Programowanie Aplikacji Webowych

laboratorium 10

Cel zajęć:

Celem laboratorium jest przećwiczenie zagadnień związanych z obsługą routingu po stronie Frontendu.

Routing

Angular pozwala nam na przechodzenie z jednego widoku do drugiego w miarę jak użytkownik podejmuje odpowiednie działania na stronie. Przeglądarka jest takim modelem w którym możemy nawigować. Po wpisaniu jakiegoś adresu możemy na niego przejść, po wybraniu jakiegoś odnośnika również. Router Angulara korzysta z tej koncepcji. Może traktować konkretny adres jako polecenie przejścia do konkretnego widoku. Może przekazywać do niego opcjonalnie różne parametry w zależności od naszych potrzeb które pomogą w tym widoku określić co konkretnie użytkownik chce wyświetlić. Możemy podstawić router do linków na naszej stronie tak, żeby kliknięcie w nie powodowało przeniesienie nas do konkretnego widoku.

Routing pozwala zawrzeć pewne aspekty stanu aplikacji w adresie URL. Dla aplikacji front-end jest to opcjonalne - możemy zbudować pełną aplikację bez zmiany adresu URL. Dodanie routingu pozwala jednak użytkownikowi przejść od razu do pewnych funkcji aplikacji. Dzięki temu aplikacja jest łatwiej przenośna i dostępna dla zakładek oraz umożliwi użytkownikom dzielenie się linkami z innymi.

Routing ułatwia:

- Utrzymanie stanu aplikacji
- Wdrażanie aplikacji modułowych
- Stosowanie ról w aplikacji (niektóre role mają dostęp do określonych adresów URL)

Konfiguracja Routingu sprowadza się do:

1. Importu odpowiednich modułów `import { RouterModule, Routes } from '@angular/router';`

Router jest opcjonalnym serwisem który pokazuje określony widok dla zdefiniowanego adresu. Nie jest częścią biblioteki *core* Angulara.

2. *Konfiguracja*

Aplikacje wykorzystujące routing mają jedną instancję serwisu routera. Nie ma on jednak określonych adresów, czy też ścieżek, po których powinien nawigować. Musimy mu je więc skonfigurować jeśli chcemy żeby cokolwiek robił. Dokonujemy tego w pliku *app.module.ts*.

Przykładowy wpis wyglądałby tak:

```
const appRoutes: Routes = [
  { path: 'glowna', component: AComponent },
  { path: 'test/:id',      component: BComponent },
  { path: 'testy',   component: CComponent,   data: { key: 'ABC' }   },
  { path: '',    redirectTo: '/testy',     pathMatch: 'full'    },
  { path: '**', component: PageNotFoundComponent }
```

Tablica *appRoutes* określa do jakich komponentów nawigować po określonych ścieżkach. Przekazujemy ją do metody *RouterModule.forRoot()* żeby router ją zaimportował i wprowadził w życie.

Ścieżki które prowadzą do określonych komponentów nie są poprzedzone znakiem /. Router buduje ostateczne ścieżki pozwalając odnosić się do relatywnych i konkretnie określonych ścieżek kiedy nawigujesz pomiędzy widokami swojej aplikacji.

Paramter *:id* w drugiej ścieżce pozwala na podstawienie w to miejsce jakieś wartości i późniejsze odniesienie się do niej właśnie przez *id*. Kiedy wpiszesz */test/42* komponent *BComponent* będzie wiedział, że dostał *id* o wartości 42.

Właściwość *data* w trzeciej ścieżce zawiera w sobie jakieś specyficzne dane przypisane dla konkretnej ścieżki. Również będą one dostępne dla widoku do którego nawiguje. Powinno się jej używać do takich rzeczy jak tytuły podstron, tekstów do nawigacji czy innych statycznych danych, tylko do odczytu.

Czwarta, pusta ścieżka prezentuje domyślną lokalizację aplikacji do której użytkownik powinien zostać przekierowany zaraz po jej odpaleniu.

Ostatnia ścieżka gdzie znajduje się podwójny znak * zostanie użyta kiedy wprowadzony przez użytkownika adres nie będzie pasował do żadnej podanej do tej pory ścieżki w naszym routerze. Jest to więc najzwyczajniej odpowiednik przekierowania do strony informującej o błędzie 404.

Kolejność podania tych ścieżek ma znaczenie. Router użyje pierwszej pasującej ścieżki żeby odnieść się do podanego mu adresu. Dlatego też te bardziej precyzyjne, skomplikowane ścieżki powinny być umieszczone poniżej tych bardziej ogólnych.

3. Określenie miejsca umieszczenia Router outlet

Kiedy przekażesz taką konfigurację do routera w swojej aplikacji i wpiszesz URL */testy* wyświetlony zostanie *CComponent* w miejscu po tagu *<router-outlet>* który należy umieścisz w swojej aplikacji. Zazwyczaj wygląda to tak, że umieszcza się go w *app.component.html* gdyż jest to główny jej komponent.

4. Definiowanie połączeń między trasami

Masz już skonfigurowane i użyte ścieżki do których router ma się odnosić. Musisz jednak dodać do swojej strony jeszcze możliwość nawigowania po nich. Oczywiście ktoś może przechodzić na konkretne podstrony poprzez wpisywanie konkretnego adresu w przeglądarce ale raczej dużo bardziej normalnym rozwiązaniem z którego korzysta większość użytkowników będzie klikanie w odpowiednie odnośniki na stronie.

Dodaj linki do tras za pomocą dyrektywy RouterLink .
Na przykład poniższy kod definiuje łącze do trasy w ścieżce component-one .

```
<a routerLink="/component-pierwszy">PierwszyKomponent</a>
```

Alternatywnie możesz nawigować do trasy, wywołując funkcję navigate na routerze:

```
this.router.navigate(['/ component-pierwszy]);
```

Przykład:

```
<h1>Angular Router</h1>
 <nav>
  <a routerLink="/glowna" routerLinkActive="active">Strona Głowna </a>
  <a routerLink="/testy" routerLinkActive="active">Testy</a>
 </nav>
 <router-outlet></router-outlet>
```

Jeśli chcesz żeby twoje linki były bardziej dynamiczne powinieneś tak jak powyżej pokazałem zastosować routerLinkActive który pozwoli na nadanie klasy active i wizualne wyróżnienie aktywnych ścieżek.

Stan

Po każdym udanym cyklu nawigacji Angular buduje drzewko obiektów ActivatedRoute które określają obecny stan routera. Możesz odwołać się do obecnego RouterState, czyli właśnie tego stanu, z dowolnego miejsca w twojej aplikacji używając serwisu Router i jego właściwości: routerState. Każdy obiekt ActivatedRoute w RouterState dostarcza metody, żeby odnieść się do nadrzędnego elementu, elementów potomnych czy sąsiadujących w hierarchii nawigacji.

ActivatedRoute

Dzięki temu serwisowi możemy odnieść się do wielu, czasem bardzo przydatnych informacji odnośnie lokalizacji w której obecnie się znajdujesz. Są to między innymi url który jest tablicą składającą się z poszczególnych członów ścieżki w której się znajdujesz, data czyli obiekt z dodatkowymi danymi dostarczony do ścieżki, paramMap – mapa pozwalającą na odwoływanie się do parametrów dostarczonych do ścieżki. parent to oczywiście element ActivatedRoute będący rodzicem, firstChild z kolei zawiera pierwszy element potomny a children je wszystkie.

Zadanie 1. Pod adresem https://jsonplaceholder.typicode.com/ znajduje się fejkowy serwer aplikacyjny udostępniający swoje dane za pomocą REST API. Zapoznaj się z jego zawartością a następnie napisz aplikację frontonową, która pobierze i wyświetli zawartość dwóch endpointów serwera /posts oraz /photos. Do tego celu wykorzystaj httpClient. Dla posts dodaj również możliwość wysyłania posta na serwer. Niech zawartość każdego z endpointów będzie wyświetlana na oddzielnej podstronie. W tym celu użyj mechanizmu Routingu. Przygotuj menu nawigacyjne zawierające 3 pozycje: Home (strona głowna z informacjami ogólnymi, Posty (zawierająca kolekcje pobranych postów) oraz Zdjecia (wyświetlająca informacje o pobranych zdjęciach).

Dla zdjęcia dodaj możliwość wyświetlania wybranego zdjęcia na oddzielnym widoku. Zastanów się jak powinien wyglądać adres takiego widoku.

Niech menu będzie niezmienne na każdej z podstron (wyświetlane na samej górze ekranu).

==========

W drugiej części lab rozpoczniemy prace na końcowym mini-projektem rozwijanym przez najbliższe 3 zajęcia laboratoryjne. Tematem rozwijanej aplikacji będzie **aplikacja do zdalnych konsultacji lekarskich.**

Aplikacja będzie miała dwa typy użytkowników: lekarza oraz pacjentów. Lekarz będzie miał możliwość definiowania harmonogramu planowanych terminów konsultacji on-line. Możliwe jest stworzenie planu cyklicznych (powtarzalnych slotów czasowych) lub planów jednorazowych. W oparciu o upublicznione terminy konsultacji pacjenci samodzielnie rezerwują sobie dogodne dla nich terminy konsultacji. Harmonogramy konsultacji mają być wyświetlane w postaci macierzowej z możliwością zarówno wertykalnego (sprawdzanie całego dnia) jak i horyzontalnego (sprawdzanie poszczególnych dni tygodnia) przeglądania planu lekarza. Pacjent po wyborze interesujących terminów konsultacji przenoszony jest do ekranu finalizującego etap rezerwacji konsultacji (symulacja płatności oraz potwierdzania wyboru).

Dzisiejsze lab to rozpoczęcie prac nad funkcjonalnością frontendową. Jako zewnętrzne źródło danych proszę używać lokalnego pliku JSON. W kolejnym lab – lab 11 przejdziemy do implementacji części backendowej. Kolejne lab - lab 12 to implementacji mechanizmów autentykacji i autoryzacji. Backend i autentykacja oparte będą na wykorzystaniu Firebase lub poprzez samodzielnie implementowanym serwerze.

Rozwój aplikacji będzie sterowany kolejnymi zadaniami, których celem jest jej ewolucyjny rozwój.

Zadanie 1. Kalendarz lekarza (4 pkt)

Prace rozpoczniemy od stworzenia komponentów pozwalających na wyświetlania i przeglądanie kalendarza dostępności lekarza. Wzorować się tutaj można na kalendarzach Googla lub temu podobnych. Domyślnym widokiem kalendarza jest widok tygodniowy. Pojedynczy slot czasowy niech będzie 0.5 h – oznacza to ze minimalny czas na konsultacje wynosi 0.5 h. Oczywiście będzie możliwe rezerwowania większej ilości slotów czasowych przez pacjenta. W widoku głównym jest możliwości przeglądania kalendarza zarówno w przyszłości jak i przeszłość. Nagłówek każdej kolumny wyświetla informacje o dniu tygodnia, dacie oraz ilości konsultacji zarezerwowanych danego dnia. Nagłówek wiersza wyświetla inflacje o godzinie. Oczywistym jest ze nie możliwym od razu wyświetlić 24h – niech widok domyślnie wyświetla harmonogram konsultacji dla max 6h. Słoty czasowe zarezerwowane przez klientów na harmonogramie głównym wyświetlane są w postaci bloków oznaczonych kolorami. Kolor oznaczać będzie typ konsultacji. Po najechaniu na dany blok wyświetla się informacja o szczegółach wizyty. Wizyty które już się odbyły wyświetlane są jako wyszarzane. Aktualny dzień (cała kolumna) powinna być w jakiś sposób wyróżniona. Podobnie aktualny czas – powinien być symulowany przez znacznik wskazujący w którym slocie czasowym aktualnie jesteśmy.

Zadanie 2. Definiowanie dostępności (2pkt)

Daj opcje definiowania przez lekarza swoje dostępności. Możliwe jest definiowania cykliczne – gdzie podawany jest przedział czasowy (od kiedy do kiedy i w jaki godzinach konsultuje). Lub jednorazowe dla konkretnego dnia. Przykład dla definiowania cyklicznego - definiuje od 1.02.2025 do 12. 03. 2025 . Potem maske dni konsultacji np. (poniedziałki, wtorki, czwartki i soboty) a następnie czasy konsultacji (i tutaj podaje się maske konsulatcji np. od 8 do 12.30 i od 16 od 21.30.

Zadanie 3. Dodawanie absencji (3 pkt)

Lekarz może również zdefiniować terminy nieobecności (np. terminy planowych wakacji lub inne planowane nieobecności). Nieobecności są całodzienne, Proszę o uwzględnienie podanych planowanych absencji do identyfikacji konfliktów z już zaplanowanymi konsultacjami lub jako element walidacyjny z zadaniu 2 (przy definiowaniu dostępności). Dni nieobecności powinny być zaznaczane na czerwono w widoku głównym harmonogramu pacjenta. W przypadku identyfikacji konfliktu z zarezerwowanymi konsultacjami zmienić status konsultacji na odwołane.

Zadanie 4. Rezerwacja konsultacji (2 pkt)

W aplikacji istnieje możliwość zarezerwowania konsultacji. W tym celu po wybraniu opcji rezerwacja konsultacji na ekranie harmonogramu zaznaczmy wolny slot czasowy. Powinien wyświetlić się formularz do definiowania szczegółów konsultacji. Potrzebne dane:

- Długość konsultacji (pamiętaj ze wolne sloty czasowe musza być bezpośrednio obok siebie), w przypadku detekcji konfliktu z kolejną konsultacją informacja o problemie oraz brak możliwości rezerwacji konsultacji.

- typ konsultacji (np. pierwsza wizyta, wizyta kontrolna, choroba przewlekła, recepta itp.).

- imie i nazwisko pacjenta

- płeć pacjenta

- wiek pacjenta

- informacje dla lekarza

Zadanie 5 Koszyk (1 pkt)

Komponent wyświetlający listę zarezerwowanych konsultacji. Zawiera funkcjonalność symulującą proces płatności.

Zadanie 6. Odwołanie rezerwacji konsultacji (1pkt)

Powinna być możliwość odwołania rezerwacji. Po zaznaczeniu wybranej konsultacji na harmonogramie powinna być możliwość odwołania konsultacji. W jej wyniku zreferowane jako konsultacje sloty czasowe staja się ponownie wolne.