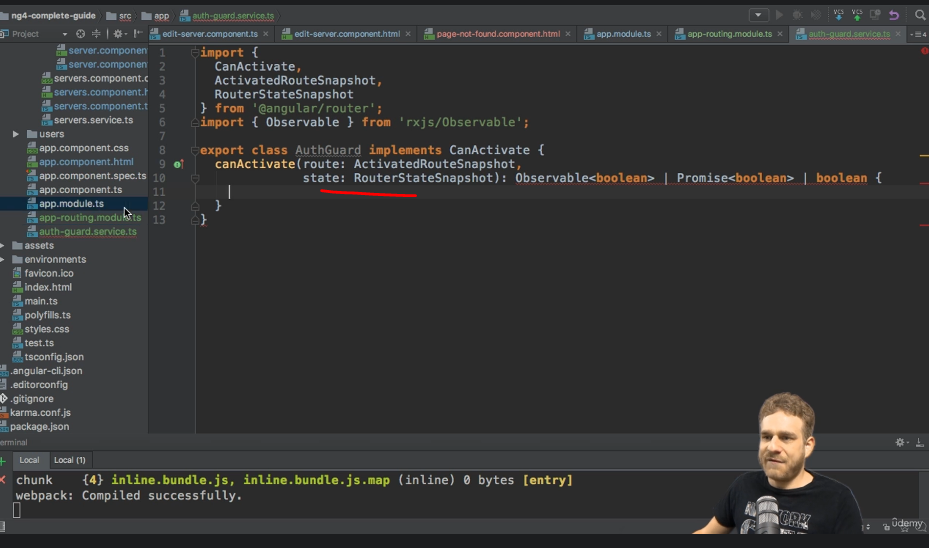
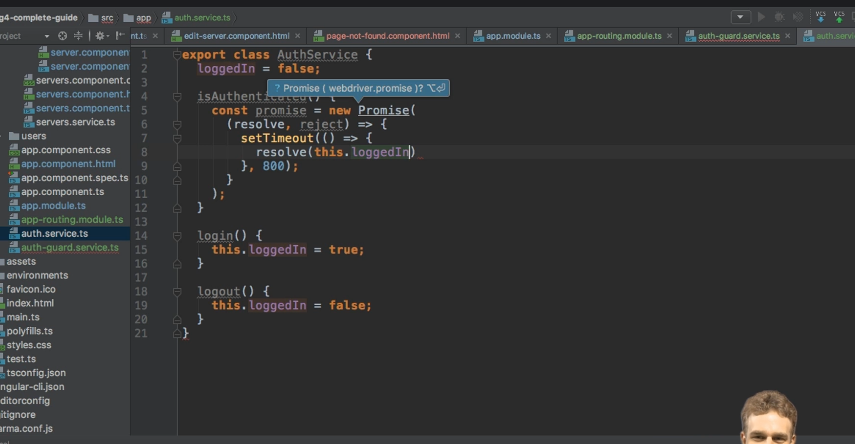
**146. Protecting Routes with canActivate:**

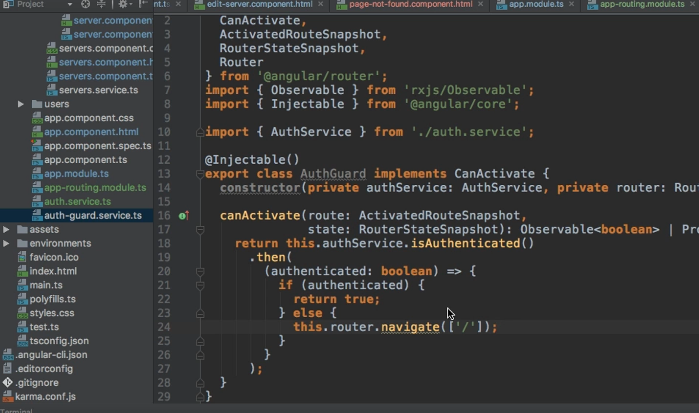
* -: As mentioned in the last lecture we want to maybe protect some of our routes.
* And we can use a feature offered by Angular which allows us to run some code at a point of time defined by us.
* **Guards:**
* I will add a new file in our route folder and I will name it **auth-guardservice.** ts because the code we run there is stored in such a service.
* I'll name it auth-guard though because the feature of the Angular router is called guards.
* It guards certain actions like navigating to around or away from it, but in the end it is a normal service and therefore as such I will export a class which I'll name AuthGuard.
* Now you could add Service here to be fully in line with all the other services, but since we will only use this as a guard, I will leave it at the AuthGuard name.
* **CanActivate Interface:**
* And now here I will implement the CanActiviate interface.
* This interface is provided by the Angular router package so you need to import it and it forces you to have a CanActivate method in this class.
* The CanActivate method now will receive two arguments, the *ActivatedRouteSnapshot* and the state of the router, so the *RouterStateSnapshot*.
* So make sure to also add these imports at the top here from @angularRouter and to add them here as types for the two parameters we get and CanActivate.
* Now you might ask where are we getting these arguments from? Keep in mind we will soon define that Angular should execute this code before a route is loaded.
* So it will give us this data and we simply need to be able to handle the data.
* So that is CanActivate with the arguments it requires.
* CanActivate also returns something.
* *It either returns an observable.*
* So make sure to import observable from rxjs observable, is the correct path in this case.
* This observable then will wrap a boolean so it will in the end resolve to a true or false value.
* Alternatively, this route returns a promise also returning the boolean in the end or it returns just a boolean.
* ***So CanActivate can run both asynchronously returning an observable or a promise or synchronously*** because you might have some guards which execute some code which runs completely on the client, therefore it runs synchronously.



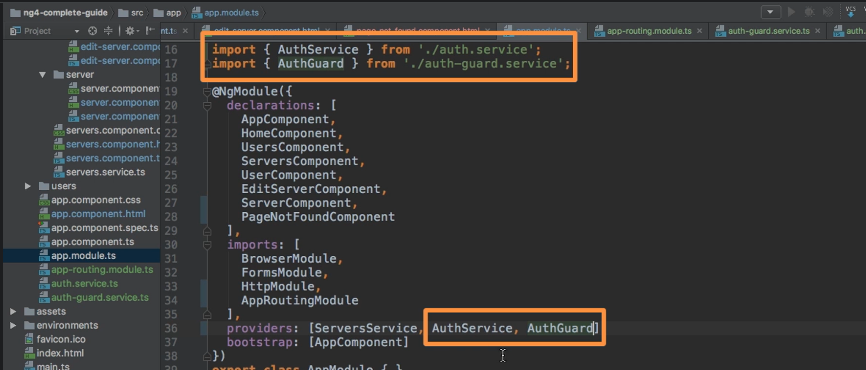
* Or you might have some code which takes a couple of seconds to finish because you use a timeout in there.
* Or you reach out to a server, so it runs asynchronously and both is possible with CanActivate, you will see an example in a second.
* So here we want to be able to log in or out.
* For this, let's say we have another service the authservice and this is just a fake service here.
* So I will name it AuthService.
* And in a real application this might reach out to a server and allow us to log in or log out and check our current authentication state.
* I will track the state here too with the loggedIn property which is set to false initially.
* And then I'll add a login method which will set loggedIn to true, just like this and a logout method which will set loggedIn to false to again fake this behavior.
* And I'll have a method which allows us to check the state authenticated.
* And here I want to simulate that this takes some time to finish because maybe we reach out to a server.
* Therefore here I will return a promise, a new promise therefore.
* This promise will as always take a method, a function as an argument with the resolve and reject methods we can execute.



* And in this promise, here I will execute set time out to wait let's say 800 milliseconds and then execute another method in which I will resolve the promise and return this loggedIn.
* Again, just to fake that, this takes a couple of seconds to finish or sometime to finish.
* So with this AuthService added I now want to use it in my AuthGuard.
* And as this is all just a service we need to add @Injectable to be able to reach out to the other service to injected service into this guard.
* So make sure to import, @injectable from Angular core and then I will add a constructor to my AuthGuard where I reach out to my fake AuthService here of type Angular service.
* Therefore make sure to add this import too.
* And now in the CanActivate guard I simply want to check whether the user is logged in or not.
* So here I can reach out to my AuthService to the isAuthenticate method, which again keep in mind returns a promise while therefore we should return this promise here I guess.
* So it returns us this promise.
* And here I then want to be able to handle that, whenever this promise and the AuthService resolves.
* I know that there I will get back a boolean.
* So this authentication state, this loggedIn boolean, it is what we resolve here in the end.
* And this could be false or true.
* So I know that I will get back a boolean.
* And here I then want to check if this is true in which case I want to return true and otherwise I want to navigate away because I don't want to allow the user access to the route he wanted to go to originally.
* I will navigate a way to force the user to go somewhere else.



* So I will inject the Angular router for this because this is how we can navigate with the navigate method.
* Make sure to add the import from @angular router and then here we can navigate with the navigate method.
* Let's say we just want to go back to our route page with just slash.
* So in this case our CanActivate method will navigate away and it will now simply return this promise it still is in the end because if we return something inside of the promise, it will give us back another promise.
* So now we're returning a promise here which gives us back true or simply navigates us away canceling the old navigation anyways therefore.
* So this now allows us to control access to whatever is controlled by the CanActivate guard here.
* Now we're still not using this guard of course.
* **Using the Guard:**
* So to use it, I'll go to my app routing module.
* And now we want to define which route or routes should be protected by this guard.
* ***And we do so by going to that route, it's the service route and all its child routes, and adding CanActivate, this property to it.***
* *Now CanActivate, takes an array of all the code basically all the guards you want to apply to this route and it will automatically get applied to all the child routes.*
* So here CanActivate, will use my AuthGuard make sure to add the import here pointing to your file.
* And this will make sure that servers is now only accessible and all the child routes only accessible if the AuthGuard CanActivate method returns true in the end which will only happen if in the AuthService loggedIn is set to true.
* Since this is set to false and right now we never call login.
* This should always deny us access.
* Now, before trying this, we need to go to our app module and add to the two new services we added.
* So add the AuthService and the AuthGuard which also is a service.



* Otherwise Angular won't be able to inject them for us.
* So with that set up, let's go to our application.
* Click on servers and I can't click there as often as I want.

Graphical user interface, text, application, email

Description automatically generated

* I can't reach it.
* Users works, but servers always navigates us back to home after these 800 milliseconds because that is how long it takes to resolve the information whether we are authenticated or not.
* We defined these 800 seconds here.
* So our guard is working, however, on our whole server's tab.
* *Now I want to be able to see the list of servers and only protect the child routes.*
* How can we do this.