Programowanie Aplikacji Webowych

laboratorium 12

Cel zajęć:

Celem laboratorium jest zaznajomienie z zagadnieniami bezpieczeństwa w aplikacjach webowych. Zapoznacie 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 (ścieżka łatwiejsza) lub samodzielnie go zaimplementujemy rozszerzając funkcjonalności serwera Webowego o funkcjonalności autentykacji i autoryzacji.

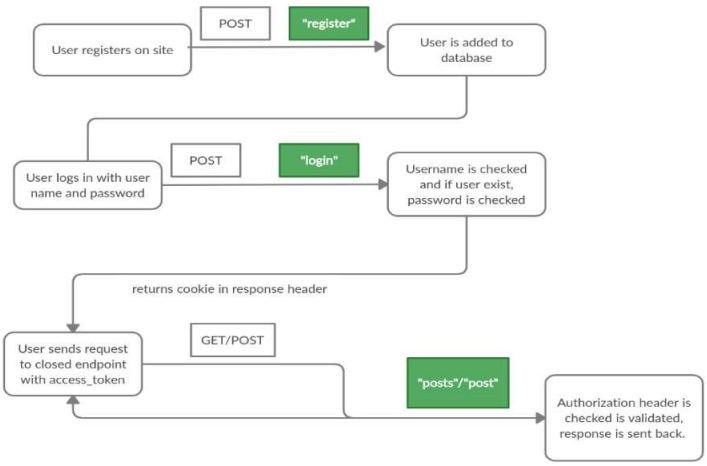
Ścieżka Łatwiejsza.

Sposób konfiguracji i instalacji odpowiednich modułów umożliwiających współpracę z Firebase z poziomu Angulara – omówiono szczegółowo na zajęciach lab nr 11 oraz materiałach wykładowych – wykład 12. Jako metodę Autentykacji wybieramy - logowanie za pomocą loginu i hasła.

Ścieżka trudniejsza.

W tym podejściu należy samodzielnie stworzyć zaimplementować fragment funkcjonalności odpowiedzialnej za autentykacje użytkowników a następnie autoryzacje poszczególnych requestów odbieranych przez serwer Webowy. Nowoczesne mechanizmy autentykacji I autoryzacji opierają sie na JWT token.

Ogólna zasada działania uwierzytelnianie w aplikacji internetowej oparta na JWT wygląda jak poniżej:



• Użytkownik wysyła zapytanie na endpoint dedykowany do Rejestracji nowego konta.

• controler odpowiedzialny za uwierzytelniania Node/Express generuje tokeny JWT po rejestracji lub logowaniu i odsyła je z powrotem do aplikacji frontendowej

• aplikacja frontendowa wykorzystuje pamięć lokalną do zapisu i trwałego przechowywania tokena JWT,

• po otrzymaniu tokena JWT aplikacja frontendowa weryfikuje tokeny JWT podczas renderowania chronionych widoków lub korzystania z chronionej akcji

• aplikacja fronetndowa dołącza do każdego zapytania wysyłanego do serwera token JWT (w sekcji nagłówkowej http) w celu uzyskiwania dostępu do chronionych tras/zasobów API.

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Zadanie 1. Zaimplementuj po stronie Frontendu funkcjonalność pozwalającą na samodzielną rejestracje 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 autentykacja 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 nazwie użytkownika zalogowanego.

Uwaga – musisz rozszerzyć model danych w aplikacji o nowy obiekt reprezentujący użytkownika. Zastanów się jakich danych potrzebujesz do opisu użytkownika i jego funkcjonalności. **(max ilość punktów 2)**

Sekcja łatwiejsza:

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.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Sekcja trudniejsza:

Zadanie 2. W przypadku wyboru ścieżki trudniejszej musisz samodzielnie zaimplementować funkcjonalności rejestracji i logowania (zarówno po stronie frontendu jak i backendu). Pamiętaj że oprócz rejestracji nowego konta w bazie danych (rejestracja) lub weryfikacji jego istnienia z poprawnymi danymi do autentykacji (e-mail, hasło) musisz również wygenerować i odesłać do aplikacji frontendowej token autoryzacyjny.

(max ilość punktów 4)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Warto wiedzieć, że możemy manipulować stanem uwierzytelnienia w różnych scenariuszach, takich jak np. zamknięcie karty i powrót (wylogować użytkownika automatycznie czy może nie?).

Przy użyciu Firbase 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 3. Zaimplementuj mechanizm wyboru tryby persystencji stanu logowania (niezależnie która ścieżkę wybrałeś (FireBase czy własny serwer). 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**)

W przypadku samodzielnej implementacji autentykacji – zaimplementuj działanie refresh tokena – zarówno po stronie serwera jak i aplikacji fromtendowej. Obsługa autentykacji przy wygasaniu tokena autoryzującego powinna z punktu widzenia użytkownika być niewidoczna. tzn. -> Gdy token autoryzujący wygaśnie (np. po 1 minucie) to przy kolejnym zapytaniu gdy z serwera dostaniesz komunikat o braku uprawnień za pomocą referesh tokena wygeneruj kolejny token autoryzujący i za jego pomocą dokończ tranzakcje. Finalnie użytkownik dostanie zasoby o które prosił, nie zdajac sobie sprawy z problemów jakie pojawiły się w trakcie**.**

Wykonaj test sprawdzający czy możesz automatycznie wylogować użytkownika w przypadku gdy spróbuje się zalogować np. w innej przeglądarce (tak aby nie było możliwe wiecej niż jedna sesja w danym momencie) **(max 2 pkt)**

Zadanie 4. Chcemy wiedzieć nie tylko, z kim mamy do czynienie ale również co może on wykonać w aplikacji a do czego nie ma uprawnień. Zrealizuj funkcjonalność serwera autoryzującego, który będzie zwracał w wersji minimalistycznej przynajmniej przypisane od użytkownika role.
Niech w naszej aplikacji istnieją dla użytkowników zalogowanych następujące role: Pacjent, Lekarz oraz Admin. Oczywiście oprócz użytkowników zalogowanych z aplikacji korzystać będą użytkownicy niezalogowani - goście.

Gość to użytkownik nie zarejestrowany który może tylko przeglądać listę dostępnych lekarzy, bez możliwości podglądu szczegółów ich harmonogramów. Co za tym idzie nie może się również rejestrować na konsultacje. Dostępne dla niego są opcje rejestracji i logowania.

Strona głowna	Lista lekarzy		Registry	Login

Pacjent (Rejestracja samodzielna w systemie) może przeglądać harmonogramy lekarzy oraz rezerwować konsultacje do wybranego lekarza zapisując je do swojego koszyka/(pełniącego role własnego harmonogramu). Może oceniać i zostawiać komentarze ale tylko dla lekarzy z których usług korzystał.

Strona głowna	Lista lekarzy	Harmonogramy lekarzy		Moj koszyk/zarezerowane wizyty	NickName	Logout

Lekarz (rejestracja lekarza może wykonać tylko admin, lekarz może tylko się logować i wylogować) może przeglądać swój harmonogram oraz go modyfikować – dodając, modyfikując lub usuwając pozycje z harmonogramu zgodnie z wytycznymi z lab 10. Nie może wystawiać ocen ale może odpowiadać na komentarze dotyczące niego.

Strona głowna	Mój harmonogram	Zarządzanie harmonogramem			Nazwa lekarza	Logout

Admin oprócz rejestracji lekarzy może dodatkowo przeglądać i zarządzać listę zarejestrowanych użytkowników. Ma możliwość banowania użytkownika. Banowanie oznacza ze użytkownik nie może zostawiać komentarzy ani oceniać lekarza. Sam admin nie może zapisywać się na konsultacje, dodawać komentarzy czy ocen, choć może usuwać dowolny komentarz (np. ze względu na wulgaryzmy) **(max ilość punktów 2)**

Zadanie5. Rozróżniamy już użytkowników. Wykorzystajmy to do realizacji funkcjonalności koszyka (listy zarezerwowanych wizyt), oraz do przechowywania historii zarezerwowanych wycieczek. Dodatkowo wykorzystaj ten fakt do udostępnienie funkcjonalności oceny konsultacji/lekarza lub dodania opinii tylko dla pacjentów, którzy byli pacjentami lekarza przy jednoczesnym zabezpieczeniu przed wielokrotnym głosowaniem. Pamiętaj aby koszyk tak zaimplementować aby nie był on współdzielony przez różnych użytkowników oraz trzymał on wartości miedzy sesjami.

(max ilość punktów 2)

Zadanie 6. Czasami lekarz chce wysłać natychmiastowe powiadomienia dla wszystkich pacjentów. Niech aktualnie zalogowani pacjenci od razu zobaczą zmiany wprowadzone przez lekarza za pomocą panelu zarzadzania harmonogramem. Nawet jeśli nie wykonują żadnych operacji wymagających załadowania nowej porcji danych.

(max ilość punktów 2)

Zadanie 7. Użytkownik może również przejść od razu na daną ścieżkę dowolnego widoku z pominięciem etapu logowania, oczywiście jeśli zna ścieżkę. Zabezpiecz dostęp do odpowiednich ekranów dedykowanych tylko dla lekarza, admina czy pacjenta. 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.)

W wersji łatwiejszej - Firebase proszę dołączyć do projektu plik zawierający opis reguł zabezpieczających (implementacje) po stronie Firebase.

(max ilość punktów 2)

Zadanie 11. Dopracuj aplikacje pod kątem wizualnym. Możesz korzystać z dowolnych bibliotek i komponentów, których według Ciebie są interesujące np. Boostrap, Material Design lub inne. Oceniać będę wrażenia estetyczne, ergonomie użycia itp.)

(max ilość punktów 1)

Zadanie 12. Oceniać będę również jakość kodu końcowego. Szczególnie mile widziane będzie używanie podejścia bazującego na obsłudze Observables wykorzystywanego do propagacji zamian w danych wewnątrz aplikacji. Staramy się minimalizować odpytania serwera do

sytuacji rzeczywiście niezbędnych, a nie traktujemy go jako mechanizm wymiany komunikatu pomiędzy komponentami. **(max ilość punktów 1)**

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. Proponuje 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 powinieneś napisać metodę *canActive()* która przekieruje anonimowego lub zalogowanego ale bez uprawnie admina użytkownika do strony głównej kiedy spróbuje przejść do funkcjonalności administratora. Sugeruje 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 { }
```