Exercise 12 - Route guards

Objective

To use route guards to prevent access to routes a user is not authorised to access, and to protect them from navigating away from a route when they have not submitted their changes.

Overview

In the first part of this exercise we will create a route guard to stop "unauthorised" access to certain parts of our application. We will follow this up with a confirmation dialogue for users who attempt to navigate away from the course-edit form, to ensure they do not lose any unsaved work.

Instructions

Part 1 - canActivate guards

1. So that we can simulate a user being logged-in with the appropriate permissions, create a user service with a public property "authLevel". Set it to 4; for the sake of this exercise we'll assume that this represents a user with course edit privileges.
2. Create a canActivate guard called AdminGuard
3. Inject the user service you just created
4. The AdminGuard should check if the user has an authLevel of 4 or greater and return true or false appropriately.
5. Given we're going to want to access these services application wide, import the users module into the root module
6. Add the guards to the routes declared for the course edit component
7. Check your application still works and you can still navigate to the course edit component, and then change the authLevel to a number lower than 4 and re-test. You should find that you won't be able to reach it.

Part 2 - canDeactivate guards

We're now going to set up a route guard to warn users before they navigate away from the course-edit component, helping them to not forget to submit changes before they lose them.

8. Start off by creating a canDeactivateGuard and provide it into the root module
9. We don't want to tell this guard about component implementation details so create a class interface CanComponentDeactivate.

10. Within this interface describe a `canDeactive()` function which returns an Observable or Promise that resolves to boolean, or just a boolean.
11. The CanDeactiveGuard should implement CanDeactivate (import this symbol from the @angular/router module) with a type of CanComponentDeactivate.
12. Within the guard write a canDeactivate() function which takes a component as its argument and returns an Observable/Promise or Boolean. This is the function that will be called whenever an attempt is made to deactivate a route guarded by this guard.
13. The canDeactivate function should return the result of calling the component's canDeactivate function.
14. Any components we wish to guard against deactivating should have this guard added to their route definition, and implement CanComponentDeactivate, which allows component to have its own implementation.
15. Add the guard to the route for the course-editor component and implement the canDeactivate function to return the result of a confirmation dialogue

```
canDeactivate() {
    return confirm('You will lose any unsubmitted changes');
}
```