Pracuję na Netbeans 8.1, wszystkie projekty będą bazowały na tym IDE lub nowszych wersjach.

Do zarządzania pakietami używam NPM w wersji 3.10.6

Dodatkowo potrzebny jest Node.JS, u mnie w wersji 6.0.0

Odpalamy IDE, wybieramy „New project…”

* Categories HTML/JavaScript
* Project: HTML/JS Application
* „Next>”
* Project Name: Angular1\_sample
* 2x „Next>”

Zostawiamy zaznaczony tylko checkbox: Create package.json. plik ten będzie potrzebny managerowi pakietów NPM.

* Finish

W projekcie wystarczy nam katalog:

* Public\_html

I pliki:

* Index.html
* Package.json

OK, mamy projekt.

Teraz przyda nam się system kontroli wersji. Wybór pada na GITa.

Jako, że pracuję na systemie Windows, skupię się na tej platformie. Instalacja GITa pod Windows jest bardzo prosta. Ze strony <https://git-for-windows.github.io/> ściągamy instalator i instalujemy. It’s all.

Opis instalacji GITa dla innych systemów znajdziecie na stronie:

https://git-scm.com/book/pl/v1/Pierwsze-kroki-Instalacja-Git

Ogólnie jest tam świetnie wyjaśnienie jak korzystać z GITa. Polecam lekturę.

Sprawdzamy czy GIT się zainstalował. W konsoli będąc w katalogu projektu wpisujemy git --version. Jeśli wszystko jest OK, powinniśmy otrzymać informacje o zainstalowanej wersji.

Musimy teraz skonfigurować GITa.

Po pierwsze ustawiamy dane użytkownika:

* git config --global user.name "Jan Kowalski"
* git config --global user.email janek111@gmail.com

pracując pod Windows przydaje się ustawić domyślny edytor tekstowy z którego ma korzystać GIT. Najzwyklejszy notatnik wystarczy. Każdy commit zmian wymaga wpisania komentarza. W notatniku będzie to łatwiejsze niż pod edytorem consolowym

* git config --global core.editor notepad

Wpisując teraz

* git config --list

zobaczymy czy zmieniliśmy te ustawienia.

OK mamy bardzo wstępnie GITa skonfigurowanego. Przydało by się teraz podłączyć go do projektu.

Wpisujemy:

* git init

w katalogu projektu stworzy się katalog .git

Wpisując

* git status

sprawdzimy jakie pliki są śledzone, a jakie nie. Dodajemy pliki jakie chcemy żeby GIT śledził w nich zmiany:

* git add <fileName>

w naszym przypadku:

* git add package.json
* git add index.html

Teraz potrzebujemy dociągnąć Angular 1.5.8

W katalogu projektu w konsoli wpisujemy:

* npm install angular@1.5.8 --save

Dodatkowa komenda „—save” powoduje, że import zostanie dopisany do pliku package.json w części „dependencies”.

„Dependencies” przechowują informacje o importach pakietów niezbędnych do uruchomienia aplikacji.

W package.json jest jeszcze część nazwana „devDependencies”. W tym miejscu przechowujemy informacje o pakietach potrzebnych, ale nie niezbędnych do uruchomienia aplikacji. Przykładem jest lite-server.

Dla samej aplikacji nie ma znaczenia, czy umieścimy wpis w „dependencies” czy w „devDependencies”.

Projekt w Angularjs możemy uruchomić bezpośrednio w przeglądarce, ale ja proponuję http-server lub lite-server. W projekcie będziemy używać lite-servera.

Aby uruchomić server postępujemy podobnie jak w przypadku instalacji Angularjs:

* npm install http-server --save-dev

Do package.json zostanie dopisana linijka w devDependencies. Ręcznie jednak musimy dopisać kawałek kodu konfiguracyjnego:

"scripts": {

"prestart": "npm install",

"start": "http-server -a localhost -p 8000 -c-1"

}

Dla lite-server podobnie:

* npm install lite-server --save-dev

Do package.json zostanie dopisana linijka w devDependencies, a ręcznie dopisujemy:

"scripts": {

"prestart": "npm install",

"start": "npm run lite",

"lite": "lite-server"

}

Wprowadzimy lekkie zmiany jeszcze do pliku package.json i powinien on wyglądać mniej więcej tak:

{

"name": "Angular1\_sample",

"version": "1.0.0",

"author": "Robert",

"dependencies": {

"angular": "^1.5.8"

},

"devDependencies": {

"lite-server": "^2.2.2"

},

"scripts": {

"prestart": "npm install",

"start": "npm run lite",

"lite": "lite-server"

}

}

Pierwszych 3 właściwości nie trzeba wyjaśniać. „dependencies” i „devDependencies” opisałem. W części „scripts” wpisujemy to, co chcemy żeby NPM zrobił wraz z uruchomieniem:

* prestart – skrypt uruchamiany przed startem
* start – start uruchamiany w momencie wpisania w konsoli: npm start
* lite – alias do lite-server

Ok, mamy środowisko uruchomieniowe. Czas się zabrać za projekt.

Nie ważne jaki serwer wybierzemy, to w projekcie tworzymy katalog app. Katalog ten będzie przechowywał wszystkie pliki projektu w przyszłości, kiedy będziemy go rozbudowywać o kolejne funkcjonalności. Plik index.html zostaje w katalogu głównym.

Mamy wszystko, więc uruchamiamy projekt. W konsoli wpisujemy npm start.

W pierwszej fazie npm dociągnie nam wszystkie pakiety wpisane w „dependencies” i w „devDependencies”.

Jeśli wybraliście lite-server, to po uruchomieniu npm start, otworzy się nasza domyślna przeglądarka i wyświetli plik index, jeśli decydujemy się na http-server musimy sami odpalić przeglądarkę z adresem

http://localhost:8000/

Tak jak pisałem wcześniej ja będę pracował na lite-serwerze.

Polecam jeszcze dodatek Browsersync (<https://www.browsersync.io/>). Instalujemy go raz globalnie:

* npm install -g browser-sync

i używamy we wszystkich projektach. Jest to dodatek który odświeża przeglądarkę po każdym zapisaniu pliku w projekcie.

Jeśli używacie Netbeansa, to teraz przeglądarka powinna Wam pokazać statyczny napis:

„TODO write content”

Zmieńmy nasz nasz plik index.html na następujący:

index.html

<!DOCTYPE html>

<html>

<head>

<title>first Angular 1 app</title>

</head>

<body>

<div>{{ 2 + 2 + 2 }}</div>

</body>

</html>

W nawiasach deklarujemy kod, który jest odczytywany i wykonywany przez Angulara.

Zapisujemy plik. Jak widać, przeglądarka zamiast wyświetlić wynik, wyświetla zapis w nawiasach.

To oznacza, że Angularjs mimo ściągnięcia nie został jeszcze uruchomiony w projekcie. Musimy zrobić jeszcze 2 rzeczy:

po 1)

wciągnąć skrypt angulara do projektu:

do pliku index dopisujemy liniję: <script src="/node\_modules/angular/angular.min.js"></script>

po 2)

musimy określić, że jest to aplikacja angularowa. Aby to zrobić to w pliku index.html w tagu <html> dopisujemy <html ng-app>

Wprowadzamy zmiany i zapisujemy index.html. That all, przeglądarka wyświetli nam wynik dodawania z nawiasów klamrowych. A to znaczy, że mamy przygotowane środowisko pod Angularjs.

OK, zapiszmy jeszcze projekt w gicie.

Wpisujemy w konsoli:

* git commit

Otworzy się notatnik, w którym wpisujemy komentarz do zmian, po zamknięciu commit zostanie przeprowadzony.

Teraz korzystamy z lokalnego repozytorium GITa. Możemy jeszcze przecież korzystać z repozytorium zdalnego. Ja skorzystam z GitHuba. Jeśli ktoś nie ma jeszcze konta na <https://github.com/> to musi to zrobić.

Po zalogowaniu na githubie tworzymy nowe repozytorium. Nazwiemy je tak jak projekt: Angular1\_sample. Pod przyciskiem „Clone or download” znajdziemy adres repozytorium.

Uwstawimy teraz adres repozytorium w projekcie. Lokalnie w konsoli wpisujemy:

* git remote add <alias> <adres>

Aliasem będziemy się posługiwać później, podczas wysyłania projektu w świat.

Aby sprawdzić czy mamy zapisane zdalne repozytorium wpisujemy:

* git remote -v

Teraz możemy wypchnąć projekt do repozytorium.

Najpierw sprawdzamy czy wszystkie pliki które chcemy wysłać są monitorowane:

* git status

Jeśli jakiś plik nie jest monitorowany dodajemy go poprzez:

* git add <plik>

jeśli dodaliśmy jakiś plik lub plik nie został scommitowany wcześniej, taka informacja pojawi się po wpisaniu git status. Wtedy przed wysłaniem projektu musimy zrobić

* git commit

Po poprawnym commit wystarczy że wpiszemy:

* git push <aliasRepo> <branch>

a projekt zostanie wysłany na server.

I to tyle jak na pierwszy raz. Wiemy jak stworzyć podstawowy projekt w Angularjs, który będzie bazą pod kolejne funkcjonalności. Wiemy też jak stworzyć repozytorium GIT i wysłać projekt na serwer.

Fajny początek ☺

Pozdrawiam i zapraszam do kolejnych artykułów.

Robert