APLIKACJE WEB SPA Z FRAMEWORKIEM

Angular

Michał Jabłoński

console.log('Kilka słów o mnie');

Michał Jabłoński

Front – End deweloper od 2007 r.

- 2007 (X)HTML + CSS + JavaScript
- 2009 Flash + HTML + AS 2.0 + PHP
- …
- 2011 Flex + AS 3.0 + Java EE + GWT + JavaScript
- …
- 2017 Node.js + ES-next + TypeScript + RxJS + Angular + React

https://github.com/michaljabi

Plan

- Czym jest SPA ?
- Środowisko – Node.js
- JavaScript a TypeScript
- Co to jest Transpilacja?
- Co to są Polyfill'e?
- Co to jest Bundling?
- Angular - start

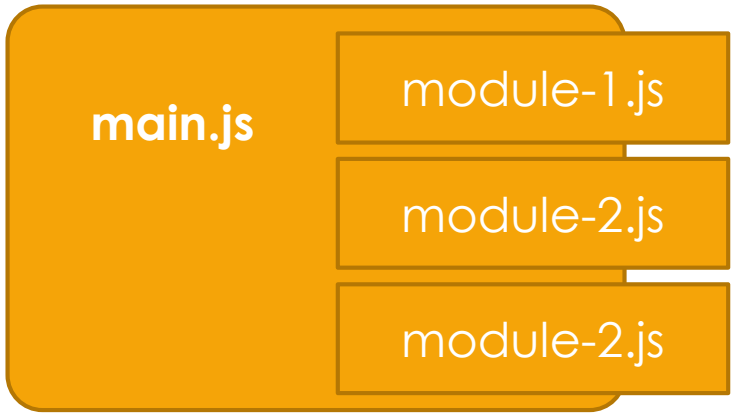
JavaScript – trochę się pozmieniało

- JS powstaje z myślą o dynamicznych modyfikacjach DOM
  - Ilość kodu potrzebna do napisania i sposób jego utrzymywania w stosunku do uzyskiwanych „dynamicznych" interakcji na stronie – skutecznie zniechęca do pisania czegoś więcej jak tylko wariatory do formularzy
- Później: pojawia się AJAX – można dynamicznie przeładowywać cześć strony
- Później: nastaje era jQuery: usystematyzowane i uproszczone API do zarządzania DOM i AJAX
- W 2009 powstaje Node.js (Server-Side JavaScript)
- Po 2015: JavaScript dostaje dużo lukru składniowego i od tej pory jest odświeżany co roku.
  - Dzięki połączeniu środowiska developerskiego w Node.js i składni z 2015 dostajemy język w którym można pisać aplikacje w sposób podobny do np. Java czy C#

Co się stało z JavaScript?

Kontekst historyczny

- W 2009 pojawia się „Node.js” – który zmienia sposób pisania kodu



- Po 2015 pojawiają się „cukiereczki” tzw. Sintactic sugar / Lukier składniowy

class | Set() | Map() | …spread | `interpolacja ${x}`

Bardziej jak…
C#
JAVA

Nowe możliwości: ES6, ES7, ES8, ES-next…

- Modułowość
- Leksykalne deklarowanie zmiennych: let, const
- Funkcje Arrow
- Domyślne wartości dla argumentów funkcji
- Składniowy lukier dla klas
- Destrukturyzacja obiektów i tablic
- Operatory: ...spread i ...rest
- Interpolacja tekstu
- Generatory
- Operatory: async / await
- I inne...

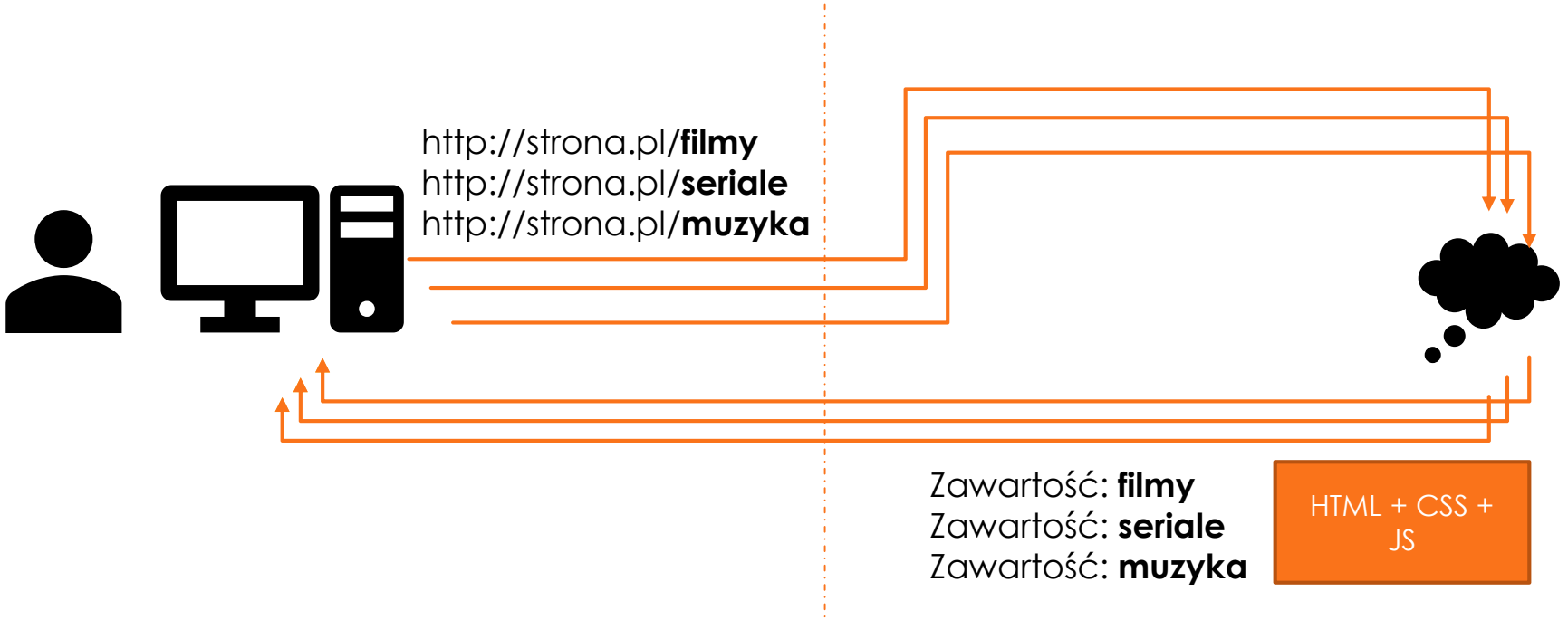
Istotne koncepcje języka nie ulegają zmianie

- Prototypowalność
- Funkcje jako First Class Citizens
- Zasięg zmiennych
- Kontekst wywołania
- Obiektowa natura JavaScriptu

SPA = Single Page Application

- Przeglądarka „nie gubi stanu” – nie przeładowujemy strony i podstron
- Większość „AKCJI” dzieje się po stronie przeglądarki Klienta

- DAWNIEJ:



SPA

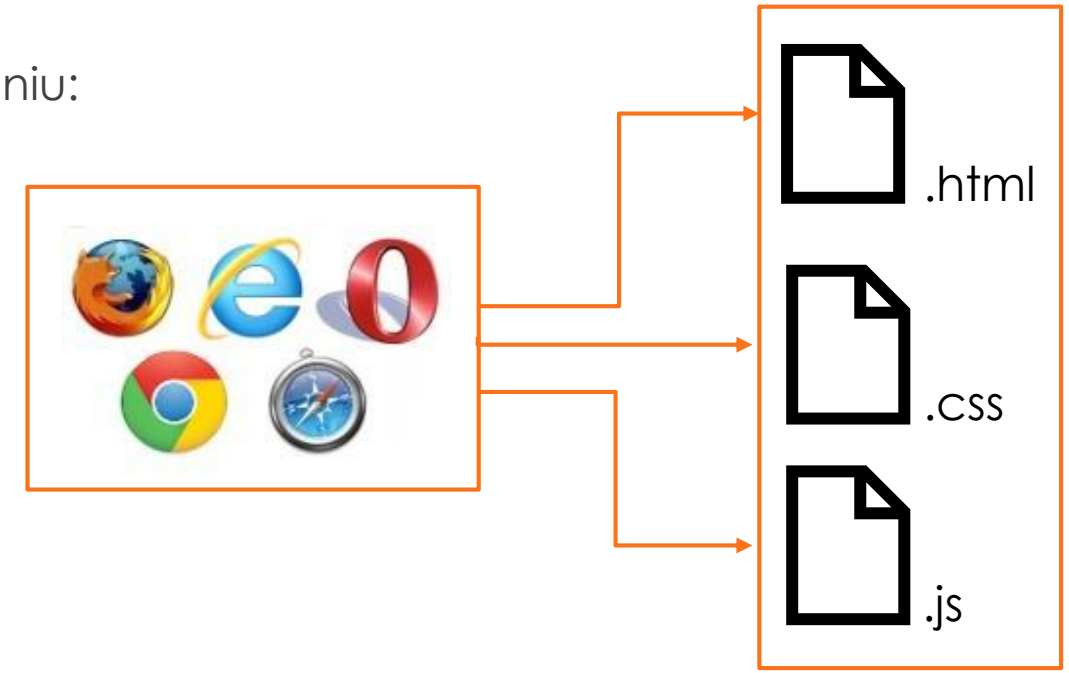
- Idea za Single Page Application
- Wewnętrzny router pilnuje „co pokazać na stronie”, zmienia pasek adresu w zależności od zawartości.

- AKTUALNIE

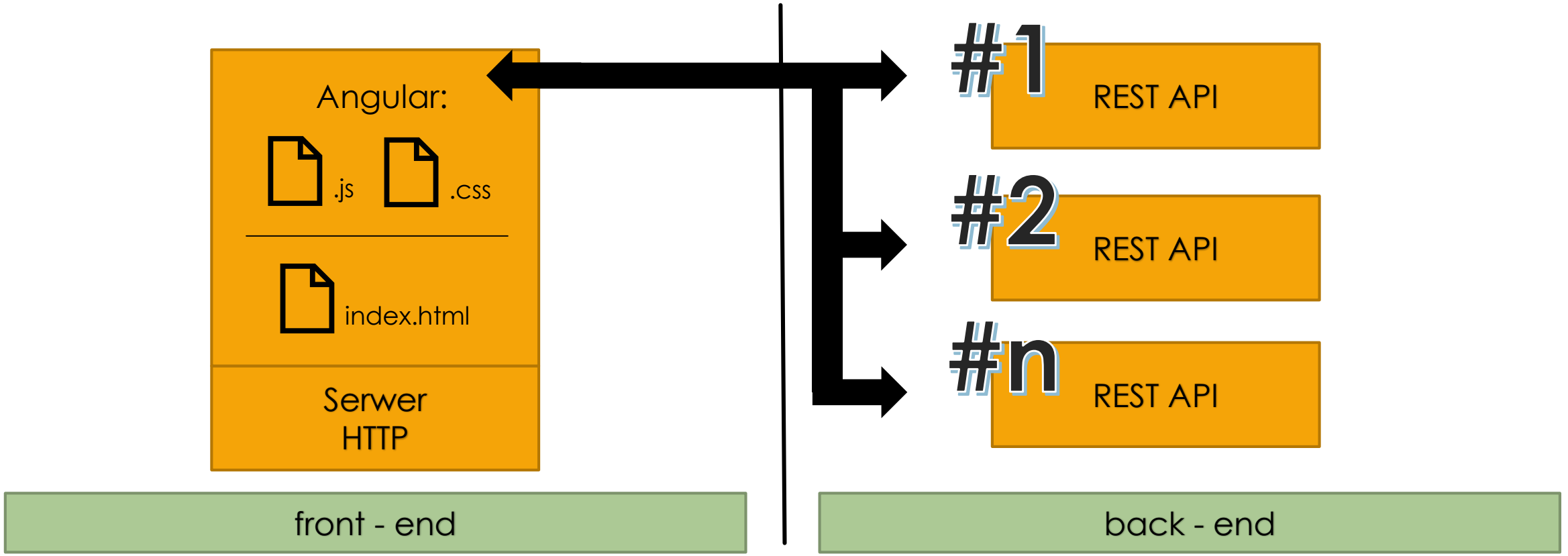
(w Frameworkach opartych o JavaScript)

Wszystko sprowadza się do podstaw

- HTML + CSS + JS
- W uproszczeniu:



W którym miejscu potrzebujemy Angulara?



Zdalne sterowanie przez JavaScript

- Frameworki JS
- Naturalny bieg i rozwój koncepcji AJAX (Asynchronous JavaScript and XML)
- JavaScript steruje DOM'em, dynamicznie „podmieniając" treści na stronie,
- Kontroluje odniesieniami „Location" dla przeglądarki
- Wysyła zapytania HTTP po dynamicznie ładowane dane

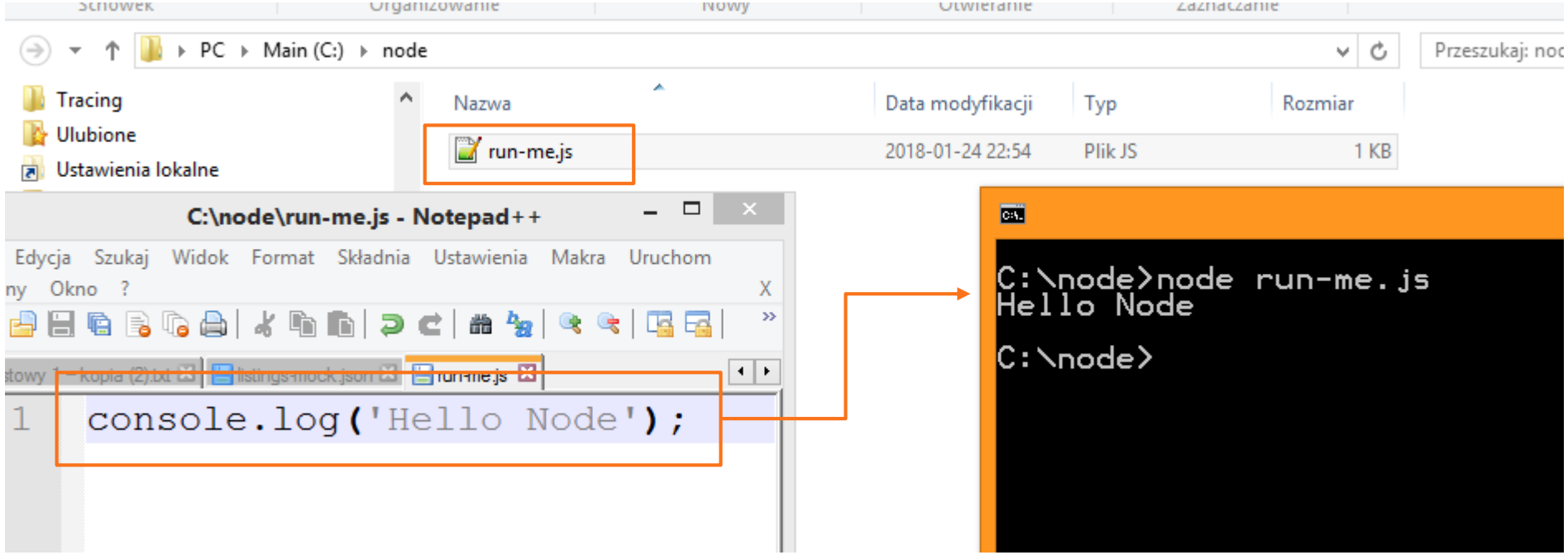
I z tego właśnie powodu…

- … mamy „Eksplozję ilości framework'ów JavaScript"

Środowisko – Node.js

- GAME – CHANGER !
- Idea: commonJS + rozwój modularyzacji JavaScriptu
- JavaScript runtime | Server Side JavaScript

https://nodejs.org/en/



Node Package Manager

- domyślny manager pakietów dla środowiska Node.js
- https://www.npmjs.com/

- Można instalować zależności „dependencies” | bibilioteki JavaScript
  - Przykłady:
  - webpack, angular, reactjs, jquery, bootstrap….
  - przykładowe wywołanie:

```
> npm install webpack -D
```

JavaScript vs TypeScript

zwolennicy

najpierw spójrzmy na

JAVA SCRIPT !

przeciwnicy

```
var helloWorld = 'Hello';

function mySuperMethod () {
  return helloWorld;
}

mySuperMethod() //?  Hello
```

Ha ha – wolność !

```
var helloWorld = 'Hello';

function mySuperMethod () {
  helloWorld = 1234;
  return helloWorld;
}

mySuperMethod() //?  1234
```

aaagh –
gdzie są moje typy !!

TypeScript czyli superset JavaScript'u

```typescript
let helloWorld = 'Hello';

function mySuperMethod(say: string): string {

    return say;
}

// mySuperMethod(helloWorld) --> 'Hello'
```

- **Podstawowe zalety:**
  - Użycie typów
  - Obsługa klas i interface'ów
  - Szybka transpilacja do JavaScriptu
  - Pilnowanie poprawności kodu

- **Używasz już dzisiaj cech języka „z przyszłości"**

http://www.typescriptlang.org/

TypeScript

Rozwój JavaScriptu

- Dobrze opisane: https://**en**.wikipedia.org/wiki/ECMAScript

- Nowa składnia: **ES6**
  - transpilery pozwalające używać tej składni już dzisiaj

- Syntactic sugar do wielu rzeczy
- Nowe metody dla natywnego kodu
- Typescript jest już zgodny z ES6, 7, 8, 9, 10

Co to jest Transpilacja?

- OK, korzystamy z super nowości po 2015, język jest „odświeżony" są nowości.
- A co z kompatybilnością wstecz ?
- A co jeśli ja lubię korzystać z IE9 ?
- Poza tym: przeglądarka nie zrozumie przecież TypeScriptu!

- Zgadza się, potrzebujemy dodatkowego kroku: Transpilacji. Zachowania poziomu abstrakcji języka JS ale w składni np. z przed 2015r.
- Tak samo musimy transpilować program napisany w TypeScript.

Co to są Polyfill'e?

Sama transpilacja – to czasem za mało…

- W JavaScript rozwija się nie tylko syntax ale również samo natywne API
- Dzięki temu np. Posiadamy nowe funkcje dla natywnego Array (np. findIndex()) lub String (.padStart() , .padEnd())
- Żeby te elementy zostały „nadbudowane" w przeglądarkach w których nie istnieją potrzebujemy tzw.

POLYFILLS

- Popularną oferującą je biblioteką jest **core-js**

https://www.npmjs.com/package/core-js

Przykład polyfill'a:

```
// https://github.com/uxitten/polyfill/blob/master/string.polyfill.js
// https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/padStart
if (!String.prototype.padStart) {
    String.prototype.padStart = function padStart(targetLength, padString) {
        targetLength = targetLength >> 0; //truncate if number, or convert non-number to 0;
        padString = String(typeof padString !== 'undefined' ? padString : ' ,);
        if (this.length >= targetLength) {
            return String(this);
        } else {
            targetLength = targetLength - this.length;
            if (targetLength > padString.length) {
                padString += padString.repeat(targetLength / padString.length);
                //append to original to ensure we are longer than needed
            }
            return padString.slice(0, targetLength) + String(this);
        }
    };
}
```

Sprawdzamy czy w prototypie String jest metoda .padStart – jeśli nie, dopisujemy ją.

Co to jest bundling?

- Ok, uzycie Node.js + TypeScript rozwiązuje nam kilka kwestii i sprawia iż pisanie kodu i jego utrzymanie staje się przyjemne

- Jednak dalej jesteśmy na „Server Side"! Nie ma tutaj DOM, nie ma BOM nie ma „document" nie ma „browser" – co dalej ?

- Z pomocą przychodzi proces bundlingu. Czyli wykorzystanie dodatkowego narzędzia, do „zebrania wszystkich naszych modułów CommonJS w jeden plik .js i wpleceniu tego w stronę HTML"

Webpack – the bundler

DEVELOPMENT

- src
  - app
    - app.component.css
    - app.component.html
    - app.component.spec.ts
    - app.component.ts
    - app.module.ts
    - app-routing.module.ts
  - assets
  - environments
    - environment.prod.ts
    - environment.ts
  - favicon.ico
  - index.html
  - main.ts
  - polyfills.ts
  - styles.css
  - test.ts
- .editorconfig
- .gitignore
- angular.json
- browserslist
- karma.conf.js
- package.json
- package-lock.json
- README.md
- tsconfig.app.json
- tsconfig.json
- tsconfig.spec.json
- tslint.json

BUILD

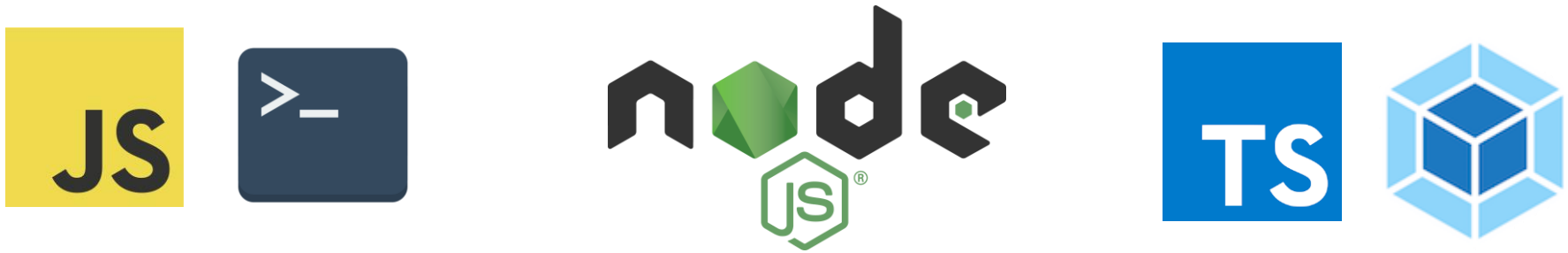
PRODUCTION

- dist
  - sample-app
    - 3rdpartylicenses.txt
    - favicon.ico
    - index.html
    - main-es5.c26bc766533fa66937f8.js
    - main-es2015.c26bc766533fa66937f8.js
    - polyfills-es5.6696c533341b95a3d617.js
    - polyfills-es2015.2987770fde9daa1d8a2e.js
    - runtime-es5.edb2fcf2778e7bf1d426.js
    - runtime-es2015.edb2fcf2778e7bf1d426.js
    - styles.3ff695c00d717f2d2a11.css

https://webpack.js.org

Podsumowanie

- Jaka wiedza przyda nam się przy budowaniu aplikacji SPA z Angular ?

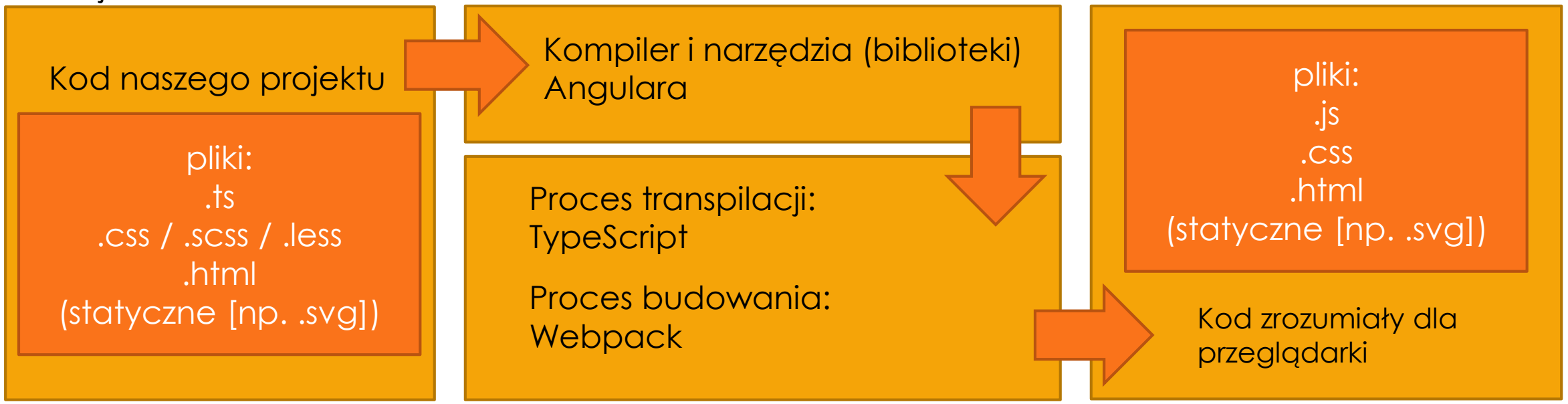


Angular - starter

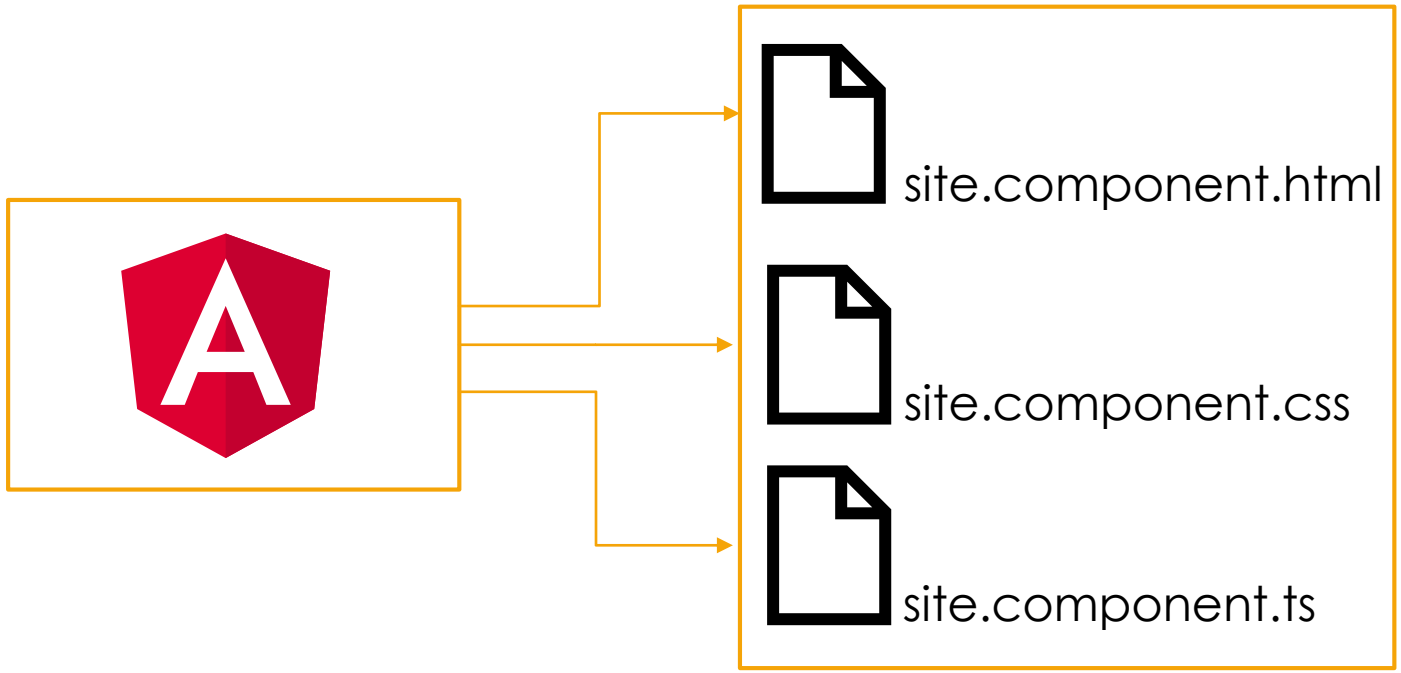
https://angular.io/

Technicznie:

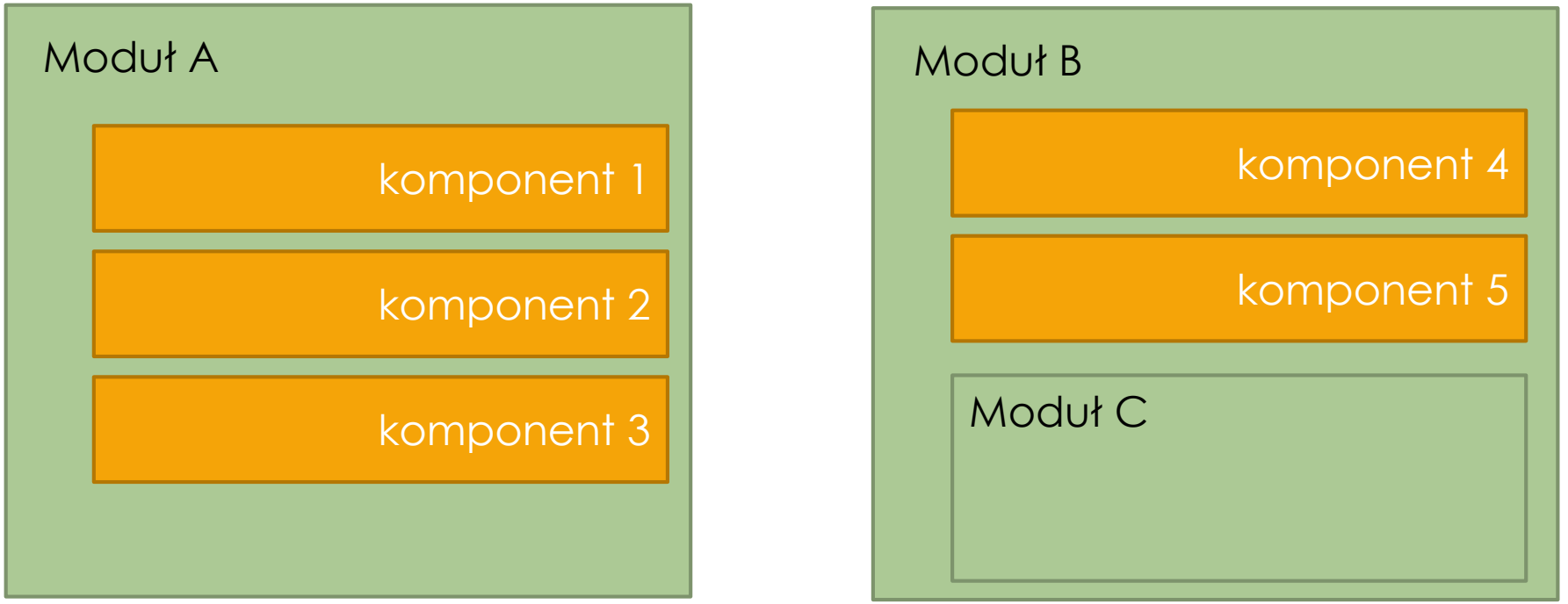
Node.js



Komponenty to podstawa !



Kontenerami na komponenty są **Moduły**



Moduły

- Zbierają wszystkie informacje o:
  - Innych używanych pod-modułach [**IMPORTS**]
  - Używanych Komponentach, Dyrektywach i Pipe'ach [**DECLARATIONS**]
  - Serwisach [**PROVIDERS**]
- Są „kluczem" do uzyskania skalowalności projektu !
- Powinny być autonomiczne
  - można „wypiąć" i „wstawić" w inne miejsce – bez większego bólu

Angular CLI

- Angular Command Line Interface
  - https://cli.angular.io/
  - projekt wzorowany na „ember-cli" podobna idea „Scaffoldera"
  - GLOBALNIE zainstalowany **KOMBAJN do tworzenia aplikacji ANGULAR**
  - Działa dzięki Node.js i NPM
  - Uruchamiany z linii poleceń
  - Komenda: **ng**

```
> ng new PROJECT-NAME
> cd PROJECT-NAME
> ng serve
```

Angular Styleguide

- Kultura pisania kodziku w Angularze
  - https://angular.io/guide/styleguide

- Przyjęte normy i konwencje w Angular
- Mają głównie zapobiec wylądowaniu z projektem jako „SPAGETTI CODE”
- Napisane w bardzo fajnej formie: **„tego unikaj” – „tak pisz”**

Porównanie znaczących frameworków JavaScript

	Angular *	**React**	**Vue**	**Ember ***
schemat i reguły budowania aplikacji	Ścisłe i odgórnie narzucone. Framework dostarcza wszystkich potrzebnych narzędzi.	Realizacja główne warstwy widoku. Pozostałe biblioteki – wg. Uznania.	Realizacja warstwy widoku (komponenty)	Określone i jasno zdefiniowane reguły budowania aplikacji
sposób budowania aplikacji	komponenty zgrupowane w moduły	komponenty widoku	komponenty widoku	komponenty i koncepcja MVC
przepływ danych	one-way, two-way data binding, RxJS (Observables) dla serwisów i operacji asynchronicznych	one way data flow	one-way lub two-way	data down – actions up.
Architektura Flux (unidirectional data flow)	Możliwa do osiągnięcia dzięki RxJS (ngrx/store)	Biblioteki: Redux, Mobx, Flux	Vuex	-
Specyficzne rozwiązania	Dependency Injection i używanie Dyrektyw mechanizm „change-detection" z zone.js	Własny język template: JSX używany bezpośrednio w plikach .js	Single File Components	Dependency Injection,

* z technicznego punktu widzenia tylko Angular i Ember można porównywać jako „równy z równym" ponieważ Vue i React zajmują się tylko warstwą widoku.