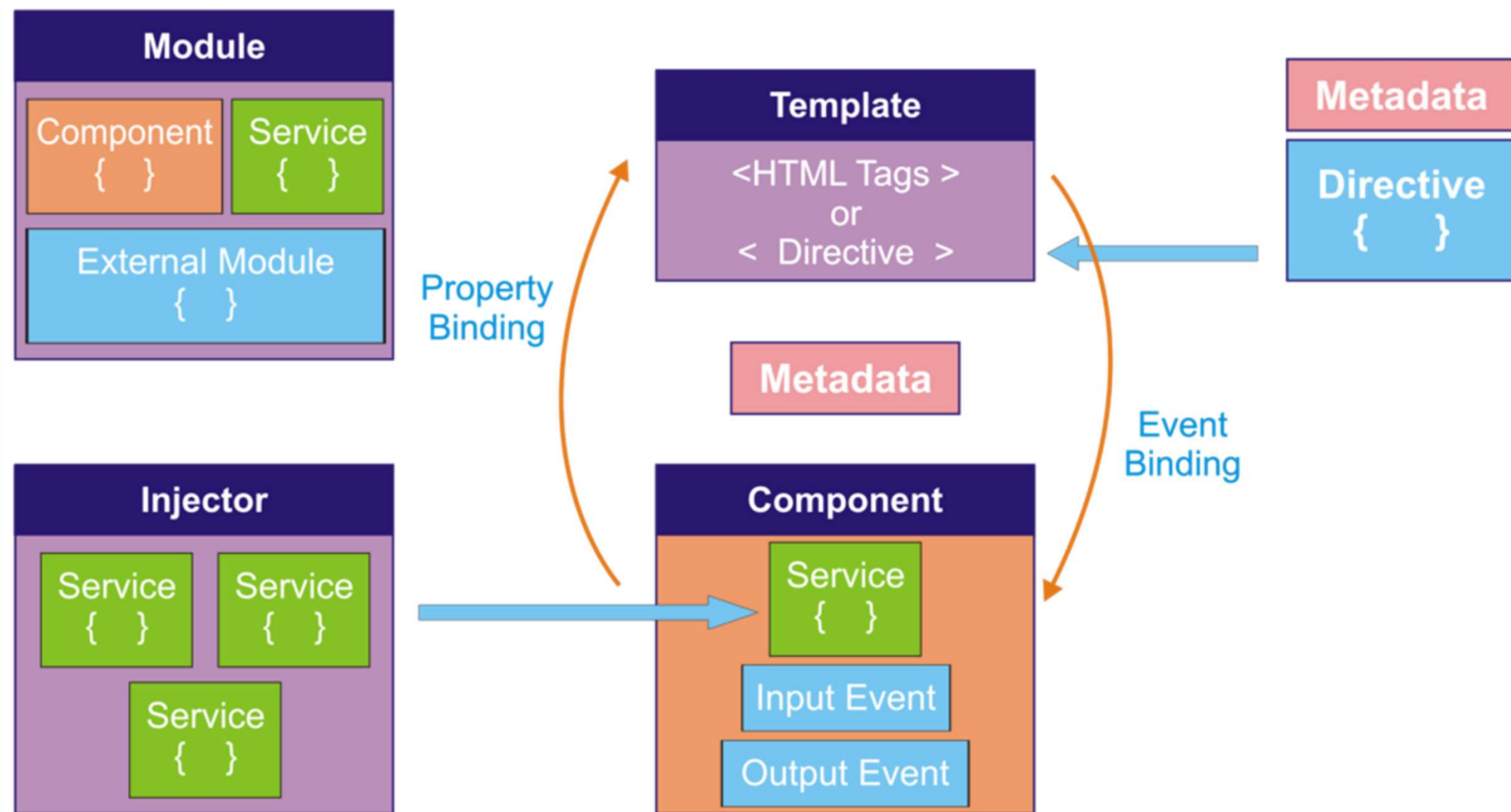


Angular

część 2

dr inż. Grzegorz Rogus

Architektura Angular



Wyzwanie – stworzenie aplikacji do dodawania komentarzy

Lista Komentarzy - Test Możliwości Angulara!

Stworz nowy komentarz

Janek

Test wstawiana poprzez formularz

Add

Janek

Test wstawiana poprzez formularz

Wyświetl

Grzegorz

Wyświetl

Iza

Wyświetl

Alicja

Fajny film wczoraj widziałam

Wyświetl

Od czego zaczynamy

1. Dodanie dwóch pól do klasy i wyświetlenie ich w szablonie:

```
export class AppComponent {  
    title = ' - Test Możliwości Angulara';  
    imie = "Grzegorz";  
    komentarz = "Pierwszy komentarz";  
}
```

```
<div style="text-align:center">  
  <h1>  
    Lista Komentarzy {{title}}!  
  </h1>  
</div>  
<div >  
  <div class="card card-block">  
    <h4 class="card-title">{{imie}}</h4>  
    <p class="card-text">{{komentarz}}</p>  
  </div>  
</div>
```

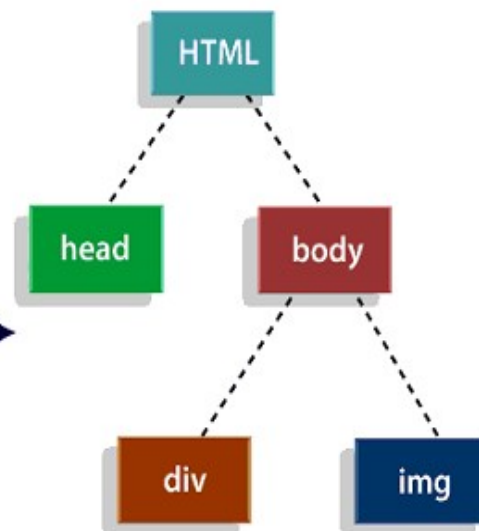


Dyrektywy

Dyrektywa modyfikuje DOM zmieniając jego wygląd lub zachowanie



DOM Manipulation



Angular 8 Directive

W Angular wyróżniamy 3 rodzaje dyrektyw:

1. **Komponenty** - dyrektywy z szablonami
2. **Dyrektywy atrybutowe** - zmieniają zachowanie komponentu/elementu ale nie wpływają na jego szablon
3. **Dyrektywy strukturalne** - zmieniają zachowanie komponentu/elementu przez modyfikację jego szablonu

Szablony Angulara są dynamiczne. Podczas ich renderowania Angular przetwarza DOM zgodnie z instrukcjami reprezentowanymi przez dyrektywy.

Lista komentarzy - Dodanie tablicy do aplikacji

```
export class AppComponent {  
  title = ' - Test Możliwości Angulara';  
  
  comments: Array<Object>;  
  
  constructor() {  
    this.comments = [  
      {  
        imie: "Grzegorz",  
        komentarz: "Pierwszy komentarz",  
        hidden: true  
      },  
      {  
        imie: "Anna",  
        komentarz: "Super strona",  
        hidden: true  
      },  
      {  
        imie: "Alicja",  
        komentarz: "Fajny film wczoraj widziałam",  
        hidden: false  
      }  
    ];  
  }  
}
```

```
<div style="text-align:center">  
  <h1>  
    Lista Komentarzy {{title}}!  
  </h1>  
</div>  
  
<div *ngFor="let comment of comments">  
  <div class="card card-block">  
    <h4 class="card-title">{{comment.imie}}</h4>  
    <p class="card-text">{{comment.komentarz}}</p>  
  </div>  
</div>
```

Lista Komentarzy - Test Możliwości Angulara!

Grzegorz

Pierwszy komentarz

Anna

Super strona

Alicja

Fajny film wczoraj widziałam

Lista komentarzy – Dodajemy możliwość ukrywania treści komentarza

```
<div *ngFor="let comment of comments">
  <div class="card card-block">
    <h4 class="card-title">{{comment.imie}}</h4>
    <p class="card-text" [hidden]="comment.hidden">{{comment.komentarz}}</p>
  </div>
</div>
```

Lista Komentarzy - Test Możliwości Angulara!

Grzegorz

Anna

Alicja

Fajny film wczoraj widziałam

Lista komentarzy – Umożliwienie samodzielnego włączania i wyłączania komentarzy

```
export class AppComponent {  
  title = ' - Test Możliwości Angulara';  
  
  comments: Array<Object>;  

```

```
  constructor() {  
    this.comments = [  
      {  
        imie: "Grzegorz",  
        komentarz: "Pierwszy komentarz",  
        hide: true  
      },  
      {  
        imie: "Anna",  
        komentarz: "Super strona",  
        hide: true  
      },  
      {  
        imie: "Alicja",  
        komentarz: "Fajny film wczoraj widziałam",  
        hide: false  
      },  
    ];  
  }  

```

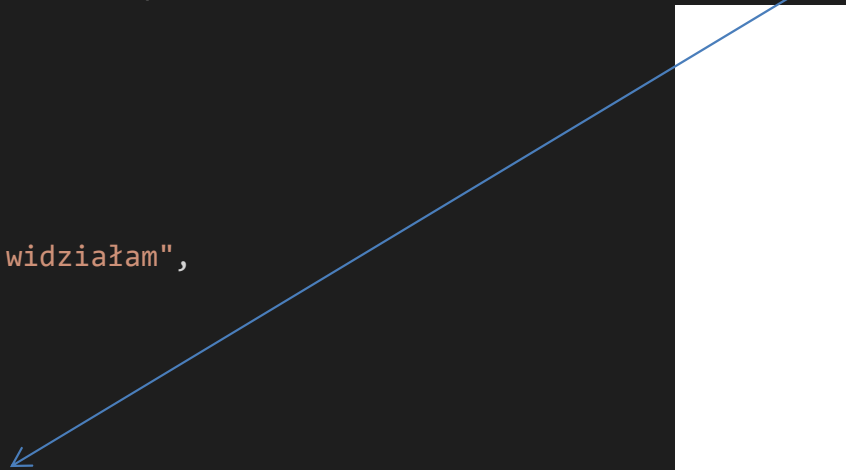
```
  toggle(comment) {  
    comment.hide = !comment.hide;  
  }  

```

```
<div *ngFor="let comment of comments">  
  <div class="card card-block">  
    <h4 class="card-title">{{comment.imie}}</h4>  
    <p class="card-text" [hidden]="comment.hide">{{comment.komentarz}}</p>  
  </div>  
  <a class="btn btn-primary" (click)="comment.hide =  
    !comment.hide">Wyświetl</a>  
</div>
```

LUB

```
<div *ngFor="let comment of comments">  
  <div class="card card-block">  
    <h4 class="card-title">{{comment.imie}}</h4>  
    <p class="card-text" [hidden]="comment.hide">{{comment.komentarz}}</p>  
  </div>  
  <a class="btn btn-primary" (click)="toggle(comment)">Wyświetl</a>  
</div>
```



Style komponentu – metody encapsulacji

- **Emulated** (default) – Stylowanie z głównego HTML przechodzą do komponentu. Style definiowane bezpośrednio w komponencie poprzez dekorator `@Component` są dostępne tylko dla tego komponentu.
- **Native** (shadow DOM) - Style zdefiniowane poza komponentem (w głównym HTML) nie są dostępne z poziomu komponentu. Style definiowane bezpośrednio w komponencie poprzez dekorator `@Component` są dostępne tylko dla tego komponentu.
- **None** – Style z komponentu są dostępne w innych częściach projektu (propagacja wsteczna) i można je używać w innych komponentach na stronie

Potoki - Pipe

- Wbudowane
 - Uppercase
 - Lowercase
 - Decimal
 - Currency
 - Date
 - Json

books-list.component.ts

```
export class BooksListComponent {  
  books: any[] = [{  
    inStock: 'yes'  
  }]  
}
```

Template

```
<h1>{{ books.inStock | uppercase }}</h1>
```



YES

Typy Potoków

```
@Pipe({  
  name: 'mojWlasnyPipe',  
  pure: false/true    <----- (default to `true`)  
})  
export class MyPipe {}
```

There are 2 types of pipes: **pure and impure**. Pipes are pure by default. You make a pipe impure by setting its pure flag to false.

Angular executes a pure pipe only when it detects a pure change to the input value. A pure change is either a change to a primitive input value such as String, Number, Boolean, Symbol or a changed object reference like Date, Array, Function or Object.

Angular executes an impure pipe during every component change detection cycle. An impure pipe is called often, even for every keystroke or mouse movement. Implement an impure pipe with great care since an expensive, long-running pipe could affect performance and user experience

Lista komentarzy - Co zmieniamy

- Dobrą praktyką podczas pisania w Angularze jest izolacja struktury danych używanych w komponencie od samego komponentu.
- Przechowywanie w jednym komponencie obsługi listy komentarzy i wyświetlania szczegółów o nich narusza zasadę pojedynczej odpowiedzialności.
- Dzielimy tę funkcjonalność na dwa osobne komponenty.

Wstrzykiwanie Komponentów

```
<component my-directive>
```

```
<sub-comp/>
```


```
<sub-comp/>
```

```
</component>
```

Komunikacja pomiędzy komponentami



```
@Component({  
  selector: "my-child",  
  template: "This is a child component"  
})  
export class ChildComponent {}
```

1. Stwórz nowy
komponent

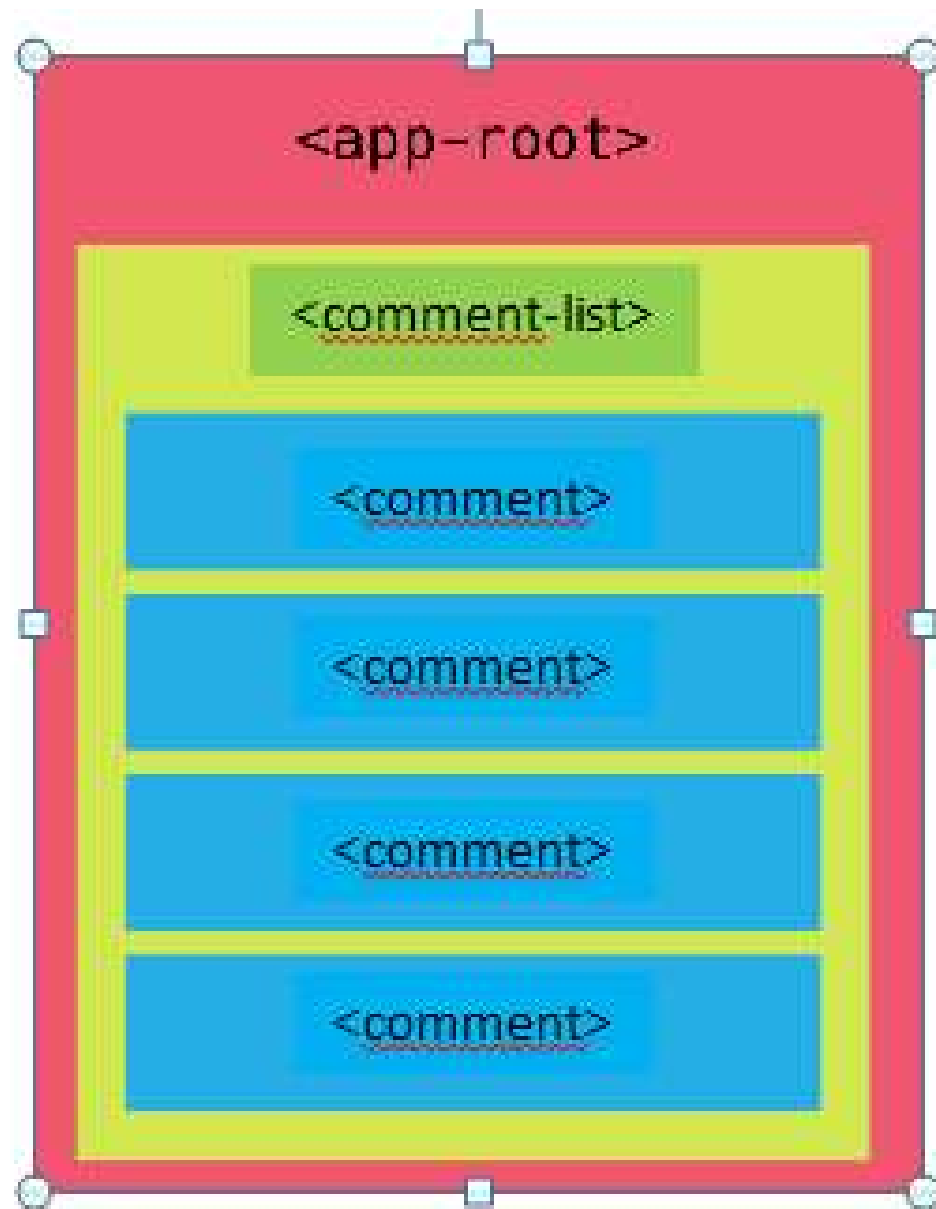


```
import { ChildComponent } from "../child.component";  
  
@NgModule({  
  declarations: [AppComponent, ChildComponent]  
})  
export class AppModule {  
  
}
```

2. Zaimportuj i
zarejestruj w
module



Lista komentarzy - Propozycja podziału na komponenty



Comment.ts

```
class Comment {  
  public imie: string;  
  public komentarz: string;  
  public hide: boolean;  
  
  constructor(imie: string, komentarz: string) {  
    this.imie = imie;  
    this.komentarz = komentarz;  
    this.hide = true;  
  }  
  
  toggle() {  
    this.hide = !this.hide;  
  }  
}
```

← Typ danych

Klasa Comment w osobnym pliku

App-component

```
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
})  
  
export class AppComponent {  
  title = ' - Test Możliwości Angulara';  
  
  constructor() {  
  }  
}
```

```
<div style="text-align:center">  
  <h1>  
    Lista Komentarzy {{title}}!  
  </h1>  
</div>  
<comment-list> </comment-list>
```


Osobny komponent do wyświetlania szczegółów

- comment-component

Przechowywanie w jednym komponencie obsługi listy komentarzy i wyświetlania szczegółów o nich narusza **zasadę pojedynczej odpowiedzialności**.

Dzielimy tę funkcjonalność na dwa osobne komponenty.

Tworzymy komponent Comment-Component.Component

```
import { Component, OnInit } from '@angular/core';
import { Comment } from '../comment';

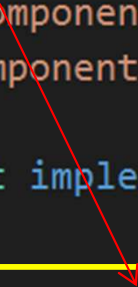
@Component({
  selector: 'comment',
  templateUrl: './comment-component.component.html',
  styleUrls: ['./comment-component.component.css']
})
export class CommentComponentComponent implements OnInit {

  com: Comment;

  constructor() { }

  ngOnInit() {
  }

}
```



```
<p>
  comment-component works!
</p>

<div class="card card-block">
  <h4 class="card-title">{{com.imie}}</h4>
  <p class="card-text" [hidden]="com.hide">{{com.komentarz}}</p>
</div>

<a class="btn btn-primary" (click)="com.toggle()">Wyświetl</a>
```

comment-list-component

```
import { Comment } from '../comment';
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'comment-list',
  templateUrl: './comment-list-component.component.html',
  styleUrls: ['./comment-list-component.component.css']
})
export class CommentListComponentComponent implements OnInit {

  comments: Array<Object>;

  constructor() {
    this.comments = [
      new Comment("Grzegorz", "Pierwszy komentarz"),
      new Comment("Iza", "Super strona"),
      new Comment("Alicja", "Fajny film wczoraj widziałam")
    ]
  }

  ngOnInit() {
  }
}
```

```
<p>
  comment-list-component works!
</p>

<comment *ngFor="let comment of comments">
</comment>
```

Lista komentarzy – po podziale na komponenty

The screenshot shows a web browser window with the address bar at `localhost:4200`. The page title is "Lista Komentarzy - Test Możliwości Angulara!". The page content displays three identical comment forms. Each form consists of a text input field, a "Wyświetl" button, and a feedback message "comment-component works!".

The browser's developer console is open, showing a red error message: "ERROR TypeError: Cannot read property 'imie' of undefined". The stack trace indicates the error occurred in `CommentComponentComponent.html:6` at `Object.eval [as updateRenderer]`, and propagated through `Object.debugUpdateRenderer [as updateRenderer]`, `checkAndUpdateView`, `callViewAction`, `execComponentViewsAction`, `checkAndUpdateView`, and `callViewAction`.

Problemem jest brak przekazania z listyKomentarzy informacji o obiekcie komentarza do komponentu Komentarza wyświetlającego jego szczegóły.

W jaki sposób komponenty się komunikują??

1. Inputs i Outputs (tylko powiązane)

2. Usługi (wszystkie)

Lista Komentarzy – komunikacja rodzic - dziecko

```
import { Component, OnInit, Input } from '@angular/core';
import { Comment } from '../comment';

@Component({
  selector: 'comment',
  templateUrl: './comment-component.component.html',
  styleUrls: ['./comment-component.component.css']
})
export class CommentComponent implements OnInit {

  @Input('komentarz') comment: Comment;

  constructor() { }
  ngOnInit() {
  }
}
```

Dziecko -
commentComponent

Lista Komentarzy - Test Możliwości Angulara!

comment-list-component works!

comment-component works!

Grzegorz

Pierwszy komentarz

Wyświetl

comment-component works!

Iza

Wyświetl

comment-component works!

Alicja

```
<p>
  comment-list-component works!
</p>

<comment *ngFor="let comment of comments" [komentarz]="comment" >
```

Rodzic comment-list- Component

Lista Komentarzy – dodanie możliwości dodawania komentarza

Lista Komentarzy - Test Możliwości Angulara!

new-comment-form works!

Stworz nowy komentarz

Podaj swoje imie

Miejsce na twój komentarz

Add

Grzegorz

Pierwszy komentarz

Wyświetl

Iza

Super strona

Wyświetl

Alicja

Fajny film wczoraj widziałam

Wyświetl

Lista Komentarzy – nowy komponent – formularz dodawania komentarzy

New-Comment-form

```
<p>
  new-comment-form works!
</p>
<div class="card card-block">
  <h4 class="card-title">Stworz nowy komentarz
  <div class="form-group">
    <input type="text"
      class="form-control"
      placeholder="Podaj swoje imie">
  </div>
  <div class="form-group">
    <input type="text"
      class="form-control"
      placeholder="Miejsce na twoj komentarz">
  </div>
</div>
<button type="button" class="btn btn-primary" (click)="AddComment()">Add
</button>
```

```
import { Component, OnInit, Output, EventEmitter } from '@angular/core';
import { Comment } from '../comment';

@Component({
  selector: 'new-comment-form',
  templateUrl: './new-comment-form.component.html',
  styleUrls: ['./new-comment-form.component.css']
})

export class NewCommentFormComponent implements OnInit {

  @Output() commentCreated = new EventEmitter<Comment>();

  AddComment() {
    this.commentCreated.emit(
      new Comment("Ula", "Test dodawania Komentarza"));
  }
}
```

Lista Komentarzy – dodawania komentarzy

Comment-List-Componet

```
<new-comment-form (commentCreated)="DodajComment($event)">
</new-comment-form>


<comment *ngFor="let comment of comments" [komentarz]="comment" >
</comment>
```

```
DodajComment(param){
  this.comments.unshift(param);
}
```


Lista Komentarzy – dodawania komentarzy – wersja ulepszona

New-Comment-form

```
div class="card card-block">
  <h4 class="card-title">Stworz nowy komentarz</h4>
  <div class="form-group">
    <input type="text" class="form-control"
      placeholder="Podaj swoje imie" #imie>
  </div>
  <div class="form-group">
    <input type="text" class="form-control"
      placeholder="Miejsce na twoj komentarz" #comment>
  </div>
</div>
<button type="button" class="btn btn-primary"
(click)="AddComment(imie.value,comment.value)">Add </button>
```



```
AddComment(imie:string, komentarz:string) {
  this.commentCreated.emit(new Comment(imie,komentarz));
}
```

We wcześniejszej wersji rozwiązania wysyłany był ciągle tekst zaszyty w kodzie
Tutaj wersja z danymi pobieranymi z formularza

Lista komentarzy - Wersja końcowa

Lista Komentarzy - Test Możliwości Angulara!

Stwórz nowy komentarz

Janek

Test wstawiana poprzez formularz

Add

Janek

Test wstawiana poprzez formularz

Wyświetl

Grzegorz

Wyświetl

Iza

Wyświetl

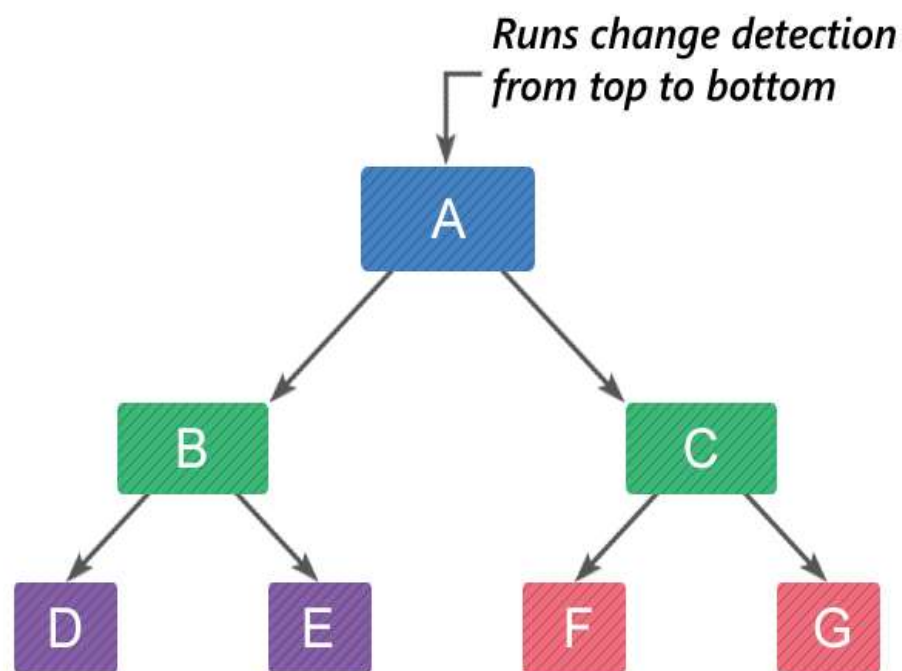
Alicja

Fajny film wczoraj widziałam

Wyświetl

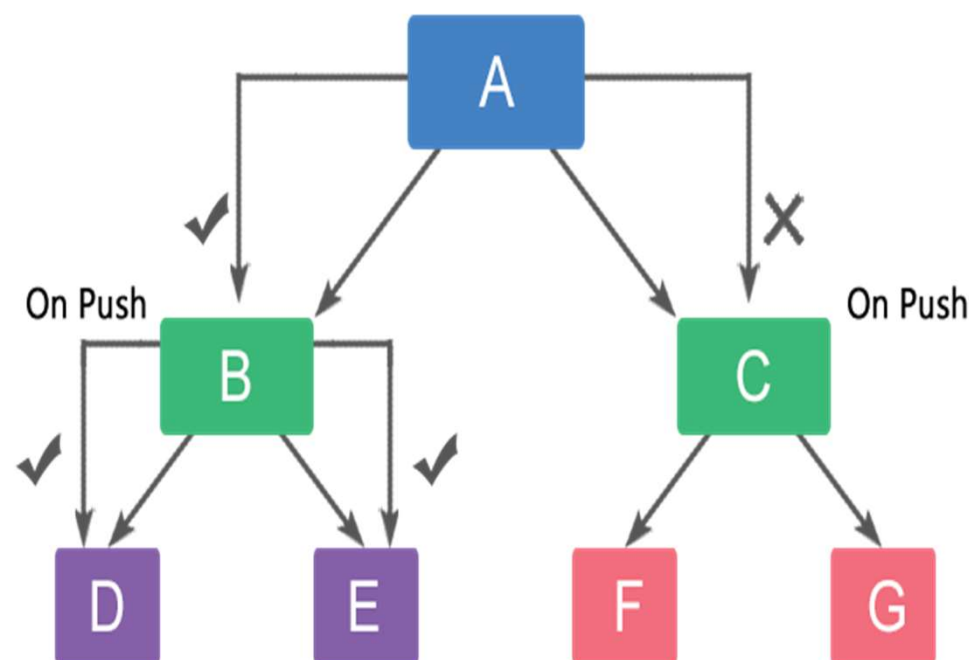
Wydajność - zmiana strategii odświeżania

ChangeDetectionStrategy.Default



Change detection will trigger for all Components

Strategia onPush



Wydajność - zmiana strategii odświeżania

Angular dla każdego komponentu tworzy odpowiadający mu (komponentowi) ChangeDetector.

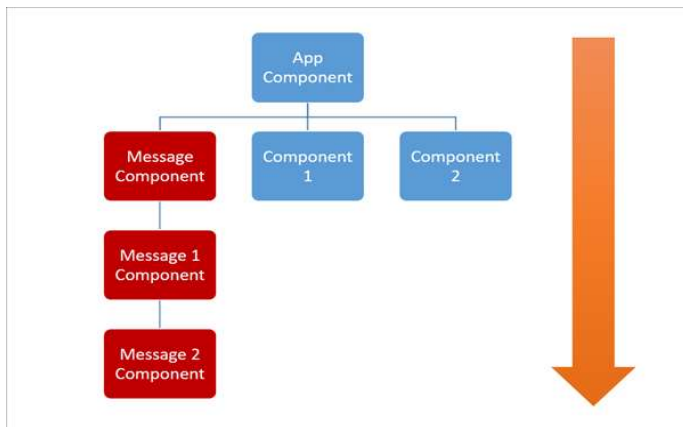
ChangeDetectionStrategy.Default - strategia CheckAlways

Strategia ta sprawia, że podczas każdej zmiany stanu aplikacji - asynchronicznego zapytania wysłanego do serwera, zdarzenia DOM, interakcji użytkownika z naszą aplikacją **sprawdzone jest całe drzewo komponentów**.

Strategia onPush

Strategia ta mówi nam, że komponent zależny jest tylko i wyłącznie od swoich inputów. Taki komponent nazywamy “czystym”. Zmiana propagowana jest w momencie zmiany referencji inputów komponentu jak i w przypadku wyemitowania zdarzenia DOM w szablonie komponentu (np. kliknięcie w przycisk - event onclick).

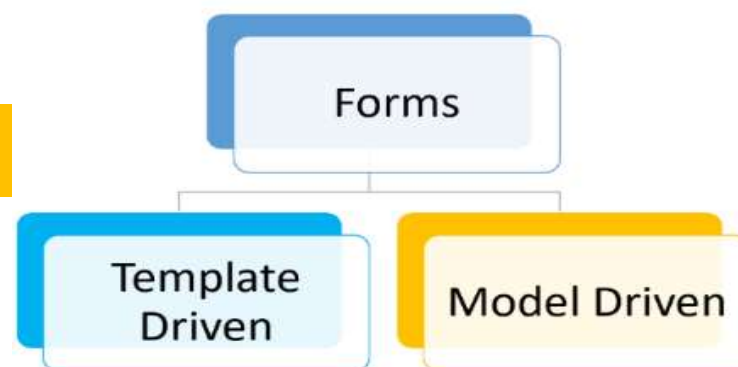
Co więcej, komponent emitujący zmianę z wykorzystaniem strategii onPush **powiadamia mechanizm detekcji Angulara, że to właśnie on wyemitował zmianę!** To drastycznie zmniejsza koszt przeszukania drzewa komponentów, gdyż Angular wie, którego komponentu szukać, albo który komponent pominąć.



```
@Component({
  ...,
  template: '{{ count$ | async }}',
  changeDetection: ChangeDetectionStrategy.OnPush
})
```

Formularze w Angular

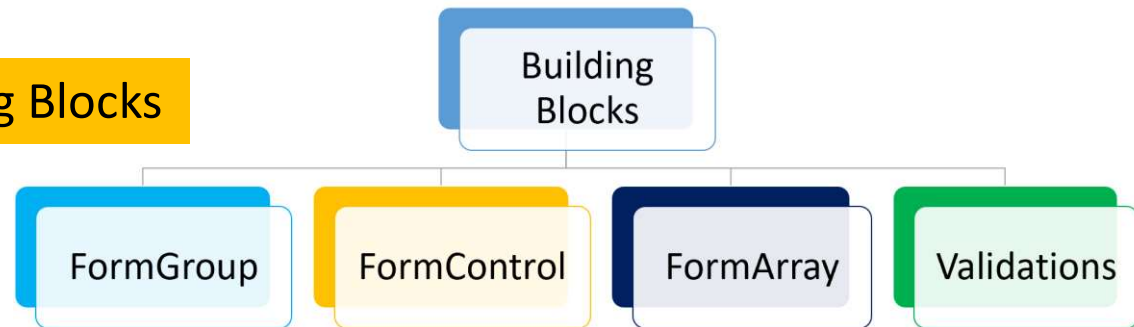
Reactive Forms a Template-driven Forms



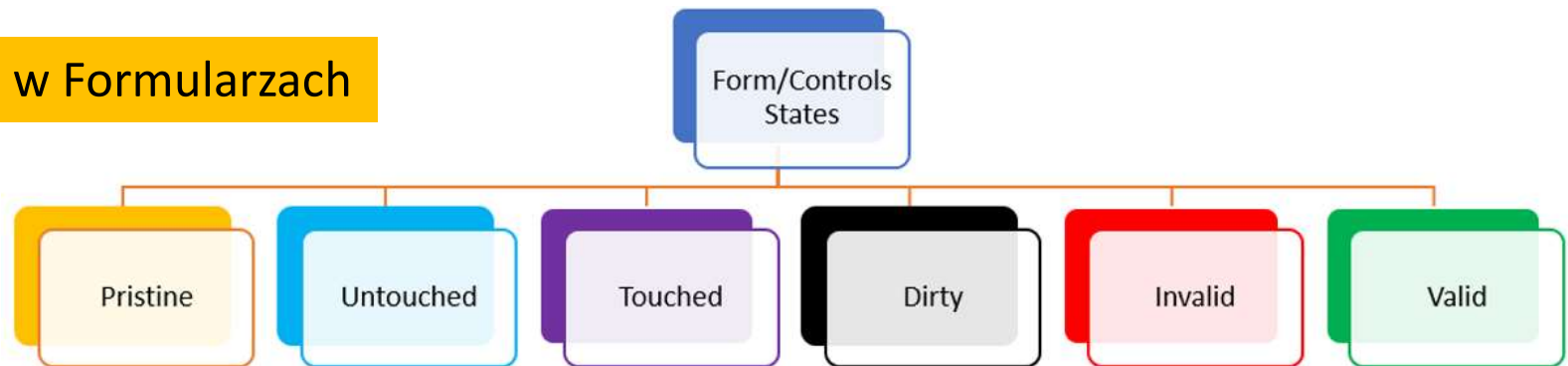
Template Driven	Model Driven Form
Formularz tworzony i konfigurowany w HTML	Formularz tworzony i konfigurowany w klasie komponentu przez FormBuilder
Idealny do prostych formularzy	Elastyczny, dla rozbudowanych formularzy
Two-way data binding	Brak data binding
Automatyczne śledzenie zmian stanu elementów w formularzu	Obsługa zmian w modelu skojarzonym z formularzem
walidacja statyczna	możemy zmieniać zasady walidacji w czasie RunTime'u
	pozwała również na dynamiczne zmiany modelu formularza, tzn. dodawanie nowych FormGroup oraz FormControl w locie

Formularze w Angular

Angular Form Building Blocks



Stany w Formularzach



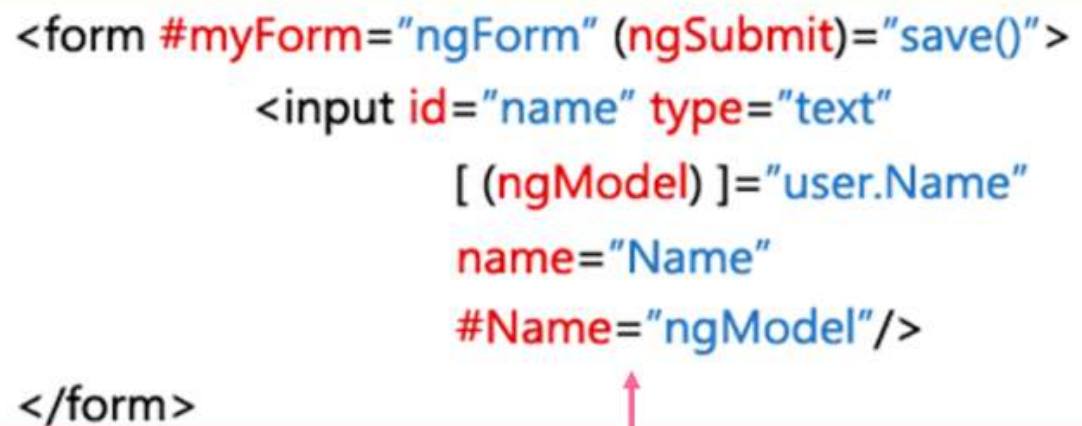
- ważny/nieważny
- nietknięte/dotknięte
- niezmieniony/zmieniony ("brudny")

Template Driven - przykłady

```
<form #tdForm="ngForm" (ngSubmit)="submit(tdForm)">
  <label>username:</label><input name="username" ngModel>
  <label>lastname:</label><input name="lastname" ngModel>
  <label>age:</label><input name="age" ngModel>
  <label>gender:</label><input name="gender" ngModel>
  <label>country:</label><input name="country" ngModel>

  <button type="submit">Send</button>
</form>
```

Form Group

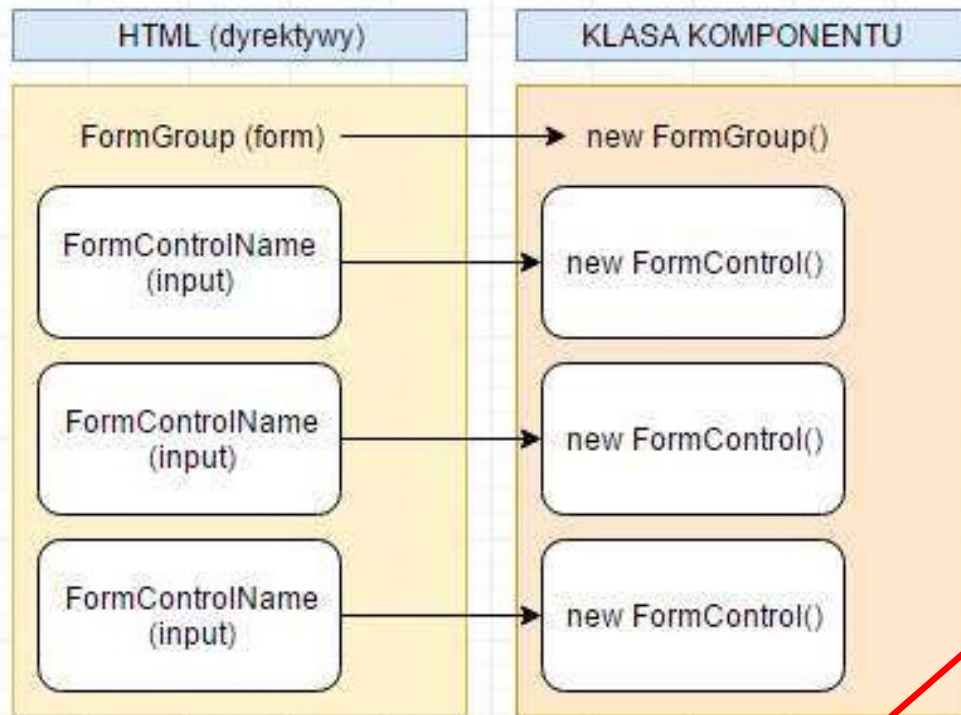


```
<form #myForm="ngForm" (ngSubmit)="save()">
  <input id="name" type="text"
    [ (ngModel) ]="user.Name"
    name="Name"
    #Name="ngModel"/>
</form>
```

Form Control

Model Driven - przykłady

Nowy sposób tworzenia



```
import { FormBuilder, FormGroup } from '@angular/forms';

@Component({
  selector: 'model-driven-form',
  ...
})

export class ModelDrivenFormComponent implements OnInit {

  modelForm : FormGroup;

  constructor(private formBuilder : FormBuilder) {}

  ngOnInit() : void {
    this.modelForm = this.formBuilder.group({
      firstname: '',
      lastname: '',
      age: '',
      gender: '',
      country: ''
    });
  }
}
```

Wygenerowany widok

```
<form>
<label>firstname:</label><input name="firstname">
<label>lastname:</label><input name="lastname">
<label>age:</label><input name="age">
<label>gender:</label><input name="gender">
<label>country:</label><input name="country">

<button type="submit">Send</button>
</form>
```

Stary sposób tworzenia

```
ngOnInit() : void {
  this.modelForm = new FormGroup({
    firstname: new FormControl(),
    lastname: new FormControl(),
    age: new FormControl(),
    gender: new FormControl(),
    country: new FormControl('Poland') // default value
  });
}

onSubmit(form) : void {
  console.log(form.value)
};
```




Usługi

Service

Klasa do specyficznych celów:

- dzielenia danych
- implementacja logiki aplikacyjnej/biznesowej
- zewnętrzna interakcja – odczyty danych z serwera

Dlaczego Usługi?

- Usługi najlepiej używać do obsługi CRUD na danych
- Pozwalając na separację danych od logiki przetwarzania danych!



Użycie zewnętrznej Usługi - service

```
@Injectable()
export class KsiazkiService {

  constructor() { }

  table = [ {title:"Node.js, MongoDB, AngularJS. Kompendium wiedzy", price:99},
    {title:"Tworzenie gier internetowych. Receptury", price:49},
    {title:"Web 2.0 Architectures. What entrepreneurs and information architects need to know", price:84.92},
    {title:"React dla zaawansowanych", price:45},
    {title:"Spring MVC 4. Projektowanie zaawansowanych aplikacji WWW", price:102}
  ];

  getKsiazki(){
    return this.table;
  }

}
```

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { KsiazkiService } from './ksiazki.service';
```

```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [
    KsiazkiService
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Rejestracja w module

Wstrzykiwanie serwisu do Komponentu



Provider



Injector

```
export class DashboardComponent  
  constructor(private dataService: DataService) { }  
}
```

Wstrzykiwanie do konstruktora – Dependency Injection

```
import { Component } from '@angular/core';
import { KsiazkiService } from '../ksiazki.service';
```

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```

```
export class AppComponent {
  name = 'Grzegorz Rogus';
```

```
  ksiazki;
```

```
  constructor(service: KsiazkiService) {
    this.ksiazki = service.getKsiazki();
  };
}
```

```
<div style="text-align:center">
```

```
  <h1>
```

```
    Witaj {{name}}!
```

```
  </h1>
```

```
</div>
```

```
<p>
```

```
  Ksiazki warté polecenia na temat technologii Webowych:
```

```
</p>
```

```
<ul>
```

```
  <li *ngFor=" let ksiazka of ksiazki">
```

```
    {{ksiazka.title}} w cenie {{ksiazka.price}}
```

```
  </li>
```

```
</ul>
```

Model dziedziny i Mock data

Dobrą praktyką projektową jest wyizolowanie struktury danych oraz ewentualnych danych od komponentu.

```
export interface Product {  
  id: number;  
  modelName: string;  
  color: string;  
  productType: string;  
  brand: string;  
  price: number;  
}
```

product.ts

```
import { Product } from '../models/product';  
  
export class MockData {  
  public static Products: Product[] = [  
    {  
      'id': 11,  
      'modelName': 'F5 Youth',  
      'color': 'Gold',  
      'productType': 'Mobile',  
      'brand': 'OPPO',  
      'price': 16990  
    },  
    {  
      'id': 12,  
      'modelName': 'Inspiron',  
      'color': 'Gray',  
      'productType': 'Laptop',  
      'brand': 'DELL',  
      'price': 59990  
    }  
  ]  
}
```

mock-product-data.ts

Mock Data



ProductService
products : Product[]

ProductsComponent
constructor(productService : ProductService)

AddProductComponent
constructor(productService : ProductService)

OtherComponents
constructor(productService : ProductService)

Lista komentarzy – implementacja usługi

- Przenosimy dane z pliku component-list do pliku mock.ts (symulującego źródło danych).
- Następnie tworzymy serwis udostepniający dane pochodzące z mock

Dlaczego usługa obsługi danych?

- Użytkownik usługi nie wie z jakiego źródła są dane.
- Dane mogą pochodzić z Web Serwisu, z lokalnego pliku albo być imitowane.
- To jest piękno korzystania z usług!
- Usługa odpowiada za dostęp do danych.
- W każdej chwili można zmienić sposób dostępu - zmiany są tylko w tej jednej usłudze.

Lista komentarzy

Realizacja usługi udostępniającej dane

```
import { Comment } from '../comment';
export const KomentarzeDane: Comment[] = [
  {imie: "Grzegorz", komentarz: "Pierwszy komentarz", hidden: true },
  {imie: "Anna", komentarz: "Super strona", hidden: false },
  {imie: "Alicja", komentarz: "Fajny film wczoraj widziałam", hidden: true },
];
```

```
import { Injectable } from '@angular/core';
import { KomentarzeDane } from '../mock';
@Injectable()
export class KomentarzeService {
  getComments() {
    // return komentarze;
    return Promise.resolve(KomentarzeDane);
  }
}
```

Komunikacja asynchroniczna

```
constructor( service: KomentarzeService ) {

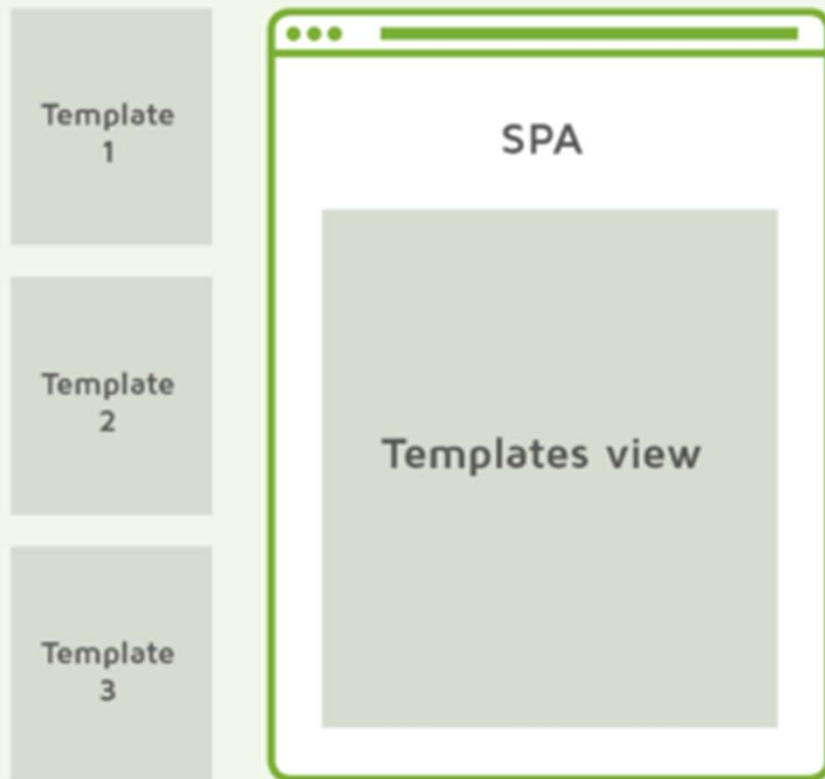
  this.comments = service.getComments();
}
```



Routing

Nawigacja po aplikacji

Single Page Application



No page refresh on request

Traditional Web Application



Whole page refresh on request

App
Shell



Header

Home

Contact

About

App Component

Footer



Routing

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)



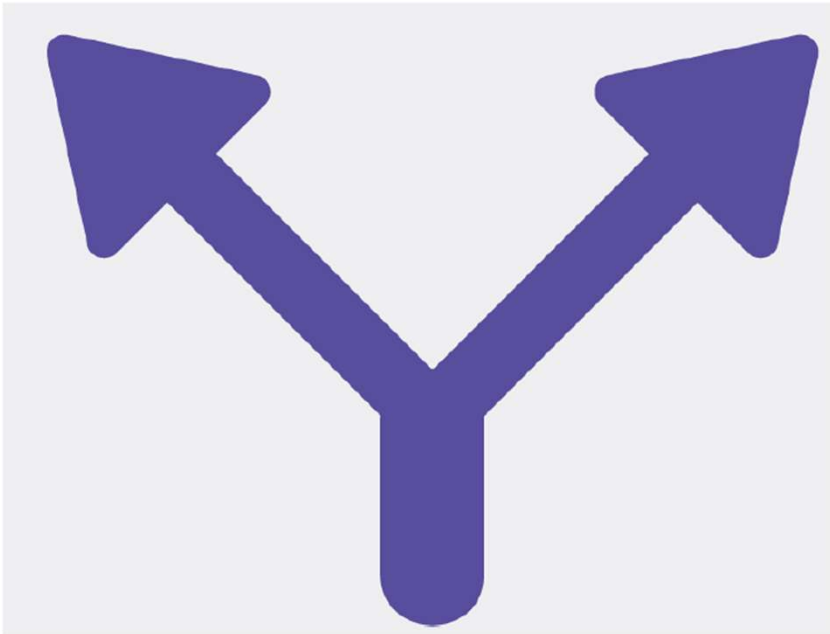
Routing

- Zadaniem routera w ramach frameworka Angular jest nawigacja między widokami
- Router Angulara bazuje na modelu nawigacji przeglądarki
 - URL wprowadzony w pasku adresu prowadzi do wskazanego widoku
 - Kliknięcie linku w aktualnym widoku powoduje przejście do innego
 - Adres URL może zawierać parametry dla widoku
 - Przyciski Back i Forward w przeglądarce nawigują po historii
- Obszar na stronie, w którym wyświetlane mają być różne komponenty zależnie od stanu routera, wskazuje się znacznikiem
`<router-outlet> </router-outlet>`



Routing

Składowe routingu



1. `<basehref="/">`
2. `import RouterModule`
3. Konfiguracja ścieżek
4. `<router-outlet>`

Konfiguracja routingu

CLI -> Would you like to add Angular routing? (y/N)

src/app/app-routing.module.ts



```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [];

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

src/app/app.component.html

```
<router-outlet></router-outlet>
```


Definiowanie tablicy routes

```
const routes: Routes = [  
  { path: 'component-one', component: ComponentOne },  
  { path: 'component-two', component: ComponentTwo }  
];
```

Definiowanie połączeń między trasami

```
<a routerLink="/component-one">Component One</a>
```

Nawigacja programowo

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

Deklaracja parametrów trasy

```
export const routes: Routes = [  
  { path: '', redirectTo: 'product-list', pathMatch: 'full' },  
  { path: 'product-list', component: ProductList },  
  { path: 'product-details/:id', component: ProductDetails }  
];
```



localhost:4200/szczegóły produktu/5

Powiązanie tras z parametrami

```
<a *ngFor="let product of products"  
  [routerLink]="['/product-details', product.id]">  
  {{ product.name }}  
</a>
```

Kontrolowanie dostępu do lub z Route

Niektóre trasy mają być dostępne tylko po zalogowaniu się użytkownika lub zaakceptowaniu Warunków.

```
const routes: Routes = [  
  { path: 'home', component: HomePage },  
  {  
    path: 'accounts',  
    component: AccountPage,  
    canActivate: [LoginRouteGuard],  
    canDeactivate: [SaveFormsGuard]  
  }  
];
```

```
import { CanActivate } from '@angular/router';  
import { Injectable } from '@angular/core';  
import { LoginService } from './login-service';
```

```
@Injectable()  
export class LoginRouteGuard implements CanActivate  
{  
  constructor(private loginService: LoginService) {}  
  
  canActivate() {  
    return this.loginService.isLoggedIn();  
  }  
}
```

Mock Data



ProductService
products : Product[]

ProductsComponent
constructor(productService : ProductService)

AddProductComponent
constructor(productService : ProductService)

OtherComponents
constructor(productService : ProductService)

Uzycie serwisu

- ng g service product

```
import { MockData } from '../mock-data/mock-product-data';
import { Injectable } from '@angular/core';
import { Product } from '../models/product';

@Injectable()
export class ProductService {
  products: Product[] = [];

  constructor() {
    this.products = MockData.Products;
  }

  getProducts(): Product[] {
    return this.products;
  }

  removeProduct(product: Product) {
    let index = this.products.indexOf(product);
    if (index !== -1) {
      this.products.splice(index, 1);
    }
  }

  getProduct(id: number): Product {
    return this.products.find( p => p.id === id);
  }

  addProduct(product: Product) {
    this.products.push(product);
  }
}
```

Wywołanie usługi – przeniesienie danych do usługi

```
export class ProductsComponent implements OnInit {  
  products: Product[] = [];  
  
  constructor(public productService: ProductService) {  
    // this.products = productService.getProducts();  
  }  
  
  ngOnInit() {  
    this.products = productService.getProducts();  
  }  
  
  deleteProduct(product: Product) {  
    this.productService.removeProduct(product);  
    this.products = this.productService.getProducts();  
  }  
}
```

