Projektowanie interfejsów webowych

NodeJS

Damian Koper

21 marca 2020

Spis treści

Spis rysunków

1 Wprowadzenie

Node.js to wieloplatformowe, asynchroniczne środowisko uruchomieniowe sterowane zdarzeniami, o otwartym kodzie do tworzenia skalowalnych aplikacji sieciowych w języku JavaScript [1]. Node.js napisany jest z użyciem języka C++ i do wykonywania kodu JavaScript używa silnika Google V8, który utworzony został na potrzeby projektu Chromium [2].

Dzięki osadzeniu na poziomie systemu operacyjnego, NodeJS pozwala na dużo więcej interakcji z jego zasobami niż zamknięte środowisko przeglądarki.

1.1 Instalacja

Instalacja NodeJS odbywa się poprzez pobranie i uruchomienie instalatora dla odpowiedniego systemu i architektury.
Odpowiedni plik można pobrać ze strony: `https://nodejs.org/en/download/`

Działanie NodeJS można zweryfikować komendą:

```
$ node -v
v12.13.0
$ npm -v
6.12.0
```

W przypadku konieczności korzystania z wielu wersji i przełączania pomiędzy nimi warto skorzystać z narzędzia NVM[3].

1.1.1 Środowisko pracy i debugowanie

Zalecanym środowiskiem pracy jest Visual Studio Code. Umożliwia ono łatwą konfigurację rozszerzeń i debugowania. `https://code.visualstudio.com/docs/nodejs/nodejs-debugging`

1.2 Pierwszy program

Uruchamianie skryptów w NodeJS działa na podobnej zasadzie co uruchamianie skryptów w języku Python. Niepodanie ścieżki do skryptu otworzy interpreter, z którego można wyjść wciskając podwójnie kombinację klawiszy CTRL+C.

```
$ node index.js
```

```
$ node
Welcome to Node.js v12.13.0.
Type ".help" for more information.
> XD = 2
2
```

Node przyjmie również ścieżkę do katalogu bez podanej nazwy pliku szukając w nim pliku `index.js` traktując katalog jak moduł.

W skryptach Node używać można dokładnie takiej samej składni jak w skryptach w przeglądarce internetowej. Istotną różnicą jest to, że interpreter nie udostępnia już globalnego obiektu `window`. Oferuje za to szereg nowych, które dostarczają informacje i obsługują interakcję z systemem operacyjnym[4].

1.2.1 Organizacja projektu

Aby utrzymać porządek w katalogu projektu w późniejszych etapach ćwiczenia, wszystkie pliki skryptów umieszczać należy w katalogu `src` i jego podkatalogach. Wszystkie pliki związane z Dockerem należy umieszczać w katalogu `docker`.

2 Ćwiczenie 1

Celem tego ćwiczenia będzie poznanie podstawy funkcjonowania środowiska NodeJS i całego ekosystemu jaki tworzy razem z dostępnymi pakietami i modułami.

2.1 Pierwsze zadanie

Zadanie 1. Proszę napisać w pliku `src/index.js` rekurencyjną funkcję `fibonacci(n)`, która oblicza dany wyraz ciągu Fibonacciego, a następnie wydrukować jej wynik do strumienia[5]:

1. `stdout` dla wartości od 1 do 7
2. `stderr` dla wartości od 8 do 16

Zakończenie programu ma kończyć się zwróceniem przez proces kodu 1 (odpowiednik w C++ `exit(1)`) Do testów należy użyć operatorów przekierowania strumienia `1>` i `2>`.
Na przykład `node src/index.js 1> /dev/null` albo `node src/index.js 1> null`.

2.2 Moduły i NPM

Node pozwala na importowanie modułów w formacie CommonJS[6]. Wyłączenie pewnych funkcjonalności do modułu pozwala na ich ponowne użycie w innych miejscach w kodzie.

W języku JavaScript wszystko jest obiektem, a każdy obiekt może zostać przez moduł wyeksportowany. Udostępniany globalnie[4] obiekt `module` umożliwia interakcję z danymi modułu. Pole `module.exports` przechowuje obiekt eksportowany przez moduł.

2.2.1 Własny moduł

Zadanie 2. Proszę wydzielić funkcję `fibonacci(n)` do modułu `src/math.js` i skorzystać z tego modułu w skrypcie głównym. Finalne działanie skryptu musi być identyczne jak wynik zadania 2.1. Wywołanie funkcji musi mieć postać `math.fibonacci(...)`.

2.2.2 Zewnętrzny moduł

NPM (Node Package Manager)[7] jest menadżerem pakietów, który pozwala na szybką instalację i wykorzystanie modułów. Pakiety umieszczone są w publicznym repozytorium[8] skąd każdy może je pobrać.

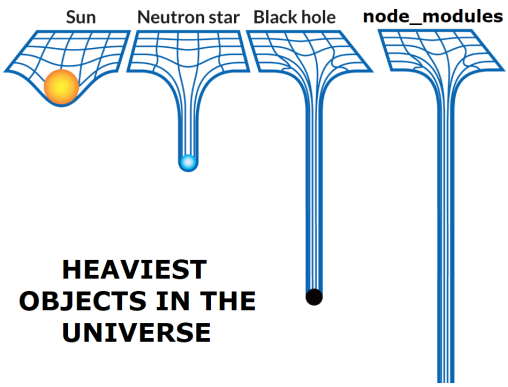
Aby ułatwić przenaszalność projektu i zarządzanie pakietami projekt sam powinien stać się pakietem. Prosze wykonać polecenie i odpowiedzieć wartościami domyślnymi na pytania:

```
$ npm init
```

Utworzy to plik `package.json`, który zawiera metadane pakietu. Nasza największą uwagę poświęcimy nieistniejącemu jeszcze polu `dependencies`, które zawiera używane przez nasz projekt pakiety. W celu jego utworzenia musimy zainstalować pierwszą zależność.

```
$ npm install one-liner-joke
```

Utworzony został wpis w `package.json` o zainstalowanej zależności. Utworzony został również plik `package-lock.json`, który zawiera dokładne, *zamrożone* informacje o naszych zależnościach, m.in. sumy kontrolne i dokładną lokalizację pliku w repozytorium. Utworzony został również katalog `node_modules`, który zawiera zainstalowany pakiet. Katalog ten zawiera ogólnodostępne pliki, więc w przypadku korzystania z systemu kontroli wersji trzeba wpisać go lokalizacji ignorowanych. W większych projektach katalog ten potrafi osiągnąć duże rozmiary na dysku, co prezentuje mem 1.



Rysunek 1: Rozmiar `node_modules`

Zadanie 3.1. Proszę wyszukać dokumentację i użyć pakietu `one-liner-joke` do wyświetlenia żartu (tylko treść). Skrypt za to odpowiedzialny proszę umieścić w pliku `src/joke.js`.

Zadanie 3.2. Proszę wyszukać dokumentację, zainstalować i użyć pakietu `axios`. Skrypt ma wykonać zapytanie do dowolnego otwartego API i wyświetlić odpowiedź w konsoli w sformatowanej formie (nie JSON). Skrypt za to odpowiedzialny proszę umieścić w pliku `src/axiosTest.js`. Propozycje API:

- `https://jsonplaceholder.typicode.com/`
- `https://randomuser.me/api/`

2.2.3 Wbudowane moduły

Node udostępnia również moduły wbudowane, które zapewniają interakcję z systemem operacyjnym.

Zadanie 4. Używając modułów `fs`[9], `http`[10] i `os`[11] proszę utworzyć serwer HTTP, który będzie miał zdefiniowane następujące endpointy GET i odpowiedzi:

1. `/ping` - odpowiedź `"pong"`,
2. `/datetime` - wysyła tekst z obecna datą dowolnie sformatowaną (można użyć pakietu `moment`),
3. `/cpus` - wysyła informacje (JSON) o zainstalowanych procesorach,
4. `/env` - wysyła informacje (JSON) o zmiennych środowiskowych serwera,
5. `/joke` - wysyła tekst z losowym żartem,
6. `/somedata` - wysyła zawartość pliku `assets/data.json`, który należy wcześniej utworzyć i wypełnić dowolnymi danymi.

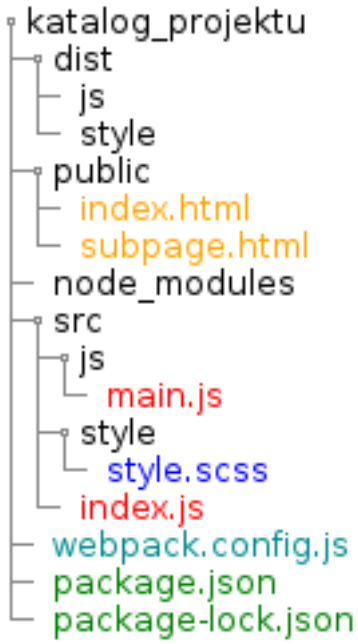
Proszę pamiętać o ustawieniu odpowiednich nagłówków (Content-Type). Skrypt proszę umieścić w pliku `src/server.js`. Testy rozwiązania mogą odbywać się poprzez przeglądarkę internetową. Zapytania testowe będą kierowane na adres `localhost:8088`. Pracę usprawnić może wykorzystanie pakietu Nodemon[12], który odpowiedzialny jest za automatyczne przeładowanie aplikacji przy edycji jej pliku źródłowego.

Zadanie 5. Dla chętnych. Używając Dockera i obrazu `node:alpine` proszę zbudować obraz i uruchomić kontener zawierający stworzony serwer z zadania 4.

3 Ćwiczenie 2

W tym ćwiczeniu zajmiemy się stworzeniem prostej strony internetowej zbudowanej z użyciem narzędzia Webpack.

Drzewo projektu wyglądać będzie teraz następująco:



Rysunek 2: Struktura katalogów projektu

Zadanie 1. Proszę w nowym folderze utworzyć potrzebne katalogi i pliki - na razie puste. Tworzenie plików `package*` i katalogu `node_modules` zostało ujęte w ćwiczeniu 1.

3.1 Wykorzystanie NodeJS do budowania strony internetowej

Strony mogą być zbudowane z użyciem wielu części i bibliotek. Analizując przypadkową stronę pod adresem np. `https://jsos.pwr.edu.pl` zauważyć można, że strona pobiera wiele zasobów z tego samego źródła. Każde pobranie zasobu obarczone jest obciążeniem serwera, który musi obsłużyć jedno żądanie i zaalokować zasoby. Może zdarzyć się sytuacja, że użytkownik *może być jednym z kilku tysięcy chętnych do odwiedzenia portalu* i serwer nie będzie w stanie obsłużyć wszystkich żądań.

Lepszym rozwiązaniem jest złączenie wszystkich plików w jeden, którego pobranie nie będzie tak obciążające dla serwera jak pobieranie ich większej ilości. Docelowo w procesie budowania

powstaną po jednym pliku `.js`, `.css` i dwa pliki `.html` odpowiadające tworzonym podstronom. Umożliwi to narzędzie Webpack.

3.1.1 Połączenie plików JS

Zadanie 2.1. Proszę zainstalować paczkę `webpack` i `webpack-cli`. Proszę również umieścić testowy `console.log()` w plikach `src/index.js` i `src/js/main.js` i zaimportować moduł `src/js/main.js` w pliku `src/index.js`. Zamiast składni `require`, możemy użyć składni `import`[21].

Zadanie 2.2. Na podstawie instrukcji konfiguracji[16] - rozdziały *Using a Configuration* i *NPM Scripts*, proszę edytować plik `webpack.config.js` i `package.json`, aby dla polecenia `npm run build` w katalogu `dist/js` utworzył się plik `main.js`. Jeśli utworzył się gdzieś indziej należy edytować odpowiednio konfigurację.

Webpack połączył dwa pliki w jeden i wywołanie `node dist/js/main.js` powinno wyświetlić testowe logi w kolejności ich umieszczenia uwzględniając importowane pliki. Po wykonaniu każdego dalszego zadania, aby zobaczyć wyniki, należy przebudować stronę.

3.1.2 Budowanie plików HTML

Budowanie plików HTML polegać będzie na ich skopiowaniu i dodaniu linków do towarzyszących plików. Wykorzystamy do tego plugin.

Zadanie 3.1. Proszę w pustych plikach HTML utworzyć standardowy szkielet strony z dowolnymi testowymi elementami. Proszę również o zainstalowanie paczki `html-webpack-plugin`.

Zadanie 3.2. Proszę zaimportować plugin `HtmlWebpackPlugin` i dodać pole w obiekcie konfiguracyjnym Webpacka z następującą treścią:

```
plugins: [
    new HtmlWebpackPlugin({
        template: path.resolve(__dirname, 'public/index.html'),
        filename: path.resolve(__dirname, 'dist/index.html')
    }),
    new HtmlWebpackPlugin({
        template: path.resolve(__dirname, 'public/subpage.html'),
        filename: path.resolve(__dirname, 'dist/subpage.html')
    })
]
```

Oczywiście będzie potrzeba zaimportowania pluginu pod nazwą `HtmlWebpackPlugin` do tego pliku. Plugin ten skopiuje pliki HTML jednocześnie dodając do nich plik `js/main.js`.

3.1.3 Połączenie plików SCSS

SCSS jest preprocessorem plików CSS, który pozwala na używanie zmiennych, zagnieżdżanie bloków i wiele innych[17].

```
$font-stack: Helvetica, sans-serif;
$primary-color: #333;
body {
    font: 100% $font-stack;
    color: $primary-color;
    p {
        margin: 0;
    }
}
```

Webpack opera się na koncepcji loaderów. Kiedy plik jest importowany, webpack sprawdza czy pasuje on do zdefiniowanych wzorców i na tej podstawie uruchamia swoje loadery, którym wejściem i wyjściem w ogólnym przypadku jest plik. W tym wypadku plikiem wejściowym będzie `style.scss`, a wyjściowym `style.css`. W domyślnej konfiguracji loader za to odpowiedzialny (`style-loader`) nie wygeneruje pliku tylko umieści kod w pliku `main.js`, który sam doda odpowiednie wpisy w sekcję `<head>` strony. Wydzielenie styli do osobnego pliku wymaga zastosowania innego loadera wraz z pluginem (`mini-css-extract-plugin`).

Zadanie 4.1. W pliku `style.scss` proszę stworzyć testowe style z wykorzystaniem elementu składni SCSS stylujące element `body` albo jakikolwiek inny utworzony element strony.

Zadanie 4.2. W oparciu tylko o rozdział *Getting Started*[18] instrukcji proszę dodać odpowiednią konfigurację i zaimportować odpowiednie pliki do pliku `src/index.js`. W przypadku brakujących modułów należy je zainstalować.

W tym momencie, po przebudowaniu strony, Webpack (w zasadzie `style-loader`) sam umieści tag `<style>` na podstawie kodu dołączonego do pliku `dist/js/main.js`.

Zadanie 4.3. Proszę, używając paczki `mini-css-extract-plugin`[19] sprawić, aby style strony pochodziły z pliku `dist/style/style.css`, a nie z tagu dodawanego dynamicznie. Wymaga to podmiany loadera `style-loader`. W przypadku problemów z pojawieniem się zbudowanego pliku w złym miejscu należy dodać ustawienia nowo użytego pluginu - pole `filename`. Niezależnie gdzie pojawi się plik, `html-webpack-plugin` poprawnie dołączy go do plików HTML. Wykonanie tego zadania nie jest konieczne do przejścia dalej.

3.2 Automatyczne przebudowywanie

Użycie komendy `webpack` z flagą `--watch` sprawi, że webpack będzie przebudowywać pliki po każdej aktualizacji kodu.

3.2.1 WebpackDevServer

Paczka `webpack-dev-server` pozwala na automatyczne utworzenie serwera, na którym działa nasza strona. Automatycznie również uruchamia proces przebudowania strony po każdej zmianie w plikach źródłowych. Zajmuje się również automatycznym odświeżeniem strony w przeglądarce.

Zadanie 5. Proszę dodać do obiektu konfiguracji Webpacka następujące pole:

```
devServer: {
    contentBase: path.join(__dirname, 'dist'),
    port: 9000
}
```

Poinformuje ono serwer gdzie szukać strony i na jakim porcie nasłuchiwać. Zgodnie z rozdziałami *Getting Started* i *Usage* pliku README[20] proszę zainstalować i uruchomić serwer. W przypadku problemów z konfiguracją serwera należy sprawdzić poprawność ścieżek w polach ścieżek do plików w pliku konfiguracyjnym.

3.3 Rozwój strony

W tej części ćwiczenia zabronione jest edytowanie sekcji `<head>` w plikach HTML.

Zadanie 6. Z wykorzystaniem stworzonego środowiska - testowego serwera, budowanych zależności plików skryptów i stylów, proszę rozwinąć puste strony w dowolnym kierunku. Wszystkie biblioteki należy importować w pliku `src/main.js` lub, jeśli zostaną stworzone, w jego importowanych modułach. Wszystkie style instalowanych bibliotek proszę importować w pliku `src/style/style.scss` albo `src/main.js`. Konfigurację Webpacka można dowolnie rozszerzać. Propozycje:

- Strona wyświetlająca dane wraz ze zdjęciami z któregoś z ogólnodostępnych API.
  Lista API: `https://github.com/public-apis/public-apis`
- Strona wyświetlająca kalendarz aktualnego miesiąca (tabela) z możliwością dodania wydarzenia (przechowywanie lokalne w ramach sesji). Wskazana biblioteka `moment`.

- Strona wyświetlająca i obsługująca grę w kółko i krzyżyk (albo dowolną inną). Wyświetlanie za pomocą elementów DOM'u albo na elemencie Canvas (możliwość wykorzystania biblioteki `Konva`).

Strona powinna używać frameworków CSS i JS takich jak:

- Axios - zapytania
- Bulma - wygląd strony
- Bootstrap - wygląd i funkcjonalność strony
- jQuery - łatwa interakcja z DOM'em i obsługa zdarzeń
- i wiele innych dostępnych, wszystko zależy od funkcjonalności...

Uwaga! W przypadku konieczności obsługi wybranego frameworka konieczne może być zmodyfikowanie konfiguracji Webpacka, np. dodanie nowych reguł dla plików, użycie nowych pluginów. Do importowania plików CSS (obecnie działa tylko SCSS), potrzeba dodać regułę z odpowiednimi loaderami kolejno `css-loader -> style-loader`. Frameworki powinny podawać instrukcję konfiguracji i importowania.

Literatura

[1] NodeJS: https://nodejs.org/en/about/

[2] V8: https://en.wikipedia.org/wiki/V8_(JavaScript_engine)

[3] NVM Linux/MacOS: https://github.com/nvm-sh/nvm
NVM Windows: https://github.com/coreybutler/nvm-windows

[4] Node Globals: https://nodejs.org/api/globals.html

[5] Console: https://nodejs.org/api/console.html

[6] CommonJS: https://flaviocopes.com/commonjs/
Dokumentacja: https://nodejs.org/docs/latest/api/modules.html

[7] NPM: https://docs.npmjs.com/about-npm/

[8] Repository: https://docs.npmjs.com/about-the-public-npm-registry

[9] File system: https://nodejs.org/dist/latest-v12.x/docs/api/fs.html#fs_fs_readfilesync_path_options

[10] HTTP: https://nodejs.org/dist/latest-v12.x/docs/api/http.html#http_class_http_server

[11] OS: https://nodejs.org/dist/latest-v12.x/docs/api/os.html#os_os_cpus

[12] Nodemon: https://nodemon.io/

[13] ECMAScript: https://en.wikipedia.org/wiki/ECMAScript#Conformance

[14] Kompatybilność https://kangax.github.io/compat-table/es2016plus/

[15] Babel: https://babeljs.io/

[16] Webpack - bazowa konfiguracja: https://webpack.js.org/guides/getting-started/#using-a-configuration

[17] SASS - https://sass-lang.com/guide

[18] SASS - Webpack: https://webpack.js.org/loaders/sass-loader/#getting-started

[19] Extract CSS - https://webpack.js.org/plugins/mini-css-extract-plugin/

[20] WebPackDevServer - https://github.com/webpack/webpack-dev-server

[21] ES6 Modules - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import
https://medium.com/backticks-tildes/introduction-to-es6-modules-49956f580da