**Highlights:**

* Understanding routing guards
* Using guards to navigate

**Demo Steps:**

**Problem Statement**: Consider a scenario where a route is only accessible on weekdays (Monday to Friday), for weekends the applications redirects back to home page.

For this demo, the link **'/home'**should not be accessible on weekends.

**Step 1**: Create a *UserService*as shown below:

1. import { Injectable } from '@angular/core';
2. @Injectable()
3. export class UserService {
5. canLoginToday(): boolean {
6. console.log("Hello")
7. let today = new Date();
8. console.log(today.getDay())
9. if(today.getDay() == 0 || today.getDay() == 6 ) {
10. return false; *// weekends*
11. }
12. else return true;
13. }
14. }

**Step 2**: Create a guard service called *AccessGuard*as shown below:

1. import { Injectable } from '@angular/core';
2. import { CanActivate, Router } from '@angular/router';
3. import { UserService } from './user-service'
4. @Injectable()
5. export class AccessGuardService implements CanActivate {
6. constructor(private serv:UserService, private router: Router ) {};
7. canActivate() {
8. let data = this.serv.canLoginToday()
9. console.log("here",this.serv.canLoginToday())
10. if(this.serv.canLoginToday()) {
11. this.router.navigate(['/login'])
12. return true;
14. } else {
15. return false;
16. }
18. }
19. }

**Step 3**: Add the following routes to *app-routing.module.ts.*

1. const routes:Routes = [
2. {path: '',redirectTo:"home", pathMatch: 'full',},
3. {path:"home",component:HomeComponent ,canActivate:[AccessGuardService]},
4. {path:"login",component:SuccessComponent}
5. ];

**Step 4**: Create a simple *HomeComponent and SuccessComponent*. Do the necessary changes in AppComponent, AppModule and AppRoutingModule, to include *HomeComponent*in the app.

**Step 5:**Add following to AppComponent.HTML:

1. <div>
2. <h3>
3. Routing guard demo<br>
4. <a routerLink="/home"></a>
5. <a routerLink ="/login"></a>
6. </h3>
7. <br/>
8. <br/>
9. <router-outlet></router-outlet>
10. </div>

**Observation**: Try navigating to *localhost:4200/home,*and observe the guard in action. The guard will not allow users to access the link on weekends.

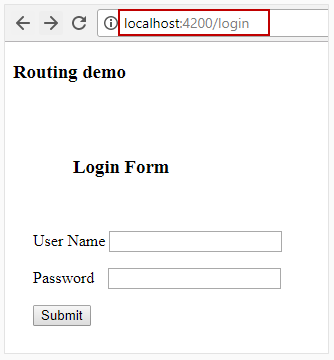
# ****Highlights:****

* Understanding routing guards
* Using guards to navigate

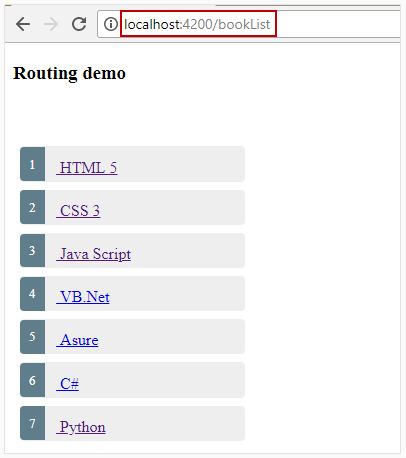
**Demo Steps:**

**Problem Statement**: Consider the same example used for routing.  Add route guard to BookComponent. Only after logging in , user should be able to access BookComponent. Output is as shown below

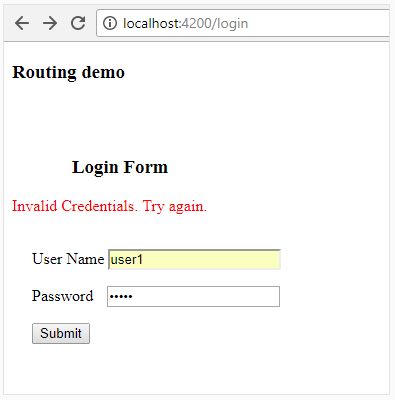
**Note:** You will add the **route guards** to **Demo** that you had created to implement routing.



* On successful validation:



* On unsuccessful validation



1. Create a Login component.

1. ng generate component login

2. Add the following code to **login.component.html** file

1. <h3 style="position:relative;left:60px">Login Form</h3>
2. <div \*ngIf="invalidCredentialMsg" style="color:red">{{invalidCredentialMsg}}</div><br/>
3. <div style="position:relative;left:20px">
4. <form [formGroup]="loginForm" (ngSubmit)="onFormSubmit()">
5. <p>User Name <input formControlName="username"></p>
6. <p>Password <input type="password" formControlName="password" style="position:relative;left:10px"></p>
7. <p><button type="submit">Submit</button></p>
8. </form>
9. </div>

**Line 2:** div tag will render error message for incorrect credentials

**Line 4-8:** A reactive form with two fields username and password is displayed

3. Add the following code to**login.component.ts**file

1. import { Component } from '@angular/core';
2. import { FormBuilder, FormGroup } from '@angular/forms';
3. import { Router } from '@angular/router';
4. import { LoginService } from './login.service';
5. @Component({
6. templateUrl: './login.component.html',
7. styleUrls: ['./login.component.css']
8. })
9. export class LoginComponent {
10. invalidCredentialMsg: string='';
11. loginForm!: FormGroup;
12. constructor(private loginService: LoginService, private router: Router, private formbuilder: FormBuilder) {
13. this.loginForm = this.formbuilder.group({
14. username: [],
15. password: []
16. });
17. }
19. onFormSubmit() {
20. let uname = this.loginForm.get('username')!.value;
21. let pwd = this.loginForm.get('password')!.value;
22. this.loginService.isUserAuthenticated(uname, pwd).subscribe(
23. authenticated => {
24. if (authenticated) {
25. this.router.navigate(['bookList']);
26. } else {
27. this.invalidCredentialMsg = 'Invalid Credentials. Try again.';
28. }
29. }
30. );
31. }
32. }

**Line 19:** onFormSubmit() method is invoked when submit button is clicked in Login Form

**Line 20-21:** Fetching username and password values

**Line 22:** Invoking isUserAuthenticated method of LoginService class which will check for the validity of username and password values and returns a Boolean value

**Line 23-29:** If the response is true, it will navigate to BookComponent else assigns error message to invalidCredentialMsg property

4. Create **user.ts** file under login folder and add the following code to **user.ts** file

1. export class User {
2. constructor(public userId:number, public username:string, public password:string) { }
3. }

**Line 1-3:** A User model class with three properties userId, username and password is created

5. Add the following code to**login.service.ts** file

1. import { Injectable } from '@angular/core';
2. import { User } from './user';
3. import { Observable, of } from 'rxjs';
4. import { map } from 'rxjs/operators';
5. const USERS = [
6. new User(1, 'user1', 'user1'),
7. new User(2, 'user2', 'user2')
8. ];
9. let usersObservable = of(USERS);
10. @Injectable()
11. export class LoginService {
12. private isloggedIn: boolean = false;
13. getAllUsers(): Observable<User[]> {
14. return usersObservable;
15. }
16. isUserAuthenticated(username: string, password: string): Observable<boolean> {
17. return this.getAllUsers().pipe(map(users => {
18. let user = users.find(user => (user.username === username) && (user.password === password));
19. if (user) {
20. this.isloggedIn = true;
21. } else {
22. this.isloggedIn = false;
23. }
24. return this.isloggedIn;
25. }));
26. }
27. isUserLoggedIn(): boolean {
28. return this.isloggedIn;
29. }
30. }

**Line 2:** Imports User model class

**Line 7-10:** Creates an array called USERS of type User

**Line 11:** Converts USERS array as an observable type

**Line 19-21:** getAllUsers() method returns usersObservable array

**Line 23:** isUserAuthenticated method takes username and password values as inputs and return a Boolean value of type Observable

**Line 24-28:** Invokes getAllUsers() methods which returns an observable array. After receiving it, it will find the entered credentials exist in the array or not. If user exists, assigns true value to isloggedIn property otherwise false value to it

**Line 37-39:** isUserLoggedIn() method returns the value of isloggedIn which we will use in LoginGuardService class

6. Create another service class called **login-guard.service** and add the following code

1. import { Injectable } from '@angular/core';
2. import { CanActivate, Router } from '@angular/router';
3. import { LoginService } from './login.service';
4. @Injectable()
5. export class LoginGuardService implements CanActivate {
6. constructor(private loginService: LoginService, private router: Router) { }
7. canActivate(): boolean {
9. if (this.loginService.isUserLoggedIn()) {
10. return true;
11. }
13. this.router.navigate(['/login']);
14. return false;
15. }
16. }

**Line 2:** Inherits CanActivate interface to LoginGuardService class

**Line 11:** Overrides canActivate() method

**Line 13-18:** Invokes isUserLoggedIn method from LoginService class which returns a Boolean value representing whether a user is logged in or not. If user logs in, canActivate returns true otherwise navigates to login component asking user to login first to access BookComponent

7. Modify **app.component.html** with the below code

1. <div>
2. <h3>
3. Routing demo
4. </h3>
5. <br/>
6. <br/>
7. <router-outlet></router-outlet>
8. </div>

7. Add the following code in **app.module.ts**

1. import { NgModule } from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. import { HttpClientModule } from '@angular/common/http';
4. import { AppComponent } from './app.component';
5. import { BookComponent } from './book/book.component';
6. import { BookService } from './book/book.service';
7. import { BookDetailsComponent } from './book-details/book-details.component';
8. import { LoginComponent } from './login/login.component';
9. import { LoginService} from "./login/login.service";
10. import {LoginGuardService} from './login/login-guard.service';
11. import { AppRoutingModule } from './app-routing.module';
12. import {ReactiveFormsModule, FormsModule} from '@angular/forms';
13. @NgModule({
14. imports: [BrowserModule, HttpClientModule,AppRoutingModule,ReactiveFormsModule,FormsModule],
15. declarations: [AppComponent, BookComponent, BookDetailsComponent, LoginComponent],
16. providers: [BookService,LoginService,LoginGuardService],
17. bootstrap: [AppComponent]
18. })
19. export class AppModule { }

8. Modify the value of **routes array** in **app-routing.modules.ts**with the value given below

1. const routes: Routes = [
2. { path: '', redirectTo: 'login', pathMatch:'full' },
3. { path: 'login', component: LoginComponent},
4. { path: 'bookList', component: BookComponent },
5. { path: 'book/:id', component: BookDetailsComponent,canActivate:[LoginGuardService]}
6. ];

9. Save the files and check the output in the browser.

Provide the credentials to the login page as

* User name: user1
* Password: user1

Click on the book title to navigate to BookDetails component.

Now re-run the application and provide the path http://localhost:4200/bookList The BookComponent gets loaded.

Now try clicking on the book title, you will be redirected to **login** component rather than the **bookDetails**component. This is because you have implemented routing guard.