ZPW lab

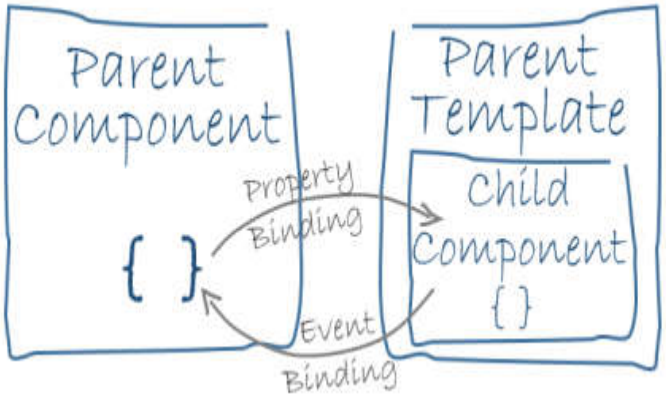
laboratorium 5

Angular - komunikacja miedzy komponentami

Tematem przewodnim laboratorium jest komunikacja pomiędzy komponentami. Wyróżniamy 3 różne przypadki:

- komunikacje rodzic –dziecko

- komunikacje dziecko – rodzic



- komunikacje pomiędzy niepowiązanymi komponentami (realizowane za pomocą serwisów).

Jednym z zadań realizowanych w ramach laboratorium 4 było stworzenie komponentu Produkty wyświetlającego liste produktów – zadanie nr 6. Dzisiejsze laboratorium rozszerzy możliwości aplikacji prezentującej listę produktów.

Zadanie 1. Nowy komponent typu dziecko – komunikacja rodzic -> dziecko

Podzielmy funkcjonalność komponentu Produkty na dwa niezależne komponenty: Produkty – zarządzające cała lista produktów oraz Produkt - wyświetlające tylko pojedynczy produkt.

Oznacza to ze w szablon komponenty **produkty.component.html** zawierałby mnie więcej taki fragment kodu:

```
<div class="card" >
  <div class="card-body">
    <h4 class="card-title">Products</h4>
```

```
    <div class="row">
      <app-product class="col-md-3" [product]="p" *ngFor="let p of products"></app-product>
    </div>
  </div>
</div>
```

gdzie <app-product> jest znacznikiem nowo stworzonego komponentu Produkt będącego komponentem typu child w stosunku do komponentu Produkty.

Zastosuj adnotacje @Input w komponencie Produkt w celu wstrzyknięcia do niego danych o konkretnym produkcie.

Zadanie 2. Komunikacja z rodzicem

Rozszerz funkcjonalność komponenty Produkt o możliwość usuwania produktu. Zrealizuj tą funkcjonalność poprzez dołożenie przycisku Usun obok produktu. Naciśniecie tego przycisku powinno usunąć danych produkt z listy produktów.

Przypominam, ze lista produktów znajduje się w komponencie Produkty, gdzie musi być zaktualizowana. Zastosuj adnotacje @Output w celu poinformowania komponenty typu rodzic o usunięciu produktu.

Zadanie 3. Refaktoryzacja modelu danych

Nasza aplikacja wymaga kolejnej refaktoryzacji. Zła praktyką projektową jest trzymanie danych bezpośrednio w komponencie. Stwórz wiec plik zawierający definicje modelu danych – sugeruje realizacje tego zadania za pomocą interfesju zamiast klasy. Następnie stwórz plik o nazwie FakeProdukts - imitujący źródło danych (zawierającego tablice przykładowych produktów).

W klasie komponentu Produkty zostawiamy publiczne pole produkty jako pustą tablicę. Dane z produktami będziemy wczytywać poprzez z usługę więc domyślnie pole produkty powinno być niezainicjowane.

Zadanie 4. Tworzenie/wykorzystanie naszej pierwszej usługi

Komponent Produkty nie zawiera już danych. Musimy mu je jakoś dostarczyć. Proponuje stworzenie usługi o nazwie ProduktSerwis której zadaniem będzie obsługa danych. Usługa ma mieć 4 metody: getProducts(), getProduct(), addProduct() oraz deleteProduct(), które realizują funkcjonalność jak w nazwie metody. Usługę tą należy wstrzyknąć do komponenty Produkty (na początku proponuje do tego użycie konstruktora zawierającego prywatne pole będące obiektem naszej usługi:

```
constructor(private productsService: ProduktSerwis) {
this.produkty = this. productsService. getProducts(); }
```

Do dekoratora @Component() komponentu Produkty należy dodać właściwość:
`providers: [ProduktSerwis]`, która pozwoli na wstrzyknięcie usługi do tego komponentu.

Zadanie 5 – Drobne ulepszenia związane z wykorzystaniem naszej usługi - ngOnInit
Poprzednie zadanie działa ale nie jest to właściwy sposób wstrzykiwania usługi do komponentu , nie powinniśmy w konstruktorze dodawać logiki, tym bardziej odczytującej dane z serwera. Robimy ulepszenie – dodajemy wstawkę programową, która pobierze dane po inicjalizacji komponentu.
1) W komponencie Produkty dodajemy metodę getProducts(), w której umieszczamy linijkę:
`this.produkty = this. productsService.` getProducts`();`
wyciętą z konstruktora. Metoda nic nie zwraca czyli piszemy, że zwraca void.
2) Do komponentu Produkty importujemy interfejs OnInit z @angular/core. Interfejs ten ma jedną metodę ngOnInit(), która jest uruchamiana, kiedy komponent jest inicjowany. Klasę komponentu oznaczamy jako implementującą interfejs OnInit.
3) W metodzie ngOnInit() umieszczamy kod wywołujący utworzoną w punkcie pierwszym metodę getProducts().

Zadanie 6 – dodatkowy komponent - dodawania nowego Produktu
Tworzymy dodatkowy komponent NewProdukt, który będzie pozwalał na dodawanie nowego produktu do listy produktów.

Zadanie 7 – Tworzymy nowy komponent Koszyk

Stwórz nowy komponent, niepowiązany z pozostałymi zawierający informacje o wybranych produktach, ich ilości oraz sumie całego zamówienia.

Zadanie 8. Zainstaluj boostrap

Bawiąc się Boostrap popracuj nad aktualnym wyglądem aplikacji.