GDG Rzeszów

Jakub Nietrzeba

Łagodne wprowadzenie do Angular2

GDG Rzeszów #1

8 Marca 2016

AngularJS

- Rozdzielenie kodu od drzewa DOM
- Projektowanie aplikacji wywrócone do góry nogami (HTML first)
- Automatyczne wiązanie (binding) zmiennych JavaScript z fragmentami strony
- Magia!

AngularJS

- Duże aplikacje po stronie klienta
- Skupione na pobieraniu i pokazywaniu danych
- Formularze, tabele, pola
- Rozdzielenie widoku od kodu serwera (API)
- Rozłączne wersjonowanie

AngularJS 1.x → 2.x

- JavaScript → TypeScript
- Controller → Component
- Directive → Directive
- Filtr → Pipe
- Widoki i serwisy nadal tu są
- Property tak, attrybuty nie
- Trudniej dobrać się do drzewa DOM
- Shadow DOM!
- Beta, beta, beta, beta, beta, beta…

TypeScript

- Mocno typowany
  - Typy pól, parametrów funkcji i wartości zwracanych
- Interfejsy
- Klasy
  - Lista inicjalizacyjna konstruktora
- Wbudowana modularność
- Kompilowany (odcukrzany) do czystego JS
- Arrow function!
  - (x) => { return Math.sin(x); }

Unboxing

- Zainstalować NodeJS

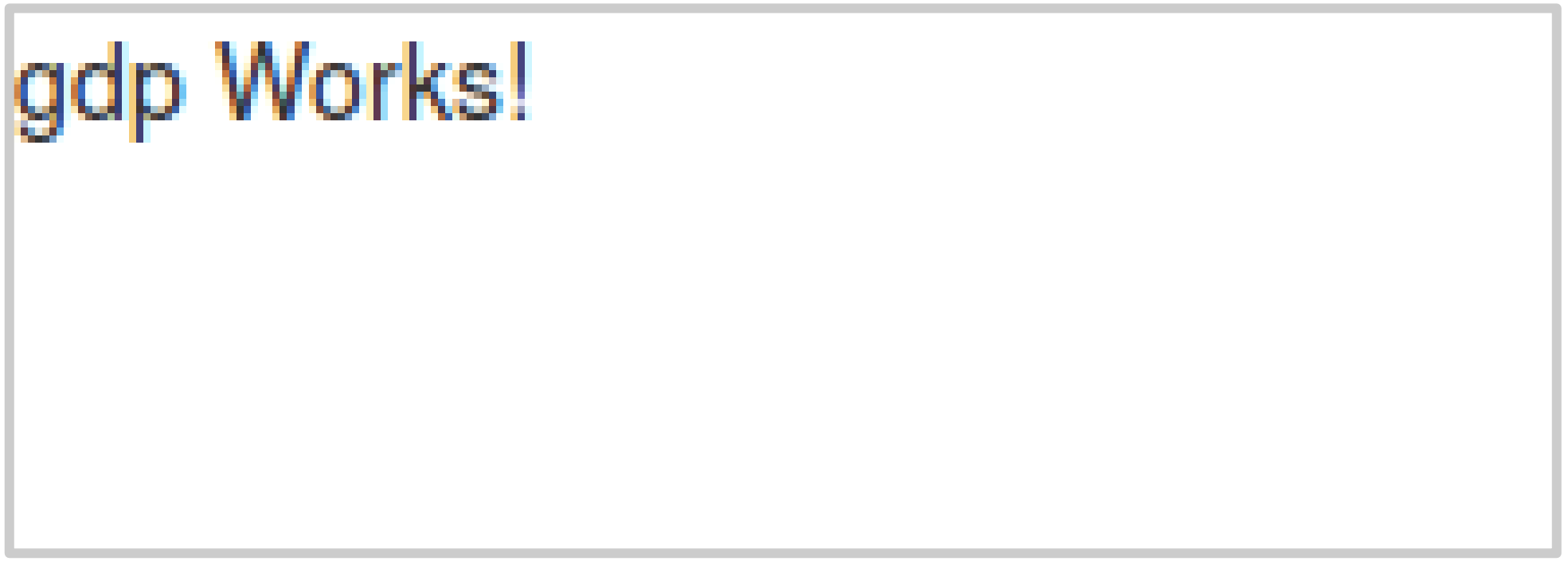
  https://gist.github.com/isaacs/579814#file-node-and-npm-in-30-seconds-sh

- npm install -g npm
- npm install -g angular-cli
- ng init -n {nazwa aplikacji}
- ng serve

  echo fs.inotify.max_user_watches=524288 | sudo tee -a /etc/sysctl.conf && sudo sysctl -p

  npm install --save zone.js

http://localhost:4200/



angular-cli (ng)

- Narzędzie rozwijane przez tych samych ludzi co AngularJS
- Bootstraping aplikacji
- Generator – ng g
- Tester – ng test
- Niszczyciel – ng destroy
- Uruchamiacz – ng serve
- ng help to kilkaset linijek możliwości

ng g route {route}

- Zostanie utworzony katalog modułu (ścieżki)
  - Detale pojedynczego elementu (html, css, ts)
  - Lista elementów (html, css, ts)
  - Komponent (definicja i ścieżki)
  - Serwis na elementy
  - Testy

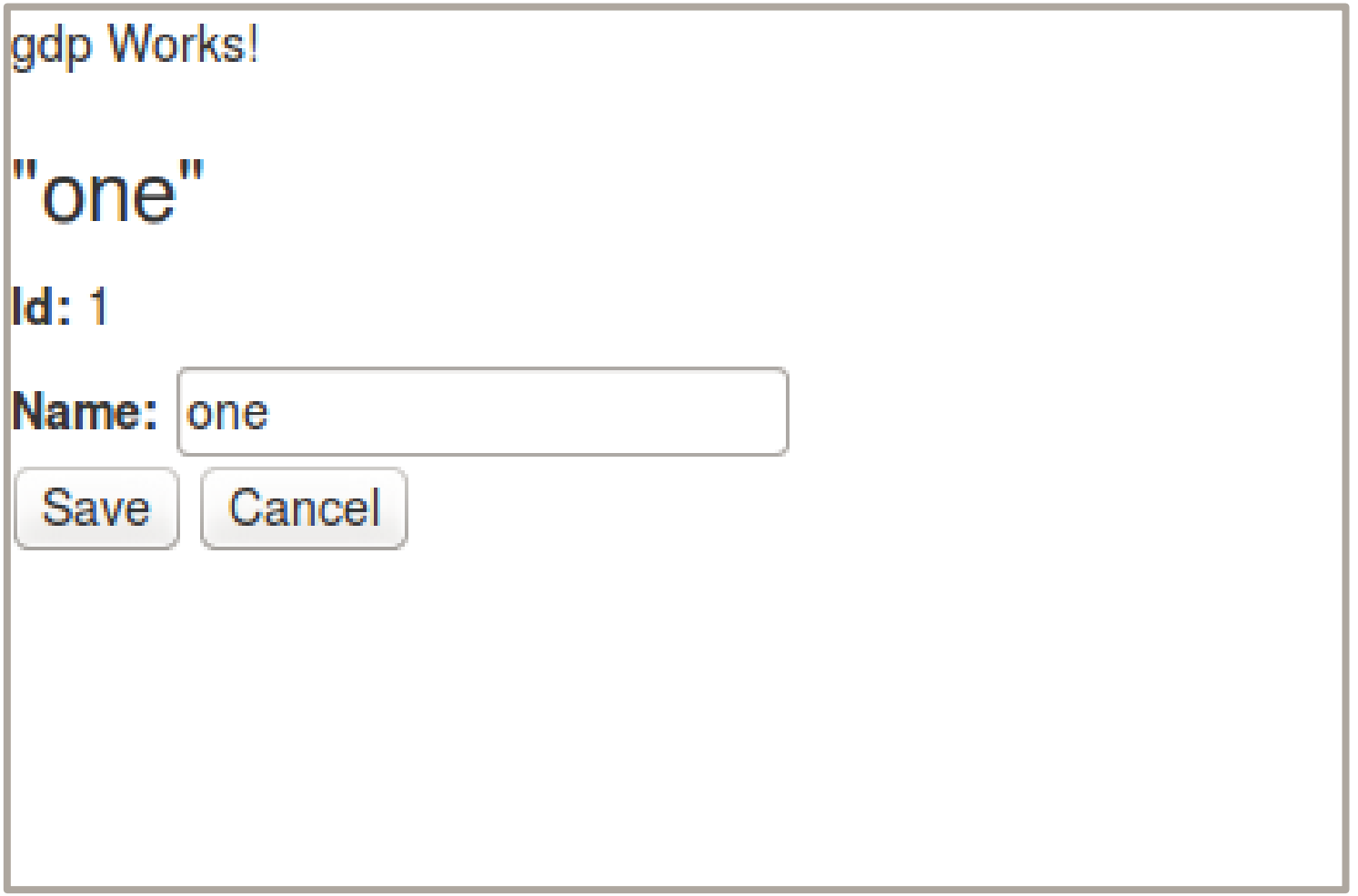
http://localhost:4200/slide

gdp Works!

Slide List

- 1 - one
- 2 - two
- 3 - three

http://localhost:4200/slide/1



slide-root.component.ts

```
import {Component} from 'angular2/core';
import {RouteConfig, RouterOutlet} from 'angular2/router';

import {SlideListComponent} from './slide-list.component';
import {SlideDetailComponent} from './slide-detail.component';
import {SlideService} from './slide.service';

@Component({
  template: '<router-outlet></router-outlet>',
  providers: [SlideService],
  directives: [RouterOutlet]
})
```

slide-root.component.ts

```
@RouteConfig([
  {path: '/', name: 'SlideList', component: SlideListComponent, useAsDefault: true},
  {path: '/:id', name: 'SlideDetail', component: SlideDetailComponent}
])
export class SlideRoot {
  constructor() {}
}
```

Co tu się dzieje?

- Importuje jakieś rzeczy z jądra
- Importuje komponent listy
- Importuje komponent detali (edycja/podgląd)
- Importuje serwis
- <router-outlet></router-outlet>
- Dekoratory, wszędzie dekoratory
  - @Component
  - @RouteConfig
- Pusta klasa komponentu?

slide-list.component.ts

```
import {Component, OnInit} from 'angular2/core';
import {Slide, SlideService} from './slide.service';
import {ROUTER_DIRECTIVES} from 'angular2/router';

@Component({
  templateUrl: 'app/slide/slide-list.component.html',
  styleUrls: ['app/slide/slide-list.component.css'],
  directives: [ROUTER_DIRECTIVES]
})
export class SlideListComponent implements OnInit {
  slides: Slide[]; // here is magic!
  constructor(private _service: SlideService) {}
  ngOnInit() { this._service.getAll().then(slides => this.slides = slides); }
}
```

Co tu się dzieje?

- Importy, importy, importy
  - TypeScript nie posiada class loadera, trzeba ręcznie
  - Jest „odsładzany” (desugared) do JS
- @Component
  - Szablon i styl jako linki
  - Używa wszystkich dyrektyw routera

Co tu się dzieje?

- Eksportowana klasa componentu coś robi!
  - OnInit – jeden ze zdarzeń cyklu życia komponentu (OnDestory, OnChanges…)
  - Domyślnie publiczna zmienna slides
  - Sprytny konstruktor – lista inicjalizacyjna konstruktora (C++ anyone?)
- No dobra, ale gdzie ta całą magia?

slide.service.ts

```
import {Injectable} from 'angular2/core';

export class Slide {
  constructor(public id: number, public name: string) {}
}

@Injectable()
export class SlideService {
...
```

slide.service.ts

```
 getAll() { return promise; }

  get(id: number) { return promise.then(all => all.find(e => e.id === id)); }

}


let mock = [new Slide(1, 'one'), new Slide(2, 'two'), new Slide(3, 'three')];


let promise = Promise.resolve(mock);
```

Co tu się dzieje?

- Dwie eksportowane klasy (Slide, SlideService)
  - W prawdziwej aplikacji każda klasa we własnym pliku
- Klasa serwisu odekorowana
  - Gotowa do wstrzykiwania
- Mockupy dla testów
  - W prawdziwej aplikacji zamiast mockupów byłyby odwołania do API

slide-list.component.html

```
<h2>Slide List</h2>

<ul>
  <li *ngFor="#slide of slides">
    <a
      [routerLink]="['SlideDetail', { id: slide.id }]">
      <strong>{{slide.id}}</strong>
      -
      {{slide.name}}
    </a>
  </li>
</ul>
```

Co tu się dzieje?

- *ngFor
  - Multiplikuje (trudne słowo!) fragment drzewa DOM
  - Łączy fragmenty ze iterowaną zmienną
  - Obserwuje zmiany tej zmiennej
- Binding! Binding?
  - [ ] - property binding (z modelu do widoku)
  - ( ) - action binding (z widoku do modelu/kontrolera)
  - [( )] - w obie strony (formularze!)

Interpolacja

- {{ }}
- Interpretuje i obserwuje wyrażenie i podmienia w DOM
  - {{ 1 + 2 }} → 3
  - {{ 1 + countValues() }} → 4
- Ograniczenie!
  - Wyrażenie jest interpretowane w locie
  - Niemożliwe: new ; = ++ -- += -= i inne…
  - Pipe & Elvis (będzie później)

slide-detail.component.html

```
<div *ngIf="slide">
  <h3>"{{editName}}"</h3>
  <div>
    <label>Id:</label>
    {{slide.id}}
  </div>
  <div>
    <label>Name:</label>
    <input [(ngModel)]="editName" placeholder="name"/>
  </div>
  <button (click)="save()">Save</button>
  <button (click)="cancel()">Cancel</button>
</div>
```

Co tu się dzieje?

- ngIf wyrzuci całą gałąź DOM gdy będzie false
- {{ editName }} wyświetli zmienną z komponentu
- {{ slide.id }} wyświetli zmienną z obiektu
- [(ngModel)]=”editName”
  - Dyrektywa ngModel jest nakarmiona powiązaniem ze zmienną editName z komponentu
  - Wiązanie jest obustronne
- (click)=”save()”
  - Property click (dyrektywa) jest powiązana z funkcją save z komponentu
  - Wiązanie jednostronne

Pipe & Elvis

- Operator .? Wybacza
  - {{ ala.?ma.?kota }} nie zawiedzie gdy kota nie ma
- Pipe
  - {{ „Cośtam” | uppercase }} → COŚTAM
  - Transformery nakładane są na dane przed wyświetleniem
  - Transformery można łączyć w łańcuchy
  - {{ createdAt | date:”Y-m-d” }} - parametry!
  - ng g pipe {nazwa}
  - Dekorator komponentu ma tablice na to

Sztuczki

- npm install -g typings
  - typings search {name}
  - Lodash i wiele innych bibliotek jest opisanych
- Bootstrap (wtop się w internetowy tłum)
  - npm install bootstrap
  - Dodaj do index.html (same style)
  - Inne podejście pod: https://github.com/valor-software/ng2-bootstrap
- ng github-pages:deploy

Źródła

- https://angular.io/docs/ts/latest/
- https://github.com/Microsoft/TypeScript/blob/master/doc/spec.md
- https://www.npmjs.com/package/typings
- https://github.com/angular/angular-cli