Programowanie Aplikacji Webowych

laboratorium 9

Cel zajęć:

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

Tematyka lab dotyczyć będzie jednego tematu przewodniego rozwijanego przez najbliższe 3 lab. Tematem rozwijanej aplikacji będzie **biuro turystyczne**. Użytkownicy będą mogli rezerwować udział w wycieczkach dostępnych na stronie biura. Celem aplikacji jest możliwość przeglądnięcia oferty biura dań (pod kątem różnego rodzaju kryteriów) z możliwością oceny wybranej wycieczki oraz zastawienia komentarza na temat wycieczki. Dodatkowo będzie możliwość przeglądania zawartości oraz zakupu wycieczki jako osoba zalogowana – realizacja rejestracji i logowania. Tylko admin będzie miał możliwość dodawania, usunięcia i edycji wycieczki. Backend i autentykacja oparte na Firebase lub samodzielnie implementowanym serwerze.

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 (a nie FetchAPI znane Ci z wcześniejszych zajęć). 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).

(2 punkty)

---------- Biuro Turystyczne ----------

Zadanie 2. Lista wycieczek (5 pkt)

Stwórz nowy projekt a w nim nowy komponent/komponenty reprezentujące Wycieczki. Zmodyfikuj tak kod aby nowy komponent był komponentem wyświetlanym na starcie naszej aplikacji. Komponent Wycieczki powinien wyświetlać listę wycieczek. Lista wycieczek powinna być zdefiniowana i wczytywana za pomocą lokalnego zewnętrznym pliku json. **(1 pkt)**

Pojedynczy obiekt Wycieczki powinien zawierać następujące pola: Nazwa, docelowy kraj wycieczki, data rozpoczęcia i zakończenia wycieczki, cena jednostkowa, max ilość miejsc, krótki opis wycieczki oraz link do poglądowego zdjęcia.

Stwórz przynajmniej 10 obiektów i użyj ich w komponencie.

Wyświetl zawartość tablicy obiektów w szablonie komponentu głównego - dyrektywa *ngFor (każda element odpowiednio wystyluj). Wyświetlane zdjęcia wycieczki proszę wyświetlać jako okrągłe. Przy każdym produkcie powinny znajdować się 2 przyciski + i - pozwalające na rezerwacje miejsca na wycieczkę lub rezygnacje z wycieczki (przycisk -).

Jeśli ilość wolnych miejsc wycieczki znajdującej się w tablicy będzie wynosiła 0 to należy wyświetlić inny komunikat niż gdy ilość dostępnych miejsc jest większa od 0.

Podobnie przy rezygnacji – jeśli ilość dostępnych wycieczek jest równa max ilości to nie powinno być możliwości zwrócenia wycieczki. **(1pkt)**

W przypadku gdy ilość miejsc spadnie do zera przycisk + powinien zostać ukryty. Nie chcemy przecież rezerwować wycieczki na której już nie ma miejsca. – dyrektywy ngStyle lub ngClass. Gdy ilość wolnych miejsc będzie zbliżała się do 0 (np. od 3 w dół) należy zaznaczyć to w sposób graficzny np. inne tło, kolor czcionki, wielkość fontów lub inny wizualny sposób. **(1pkt)**

Podobnie należy rozróżnić wycieczki o najniższej cenie jednostkowej oraz najwyższej – za pomocą dodatkowego obramowania obejmującego dany produkt - zielone – najdroższy, czerwone najtańszy.

Wypisz nazwę wycieczki oraz kraj dużymi literami -> skorzystaj z odpowiedniego typu pipe. Wyświetl cenę wycieczki wraz z nazwa (lub znakiem płatniczym) skojarzonym z walutą np. USD - $, euro lub złotówki. Zaimplementuj możliwości przewalutowania cen wycieczek. **(1pkt)**

Wyświetl również sumaryczną ilość aktualnie zarezerwowanych wycieczek - jeśli wynosi on więcej niż 10 ma być wyświetlana na zielonym tle, jeśli poniżej 10 na czerwonym tle. **(1pkt)**

Uwaga!! Oceniać będę również zaproponowaną architekturę rozwiązania. Powinna być zwinna i elastyczna – pamiętajmy o zasadzie SOLID.

Zadanie 3. Usuwanie wycieczki (1pkt)

Rozszerz funkcjonalność aplikacji o możliwość usuwania wycieczki. Zrealizuj tą funkcjonalność poprzez dołożenie przycisku Usun obok wycieczki. Naciśniecie tego przycisku powinno skutkować usunięciem wycieczki z listy wycieczek.

Zadanie 4. Dodawanie wycieczki (2pkt)

Skoro jest możliwość usuwanie wycieczki z listy, niech będzie także dostępna możliwość dodawania nowej wycieczki. Dodawania odbywa się za pomocą formularza – sugeruje zastosowanie formularza typu Model Driven Forms. Na razie podobnie jak z usuwaniem wycieczki dostęp do tej funkcjonalności będą mieli wszyscy użytkownicy – potem tylko z odpowiednimi uprawnieniami. Zastosuj mechanizm walidacji.

Zadanie 5. Ocena wycieczki (1pkt)

Rozszerzmy funkcjonalność pojedynczej wycieczki o możliwość oceniania atrakcyjności wycieczki przez klienta (np. wybór ilość gwizdek lub jakaś inna interesująca forma oceniania). Na razie oceniać wycieczkę będzie mógł każdy klient. Później po wprowadzeniu autoryzacji tylko osoba która kupiła i zrealizowała wycieczkę. Docelowo funkcjonalność ta będzie dostępna tylko z poziomu karty poszczególnej wycieczki. Preferowane samodzielna realizacja oceny bez korzystania z gotowych bibliotek.

Zastanów się w jaki sposób zrealizujesz ocenę (oddzielny komponent? a może tylko atrybut komponentu Wycieczki?)

Zadanie 6. – filtrowanie listy wycieczek

Tworzymy dodatkowy komponent służący do filtrowania wyświetlanych wycieczek. Kryteriami po których możemy filtrować są: lokalizacja wycieczki, cena (zakres), data (zakres), ocena. Proponuje do realizacji tej funkcjonalności zastosowanie samodzielnie zdefiniowanych potoków. Sposób realizacji opisany w sekcji poniżej **(1 pkt)**

Wersja rozszerzona

Możliwość wyboru kilku wartości dla danego kryterium np. wybór kilku lokalizacji z listy dostępnych, lub oceny 4 i 5 gwiazdek.

Kryteria filtrowania można łączyć tzw. przykładowe kryteria filtrowania: interesują mnie wycieczki o ocenie 3 i 4 gwiazdki odbywające się w takim a taki terminie. Wyniki filtrowania powinny być dostępne online już podczas wyboru wartości w filtrze. **(1 pkt)**

Filtry zwierają tylko wartości dostępne w liście wycieczek - dotyczy np. zakresu cen. Nie od 0 do 10000 tylko od dostępna cena minimalna do cena maksymalna – wartości powiązane z aktualnymi wynikami filtrowania. **(1 pkt)**

Zadanie 7. Komponent do wyświetlania sumarycznej wartości rezerwacji (2 pkt)

Stwórz nowy komponent, niepowiązany z pozostałymi zawierający informacje o wybranych wycieczkach tzn. ich ilości oraz sumie całego zamówienia. Jej zawartość będzie powiązana z lista wycieczek. Jeśli dodaje wycieczkę wartości w widgecie rosną, jeśli usuwam maleją. Widget będzie umieszczany w menu, które będzie ciągle dostępne i widzialne przez użytkownika.

Zadanie 8. Koszyk rezerwacji. (2 pkt)

Stwórz komponent/komponenty do wyświetlania szczegółów zarezerwowanych wycieczek. Jest to tzw. Koszyk. W koszyku wyświetlane są lista zarezerwowanych wycieczek wraz z możliwością zwrotu lub rezerwacji dodatkowych miejsc na wycieczce. Obok każdej wycieczki jest wartość zarezerwowanej wycieczki oraz checkbox służący do wyboru wycieczki do zakupu, a na dole sumaryczna wartość wszystkich wycieczek. Dodatkowo mamy przycisk kup wycieczki, którego użycie oznacza ze wszystkie zaznaczone w koszyku wycieczki (domyślnie wszystkie są zaznaczone) będą kupione. Kupno oznacza ze wycieczka jest usuwana z koszyka oraz nie można jej już zwrócić a pojawi się w historii zakupów.

Zadanie 9. Historia zakupów - lista zakupionych wycieczek (rozszerzenie) (2 pkt)

Wybrane wycieczki znajdują się w koszyku. Z jego widoku możliwy jest zakup wycieczki. Przy każdej wycieczce powinien być możliwy zakup wycieczki, który zmiana stan wycieczki na kupiona. Wycieczka znika z koszyka i zostaje dodana do widoku historia zakupów. Prezentuje się tam lista zakupionych wycieczek wraz z info, kiedy kupiona oraz danymi dotyczącymi wycieczki (cena, data startu i zakończenia, lokalizacja, ilości biletów). Każda wycieczka ma status – przed (oczekiwania na rozpoczęcie), w trakcie (aktywna), zakończona (archiwalna).

System sam przypomina wyświetlając w sekcji menu – info (odpowiednia ikona) o zbliżającym się starcie wycieczki. (1pkt)

Widok pozwala na filtrowanie wycieczek po statusie.

Zadanie 10. Biuro Turystyczne – uporządkowanie widoków. (2 pkt)

Wykorzystując nabytą w zadaniu 1 umiejętność zarządzania nawigacja po stronie – uporządkuj zawartość poszczególnych ekranów aplikacji. Niech będą jako niezależne ekrany wyświetlające:

- **widok startowy naszego biura**,
- widok z ofertą wycieczek (zawierający listę wycieczek filtrami),
- dodawanie nowej wycieczki,
- podgląd zawartości koszyka
- oraz widok z historia zakupionych i odbytych wycieczek.

Oczywiście lista nawigacyjne zrealizowana jako menu responsywne widziane na każdym widoku tak aby w łatwy sposób nawigować po stronie. Skonfiguruj moduł Routingu tak aby możliwe było przemieszczanie się pomiędzy poszczególnymi widokami. Oprócz listy nawigacyjnej wyświetl także nagłówek zawierający dane o koszyku (wykorzystaj tutaj widget z zadania 7 -> ilość i suma wybranych wycieczek), symbol powiadomienia o zbliżajcie się wycieczce oraz w przyszłości dane o użytkowniku

Widok startowy biuraTurytycznego powinien zawierać przygotowaną „artystycznie" wizytówkę naszego Biura + dane kontaktowe + mapa z lokalizacją. **(1 pkt)**

Extra **(1pkt)** za ocenę estetyczna rozwiązania.

---------- Potoki własne – implementacja przykładowa ----------

Angular pozwala tworzyć własne potoki . Wymagane jest aby:

- użyć dekorator @Pipe z metadanymi potoku, wśród których jest własność name. Ta wartość zostanie wykorzystana do wywołania potoku
- Implementować metodę transformacji interfejsu PipeTransform. Ta metoda pobiera z potoku wartość oraz zmienną liczbę argumentów dowolnego typu i zwracają wartość przekształconą (piped).

```
// Przykładowa implementacja potoku typu wyszukaj po podanym tekście
@Pipe({ name: 'searchPipe' })
export class SearchPipe implements PipeTransform {
    transform(courses: Course[], searchText: string): Course[] {
        if (!courses)
```

```
            return [];
        if (!searchText)
            return courses;
        searchText = searchText.toLowerCase();
          return courses.filter(course => {
              return course.name.toLowerCase().includes(searchText);
          });
    }
```

Użycie zdefiniowanego potoku w szablonie komponentu. Każdy parametr rozdzielany dwukropkami w szablonie odwzorowuje jeden argument metody w tej samej kolejności

```
<div>
<div *ngFor="let p of (getCourses() | searchPipe : search )">
        ……….
</div>
</div>
```