Hide Navbar Menu from Login page

Outline

- Example1: Using *ngIf to “hide” the NavBar
  - The app-material module
  - Creating the Login component
  - Creating the Home component
  - Creating the AuthService
  - Configuring the Router and the AuthGuard
  - Updating the AppComponent
  - Creating the Navigation Bar

Example 1: Using *ngIf to “hide” the NavBar

- We will have only one page layout and we will verify if the user is logged in and use *ngIf to verify if the application should display the navigation bar or not.
- This is the most common example we find when searching for how to hide the navbar when displaying the login page.

```
ng new angular-login-hide-navbar-ngif --routing --style=scss
```

Example 1(cont.)

- We will need to create some components, a module, a service, a route guard and a model interface:

ng g m app-material

ng g s auth/auth *--module=app.module*

ng g g auth/auth *--module=app.module*

ng g i auth/user

ng g c header -is

ng g c home -is -it

ng g c login

our app.module.ts and import AppMaterialModule

1.1: The app-material module

- We will need some UI components.
- Since we are using Angular Material, we will need the following Material modules to be imported by our application.
- Our app.module.ts and import AppMaterialModule

```
import { NgModule } from '@angular/core';

import { MatToolbarModule } from '@angular/material/toolbar';
import { MatCardModule } from '@angular/material/card';
import { MatButtonModule } from '@angular/material/button';
import { MatInputModule } from '@angular/material/input';
import { MatFormFieldModule } from '@angular/material/form-field';

@NgModule({
  exports: [
    MatToolbarModule,
    MatCardModule,
    MatInputModule,
    MatFormFieldModule,
    MatButtonModule
  ]
})
export class AppMaterialModule {}
```

1.1(cont.)

- We imported MatInputModule, this means we will work with forms.
- So we also need to import ReactiveFormsModule or FormsModule
  - if you prefer to work with template driven forms

```
import { BrowserAnimationsModule } from '@angular/platform-
browser/animations';

import { ReactiveFormsModule } from '@angular/forms';

import { AppMaterialModule } from './app-material/app-material.module';

@NgModule({
  // ...
  imports: [
    // ...
    ReactiveFormsModule,
    BrowserAnimationsModule,
    AppMaterialModule
  ],
  // ...
})
export class AppModule { }
```

1.2: Creating the Login component

- Our login screen is going to be a simple form with two fields: user name and password and they are both required ({5}).
- So first we need to declare a form variable ({1}).
- Use the FormBuilder instead of instantiating each FormGroup and FormControl, so in this case we also need to inject it in our component's contructor ({3}).
- we are in the constructor, when the user clicks on the login button and the form is valid, we will submit its values ({7}) to the AuthService that will be responsible for the login logic.
- We also need to inject it in the component's contructor ({4}).
- To display some validation error messages in our form, we'll verify if the field is invalid or has been touched (received focus) ({6}).
- We'll also be using the submit attempt flag approach ({2} and {8}).

In login.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormGroup, FormBuilder, Validators } from
'@angular/forms';
import { AuthService } from './../auth/auth.service';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss']
})
export class LoginComponent implements OnInit {
  form: FormGroup;                      // {1}
  private formSubmitAttempt: boolean; // {2}

  constructor(
    private fb: FormBuilder,          // {3}
    private authService: AuthService // {4}
  ) {}
```

```
ngOnInit() {
    this.form = this.fb.group({     // {5}
      userName: ['', Validators.required],
      password: ['', Validators.required]
    });
  }

  isFieldInvalid(field: string) { // {6}
    return (
      (!this.form.get(field).valid && this.form.get(field).touched) ||
      (this.form.get(field).untouched && this.formSubmitAttempt)
    );
  }

  onSubmit() {
    if (this.form.valid) {
      this.authService.login(this.form.value); // {7}
    }
    this.formSubmitAttempt = true;             // {8}
  }
}
```

In login.component.scss

```
mat-card {
  max-width: 400px;
  margin: 2em auto;
  text-align: center;
}
.signin-content {
  padding: 60px 1rem;
}
.full-width-input {
  width: 100%;
}
```

After login.component.html

```
<div class="signin-content">
  <mat-card>
    <mat-card-content>
      <form [formGroup]="form" (ngSubmit)="onSubmit()">
        <p>Please login to continue</p>
        <mat-form-field class="full-width-input">
          <input matInput placeholder="User" formControlName="userName"
required>
          <mat-error *ngIf="isFieldInvalid('userName')">
            Please inform your user name
          </mat-error>
        </mat-form-field>
        <mat-form-field class="full-width-input">
          <input matInput type="password" placeholder="Password"
formControlName="password" required>
          <mat-error *ngIf="isFieldInvalid('userName')">
            Please inform your password
          </mat-error>
        </mat-form-field>
        <button mat-raised-button color="primary">Login</button>
      </form>
    </mat-card-content>
  </mat-card>
</div>
```

1.3 Creating the Home component

- Our Home component is going to be very simple. Just a message confirming the user is logged in:

```
import { Component } from '@angular/core’;
@Component({
    selector: 'app-home’,
    template: '<p>Yay! You are logged in!</p>’,
    styles: []
})
export class HomeComponent {}
```

1.4 Creating the AuthService

- For the purpose of this example, we will not integrate the service with any backend API.

```
import { Injectable } from '@angular/core';
import { Router } from '@angular/router';
import { BehaviorSubject } from 'rxjs';
import { User } from './user';
```

```
@Injectable()
export class AuthService {
  private loggedIn = new BehaviorSubject<boolean>(false); // {1}

  get isLoggedIn() {
    return this.loggedIn.asObservable(); // {2}
  }

  constructor(
    private router: Router
  ) {}
```

```
login(user: User){
    if (user.userName !== '' && user.password !== '' ) { // {3}
      this.loggedIn.next(true);
      this.router.navigate(['/']);
    }
  }

  logout() {                                  // {4}
    this.loggedIn.next(false);
    this.router.navigate(['/login']);
  }
}
```

1.4(cont.)

- To control if the user is logged in or not, we will use a BehaviorSubject ({1}).
- We will also create a getter to expose only the get method publicly ({2}) as also expose the Subject as an Observable.
- When the user clicks on the login button from the form, the login method is going to be called receiving the form values. Our validation are only checking if the values are not empty.
- If we received a userName and a password ({3}), then we authenticate the user. This means we need to emit that the user is now logged in and also redirect the routing to the HomeComponent.
- The user logs out of the application ({4}), redirect to the login page.

1.4(cont.)

- In the User inteface:

```
export interface User {
    userName: string;
    password: string;
}
```

1.5: Configuring the Router and the AuthGuard

- Will be the routes we will use in this example:

```
const routes: Routes = [
{ path: '', component: HomeComponent, canActivate: [AuthGuard] },
{ path: 'login', component: LoginComponent }
];
```

- Now let's take a look the auth/auth.guard:

```
import { Injectable } from '@angular/core';
import { ActivatedRouteSnapshot,
  CanActivate,
  Router,
  RouterStateSnapshot } from '@angular/router';
import { Observable } from 'rxjs';
import { map, take } from 'rxjs/operators';

import { AuthService } from './auth.service';

@Injectable()
export class AuthGuard implements CanActivate {
  constructor(
    private authService: AuthService,
    private router: Router
  ) {}
```

```
canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): Observable<boolean> {
    return this.authService.isLoggedIn          // {1}
      .pipe(
        take(1),                                // {2}
        map((isLoggedIn: boolean) => {          // {3}
          if (!isLoggedIn){
            this.router.navigate(['/login']);  // {4}
            return false;
          }
          return true;
        })
      )
  }
}
```

1.5 (cont.)

- First we are going to retrieve the isLoggedIn ({1}) getter from the AuthService, which is an Observable.
- Since we are only interested in checking the value from the Observalbe a single time (if the user is logged in or not), we will use the take operator ({2}).
- We will verify the value emitted by the BehaviorSubject ({3}).
- If not logged in we will navigate to the login screen ({4}) and return false. The AuthGuard will return true in case the user is logged in, meaning the user can access the route (path: ‘’) which renders the HomeComponent.

1.6: Updating the AppComponent

- We AppComponent will be our main component:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
    <app-header></app-header>
    <router-outlet></router-outlet>
  `,
  styles: []
})
export class AppComponent {}
```

1.7: Creating the Navigation Bar

- We create the navigation bar with the simplest template:

```
<mat-toolbar color="primary">
  <span>
    <img src="assets/img/angular_whiteTransparent.svg" class="angular-logo">
    Angular NavBar + Login Example #01
  </span>
  <span class="fill-remaining-space"></span>
  <button mat-button>Menu Option 01</button>
  <button mat-button>Menu Option 02</button>
  <button mat-button routerLink="login">Login</button>
  <button mat-button (click)="onLogout()">Logout</button>
</mat-toolbar>
```

In header.component

```
import { Observable } from 'rxjs/Observable';
import { AuthService } from './../auth/auth.service';
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styles: [
    `.angular-logo {
        margin: 0 4px 3px 0;
        height: 35px;
        vertical-align: middle;
    }
    .fill-remaining-space {
      flex: 1 1 auto;
    }
    `
  ]
})
```

```
export class HeaderComponent implements OnInit {

  isLoggedIn$: Observable<boolean>;                    // {1}

  constructor(private authService: AuthService) { }

  ngOnInit() {
    this.isLoggedIn$ = this.authService.isLoggedIn; // {2}
  }

  onLogout(){
    this.authService.logout();                        // {3}
  }
}
```

1.7 (cont.)

- First, we will declare an Observable ({1}).
- to receive the value emitted from the AuthService ({2}).
- We are using $ at the end of the Observable identifier.
- When the user clicks on the logout we will call the logout method ({3}) from the AuthService.

1.7 (cont.)

- Let’s back to the HTML template and use *ngIf to display the navbar or not:

```
<mat-toolbar color="primary" *ngIf="isLoggedIn$ | async">
```

- We also want to verify if the user is logged in and display the Logout button:

```
<button mat-button (click)="onLogout()" *ngIf="isLoggedIn$ |
async">Logout</button>
```

- We are subscribing to the isLoggedIn$ twice

1.7 (cont.)

- We can write a better code and only subscribe once by using the as alias introduced in Angular v4.0 enhanced *ngIf and *ngFor:

```
<mat-toolbar color="primary" *ngIf="isLoggedIn$ | async as isLoggedIn">
  <!-- more HTML template code -->
  <button mat-button (click)="onLogout()" *ngIf="isLoggedIn">Logout</button>
</mat-toolbar>
```

- The code above means we are subscribing to the isLoggedIn$ and storing its value in the isLoggedIn local template variable.