

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
Інститут комп'ютерних систем
Кафедра інформаційних систем

Звіт
Лабораторна робота № 2

З дисципліни «Проектний підхід до розробки ПЗ»

**Тема: «Інструменти механізму аутентифікації і авторизації в
Angular»**

Виконав:
студент групи УІ-191
Кисельов Д.С.

Перевірив:
Малерик Р.П.

Тема: Інструменти механізму аутентифікації і авторизації в Angular.

Мета роботи:

- розглянути можливі сховища клієнтських додатків;
- розглянути варіанти використання angular interceptors;
- розглянути механізми аутентифікації;
- інструменти для авторизації і обмеження доступу до компонентів.

Структура протоколу до лабораторної роботи:

1. Завдання.
2. Код на МП Typescript.
3. Знімки екрану з поясненнями.
4. Висновки про виконану роботу.

1. Завдання

1. Створіть компонент з формою для логіна.
2. Створіть сервіс AuthService з методом login (username: string, password: string), який буде відправляти get-запит на адресу <https://pnitfunctions.azurewebsites.net/api/token?userName={login}&password={pw}> для отримання JWT токена. Відкрийте відповідні повідомлення успішного і невдалого входу на сторінці.
3. За допомогою приватного методу і ripable оператора tap () виконайте збереження токена в localStorage або cookies.
4. Протестуйте форму, використовуючи дані (логін: пароль): admin: 12345, user: 55555.
5. За допомогою механізму інтерсепторів реалізуйте впровадження токена для http-запитів, а також логіну і переадресацію на сторінку логіна при закінченні сесії.
6. Створіть сервіс OrdersService з методом getAll () і додайте його виклик в відповідний компонент. API url для отримання даних:
<https://pnitfunctions.azurewebsites.net/api/GetOrders>

2. Код на МП Typescript

2.1 auth.service.ts:

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Router } from '@angular/router';

@Injectable({
  providedIn: 'root'
})

export class AuthService {

  private readonly accessTokenKey = 'access_token';

  authorizationError = false;

  constructor(private http: HttpClient, private router: Router) { }

  login(username: string, password: string): any {
    return
    this.http.get('https://pnitfunctions.azurewebsites.net/api/token?userName=' +
    username + '&password=' + password);
  }

  setToken(username: string, password: string): void {
    const temp = this.login(username, password);
    temp.subscribe(
      (readableToken) => {
        console.log('Server response', readableToken);
        const token = readableToken.access_token;
        localStorage.setItem(this.accessTokenKey, token);
        this.authorizationError = false;
        this.router.navigate(['/table']);
      }
    );
  }

  getToken(): string {
    return localStorage.getItem(this.accessTokenKey);
  }

  logout(): void {
    localStorage.removeItem(this.accessTokenKey);
    console.log('Logout!');
    this.router.navigate(['/login']);
  }

  checkForExperation(): boolean {
    const token = this.getToken();
    const expiry = (JSON.parse(atob(token.split('.')[1]))).exp;
```

```

    console.log(expiry);

    // tslint:disable-next-line:new-parens
    if ((Math.floor((new Date).getTime() / 1000)) >= expiry) {
        return true;
    }
    else {
        return false;
    }
}
}

```

2.2 auth.interceptor.ts:

```

import { Injectable } from '@angular/core';
import {
    HttpRequest,
    HttpHandler,
    HttpEvent,
    HttpInterceptor,
    HttpResponse
} from '@angular/common/http';
import { Observable, throwError } from 'rxjs';

import { AuthService } from '../services/auth.service';
import { catchError } from 'rxjs/operators';

@Injectable()
export class AuthInterceptor implements HttpInterceptor {

    constructor(private authService: AuthService) {}

    intercept(request: HttpRequest<any>, next: HttpHandler):
    Observable<HttpEvent<any>> {

        request = request.clone({
            setHeaders: {
                Authorization: `Bearer ${this.authService.getToken()}`
            }
        });

        return next.handle(request).pipe(
            catchError((error: HttpResponse) => {
                let errorMsg = '';
                if (error.error instanceof ErrorEvent) {
                    console.log('This is client side error');
                    errorMsg = `Error: ${error.error.message}`;
                }
                else {
                    console.log('This is server side error');

```

```

        errorMsg = `Error Code: ${error.status}, Message: ${error.message}`;
        this.handleAuthError(error);
    }
    console.log(errorMsg);
    return throwError(errorMsg);
  })
);
}

private handleAuthError(err: HttpResponse): Observable<any> {
  if (err.status === 401 || err.status === 403) {
    this.authService.logout();
    this.authService.authorizationError = true;
  }
  return throwError(err);
}
}

```

2.3 auth.guard.ts

```

import { Injectable } from '@angular/core';
import { ActivatedRouteSnapshot, CanActivate, Router, RouterStateSnapshot,
  UrlTree } from '@angular/router';
import { Observable } from 'rxjs';
import { AuthService } from '../services/auth.service';

@Injectable({
  providedIn: 'root'
})
export class AuthGuardGuard implements CanActivate {

  constructor(private authService: AuthService, private router: Router) { }

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> |
    Promise<boolean | UrlTree> | boolean | UrlTree {
    if (!this.authService.getToken()) {
      this.router.navigate(['login']);
      return false;
    } else {
      if (this.authService.checkForExpiration()) {
        this.router.navigate(['login']);
        alert('Your session has been expired. New authorization required');
        this.authService.logout();
        return false;
      }
      return true;
    }
  }
}

```

2.4 login.component.ts:

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

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss']
})
export class LoginComponent implements OnInit {

  form: FormGroup;
  isEmpty: boolean;
  correct = true;

  constructor(private fb: FormBuilder, public authService: AuthService) {

    this.form = this.fb.group({
      login: ['', Validators.required],
      password: ['', Validators.required]
    });
  }

  login(): void {

    console.log('Click!');

    const val = this.form.value;
    if (val.login && val.password) {
      this.authService.login(val.login, val.password)
        .subscribe(
          () => {
            this.correct = true;
            console.log('User is logged in');
            this.authService.setToken(val.login, val.password);
            console.log('Token: ' + this.authService.getToken());
          }
        );
    }

    else {
      this.isEmpty = true;
    }

    if (!(this.isEmpty)) {
      this.correct = false;
    }
  }
}
```

```

    ngOnInit(): void {
    }
}

```

2.5 orders-service.ts

```

import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { map } from 'rxjs/operators';
import { Order } from '../model/order.model';

@Injectable({
  providedIn: 'root'
})
export class OrdersService {

  constructor(private http: HttpClient) { }

  getAll(): Observable<Order[]> {
    return
this.http.get('https://pnitfunctions.azurewebsites.net/api/GetOrders')
    .pipe(map((data: any[]) => data.map((order: any) => new Order(order.name,
order.price, order.category))));
  }

}

```

2.6 table-component.ts

```

import { Component, OnInit } from '@angular/core';
import { Order } from '../model/order.model';
import { OrdersService } from '../services/orders.service';

@Component({
  selector: 'app-orders',
  templateUrl: './table.component.html',
  styleUrls: ['./table.component.scss']
})
export class TableComponent implements OnInit {
  orders: Order[];

  constructor(private orderService: OrdersService) {
  }

  ngOnInit(): void {
    this.orderService.getAll().subscribe((orders: Order[]) => this.orders =
orders);
  }
}

```

```
}
```

2.7 app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { AuthGuardGuard } from '../guards/auth-guard.guard';
import { LoginComponent } from '../login/login.component';
import { TableComponent } from '../table/table.component';

const routes: Routes = [
  {
    path: 'login',
    component: LoginComponent
  },
  {
    path: 'table',
    component: TableComponent,
    canActivate: [AuthGuardGuard],
  }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModuleModule { }
```

2.8 app-module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HTTP_INTERCEPTORS } from '@angular/common/http';
import { HttpClientModule } from '@angular/common/http';
import { AppRoutingModuleModule } from '../app-routing.module';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { AppComponent } from '../app.component';
import { LoginComponent } from '../login/login.component';
import { AuthService } from '../services/auth.service';
import { AuthInterceptor } from '../interceptors/auth.interceptor';
import { TableComponent } from '../table/table.component';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

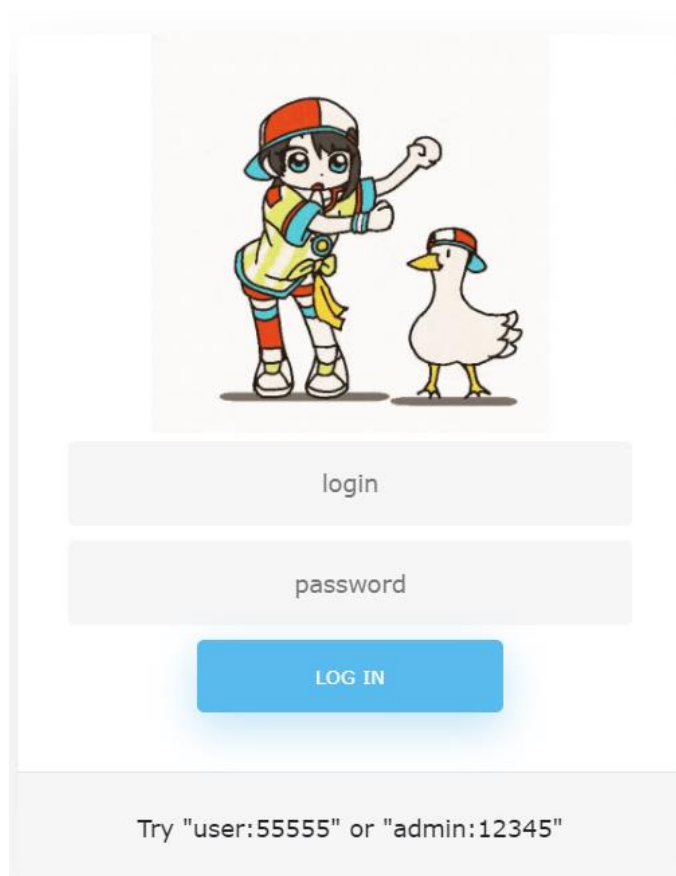
@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    TableComponent,
  ],
  imports: [
    BrowserModule,
```



```
AppRoutingModule,  
ReactiveFormsModule,  
FormsModule,  
HttpClientModule,  
BrowserAnimationsModule,  
],  
providers: [  
  {  
    provide: HTTP_INTERCEPTORS,  
    useClass: AuthInterceptor,  
    multi: true  
  }  
],  
bootstrap: [AppComponent]  
})  
export class AppModule { }
```


3. Знімки екрану з поясненнями

3.1 Форма для логіну у стандартному вигляді



The screenshot shows a login form with a light gray background. At the top, there is a cartoon illustration of a girl with black hair, wearing a red and white cap, a yellow shirt, and red and white striped pants, standing next to a white duck wearing a red and white cap. Below the illustration, there are two input fields: one labeled 'login' and one labeled 'password'. Below these fields is a blue button labeled 'LOG IN'. At the bottom of the form, there is a gray box containing the text: 'Try "user:55555" or "admin:12345"'.

3.2 Спроба надіслати порожню форму



login

password


LOG IN

Form is empty!

Try "user:55555" or "admin:12345"

У разі надсилання порожньої форми у розмітці компоненту буде виведено відповідне повідомлення.

3.3. Надсилання некоректних даних



admin

admin

LOG IN

Wrong login or password!

Try "user:55555" or "admin:12345"

У разі надсилання на сервер некоректного логіну або паролю прийде помилка «401». Вона буде перехоплена та у розмітці компоненту буде виведено відповідне повідомлення.

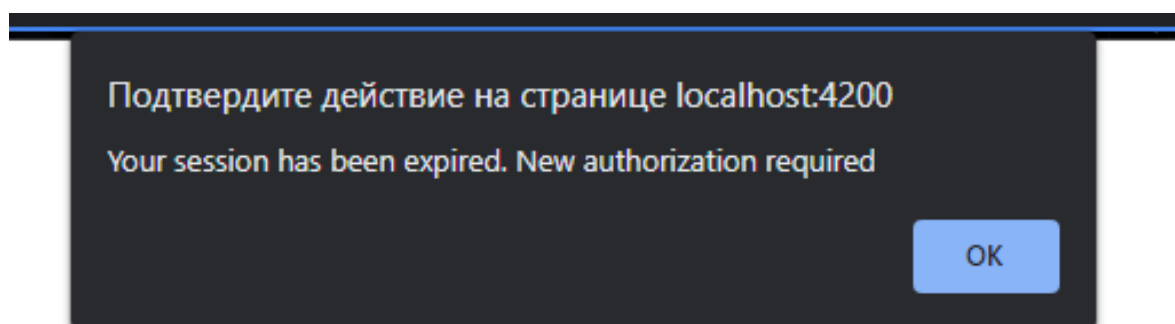
3.4 Таблиця, яка заповнюється даними з серверу

Orders

#	Name	Category	Price
0	Order 1	150.99	Category 1
1	Order 2	400.99	Category 1
2	Order 3	200.99	Category 1
3	Order 4	103.99	Category 1
4	Order 5	19.99	Category 1
5	Order 6	10.99	Category 2
6	Order 7	10.99	Category 2
7	Order 8	11.99	Category 2
8	Order 9	100.99	Category 2
9	Order 10	10.99	Category 3
10	Order 11	15.99	Category 3
11	Order 12	140.99	Category 3
12	Order 13	140.99	Category 3

Доступ до таблиці можна отримати лише за умови авторизації.

3.5 Попередження про прострочений токен



У `auth.guard.ts` при будь-якій дії користувача у застосунку виконується перевірка актуальності токена. У випадку, якщо токен прострочено, буде виведено відповідне повідомлення, а користувача направлено на сторінку авторизації.

4. Висновки про виконану роботу

В результаті виконання лабораторної роботи було створено веб-застосунок, який підтримує процедуру авторизації користувача. Авторизація в даній лабораторній роботі була основана на JWT-токені, який отримувався з віддаленого серверу за допомогою механізму http-інтерцепторів. Окрім цього, були розглянуті можливі сховища клієнтських додатків, але у даній роботі токен зберігався у такому сховищі, як `LocalStorage`.

Також у цій роботі використовувалися інструменти для авторизації і обмеження доступу до компонентів, в Angular відомі як `guards`. За завданням, за допомогою цих механізмів доступ обмежувався до таблиці – тобто, користувач міг отримати до неї доступ лише за умови наявності у нього непростроченого токена, який зберігався у локальному сховищу у випадку надсилання на сервер коректних реєстраційних даних.