Wprowadzenie do Angular 7

Tworzenie serwisów Web 2.0

dr inż. Robert Perliński
rperlinski@icis.pcz.pl

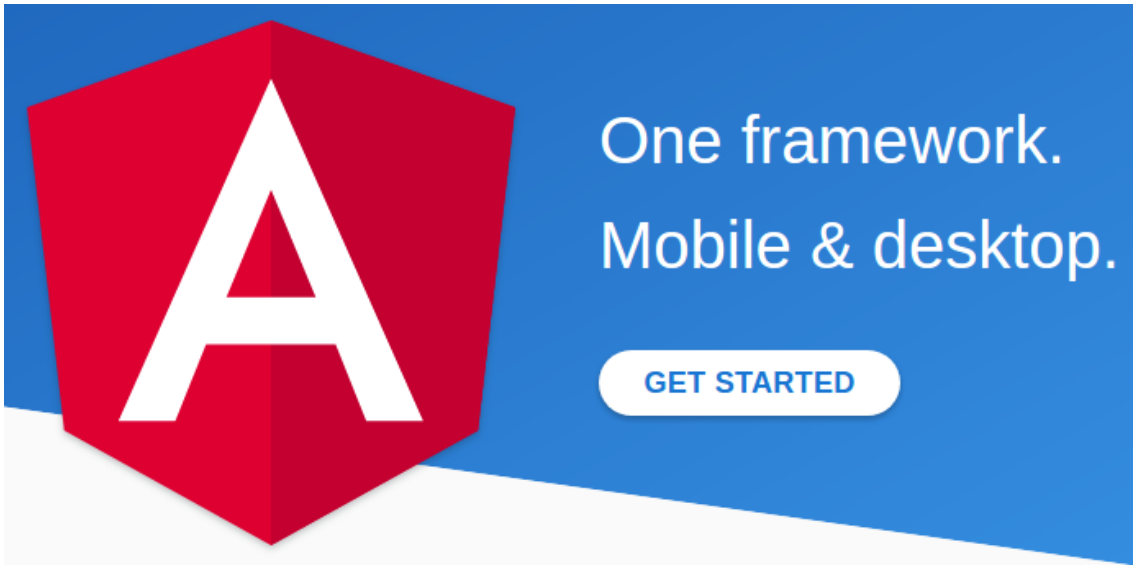
Politechnika Częstochowska
Instytut Informatyki Teoretycznej i Stosowanej

13 kwietnia 2019

Plan prezentacji

1. Trochę informacji o frameworku Angular
2. Angular CLI
3. Hello World w Angular
   - Przygotowanie projektu
   - Pierwszy komponent
4. Aplikacja działająca na serwerze
5. Aktualizacje, wydania, wersjonowanie Angulara
6. Źródła

Angular



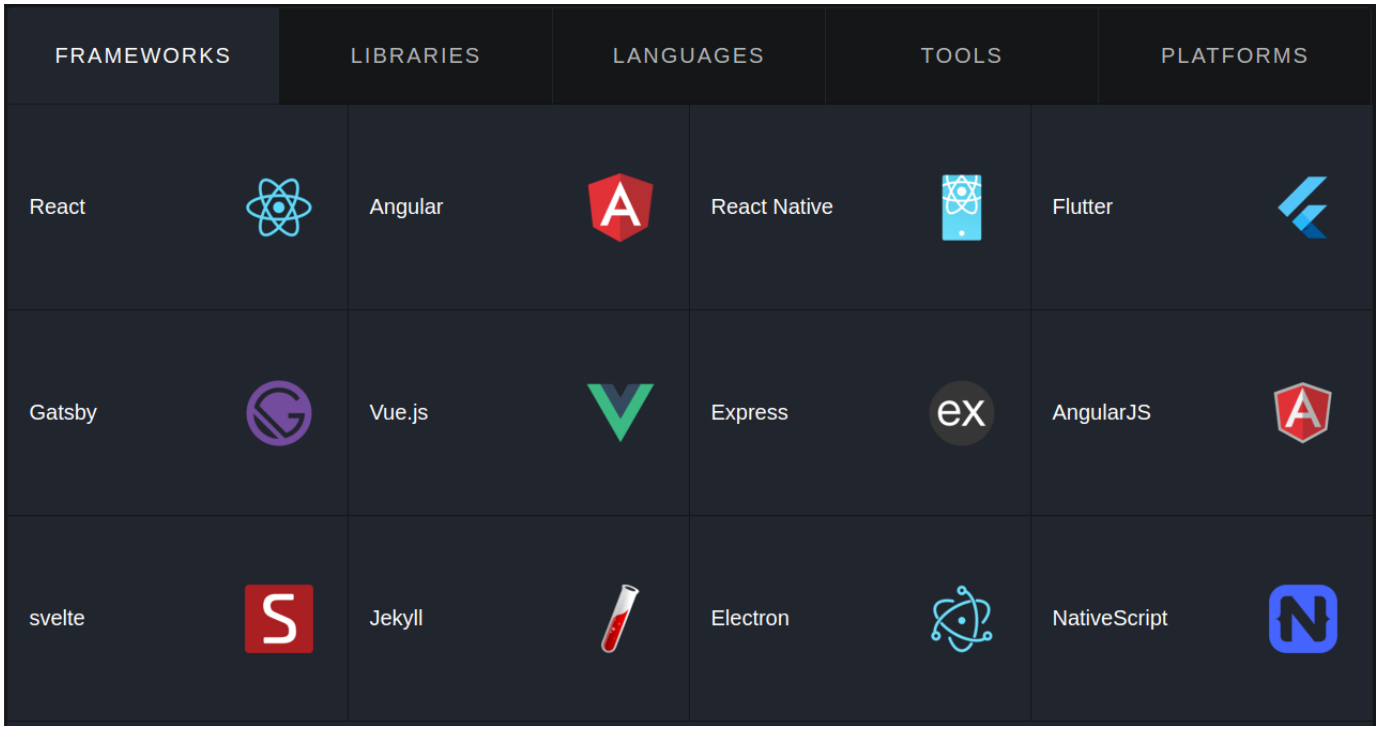
https://angular.io/

Różne kursy online:

- https://egghead.io/browse/frameworks/angular - kursy online do Angular
- https://www.udemy.com/learn-angular-from-scratch/
- https://ultimateangular.com/ - inne kursy, fajne ale dobrze kosztują
- https://www.codeschool.com/courses/accelerating-through-angular

egghead.io - nauka technologii internetowych

https://egghead.io



Cechy Angular - jeden kod na wszystkie platformy

Nowoczesne aplikacje sieciowe:

- wykorzystanie nowoczesnych możliwości platformy sieciowej pozwala na dostarczanie stron wyglądających i działających jak aplikacje desktopowe,
- wysoka wydajność,
- działaie bez dostępu do internetu,
- brak potrzeby instalacji.

Aplikacje natywne:

- tworzenie natywnych aplikacji korzystając z technologii:
  - Cordova - `https://cordova.apache.org/`,
  - Ionic Framework - `http://ionicframework.com/`,
  - Native Script - `https://www.nativescript.org/`

Aplikacje na komputery stacjonarne:

- pozwala tworzyć aplikacje z wersją instalacyjną na Mac, Windows i Linux'a,
- używamy tych samych metod co w technologiach sieciowych,
- możliwość korzystania z natywnego API danej platformy.

Cechy Angular - szybkoś i wydajność

Generowanie kodu:

- Angular przekształca szablony w kod wysoko zoptymalizowany dla dzisiejszych maszyn wirtualnych JavaScript,
- dostarcza wszystkich zalet kodu pisanego ręcznie z wykorzystaniem produktywności szablonu aplikacji.

Universalność:

- dostarcza podstawowy wygląd aplikacji zbudowanej w oparciu o Node.js, .NET, PHP czy inne serwery,
- renderuje widok do HTML i CSS niemal natychmiastowo,
- przygotowuje dobry grunt w aplikacji pod optymalizację SEO.

Podział kodu:

- szybkie wczytywanie aplikacji dzięki komponentowi Router (automatyczne dzielenie kodu),
- użytkownicy wczytują tylko ten kod, który jest potrzebny do wygenerowania żądanego widoku.

Cechy Angular - produktywność

Szablony:

- szybkie tworzenie widoków, interfejsów użytkownika, dzięki prostej ale potężnej składni szablonów.

Angular CLI:

- interfejs lini komand dla Angular, coś na wzór npm dla Node.js,
- szybkie tworzenie szablonu aplikacji,
- dodawanie komponentów i testów,
- natychmiastowe instalowanie/wdrażanie,
- `https://cli.angular.io/`

Zintegrowane środowisko programistyczne:

- inteligentne dopełnianie kodu,
- wykrywanie błędów i inne informacje zwrotne w popularnych edytorach.

Cechy Angular - pełne wsparcie przy tworzeniu

Testowanie:

- środowisko Karma dla testów jednostkowych, informacja o błędach przy każdym zapisywaniu projektu (`https://karma-runner.github.io/`),
- Protractor zapewnia szybkość i stabilność scenariuszom testowym (`https://www.protractortest.org`).

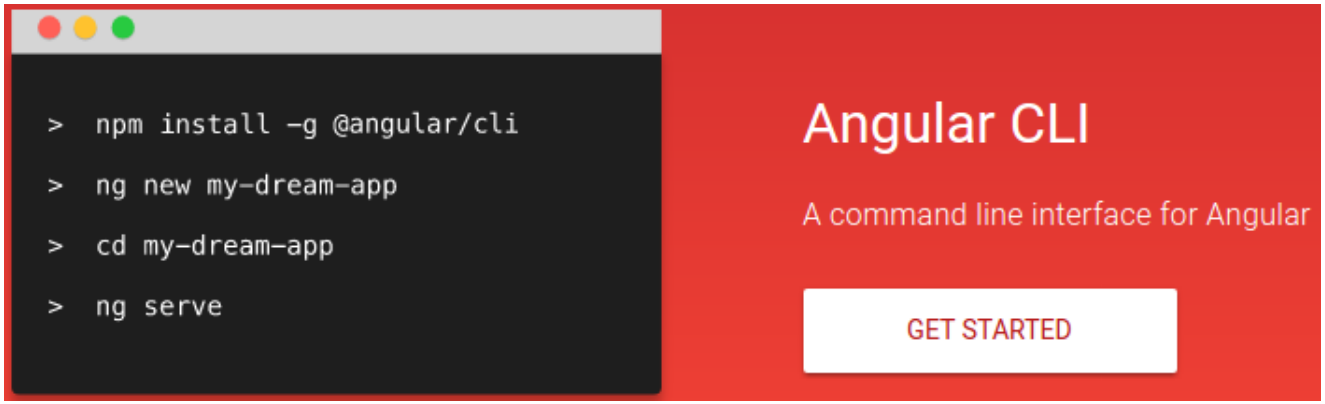
Animacje:

- tworzenie skomplikowanych choreografi i przebiegów czasowych animacji wysokiej wydajności,
- wystarczy mała ilość kodu dzięki intuicyjnemu API Angular.

Dostępność (Accessibility):

- tworzenie dostępnych aplikacji z komponentami dla ARIA (aplikacje internetowe wysokiej dostępności, np. dla osób niepełnosprawnych),
- przewodniki dla programistów (jak tworzyć wysoko dostępne aplikacje),
- wbudowana infrakstuktura do testowania dla projektu a11y (`http://a11yproject.com/`).

Angular CLI



`https://cli.angular.io/`

Instalacja i najważniejsze polecenia:

- `sudo npm install -g @angular/cli`
- `ng help` - informacja o dostępnych poleceniach
- `ng version` - wersja narzędzia Angular CLI (obecnie 7.1.4)
- `ng new` - tworzy nowy katalog z nową aplikacją wewnątrz
- `ng serve` - buduje i uruchamia aplikację, automatyczne przebudowanie przy zmianie pliku, domyślny port to 4200, zmiana hosta i portu:
  `ng serve --host 0.0.0.0 --port 4201`

Angular CLI

Najważniejsze polecenia:

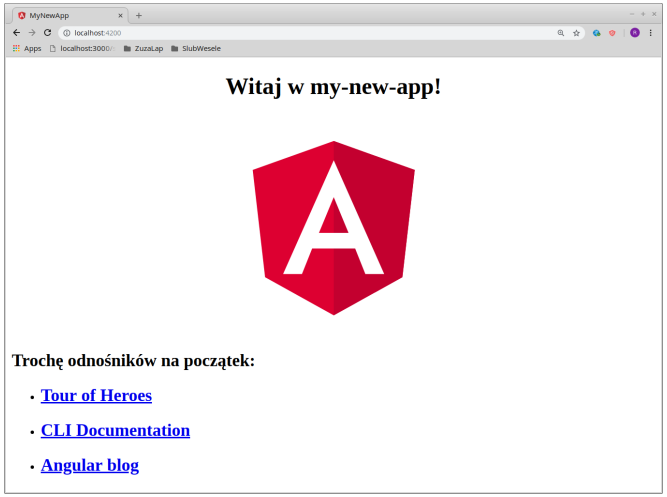
- `ng add` - dodanie wsparcia dla wybranej biblioteki do projektu
- `ng generate` - pozwala na generowanie komponentów, dyrektyw, klas, interfejsów, modułów, usług, strunieni, ...; przykład:

```
$ ng generate component users-list
CREATE src/app/users-list/users-list.component.css (0 bytes)
CREATE src/app/users-list/users-list.component.html (29 bytes)
CREATE src/app/users-list/users-list.component.spec.ts (650 bytes)
CREATE src/app/users-list/users-list.component.ts (284 bytes)
UPDATE src/app/app.module.ts (410 bytes)

$ ng g service clients
CREATE src/app/clients.service.spec.ts (380 bytes)
CREATE src/app/clients.service.ts (136 bytes)
```

- `ng build` - kompilacja aplikacji do katalogu wynikowego (domyślnie `dist`)
- `ng update` - aktualizacja aplikacji do najnowszej wersji

Hello World w Angular



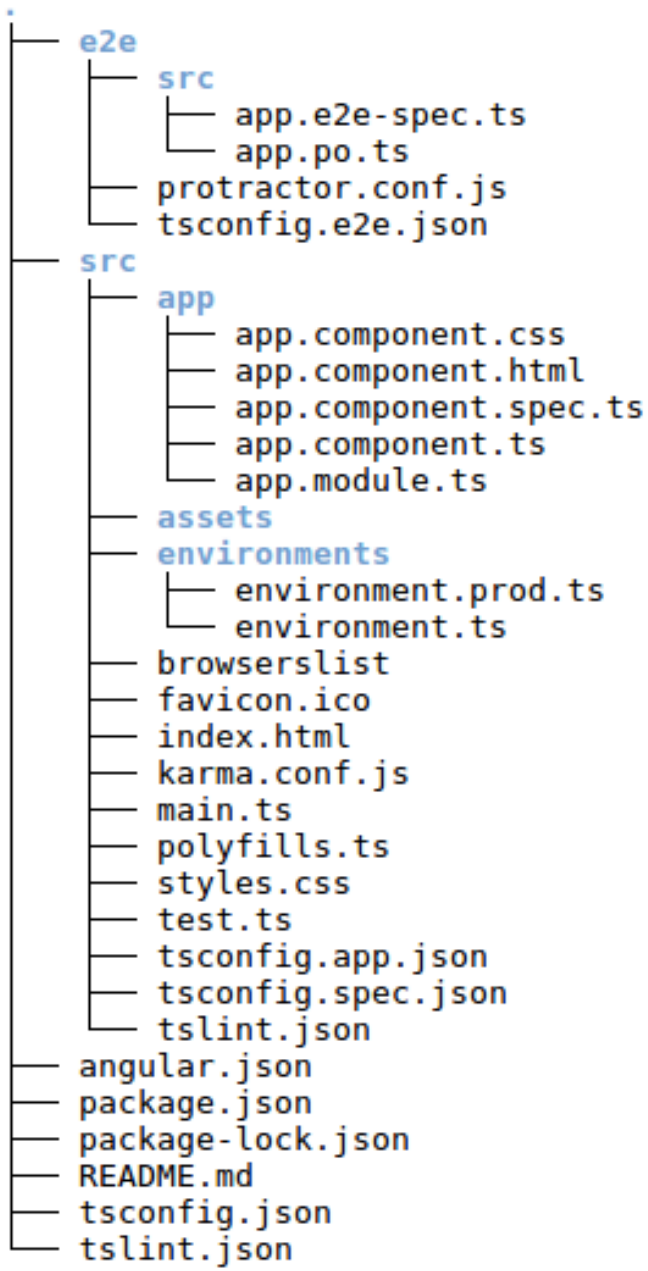
Przygotowanie "pustego" projektu czyli Hello World:

- instalacja `node` (wersja 8.x lub wyższa), `npm` (wersja 5.x lub wyższa) i `ng`
- utworzenie przykładowego, niemal "pustego" projektu (320 MB):

```
ng new my-app
cd my-app
ng serve
// ng serve --open
```

- warto zobaczyć zawartość utworzonego, "pustego" projektu
- domyślnie po utworzeniu projekt ma zainicjowane repozytorium git'a

Pełna struktura utworzonej aplikacji (src i reszta)



Zawartość katalogu src I

Aplikacja znajduje się w folderze `src`. Wszystkie elementy Angular, szablony, style, obrazy i wszystko, czego potrzebuje aplikacja, znajduje się w `src`. Wszelkie pliki spoza `src` służą do tworzenia aplikacji.

- `app/app.component.{ts,html,css,spec.ts}`
  definicja głównego komponentu aplikacji (`AppComponent`) razem z szablonem HTML, arkuszem stylów CSS i testami jednostkowymi (*.spec.ts). Wraz z rozwojem aplikacji przerodzi się w drzewo zagnieżdżonych komponentów.
- `app/app.module.ts`
  definicja głównego modułu aplikacji (`AppModule`), który określa całą jej budowę. Na początku zawiera tylko jeden komponent - `AppComponent`, później będzie więcej.
- `assets/*`
  Folder, w którym można umieszczać obrazy i inne elementy, które mają być kopiowane hurtowo podczas tworzenia aplikacji.

Zawartość katalogu `src` II

- `environments/*` Folder zawiera jeden plik dla każdego ze środowisk docelowych (produkcyjnych), każdy eksportuje proste zmienne konfiguracyjne, używane w aplikacji, np. `production: false`
  Można używać różnych plików do środowiska deweloperskiego, testowego czy produkcyjnego.
- `browserslist`
  Plik konfiguracyjny biblioteki udostępniającej docelowe przeglądarki między różnymi narzędziami dla front-end takimi jak: Autoprefixer, Babel, postcss-preset-env, ...
- `favicon.ico` - Ikona na pasku zakładek.
- `index.html`
  Główna strona HTML wyświetlana przy odwiedzaniu aplikacji. Zwykle nie trzeba go edytować. Angular CLI dodaje do niego automatycznie wszystkie pliki js i css podczas budowania. Nie potrzeba dodawać elementów `<script>` i `<link>`.

Zawartość katalogu src III

- `karma.conf.js`
  Konfiguracja testów jednostkowych dla narzędzia Karma, wykorzystywana przy poleceniu `ng test`.
- `main.ts`
  Główny punkt wejścia do aplikacji. Określa kompilację aplikacji JIT przy `ng serve` albo `ng build`. Uruchamia główny moduł aplikacji (`AppModule`) do działania w przeglądarce. Można też użyć kompilatora AOT (Ahead-of-Time).
- `polyfills.ts`
  Różne przeglądarki mają różne poziomy wsparcia dla standardów internetowych. Polyfills pomaga znormalizować te różnice. Użycie `core-js` i `zone.js` powinno wystarczyć w znacznej większości przypadków.
- `styles.css`
  Dodajemy tutaj globalne style wykorzystywane w całej aplikacji. Można też tutaj importować inne style, pliki CSS. Większość styów będzie jednak określanych lokalnie razem z konkrentym komponentem.

Zawartość katalogu `src` IV

- `test.ts`
  Plik będący głównym punktem wejścia dla testów jednostkowych. Wymagany przez `karma.conf.js`. Wczytuje rekurencyjnie wszystkie pliki `*.spec` i frameworka Angular. Nie trzeba go ręcznie edytować.
- `tsconfig.app|spec.json`
  Konfiguracja kompilatora TypeScript dla aplikacji Angular (`tsconfig.app.json`) i dla testów jednostkowych (`tsconfig.spec.json`).
- `tslint.json`
  Dodatkowa konfiguracja Linting dla TSLint razem z Codelyzer, używana podczas uruchamiania ng lint. Linting pomaga zachować spójność stylu kodu.

Pozostałe pliki katalogu głównego I

Pozostałe pliki katalogu głównego (`root`) pomagają budować, testować, konserwować, dokumentować i wdrażać aplikację. Pliki te znajdują się w folderze głównym obok `src`.

- `e2e/`
  Katalog `e2e` przechowuje testy end-to-end. Nie można ich przechowywać w `src` ponieważ są one osobną aplikacją, która testuje naszą główną aplikację. Dlatego ma osobny, własny plik `tsconfig.e2e.json`.
- `node_modules/`
  Folder utworzony przez Node.js. Zawiera wszytkie moduły wymagane w naszej aplikacji wymienione w `package.json`.
- `.editorconfig`
  Prosta konfiguracja edytora tekstowego, zapewnia taką samą konfigurację wszystkim programistom, np. kodowanie, rodzaj wcięć, nowa linia. Zobacz `http://editorconfig.org`.
- `.gitignore`
  Konfiguracja plików ignorowanych przez system kontroli wersji Git.

Pozostałe pliki katalogu głównego II

- `angular.json`
  Konfiguracja dla Angular CLI. Można ustawić niektóre wartości domyślne, jakie pliki będą dołączane podczas budowania.
- `package.json`
  Konfiguracja `npm` z pakietami wykorzystywanymi w aplikacji. Można dodać tutaj także własne niestandardowe skrypty.
- `protractor.conf.js`
  Konfiguracja testów kompleksowych dla narzędzia Protractor uruchamiania poleceniem `ng e2e`.
- `README.md`
  Plik z podstawową dokumentacją naszego projektu. Wstępnie zawiera informacje o poleceniach CLI. Ważny dla użytkowników naszego projektu!
- `tsconfig.json`
  Konfiguracja kompilatora języka TypeScript na naszego IDE.
- `tslint.json`
  Dodatkowa konfiguracja dla TSLint razem z Codelyzer, jak wyżej.

package.json - podstawowy zbiór pakietów

package.json

```
{
  "name": "my-app",
  "version": "0.0.0",
  "scripts": {
    ...
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "~7.1.0",
    "@angular/common": "~7.1.0",
    "@angular/compiler": "~7.1.0",
    "@angular/core": "~7.1.0",
    "@angular/forms": "~7.1.0",
    "@angular/platform-browser": "~7.1.0",
    "@angular/platform-browser-dynamic": "~7.1.0",
    "@angular/router": "~7.1.0",

    "core-js": "^2.5.4",
    "rxjs": "~6.3.3",
    "tslib": "^1.9.0",
    "zone.js": "~0.8.26"
  },
  "devDependencies": {
    ...
  }
}
```

package.json - podstawowy zbiór pakietów

package.json

```
{
  ...
  "scripts": {
    ...
  }, "private": true,
  "dependencies": {
    ...
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "~0.11.0",
    "@angular/cli": "~7.1.3",
    "@angular/compiler-cli": "~7.1.0",
    "@angular/language-service": "~7.1.0",
    "@types/node": "~8.9.4",
    "@types/jasmine": "~2.8.8",
    "@types/jasminewd2": "~2.0.3",
    "codelyzer": "~4.5.0",
    "jasmine-core": "~2.99.1",
    "jasmine-spec-reporter": "~4.2.1",
    "karma": "~3.1.1",
    "karma-chrome-launcher": "~2.2.0",
    "karma-coverage-istanbul-reporter": "~2.0.1",
    "karma-jasmine": "~1.1.2",
    "karma-jasmine-html-reporter": "^0.2.2",
    "protractor": "~5.4.0",
    "ts-node": "~7.0.0",
    "tslint": "~5.11.0",
    "typescript": "~3.1.6"
  }
}
```

package.json

Aplikacje Angular i sam Angular zależy od funkcjonalności dostarczanej przez innych dostawców. Zarządzamy tym z użyciem menadżera NPM. Zbiór pakietów przedstawiony w `package.json`:

- dobrze współpracuje ze sobą,
- zawiera wszystko co potrzebne do zbudowania przykładowej aplikacji
- i jest to więcej niż potrzeba w wielu aplikacjach,
- pobranie większej liczby pakietów nie wpływa negatywnie na działanie aplikacji,
- na serwer wysyła się tylko to, czego aplikacja naprawdę potrzebuje.

Zależności:

- `dependencies` - pakiety wymagane do działania aplikacji,
- `devDependencies` - pakiety wymagane tylko przy tworzeniu aplikacji
- kompilacja aplikacji do wersji produkcyjnej (zawartość katalogu `dist`):

```
ng build --prod --build-optimizer
```

package.json

Dwie kategorie wymaganych pakietów:

- użytkowe (ang. *feature*) - dostarczają naszej aplikacji podstawowe, użytkowe możliwości,
- pozostałe pakiety wspierające naszą aplikację, np. `bootstrap` czy `rxjs` albo dostarczające funkcjonalności, która nie jest obecna we wszystkich przeglądarkach, np. `core-js` czy `zone.js`.

package.json

Pakiety użytkowe:

- `@angular/core` - najważniejsze pakiety dla działania aplikacji, obsługa wszystkich metadanych dotyczących: dekoratorów, komponentów, dyrektyw, wstrzykiwania zależności oraz cyklu życia komponentów i związanych z tym zdarzeń i wywołań,
- `@angular/common` - często używane usługi, strumienie, dyrektywy dostarczone razem z Angular,
- `@angular/compiler` - kompilator szablonów dostarczany razem z Angular,
- `@angular/platform-browser` - obsługa funkcjonalności związanej z DOM i przeglądarką,
- `@angular/platform-browser-dynamic` - dostarcza i uruchamia metody pozwalające aplikacji kompilować szablony po stronie klienta, używany do uruchamiania aplikacji podczas jej tworzenia,
- `@angular/http` - klient http w Angular,
- `@angular/router` - komponent do obsługi routingu.

package.json

Pozostałe pakiety:

- `core.js` - Modularna biblioteka standardowa dla JavaScript. Zawiera polyfills dla ECMAScript 5, ECMAScript 6: obietnice, symbole, kolekcje, iteratory, wpisane tablice, propozycje ECMAScript 7+, setImmediate, itp.
- `rxjs` - Reactive Extensions Library for JavaScript; odpowiada za wybór właściwej wersji typów obserwowanych (ang. *Observable type*) bez konieczności oczekiwania na wydanie nowej wersji Angular,
- `zone.js` - obsługa właściwego kontekstu wywoływania funkcji asynchronicznych; tutaj również chodzi o uniezależnienie tej funkcjonalności od wersji Angular.

Inne:

- `angular2-in-memory-web-api` - biblioteka służąca do symulowania internetowego api, dobra we wczesnej fazie projektu,
- `bootstrap` - HTML i CSS framework, ładny wygląd, tworzenie RWD,
- ...

package.json - devDependencies

Pakiety wykorzystywane przy tworzeniu:

- `@angular/cli`, `@angular/compiler-cli` - Angular CLI
- `@types/node` - definicje typów z Node.js dla TypeScript
- `codelyzer` - analizator kodu,
- `karma`, `karma-jasmine` - testy jednostkowe w frameworku Jasmine (`https://jasmine.github.io/`)
- `typescript` - obsługa języka TypeScript i kompilator `tsc`,
- `tslint` - narzędzie do statycznej analizy kodu w TypeScript,
- ...

package.json - skrypty

package.json - scripts

```
{
  "name": "my-app",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e"
  },
  "private": true,
  "dependencies": {
    ...
  },
  "devDependencies": {
    ...
  }
}
```

package.json - skrypty

Skrypty (uruchamiane poleceniem `ng ...` albo `npm run`):

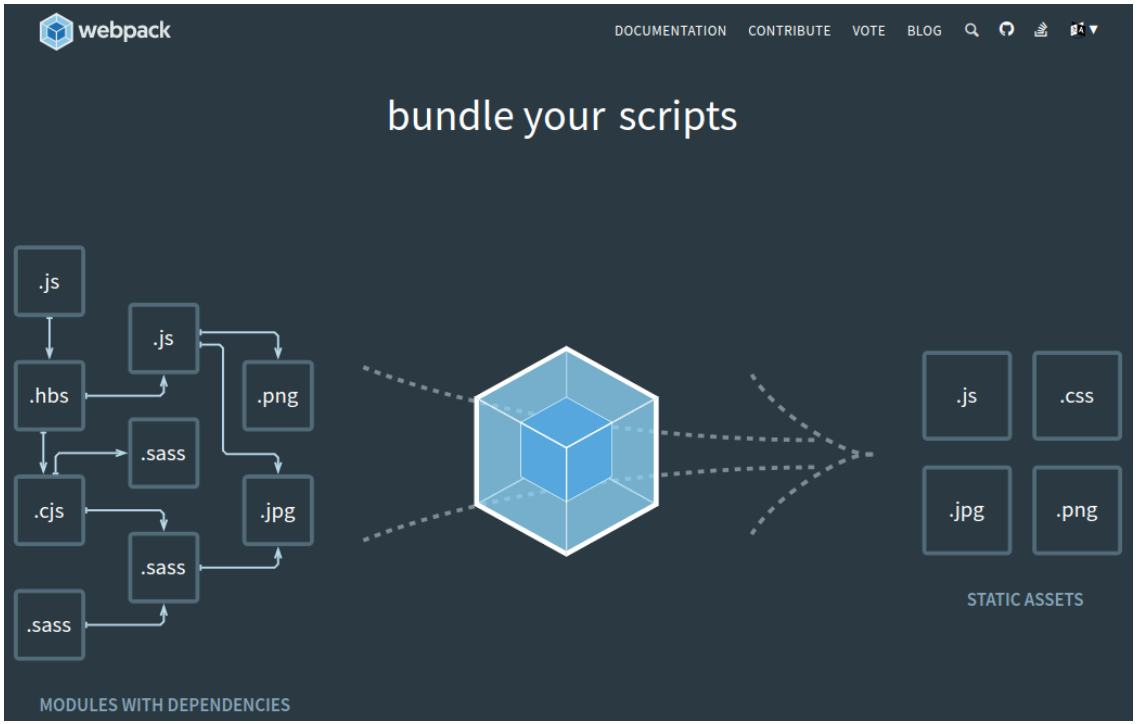
- `ng serve` albo `npm start` - uruchamia kompilator i serwer w tym samym czasie, oba w trybie śledzenia zmian,
- `ng build` - jednorazowe uruchomienie kompilatora `tsc`,
- `ng build --watch` - kompilator w trybie śledzenia zmian,
- `ng test` - kompilacja i uruchomienie testów jednostkowych, np. dla `AppComponent`,
- `ng lint` - linting dla naszej aplikacji,
- `ng e2e` - uruchomienie testów kompleksowych naszej aplikacji - testy end-to-end.

tsconfig.json - ustawienia kompilatora tsc

tsconfig.json - ustawienia kompilatora tsc

```
{
  "compileOnSave": false,
  "compilerOptions": {
    "baseUrl": "./",
    "outDir": "./dist/out-tsc",
    "sourceMap": true,
    "declaration": false,
    "module": "es2015",
    "moduleResolution": "node",
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "importHelpers": true,
    "target": "es5",
    "typeRoots": [
      "node_modules/@types"
    ],
    "lib": [ "es2018", "dom" ]
  }
}
```

webpack



`https://webpack.js.org/`

- narzędzie do wczytywania rożnorodnych zależności naszej aplikacji, m.in. poszczególnych modułów angulara,
- alternatywą może być np. `SystemJS` (`https://github.com/systemjs/systemjs`).

webpack - wczytywanie modułów

Co to jest webpack?

- Narzędzie, które czyta wskazany przez nas plik JS, a następnie wykonuje na nim zadane czynności.
- Jeżeli znajdzie w kodzie importowanie innego modułu (innego pliku JS), dołączy go do naszego kodu.
- Webpack może łączyć skrypty, minimalizować je, może zamieniać nasze SCSS na odpowiedni kod JS (takie style można potem używać w js).
- Może optymalizować kod html, dołączać do js zakodowane grafiki itp.
- W wyniku takich akcji nasze źródłowe skrypty są zamieniane na jeden wynikowy, zminimalizowany i zoptymalizowany plik.
- Słynie z dość trudnej konfiguracji.

Analiza kodu - samodzielny komponent

src/app/app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'my-new-app';
}
```

- AppComponent będzie korzeniem całej aplikacji.
- Każda aplikacja ma przynajmniej jeden taki komponent.
  Zwykle nazywa się go AppComponent.
- Komponenty są podstawowymi blokami konstrukcyjnymi aplikacji Angular.
- Kontrolują jakiś obszar ekranu - widok - poprzez związany z nimi szablon.

Struktura komponentu

Struktura komponentu:

src/app/app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'my-new-app';
}
```

- jedno lub więcej poleceń `import` do tego co potrzebujemy,
- dekorator `@Component` - mówi aplikacji Angular, jakiego szablonu użyć i jak utworzyć komponent,
- klasa komponentu, kontroluje wygląd i zachowanie widoku poprzez jego szablon.

Importowanie

- Aplikacje Angular mają modułową budowę.
- Składają się z wielu plików, z których każdy ma określone przeznaczenie.
- Angular sam w sobie jest zbudowany z modułów. Jest kolekcją modułów, każdy składający się z kilku powiązanych funkcji.
- Kiedy potrzebujemy czegoś z bibliotek - importujemy to:

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

import z podstawowej biblioteki Angular, dostęp do wzorca dekorator.

Dekorator @Component

src/app/app.component.ts

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

- `Component` to funkcja dekoratora, pobiera jeden argument - objekt JS zawierający metadane.
- Wywołujemy tę funkcje na obiekcie klasy poprzedzając ja symbolem `@`.
- `@Component` to dekorator - pozwala związać metadane z klasą komponentu.
- Metadane mówią aplikacji Angular jak utworzyć i używać komponentu.
- `selector` określa prosty selektor CSS dla elementu HTML - wszędzie gdzie wystąpi znacznik `app-root` będzie instancja `AppComponent`.
- `templateUrl` - ścieżka do pliku szabon dla widoku - renderowany w czasie tworzenia widoku;
- Może zawierać zmienne, właściwości komponentu, może się odwoływać do innych komponentów tworząc **drzewo komponentów**.

Klasa komponentu

src/app/app.component.ts

```
export class AppComponent {
  title = 'my-new-app';
}
```

- Tutaj klasa komponentu jest prawie pusta, nic nie robi.
- Później można ją rozwinąć o właściwości i jakąś logikę.
- Eksportujemy klasę na zewnątrz (**export**) - można ja zaimportowć gdziekowiek w aplikacji (**import**).

Główny moduł naszej aplikacji

src/app/app.module.ts

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

import { AppComponent } from './app.component';

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

- Utworzony komponent importujemy w głównym module aplikacji.
- Deklarujemy go i uruchamiamy.
- Na koniec z pliku eksportujemy cały moduł.

Plik wejściowy aplikacji

src/main.ts

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

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

if (environment.production) { enableProdMode(); }

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

- Trzeba gdzieś wczytać utworzony moduł.
- Importujemy dwie rzeczy:
  - mechanizm uruchamiania aplikacji w przeglądarce (`platformBrowserDynamic`),
  - utworzony przez nas moduł `AppModule` z odpowiedniego pliku,
- odpalamy aplikację (`bootstrapModule()`) z zaimportowanym modułem.
- Funkcja uruchamiająca aplikację zależy od platformy, nie jest więc w `@angular/core`.
- Aplikację można załadować na przykład na tablecie z NativeScript.
- Podział na moduł i plik go ładujący jest wzorcowy, zapewnia porządek.

Plik index.html

src/index.html

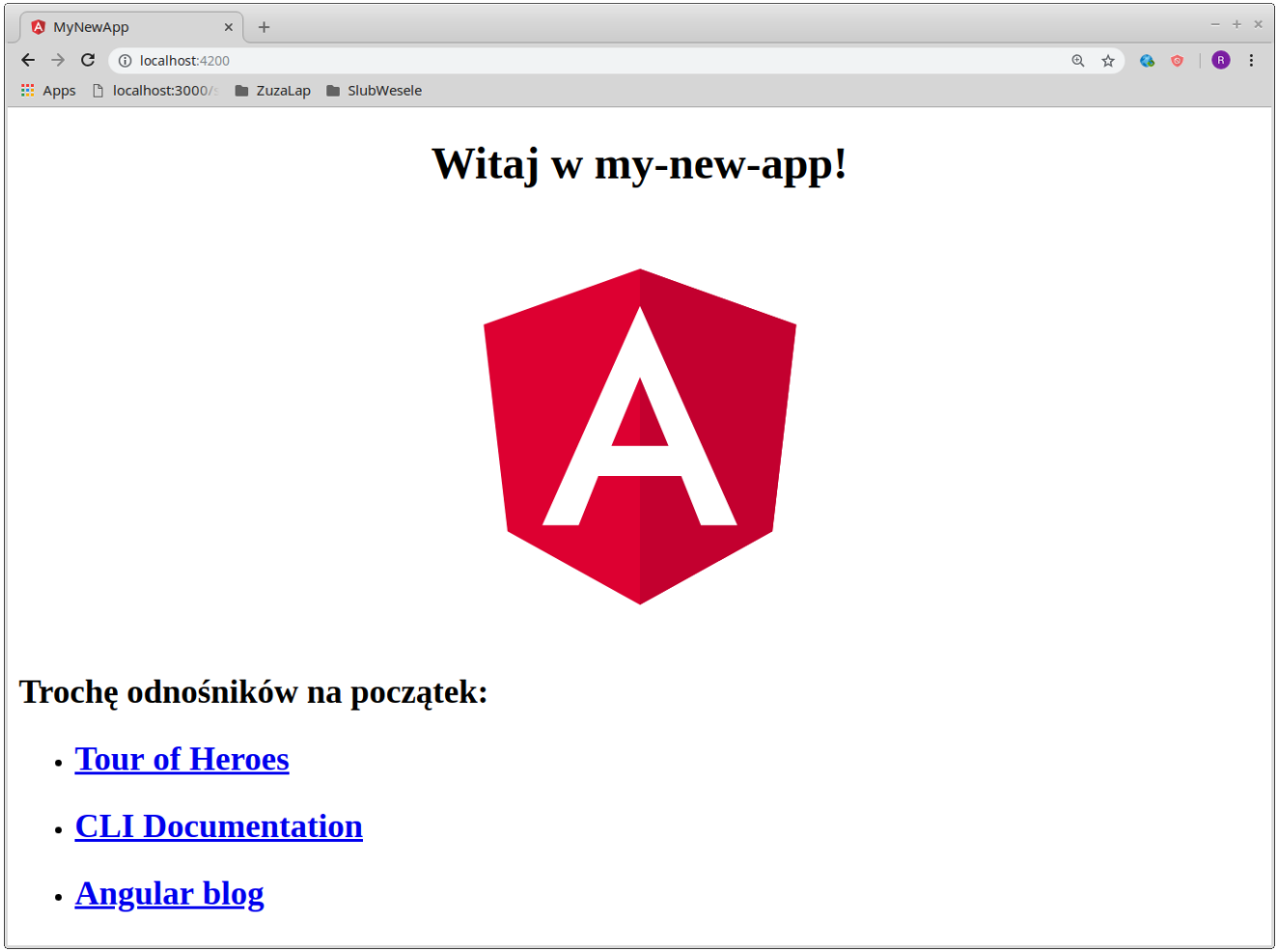
```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>MyNewApp</title>
  <base href="/">

  <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>
```

Zbudowanie i uruchomienie aplikacji

Kompilujemy i uruchamiamy aplikację:

```
ng serve
```



Aplikacja działająca na serwerze

Najprostszy sposób na wdrożenie (działająca wersja deweloperska):

- kompilujemy wersję deweloperską:

  ```
  ng build
  ```

- kopiujemy wszystko z katalogu wynikowego (domyślnie `dist/`) do głównego katalogu serwera
- jeśli aplikacja jest na serwerze w podfolderze, to przy kompilacji ustawiamy odpowiednią ścieżkę bazową; opcja `--base-href` ustawia odpowiednio `<base href>` w pliku `index.html`
- jeśli plik `index.html` jest w `/my-app/prod/index.html` to ścieżkę w `<base href="/my-app/prod/">` ustawimy poleceniem:

  ```
  ng build --base-href=/my-app/prod/
  ```

- powinniśmy skonfigurować serwer, żeby przekierowywał błędne adresy do strony głównej (pliku `index.html`)

Wersja deweloperska nie jest optymalizowana pod szybkie działanie, jest raczej do dzielenia się kodem z innymi programistami.

Wersja produkcyjna I

Kompilacja do wersji produkcyjnej (parametr `--prod`):

```
ng build --prod
```

Flaga `prod` uruchamia następującą optymalizację:

- kompilacja z wyprzedzeniem (Ahead-of-Time, AOT): następuje prekompilacja szablonów komponentów
- włączenie trybu produkcyjnego (flaga `--environment=prod`) wyłącza wiele elementów potrzebnych tylko przy tworzeniu aplikacji, np. dwuzmianowe cykle detekcji zmian (ang. *dual change detection cycles*)
- grupowanie, następue łączenie wielu plików aplikacji i bibliotek w kilka pakietów
- minifikacja (doc. uglification): zastępowanie nazw zmiennych i funkcji maksymalnie krótkimi nazwami
- eliminacja martwego kodu: automatyczne usuwanie dużej ilości nieużywanego kodu i modułów, do których nic się nie odnosi

Wersja produkcyjna II

Dodatkowa optymalizacja (parametr `--build-optimizer`):

```
ng build --prod --build-optimizer
```

zmniejsza dodatkowo wielkość tworzonych pakietów.

Kompilacja: `ng build`

```
my-app
├── 3rdpartylicenses.txt
├── favicon.ico
├── index.html
├── main.js
├── main.js.map
├── polyfills.js
├── polyfills.js.map
├── runtime.js
├── runtime.js.map
├── styles.js
├── styles.js.map
├── vendor.js
└── vendor.js.map
```

Wielkość: 7,2 MB

Kompilacja: `ng build --prod`

```
my-app-prod
├── 3rdpartylicenses.txt
├── favicon.ico
├── index.html
├── main.7996be6042e01978336e.js
├── polyfills.2903ad11212d7d797800.js
├── runtime.6afe30102d8fe7337431.js
└── styles.34c57ab7888ec1573f9c.css
```

Wielkość: 249,8 kB

Konfiguracja serwera I - zwracanie pliku `index.html`

- Aplikacje Angulara świetnie sprawują się na prostym, statycznym serwerze plików HTML
- Nie potrzeba silnika do obsługi aplikacji po stronie serwera - Angular zrobi to po stronie klienta
- Jeśli aplikacja używa trasowania z Angulara, to należy skonfigurować aplikację aby zwracała plik `index.html` w przypadku odwołania do innego zasobu w adresie URL
- Aplikacja powinna obsługiwać głębokie linki, np. `http://www.mojastrona.pl/bohater/31` - głęboki link do szczegółów bohatera o id 31
- Nie ma problemu jeśli działamy z wnętrza aplikacji - Angular przechwyci ten adres
- Kliknięcie odnośnika w wiadomości email, wprowadzenie go do paska adresu czy proste odświeżenie okna powoduje obsługę adresu z bezpośrednio z przeglądarki pomijają aplikację i jej mechanizm trasowania
- Serwer zwraca `index.html` dla adresu `http://www.mojastrona.pl/` ale adres `http://www.mojastrona.pl/bohater/31` zwróci błąd `404 - Not found`
- W przypadku głębokich linków serwer powinien zwracać plik `index.html`

Konfiguracja serwera II - zwracanie pliku `index.html`

Serwery deweloperskie:

- Lite-server: domyślny serwer deweloperski dla repozytorium Quickstart (`https://github.com/angular/quickstart`), skonfigurowany do przekierowań do `index.html`
- serwer deweloperski Webpack: wymaga odpowiedniego ustawienia wpisu `historyApiFallback`

Serwery produkcyjne:

- Apache: należy dodać regułę `rewrite` do `.htaccess`:

```
RewriteEngine On
    # If an existing asset or directory is requested go to it as it is
    RewriteCond %{DOCUMENT_ROOT}%{REQUEST_URI} -f [OR]
    RewriteCond %{DOCUMENT_ROOT}%{REQUEST_URI} -d
    RewriteRule ^ - [L]
    # If the requested resource doesn't exist, use index.html
RewriteRule ^ /index.html
```

- NGinx: używamy dyrektywy `try_files`
- IIS: dodajemy odpowiednią regułę rewrite do pliku `web.config`

Wydania i wersjonowanie frameworka Angular

Wersjonowanie Angulara: `major.minor.patch`, np. dla wersji `7.2.11` mamy:

- 7 to wersja główna,
- 2 to drugorzędny numer wersji,
- 11 to kolejny numer wydanych łatek.

Wydawanie kolejnych wersji:

- co 6 m-cy jest kolejne główne wydanie, które może zawierać zmiany wymagające ingerecji w kod (wydania w maju i listopadzie),
- w czasie tych 6 m-cy jest od 1 do 3 mniejszych wydań,
- różne łaty z błędami wydawane są niemal w każdym tygodniu.

- Do każdego głównego wydania jest aktywne wsparcie przez 6 m-cy (regularne aktualizacje i poprawki błędów).
- Później przez kolejne 12 m-cy dla danej wersji jest wsparcie LTS (tylko krytycze poprawki i poprawki błędów bezpieczeństwa).

Wydania i wersjonowanie frameworka Angular

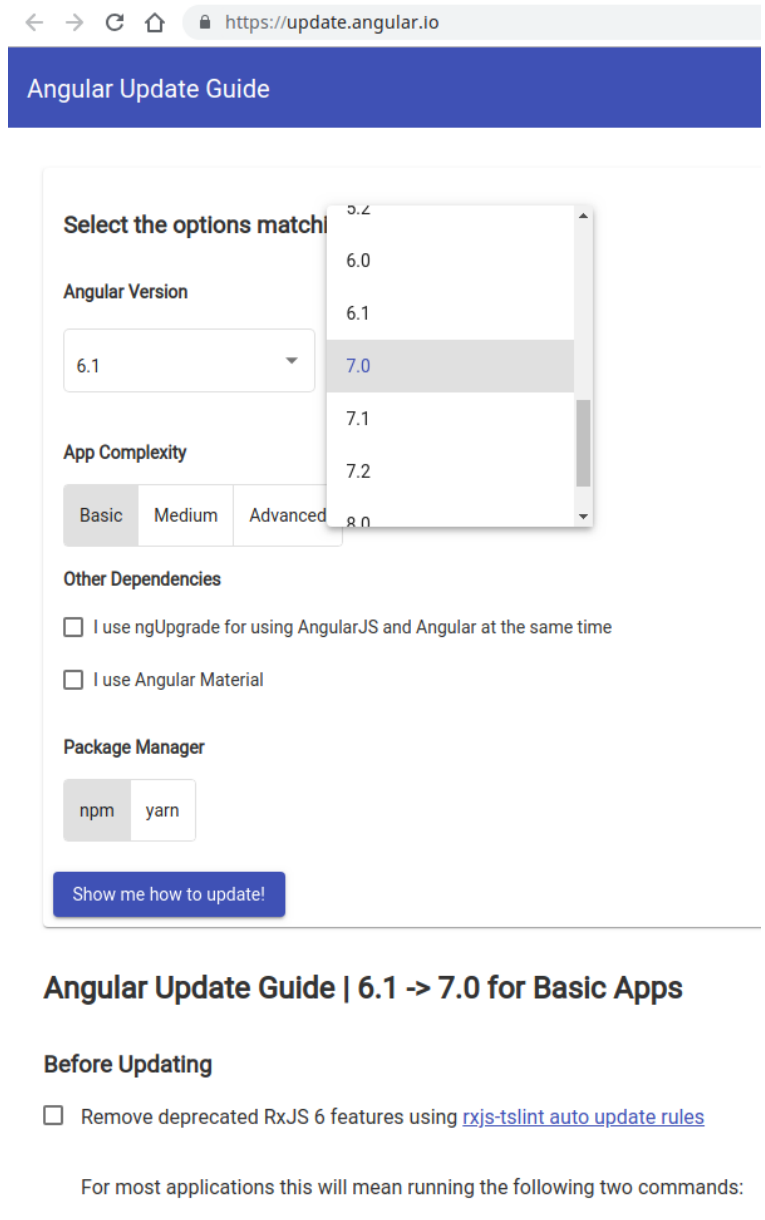
Kolejne wersja Angulara:

- 8.0.0 - maj 2019
- 7.0.0 - 18 października 2018
- 6.0.0 - 3 maja 2018
- 5.0.0 - 1 listopada 2017
- 4.0.0 - 23 marca 2017
- 2.0.0 - 14 września 2016

- Są wydawane wersje beta i wydania kandydackie (ang. release candidates).
- Stare, nieaktualne funkcje są porzucane na korzyść nowych.
- Każda stara, poszucona funkcjonalność jest wspierana jeszcze przez 12 m-cy po jej porzuceniu.

Aktualizacja aplikacji napisanej w Angular

- Aktualizację z jednej głównej wersji do następnej robimy bez większych problemów korzystając ze specjalnego przewodnika: `update.angular.io`.
- Aktualizację z jednej głównej wersji innej, oddalonej od niej o więcej niż jeden numer, np. z wersji 5.x.x do 7.x.x **robimy iteracyjnie**, najpierw do najowszej wersji 6.x.x, a dopiero później do 7.x.x.
- Aktualizacja z poprzedniej wersji frameworka 1.x.x jest już sprawą dużo bardziej złożoną.



Źródła

- https://angular.io/
- https://pl.wikipedia.org/wiki/TypeScript
- https://www.typescriptlang.org/
- http://codeguru.geekclub.pl/baza-wiedzy/wprowadzenie-do-programowania-w-typescript-wstep,3575
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions