

Wprowadzenie do aplikacji Internetowych

laboratorium 6

Celem laboratorium jest zaznajomienie z zagadnieniami bezpieczeństwa w aplikacjach webowych. Zapoznać się Państwo z zagadnieniami autentykacji i autoryzacji jako mechanizmowi kontroli dostępu do wybranych zasobów aplikacji. Następnie wykorzystując dane dostępowe zaimplementujemy mechanizm udostępniania danych i widoków tylko dla uprawnionych użytkowników.

Nasza aplikacja nie potrafi rozróżnić użytkowników. Dlatego pora na wprowadzenie autentykacji. Do tego celu wykorzystamy moduł Autentykacji dostępny na platformie Firebase – omawiany na wykładzie nr 7.

Sposób konfiguracji i instalacji odpowiednich modułów umożliwiających współpracę z Firebase z poziomu Angulara – omówiono szczegółowo na zajęciach lab nr 5. Jako metodę Autentykacji wybieramy - logowanie za pomocą loginu i hasła.

Zadanie 1.

Zaimplementuj funkcjonalność pozwalającą na rejestrację nowego użytkownika a potem jego logowanie. Stwórz nowe komponenty do wyświetlania dedykowanego widoku. Zabezpiecz aplikację przed wejściem do widoków użytkownika zalogowanego bezpośrednio z adresu w przeglądarce. (patrz securityGuard na końcu materiałów lab)

W celu wykorzystania metod autentykacji potrzebujemy zaimplementować funkcjonalności: Rejestracja, Logowanie i Wylogowanie. Przy okazji dokonaj refaktoryzacji istniejącej sekcje menu. Rozszerz ja o pozycje Rejestracja, Logowanie/Wylogowanie. Dokonaj odpowiedniej modyfikacji w polityce routingu tak aby poszczególne ekrany z sobą współgrały. Pamiętaj aby komponenty związane z autentykacją były dostępne kontekstowo tzn. gdy jesteś osoba niezalogowana dostępne są pozycje w menu: Rejestracja i Logowanie. Po zalogowaniu dostępna jest tylko opcja wylogowanie. Niech w pasku menu wyświetla się także informacja o użytkowniku zalogowanym.

Uwaga – musisz rozszerzyć model danych w aplikacji o nowy obiekt reprezentujący użytkownika. Zastanów się jakich danych potrzebujesz do opisu użytkownika i jego funkcjonalności.

Wskazówka. Zastosuj AngularFireAuth jako moduł dostarczający funkcjonalności autentykacji. Główne metody to: `currentUser()`, `signInWithEmailAndPassword(email,password)`, `createUserWithEmailAndPassword(email, password)`, `signOut()`.

Warto zwrócić uwagę na poniższa konstrukcje:

```

userData: Observable<firebase.User>;

constructor(private angularFireAuth: AngularFireAuth) {

  this.userData = angularFireAuth.authState;

}

```

AuthState jest strumieniem, który emituje zalogowanego użytkownika. Jeśli użytkownik zostaje wylogowany, strumień wyemituje **null**. Można to wykorzystać do otrzymywania informacji o zmianie stanu zalogowanego użytkownika.

(max ilość punktów 4)

Warto wiedzieć, że możemy manipulować stanem uwierzytelnienia w różnych scenariuszach, takich jak np. zamknięcie karty i powrót (wylogować użytkownika automatycznie czy może nie?). Otrzymujemy możliwość zmiany właściwości PERSISTENCE, np. w poniższy sposób:

```

session = this.fireAuth.auth.Persistence.SESSION;

return this.fireAuth.auth.setPersistence(session).then(() => {

  .....

});

```

Możliwe 3 opcje do wyboru:

- LOCAL (DOMYŚLNE) – użytkownik nadal zostaje zalogowany po zamknięciu karty, trzeba jawnie użyć metody `signOut` aby wyczyścić stan zalogowania. Przydatne, jeśli po zamknięciu karty i ponownym powrocie, chcemy użytkownikowi pozwolić pozostać zalogowanym.
- SESSION – stan zalogowanego użytkownika jest aktywny wyłącznie dla aktualnej sesji i zostanie wyczyszczony w przypadku zamknięcia okna/karty. Przydatne np. w aplikacjach, które są publicznie dostępne na komputerach, z których korzysta wielu użytkowników (np. w bibliotece).
- NONE – stan zalogowania jest przetrzymywany w pamięci i zostanie wyczyszczony po odświeżeniu okna.

Zadanie 2. Zaimplementuj mechanizm wyboru tryby persystencji stanu logowania. Niech będzie dostępna dla admina opcja zmiany tego stanu na jeden z powyżej wymienionych. Przetestuj działanie aplikacji przy każdej z wybranych opcji.

(max ilość punktów 1)

Zadanie 3. Chcemy wiedzieć nie tylko, z kim mamy do czynienia ale również co może on wykonać w aplikacji a do czego nie ma uprawnień. W oparciu o bazy danych firebase zrealizuj funkcjonalność serwera autoryzującego, który będzie zwracał w wersji minimalistycznej przynajmniej przypisane od użytkownika role.

Niech w naszej aplikacji istnieją następujące role: Reader, Pracownik, Admin, VIP.

Reader – widzi tylko oferty aktywne (data aktualna, są jeszcze miejsca do rezerwacji)

VIP - VIP widzi wszystkie oferty nawet te z wykupionymi miejscami

Pracownik może dodawać i modyfikować oferty, nie może ich jednak usuwać, widzi wszystkie oferty nawet te już archiwalne.

Admin – może wszystko

(max ilość punktów 3)

Zadanie5. Rozróżniamy już użytkowników. Wykorzystajmy to do realizacji funkcjonalności koszyka, oraz do przechowywania historii zamówień wycieczek. Dodatkowo wykorzystaj ten fakt do udostępnienia funkcjonalności oceny wycieczki tylko dla użytkowników, którzy zapisali się na wycieczkę przy jednoczesnym zabezpieczeniu przed wielokrotnym głosowaniem.

(max ilość punktów 3)

Zadanie 6. Dla admina/pracownika przygotuj dedykowany panel do zarządzania listą wycieczek.

Sekcja zarządzania wycieczkami umożliwia przeglądanie oferty biura podróży. Przy każdej pozycji można dokonać jej aktualizacji, usunąć.

Dodatkowo istnieje sekcja dodawania nowych wycieczek pozwalająca na wprowadzenie nowej oferty.

Dzięki stworzeniu dedykowanego panelu jednocześnie można usunąć istniejące przyciski w głównym ekranie prezentacyjnym aplikacji dostępnym dla użytkowników zalogowanych jak i niezalogowanych.

(max ilość punktów 3)

Zadanie 7. Użytkownik może również przejść od razu na daną ścieżkę dowolnego widoku z pominięciem etapu logowania, oczywiście jeśli zna ścieżkę. Zabezpiecz dostęp do sekcji Zarządzania kursami tylko dla admina. Zastosuj AuthGuard-a. Omówienie tej techniki znajdziesz na końcu materiałów laboratoryjnych. Uszczelnij również backend wprowadzając mechanizm reguł dostępowych opartych na weryfikacji uprawnień. (Zabezpiecz backend w ten sposób aby tylko osoby mające odpowiednie uprawnienia mogły wykonywać operacje modyfikujące na bazie.)

(max ilość punktów 2)

Zad 8. Zarządzanie kontami użytkowników. Domyślnie użytkownicy mają ręcznie przypisane role w bazie. Niech admin ma możliwość definiowania ról dla użytkowników zarejestrowanych w systemie poprzez aplikację. Niech będzie możliwości przypisywania mu kilku ról : Reader, Pracownik, admin, VIP.

(max ilość punktów 2)

Zad 9. Dopracuj aplikację pod kątem wizualnym. Możesz korzystać z dowolnych bibliotek i komponentów, których według Ciebie są interesujące np. Bootstrap, Material Design lub inne. Oceniać będę wrażenia estetyczne, ergonomię użycia itp.)

(max ilość punktów 2)

Ochrona ścieżek

Na ten moment dowolny użytkownik może przejść w dowolne miejsce naszej aplikacji a to nie zawsze jest pożądane przez nas zachowanie. Musimy więc napisać funkcjonalność która będzie chroniła niektóre ścieżki przed nieuprawnionym dostępem. Chronić dostęp do swojej aplikacji możesz na wiele sposobów. Proponuję zastosowanie **CanActive**

CanActive: wymagana autoryzacja

Aplikacje często ograniczają dostęp do poszczególnych części w zależności od tego kim jest użytkownik. Możesz ograniczyć dostęp tylko do zalogowanych użytkowników albo nawet do użytkowników którzy pełnią określone role na stronie, są chociażby moderatorami lub administratorami. Możesz również ograniczyć dostęp dla poszczególnego konta dopóki nie zostanie aktywowane.

Ochrona funkcjonalności administratora

Nowe funkcjonalności administratora powinny być dostępne tylko dla zweryfikowanych użytkowników.

Możesz ukrywać link do tych funkcjonalności do momentu kiedy użytkownik się nie zaloguje ale jest to trochę sztuczka i stanie się trudna w utrzymaniu.

Zamiast tego powinienś napisać metodę `canActive()` która przekieruje anonimowego lub zalogowanego ale bez uprawnień admina użytkownika do strony głównej kiedy spróbuje przejść do funkcjonalności administratora. Sugeruję do tego celu stworzenie dedykowanej usługi

```
ng generate guard guard/auth
```

[guard/auth.guard.ts](#)

```

import { Injectable } from '@angular/core';

import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, Router } from
'@angular/router';

import { AuthService } from "../../shared/services/auth.service";

import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})

export class AuthGuard implements CanActivate {

  constructor(    public authService: AuthService,    public router: Router )

  {}

  canActivate( next: ActivatedRouteSnapshot, state: RouterStateSnapshot):
Observable<boolean> | Promise<boolean> | boolean {

    if(this.authService.isLoggedIn !== true) {

      this.router.navigate(['logowanie'])

    }

    return true;

  }
}

```

Lub tak:

```

  canActivate( next: ActivatedRouteSnapshot, state: RouterStateSnapshot):
Observable<boolean> {

    return this.authService.authState$.pipe(map(state =>

      if(state !== null) {

        return true;

      }

    ))
  }

```

```

    this.router.navigate(['logowanie']) ;

    return false;

  }

}

```

Podczas przejścia na widok inny niż login, sprawdzimy czy `authState` przechowuje stan użytkownika, jeśli tak, to go przepuścimy, natomiast jeśli wyemituje null, no to cofniemy go do widoku loginu

Guard **canActivate** zwraca strumień z true/false, w zależności czy **authState\$** wyemitował stan użytkownika (emisja true) czy wyemitował null (emisja false).

Strumień jest już obsługiwany przez mechanizm routingu w Angularze, także nie musimy się martwić o manualną subskrypcję w żadnym miejscu. Inym możliwym scenariuszem mogłoby być przekierowanie użytkownika np. na widok z informacją, że nie jest autoryzowany i prosimy o zalogowanie ael to już jest

Po implementacji usługi przyszedł czas na zastosowanie **canActivate** w routingu.

[app-routing.module.ts](#)

```

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

import { Routes, RouterModule } from '@angular/router';

// import komponentów
.....

// Import usługi canActivate guard
import { AuthGuard } from "../.. / guard/auth.guard";

// dodanie route guard do tablicy routing
const routes: Routes = [

  { path: '', redirectTo: '/logowanie', pathMatch: 'full'},

  { path: 'dashboard', component: DashboardComponent, canActivate: [AuthGuard] },

```

```
];
```

```
@NgModule({
```

```
  imports: [RouterModule.forRoot(routes)],
```

```
  exports: [RouterModule]
```

```
})
```

```
export class AppRoutingModule { }
```