Auth Guard

TABLE OF CONTENTS

Introduction 3

Auth Guard 3

What is Auth Guard? 3

How Auth Guard works? 3

Workflow Diagram 3

Implementation 4

Step 1: Create Auth Guard with CLI 4

Step 2: Implement canActivate method 4

Step 3: Use Auth Guard to protect routes 6

Conclusion 6

Anandhakumar R

# Introduction

We need to protect the routes of our application from the Unauthorized user. That’s why we are using Auth guard. This documentation will provide a detailed explanation of Auth Guard as well as a step-by-step procedure for better implementing them.

# Auth Guard

## What is Auth Guard?

Auth Guard is an angular guard that is used to protect routes from unauthorized access in Angular.

## How Auth Guard works?

Authguard provides lifecycle events like canActivate. canActivate is an Interface that a class can implement to be a guard deciding if a route can be activated. If all guards return true, navigation continues. If any guard returns false, navigation is cancelled. If any guard returns a UrlTree, the current navigation is cancelled and a new navigation begins to the UrlTree returned from the guard.

We can write our user authorization and authentication logic inside the canActivate function.

## Workflow Diagram

Diagram

Description automatically generated

# Implementation

## Step 1: Create Auth Guard with CLI

Open the CLI inside your Angular project, move to the project directory, and enter the below command to create the Authguard. This command will create Auth Guard in the guards folder located within the core module.



The above command will give you options to select which interface you want to implement, as shown in the image below. Select canActivate to implement.

Text

Description automatically generated

After selecting the canActivate interface, Angular generates the AuthGuard class, as shown in the below image.

Graphical user interface, text, application

Description automatically generated

## Step 2: Implement canActivate method

Inside the canActivate method, we need to write the logic to check whether the user is logged in or not, and based on that, we need to return the result. If the user is logged in, we have to return true; otherwise, we need to return false or urlTree.

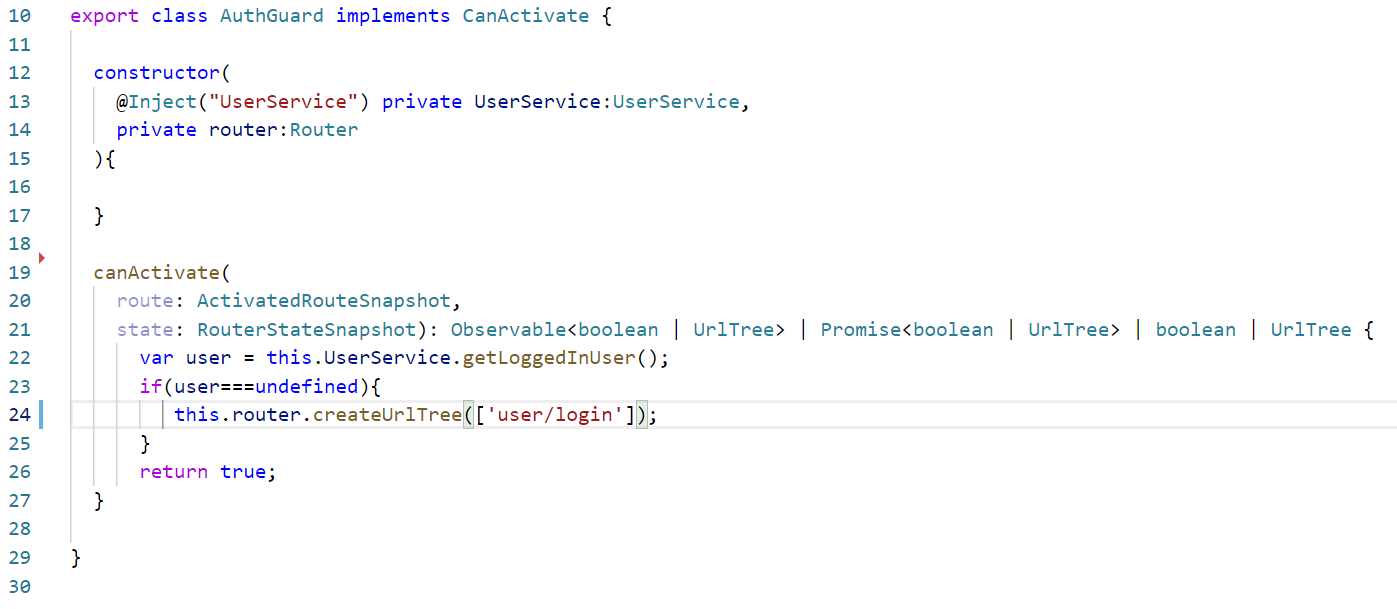
If we return true, the routing will be procceded. If we return urlTree, the url requested by the user will be terminated and the url you are returning will be continued.

For example, if the user is not logged in and he is trying to access the page, we use urlTree to move him to the login page.

Graphical user interface, text

Description automatically generated

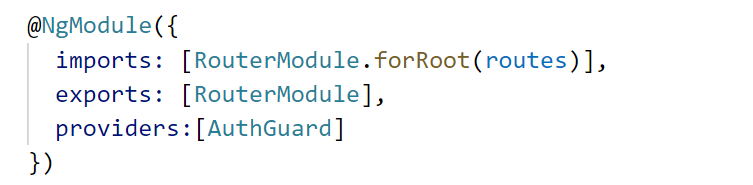
In the above image, I have the getLoggedInUser method in UserService. If user is logged in, it will return user; otherwise, it will return undefined.



With the getLoggedInUser method, canActivate will check whether the user is logged in or not. If user is logged in, canActivate will return true and routing will proceed; otherwise, it will return urlTree to the login page.

## Step 3: Use Auth Guard to protect routes

Provide the Authguard service in the providers of the module in which you are going to use Authguard.



We have to update our routing file so that we know which routes are protected by AuthGuard and which routes are accessible to every user.

In the example below, I use authguard for the booking list url. So only logged-in users can reach the booking list page.



# Conclusion

In this documentation, we have created the sample Auth Guard with canActivate method that will get the logged-in user from the User Service. And based on that value, Auth Guard will work. And we added Auth Guard to some of the routes in our application to see how it worked.