SYNERGY CODES
SOFTWARE HOUSE

Wstęp do Angulara

O sobie

Adam Kowalski

- Absolwent informatyki Uniwersytetu Wrocławskiego
- Full-stack Developer w firmie Synergy Codes

adam.kowalski@synergycodes.com

https://www.synergycodes.com

Plan wykładu

Angular – czym jest i do czego służy ?

Typescript – przypomnienie najważniejszych informacji

- typy, klasy, interfejsy, funkcje, moduły
- Dekoratory

Środowisko pracy z Angularem

- template aplikacji
- komponenty (definicja, data bindings, komunikacja, cykl życia)
- serwisy
- routing
- formularze (Template driven, Reactive Forms)

Framework Angular

Otwarty framework i platforma do tworzenia SPA stworzona i wspierana przez Google. Pierwsza wersja Angulara (wersja 2.0) została wydana w maju 2016 roku (aktualna stabilna wersja 7.0, październik 2018)

Najważniejsze cechy frameworka Angular:

- Napisany w Typescript,
- Modularność
- Zorientowanie na komponenty (lepsze odseparowanie warstwy logiki od warstwy prezentacji danych)
- Własne dyrektywy (elementy dodawane do HTML-owych tagów)
- Możliwość tworzenia własnych HTML-owych tagów
- Nie powoduje przeładowań strony
- Modyfikuje strukturę dokumentu DOM „na żywo”

Angular vs AngularJS

AngularJS nie należy mylić z Angularem

Początkowo Angular 2.0 miał być wydany jako kolejna wersja frameworka AngularJS, ale zbyt duże różnice spowodowały, że Google zdecydował się wydać go jako osobny byt (brak kompatybilności wstecznej, brak prostej ścieżki aktualizacji aplikacji napisanych w AngularJS do Angular 2)

Najważniejsze zmiany wprowadzone w Angular 2.0:

- Renderowanie po stronie serwera
- Przepisanie z JS-a do Typescripta
- Lepsze wsparcie dla przeglądarek mobilnych
- Narzędzia konsolowe ułatwiające rozwijanie aplikacji w Angularze

W ramach ciekawostki -> https://angularjs.org/

Typescript

https://www.typescriptlang.org/

Język programowania stworzony przez Microsoft jako nadzbiór języka Javascript.

Cechy Typescript:

- Otwarty kod źródłowy (licencja Open Source)
- Wprowadzenie typów umożliwia statyczną weryfikację kodu
- Możliwość programowania obiektowego zorientowanego na klasach
- Typescript kompiluje się do Javascript
- Wsparcie wielu narzędzi programistycznych

Typescript

Typescript udostępnia programiście elementy składni z nadchodzących edycji ECMAScript jak i dodatkowe możliwości:

- Typowane zmienne, argumenty funkcji
- Zasięg zmiennych (var, const, let)
- Klasy
- Interfejsy
- Moduły
- Enum
- Opcjonalne parametry funkcji
- Dekoratory
- Array functions

Środowisko pracy z Angularem

- Node.js
  - https://nodejs.org/en
- NPM
  - https://www.npmjs.com
- Visual Studio Code
  - https://code.visualstudio.com
- Typescript
  - https://www.typescriptlang.org
- Visual Studio Code Extension
  - Angular Extension Pack
- Angular CLI
  - https://cli.angular.io/

Angular Component

Angular CLI - ng generate component */nazwa_komponentu/*

Aplikacja Angularowa budowana jest z komponentów
Komponent powiązany jest z plikiem html i plikiem od styli (np. css)

Komponent posiada dekorator @Component

- selector - informacja o selektorze (tag html)
- templateUrl – informacja o pliku z widokiem html
- styleUrls – tablica ścieżek do plików ze stylami danego (css, scss)

W komponencie kryje się logika odpowiedzialna za wyświetlenie danych w widoku html
Komponent musi być przypisany do NgModule, aby mógł być używany
Komponent może być wielokrotnie używany w aplikacji

Wyświetlanie danych w widokach HTML-owych

Dyrektywy wyświetlania danych:

- Interpolacja stringów {{*nazwa*}}
- Wyrażenie warunkowe *ngIf
- Pętla *ngFor (możliwe odwołanie do indeksów: index, odd, even, first, last)
- <ng-container></ng-container>
- Dyrektywa <ng-template *#nazwa_taga*></ng-template>
- [ngSwitch] = ”hobby” *ngSwitchCase=”’football’” *ngSwitchDefault=”art”

Formatowanie danych (Pipes)

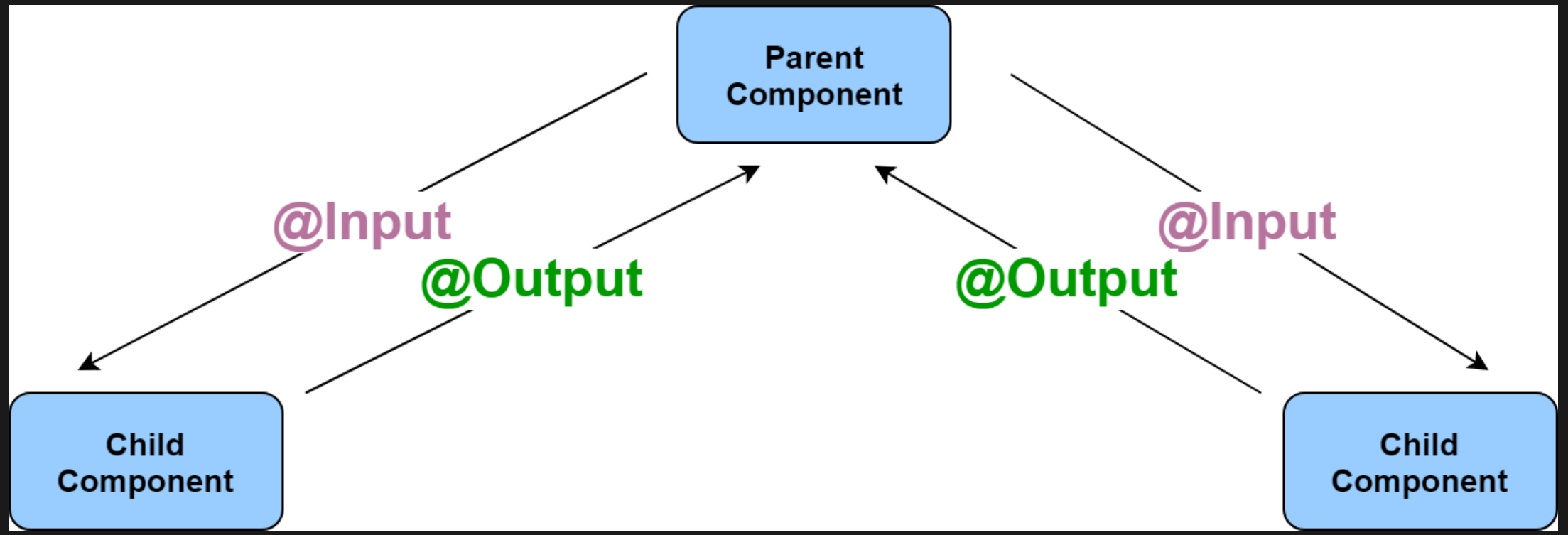
- Symbol | przykład: {{*nazwa* | uppercase }}
- https://angular.io/guide/pipes
- https://angular.io/api?type=pipe

Data Binding

Powiązanie danych komponentu z informacjami wyświetlanymi w widoku html komponentu

Property Binding [target]=*”wyrażenie”*
Two-Way-Binding [(ngModel)]=”wyrażenie”
Event Binding (target)=”wyrażenie”

Interakcja/komunikacja komponentów



Inne sposoby

- Dekorator @ViewChild() dodany w komponencie rodzica i lokalna referencja dodana w selektorze komponentu dziecka
- Content projection - <ng-content></ng-content>

Cykl życia komponentów

Interfejsy

- OnInit
  - Uruchamia się raz podczas inicjalizacji komponentu
  - Po konstruktorze komponentu i po ngOnChange
- OnChanges
  - Uruchamia się na początku (przed OnInit)
  - Sprawdza czy zmieniły się zbindowane pola komponentu
- DoCheck
  - Uruchamia się przy każdej zmianie komponentu, wywołaniu eventu
- OnDestroy
  - Uruchamiany po usunięciu komponentu z drzewa DOM (np. ngIf)

Cykl życia komponentów

Interfejsy

- AfterViewInit
  - Uruchamia się po inicjalizacji widoku
- AfterViewChecked
  - Uruchamia się po każdej zmianie widoku
- AfterContentInit
  - Uruchamia się po inicjalizacji
- AfterConentChecked
  - Uruchamia się po każdej zmianie

Serwisy

Angular CLI – ng generate service *nazwa-serwisu*

Powody dla których warto tworzyć serwisy:

- Pomagają w komunikacji z zagnieżdżonymi komponentami
- Zawierają logikę odpowiedzialną za komunikację z serwerem
- Odpowiadają za komunikację z bazą danych
- Mogą zawierać powtarzającą się logikę/obliczenia
- Zawierają logikę biznesową

Aby użyć metod serwisu w komponencie należy:

- „wstrzyknąć” instancję serwisu do komponentu przez konstruktor
- Dodać serwis do tablicy **providers: []** w dekoratorze **@Component**
- w definicji serwisu w dekoratorze **@Injectable** przypisać właściwość **providedIn: root** – serwis dostępny globalnie (od wersji 6 Angulara)

Routing

Utworzenie nowego modułu i rejestracja ścieżek
Rejestracja modułu w app.module
W pliku index.html należy dodać <base href="/">

```
@NgModule({
  declarations: [
    AppComponent,
    MyComponentComponent,
    InlineComponent,
    DataBindingsComponent,
    ChildComponent,
    ParentComponent,
    SecondChildComponent,
    SumComponent,
    SameInstanceOfServiceComponent,
    ChangeStateComponent,
    NavbarComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { ParentComponent } from './parent/parent.component';
import { SumComponent } from './sum/sum.component';

const routes: Routes = [
  {
    path: 'componentsCommunication',
    component: ParentComponent
  },
  {
    path: 'sum',
    component: SumComponent
  }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Routing

Dodanie taga <a> z ustawioną właściwością **[routerLink]=[’/path’]**
(path to jedna ze ścieżek zdefiniowana w module app-routing.module)

Dodać dyrektywę **<router-outlet></router-outlet>** w komponencie
w którym chcemy wyświetlać komponenty wskazywane przez ścieżki routingu

Formularze – Template driven

W Angularze można implementować formularze na dwa sposoby
https://angular.io/guide/forms

- Proste w użyciu
- Dobre rozwiązanie dla prostych formualrzy
- Konfiguracja po stronie widoku HTML
- Bindowanie za pomocą Two-Way Binding
- Mało kodu w komponencie
- Trudne testowanie (testy jednostkowe)

Walidacja formularza - https://angular.io/guide/form-validation

Formularze – Reactive Forms

https://angular.io/guide/reactive-forms

- Trudniejsze w implementacji
- Dużo kodu w komponencie
- Mało kodu w widoku HTML
- Bardziej elastyczne – duże możliwości modyfikacji formularza
- Możliwość dynamicznego dodawania komponentów
- Proste w testowaniu

SYNERGY CODES
SOFTWARE HOUSE

Dziękuję za uwagę

adam.kowalski@synergycodes.com

https://www.synergycodes.com