

Wstęp do Angular

dr inż. Grzegorz Rogus

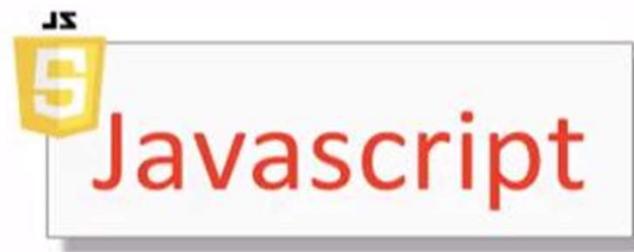
W stronę app webowej – co już wiemy



Wstawienie div czy akapitu
Dodanie przycisku



Określenie wyglądu



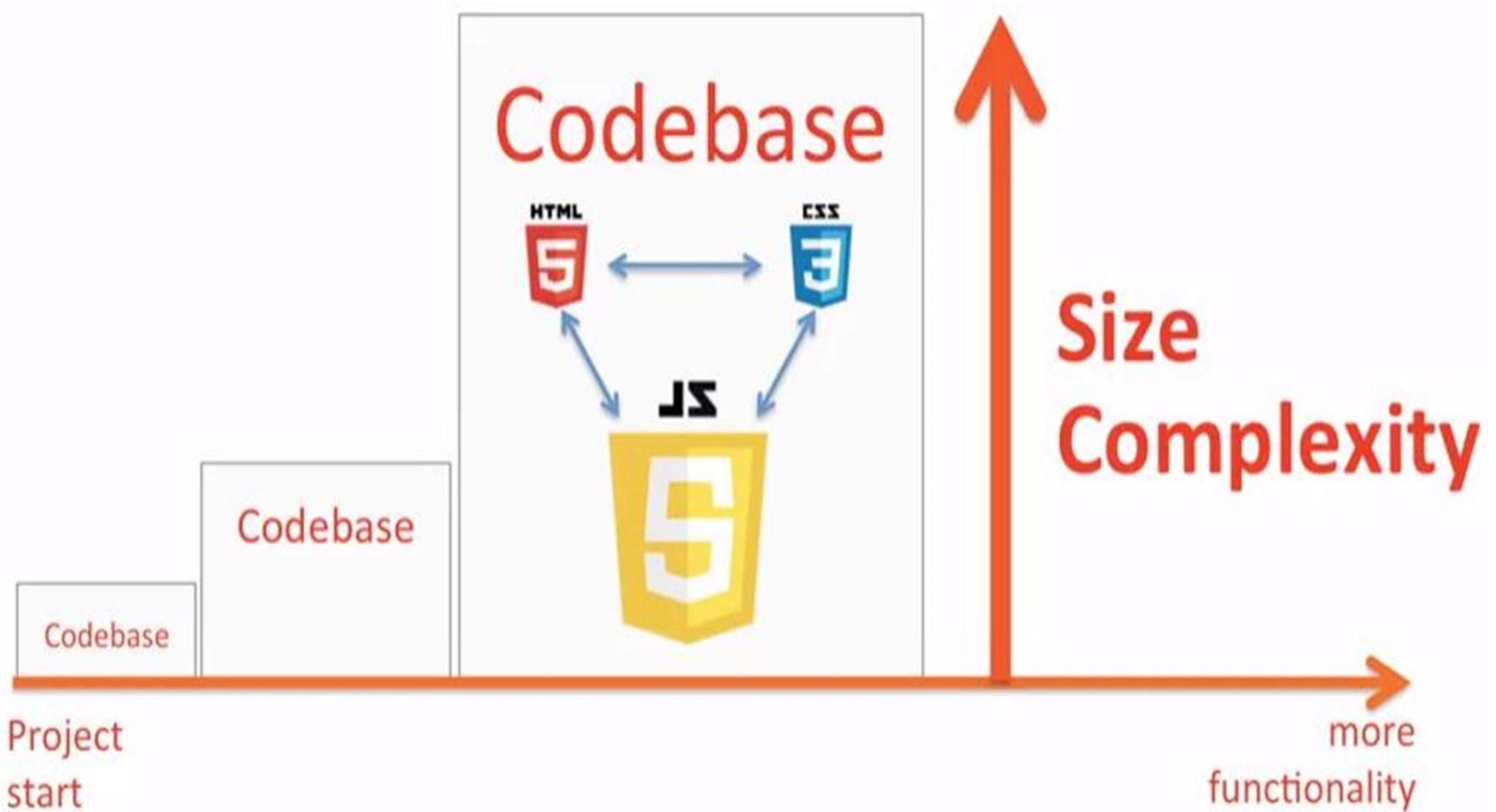
Kod do wyświetlenia daty/czasu
Odczyt zawartości akapitu z modelu DOM
Aktualizacja zawartości akapitu
Kod do obsługi zdarzeń związanych z przyciskiem

Fundamenty Front-end



- Dlaczego nie możemy na tym poprzestać?
- Po co korzystać z jakiś frameworków, bibliotek itp.?

Wzrost złożoności oprogramowania



Oczekiwania developerów

LEPSZA ORGANIZJA KODU ---> Szybkość odnajdywania odpowiedniego fragmentu kodu (przez dowolnego użytkownika)

NIEZALEŻNA MODYFIKACJA FRAGMENTU KODU --->
Modyfikacja funkcjonalności bez potrzeby dokonywania zmian w innych częściach kodu

KOD WIELOKROTNEGO UŻYTKU ---> nie ma potrzeby pisania podobnego kodu dwa razy

ŁATWOŚĆ TESTOWANIA KODU ---> mała granulacja kodu

Celem stosowania nowych technologii, frameworków jest walka ze złożonością

Dlaczego frameworki?

W zasadzie ich nie potrzebujemy!

- Musisz napisać cały kod samodzielnie („wymyślić na nowo koło”)
- Możesz napisać nieoptymalny kod lub będzie zawierał błędy
- Praca w zespole może być trudniejsza, ponieważ nie wszyscy znają strukturę Twojego projektu i „filozofię kodowania”

ale lepiej używać frameworka w większych projektach

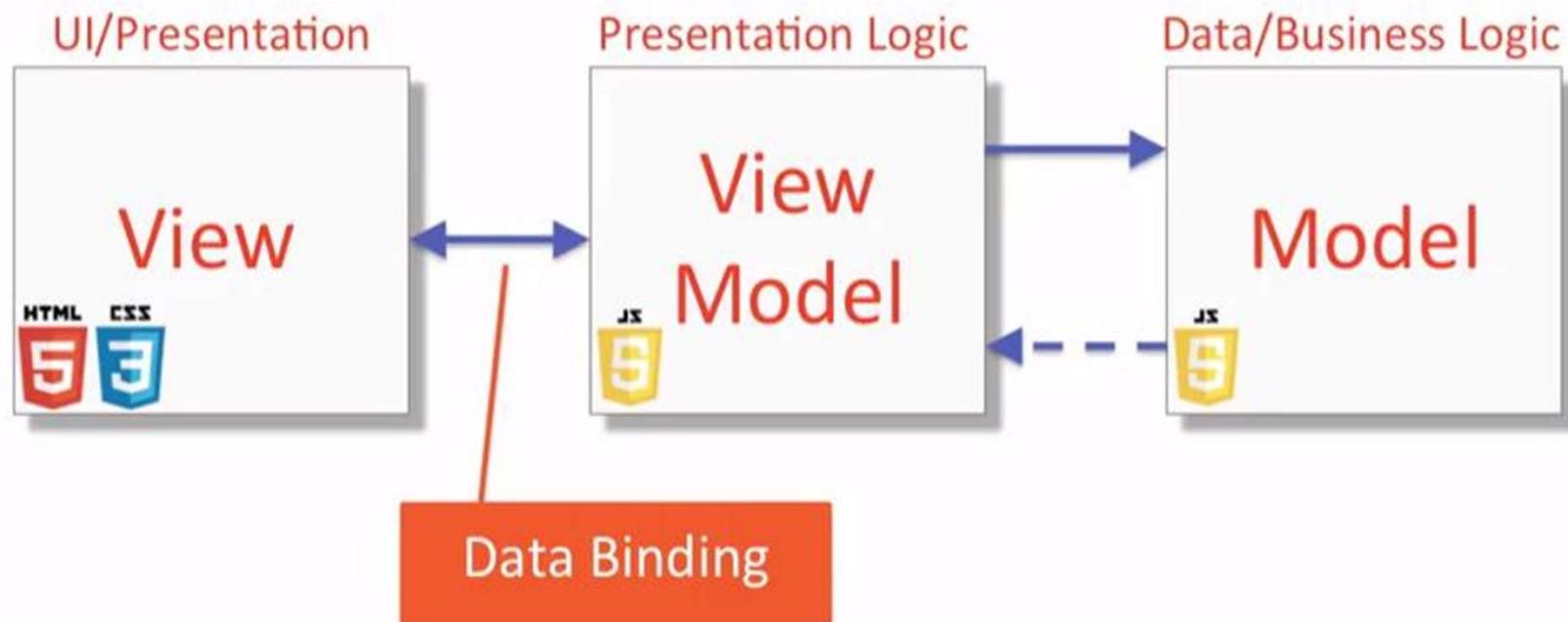
Wzorzec rozwiązuający problem złożoności

Model-View-ViewModel



Wzorzec rozwiązuający problem złożoności

Model-View-ViewModel



Wzorzec rozwiązuający problem złożoności – realizacja

Model-View-ViewModel

Model

Reprezentuje i przechowuje dane

- Cześć z tych danych w różnej formie może być wyświetlonych w widoku
- Może zawierać logikę wykorzystywaną do pobierania danych z różnych źródeł (AJAX, REST API itp.)
- Nie powinien zawierać logiki powiązanej z wyświetlaniem modelu

Model-View-ViewModel

View (Widok)

Interfejs użytkownika

- W aplikacjach Webowych realizowany w HTML i CSS
- Wyświetla tylko dane
- Nigdy nie powinien zmieniać danych
- Jest źródłem zdarzeń lecz nigdy nimi nie zarządza ani nie obsługuje

Model-View-ViewModel

ViewModel

Reprezentuje stan widoku

- Przechowuje wartości danych wyświetlanych w widoku
- Odpowiada na zdarzenia widoku poprzez logikę prezentacji
- Wywołuje inne funkcje w celu realizacji złożonej logiki biznesowej
- Nigdy bezpośrednio nie wymusza na widoku wyświetlania czegokolwiek

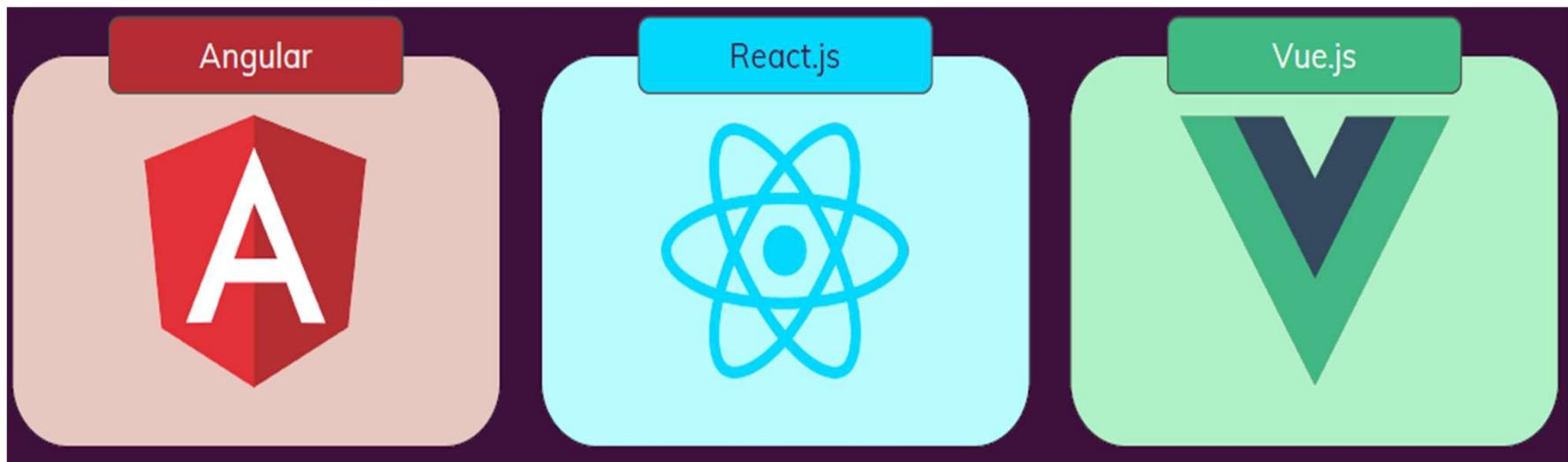
Model-View-ViewModel

Data Binder

Łączy model z ViewModel z widokiem View

- Nie wymaga żadnego dodatkowego kodu - zajmuje się tym „magicznie” framework
- Klucz funkcjonowania całego wzorca MVVM

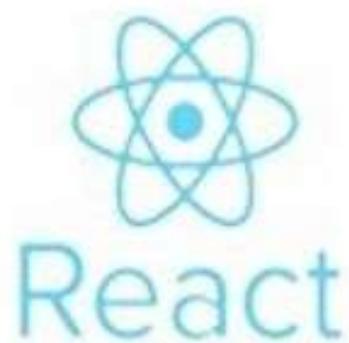
Top 3 JS Frameworków



Podobieństwo między tymi 3 frameworkami:

- Bardzo popularny, używany przez topowe firmy
- Dojrzałe, stabilne, finansowane długoterminowe wsparcie
- Kod zorientowana na komponenty
- Nadaje się do nowoczesnych stosów technologicznych (ES6 + / TypeScript)
- Duży ekosystem komponentów i narzędzi

Top 3 JS Frameworków



Czym jest Angular?

Angular to framework JavaScript, który pozwala na tworzenie reaktywnych aplikacji typu Single Page Application (SPA)



<https://angular.io/>

Główna motywacja:

HTML jest odpowiedni dla stron WWW, ale nieodpowiedni dla aplikacji webowych

Cechy Angular

- Framework również dla urządzeń mobilnych
- Większa modularność
- Łatwość i szybkość tworzenia kodu (wysoka produktywność)
- Wsparcie tylko współczesnych przeglądarek
 - Uproszczenie frameworka przez brak workarounds dla starszych
- Orientacja na język TypeScript
 - Choć możliwe jest programowanie w czystym JavaScript
- Wykorzystanie elementów ECMAScript 2015
- Dynamiczne ładowanie
- Prostszy routing, serwisy do logiki biznesowej
- Poprawione wstrzykiwanie zależności
- Komponenty i dyrektywy jako podstawa
- Programowanie reaktywne (RxJs)
- Google + Microsoft

W czym pisać?



ES5

Dart

CoffeeScript

ES2015

ES7

Typescript

Dlaczego TypeScript zamiast JavaScript?

Jezyk JavaScript:

- oprócz licznych zalet posiada jednak kilka słabych stron,
- od samego początku nie był tworzony z myślą o dużych aplikacjach webowych,
- obecnie aplikacje oparte o JavaScript zawierają często ponad 50 tysięcy linii kodu.

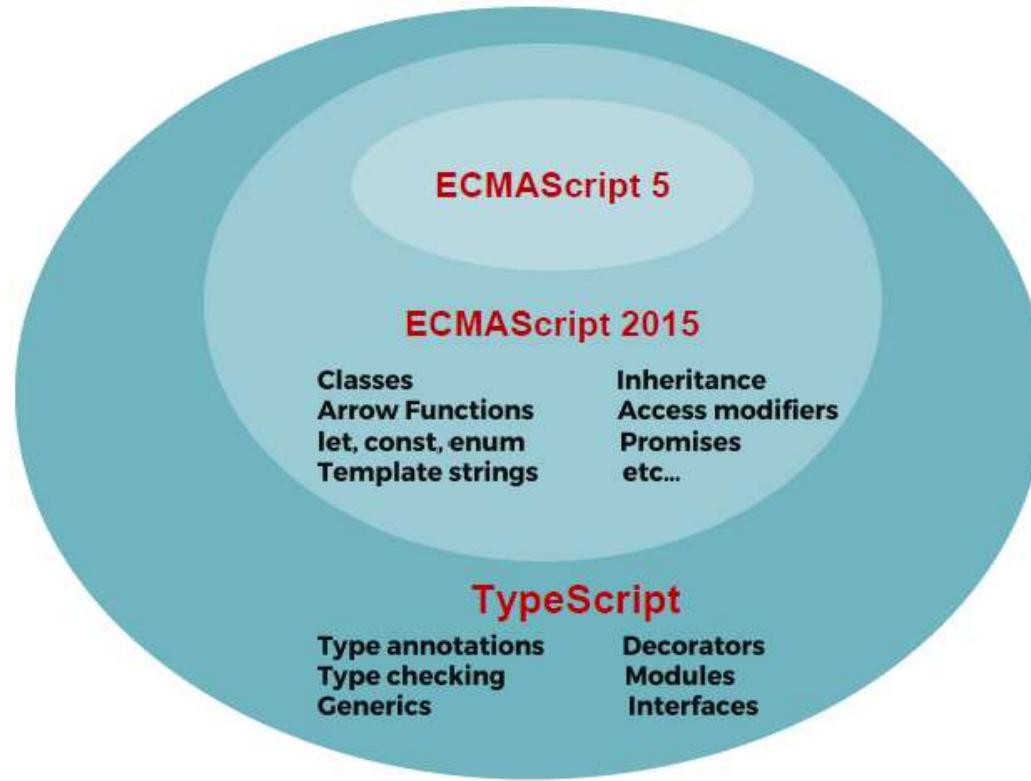
Powstanie TypeScript:

- z inicjatywy Microsoft w 2012 roku powstaje TypeScript,
- jedna z osób, które pracują nad nim jest Anders Hejlsberg, główny architekt języka C#,
- syntaktycznie (składniowo) jest on podobny do JS.

TypeScript można zainstalować na dwa sposoby:

- korzystając z npm (menadżer pakietów w Node.js):
npm install -g typescript
- dodając plugin TypeScript do VisualStudio (w nowszych wersjach jest dodany domyślnie).

Dlaczego TypeScript?



Najważniejsze cechy języka TypeScript:

- Klasy, moduły, funkcje strzałkowe, param. opcjonalne/domyślne (ES6)
- Składowe public (domyślnie), protected i private, przestrzenie nazw
- Adnotacje typu i kontrola typów na etapie kompilacji
- Automatyczna dedukcja typu (type inference)
- Interfejsy, klasy używane jako interfejsy (implements)
- Typy generyczne (<>), unie (union type), krotki (tuple)
- Dekoratory (@)

TypeScript: typy

| | typ: | przykładowe wartości: |
|-----------------------------------------|------------------|-------------------------------------------------|
| Typy proste (przez wartość) | boolean | true, false |
| | number | liczby, Infinity, NaN |
| | string | znaki, ciągi znaków |
| Typy referencyjne (przez referencje) | [] | tablice innych typów, np. number[], boolean[] |
| | { } | obiekt |
| | undefined | nie zdefiniowany |

TypeScript: typy

| typ: | przykładowe wartości: |
|-------------|---------------------------------------------------------------------------------------------------|
| enum | wylistowanie, np. { Admin, Publisher, User } |
| any | dowolny typ |
| void | nic |
| - | w przypadku braku określenia typu, TypeScript może się go "domyślić" (<i>type inference</i>) |

```
const title: string = `Ala ma kotka, a 2 + 2 wynosi 4`;
```

=

```
const title = `Ala ma kotka, a 2 + 2 wynosi 4`;
```

Definiowanie zmiennych w TS

Deklaracja typów prostych

```
let x: number;  
let name: string;  
let empty: boolean;
```

Tupla (krotka) to skończona lista elementów. W TypeScriptie jest to tablica, której długość jest dokładnie znana, a typy wszystkich elementów jasno określone:

Deklaracja typu tablicowego

```
let tab1: number[];  
let tab2: Array<number>;
```

skończona lista elementów

```
let tuple: [number, boolean,  
string];
```

Powyższy zapis oznacza, że pod indeksem 0 musi znajdować się wartość typu number, pod indeksem 1 boolean a 2 string.

Funkcje i interfejs w TS

```
function log(msg: string): void {  
    console.log(msg);  
}
```

```
function multiply(a: number, b: number): number  
{  
    return a * b;  
}
```

```
interface Person {  
    name: string;  
    age?: number;  
  
    order(): void;  
}
```

TypeScript

```
1 interface Person {  
2     firstName: string;  
3     lastName: string;  
4 }  
5  
6 let greeter = (person: Person) => {  
7     return `Hello, ${person.firstName} ${person.lastName}`;  
8 }  
9  
10 let user = { firstName: 'Jon', lastName: 'Doe' };  
11 greeter(user); // Hello, Jon Doe
```

Javascript

```
1  
2  
3  
4  
5  
6 var greeter = function (person) {  
7     return "Hello, " + person.firstName + " " + person.lastName;  
8 };  
9  
10 var user = { firstName: 'Jon', lastName: 'Doe' };  
11 greeter(user); // Hello, Jon Doe
```

TypeScript: klasy

```
class Person {  
  
    dateOfBirth: number;          // publiczna własność  
    private verified: boolean;   // prywatna własność  
  
    constructor(  
        name: string,             // definicja parametru  
        public city: string,       // i publicznej własności  
        age?: number              // parametr opcjonalny  
    ) {  
        /* ... */  
    }  
}  
  
var p1 = new Person('Tomek', 'Gdynia', 33);
```

p1.

- ➊ city (property) Person.city: string
- ➋ dateOfBirth

Konstruktor w TS

```
class Square {  
    private radius:  
    number; constructor(radius: number) {  
        this.radius = radius;  
    }  
    get radius(): number {  
        return this._radius;  
    }  
    set radius(value: number) {  
        if (value > 0) {  
            this._radius = value;  
        }  
    }  
  
const square = new Square(5);  
console.log(square.radius);
```

```
class Square {  
  
    number; constructor( private radius: number) {  
        this.radius = radius;  
    }  
    ...  
}
```

Poprzedzenie parametru konstruktora
prawem dostępu tworzy pole klasy i od razu
je inicjalizuje.

Historia wersji Angulara



Biblioteki Angulara

| | |
|-------------------|--------------|
| @angular/core | 2.3.0 |
| @angular/compiler | 2.3.0 |
| | |
| @angular/http | 2.3.0 |
| | |
| @angular/router | 3.3.0 |
| | |

AngularJS (1.x)
Angular (2+)

1. Środowisko Projektowe

Zainstaluj



nodejs.org

Node Package Manager

Angular CLI
Command Line Interface

```
C:\Users\Grzesiek>node --version  
v12.13.0
```

```
C:\Users\Grzesiek>npm -v  
6.12.0
```

```
C:\Users\Grzesiek>npm install -g @angular/cli
```

```
npm install -g @angular/cli
```

```
C:\Users\Grzesiek>ng version
```



```
Angular CLI: 8.3.17  
Node: 12.13.0  
OS: win32 x64  
Angular:  
...  


| Package                    | Version  |
|----------------------------|----------|
| @angular-devkit/architect  | 0.803.17 |
| @angular-devkit/core       | 8.3.17   |
| @angular-devkit/schematics | 8.3.17   |
| @schematics/angular        | 8.3.17   |
| @schematics/update         | 0.803.17 |
| rxjs                       | 6.4.0    |


```



npm - menadżer pakietów dla node.js (dla JavaScript).

- zbiór pakietów/modułów dla Node.js dostępnych online:
<https://www.npmjs.com/>
- narzędzie do instalacji pakietów Node.js, zarządza wersjami i zależnościami pakietów Node.js
- dokumentacja - <https://docs.npmjs.com/>
- najczęściej instalowane pakiety: npm, express, less, browserify, pm2, ...

```
$ npm --version
$ sudo npm install npm -g
$ npm install <Nazwa Modułu>
```

Tworzenie nowej aplikacji w Angular

The screenshot shows the Angular CLI homepage. At the top, there's a navigation bar with links for 'Bezpieczna' (Secure), 'https://cli.angular.io', 'ANGULAR CLI' (with a logo), 'DOCUMENTATION', and 'GITHUB'. A large orange header section contains the text 'Angular CLI jest narzędziem służącym do tworzenia aplikacji Angular i zarządzania nią.' (Angular CLI is a tool for creating Angular applications and managing them). Below this, there's a terminal window showing command-line instructions:

```
> npm install -g @angular/cli  
> ng new my-dream-app  
> cd my-dream-app  
> ng serve
```

To the right of the terminal window, the text 'Angular CLI' is displayed in large white letters, followed by 'A command line interface for Angular' and a 'GET STARTED' button.

ng new [nazwa] – generowanie nowej aplikacji

ng serve - uruchomienie próbne na porcie 4200

ng build - buduje aplikację w katalogu dist/ .

np: ng serve --port = 8080

Generowanie elementów aplikacji:

ng g component [nazwa] - Komponent

ng g directive [nazwa] - Dyrektywa:

ng g pipe [nazwa] - Potok

ng g service [nazwa] - Usługa

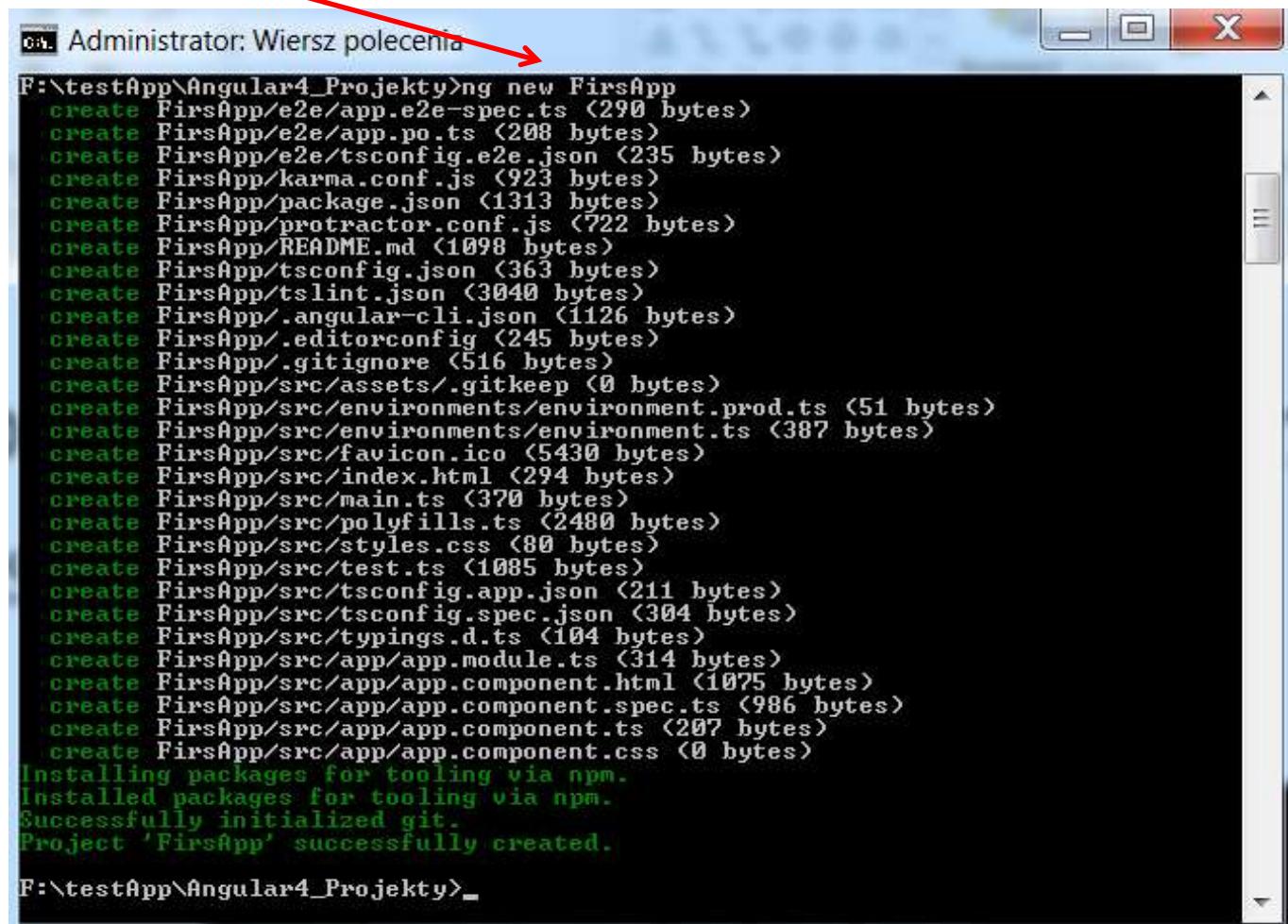
ng g class [nazwa] - Klasa

ng g route [nazwa] - Trasa

Tworzenie nowej aplikacji w Angular

1. Stwórz nowy projekt

ng new projectName



A screenshot of a Windows command prompt window titled "Administrator: Wiersz polecenia". The window shows the output of the command "ng new FirsApp". The output lists the creation of various files and folders, including "e2e", "package.json", "protractor.conf.js", "README.md", "tsconfig.json", "tslint.json", "angular-cli.json", "editorconfig", ".gitignore", "src/assets/.gitkeep", "environments/environment.prod.ts", "environments/environment.ts", "favicon.ico", "index.html", "main.ts", "polyfills.ts", "styles.css", "test.ts", "tsconfig.app.json", "tsconfig.spec.json", " typings.d.ts", "src/app/app.module.ts", "src/app/app.component.html", "src/app/app.component.spec.ts", "src/app/app.component.ts", "src/app/app.component.css", and "node_modules". It also shows the installation of tooling via npm, the successful initialization of git, and the successful creation of the project 'FirsApp'.

```
F:\testApp\Angular4_Projekty>ng new FirsApp
create FirsApp/e2e/app.e2e-spec.ts (290 bytes)
create FirsApp/e2e/app.po.ts (208 bytes)
create FirsApp/e2e/tsconfig.e2e.json (235 bytes)
create FirsApp/karma.conf.js (923 bytes)
create FirsApp/package.json (1313 bytes)
create FirsApp/protractor.conf.js (722 bytes)
create FirsApp/README.md (1098 bytes)
create FirsApp/tsconfig.json (363 bytes)
create FirsApp/tslint.json (3040 bytes)
create FirsApp/.angular-cli.json (1126 bytes)
create FirsApp/.editorconfig (245 bytes)
create FirsApp/.gitignore (516 bytes)
create FirsApp/src/assets/.gitkeep (0 bytes)
create FirsApp/src/environments/environment.prod.ts (51 bytes)
create FirsApp/src/environments/environment.ts (387 bytes)
create FirsApp/src/favicon.ico (5430 bytes)
create FirsApp/src/index.html (294 bytes)
create FirsApp/src/main.ts (370 bytes)
create FirsApp/src/polyfills.ts (2480 bytes)
create FirsApp/src/styles.css (80 bytes)
create FirsApp/src/test.ts (1085 bytes)
create FirsApp/src/tsconfig.app.json (211 bytes)
create FirsApp/src/tsconfig.spec.json (304 bytes)
create FirsApp/src/typings.d.ts (104 bytes)
create FirsApp/src/app/app.module.ts (314 bytes)
create FirsApp/src/app/app.component.html (1075 bytes)
create FirsApp/src/app/app.component.spec.ts (986 bytes)
create FirsApp/src/app/app.component.ts (207 bytes)
create FirsApp/src/app/app.component.css (0 bytes)
Installing packages for tooling via npm.
Installed packages for tooling via npm.
Successfully initialized git.
Project 'FirsApp' successfully created.

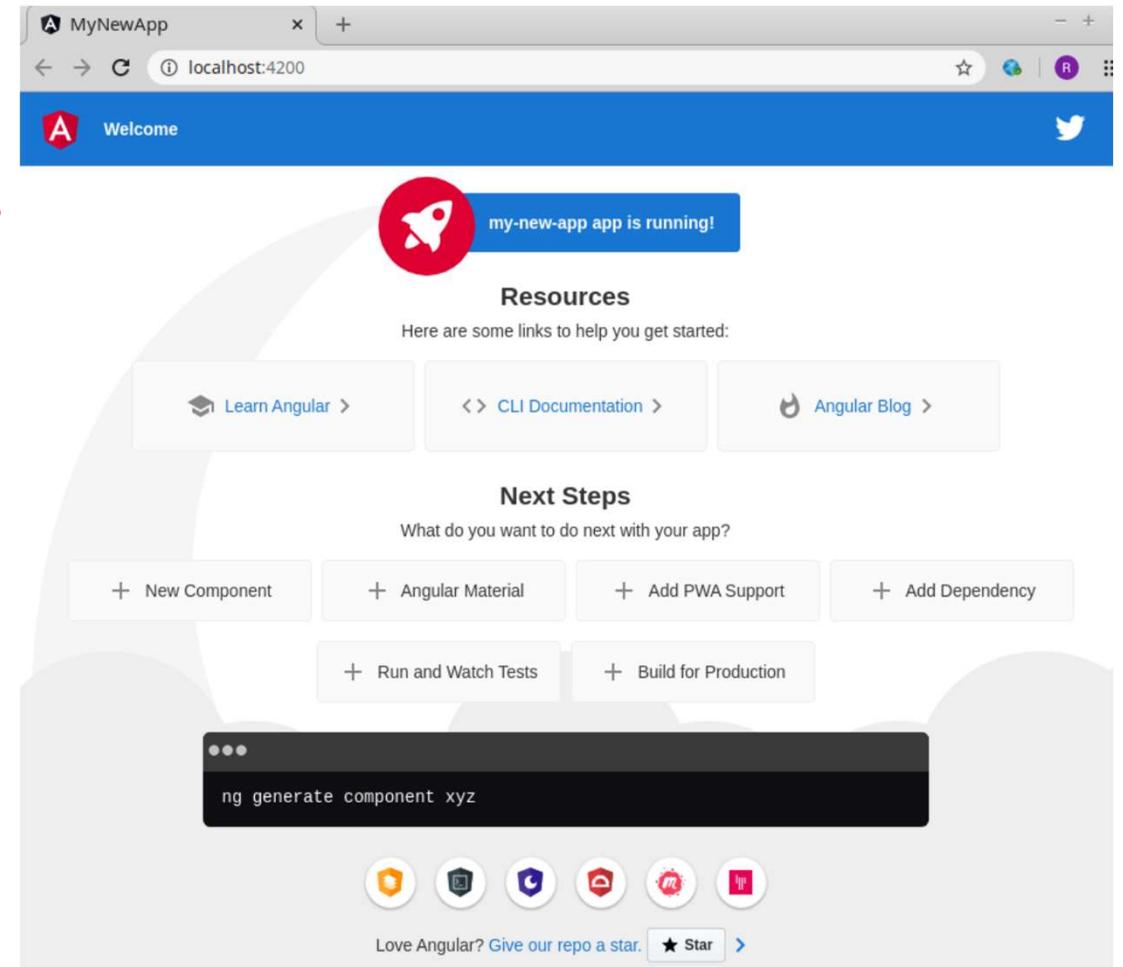
F:\testApp\Angular4_Projekty>
```

Tworzenie nowej aplikacji w Angular

1. Uruchom aplikację

*cd projectName
ng serve*

localhost:4200



Struktura projektu Angular

The screenshot shows the Visual Studio Code interface with the following elements:

- Title Bar:** index.html — PierwszyAngular — Visual Studio Code
- File Menu:** File Edit Selection View Go Debug Tasks Help
- Explorer Bar:** Shows the project structure:
 - OPEN EDITORS: index.html
 - PIERWSZYANGULAR:
 - e2e
 - node_modules
 - src (selected)
 - .angular-cli.json
 - .editorconfig
 - .gitignore
 - karma.conf.js
 - package.json
 - protractor.conf.js
 - README.md
 - tsconfig.json
 - tslint.json
- Editor Area:** Displays the content of index.html, which includes the following code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>PierwszyAngular</title>
    <base href="/">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
    <app-root></app-root>
</body>
</html>
```
- Annotations:**
 - A yellow box labeled "Sekcja dla umieszczania testów" has a red arrow pointing to the "e2e" folder in the Explorer bar.
 - A yellow box labeled "Zaimportowane moduły angulara" has a red arrow pointing to the "node_modules" folder in the Explorer bar.
 - A yellow box labeled "Kod źródłowy aplikacji" has a red arrow pointing to the "src" folder in the Explorer bar.

Każda aplikacja ma przynajmniej 1 moduł i 1 komponent z nim zawarty.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** On the left, it lists the project structure under "PIERWSZYANGULAR". A red arrow points to the "app" folder in the "src" directory.
- Editor:** The main editor area displays the content of "app.component.html".
- Code Content:**

```
1  <!--The content below is only a placeholder and can be replaced.-->
2  <div style="text-align:center">
3      <h1>
4          | Welcome to {{title}}!
5      </h1>
6      
13     <li>
14         <h2><a target="_blank" rel="noopener" href="https://github.com/ang
15     </li>
16     <li>
17         <h2><a target="_blank" rel="noopener" href="https://blog.angular.i
18     </li>
19 </ul>
```

Struktura prostej aplikacji - istotne pliki

Struktura naszej prostej aplikacji - istotne pliki:

```
hello-app
|--- node_modules
|   |--- ...
|
|--- package.json
'--- src
    |--- folderApp
    |   |--- app.component.ts
    |   |--- app.module.ts
    |--- index.html
    |--- main.ts
    |--- styles.css
    |--- systemjs.config.js
    '--- tsconfig.json
```

The diagram illustrates the dependencies between files in the application structure. Blue arrows point from each file to its corresponding dependency number:

- package.json → 1
- index.html → 1
- main.ts → 2
- app.module.ts → 3
- app.component.ts → 4

Najważniejsze pliki

- **index.html** - jest to strona, na której będzie renderowany komponent
- **main.ts** - główny plik uruchomieniowy - łączący komponent ze stroną
- **app/app.module.ts** - struktura modułu (app) - określa użyte komponenty, potoki i usługi
- **app/app.component.ts** - tutaj definiujemy nasz główny komponent,

```
<!DOCTYPE html>
<html>
  <head>
    <title>Pierwsza aplikacja w Angular 4.0</title>
    <base href="/">
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="styles.css">
    <!-- Polyfill(s) for older browsers -->
    <script src="node_modules/core-js/client/shim.min.js"></script>

    <script src="node_modules/zone.js/dist/zone.js"></script>
    <script src="node_modules/systemjs/dist/system.src.js"></script>

    <script src="systemjs.config.js"></script>
    <script>
      System.import('main.js').catch(function(err){ console.error(err); });
    </script>
  </head>
  <body>
    <my-app>Wczytywanie...</my-app>
  </body>
</html>
```

Index.html

Uruchomienie aplikacji

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic'

import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.log(err));
```

- plik app/main.ts zawiera użycie funkcji do uruchomienia procesu ładowania.
- aplikację uruchamiamy przez odpalenie głównego modułu.
- funkcja uruchamiająca aplikację zależy od platformy, nie jest więc w @angular/core.
- w procesie ładowania możemy zaimportować platformę, z której chcielibyśmy korzystać, w zależności od środowiska, w którym działamy (NativeScript, Cordova, przeglądarka itp.)

app/main.ts

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic'

import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.log(err));
```



Angular Bootstrapper

app/app.module.ts

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

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

app/AppComponent.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app'
  template: "<p>Witaj w moim projekcie... </p>"
})
export class AppComponent {
```

App module

Root component

Module

component

Module

component

Module

component

Kompilujemy i uruchamiamy aplikację:

```
npm start
```



Witaj w moim projekcie...

package.json - podstawowy zbiór pakietów

package.json

```
{  
  "name": "angular-quickstart",  
  "version": "1.0.0",  
  "description": "QuickStart package.json from the documentation, supplemented with testing support",  
  "scripts": {  
    ...  
  },  
  "license": "MIT",  
  "dependencies": {  
    "@angular/common": "^4.0.0",  
    "@angular/compiler": "^4.0.0",  
    "@angular/core": "^4.0.0",  
    "@angular/forms": "^4.0.0",  
    "@angular/http": "^4.0.0",  
    "@angular/platform-browser": "^4.0.0",  
    "@angular/platform-browser-dynamic": "^4.0.0",  
    "@angular/router": "^4.0.0",  
  
    "angular-in-memory-web-api": "^0.3.0",  
    "systemjs": "0.19.40",  
    "core-js": "^2.4.1",  
    "rxjs": "5.0.1",  
    "zone.js": "^0.8.4"  
  },  
  "devDependencies": {  
    ...  
  }  
}
```

Tworzenie pakietu, instalacja

- Instalacja pakietów

`cd rest`

`npm init` - inicjalizacja, tworzy plik package.json

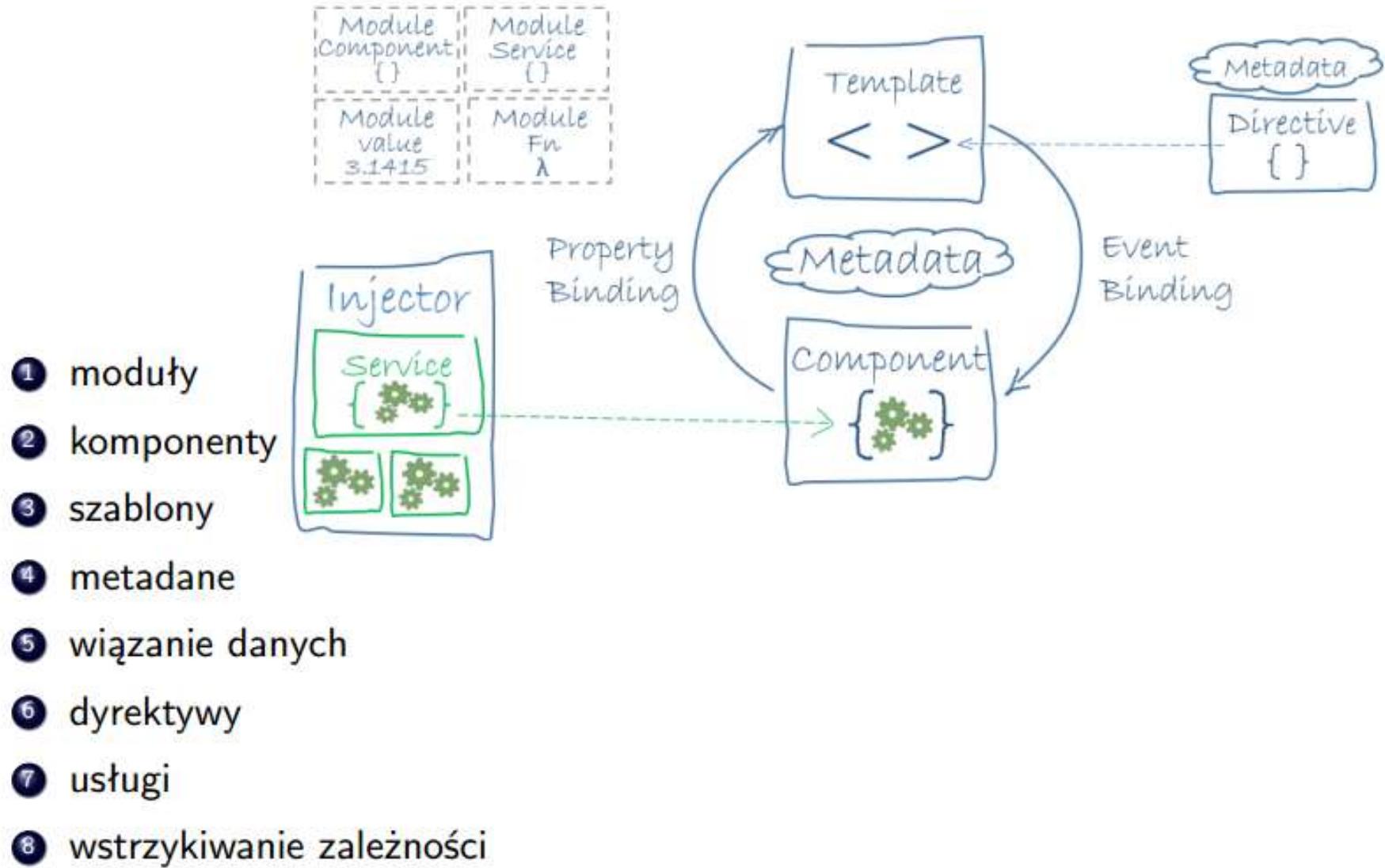
`npm install --save nazwapakietu` - dodaje zależności do pliku package.json:

```
...
"dependencies": {
    "nazwapakietu": "^4.13.4",
    ....
}
```

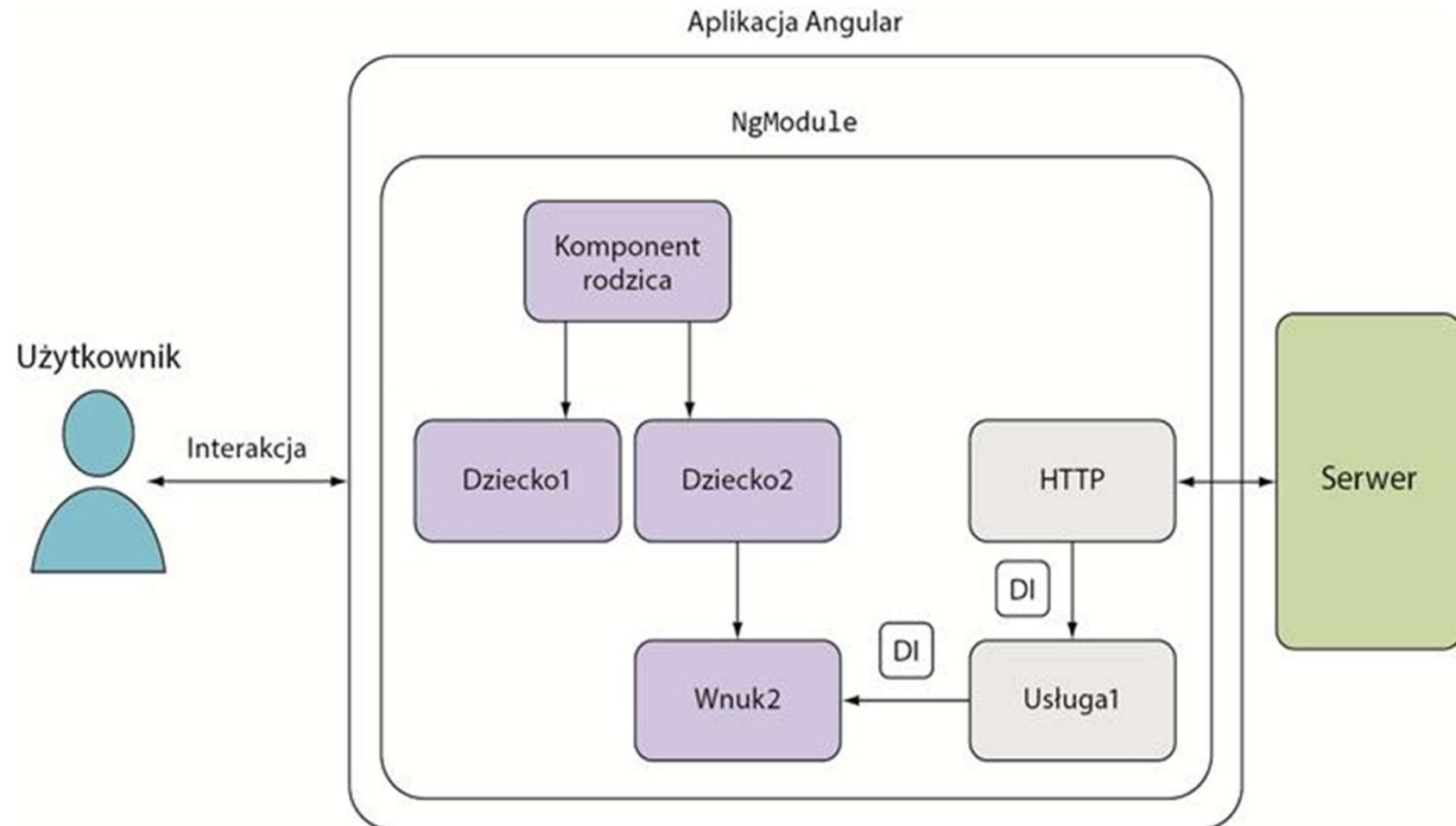
- Ponowna instalacja pakietów np. po przeniesieniu projektu

`npm install`

Architektura Angular - 8 głównych bloków konstrukcyjnych



Architektura Angulara w praktyce



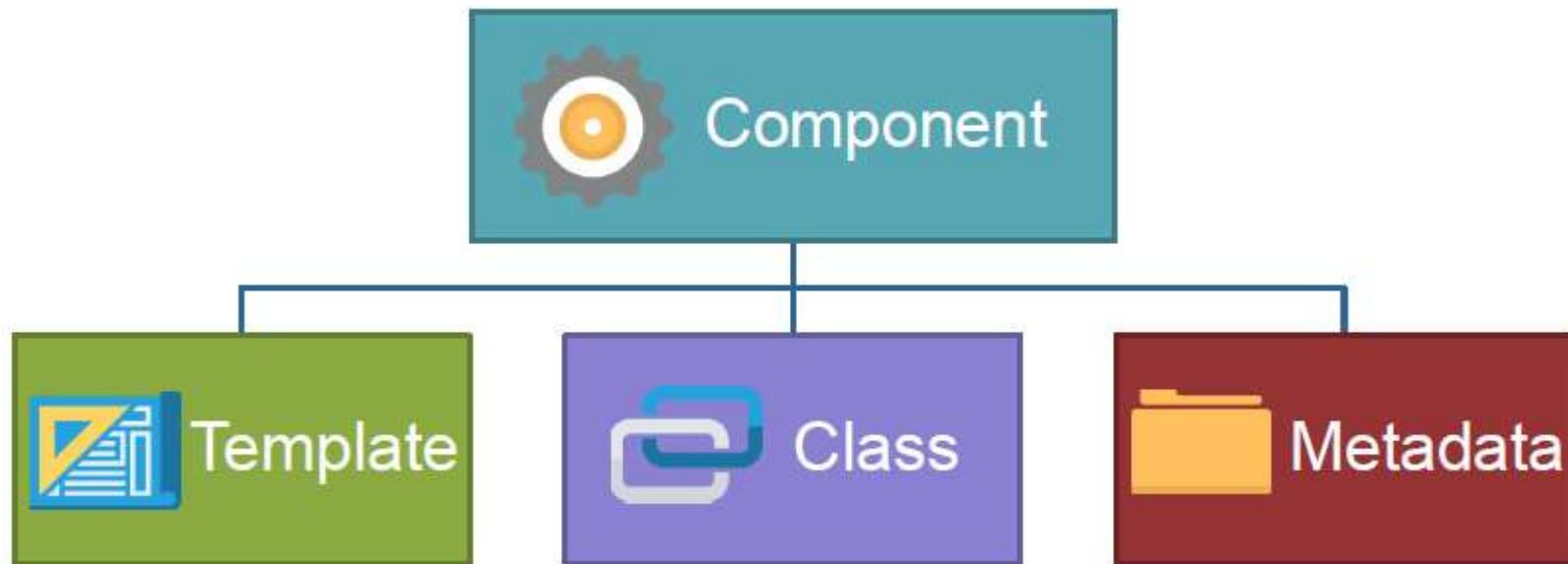
KOMPONENTY



- Podstawowa koncepcja aplikacji w Angular
- Reużywalna blok składowy UI
- Cała aplikacja jest drzewem komponentów

Czym jest komponent w Angular

- Komponenty są podstawowymi blokami konstrukcyjnymi aplikacji Angular.
- Kontrolują jakiś obszar ekranu - widok - poprzez związany z nimi szablon.
- Wewnątrz klasy komponentu definiujemy logikę aplikacji - określamy jak komponent obsługuje widok.
- Klasa komponentu komunikuje się z widokiem poprzez API pól i metod.





Komponenty

"Komponent kontroluje wycinek obszaru ekranu, który możemy nazwać widokiem i definiuje reużywalne bloki składowe UI w aplikacji."



```
import { Component } from '@angular/core';
@Component({
  selector: 'my-component',
  template: `<div>Czesc jestem {{name}}.
    <button (click)="czesc()">Witaj</button>
  </div>`
})
export class MyComponent {
  constructor() {
    this.name = 'Grzegorz'  }
  czesc() {
    console.log('Mam na imie ', this.name)
  }
}
```

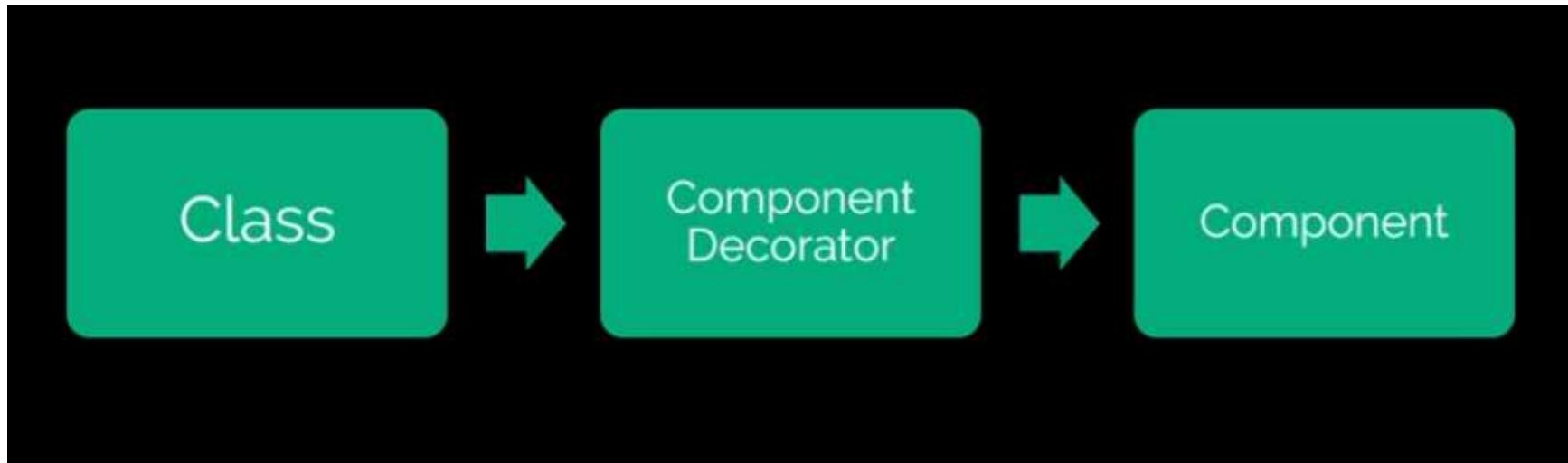
Przykład komponentu

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-component',
  template: `<div>Witam – jestem {{name}}.  
  <button (click)="start()"> test </button></div>`  
})
export class MyComponent {  
  constructor() {  
    this.name = 'Grzegorz'  
  }  
  start() {  
    console.log('to ja', this.name)  
  }  
}
```

Praca z komponentami

1. Stworzenie komponentu (ręcznie lub za pomocą generatora)
2. Zarejestrowanie komponentu w module (samodzielne lub automatyczne)
3. Dodanie elementów do szablonu HTML



1. Tworzymy nowy komponent

- `ng g ComponentName`

Wygenerowane pliki

```
B:\Angular4_Projekty\PierwszyAngular>ng g component NowyKomponent
create  src/app/nowy-komponent/nowy-komponent.html (33 bytes)
create  src/app/nowy-komponent/nowy-komponent.component.spec.ts (678 bytes)
create  src/app/nowy-komponent/nowy-komponent.component.ts (300 bytes)
create  src/app/nowy-komponent/nowy-komponent.component.css (0 bytes)
update  src/app/app.module.ts (488 bytes)
```

2. Rejestracja komponentu

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

import { AppComponent } from './app.component';
import { NowyKomponentComponent } from './nowy-komponent/nowy-komponent.component';

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

Komponent w Angular

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

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

  constructor() { }

  ngOnInit() {
  }
}
```

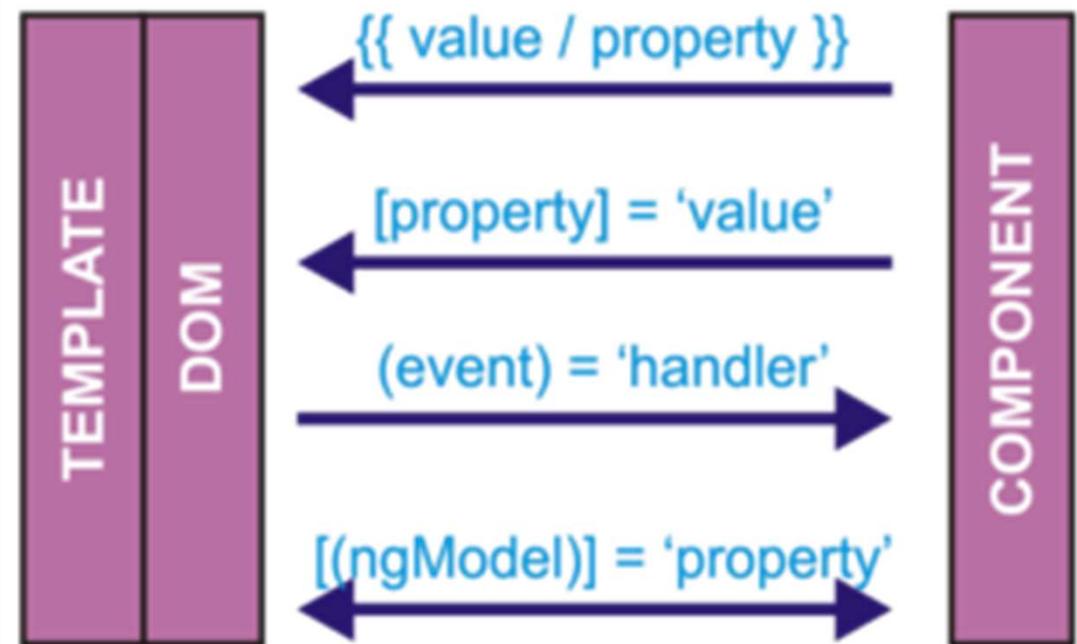
Typy wiązania danych

Bez wykorzystania frameworka sami jesteśmy odpowiedzialni za:

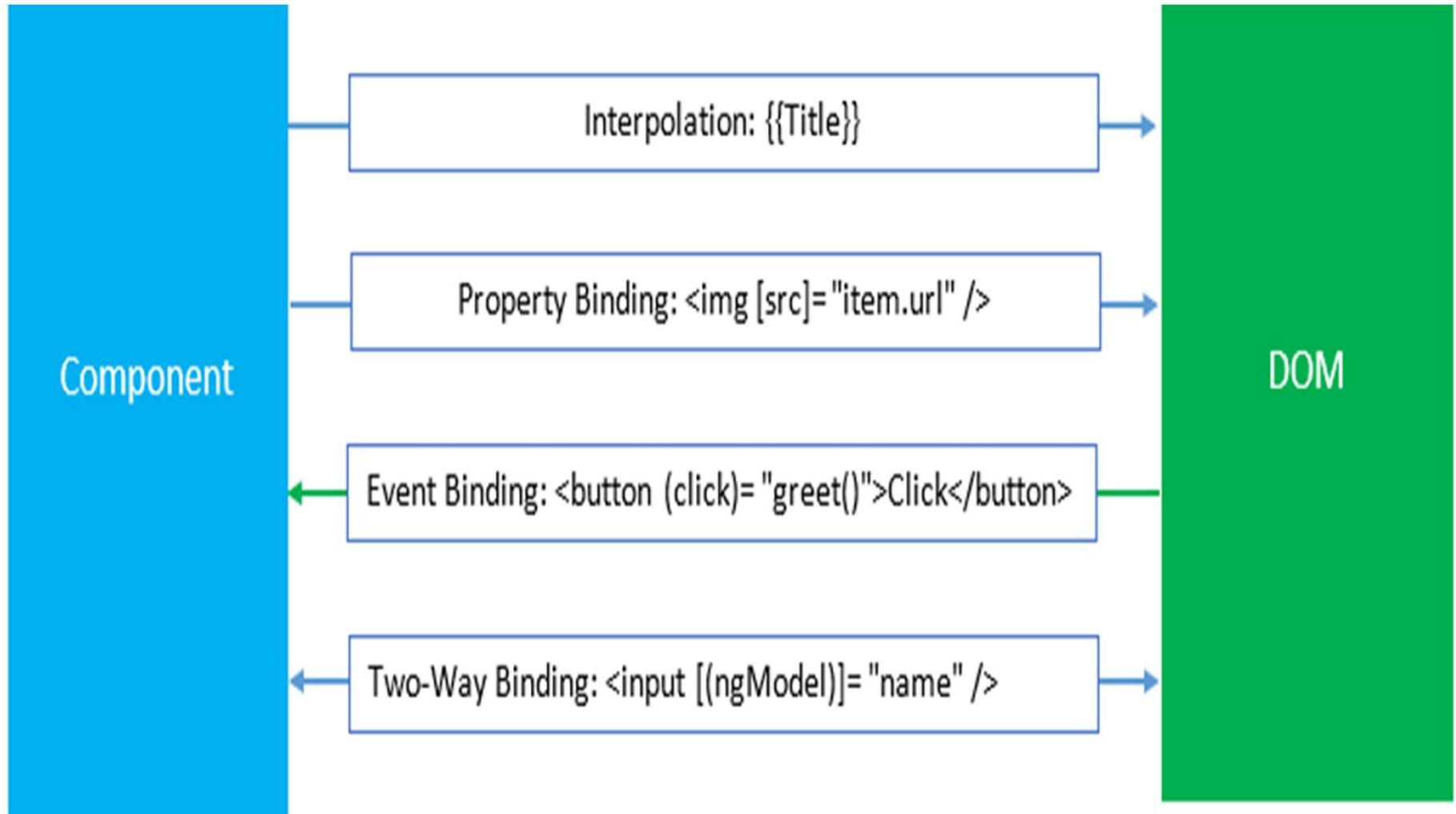
- wstawianie danych do kontrolek HTML
- zamianę reakcji użytkownika w akcje i aktualizacje wartości.

Pisanie takiej logiki wstaw/wyciągnij ręcznie jest uciążliwe, sprzyja błędom i jest koszmarem przy czytaniu kodu.

- Angular wspiera wiązanie danych, mechanizm wiążący część szablonu z częściami komponentu.
- Dodajemy oznaczenia wiązania danych w szablonie HTML, zeby określić jak Angular ma łączyć obie strony.



Wiązanie danych



1-way Binding

Interpolation



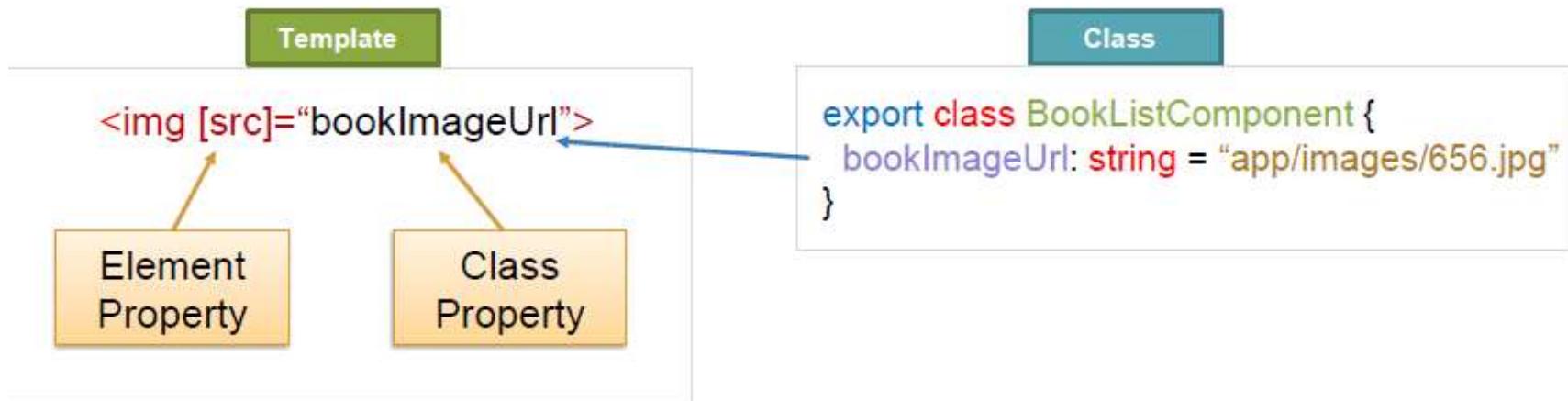
safe navigation operator (?)

`<p> Moje imię to: {{item?.name}}</p>`

gdy item = null wyświetli się pusty tekst –
W consoli nie będzie błędów, związanych z
odwołaniem do obiektu pustego

1-way Binding

Property Binding



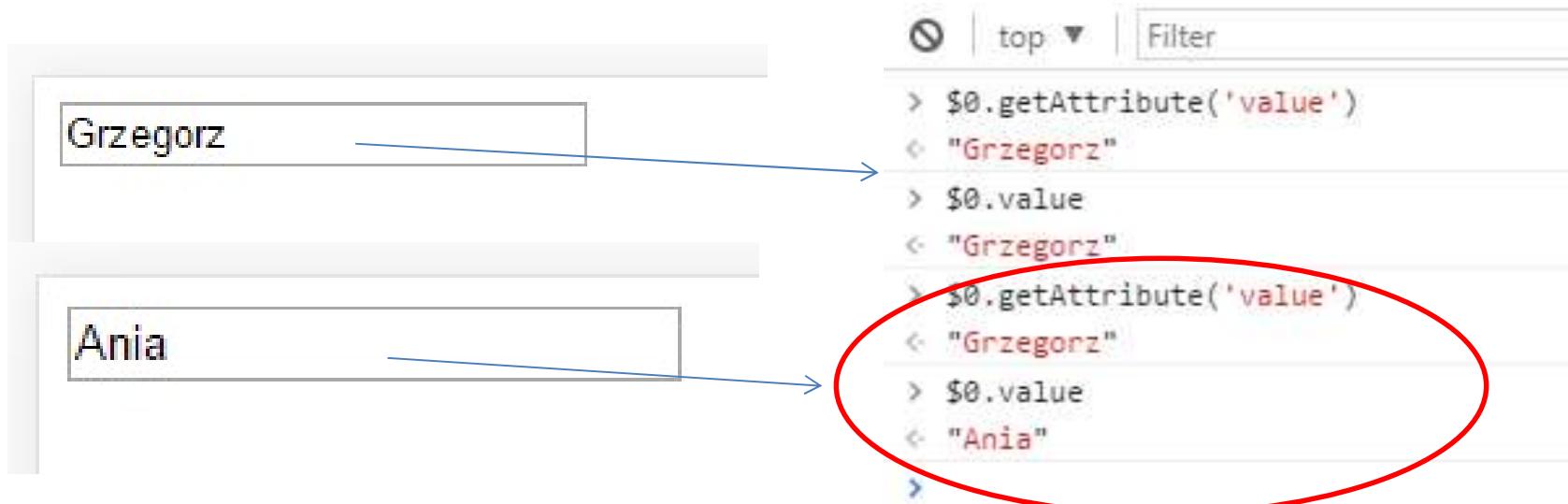
Property Binding

Atrybuty != Property
HTML DOM

Nie ulegają zmianie
po inicjalizacji

Zmieniają wartość
inicjalizowane przez atrybut

```
<input id="test1" type="text" value="Grzegorz" >
```



Event Binding

Template

```
(click)="hidelImage()"
```



Class

```
export class AppComponent {  
  hidelImage(): void {  
    this.showImage = !this.showImage;  
  }  
}
```

Binding - Widok do modelu danych

Podpięcie zdarzenia:

```
<div class="container">  
  
  <button type="button" (click)="clickListener($event)">Button</button>  
  
</div>
```

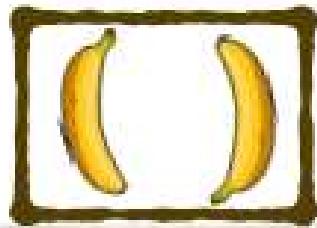
Naciśnięcie przycisku spowoduje wywołanie metody `clickListener($event)` i przekazanie parametru `$event` jako argument funkcji. Podpiąć możemy wszystkie dostępne standardowo zdarzenia drzewa DOM

```
<button type="button" (click) = "clickListener($event)" > Przycisk  
1</button>  
<button type="button" on-click = "clickListener($event)"> Przycisk  
2</button>
```

Dwukierunkowe wiązanie danych

- Połączenie komunikacji "z" i "do"
- Wykorzystujemy do tego dyrektywę `ngModel`

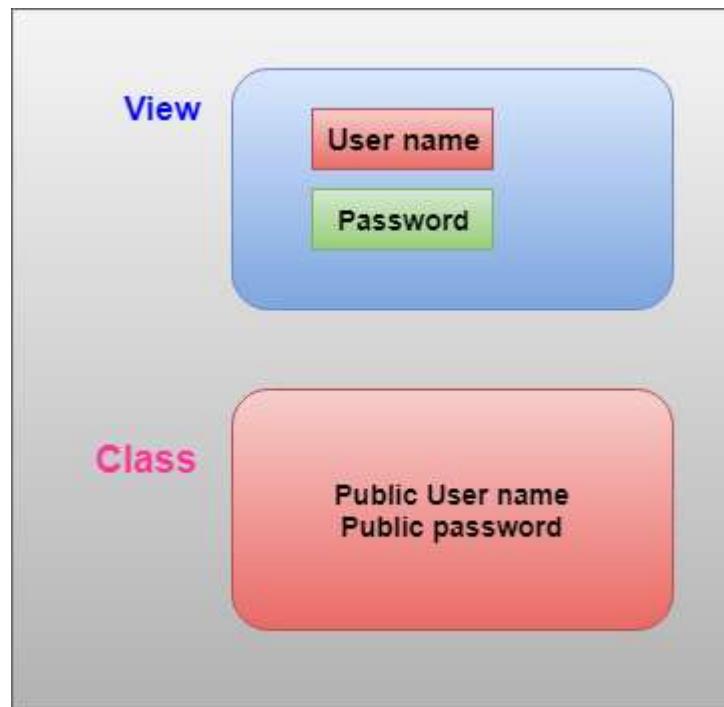
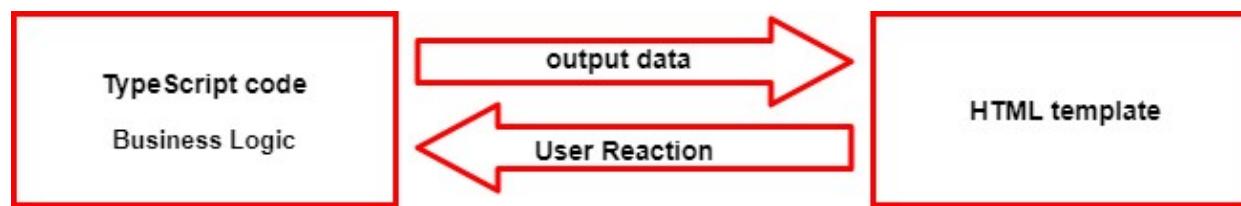
```
<input [ngModel]="name" (ngModelChange)="name=$event">  
  
// Skrócony zapis:  
<input [(ngModel)]="name">
```



[(ngModel)]

Dwukierunkowe wiązanie danych

`[(ngModel)]="[{właściwość Komponentu}]"`



```
<h2>Two-way Binding Przykład</h2>
<input [(ngModel)]="fullName" />
<br/><br/>
<p> {{fullName}} </p>
```

Cykl życia

constructor

ngOnChanges

Zdarzenie wywoływanie przy każdej zmianie składowych `@Input()`

ngOnInit

Zdarzenie wywoływanie po inicjalizacji składowych `@Input()`,
po pierwszym zdarzeniu `ngOnChanges()`

ngDoCheck

Zdarzenie wywoływanie przy każdorazowej detekcji zmian
składowych komponentu (po wykryciu każdej zmiany)

ngAfterContentInit

Zdarzenie wywoływanie po zainicjowaniu zawartości komponentu

ngAfterContentChecked

po każdym sprawdzeniu zawartości komponentu

ngAfterViewInit

po zainicjowaniu widoków komponentu

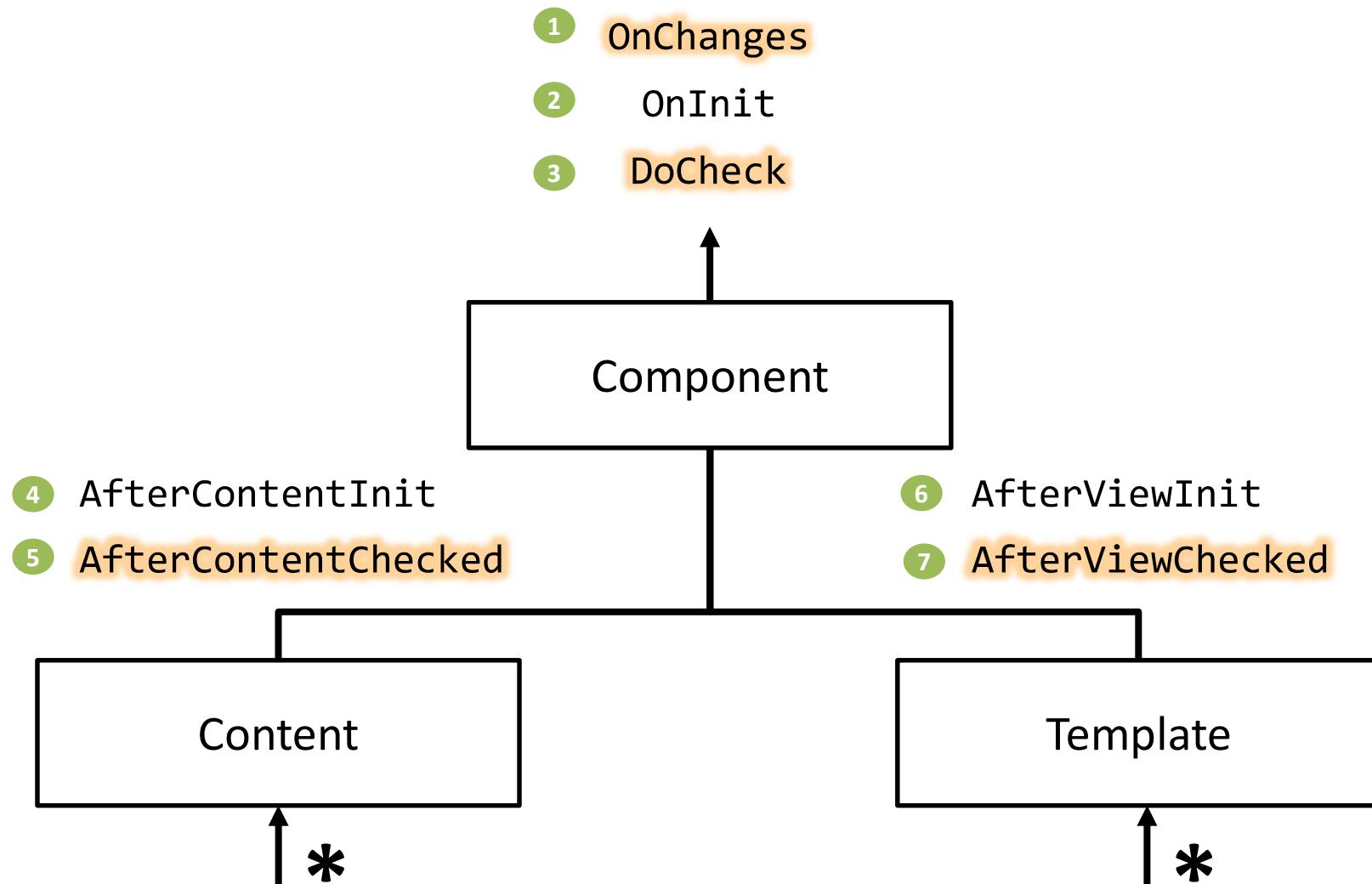
ngAfterViewChecked

po każdym sprawdzeniu widoku (ów) komponentu

ngOnDestroy

tuż przed zniszczeniem komponentu

Kolejność wywoływania



books-list.component.ts

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

```
@Component({  
  selector: 'bs-books-list'  
})
```

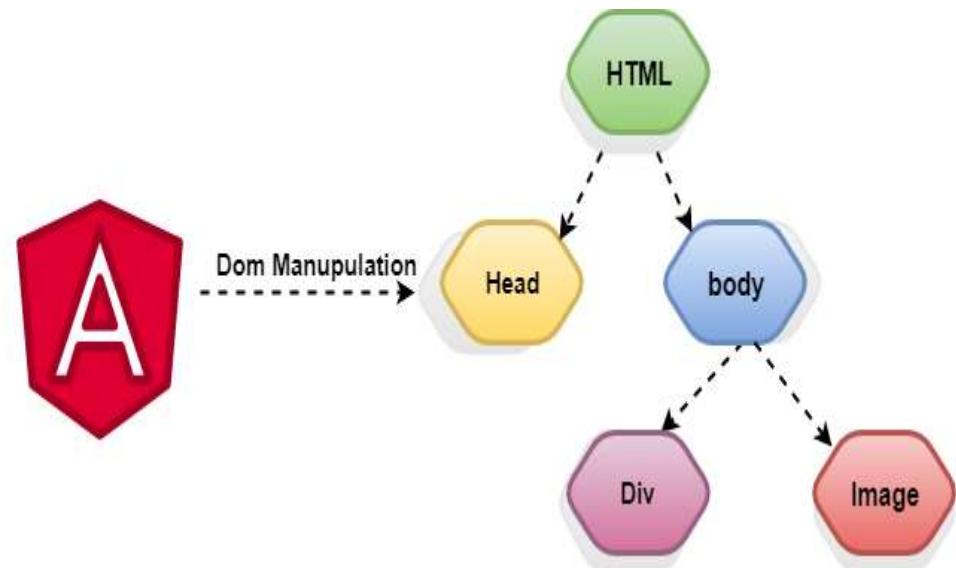
```
export class BooksListComponent implements OnInit {
```

```
  ngOnInit(): void {  
    console.log('Inside OnInit');  
  }  
}
```



Dyrektywy

Dyrektywa modyfikuje DOM zmieniając jego wygląd lub zachowanie



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.

Typy Dyrektyw

Component

- <bs-app></bs-app>

welcome.component.ts

```
@Component({  
  selector: 'bs-welcome'  
})
```

Structural

- NgIf, NgFor, NgStyle

app.component.ts

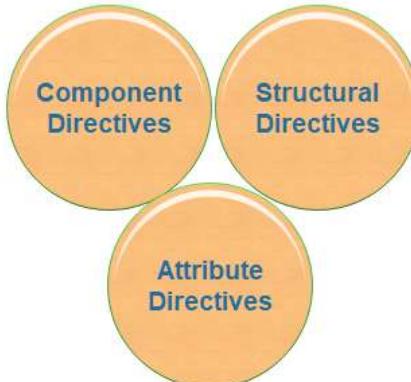
```
@Component({  
  selector: 'bs-app',  
  template: <bs-welcome></bs-welcome>  
})
```

Attribute

- <p highlight></p>

Modyfikuje strukturę DOM

Modyfikuje atrybuty elementów DOM



Wyświetlanie lub ukrywanie elementów DOM

```
<p> Wersja z atrybutem hidden</p>
<div [hidden]="ksiazki.length == 0" >
    lista Ksiazek
</div>
<div [hidden]="ksiazki.length > 0">
    Brak ksiazek na liscie
</div>
```

Atrybut [hidden]

```
<div *ngIf="ksiazki.length > 0" >
    lista Ksiazek
</div>

<div *ngIf="ksiazki.length == 0">
    Bark ksiazek na liscie
</div>
```

Dyrektywa *ngIf

ngIf="<condition>*"

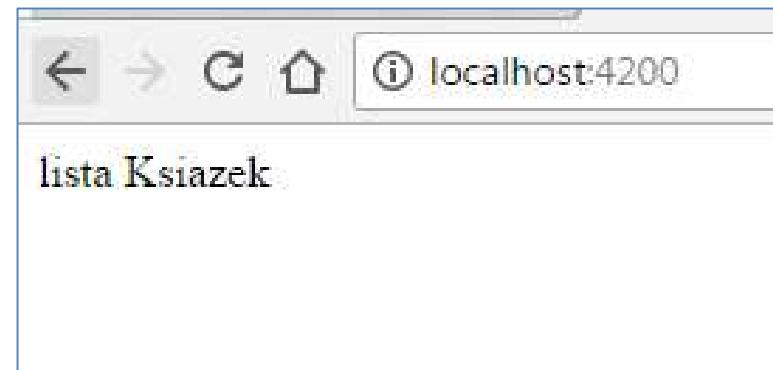
books-list.component.html

```
<img *ngIf="showImage">
```

showImage –
wartosc boolowska

ngIf w praktyce – sposoby implementacji

```
<div *ngIf="ksiazki.length > 0">  
    lista Ksiazek  
</div>  
  
<div *ngIf="ksiazki.length == 0">  
    Brak ksiazek na liscie  
</div>
```



```
<div *ngIf="ksiazki.length > 0; else nobooks">  
    lista Ksiazek  
</div>  
  
<ng-template #nobooks>  
    Brak ksiazek na liscie  
</ng-template>
```

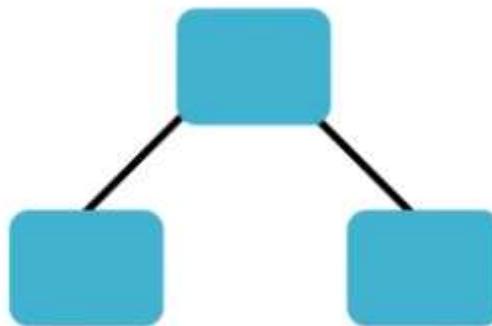
```
export class PierwszyPrzykladDyrektywComponent implements OnInit {  
    ksiazki = ["Pierwsza ksiazka", "Druga ksiazka", "Trzecia ksiazka"];
```

```
<div *ngIf="ksiazki.length > 0; then booksList else nobooks" ></div>  
<ng-template #bookList>  
    lista Ksiazek  
</ng-template>  
<ng-template #nobooks>  
    Brak ksiazek na liscie  
</ng-template>
```

Zarządzanie modelem DOM – kiedy która technika

[hidden]

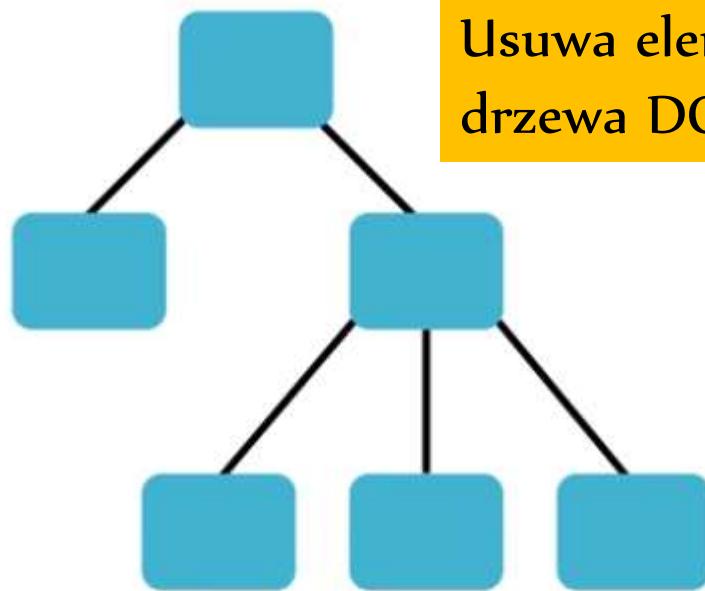
Ukrywa element w DOM



Dla drzewa z małą
ilością elementów

*ngIf

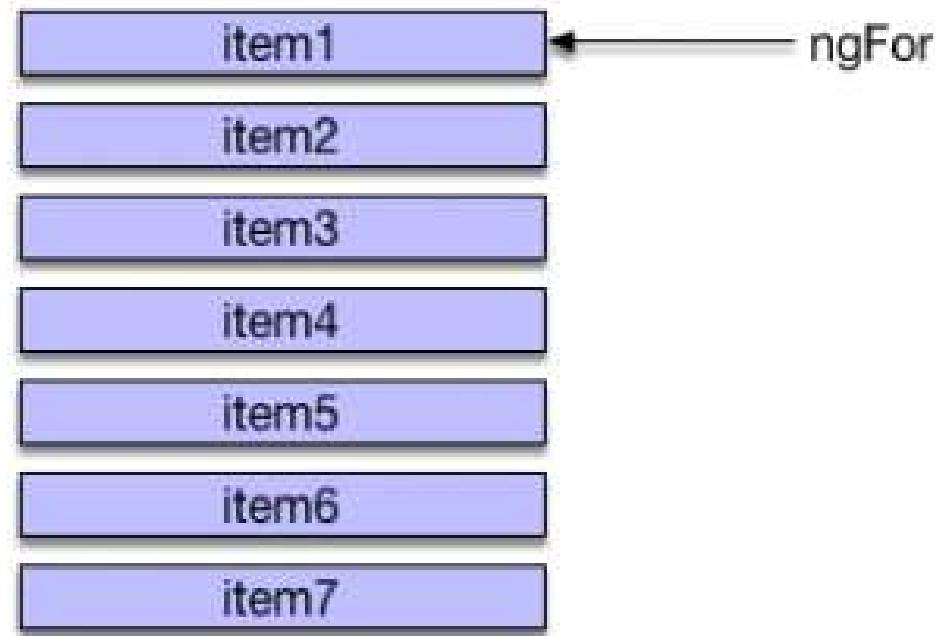
Usuwa element z
drzewa DOM



Dla drzewa z dużą
ilością elementów

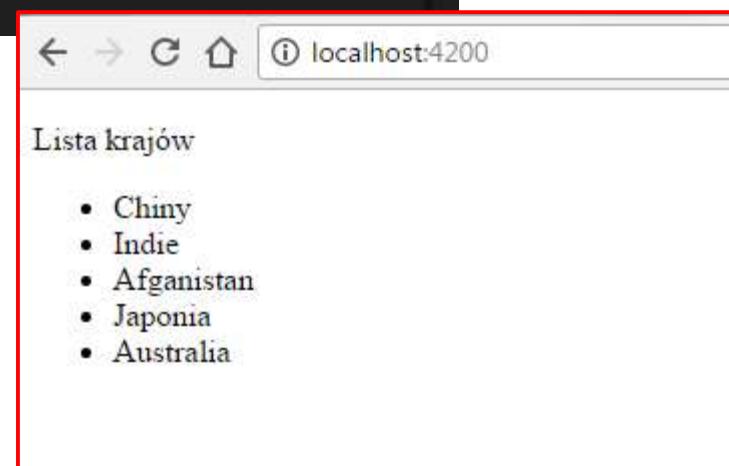
Angular *ngFor Directive

```
<li *ngFor="let item of items ;">....</li>
```



Wyświetlanie zawartości tablicy - ngFor

```
@Component({
  selector: 'app-root',
// templateUrl: './app.component.html',
  template: `
    <p> Lista krajów </p>
    <ul>
      <li *ngFor="let tab of tablica"> {{tab}} </li>
    </ul> `,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  tablica = ["Chiny", "Indie", "Afganistan", "Japonia", "Australia"]
}
```



Wypisanie tablicy – realizacja w komponencie

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

export class AppComponent {
  name = 'Grzegorz Rogus';

  ksiazki = [{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.ksiazki;
  }
}

<ul>
  <li *ngFor="let ksiazka of getKsiazki()">
    {{ksiazka.title}}  w cenie  {{ksiazka.price}}
  </li>
</ul>
```

Komponent w Angular10

lub tak

```
<div style="text-align:center">
  <h1>
    Witaj {{name}}!
  </h1>
</div>

<p>
  Ksiazki warte polecenia na temat technologii Webowych:
</p>
<ul>
  <li *ngFor="let ksiazka of ksiazki">
    {{ksiazka.title}}  w cenie  {{ksiazka.price}}
  </li>
</ul>
```



Witaj Grzegorz Rogus!

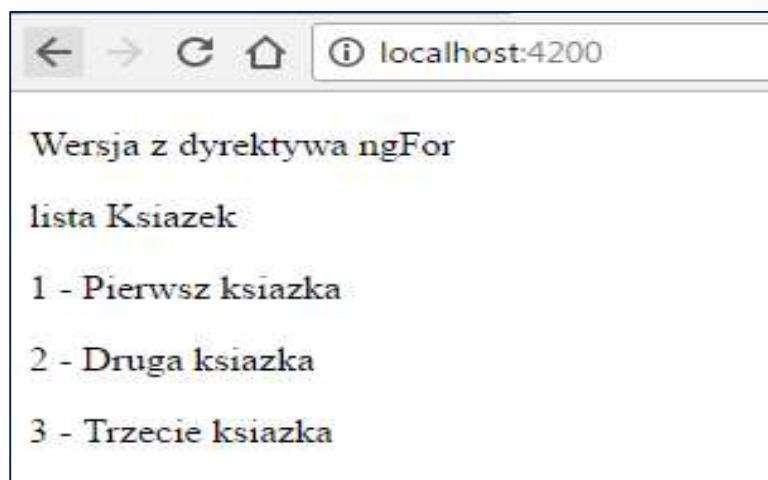
Książki warte polecienia na temat technologii Webowych:

- Node.js, MongoDB, AngularJS. Kompendium wiedzy w cenie 99
- Tworzenie gier internetowych. Receptury w cenie 49
- Web 2.0 Architectures. What entrepreneurs and information architects need to know w cenie 84.92
- React dla zaawansowanych w cenie 45
- Spring MVC 4. Projektowanie zaawansowanych aplikacji WWW w cenie 102

Wyświetlanie zawartości tablicy - ngFor

```
<p> Wersja z dyrektywa ngFor</p>
<div *ngIf="ksiazki.length > 0; then booksList else nobooks" ></div>
<ng-template #booksList>
  lista Ksiazek
  <div *ngFor="let book of ksiazki, index as i">
    <p> {{i+1}} - {{book}} </p>
  </div>
</ng-template>
<ng-template #nobooks>
  Brak ksiazek na liscie
</ng-template>
```

- index: number : The index of the current item in the iterable.
- first: boolean : True when the item is the first item in the iterable
- last: boolean : True when the item is the last item in the iterable.
- even: boolean : True when the item has an even index in the iterable
- odd: boolean : True when the item has an odd index in the iterable



Źródła zmian w systemie

DOM
Events

AJAX
Requests

Timers



Change
Detection

```
export class PierwszyPrzykladDyrektywComponent {  
  ksiazki = ["Pierwsz ksiazka", "Druga ksiazka", "Trzecie ksiazka"];  
  
  onAdd() {  
    this.ksiazki.push("Czwarta ksiazka");  
  }  
}
```

Dodaj lista Ksiazek

1 - Pierwsz ksiazka

2 - Druga ksiazka

3 - Trzecie ksiazka

4 - Czwarta ksiazka

```
<button (click)="onAdd()">Dodaj</button>  
<div *ngIf="ksiazki.length > 0; then booksList else nobooks" ></div>  
<ng-template #booksList>  
  lista Ksiazek  
  <div *ngFor="let book of ksiazki, index as i, even as isEven">  
    <p> {{i+1}} - {{book}} </p>  
  </div>  
</ng-template>  
<ng-template #nobooks>  
  Brak ksiazek na liscie  
</ng-template>
```

```
export class PierwszyPrzykladDyrektywComponent {
  ksiazki = ["Pierwsza ksiazka", "Druga ksiazka", "Trzecie ksiazka"];

  onAdd() {
    this.ksiazki.push("Czwarta ksiazka");
  }
  onRemove(book) {
    let index = this.ksiazki.indexOf(book);
    this.ksiazki.splice(index, 1);
  }
}
```

Wersja z dyrektywą ngFor

Dodaj lista Ksiazek

1 - Druga ksiazka

Usun

2 - Trzecie ksiazka

Usun

3 - Czwarta ksiazka

Usun

```
<button (click)="onAdd()">Dodaj</button>
<div *ngIf="ksiazki.length > 0; then booksList else nobooks" ></div>
<ng-template #booksList>
  lista Ksiazek
  <div *ngFor="let book of ksiazki, index as i, even as isEven">
    <p> {{i+1}} - {{book}} </p>
    <button (click)="onRemove(book)">Usun</button>
  </div>
</ng-template>
<ng-template #nobooks>
  Brak ksiazek na liscie
</ng-template>
```

Dynamiczna stylizacja komponentów

Dyrektywy atrybutowe

- ngStyle:

```
<div [ngStyle]="{'font-size': mySize+'px'}">...</div>
<div [ngStyle]="{'font-size.px': mySize}">...</div>
<div [ngStyle]="myStyle">...</div>
```

- ngClass:

```
<div [ngClass]="'first second'">...</div>
<div [ngClass]=["'first', 'second']">...</div>
<div [ngClass]={first: true, second: -1, third: 0}>...
```

Pozwalają na dynamiczną stylizację poszczególnych elementów DOM

ngStyle

Statyczne przypisanie

```
<div [ngStyle] = "{'background-color': 'green'}"></<div>
```

```
<div [ngStyle] = "{'background-color': ksiazki.length == 3 ? 'green' : 'red'}">  
</<div>
```

Dynamiczne przypisanie

```
<div [ngStyle] = "{property : value}"></<div>
```

tak albo tak

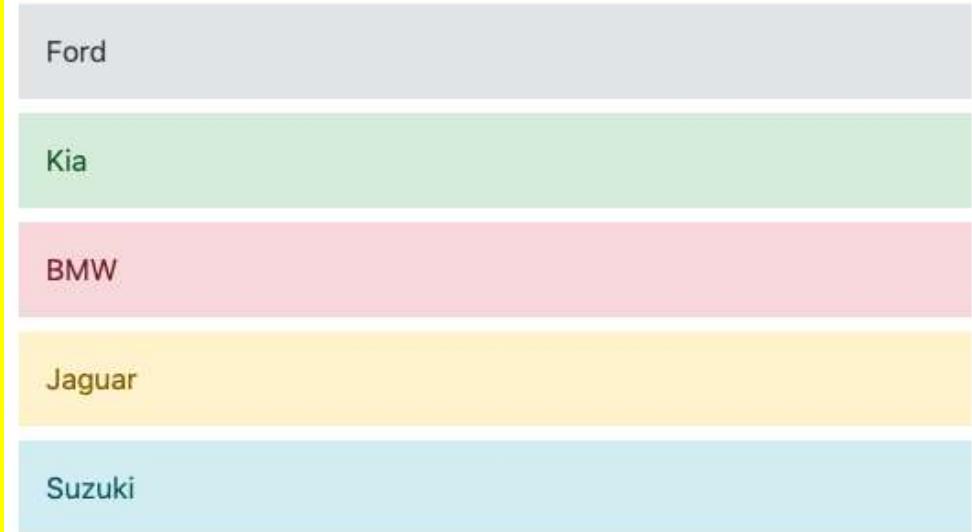
```
<div [style.property] = "{value}"></<div>
```

```
<ul *ngFor="let person of ludzie">  
  <li [ngStyle] = "{'font-size.px': 24}"  
    [style.color] = "getColor(person.kraj)">  
    {{ person.imie }} ({{ person.kraj }}) </li> </ul>
```

Angular ngClass Directive

```
<element ng-class=""expression""></element>
```

```
<div class="container">
<div *ngFor="let car of cars" [ngClass] ="{ 
  'alert-secondary':car.name==='Ford',
  'alert-success':car.name==='Kia',
  'alert-danger':car.name==='BMW',
  'alert-warning':car.name==='Jaguar',
  'alert-info':car.name==='Suzuki'
}>
  {{car.name}}
</div>
</div>
```



ngClass

books-list.component.html

```
<div [ngClass]={"'redClass': showImage, 'yellowClass': !showImage}">
```

Dodaje lub usuwa klasy CSS dla elementu HTML w zależności od wartości zmiennej

Rotacja klasy elementu (klasa underline jest dodana tylko wtedy gdy isUnderlined = true):

```
<div class="container">  
  <p [class.underline]="isUnderlined"></p>  
</div>
```

ngClass – Zarządzanie wieloma klasami

```
public getClassNames() {  
  
    return {  
  
        'underline': this.isUnderlined,  
  
        'active': this.isActive  
  
    }  
}
```

Natomiast w szablonie wystarczy:

```
<div class="container">  
  
    <p [ngClass]="getclassNames()"></p>  
  
</div>
```

ngStyle ngClass - przykład zastosowania

test dyrektyw ngStyle ngClass

Dodaj

lista Ksiazek 2

- 1 - Pierwsza ksiazka Usun
- 2 - Czwarta ksiazka Usun

test dyrektyw ngStyle ngClass

Dodaj

lista Ksiazek 5

- 1 - Pierwsza ksiazka Usun
- 2 - Druga ksiazka Usun
- 3 - Trzecie ksiazka Usun
- 4 - Czwarta ksiazka Usun
- 5 - Czwarta ksiazka Usun

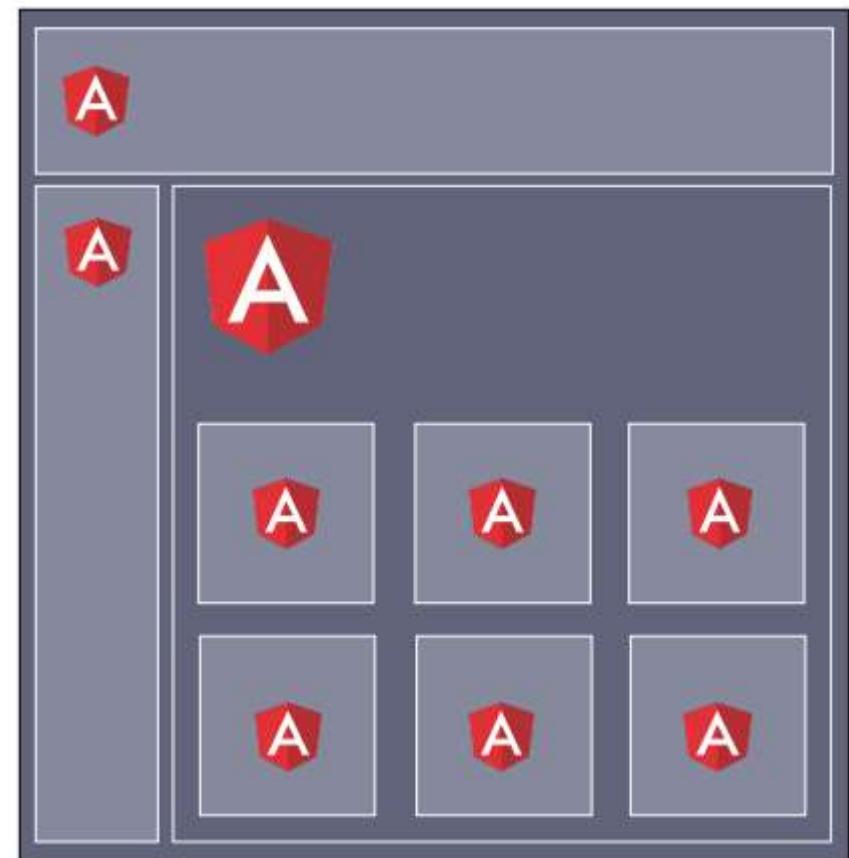
```
getColor(): string {
  return this.ksiazki.length > 3 ? 'red' : 'green';
}
```

```
<p [ngStyle] ="{'color': getColor()}"> lista Ksiazek {{ksiazki.length}} </p>
<ul>
  <div *ngFor="let book of ksiazki, index as i, even as isEven">
    <li [ngClass] = "{ 'oddtest': isEven, 'eventest': !isEven }" >
      {{i+1}} - {{book}} <button (click) = "onRemove(book)"> Usun </button>
    </li>
  </div>
</ul>
</ng-template>
```

```
.oddtest{
  background-color: grey;
}
.eventest {
  background-color: linen;
}
```

Aplikacja składa się z komponentów

W Angular aplikacja zbudowana jest z drzewa komponentów. Każdy komponent może mieć zestaw komponentów „dzieci” oraz rodzica. Naszym głównym komponentem jest jego korzeń, tzw. root component.



Połączenie pomiędzy komponentami

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

POTOMEK

Dodaj selektor
komponentu dziecka w
szablonie rodzica

```
@Component({  
  selector: "app",  
  template: "<my-child></my-child>",  
})  
export class AppComponent {}
```

RODZIC

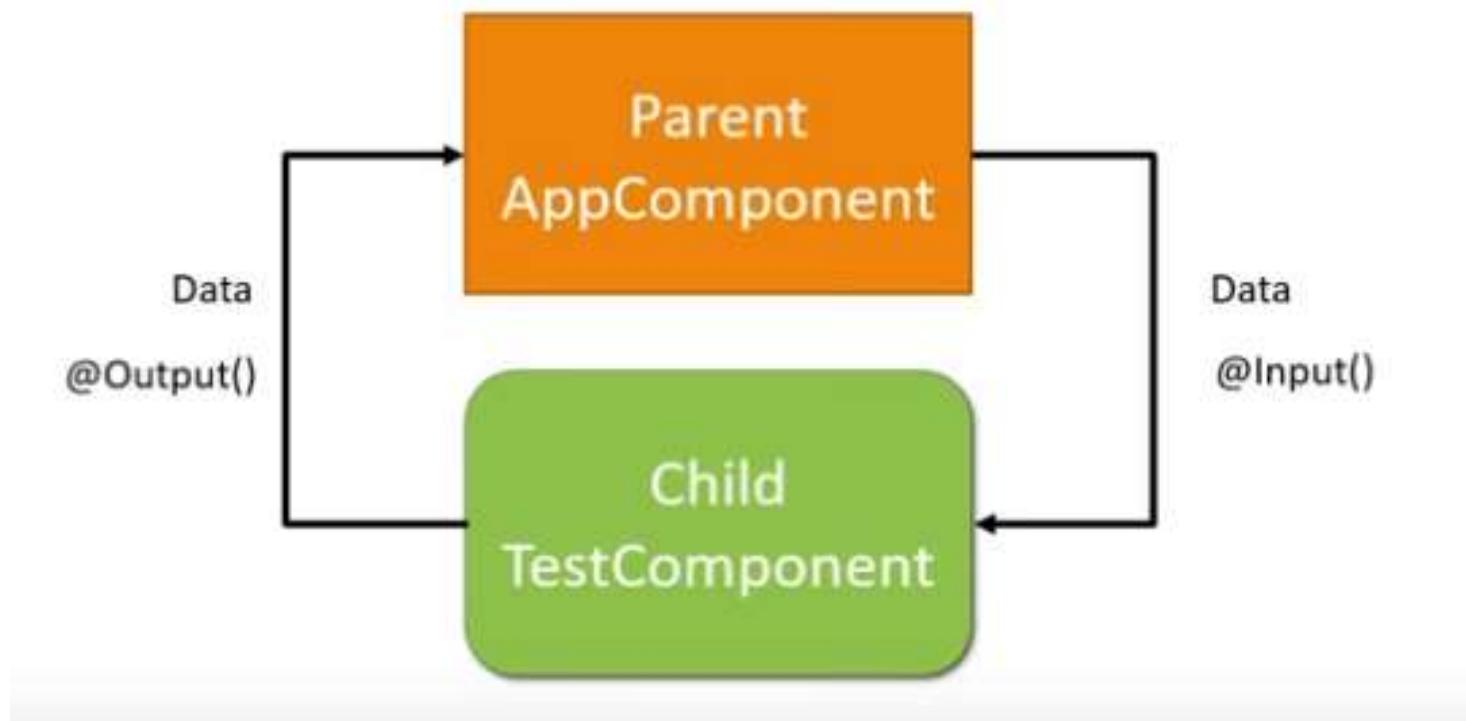
W jaki sposób komponenty się komunikują??

1. Inputs i Outputs (tylko powiązane)
2. Usługi (wszystkie)

Komunikacja między komponentami



Komunikacja między komponentami



Input

Container Component

Nested Component

Input

```
@Input() reviews: number;
```

komponent rodzica ma możliwość przekazania do dziecka danych, które mogą determinować zachowanie komponentu lub w ogóle – pozwolić na jego odpowiednie wyrenderowanie. Jest to możliwe dzięki adnotacji `@Input()`

Komponent: komunikacja "do"

```
@Component({  
  selector: 'test-input',  
  template: '...'  
})  
export class GR_Test {  
  @Input() item: myType;  
}
```

```
// lub:  
@Component({  
  selector: 'test-input',  
  inputs: ['item'],  
  template: '...'  
})  
export class GR_Test {  
  item: myType;  
}
```

```
// przekazywanie przez zmienną:  
<test-input [item]="myItem-GR"></test-input>  
  
// przekazywanie wartości bezpośrednio  
<test-input item="myItem-GR"></test-input>
```

Krok 1. Dodanie nowej własności typu input

```
import {Component, Input} from "@angular/core";  
  
@Component({  
    selector: "my-child",  
    template: "This is a child component with a message: {{ message }}"  
})  
export class ChildComponent {  
    @Input() message: string;  
}
```



Krok 2: Powiąż zmienną rodzica z tą właściwością

```
@Component({  
  selector: "app",  
  template: `<my-child [message]= "pozdrowienia" ></my-child>`,  
  directives: [ChildComponent]  
})  
export class AppComponent {  
  pozdrowienia = " Pozdrowienia od GR ";  
}
```



Input

child_component.ts

```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'child'
})

export class ChildComponent {
  @Input() reviews: number;
}
```

child_component.html

```
<div>
  <p>{{ reviews }}</p>
</div>
```

parent_component.ts

```
export class ParentComponent {
  books: any[] = [
    { bookReviews: 15 }
  ]
}
```

parent_component.html

```
<div><h1>Parent Title</h1>
<p>body text...</p>
<child [reviews]="book.bookReviews">
</child>
</div>
```

Komponent: komunikacja "z" - zdarzenia

```
// lub:  
@Component({  
  selector: 'test-output',  
  template: '...'  
})  
export class Hello {  
  @Output() completed:  
  EventEmitter<boolean> = new EventEmitter<boolean>();  
}
```

```
@Component({  
  selector: 'test-output',  
  outputs: ['completed'],  
  ...  
})
```

```
<test-output (completed)="saveProgress(item)"></test-output>  
  
// alternatywna składnia - dlatego nie prefixujemy zdarzeń "on"  
<test-output on-completed="saveProgress(item)"></test-output>
```

Implementacja

```
import {Component, Input, EventEmitter, Output} from "@angular/core";
```

```
@Component({  
    selector: "my-child",  
    template: `This is a child component with a message: {{ message }}`  
})  
export class ChildComponent {  
    @Input() message: string;  
    @Output() signaledIsHungry = new EventEmitter<string>();  
}
```

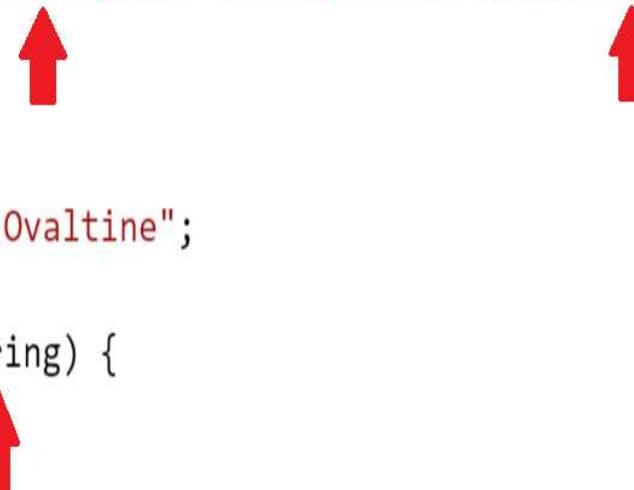


EventEmitter jest klasą generyczną – szablonem.
parametr jaki będzie przekazywał w postaci argumentu, będzie typu **string**.

Dodanie funkcji subskrybujacej

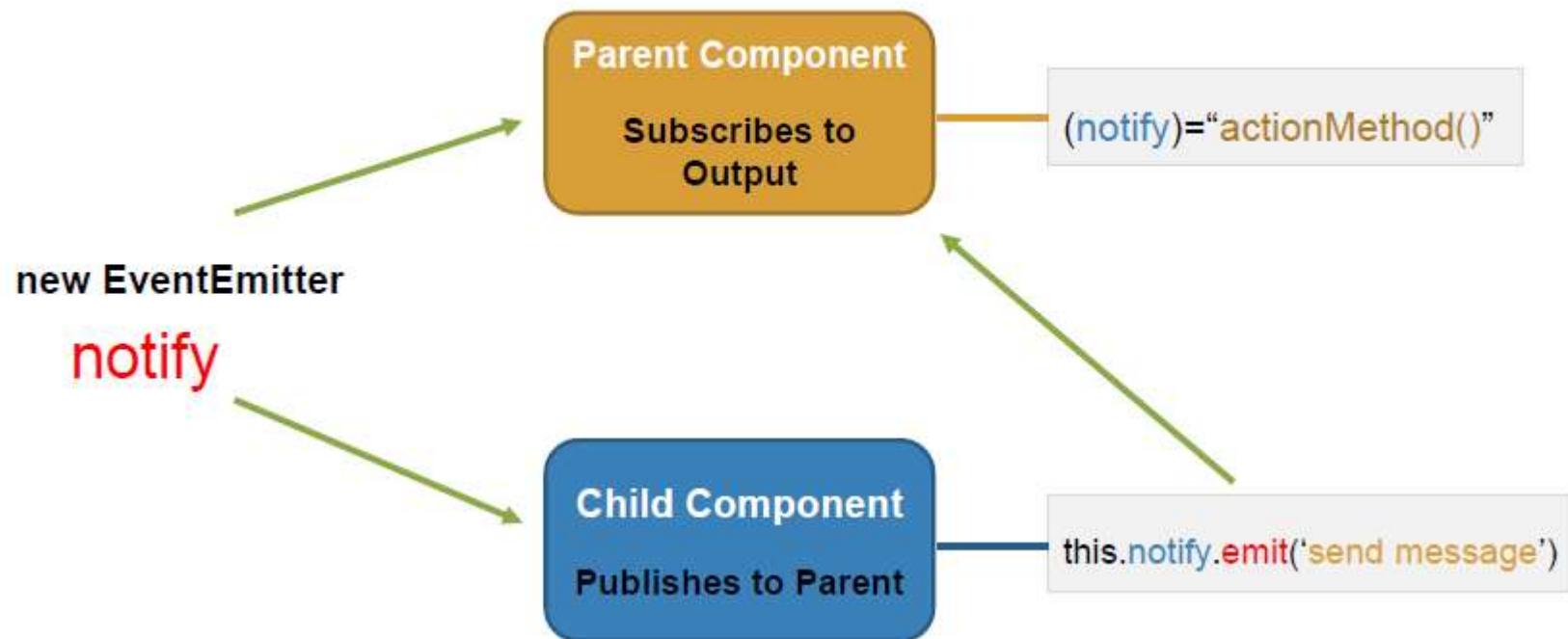
```
@Component({
  selector: "app",
  template: `<my-child [message]="marketingPhrase"
                (signaledIsHungry)="didSignalIsHungry($event)"></my-child>`,
  directives: [ChildComponent]
})
export class AppComponent {
  marketingPhrase = "Drink more Ovaltine";

  didSignalIsHungry(message: string) {
    console.log(message);
  }
}
```



W tym kodzie dodaliśmy funkcję `didSignalIsHungry`, która jest wywoływana, gdy zdarzenie `(signaledIsHungry)` jest emitowane z komponentu `my-child`. Ta funkcja loguje przekazaną do niej wiadomość.

Subskrypcja zdarzeń



Output

child_component.ts

```
import { Output, EventEmitter } from '@angular/core';

export class ChildComponent {
  Output() notify: EventEmitter<string> =
    new EventEmitter<string>();

  onClick(): void {
    this.notify.emit('Message from child');
  }
}
```

child_component.html

```
<div
  (click)="onClick()"
>
<button>Click Me</button>
</div>
```

parent_component.ts

```
export class ParentComponent {

  onNotifyClicked(message: string): void {
    this.showMessage = message;
  }
}
```

parent_component.html

```
<div><h1>Parent Title</h1>
<p>{{ showMessage }}</p>
<child (notify)="onNotifyClicked($event)">
</child>
</div>
```



Usługi Service

Klasa do specyficznych celów:

- dzielenia danych
- Implementacja logiki aplikacyjnej/biznesowej
- Zewnętrzna interakcja – odczyty danych z serwera

Usługi

- Klasy implementujące funkcje potrzebne w aplikacji
 - Pojedyncza usługa powinna odpowiadać za konkretną funkcjonalność
- Podział logiki między klasy komponentów i usług
 - Klasa komponentu powinna zawierać logikę specyfczną dla widoku
 - Właściwości i metody do wiązania danych z widoku
 - Pośredniczenie między widokiem a logiką biznesową (modelem)
 - Klasy usług nie powinny zależeć od widoków
 - Komunikacja z backendem, walidacja, obsługa logu itd.
- Usługi są udostępniane komponentom i innym usługom przez wstrzykiwanie zależności (ang. dependency injection)
 - Dekorator @Injectable oznacza klasy będące usługami oraz klasy i komponenty zależne od usługi
 - Wstrzykiwanie realizowane przez typowany parametr konstruktora
 - Dostawcy usług (ang. providers), typowo klasy usług, rejestrowani na poziomie modułów lub komponentów

Implementacja usług - uwagi

- Klasy komponentów powinny być szczupłe, bez nadmiaru (kodu, funkcjonalności). Nie pobierają danych z serwera, walidują danych wejściowych od użytkownika czy zapisują logi bezpośrednio do konsoli. Takie zadania są delegowane do usług.
- **Zadaniem komponentu jest udostępnienie użytkownikowi danej funkcjonalności i nic ponad to.**
- Komponent pośredniczy pomiędzy widokiem (renderowanym na bazie szablonu) i logiką aplikacji (która często zawiera jakąś wiedzę o modelu).
- Dobry komponent prezentuje właściwości i metody do wiązania danych. Wszystkie nietrywialne zadania są delegowane do usług.
- Angular pomaga nam *przestrzegać* tych wytycznych poprzez ułatwianie nam podzielenia logiki aplikacji na usługi i uczynienie tych usług dostępnymi w komponentach poprzez *wstrzykiwanie zależności*

Usługi - service

- Generacja automatyczna ng –g s nazwaUsługi

```
import { Injectable } from '@angular/core';

@Injectable()
export class CountryService {

    constructor() { }

    getCountry() {
        return ["Polska", "niemcy", "Rosja", "czechy"];
    }
}

import { CountryService } from './country.service';
import { Component, OnInit } from '@angular/core';

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

    buttonStatus = true;
    kraje;

    constructor() {
        let service = new CountryService();
        this.kraje = service.getCountry();
    }
}
```

Usługa z DI i bez - porównanie

Without DI

```
class Car{
    engine;
    tires;
    constructor()
    {
        this.engine = new Engine();
        this.tires = new Tires();
    }
}
```

With DI

```
class Car{
    engine;
    tires;
    constructor(engine, tires)
    {
        this.engine = engine;
        this.tires = tires;
    }
}
```

```
class Engine{
    constructor(){}
}
class Tires{
    constructor(){}
}
```

```
class Car{
    engine;
    tires;
    constructor()
    {
        this.engine = new Engine();
        this.tires = new Tires();
    }
}
```

Wstrzykiwanie zależności

Klasę rozszerzoną dekoratorem (*np. @Injectable*)

```
@Injectable()
class TodosService {
  constructor() {}
}
```

... i zarejestrowaną jako *provider*

```
@NgModule({
  providers: [
    TodoService,
```

... możemy wstrzyknąć do konstruktorów innych klas

```
class TodoItemComponent {
  constructor(ts: TodosService) {} // skrócony zapis
// constructor(@Inject(TodosService) ts) {} // pełen zapis
}
```

Wstrzykiwanie serwisu do Komponentu



Provider



Injector

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

Wstrzykiwanie zależności

Dependency Injection

```
constructor() {  
    let service = new CoursesService();  
    this.courses = service.getCourses();  
}
```

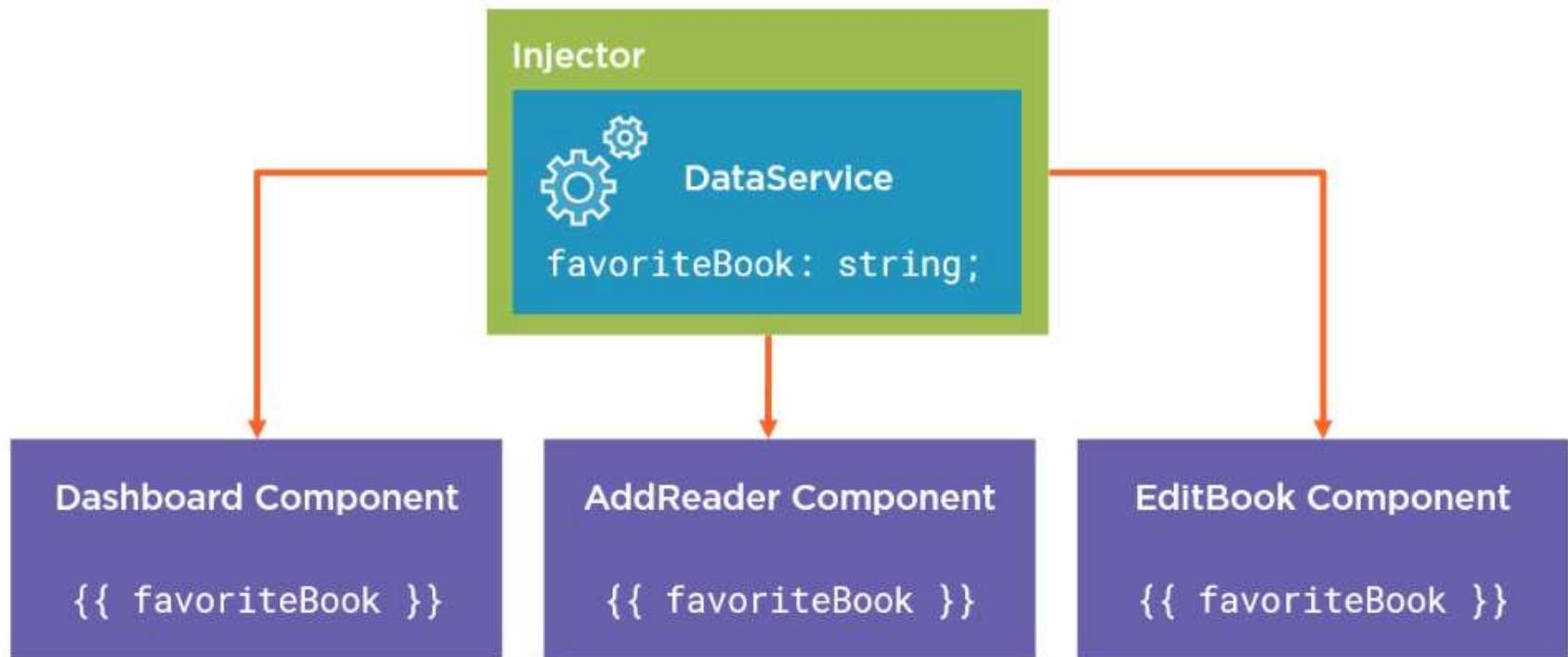
ZLE

VS

```
constructor(service: CoursesService) {  
    this.courses = service.getCourses();  
}
```

DOBRZE

Wymiana danych poprzez usługę



Usługa – wstrzykiwanie zależności

