Angular

Infolinia: 0 801 258 566
www.altkomakademia.pl

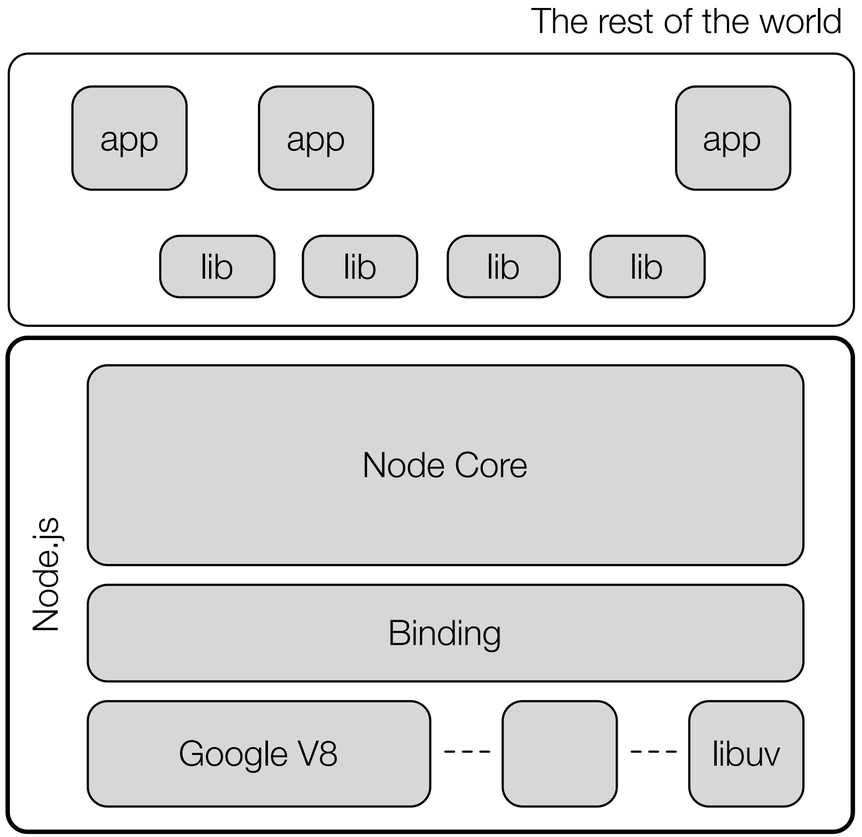
Altkom Akademia

Node.js

Node.js - Charakterystyka

Node.js

Środowisko uruchomieniowe zbudowane na maszynie V8 (Google's JS engine).
Zoptymalizowane pod kątem aplikacji serwerowych czasu rzeczywistego.
Pozwala na uruchamianie kodu JS poza przeglądarką internetową.
Napisane w języku C++



https://nodejs.org

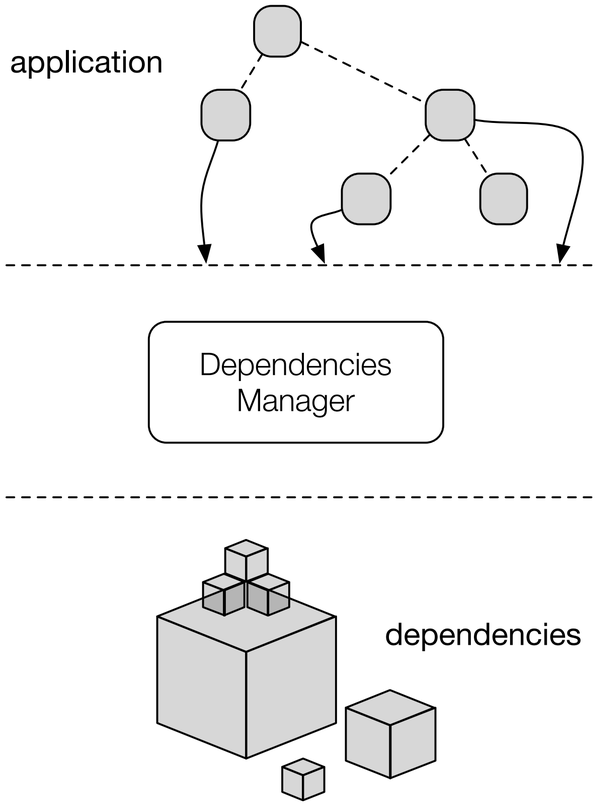
Node.js - Zależności projektowe

npm.js

Narzędzie zarządzania pakietami i ich zależnościami dla Node.js.
Pierwotnie przeznaczone do zarządzania modułami Node.js.
Wykorzystywane też do zależności aplikacyjnych.
Konfiguracja zależności oparta o plik package.json
Pobrane pakiety przechowywane w folderze node_modules.

Bower.js

Narzędzie zarządzania pakietami i zależnościami pomiędzy nimi.
Zoptymalizowany do zarządzania pakietami aplikacyjnymi.
Konfiguracja zależności oparta o plik bower.json.
Pobrane pakiety przechowywane w folderze bower_components.



https://www.npmjs.com, https://bower.io

Node.js - Instalacja

https://nodejs.org/en/download/

Instalacja Node.js wraz z managerem pakietów npm.

node -v

Sprawdza wersję platformy node.

npm -v

Sprawdza wersję managera pakietów npm.

npm

npm - Zasięg pakietów

Pakiety lokalne

Biblioteki zlokalizowane w katalogu, w którym wykonywana jest funkcja npm.
Najczęściej jest to główny katalog projektu.
Zarządzanie pakietami dobywa się domyślnie w trybie lokalnym.

Pakiety globalne

Biblioteki zlokalizowane w globalnym katalogu /usr/local/lib/node_modules.
Zarządzanie pakietami w trybie globalnym wymaga uruchomienia programu npm z odpowiednim parametrem.

-g

Flaga zmieniająca mod wykonania komendy na globalny. (domyślny mod to local)

npm - Rodzaje pakietów

```
"dependencies" : {
  "rxjs": "^6.0.0",
  "zone.js": “^0.8.26"
}
```

dependencies

Pakiety wymagane przez aplikację w środowisku produkcyjnym.

devDependencies

Pakiety wymagane w środowisku developerskim.

```
"devDependencies" : {
  "typescript": “~2.7.2”,
  "jasmine-core": "~2.99.1"
}
```

npm - Wersjonowanie pakietów, SemVer

Semantic Versioning (SemVer)

Definiuje konwencję wersjonowania pakietów (publicznego API).

Major

Zmiana mogąca powodować niekompatybilność z poprzednią wersją.
Zmiana na poziomie interfejsu.

Minor

Zmiana zachowująca kompatybilność wsteczną.

Path

Zmiana zachowująca kompatybilność wsteczną.
Najczęściej związana z nanoszeniem poprawek (bug fixes).

0.x.x Versions (major=0)

Oznacza “initial development”.
W tym przypadku człon minor praz path pozostają bez znaczenia.

```
major.minor.path
1.2.4, major: 1, minor:2, path:4
```

npm - Wersjonowanie pakietów, SemVer

Zasięg

Daje możliwość określania maksymalnej wersji pakietu.

*Definiowana przez * lub x oraz symbole -, >, >=, <, <=, oraz ~, ^*

```
"1", "1.x.x"                                -> 1.2.4, 1.0.2
"2.5", "2.5.*"                              -> 2.5.4, 2.5.1
"1.4.2 - 2.5.8", ">=1.4.2 <=2.5.8" -> 1.4.2, 1.5.4, 2.4.4, 2.5.8
">=1.4.0 <1.8.2"                            -> 1.4.0, 1.5.2, 1.8.0, 1.8.1
“~1.4.5"                                    -> ">=1.4.5 <1.5.0”
“^1.4.5"                                    -> ">=1.4.5 <2.0.0"
```

npm - Funkcje

:~$ npm

```
Usage: npm <command>

where <command> is one of:

    access, adduser, bin, bugs, c, cache, completion, config,
    ddp, dedupe, deprecate, dist-tag, docs, doctor, edit,
    explore, get, help, help-search, i, init, install,
    install-test, it, link, list, ln, login, logout, ls,
    outdated, owner, pack, ping, prefix, profile, prune,
    publish, rb, rebuild, repo, restart, root, run, run-script,
    s, se, search, set, shrinkwrap, star, stars, start, stop, t,
    team, test, token, tst, un, uninstall, unpublish, unstar,
    up, update, v, version, view, whoami
```

```
npm <command> -h quick help on <command>
npm -l           display full usage info
npm help <term>  search for help on <term>
npm help npm     involved overview
```

npm - Funkcje, Cache

npm cache

Zarządzanie lokalnymi, podręcznymi plikami bibliotek.
Dodawanie, usuwanie i weryfikacja plików.

~/.npm/_cacache

Domyślny katalog dla plików podręcznych.

npm cache clean --force

Usuwa wszystkie pliki podręczne.
Komenda nie powinna być nadużywana (zachowanie integralności danych).
Zaleca się użycie tyko w celu zwolnienia przestrzeni dyskowej.

npm cache verify

Sprawdzanie poprawności danych podręcznych.
Usuwa zbędne pliki i weryfikuje indeks przechowywanych plików.

```
:$ npm cache clean
npm ERR! As of npm@5, the npm cache self-heals
from corruption issues and data extracted from
the cache is guaranteed to be valid.

If you want to make sure everything is
consistent, use 'npm cache verify' instead.

If you're sure you want to delete the entire
cache, rerun this command with –force.
```

```
:$ npm cache clean ––force
npm WARN using ––force I sure hope you know what
you are doing.
```

```
:$ npm cache verify
Cache verified and compressed (~/.npm/_cacache):
Content verified: 1841 (64969467 bytes)
Index entries: 2872
Finished in 44.671s
```

npm - Funkcje, ls

npm ls

Listuje zainstalowane pakiety wraz z zależnościami. Zależności przedstawione są w formie drzewa.

-json

Zależności przedstawione są w formacie JSON.

-depth=<1>

Drzewo zależności będzie wyświetlane do poziomu zagnieżdżenia określonego przez parametr.

npm ls <pakiet>

Listuje zależności dla podanego pakietu.

-dev

Listuje zainstalowane pakiety skonfigurowane jako zależności developerskie.

-prod

Listuje zainstalowane pakiety skonfigurowane jako zależności produkcyjne.

```
:~/app$ npm ls -depth=0 -prod
my-app@0.0.0 /Users/tom/my-app
├── @angular/animations@6.0.4
├── @angular/common@6.0.4
├── ...
├── ...
├── core-js@2.5.7
├── rxjs@6.2.0
└── zone.js@0.8.26
```

```
:~/app$ npm ls -depth=0 -dev
my-app@0.0.0 /Users/tom/my-app
├── @angular/cli@6.0.8
├── @angular/compiler-cli@6.0.4
├── ...
├── ...
├── karma-jasmine@1.1.2
├── tslint@5.9.1
└── typescript@2.7.2
```

```
:$ npm ls -g -depth=1
/Users/tom/.npm-global/lib
└─┬ @angular/cli@6.0.8
  ├── @angular-devkit/architect@0.6.8
  ├── @angular-devkit/core@0.6.8
  ├── @angular-devkit/schematics@0.6.8
  ├── @schematics/angular@0.6.8
  ├── @schematics/update@0.6.8
  ├── opn@5.3.0
  ├── resolve@1.7.1
  ├── rxjs@6.2.0
  ├── semver@5.5.0
  ├── silent-error@1.1.0
  ├── symbol-observable@1.2.0
  └── yargs-parser@10.0.0
```

npm - Funkcje, Outdated

npm outdated

Listuje zainstalowane pakiety, dla których dostępne są nowsze wersje

npm outdated <pakiet>

Sprawdza dostępność nowszej wersji dla pakietu podanego jako parametr.

-depth=<0>

Sprawdza wersję pakietów wgłąb drzewa zależności do głębokości zdefiniowanej jako parametr.
Domyślnie głębokość skanowania przyjmuje wartość 0.

-l

Dodatkowo wyświetla informacje o konfiguracji pakietu (dependencies, devDependencies)

```
:~/app$ npm outdated
Package                       Current  Wanted  Latest  Location
@types/node                     8.9.5   8.9.5  10.3.1  app
codelyzer                       4.2.1   4.2.1   4.3.0  app
jasmine-core                   2.99.1  2.99.1   3.1.0  app
karma                           1.7.1   1.7.1   2.0.2  app
karma-jasmine-html-reporter     0.2.2   0.2.2   1.1.0  app
ts-node                         5.0.1   5.0.1   6.1.0  app
tslint                          5.9.1   5.9.1  5.10.0  app
typescript                      2.7.2   2.7.2   2.9.1  app
```

npm - Funkcje, Outdated

Current

Wersja aktualnie zainstalowanego pakietu.

Wanted

Wersja pakietu zdefiniowana w package.json.
Dla pakietów globalnych jest to wersja akietu w jakiej został zainstalowany.

Latest

Wersja pakietu z flagą @latest.
Nie jest wymagane, aby była to wersja maksymalna.

```
:~/app$ npm outdated
Package                      Current  Wanted  Latest  Location
@types/node                    8.9.5   8.9.5  10.3.1  app
codelyzer                      4.2.1   4.2.1   4.3.0  app
jasmine-core                  2.99.1  2.99.1   3.1.0  app
karma                          1.7.1   1.7.1   2.0.2  app
karma-jasmine-html-reporter    0.2.2   0.2.2   1.1.0  app
ts-node                        5.0.1   5.0.1   6.1.0  app
tslint                         5.9.1   5.9.1  5.10.0  app
typescript                     2.7.2   2.7.2   2.9.1  app
```

npm - Funkcje, Install

npm install

Instaluje pakiety wraz z ich zależnościami.
Wywołana z poziomu folderu, w którym znajduje się plik package.json,
instaluje wszystkie pakiety w nim zdefiniowane.

npm install <pakiet>@<version>

Instaluje pojedynczy pakiet i jego zależności.
Jeżeli brak specyfikacji wersji pakietu, zostanie zainstalowana ta z flagą @latest.
Dodaje wpis w sekcji dependencies w pliku package.json
Wersję należy podać zgodnie z SemServ, np.: npm install sax@">=0.1.0 <0.2.0"

npm - Funkcje, Install

--save-prod

Dodaje wpis w sekcji dependencies w pliku package.json

--save-dev

Dodaje wpis w sekcji devDependencies w pliku package.json

--no-save

Plik package.json nie będzie uaktualniony.

-f

W przypadku pakietów już zainstalowanych wymusza ponowne popranie pakietu z repozytorium.

npm - Funkcje, Uninstall

npm uninstall

Odinstalowuje pakiety.

npm uninstall <pakiet>

Odinstalowuje pojedynczy pakiet i jego zależności.

--save

Usuwa wpis z sekcji dependencies pliku package.json

--save-dev

Usuwa wpis z sekcji devDependencies pliku package.json

npm - Funkcje, Update

npm update

Uaktualnia wszystkie pakiety zdefiniowane w pliku packages.json.
Pakiety będą uaktualnione do najnowszej wersji z uwzględnieniem konfiguracji SemServ.
Od wersji npm@2.6.1 aktualizowane są tylko pakiety główne.
Dla nowszych wersji: aby aktualizować wszystkie zależności: ***npm --depth 9999 update***
Dla starszych wersji: aby aktualizować tylko pakiety główne: ***npm --depth 0 update***

npm update <pakiet>

Uaktualnia pojedynczy pakiet.

--save

Modyfikuje wersję pakietu w sekcji dependencies w pliku package.json

--save-dev

Modyfikuje wersję pakietu w sekcji devDependencies w pliku package.json

TypeScript

TypeScript - Charakterystyka

Nadzbiór języka JavaScript

Wszystko co jest poprawne w JS, jest również poprawne w TS.
Posiada dodatkowe funkcjonalności usprawniające proces programowania.

Kompilacja

Kod źródłowy TS jest kompilowany (transpilowany) do postaci kodu JS.

Statyczne typowanie

Zwiększa stopień wykrywalności błędów przed uruchomieniem aplikacji.
Nie ma konieczności definiowania typu dla zmiennej lub funkcji. (optional statically typed language)
Potrafi wywnioskować typ (infer types) zmiennej po wartości do niej przypisanej.
Do zmiennej ze zdefiniowanym typem ogólnym ‘any’ można przypisywać wartości różnych typów.

http://typescriptlang.org

TypeScript - Charakterystyka

Dodatkowe elementy języka

Wprowadza: interfejsy, typy generyczne, enums, klasy abstrakcyjne …
Pozwalają na zwiększenie możliwości tworzenia skalowalnych aplikacji z zastosowaniem architektury obiektowej.
Nowe elementy języka istnieją w trakcie pisania aplikacji (design time), nie są obecne w kodzie wykonywanym (runtime code).

ECMAScript

Wykorzystuje cechy języka określone w ECMAScript6, ECMAScript7.

https://www.ecma-international.org

TypeScript

Operator equal

TypeScript - Operator equal

==

Równa wartość. (!)

===

Równa wartość i typ.

TypeScript - Operator equal

```
== equal value
=== equal value and equal type

4 == '4'  //true
4 === '4' //false
//ale
true == 'true' //false (brak konwersji string <-> boolean)
true === 'true' //false
```

TypeScript - Operator equal

```
!= not equal value
!== not equal value or not equal type

4 != '4'  //false (ponieważ ta sama wartość)
4 != '8'  //true (ponieważ różna wartość)
4 !== '4' //true (ponieważ różny typ)
4 !== '8' //true (ponieważ różny typ lub wartość)
//ale
true != 'true' //true (brak konwersji string <-> boolean)
true !== 'true' //true
```

TypeScript

Typy i zmienne

TypeScript - Typy i zmienne, Zasięg

var

Posiada zasięg najbliższego bloku funkcyjnego lub zasięg globalny jeśli zdefiniowana poza funkcją.

let

Posiada zasięg najbliższego bloku lub zasięg globalny jeśli zdefiniowana jest poza jakimkolwiek blokiem.

const

Zasięg taki jak let.

TypeScript - Typy i zmienne, Boolean

```
let isHidden: boolean = false;
```

TypeScript - Typy i zmienne, Number

```
let itemIndex: number = 1;
let redColor: number = 0xf;
let mask: number = 0b100;
let volume: number = 0o125;
```

TypeScript - Typy i zmienne, String

```
let firstName: string = "Tom";
let lastMark: string = 'Smith';
let fullName: string =  firstName + ' ' + lastName;

let description = `First name: ${firstName} Last name: ${lastName}`
```

TypeScript - Typy i zmienne, Array

```
let indexes: number[] = [1, 2, 3, 4];
let indexes: Array<number> = [1, 2, 3, 4];
let indexes: Array<number> = new Array(4);
let names: string[] = new Array('Ana', 'Kate');
```

TypeScript - Typy i zmienne, Tuples

```
let fullName: [string, string] = ['Tom', 'Smith'];
let person: [number, string, string] = [12, 'Tom', 'Smith'];
person[0]; person[1]; person[2]

type Person = [number, string, string];
let user: Person = [1, 'Ana', 'Smith']
```

TypeScript - Typy i zmienne, Union

```
let item :string|number;
```

```
if(typeof item === 'string') {
  //...
} else if(typeof item === 'number') {
  //...
}
```

TypeScript - Typy i zmienne, Any

```
let box: any = new Box();
box = 4; //ok
box = 'text'; //ok

let items: any[] = ['text', true, 8, false];
items[0] = 1; //ok
```

TypeScript - Typy i zmienne, Enum

```
enum Color {Red, Green, Blue};

let color: Color = Color.Green;
```

TypeScript - Typy i zmienne, Type, Generics

```
type MyArray = Array<number|string>;
type MyNumber = number;
type MyHandler = () => void;
```

TypeScript - Typy i zmienne, typeof, instanceof

typeof

zwraca typ zmiennej.

instanceof

Zwraca true jeżeli zmienna jest określonego typu.

TypeScript - Typy i zmienne, typeof

```
let word = "Hello";
let index = 5;
let createdAt = new Date();
let handler = function isActive() { return true; };
class Item {};
let anItem = new Item();
let person = {name:'John', age:34}
```

```
typeof word;  //string
typeof index;  //number
typeof createdAt;  //object
typeof handler;  //function
typeof anItem;  //object
typeof Item;  //function (!)
typeof person = //object
```

TypeScript - Typy i zmienne, instanceof

```
class Item {};
let anItem = new Item();

if(anItem instanceof Item) {
  //...
}
```

TypeScript - Typy i zmienne, undefined, null

undefined

Brak przypisanej wartości, nieznany typ.

null

Brak przypisanej wartości, typ object.

```
typeof null; //object
typeof undefined; //undefined

null === undefined; //false
null == undefined; //true
```

TypeScript - Typy i zmienne, undefined, null

```
let item; //brak przypisanej wartości, nieznany typ
console.log(item); //undefined
let anotherItem = null; //brak wartości, typ object
```

TypeScript - Typy i zmienne, void

void

Brak typu. Np. funkcja nie zwracająca żadnej wartości.

TypeScript

Funkcje

TypeScript - Funkcje, Named function

```
function doJob(param? : string) : string {
  let result: string;
  if(param){
    //...
  }
  else {
   //...
  }
  return result;
}
```

TypeScript - Funkcje, Anonymous function

```
var doJob = function(param? : string) : string {
  let result: string;
  if(param){
    //...
  }
  else {
   //...
  }
  return result;
}
```

TypeScript - Funkcje, Arrow functions

```
var doJob = (param? : string) : string => {
  let result: string;
  if(param){
    //...
  }
  else {
   //...
  }
  return result;
}
```

TypeScript - Funkcje, Arrow functions

```
var doJob: (param? : string) => string =

  function(param? : string) : string{
    let result: string;
    if(param){
      //...
    }
    else {
     //...
    }
    return result;
}
```

TypeScript - Funkcje, Arrow functions

```
function task(index: number,
              handler: (result: number) => void) {
  //...
  handler(index);
}
```

TypeScript

Klasy

TypeScript - Klasy

```
class Person {
  firstName: string;
  lastName: string;

  constructor(firstName: string, lastName: string) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

 fullName(): string {
   return `${this.firstName} ${this.lastName}`;
  }
}
```

TypeScript - Klasy

```
let person: Person = new Person('Ana', 'Smith');
person.fullName();
```

TypeScript

Interfejsy

TypeScript - Interfejsy

```
interface IFullName {
  fullName(): string;
}

class Person implements IFullName {
  //...

 fullName(): string {
   return `${this.firstName} ${this.lastName}`;
  }
}
```

TypeScript - Interfejsy

```
let person: IFullName = new Person('Ana', 'Smith');
person.fullName();
```

TypeScript - Interfejsy

```
interface IFullName {
  firstName: string;
  lastName: string;

  fullName(): string;
}

class Person implements IFullName {
  firstName: string;
  lastName: string;
  //...

 fullName(): string { return `${this.firstName} ${this.lastName}`; }
}
```

TypeScript - Interfejsy

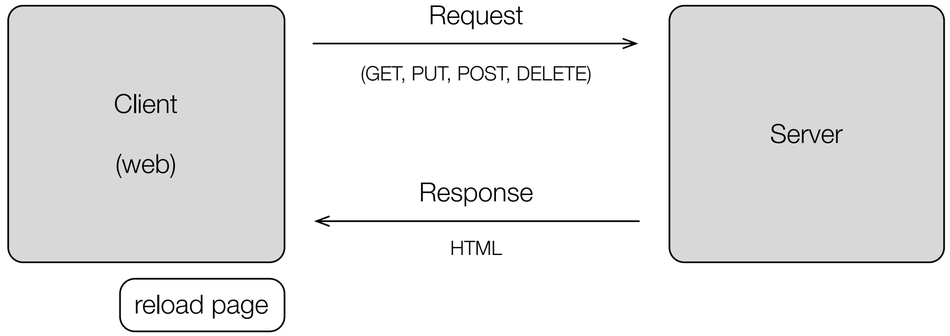
```
let person: IFullName = {
  firstName:'Ana',
  lastName: 'Smith',

  fullName: function(): string {
    return `${this.firstName} ${this.lastName}`
  }
}

person.fullName();
```

Architektura SPA

Architektura SPA - Komunikacja

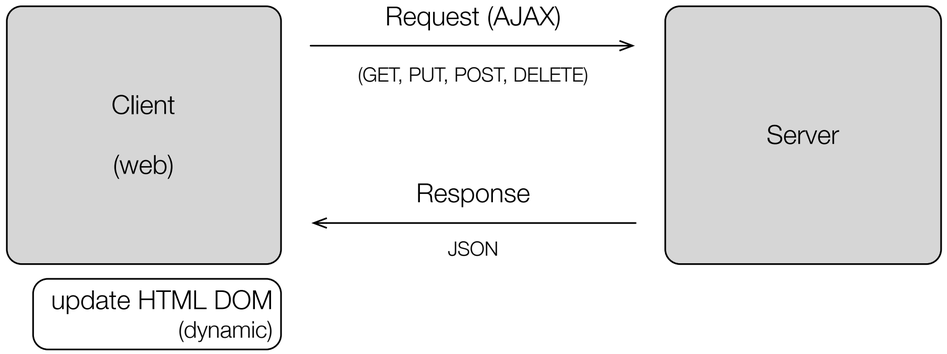
Model tradycyjny

Jedno zapytanie do serwera w celu pobrania danych i struktury całego widoku. Pojedyncze zapytanie z poziomu aplikacji to przeładowanie całej struktury strony (DOM) po otrzymaniu odpowiedzi.



Model dynamiczny

Wiele zapytań do serwera w celu pobrania danych dla jednego widoku. Każde pojedyncze zapytanie to przeładowanie części widoku (części struktury drzewa DOM)



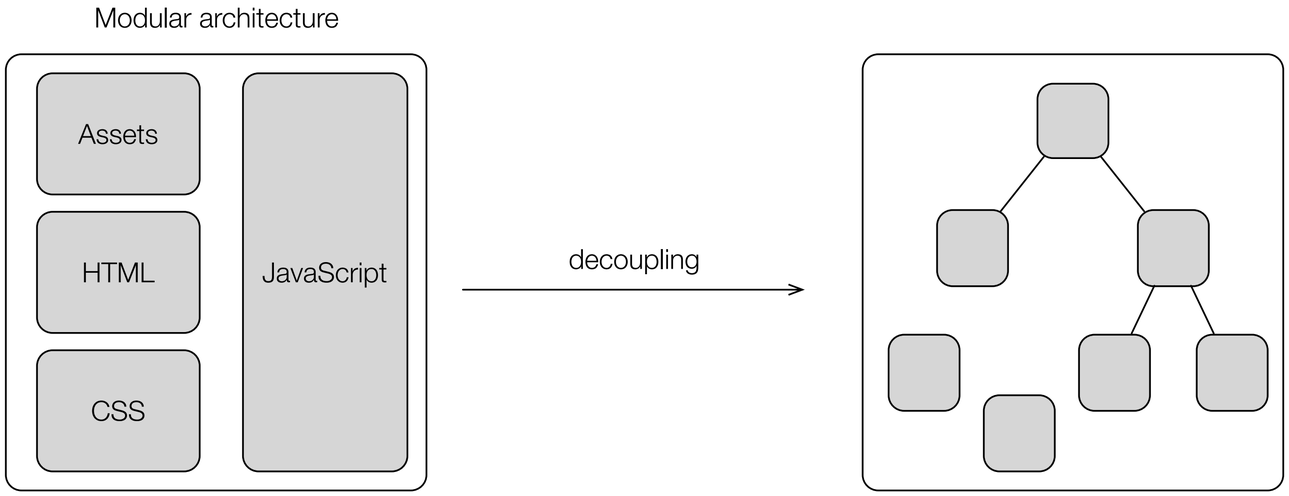
Architektura SPA - Widoki

Struktura modularna

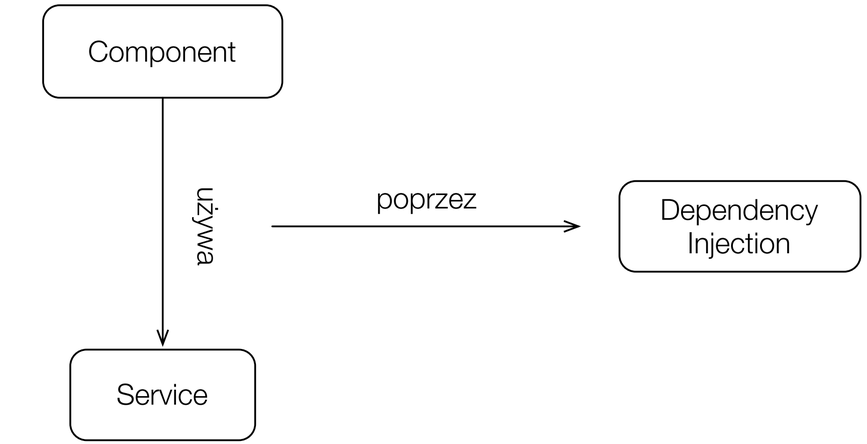
Trudna w utrzymaniu, testowaniu i skalowaniu.
Utrudnione ponowne użycie fragmentów widoków.

Struktura komponentowa

Układ wielu komponentów tworzących strukturę drzewiastą widoku.
Pojedynczy komponent dostarcza dane dla pewnego fragmentu widoku i zarządza jego wyświetlaniem.



Architektura SPA - Kod



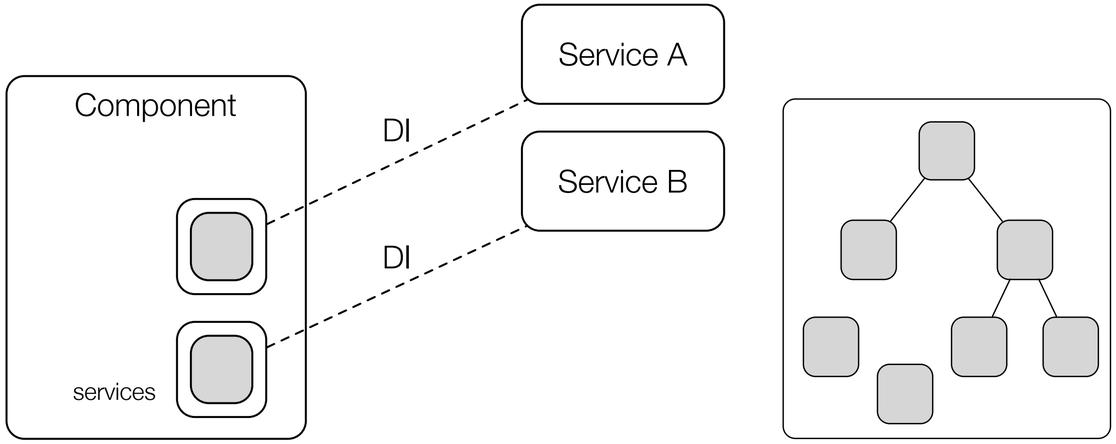
Model warstwowy

Wprowadza logiczną separację kodu w warstwach:

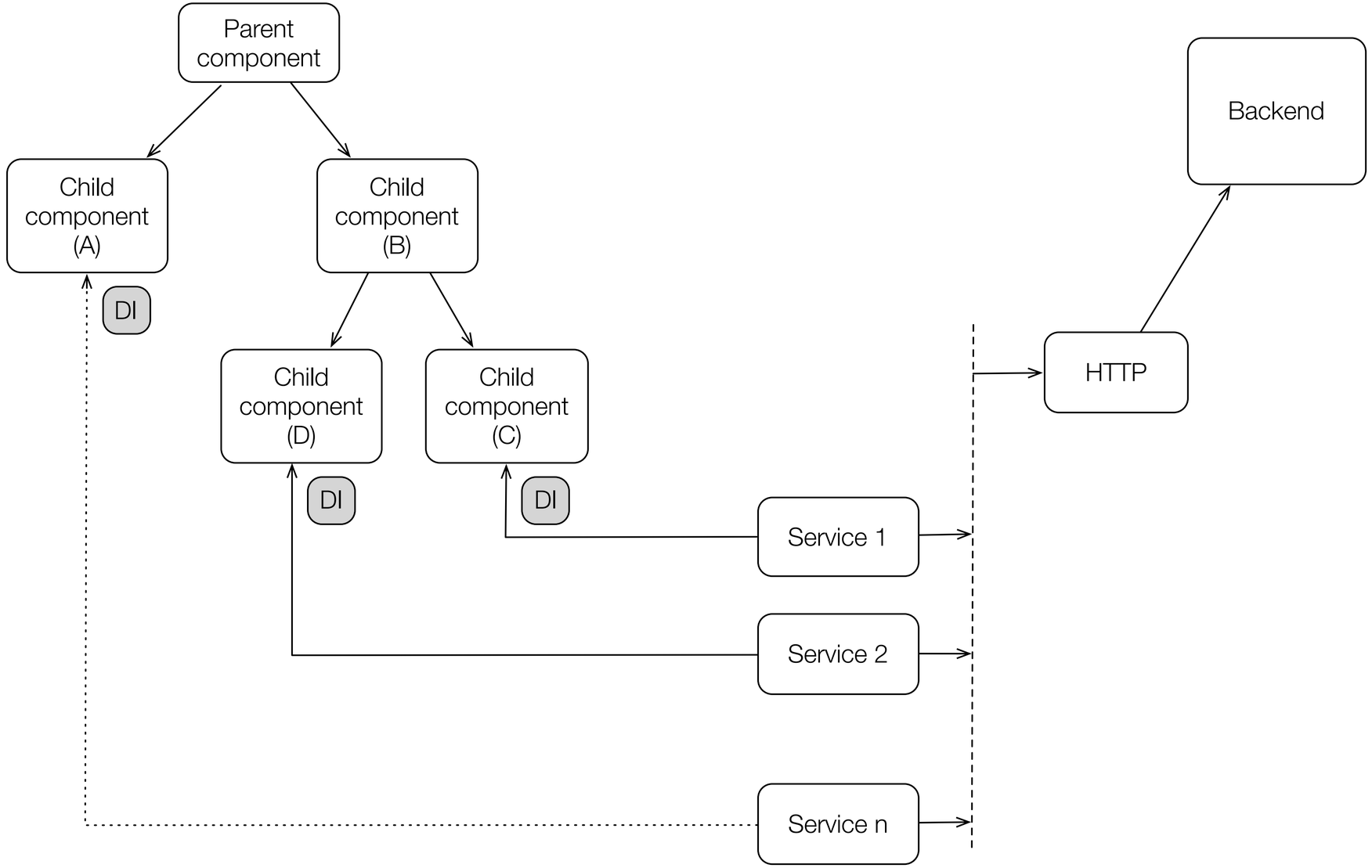
- *serwisu*
- *komponentu*
- *widoku*

Wstrzykiwanie zależności

Rozluźnienie powiązań pomiędzy współpracującymi obiektami.
Eliminuje konieczność umieszczania kodu tworzącego obiekty.



Architektura SPA - Komunikacja



Architektura SPA - Angular

Model, widok, kontroler - MVC

Koncepcja separacji warstwy prezentacji od warstwy logiki biznesowej.

Komponent

Komponent pełni rolę kontrolera łączącego model danych z widokiem.

Metadane

Metadane umieszczone w kodzie komponentu konfigurują między innymi:

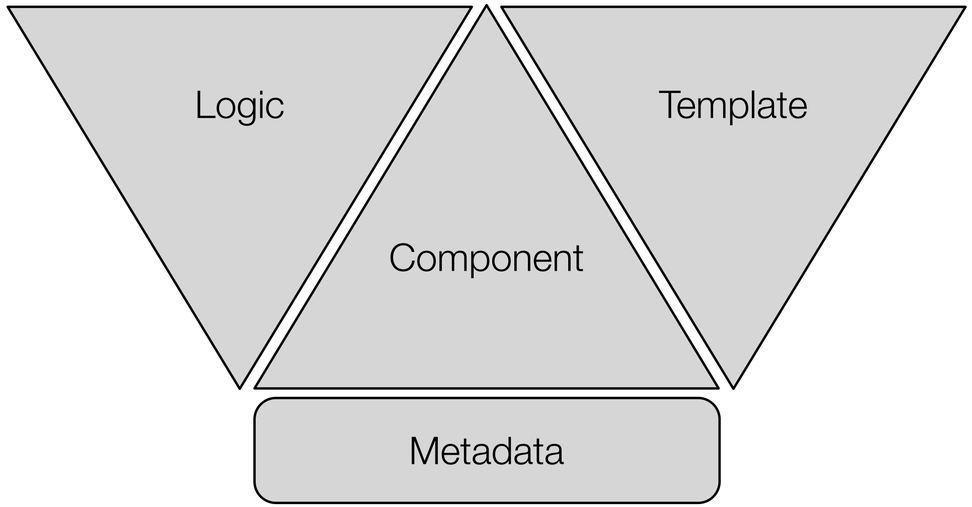
- *powiązanie z plikiem widoku,*
- *przypisanie stylów CSS do widoku,*
- *zależności od innych objektów (np.: warstwy serwisowej).*

Widok

Plik HTML definiujący warstwę prezentacji.

Model

Warstwa definiująca logiczny model danych biznesowych.



Angular

Angular

Środowisko robocze

Angular - Środowisko robocze

SystemJS

Ładowanie modułów aplikacji.

TSC

Transpilacja kodu źródłowego z TypeScript to JavaScript.

TSLint

Statyczna analiza kodu, jakość kodu.

Jasmine

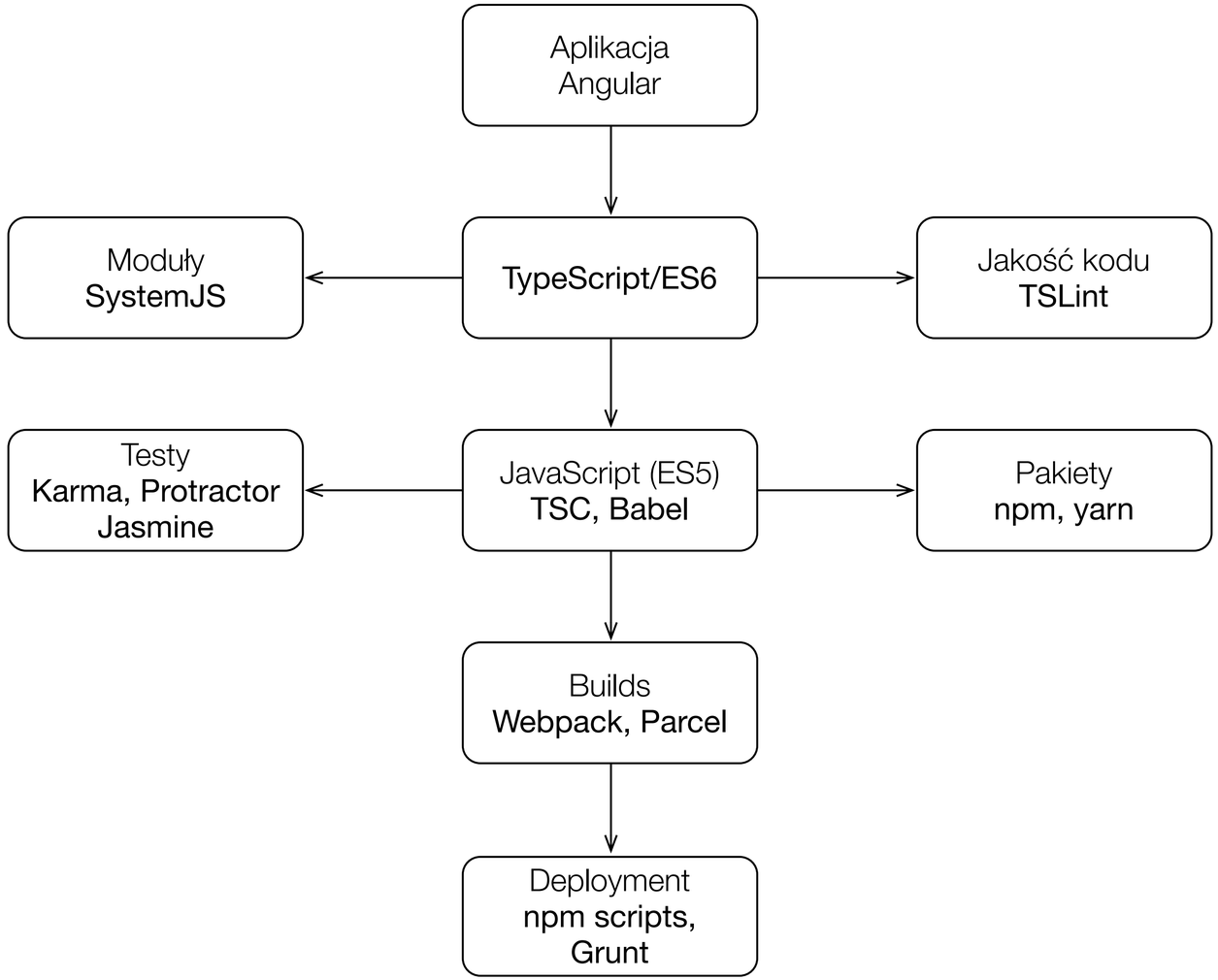
Biblioteka do pisania testów jednostkowych.

Webpack

Pakowanie modułów do formatu wdrożeniowego.
Minimalizacja źródeł.

npm scripts

Zarządzanie zadaniami i automatyzacja ich wykonania.



Angular

Instalacja CLI

Angular - Instalacja CLI

npm install -g @angular/cli

Instaluje Angular CLI.

ng --version

Wyświetla wersję wersję zainstalowanych pakietów.

```
:~/$ ng --version
```

```
     _                      _                 ____ _     ___
    / \   _ __   __ _ _   _| | __ _ _ __     / ___| |   |_ _|
   / △ \ | '_ \ / _` | | | | |/ _` | '__|   | |   | |    | |
  / ___ \| | | | (_| | |_| | | (_| | |      | |___| |___ | |
 /_/   \_\_| |_|\__, |\__,_|_|\__,_|_|       \____|_____|___|
                |___/
```

```
Angular CLI: 6.0.8
Node: 8.11.2
OS: darwin x64
Angular:
...

Package                            Version
------------------------------------------------------------
@angular-devkit/architect    0.6.8
@angular-devkit/core         0.6.8
@angular-devkit/schematics   0.6.8
@schematics/angular          0.6.8
@schematics/update           0.6.8
rxjs                         6.2.1
typescript                   2.7.2
```

Angular - Uaktualnienie CLI

Globalnie

npm uninstall -g @angular/cli

Usuwa zainstalowaną wersję.

npm cache verify

Czyści pliki tymczasowe.

npm install -g @angular/cli

Instalacja najnowszej wersji.

Lokalnie (w projekcie)

rm -rf node_modules dist

Usuwa pliki pakietów oraz folder z kodem produkcyjnym.

npm install --save-dev @angular/cli

Instalacja aktualnej wersji.

Angular

Projekt

Angular - Projekt, Tworzenie, Uruchamianie

ng new my-app

Generowanie szkieletu nowej aplikacji.

ng serve

Uruchomienie aplikacji: http://localhost:4200/

--open

Automatyczne uruchomienie przeglądarki po wystartowaniu serwera.

--host 0.0.0.0 --port 4210

Uruchomienie aplikacji: http://localhost:4210/

Angular - Projekt, Budowanie

ng build

Uruchamia proces budowania aplikacji.
Domyślnie, pliki wynikowe zapisywane są w katalogu /dist.

--prod

Budowanie projektu dla profilu produkcyjnego.
W trakcie generowania struktury projektu zostały utworzone dwa profile: nie produkcyjny i produkcyjny.
ng build bez flagi --prod uruchamia proces budowania dla profilu nie produkcyjnego.

Angular - Projekt, Testy

Jasmine

Biblioteka wspierająca tworzenie testów jednostkowych.

Karma

Środowisko uruchomieniowe dla testów jednostkowych.
Testy wykonywane w izolowanym środowisku na elementach aplikacji (bez tworzenia instancji aplikacji).

Protractor

Środowisko uruchomieniowe dla testów end-to-end.
Są to testy wykonywane na instancji aplikacji.

https://jasmine.github.io
http://karma-runner.github.io
http://protractortest.org

Angular - Projekt, Uruchamianie testów

ng test

Uruchamia testy jednostkowe (Karma, Jasmine)

ng e2e

Uruchamia testy end-to-end (Protractor, Jasmine)

Angular - Projekt, Struktura

Warstwy

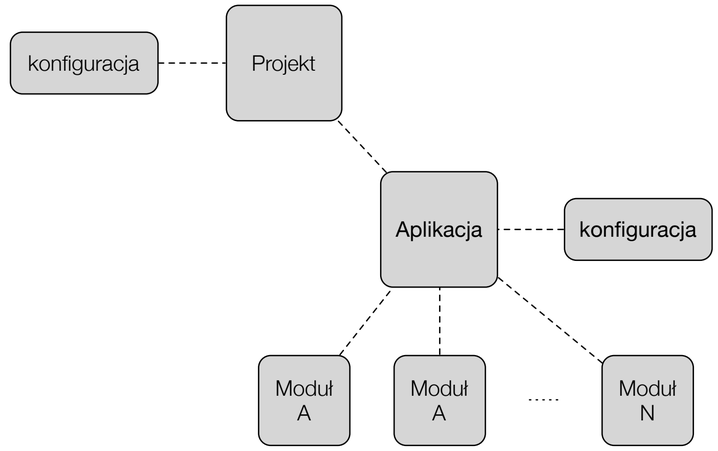
Angular posiada dwie warstwy struktury projektu.

Warstwa bazowa

Pliki konfigurujące projektu jako całości.

Warstwa aplikacji

Korzysta z konfiguracji warstwy bazowej.
Zawiera kod źródłowy i pliki konfigurujące specyficzne aspekty aplikacji.



Angular - Projekt, Struktura, Warstwa bazowa

/angular.json

Plik konfiguracyjny projektu.

/package.json

Plik konfiguracji zależności projektowych dla programu npm.

/node_modules

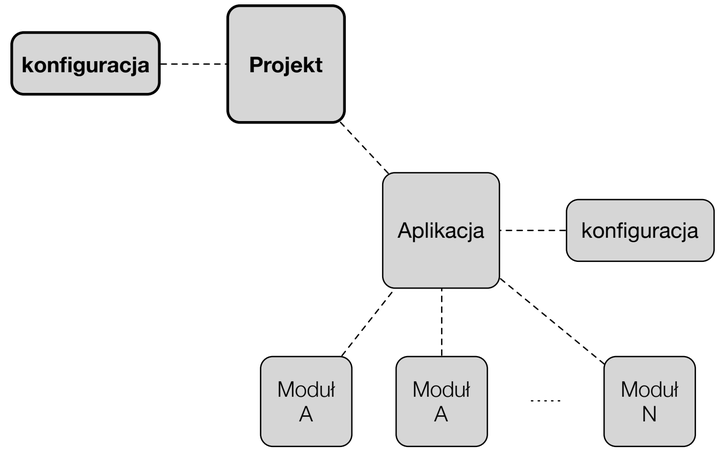
Katalog z bibliotekami zależności projektowych zarządzanymi przez program npm.

/tsconfig.json

Globalny plik konfiguracyjny kompilatora.

/tslint.json

Globalny plik konfiguracyjny statycznej analizy kodu.



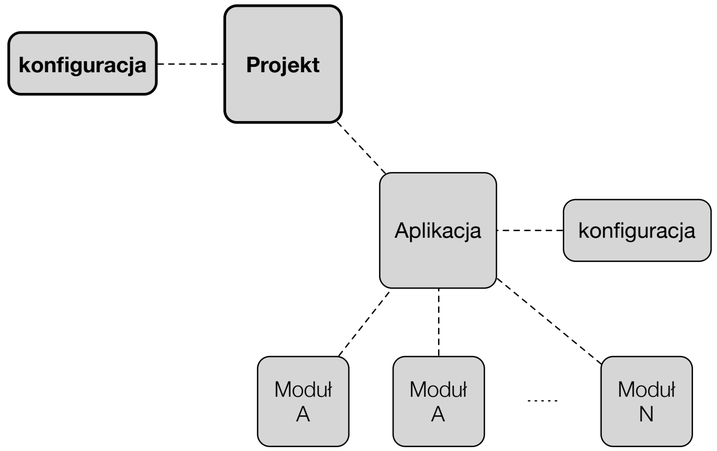
Angular - Projekt, Struktura, Warstwa bazowa

/e2e

Katalog z testami end-to-end oraz ich konfiguracją.

/e2e/protractor.conf.js

Plik konfiguracyjny środowiska testów end-to-end.



Angular - Projekt, Struktura, Warstwa aplikacji

/src

Katalog ze źródłami i konfiguracją aplikacji.

/src/app

Katalog z kodem źródłowym modułów aplikacji i kodem testów jednostkowych.

/src/app/app.module.ts

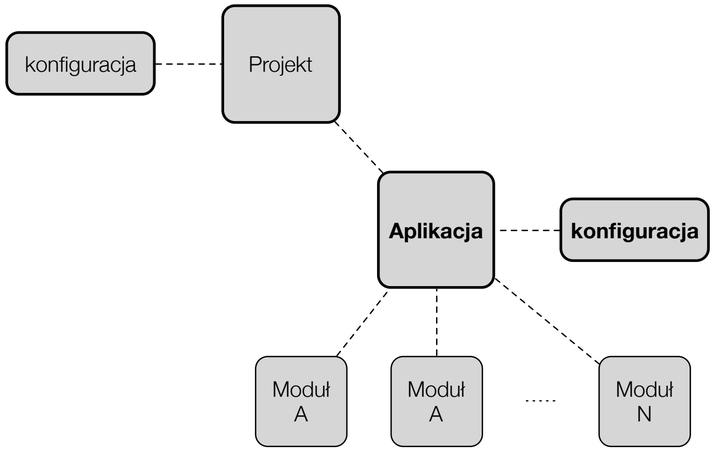
Plik konfigurujący artefakty składające się na moduł główny aplikacji.

/src/environments

Katalog z konfiguracją profili aplikacji.

/src/assets

Katalog z plikami, które zostaną przekopiowane do aplikacji w fazie budowania.
Najczęściej zawiera pliki graficzne.



Angular - Projekt, Struktura, Warstwa aplikacji

/src/karma.conf.js

Plik konfiguracyjny środowiska uruchamiania testów jednostkowych (Karma).

/src/test.ts

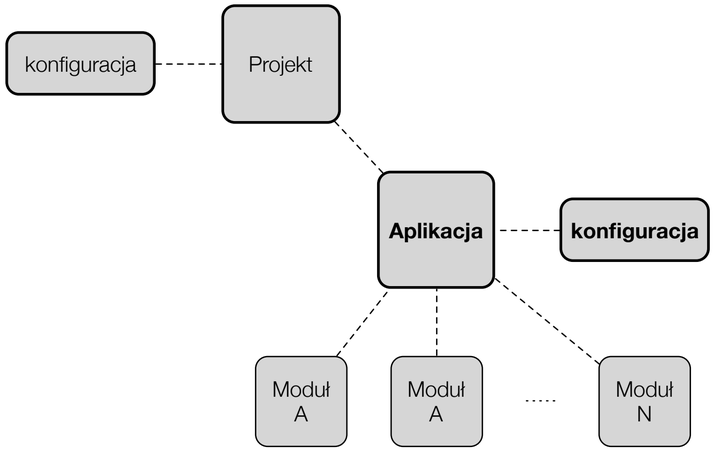
Plik sterujący tworzeniem środowiska testowego.

/src/main.ts

Plik źródłowy z kodem uruchamiającym aplikację.

/src/polyfills.ts

Plik dostarczający kod implementujący wymagane funkcjonalności nie wspierane przez przeglądarki, na których ma działać aplikacja.



Angular - Projekt, Struktura, Warstwa aplikacji

/src/tsconfig.app.json

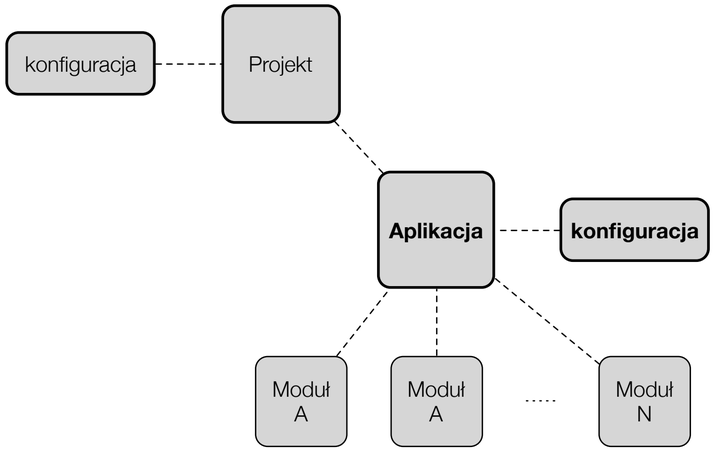
Plik konfiguracyjny kompilatora dla aplikacji.

/src/tsconfig.spec.json

Plik konfiguracyjny kompilatora dla testów jednostkowych.

/src/tslint.json

Plik konfiguracyjny statycznej analizy kod dla aplikacji.



Angular - CLI, ng

ng g component my-component

Generuje szkielet komponentu
--flat, pliki w /src/app,
--inline-style, brak pliku css,
--no-spec, brak pliku z testami,
--spec=false, brak pliku z testami,
--inline-template, brak pliku z template'em
--module=module-name, kontekst modułu

ng g service my-service

Generuje szkielet serwisu.
--flat, pliki w /src/app,
--module=module-name, kontekst modułu
--spec=false, brak pliku z testami,

ng g module my-module

Generuje szkielet modułu.

ng g interface my-interface

Generuje szkielet interfejsu.

ng g class my-class

Generuje szkielet klasy.

ng g enum my-enum

Generuje szkielet dla typu enum.

ng g directive my-directive

Generuje szkielet dyrektywy.

ng g pipe my-pipe

Generuje szkielet transformatora danych.

Angular

Moduł

Angular - Moduł

Moduł główny *(root module)*

Można zdefiniować więcej niż jeden moduł ale tylko jeden może być modułem głównym.
Przez konwencję modułowi głównemu nadawana jest nazwa AppModule.

main.ts

W pliku startowym aplikacji określa się, który moduł pełni rolę głównego.

```
//main.ts
import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule);
```

Angular - Moduł

Zbiór funkcjonalności

Moduł może zawierać dowolne artefakty: komponenty, serwisy, dyrektywy, logikę biznesową, itd.
Kod zgrupowany według wybranego klucza zapewniającego spójność.

Kontekst kompilacji *(compilation context)*

Z punktu widzenia kompilatora moduł jest zbiorem plików kodu źródłowego.
Z perspektywy parsowanego fragmentu HTML dostarcza kontekst kompilacji dla powiązanych z nim artefaktów.

Angular - Moduł

Widoczność

Artefakty składające się na pojedynczy moduł mogą mieć widoczność prywatną bądź publiczną.

Importowanie

Z poziomu jednego modułu można importować inny moduł.
Dostęp do publicznych artefaktów importowanego modułu.

Lazy-loading

Angular dostarcza modułom mechanizmu ładowania artefaktów na żądanie, w trakcie działania aplikacji.
Skraca ilość kodu ładowanego do pamięci przy starcie aplikacji.

Angular - Moduł

ng generate module <module-name>

Generuje nowy moduł.

@NgModule

Dekorator, funkcja przyjmująca jako argument obiekt z metadanymi definiującymi ten moduł.

```
//my.ts
import { NgModule } from '@angular/core';

@NgModule({
})
export class MyModule { }
```

Angular - Moduł

imports

Inne moduły, których upublicznione artefakty będą wykorzystywane w tym module.

declarations

Komponenty, dyrektywy i konwertery (pipes), które wchodzą w skład tego modułu.
Obecność artefaktu w tej sekcji nie oznacza, że będzie on widoczny na zewnątrz modułu.

exports

Podzbiór artefaktów, które będą dostępne na zewnątrz tego modułu.

providers

Serwisy z tego modułu, które będą dostępne z dowolnego miejsca w ekosystemie aplikacji.

```
//my.ts
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';


@NgModule({
  imports: [CommonModule],
  declarations: [],
  exports: [],
  providers: []
})
export class MyModule { }
```

Angular - Moduł

bootstrap

Definiuje główne komponenty (widoki) aplikacji.
Parametr ustawiana tylko w module głównym.

```
//my.ts
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';


@NgModule({
  imports: [CommonModule],
  declarations: [],
  exports: [],
  providers: [],
  bootstrap: []
})
export class MyModule { }
```

Angular - Moduł

BrowserModule

Zapewnia artefakty niezbędne do uruchomienia aplikacji w przeglądarce.
Importowany w module głównym.
Eksportuje moduł CommonModule.

CommonModule

Dostarcza dyrektyw takich jak NgIf, NgFor, używanych przy budowaniu struktury widoku.
BrowserModule nie powinien być importowany w innych modułach niż główny.

RouteModule

Odpowiada, za zarządzanie stanami i przejściami spomiędzy nimi w aplikacji.
Dla aplikacji web, zapewnia reprezentację zmiany stanu w w przestrzeni URL.

Angular - Moduł

FormsModule *(template-driven forms)*

Dostarcza artefakty ułatwiające definiowanie formularzy HTML.

ReactiveFormsModule

Dostarcza artefakty umożliwiające definiowanie formularzy w stylu reaktywnym.

HTTPClientModule

Moduł zapewniający artefakt klienta HTTP.

Angular

Komponent

Angular - Komponent

Komponent

Kontroluje fragment widoku aplikacji.
Zawiera kod (logikę komponentu):

- *dostarczający dane dla widoku,*
- *sterujący zdarzeniami (akcje użytkownika)*

Metadane

Dekorator @Component zawiera metadane konfigurujące komponent.

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

@Component({ ... })
export class MyComponent { ... }
```

Angular - Komponent, Metadane, Selecktor

Selektor

Określa selektor CSS dla komponentu.

Angular wstawia widok komponentu wszędzie tam, gdzie znajdzie tag HTML korespondujący z selektorem.

```
@Component({
  selector: 'my-component'
})
export class MyComponent { ... }
```

Angular - Komponent, Metadane, Widok

```
//inline template
template:
  `<div class="section">
     <h1>Hello Word!</h1>
     <p>inline</p>
   </div>
  `
```

Widok

Parametr templateUrl definiuje ścieżkę do pliku HTML definiującego widok komponentu.
Parametr template to miejsce definicji widoku (inline template) wewnątrz dekoratora @Component.

```
@Component({
  templateUrl: './my.component.html'
})
export class MyComponent { ... }
```

Angular - Komponent, Metadane, Style

```
//inline styles (layout and text)
styles:
  [
   `.section { margin: 20px; }`,
   `h1 { font-size: 28px; }
    p { font-size: 18px; }`
  ]
```

Style

Parametr sytelsURLs definiuje ścieżki do plików CSS definiujących lokalne style dla widoku komponentu.
Lokalne style komponentu nadpisują te zdefiniowane globalnie.
Parametr styles pozwala zdefiniować style (inline styles) wewnątrz dekoratora @Component

```
@Component({
  styleUrls: ['./my.component.css']
})
export class MyComponent { ... }
```

Angular - Komponent, Metadane, Zależności

```
class MyComponent {
  constructor(private myService: MyService) { }
}
```

Zależności

Parametr providers zawiera listę zależności, z których korzysta komponent.
Są to obiekty, wykorzystywane w kodzie komponentu, których instancji dostarcza Angulara.

```
import { MyService } from './MyService.service';
@Component({
  providers: [MyService]
})
export class MyComponent { ... }
```

Angular - Komponent, Data Binding

Data binding

Mechanizm przekazywania danych pomiędzy komponentem a widokiem.
Technika wywoływania zdarzeń (events) pomiędzy komponentem a widokiem.

Kierunek wiązania

Komunikacja pomiędzy komponentem a widokiem odbywa się na trzy sposoby:

- *w jedną stronę: w kierunku od komponentu do widoku,*
- *w jedną stronę: w kierunku od widoku do komponentu,*
- *w obie strony: w kierunku od komponentu do widoku i odwrotnie.*

```
@Component({ ... }))
export class MyComponent {
  myData: String;

  myAction() { ... }
}
```

Angular - Komponent, Data Binding, Interpolacja

```
<p>{{ myData }}</p>
```

one-way

Interpolacja {{ … }}

Przekazywanie danych z komponentu do widoku.

```
@Component({ ... }))
export class MyComponent {
  myData: String;

  myAction() { ... }
}
```

Angular - Komponent, Data Binding, Actions

```
<button (action)="myAction()">Action</button>
```

one-way

Zdarzenia (…)

Wyzwalanie akcji w komponencie z poziomu widoku (event).

```
@Component({ ... }))
export class MyComponent {
  myData: String;

  myAction() { ... }
}
```

Angular - Komponent, Data Binding, Two-way

```
<input [(ngModel)]="login">
```

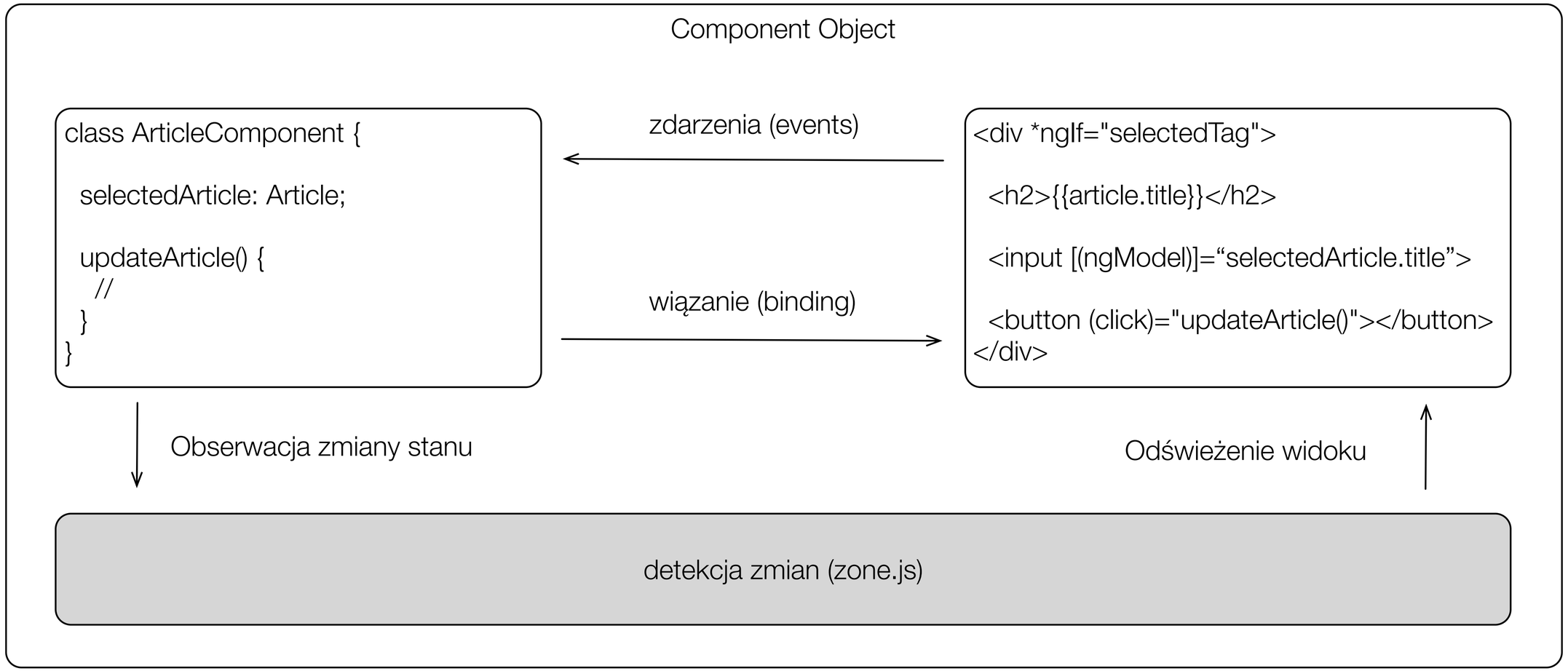
two-way

Two-way [(…)]

Przekazywanie danych z komponentu do widoku i ich aktualizacja w komponencie po zmianie wartości na widoku.

```
@Component({ ... }))
export class MyComponent {
  login: String;

  myAction() { ... }
}
```

Angular - Komponent, Data Binding, Big Picture



Angular

Template

Angular - Template

```
@Component({ ... }))
export class MyComponent {
  invoice: Invoice;
  basket: Basket;
}
```

one-way

Wyrażenia

Można stosować wyrażenia arytmetyczne oraz wywołania metod obiektów wyeksponowanych przez komponent.

```
//Dane pobierane z obiektu.
<div>Invoice number: {{invoice.id}}</div>

//Wyrażenie arytmetyczne, wywołanie metod na obiekcie.
<div>Koszt: {{basket.cost() + basket.shipmentCost()}}</div>
```

Angular - Template, Interpolacja

```
@Component({ ... }))
export class MyComponent {
  invoice: Invoice;
  basket: Basket;
  product: Product
}
```

one-way

Interpolacja

Istnieją trzy sposoby zapisu interpolacji.

Stosowana w przypadku: wartości (1), właściwości (properties) elementów (obiektów) DOM (2), arkuszy stylów (3).

```
//Wartość: interpolacja (1)
<h1>{{product.name}}</h1>
//Atrybut: interpolacja (2)
<img src={{product.imageUrl}}>
//Atrybut: [] lub bind- (3), notacja nazywana często: binding
<img [src]="product.imageUrl">
<img bind-src="product.imageUrl">
```

Angular - Template, Interpolacja vs Binding

```
@Component({ ... }))
export class MyComponent {
  isActive: boolean;
}
```

one-way

Interpolacja vs Binding

Interpolacja to alternatywa dla binding.
Z jednym wyjątkiem, żadna z form wiązania nie jest wyróżniona.
Należy stosować binding w przypadku atrybutów HTML (właściwości obiektów DOM), które nie posiadają wartości typu tekstowego.

```
//Atrybut
//Źle!! <button disabled={{!isActive}}>Action</button>
<button [disabled]="!isActive">Action</button>
<button bind-disabled="!isActive">Action</button>
```

Angular - Template, Interpolacja vs Binding

```
@Component({ ... }))
export class MyComponent {
  isVisible: boolean;
}
```

one-way

Interpolacja vs Binding

Interpolacja to alternatywa dla binding.
Z jednym wyjątkiem, żadna z form wiązania nie jest wyróżniona.
Należy stosować binding w przypadku atrybutów HTML (właściwości obiektów DOM), które nie posiadają wartości typu tekstowego.

```
//Atrybut
//Źle!! <div hidden={{!isVisibel}}>...</div>
<div [hidden]="!isVisibel">...</div>
<div bind-hidden="!isVisibel">...</div>
```

Angular - Template, Klasy arkuszy stylów

```
@Component({ ... }))
export class MyComponent {
  mainBox: string;
}
```

one-way

Klasy arkuszy stylów

Można stosować zarówno notację {{…}} jak i [class], bind-class

```
<div class={{mainBox}}></div>
<div [class]="mainBox"></div>
<div bind-class="mainBox"></div>
```

Angular - Template, Klasy arkuszy stylów

```
@Component({ ... }))
export class MyComponent {
  isSelected: boolean;
}
```

one-way

Klasy arkuszy stylów - warunkowe dołączenie klasy.

Poprawna notacja to: [class.] lub bind-class.**

```
//Źle!! <div class.selectedBox={{isSelected}}></div>
<div [class.selectedBox]="isSelected"></div>
<div bind-class.selectedBox]="isSelected"></div>
```

Angular - Template, Style

```
@Component({ ... }))
export class MyComponent {
  textColor: string;
}
```

one-way

Style

Można stosować zarówno notację {{…}} jak i [style.], bind-style.**

```
<div style.color={{textColor}}></div>
<div [style.color]="textColor"></div>
<div bind-style.color="textColor"></div>
```

Angular - Template, Style

```
@Component({ ... }))
export class MyComponent {
  isNew: boolean;
  isSpecial: boolean;
  task: Task;

  boxClasses =  {
    'new': this.isNew,
    'special': this.isSpecial,
    'done':  this.task.isDone
  };
}
```

one-way

NgClass

Zarządzanie większą liczbą klas jednocześnie.

```
//css
.new { ... }
.special { ... }
.done { ... }
```

```
<div [ngClass]="boxClasses"></div>
```

Angular - Template, Style

```
@Component({ ... }))
export class MyComponent {
  isNew: boolean;
  isSpecial: boolean;
  task: Task;

  boxStyles =  {
    'border-style': this.isNew ? 'solid' : 'none',
    'background-color': this.isSpecial ? 'green' : 'white',
    'border-color':  this.task.isDone ? 'black' : 'green'
  };
}
```

one-way

NgStyle

Zarządzanie większą liczbą stylów jednocześnie.

```
<div [ngStyle]="boxStyles"></div>
```

Angular - Template, Input

```
@Component({ ... }))
export class MyComponent {
  name: String;

  myAction() { ... }
}
```

two-way

NgModel

Przekazywanie danych z komponentu do widoku i ich aktualizacja w komponencie po zmianie wartości na widoku.

```
<input [(ngModel)]="name">
```

Angular - Template, Input

```
@Component({ ... }))
export class MyComponent {
  name: String;

  myAction(event) { ... }
}
```

one-way

NgModel - value

Przekazywanie danych z komponentu do widoku (elementu input).

```
<input value={{name}}>
<input [value]="name">
<input bind-value="name">
```

Angular - Template, Input

```
@Component({ ... }))
export class MyComponent {
  name: String;

  myAction(event) { ... }
}
```

one-way

NgModel - input

Przekazywanie danych z widoku (elementu input) do komponentu.

```
<input (input)="name=$event.target.value">
```

Angular - Template, Input

```
@Component({ ... }))
export class MyComponent {
  name: String;

  myAction(event) { ... }
}
```

two-way

NgModel - value, input

Dwustronne wiązanie danych.

```
<input [(ngModel)]="name">
```

```
<input [value]="name" (input)="name=$event.target.value" />
```

Angular - Template, Input

```
@Component({ ... }))
export class MyComponent {
  name: String;

  myAction(event) { ... }
}
```

two-way

NgModel, ngModelChange

Dwustronne wiązanie danych z wykorzystaniem ngModelChange.

```
<input [(ngModel)]="name">
```

```
<input [ngModel]="currentHero.name" (ngModelChange)="myAction($event)">
```

Angular - Template, Input

```
@Component({ ... }))
export class MyComponent {
  selectedItem: Item;
  items: Array<Item>;

}
```

two-way

NgModel - select

Dwustronne wiązanie danych dla elementu select.

```
<select [(ngModel)]="selectedItem">
  <option *ngFor="let item of items" [value]=“item">
    {{item.name}}
  </option>
</select>
```

Angular - Template, Dyrektywy atrybutów

NgClass

Dynamiczna podmiana zbioru klas CSS.

NgStyle

Dynamiczna podmiana zbioru styli CSS.

NgModel

Dwustronne wiązanie danych.

Angular - Template, Akcje

```
@Component({ ... }))
export class MyComponent {
  select() { ... }
  buy(event) { ... }
}
```

one-way

Akcje

Dwie notacje na podpinanie akcji do elementów HTML.

```
//Akcja kliknięcia podpięta pod element HTML
<div (click)="select()"></div>

//Podpięcie akcji w notacji z prefixem: on-*
<div on-click="select()"></div>
```

Angular - Template, Akcje

```
@Component({ ... }))
export class MyComponent {
  select() { ... }
  buy(event) { ... }
}
```

one-way

$event

Do akcji można przekazać jako atrybut obiekt $event.

```
//Obiekt $event (JS) przekazany do metody jako parametr.
<button (click)="buy($event)"></div>
```

Angular - Template, Variable

Zmienna w szablonie

Zmienna referencyjna, lokalna, zdefiniowana wewnątrz szablonu (template variable).
Definiowana z prefixem #.
Widoczność zmiennej ograniczona tylko do szablonu, w którym została zdefiniowana.

```
<input #title></input>
<button (click)="save(title.value)">Save</button>
```

Angular - Templat, NgIf

```
@Component({ ... }))
export class MyComponent {
  message: Message;
}
```

Instrukcja warunkowa - if

Dodaje lub usuwa element do drzewa DOM (element sterowany) w zależności od zdefiniowanego warunku.
W tym przypadku element sterowany określony jest przez miejsce wystąpienia instrukcji if.

```
<div *ngIf="message.isreceived"></div>
```

Angular - Templat, NgIf

```
@Component({ ... }))
export class MyComponent {
  isActive: boolean;
}
```

then

Zewnętrzna definicja elementu sterowanego instrukcją if

```
<div *ngIf="isActive; then active"></div>

<ng-template #active> ... </ng-template>
```

Angular - Templat, NgIf

```
@Component({ ... }))
export class MyComponent {
  isActive: boolean;
}
```

else

Definiuje element dodawany do drzewa DOM w przypadku gdy warunek instrukcji if ma wartość false.

```
<div *ngIf="isActive; else inactive"></div>
<ng-template #inactive> ... </ng-template>
```

Angular - Templat, NgIf

```
@Component({ ... }))
export class MyComponent {
  isActive: boolean;
}
```

if, then, else

Instrukcja warunkowa w pełnej postaci.

```
<div *ngIf="isActive; then active; else inactive”></div>

<ng-template #active> ... </ng-template>
<ng-template #inactive> ... </ng-template>
```

Angular - Template, NgFor

```
@Component({ ... }))
export class MyComponent {
  items: Item[];
}
```

Pętla - for

Dodaje kolekcję elementów do drzewa DOM.

```
<div *ngFor="let item of items">{{item.name}}</div>
```

Angular - Template, NgFor

```
@Component({ ... }))
export class MyComponent {
  items: Item[];
}
```

Pętla - for, index

Konstrukcja for przewiduje możliwość dostępu do aktualnego indeksu.

```
<div *ngFor="let item of items;
             let i = index">{{i}}: {{item.name}}</div>
```

Angular - Template, NgFor

```
@Component({ ... }))
export class MyComponent {
  items: Item[];

  trackById(index: number, item: Item): Item {
    return item ? item.id : undefined;
  }
}
```

Pętla - for, efektywność działania

Ryzyko wystąpienia kaskadowego uaktualniania elementów drzewa DOM w przypadku:

- *pojedynczej zmiany stanu elementu kolekcji,*
- *dodania lub usunięcia elementu z kolekcji,*
- *ponownego przypisania kolekcji do zmiennej*

trackBy

Rozwiązuje problem kaskadowego uaktualniania drzewa DOM.

Wskazuje metodę w komponencie dzięki której NgFor określa, które elementy drzewa DOM kolekcji należy uaktualniać.

```
<div *ngFor="let item of items;
                trackBy: trackById"</div>
```

Angular - Template, NgSwitch

Instrukcja warunkowa - switch

```
<div [ngSwitch]="box.category">
  <list *ngSwitchCase="'list'" [list]="box.items"></list>
  <details *ngSwitchCase="'details'"[item]="selectedItem"></details>
  <summary *ngSwitchCase="'summary'" [list]="box.items"></summary>
  <graph  *ngSwitchDefault [list]=“box.items"></graph>
</div>
```

Angular

Interakcja komponentów

Angular - Interakcja komponentów

Interakcja pomiędzy komponentami

Komponenty mogą współdziałać na kilka sposobów:

- *przekazywanie danych za pośrednictwem szablonu html (@Input)*
- *wstrzykiwanie instancji (@ViewChild),*
- *poprzez emisje zdarzenia (EventEmmiter, @Output)*

Angular - Interakcja komponentów, @Input

```
//Child component
@Component({
  selector: 'child',
  template: `<div>cItem.name</div>`
}))
export class CComponent {
  @Input cItem: Item;
}
```

@Input

Przekazywanie danych pomiędzy komponentami za pośrednictwem szablonu html.

```
@Component({
  template: `<child [cItem]="pItem"></child>`
}))
export class PComponent {
  pItem: Item;
}
```

Angular - Interakcja komponentów, @Input

```
@Component({
  selector: 'child',
  template: `<div>cItem.name</div>`
})
export class CComponent {
  @Input('cItemName') cItem: Item;
}
```

@Input

Opcjonalnie można zdefiniować własną nazwę dla atrybutu (właściwość w obiekcie DOM).

```
@Component({
  template: `<child [cItemName]="pItem"></child>`
})
export class PComponent {
  pItem: Item;
}
```

Angular - Interakcja komponentów, @Output

@Output

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

@Component({
  selector: 'child',
  template: `<div (click)="action(1)">...</div>`
}))
export class CComponent {
  @Output()
  emitter: EventEmitter<any>();

  action(value: any) { this.emitter.emit(value); }
}
```

```
@Component({
  template: `<child (emitter)="pAction($event)"></child>`
}))
export class PComponent {
  pAction(event) { ... }
}
```

Angular - Interakcja komponentów, @Output

@Output

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

@Component({
  selector: 'child',
  template: `<div (click)="action(1)">...</div>`
})
export class CComponent {
  @Output('emitterName') emitter: EventEmitter<any>();

  action(value: any) { this.emitter.emit(value); }
}
```

```
@Component({
  template: `<child (emitterName)="pAction($event)"></child>`
}))
export class PComponent {
  pAction(event) { ... }
}
```

Angular - Interakcja komponentów, @ViewChild

```
@Component({
  selector: 'child'
}))
export class CComponent { ... }
```

@ViewChild

```
import {ViewChild} from ‘@angular/core’;

@Component({ template: `<child></child>` }))
export class PComponent {
  @ViewChild(CComponent)
  child: CComponent;
}
```

Angular - Interakcja komponentów, @ViewChild

```
@Component({
  selector: 'child'
}))
export class CComponent { ... }
```

AfterViewInit

```
import {AfterViewInit} from ‘@angular/core’;

@Component({ template: `<child></hild>` }))
export class PComponent implements AfterViewInit {
  @ViewChild(CComponent) child: CComponent;

  ngAfterViewInit() { this.child; }
}
```

Angular - Interakcja komponentów, @ViewChildren

```
@Component({
  selector: 'child'
}))
export class CComponent { ... }
```

@ViewChildren

```
import {ViewChildren, QueryList} from ‘@angular/core’;

@Component({ template: `<child></child><child></child>` }))
export class PComponent {

  @ViewChildren(CComponent)
  children: QueryList<CComponent>;
}
```

Angular - Interakcja komponentów, @ViewChildren

AfterViewInit

```
@Component({
  selector: 'child'
}))
export class CComponent { ... }
```

```
import {AfterViewInit} from ‘@angular/core’;

@Component({ template: `<child></child><child></child>` }))
export class PComponent implements AfterViewInit {
 @ViewChildren(CComponent) children: QueryList<CComponent>;

   ngAfterViewInit() {
     let cList: CComponent[] = this.children.toArray();
   }
}
```

Angular

HTTPClient

Angular - HTTPClient, Moduł

```
//...
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  imports: [
    //...
    HttpClientModule,
  ],
})
export class AppModule {}
```

HTTPClient

W postaci obiektu HTTPClient Angular dostarcza funkcjonalność komunikacji z serwerem.

Angular - HTTPClient, Serwis

Wstrzykiwanie obiektu HTTPClient

Typowy scenariusz użycia klienta http to wstrzyknięcie instancji HTTPClient do obiektów pełniących rolę serwisów. Serwisy do obiekty tworzące warstwę dostępu do danych dla komponentów.

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

@Injectable()
export class PostsService {

  constructor(private http: HttpClient) { }
}
```

Angular - HTTPClient, Request

```
@Injectable()
export class PostsService {

  constructor(private http: HttpClient) { }
}
```

HTTPClient - GET, POST, PUT, DELETE

HTTPClient dostarcza interfejs pozwalający wykonywać zapytania przewidziane standardem HTTP.

```
getItems() {
  return this.http.get('/api/items');
}
```

Angular - HTTPClient, Observable

```
@Injectable()
export class PostsService {
  requestURL: '/api/items';

  constructor(private http: HttpClient) { }
}
```

Observable

HTTPClient wykonuje zapytania asynchroniczne.

Metody zapytań zwracają obiekt Observable, pozwalający obsłużyć późniejszą odpowiedź z serwera.

```
getItems(): Observable<any> {
  return this.http.get('/api/items');
}
```

Angular - HTTPClient, Komponent

```
@Injectable()
export class PostsService {
  requestURL: '/api/items';

  constructor(private http: HttpClient) { }
  getItems(): Observable<any> { ... }
}
```

Dostęp do serwisu z poziomu komponentu

Typowy scenariusz wykorzystania obiektu serwisowego wstrzyknięcie go do odpowiedniego komponentu.

```
//...
import { ItemService } from '../services/item.service';

@Component({ ... })
export class ItemComponent {
  constructor(private postsService: ItemService){}
}
```

Angular - HTTPClient, OnInit

OnInit w kontekście pobierania danych

Metoda cyklu życiu komponentu, wewnątrz której najczęściej używa się metod serwisowych mających dostarczyć danych dla komponentu.

```
//...
import { ItemService } from '../services/item.service';
import { Item } from '../model/post'

@Component({ ... })
export class ItemComponent implements OnInit {

  items: Item[];

  ngOnInit(): void { ... }
}
```

```
ngOnInit(): void {
  this.itemService.getItems();
}
```

Angular - HTTPClient, Observable, Subscribe

```
@Injectable()
export class PostsService {
  requestURL: '/api/items';

  constructor(private http: HttpClient) { }
  getItems(): Observable<any> { ... }
}
```

```
//...
import { ItemService } from '../services/item.service';
import { Item } from '../model/post'

@Component({ ... })
export class ItemComponent implements OnInit {

  items: Item[];

  ngOnInit(): void { ... }
}
```

Subscribe

Odpowiedź z serwera przetwarzana jest w metodzie subscribe obiektu observable.

```
ngOnInit(): void {
  this.itemService.getItems()
    .subscribe((data: Items[])  => { this.items = data; });
}
```

Angular - HTTPClient, Observable, Subscribe

```
@Injectable()
export class PostsService {
  requestURL: '/api/items';

  constructor(private http: HttpClient) { }
  getItems(): Observable<any> { ... }
}
```

```
//...
import { ItemService } from '../services/item.service';
import { Item } from '../model/post'

@Component({ ... })
export class ItemComponent implements OnInit {

  items: Item[];

  ngOnInit(): void { ... }
}
```

Subscribe i obiekt błędu

W metodzie subscribe można obsłużyć zarówno scenariusz powodzenia jak i niepowdzenia realizacji zapytania HTTP.

```
ngOnInit(): void {
  this.itemService.getItems()
    .subscribe(
      (data: Items[])  => { this.items = data; },
      (error: HttpErrorResponse) => { ... }
    );
}
```

Angular - HTTPClient, Observable, Subscribe

```
class HttpErrorResponse extends HttpResponseBase implements Error {
  get name: 'HttpErrorResponse'
  get message: string
  get error: any | null
  get ok: false

  get headers: HttpHeaders
  get status: number
  get statusText: string
  get url: string | null
  get ok: boolean
  get type: HttpEventType.Response | HttpEventType.ResponseHeader
}
```

HttpErrorResponse

Obiekt modelujący błąd związany z wykonaniem zapytania HTTP.
Rozróżnia się dwie kategorie błędów:

- *błąd po stronie serwera (modelowany odpowiednim statusem HTTP)*
- *błąd po stronie klienta (np.: problem z siecią, request nie wykonany)*

Angular - HTTPClient, Observable, Subscribe

HttpErrorResponse - błąd po stronie serwera

W zmiennej error znajduje się body ramki HTTP z odpowiedzią

```
class HttpErrorResponse extends HttpResponseBase implements Error {
  get name: 'HttpErrorResponse'
  get message: string
  get error: any
  get ok: false

  get headers: HttpHeaders
  get status: number
  get statusText: string
  get url: string | null
  get ok: boolean
  get type: HttpEventType.Response | HttpEventType.ResponseHeader
}
```

Angular - HTTPClient, Observable, Subscribe

HttpErrorResponse - błąd po stronie klienta

W zmiennej error znajduje się obiekt typu ErrorEvent

```
class HttpErrorResponse extends HttpResponseBase implements Error {
  get name: 'HttpErrorResponse'
  get message: string
  get error: ErrorEvent
  get ok: false

  get headers: HttpHeaders
  get status: number
  get statusText: string
  get url: string | null
  get ok: boolean
  get type: HttpEventType.Response | HttpEventType.ResponseHeader
}
```

Angular - HTTPClient, Observable, Subscribe

ErrorEvent

W zmiennej error znajduje się obiekt Javascript typu ErrorEvent.

ErrorEvent.message

Zmienna zwierająca opis błędu.

```
class HttpErrorResponse extends HttpResponseBase implements Error {
  get name: 'HttpErrorResponse'
  get message: string
  get error: ErrorEvent
  get ok: false

  get headers: HttpHeaders
  get status: number
  get statusText: string
  get url: string | null
  get ok: boolean
  get type: HttpEventType.Response | HttpEventType.ResponseHeader
}
```

Angular

Cykl życia komponentu

Angular - Cykl życia komponentu

constructor()

Konstruktor komponentu (nie wywoływany jawnie z kodu).

ngOnInit()

Wywołana raz po inicjalizacji komponentu, po konstruktorze.
Po pierwszym wywołaniu ngOnChanges(), po inicjalizacji input properties - @Input()

ngOnDestroy()

Wywołana raz przed zniszczeniem komponentu.

Angular - Cykl życia komponentu

ngOnChanges(change)

Wywoływana przy każdej zmianie input properties - @Input().
Przekazany parametr zawiera starą i nową wartość zmiennej.

ngDoCheck()

Wywołana po ngOnInit() (za pierwszym razem) oraz po każdym ngOnChanges().
Wykorzystywana do manipulacji danymi nie zwartami w parametrze change metody ngOnChanges(change)

Angular - Cykl życia komponentu

```
ngOnChanges(changes: SimpleChanges) {
  let property = changes[propertyName];
  property.currentValue;
  property.previousValue;
}
```

Angular - Cykl życia komponentu

ngAfterViewInit()

Wywoływane po inicjalizacji template'u dla komponentu.

ngAfterViewChecked()

Wywoływana po zmianie na widoku komponentu.

Angular - Cykl życia komponentu

```
ngAfterViewChecked() {
    if (this.item == this.child.item) {
      //...
    }
  }
```

```
//parent-template.html
<child></child>
```

```
//parent-component.ts
@ViewChild(ChildComponent) child: ChildComponent;
```

Angular

Pipe

Angular - Pipe

@Pipe

Dekorator wykorzystywany przy klasach modelujących transformacje danych.

```
@Pipe({name: 'name'})
export class NamePipe { }
```

Angular - Pipe

```
{{ 'value' | name:param }}
```

```
interface PipeTransform {
  transform(value: any, ...args: any[]): any
}
```

PipeTransform

Interfejs, który implementują klasy dekorowane przez @Pipe.

```
@Pipe({name: 'name'})
export class NamePipe implements PipeTransform {
  transform(value: any, param: any) { }
}
```

Angular - Pipe

```
{{ 'value' | name:pA:pB }}
```

```
interface PipeTransform {
  transform(value: any, ...args: any[]): any
}
```

PipeTransform

Interfejs, który implementują klasy dekorowane przez @Pipe.

```
@Pipe({name: 'name'})
export class NamePipe implements PipeTransform {
  transform(value: any, paramA: any, paramB: any) { }
}
```

Angular

Dependency Injection

Angular - Dependency Injection

@Injectable()

Najczęściej stosowana w przypadku klas pełniących rolę serwisów.
Dekoruje klasę, która będzie mogła podlegać mechanizmowi wstrzykiwania.

Mechanizm wstrzykiwania nie ogranicza się jedynie do serwisów.
Może być stosowany przy dowolnej klasie, również w przypadku klas komponentów.
@Component, @Pipe są podtypem @Injectable().

Angular - Dependency Injection

```
@Injectable()
class InvoiceService { ... }
```

Angular - Dependency Injection

```
import {Http} from '@angular/http';
import {Injectable} from “@angular/core”;

@Injectable()
export class InvoiceService {
  constructor(private http:Http) { ... }
 //...
}
```

Angular - Dependency Injection

Providers

Zawiera informacje dla Angular'a w jaki sposób tworzyć instancje obiektów w procesie wstrzykiwania w obiektach docelowych.

Angular - Dependency Injection

```
providers:[InvoiceService]
```

Krótka forma definicji tworzenia instancji obiektu.

```
providers:[{provide:InvoiceService, useClass:InvoiceService}]
```

Wersja pełna. Definicja typu i nazwy.

Angular - Dependency Injection

```
providers:[ProductService, InvoiceService]
```

Więcej niż jedna definicja w wersji krótkiej.

```
providers:[
  {provide:InvoiceService, useClass:InvoiceService},
  {provide:ProductService, useClass:ProductService}
]
```

Więcej niż jedna definicja w wersji pełnej.

Angular - Dependency Injection

```
constructor(productService: ProductService,
            invoiceService: InvoiceService) {

  this.productService = productService;
  this.invoiceService = invoiceService;
}
```

Wstrzykiwanie zależności.

Angular - Dependency Injection

```
constructor(@Inject(ProductService) productService,
            @Inject(InvoiceService) invoiceService) { ... }
```

Bezpośrednie podanie typu konstruktora (ES6)

Angular - Dependency Injection

```
providers:[{provide:MY_DATA, useValue:'abcdefgh'}]
```

Wstrzykiwanie wartości.

Powstaje problem unikalności tokenów.

Angular - Dependency Injection

```
export const MY_DATA  = new OpaqueToken('MY_DATA');
```

Zapewnia unikalny token.

```
providers:[{provide: MY_DATA, useValue: 'abcdefgh'}]
```

Definiuje sposób tworzenia danych.

```
constructor(@Inject(MY_DATA) public myData: string)
```

Wstrzykuje dane.

Angular

Routing

Angular - Routing, Base URL

Base URL

Wskazuje bazowy adres URL dla wszystkich relatywnych adresów w aplikacji.

```
<base href="/">
```

Angular - Routing

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

Angular Router

Odpowiedzialny jest za:

- *mapowanie adresów URL na komponenty*
- *nawigacje pomiędzy widokami (komponentami) z poziomu adresów URL*
- *zapewnienie poprawnej obsługi akcji back w przeglądarce - akcji wracania do poprzedniej strony.*

Singleton service

Aplikacja posiada jedną globalną instancję serwisu routera.

Może być importowany wielokrotnie - raz w każdym module (lazy-loading).

Angular - Routing, Konfiguracja w module

```
:~/app$ ng generate module app-routing --flat --module=app
```

Definicja routingu

Routing można zdefiniować w pliku konfigurującym moduł.
Zalecane jest definiowanie routingu w osobnym, dedykowanym do tego celu pliku.

```
//app.module.ts

import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { AppRoutingModule } from './app-routing.module';

@NgModule({
  imports: [AppRoutingModule]
})
export class AppModule { }
```

```
//app-routing.module.ts

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

@NgModule({
  exports: [ RouterModule ]
})
export class AppRoutingModule {}
```

Angular - Routing, Rejestracja

Rejestracja głównego routingu

Główny routing (top-level routing) rejestrowany jest przez metodę RootModule.forRoot(routes: Routes)

Rejestracja routingu dla modułów lazy-loading

Rejestrowane przez metodę RootModule.forChild(routes: Routes)

Angular - Routing, Rejestracja

Routes

Obiekt, w którym definiowane jest mapowanie ścieżek na komponenty.

RouterModule.forRoot

Rejestracja głównego routingu.

```
//app-routing.module.ts

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

const routes: Routes = []

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

Angular - Routing, Konfiguracja

Mapowanie ścieżki na komponent

Powiązanie ścieżki z komponentem definiują parametry path oraz component.

```
const routes: Routes = Routes = [
  {
    path: 'articles',
    component: ArticlesListComponent
  }
];
```

Angular - Routing, Konfiguracja

Ścieżka domyślna

Konfiguruje domyślny routing dla aplikacji w przypadku braku ścieżki w adresie url.

```
const routes: Routes = Routes = [
  {
    path: '',
    component: AppComponent
  }
];
```

Angular - Routing, Konfiguracja

Wildcard

Konfiguracja dla adresów URL, które nie pasują do żadnego z wcześniej zdefiniowanych mapowań.

First-match wins

Kolejność konfigurowanych mapowań ma znaczenie.
Bardziej szczegółowe ścieżki powinny być konfigurowane przed bardziej ogólnymi.

```
const routes: Routes = Routes = [
  {
    path: '**',
    component: PageNotFoundComponent
  }
];
```

Angular - Routing, Konfiguracja

Parametry routingu

Parametry zdefiniowane w ścieżce pozwalają na wymianę danych pomiędzy komponentami.

```
const routes: Routes = Routes = [
  {
    path: 'article/:id',
    component: ArticlesComponent
  }
];
```

Angular - Routing, Konfiguracja

Dodatkowe dane dołączane do routingu

Poprzez parametr data można dołączyć do routingu dane typu read-only.

```
const routes: Routes = Routes = [
  {
    path: 'article/:id',
    component: ArticlesComponent,
    data: { title: '', info: ''}
  }
];
```

Angular - Routing, RouterOutlet

Miejsce renderowania routingu

Komponenty są renderowane w miejscu wskazanym przez RouterOutlet.

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

Angular - Routing, RouterLink

Linki w szablonach HTML

RouterLink pozwala na zdefiniowanie linków wewnątrz szablonów.

Klasy CSS dla aktywnego linku

RouterLinkActive pozwala na zdefiniowanie klasy CSS dla aktywnego odnośnika.

```
<a routerLink="/articles"
   routerLinkActive="nav-item-active">Articles</a>

<a [routerLink]="['/article']">Detail</a>
```

Angular - Routing, RouterLink

Parametry w linkach

Parametry do ścieżek można dodać z wykorzystaniem interpolacji.

```
<a routerLink="/article/{{article.id}}">Detail</a>
<a [routerLink]="[‘/article/',article.id]">Detail</a>
```

Angular - Routing, RouterLink

Dynamiczne generowanie linków

RouterLink łatwo integruje się z ngFor.

```
<div *ngFor="let article of collection">
  <a [routerLink]="['/article/',article.id]">Detail</a>
</div>
```

Angular - Routing, Router

Nawigacja z poziomu komponentu

Obiekt Router dostarcza metodę navigate().

```
//path: /articles
this.router.navigate(['articles']);

//path: /article/:id
this.router.navigate(['articles', 1])
```

Angular - Routing, Router

Nawigacja z poziomu komponentu

Obiekt Router dostarcza metodę navigate().

```
//path: /items/:catId/:subCatId
this.router.navigate(['items', 1, 2]);

//path: /items/:catId/subCategory/:subCatId
this.router.navigate(['articles', 1, 'subCategory', 2]);
```

Angular - Routing, Router

Nawigacja z poziomu komponentu

Obiekt Router dostarcza metodę navigate().

```
//parametry opcjonalne
//path: /article/:id
this.router.navigate(['article',  1, {optParamA:12, optParamB:11}]);
```

Angular - Routing, Router

Pobieranie informacji o routingu

Dane routingu dostępne są w obiekcie Router.
Obiekt Route możne być użyty w dowolnym miejscu aplikacji.

RouterState

Umożliwia poruszanie się po drzewie stanów routingu.

ActivatedRoute

Obiekt drzewa nawigacji tworzonego przez Router. Przechowuje stan routingu (przekazane parametry, query).

```
let router: Router;
let routerState: RouterState = router.routerState;
let child: ActivatedRoute = routerState.root.firstChild;
```

Angular - Routing, Router

Wstrzykiwanie obiektu Route

Obiekt route może być użyty w dowolnym miejscu aplikacji.

Wstrzykiwanie obiektu ActivatedRoute

Szybki sposób na dostęp do parametrów aktualnego routingu.

```
constructor(private route: Router) { }
constructor(private routeData: ActivatedRoute) { }
```

Angular - Routing, Router

```
interface ParamMap {
  get keys: string[]
  has(name: string): boolean
  get(name: string): string | null
  getAll(name: string): string[]
}
```

```
type Params = {
    [key: string]: any;
};
```

Dostęp do parametrów

Obiekt AcitvatedRoute dostarcza interfejs umożliwiający dostęp do parametrów adresu URL.

```
interface ActivatedRoute {
  params: Observable<Params>
  queryParams: Observable<Params>
  data: Observable<Data>

  get paramMap: Observable<ParamMap>
  get queryParamMap: Observable<ParamMap>
}
```

Angular - Routing, Location

Nawigacja - back / forward z poziomu komponentu

Obiekt Location zapewnia funkcjonalność nawigowania po historii przeglądarki

```
constructor(private location: Location) { }

location.back();
location.forward();
```

Angular

Formularze

Angular - Forms

Template-Driven Forms

Formularz stworzony w szablonie HTML.
Pola formularza powiązane z komponentem poprzez sytem wiązania danych.
Drzewo formularza tworzone (asynchronicznie) przez framework.
Skutkuje mniejszą ilością kodu po stronie komponentu.
Z powdu asynchronicznej natury może utrudnić pisanie testów jednostkowych.

Reactive Forms

Formularz stworzony w szablonie HTML.
Po stronie komponentu drzewo formularza tworzone (synchronicznie) przez programistę.

Dynamic Forms

Pozwala na budowanie formularzy na podstawie konfiguracji (metadanych).
Ułatwia pracę, kiedy zaistnieje potrzeba stworzenia wielu formularzy.
Szczególnie przydatna forma w przypadku budowania wielu podobnych do siebie formularzy.

Angular - Forms, Import

FormsModule

Moduł do tworzenia formularzy typu template forms.

```
//Konfiguracja modułu dla template forms

import { FormsModule } from '@angular/forms';

@NgModule({
  imports: [
    FormsModule
    //...
  ],
})
export class AppModule { }
```

ReactiveFormsModule

Moduł pozwalający budować formularze typu reactive forms

```
//Konfiguracja modułu dla reactive forms

import { ReactiveFormsModule } from '@angular/forms';

@NgModule({
  imports: [
    ReactiveFormsModule
    //...
  ],
})
export class AppModule { }
```

Angular - Forms, Template-driven

```
@Component({ ... })
export class FormComponent {
  firstName = '';

  action() { ... }
}
```

Powiązanie danych pomiędzy formularzem a komponentem

Wykorzystany jest standardowy mechanizm wiązania danych - NgModel.

Akcja submit

Zdarzenie NgForm.ngSubmit wykorzystywane jest do powiązania metody z komponentu z akcją submit na formularzu.

```
<form (ngSubmit)="action()">
  <label for="first-name">First Name</label>
  <input type="text"
         id="first-name"
         [(ngModel)]=firstName
         maxlength=8
         required>

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

Angular - Forms, NgForm

Stan formularza

Zmienna szablonowa, do której przypisany jest obiekt formularza (referencja do dyrektywy NgForm) to wygodny sposób dostępu do informacji o stanie formularza wewnątrz szablonu

```
<form #form="ngForm">
  ...
</form>
```

Angular - Forms, NgForm, Valid

Blokada akcji submit

Stan walidacji danych formularza można odczytać z NgForm.valid lub NgForm.invalid

```
<form (ngSubmit)="action()" #form="ngForm">
  <button type="submit"
          [disabled]="!form.valid">Action</button>
</form>
```

Angular - Forms, NgForm, Reset

```
@Component({ ... })
export class FormComponent {
  firstName = '';

  action() { ... }
  clear() { ... }
}
```

Akcja reset

NgForm posiada metodę reset(), która wykorzystywana jest do wyczyszczenia danych w formularzu.
W przykładzie dodatkowo wywołana jest metoda clear() zdefiniowana w komponencie..

```
<form (ngSubmit)="action()" #form="ngForm">
  <button type="button"
          (click)="clear(); form.reset()">Reset</button>
</form>
```

Angular - Forms, NgForm

Stan edycji formularza

Z NgForm można odczytać dodatkowe dane mówiące o tym czy

- *formularz został kliknięty: NgForm.touched, NgForm.untouched*
- *została zmieniona wartość w formularzu: NgForm.dirty, NgForm.pristine*

Angular - Forms, NgModel

```
//div z informacja walidacyjną
<div [hidden]="nameField.valid || nameField.pristine”>
```

Stan poszczególnych pól formularza

Stan pola formularza można odczytać poprzez odwołanie się do referencji ngModel.

Podobnie jak w NgForm, do dyspozycji mamy właściwości:

- *NgMode.valid, NgModelInvalid*
- *NgModel.touched, NgModel.untouched*
- *NgModel.dirty, NgModel.pristine*

```
<input type="text"
       id="first-name"
       [(ngModel)]=firstName
       #nameField=ngModel
       maxlength=8
       required>
```

Angular - Forms, Template, Dedykowany walidator

Dedykowany walidator

Wymaga napisania dyrektywy i zarejestrowania jej tak aby angular uwzględnił ją w procesie walidacji.

```
<input type="text"
       id="email"
       [(ngModel)]=emailAddress
       regExValidator="emailRegEx"
       #emailField=ngModel>
```

Angular - Forms, Template, Dedykowany walidator

Dedykowany walidator

Dyrektywa zarejestrowana jako NG_VALIDATORS.
Implementuje interfejs Validator.

```
@Directive({
  selector: '[regExValidator]',
  providers: [{provide: NG_VALIDATORS, useExisting: RegExValidatorDirective, multi: true}]
})
export class RegExValidatorDirective implements Validator {
  @Input('regExValidator') pattern: string;

  validate(control: AbstractControl): {[key: string]: any} | null {
    return pattern ? regExValidator(new RegExp(this.pattern, 'i'))(control) : null;
  }
}
```

Angular - Forms, Walidacja

Walidator dla template-driven forms

Dodawanie walidatorów poprzez atrybuty w szablonie html.

Walidator dla reactive forms

Dodawanie funkcji walidatora bezpośrednio do obiektu FormControl z poziomu komponentu.

Angular - Forms, Reactive, Walidacja

Walidator dla template-driven forms

Dodawanie walidatorów poprzez atrybuty w szablonie html.

Walidator dla reactive forms

Dodawanie funkcji walidatora bezpośrednio do obiektu FormControl z poziomu komponentu.

Angular - Forms, Reactive, Dedykowany walidator

Dedykowany walidator

Wymaga zdefiniowania funkcji zwracającej ValidatorFn

```
export function regExValidator(pattern: RegExp): ValidatorFn {

  return (control: AbstractControl): {[key: string]: any} | null => {
    const result = pattern.test(control.value);
    return result ? {'result': {value: control.value}} : null;
  };
}
```

Angular - Forms, Reactive

Powiązanie danych pomiędzy formularzem a komponentem

Wykorzystywany jest w tym celu obiekt FormControl.

```
import {FormControl} from '@angular/forms';


@Component({ ... })
export class FormComponent {
  firstName: FormControl;

  constructor() {
    this.firstName = new FormControl();
  }

  action() { ... }
}
```

```
<form (ngSubmit)="action()">
  <label for="first-name">First Name</label>
  <input type="text"
         id="first-name"
         [formControl]=firstName
         maxlength=8
         required>

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

Angular - Forms, Reactive vs Template-Driven

Template-Driven

```
@Component({ ... })
export class FormComponent {
  firstName = '';

  action() { ... }
}
```

```
<form (ngSubmit)="action()">
  <label for="first-name">First Name</label>
  <input type="text"
         id="“first-name"
         [(ngModel)]=firstName
         maxlength=8
         required>

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

Reactive

```
@Component({ ... })
export class FormComponent {
  firstName: FormControl;

  constructor() {
    this.firstName = new FormControl();
  }

  action() { ... }
}
```

```
<form (ngSubmit)="action()">
  <label for="first-name">First Name</label>
  <input type="text"
         id="first-name"
         [formControl]=firstName
         maxlength=8
         required>

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

Angular - Forms, Reactive, Walidatory

Validators

Walidatory konfigurowane są z poziomu komponentu.
Klasa Validators dostarcza kilka wbudowanych walidatorów.

```
class Validators {
  static min(min: number): ValidatorFn
  static max(max: number): ValidatorFn
  static required(control: AbstractControl): ValidationErrors | null
  static requiredTrue(control: AbstractControl): ValidationErrors | null
  static email(control: AbstractControl): ValidationErrors | null
  static minLength(minLength: number): ValidatorFn
  static maxLength(maxLength: number): ValidatorFn
  static pattern(pattern: string | RegExp): ValidatorFn
  static nullValidator(c: AbstractControl): ValidationErrors | null
  static compose(validators: (ValidatorFn | null | undefined)[] | null): ValidatorFn | null
  static composeAsync(validators: (AsyncValidatorFn | null)[]): AsyncValidatorFn | null
}
```

Angular - Forms, Reactive, Walidatory

Konfiguracja walidatorów w komponencie

```
import {FormControl, Validators} from ‘@angular/forms';

@Component({ ... })
export class FormComponent {
  firstName: FormControl;

  constructor() {
    let emailRegEx = '';
    let firstNameRules = [Validators.required,
                          Validators.pattern(emailRegEx),
                          Validators.maxLength(8)];

    this.firstName = new FormControl(‘’, firstNameRules);
  }

  action() { ... }
}
```

Angular - Forms, Reactive, Walidatory

Stan pola formularza

Z poziomu obiektu FormControl można odczytać stan pola:

- *FormControl.valid, FormControl.invalid*
- *FormControl.touched, FormControl.untouched*
- *FormControl.dirty, FormControl.pristine*

```
import {FormControl, Validators} from ‘@angular/forms';

@Component({ ... })
export class FormComponent {
  firstName: FormControl;

  constructor() { ... }

  action() {
    if(this.firstName.valid) {
      //...
    }
  }
}
```

Angular - Forms, Reactive, Dedykowany walidator

Dedykowany walidator

```
import {FormControl, Validators} from ‘@angular/forms';
import { regExValidator } from ‘../regex-validator.directive';

@Component({ ... })
export class FormComponent {
  firstName: FormControl;

  constructor() {
    let pattern = '';
    let firstNameRules = [Validators.required,
                          Validators.maxLength(8),
                          regExValidator(pattern)];

    this.firstName = new FormControl(‘’, firstNameRules);
  }

  action() { ... }
}
```

Testowanie

Jasmine

/spec

Folder z testami jednostkowymi.

/src

Folder z kodem źródłowym.

/lib

Folder z kodem biblioteki Jasmine.

SpecRunner.html

Plik uruchamiający testy.

Jasmine - SpecRunner.html

```
<head>
  <!-- import skryptów biblioteki jasmine -->

  <!-- import skryptów z kodem przeznaczonym do testowania -->
  <script src="src/logic/Invoice.js"></script>
  <!-- ... -->

  <!-- import skryptów z testami, które mają być uruchomione -->
  <script src="spec/InvoiceSpec.js"></script>
  <!-- ... -->
</head>
```

Jasmine

describe

Funkcja tworząca blok/grupę dla testów.

xdescribe

Blok testów pomijanych.

```
describe("Suit A", function() {
  describe("A-1", function() {
    //tests
  });
  describe("A-2", function() {
    //tests
  });
});
```

Jasmine

it

Funkcja, w której definiowany jest pojedynczy test.
Test jest pomijany gdy znajduje się w funkcji xdescribe i wówczas nie występuje w podsumowaniu.

xit

Test oczekujący. Nie wykonuje się.
Występuje w podsumowaniu jako oczekujący.
Test jest pomijany gdy zdefiniowany jest w funkcji xdescribe ale występuje w podsumowaniu.

Test oczekujący

To taki, który:

- *jest zdefiniowany z pustym body*
- *lub wewnątrz definicji, którego wywołano funkcję pending()*
- *lub taki, który zdefiniowany jest przy pomocy xit.*

```
describe("Suit A", function() {
  it(“scenario 1", function(){
    //...
  });

  it("scenario 2", function(){
    //...
  });
});
```

Jasmine

expect

Funkcja, do której przekazywana jest wartość testowana.

Zwraca obiekt dostarczający mechanizm porównywania wartości testowanej z oczekiwaną.

```
describe("Numbers", function() {
    it("value should be equal 8", function() {
      let actualValue = 8;
      expect(actualValue);
    });
});
```

Jasmine

toEqual

Jedna z wielu funkcji porównujących.
Jako argument przyjmuje wartość oczekiwaną.

```
describe("Numbers", function() {
    it("value should be equal 8", function() {
      let actualValue = 8;
      expect(actualValue).toEqual(8);
    });
});
```

Jasmine

toMatch, not.ToMach

Porównywanie w kontekście wyrażenia regularnego.
expect(strPhone).toMatch(/\d{3}-\d{3}-\d{2}-\d{2}/)

toEqual, not.ToEqual

Porównywanie w kontekście operatora “jest równe”.
expect(number).toEqual(8)

toBe, not.toBe

Porównywanie w kontekście obiektów.
expect(obj).toEqual(myObj)

toBeDefined, not.ToBeDefined

Porównywanie w kontekście przypisania wartości.
expect(obj.property).toBeDefined()

toBeUndefined, not.ToBeUndefined

Porównywanie w kontekście “undefined”.
expect(myVariable).toBeUndefined()

Jasmine

toBeNull, not.ToBeNull

Porównywanie w kontekście wartości null.

```
expect(null).toBeNull()
expect(notNullVar).not.toBeNull()
expect(valueUndefined).not.toBeNull()
```

toBeTruthy, not.ToBeTruthy

Porównywanie w kontekście wartości logicznej true.

```
expect(true).toBeTruthy()
expect(false).not.ToBeTruthy()
```

toBeFalsy, not.ToBeFalsy

Porównywanie w kontekście wartości logicznej false.

```
expect(false).toBeFalsy()
expect(true).not.toBeFalsy()
```

Jasmine

toContain, not.ToContain

Poszukiwanie elementów w tablicy.
expect([1, 2, 4]).toContain(2)
expect([1, 2, 4]).not.toContain(5)

toBeLessThan, not.ToBeLessThan

Porównywanie w kontekście operatora < “jest mniejsze niż”.

toBeGreaterThan, not.ToBeGreaterThan

Porównywanie w kontekście operatora > “jest większe niż”.

toBeCloseTo, not.ToBeCloseTo

Porównywanie w kontekście operacji zaokrąglania.
expect(8.224).toBeCloseTo(8.2245, 2)
expect(8.224).not.toBeCloseTo(8.4, 2)

Jasmine

beforeEach

Funkcja wykonywana przed każdym testem.

afterEach

Funkcja wykonywana po każdym teście.

```
describe("invoice", function() {
  var invoice;
  beforeEach(function() {
    invoice = new Invoice();
  });
  afterEach(function() {
    invoice = null
  });
  it("test-1", function() { ... });
  it(“test-2", function() { ... });
});
```

Jasmine

beforeEach

Funkcja wykonywana przed każdym testem.

afterEach

Funkcja wykonywana po każdym teście.

this

Używane dla zmiennej globalnej w beforeEach, afterEach oraz it.

```
describe("invoice", function() {
  beforeEach(function() {
    this.invoice = new Invoice();
  });
  afterEach(function() {
    this.invoice = null
  });
  it("test-1", function() { ... });
  it(“test-2", function() { ... });
});
```