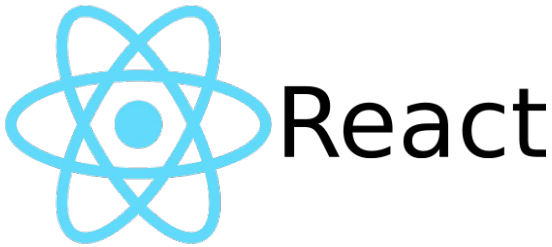
Niezależny frontend z Angular 2

Bielsko-Biała, 17 maja 2016r.

Niezależny frontend z...



Bielsko-Biała, 17 maja 2016r.

O mnie

Tomasz Borowski

Frontend developer w **JCommerce** oraz **Roche**.

Angular, Ember, Ionic, Ruby on Rails.

- Typy frontendowych środowisk developerskich
- Elementy niezależnego środowiska do developmentu aplikacji frontendowej
- Praca z fake’ym API

Frontendowe środowiska developerskie

Non-SPA Web App

Aplikacja nie jest podzielona na część frontendową i backendową. Z reguły skupia się ona wokół jednego frameworka.

Czynniki uzależniające:

- HTML zawiera zmienne pochodzące z technologii backendowej
- Aplikacja nie serwuje danych, tylko wysyła gotowy markup do wyrenderowania w przeglądarce.

SPA osadzone w Web App

Aplikacja składa się z dwóch częśći: frontend app i backend app, które komunikują się poprzez API

Czynniki uzależniające:

- ładowanie bibliotek js poprzez wrappery backendowe
- przygotowywanie dystrybucji przez biblioteki backendowe
- konieczność serwowania aplikacji poprzez serwer backendowy
- konieczność pracy na prawdziwej bazie danych
- testowanie integracyjne przy użyciu backendowych bibliotek

SPA standalone

Aplikacja składa się z dwóch częśći: frontend app i backend app, które komunikują się poprzez API

Czynniki niezależności, czyli do czego backend nam nie jest potrzebny:

- instalacja bibliotek frontendowych poprzez dedykowane JS narzędzia
- serwowanie aplikacji
- projektowanie i praca z fake'owym API
- budowanie dystrybucji aplikacji
- testowanie jednostkowe / integracyjne

Dlaczego warto być niezależnym?

- backend developerzy nie blokują frontend developerów
- pełna kontrola nad implementowaną aplikacją
- praca w szybkim, sprawnym środowisku

Tworzenie niezależnego środowiska frontendowego

Co daje niezależność?

- Korzystanie z managerów pakietów Javascript
- Korzystanie z narzędzi do automatyzacji procesów
- Korzystanie z narzędzi do pakowania modularnego kodu
- Korzystanie z fake'owe API, które możemy w każdym momencie dostosować do naszych potrzeb

Duży wybór rozwiązań

Manager pakietów

Automatyzacja zadań

Pakowanie kodu

Duży wybór rozwiązań

Język markupu

Język skryptowy

Testowanie kodu

Język stylowania

Generowanie metryk kodu

Setup manualny

czasochłonny, musimy wiedzieć czego chcemy

Setup predefiniowany / automatyczny

szybki, skupiony na best practices

We make IT JCommerce

Korzystanie z tzw. boilerplates

Kod, który może być wykorzystywany wielokrotnie w różnych projektach. Najczęściej stanowi podstawę aplikacji, definiując podstawową konfigurację środowiska developerskiego.

Najczęściej dedykowane danemu frameworkowi do tworzenia aplikacji.

+

+

Setup aplikacji z Yeoman: przygotowanie

Instalacja niezbędnych pakietów npm
npm install -g gulp-cli yo generator-karma generator-angular2

Generowanie aplikacji
yo angular2

Bootstrap testów
yo karma

Setup aplikacji z Yeoman: co dostajemy

Serwowania aplikacji
npm start

Testowanie aplikacji
npm test

Przygotowanie dystrybucji
npm build

Setup aplikacji z Yeoman: czego brakuje

Wygenerowana aplikacja nie posiada zdefiniowanego źródła danych.

Jeśli planujemy korzystać z RESTowego API, które udostępni nam backend, to czy wpłynie to na niezależność?

json-server – fake’owe API dla frontendu

json-server: REST

API RESTowe, opierające się o fake'ową bazę danych zapisaną w pliku JSON. Dla bazy zdefiniowanej obok mamy następujące akcje dotyczące postów:

GET	/posts
GET	/posts/1
POST	/posts
PUT	/posts/1
DELETE	/posts/1

```
{
  "posts": [
    { "id": 1, "title": "great!", "author": "john" }
  ],
  "comments": [
    { "id": 1, "body": "some comment", "postId": 1 }
  ],
  "profile": { "name": "john" }
}
```

json-server: operacje na kolekcjach

Podstawowe operacje na kolekcjach danych np.
Paginacja, sortowanie, filtrowanie itp.

GET /posts/1/comments?_start=20&_limit=10
GET /posts/1/comments?_sort=votes&_order=ASC
GET /posts?q=internet

json-server: custom routes

Definiowanie customowych ścieżek, w których możemy dokonać przekierowania, wysłać plik, przesłać określony response.

GET /kittens/1/says

```
var customRoutes = {
  '/kittens/:id/says': {
    method: 'GET',
    handler: function(req, res) {
      var sounds = ['meow', 'purr'];
      var message = {
        message: _.sample(sounds)
      };
      return res.json(message);
    }
  }
};
```

json-server: setup

Należy również skorzystać z pluginu zapewniającego proxy dla dla osobnego korzystania z API , który pozwala na:

- ominięcie problemu CORS, gdyż json-server i aplikacja frontendowa będą działać na różnych portach.
- elastyczne przełączanie się pomiędzy prawdziwym a fake'owym API np.

grunt serve --api=fake

json-server: Gruntfile.js

Wczytanie ustawień wybranego API:

```
grunt.loadNpmTasks('grunt-json-server');
grunt.loadNpmTasks('grunt-connect-proxy');

var apiConfig = {
  default: 'localhost:8080',
  fake:    'localhost:9010'
};
var selectedApi = apiConfig[grunt.option('api')] || apiConfig.default;
var selectedApiConfig = selectedApi.split(':');

var appConfig = {
  api: {hostname: selectedApiConfig[0], port: selectedApiConfig[1]}
};
```

json-server: Gruntfile.js

Konfiguracja w wywołaniu grunt.initConfig:

```
json_server: {
  custom_options: {
    options: {
      hostname: 'localhost',
      port: 9010,
      db: 'fake_api/fake_db.json',
      customRoutes: {
        '/kittens/:id/says': {
          method: 'GET',
          handler: function(req, res) {
            return res.status(400).json({error: 'Sleeps!'});
          }}}}}}
```

json-server: Gruntfile.js

Konfiguracja w wywołaniu grunt.initConfig:

```
connect: {
  proxies: [
    {
      context: '/api',
      host: appConfig.api.hostname,
      port: appConfig.api.port
    }
  ]
}
```

json-server: Gruntfile.js

Konfiguracja w wywołaniu grunt.initConfig:

```
concurrent: {
  server: {
    tasks: [
      'watch'
    ]
  },
  server_with_fake_api: {
    tasks: [
      'json_server',
      'watch'
    ]
  }
}
```

json-server: Gruntfile.js

Definicja tasku serve:

```
grunt.registerTask('serve', 'Start a web server', function (target) {
  var concurrentServerTask = 'concurrent:server';
  if (grunt.option('api') === 'fake') {
    concurrentServerTask = 'concurrent:server_with_fake_api';
  }

  grunt.task.run([concurrentServerTask]);
});
```

json-server: niezbędne pakiety

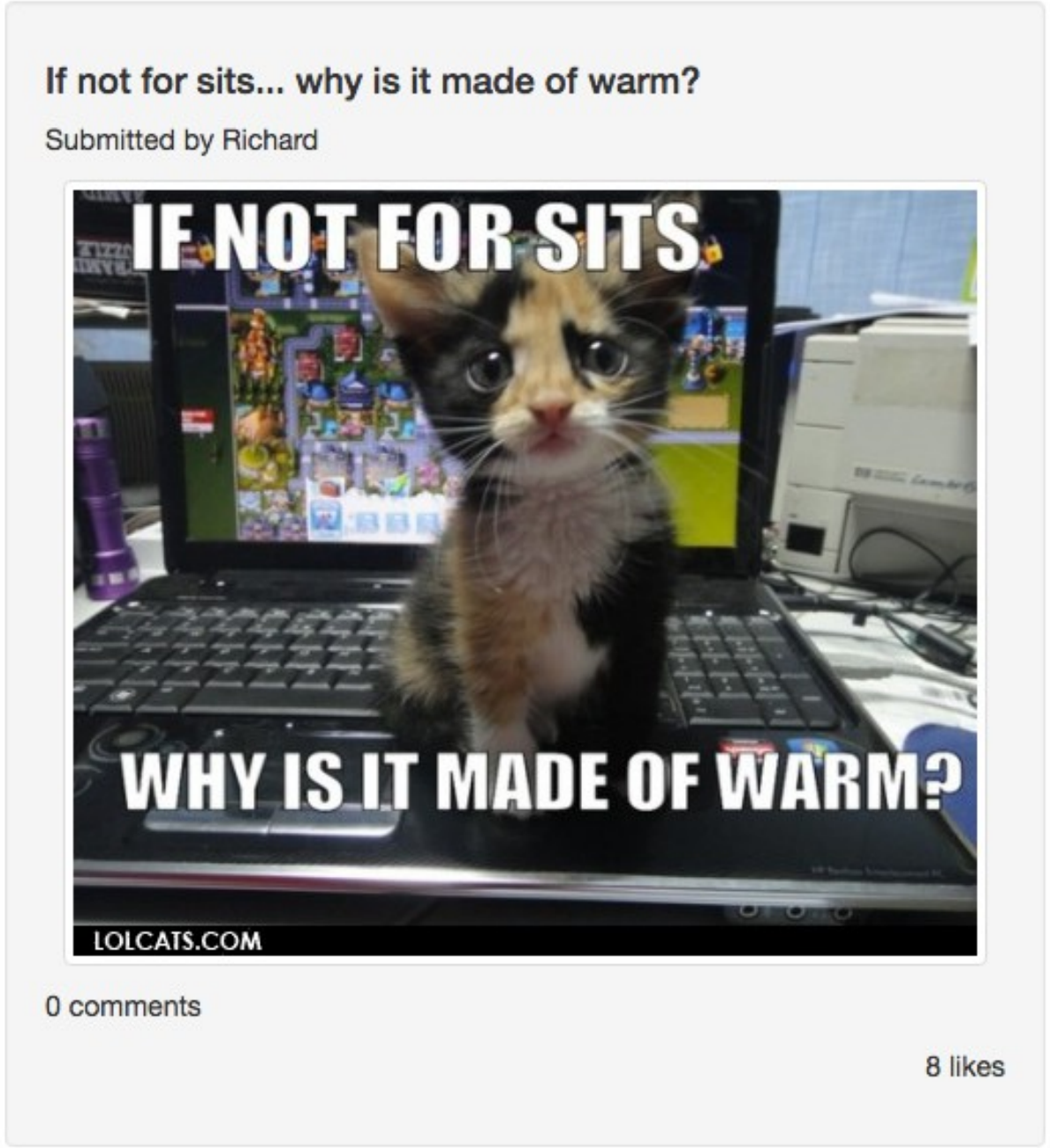
W zależności od wykorzystywanej biblioteki do automatyzacji zadań oraz serwera HTTP musimy wybrać właściwe pakiety, które pozwolą uruchomić nam json-server podczas uruchomienia aplikacji.

task runner	json-server	proxy
Grunt	grunt-json-server	grunt-connect-proxy
Gulp	gulp-json-srv	gulp-connect-proxy

json-server: demo

Aplikacja do przeglądania śmiesznych kotów

https://github.com/tbprojects/kittensApp



Pytania

We make IT JCommerce

We make IT JCommerce

I ty możesz zostać niezależnym developerem!

Tomasz Borowski

Frontend Developer

tomasz.borowski@jcommerce.pl