

# Programowanie Aplikacji Webowych

## laboratorium 12

### Cel zajęć:

Celem laboratorium jest zaznajomienie z zagadnieniami bezpieczeństwa w aplikacjach webowych. Zapoznanie 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.

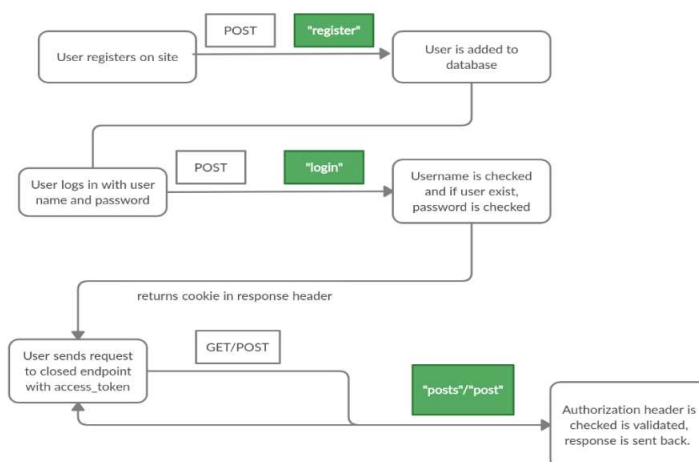
### **Ścieżka łatwiejsza.**

Sposób konfiguracji i instalacji odpowiednich modułów umożliwiających współpracę z Firebase z poziomu React – 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 autentykację użytkowników a następnie autoryzację poszczególnych requestów odbieranych przez serwer Webowy. Nowoczesne mechanizmy autentykacji i autoryzacji opierają się 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 frontendowa dołącza do każdego zapytania wysyłanego do serwera token JWT ( w sekcji nagłówkowej http) w celu uzyskiwania dostępu do chronionych tras/zasobów API.

**Zadanie 1.** Zaimplementuj po stronie Frontendu funkcjonalność pozwalającą na samodzielną 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.

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 Firebase/Auth moduł dostarczający funkcjonalności autentykacji. Główne metody to: `currentUser()`, `signInWithEmailAndPassword(email,password)`, `createUserWithEmailAndPassword(email, password)`, `signOut()`.

```
import { getAuth, onAuthStateChanged } from "firebase/auth";

const auth = getAuth();

onAuthStateChanged(auth, (user) => {
```

```
if (user) {      // User jest zalogowany

    const uid = user.uid;

    // ...

} else {

    // User został wylogowany

    // ...

}

});

}
```

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

[illegible]

### 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 5)**

[illegible]

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 frontendowej. 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 refresh tokena wygeneruj kolejny token autoryzujący i za jego pomocą dokończ transakcję. Finalnie użytkownik dostanie zasoby o które prosił, nie zdając 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 więcej niż jedna sesja w danym momencie) (max 2 pkt)

**Zadanie 4.** Chcemy wiedzieć nie tylko, z kim mamy do czynienia 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 główna	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 główna	Lista lekarzy	Harmonogramy lekarzy		Moj koszyk/zarezerwowane 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 główna	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 że 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). 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 między 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 zarządzania 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. 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 3)**

**Zadanie 11.** 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 4)**

**Zadanie 12.** Oceniać będę również jakość kodu końcowego. Szczególnie mile widziane będzie używanie podejścia bazującego na obsłudze Observables wykorzystywanego do propagacji zmian 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 6)**