

Node.js

1. Node - wprowadzenie

wieloplatformowe środowisko uruchomieniowe o otwartym kodzie do tworzenia aplikacji typu server-side napisanych w języku JavaScript. Przyczynił się do stworzenia paradygmatu „JavaScript everywhere” umożliwiając programistom tworzenie aplikacji w obrębie jednego języka programowania zamiast polegania na odrębnych po stronie serwerowej.

Node.js składa się z silnika V8 (stworzonego przez Google), biblioteki libUV oraz kilku innych bibliotek. Został stworzony przez Ryana Dahla na początku 2009 roku, jego rozwój sponsorowany był przez firmę Joyent, w której pracował.

Pierwotnym celem Dahla było dodanie do stron internetowych możliwości technologii push, widocznej w aplikacjach pocztowych takich jak Gmail. Po wypróbowaniu różnych języków zdecydował się na JavaScript, ze względu na brak istniejącego API wejścia/wyjścia. Dało mu to możliwość wykorzystania nieblokującego, sterowanego zdarzeniami wejścia/wyjścia.

Domyślnym managerem pakietów dla Node.js jest Npm.

1.1. Zalety Node'a

1. Asynchroniczność
2. Wielowątkowość
3. npm - zbiór bibliotek

1.2. Różnice pomiędzy Nodem a JSem:

- w NodeJS brak globalnego obiektu `window`
- umiejętność czytania plików przez NodeJS (czego sam JS nie potrafi)
- brak wsparcia dla API przeglądarki jak to ma miejsce w JSie (np. brak obsługi metod typu: `document.getElementById(" ")` , `document.querySelector("#demo")`

2. Instalacja Node'a

Ściągnij Node'a zgodnie ze swoim systemem <https://nodejs.org/en/download/> a następnie zainstaluj postępując zgodnie z instrukcjami na ekranie. Po instalacji ponownie uruchom

komputer.

3. Pierwsze kroki

Aby sprawdzić, czy Node został prawidłowo uruchomiony w oknie terminala wpisz:

```
node -v
```

Powinieneś zobaczyć numer wersji zgodny z tym, który przed chwilą ściągnąłeś ze strony internetowej.

Aby wejść w tryb node'a wpisz w terminalu polecenie:

```
node
```

3.1. Podstawowe komendy

Jak zauważysz, wygląd terminalu przypomina najzwyklejszy terminal. Możesz w nim (w oknie terminala) wykonywać operacje JavaScript. Wpisz:

```
const x = 2 [enter]  
const y = 5 [enter]  
const z = x + y [enter]
```

Aby wyjść z trybu node'a należy dwukrotnie wcisnąć kombinację klawiszy `ctrl + c`

3.2. Pierwsza aplikacja

Utwórz na pulpicie folder o nazwie `kurs_node_js` (lub inną dowolną nazwę, ale bez spacji!), a następnie otwórz ten folder w VS Code. Otwórz również terminal.

Utwórz nowy plik o nazwie:

```
app.js
```

a w pliku tym wpisz:

```
console.log("Hello World")
```

Zapisz plik i w terminalu wpisz:

```
node app.js
```

Uruchomiłeś właśnie pierwszą aplikację - powinieneś zobaczyć **Hello World** w terminalu.

Zadanie:

Napisz funkcję (ES6), która będzie wyświetlała "Hello World".

Odpowiedź:

```
const sayHello = () => { console.log("Hello World") }  
sayHello(); //wywołanie funkcji
```

Jak widzisz, jest to zwykła funkcja. Podobnie korzystaliśmy już z niej przy okazji omawiania webpacka, gdzie tworzyliśmy funkcję do sumowania, a następnie wywoływaliśmy wynik w konsoli.

```
const add = (a,b) => {  
    console.log(a + b)  
}  
  
add(2,3)
```

3.3 Wiele plików *.js

Aby móc zapanować nad kodem stosuje się wiele plików JSowych, które to następnie są ze sobą linkowane aby mogły nawzajem stosować korzystać z napisanych elementów.

Utwórz nowy plik

```
functions.js
```

a w nim:

```
const sayHello = () => { console.log("Hello World") }  
  
const add = (a, b) => {  
    console.log(a + b)  
}  
  
module.exports = {  
    sayHello,  
    add  
}  
  
console.log("Everything is loaded correctly")
```

Z kolei w pliku `app.js` niech będzie tylko:

```
const functions = require('./functions');
functions.sayHello();
functions.add(3, 2);
```

3.4 Tworzenie serwera HTTP

Celem jest stworzenie aplikacji dostępnej z poziomu przeglądarki!

Usuń zawartość z pliku `app.js` i wpisz:

```
const http = require('http');
const server = http.createServer();
const port = 3000;
server.listen(port, () => {
  console.log(`Serwer działa na porcie ${port}`);
})
```

Gdy uruchomisz aplikację (`node app.js`) serwer będzie działał, ale strona internetowa na `localhost:3000` jeszcze się nie wczyta, gdyż nie powiedzieliśmy, co ma się wyświetlić.

Do tego jest nam potrzebna funkcja, która będzie odpowiadała za wyświetlanie. Przyjęło się, że nazywa jest ona jako `handler` i jest on callbackiem dla metody `createServer` która przyjmuje tylko dwa parametry: `request` oraz `response` .

Całość wyglądałaby zatem tak:

```
const http = require('http');
const handler = (request, response) => {
  console.log("sample message");
}
const server = http.createServer(handler);
const port = 3000;
server.listen(port, () => {
  console.log("serwer działa na porcie", port);
})
```

Zapisz i uruchom plik komendą `node app.js` i poczekaj chwilkę nim pojawi się `sample message` . Wejdź na stronę `localhost:3000` i zauważ, ilekroć odświeżysz stronę, tyle razy nowa wiadomość pojawi się w oknie terminala.

Na stronie internetowej nadal nic się nie pojawia. Dopisz linijkę:

```
const http = require('http');
const handler = (request, response) => {
  console.log("sample message");
  response.end("Hello World");
}
```

Uruchom ponownie skrypt i zobacz, że napis na stronie na porcie 3000 został wydrukowany.

Wyświetlenie błędu serwera

Jeżeli byłby problem z uruchomieniem serwera:

```
server.listen(port, (err) => {
  if (err) {
    return console.log(`Brr, coś poszło nie tak: ${err}`)
  }
  console.log("serwer działa na porcie", port)
})
```

4. Express

Powyższy sposób pisania jest niezbyt prakyczny i długi. Aby usprawnić proces stosuje się bibliotekę `express`.

Informacje: <https://www.npmjs.com/package/express>

4.1 Instalacja

W terminalu wpisz: `npm init -y`

Zainstaluj express: `npm install express@4.17.1 --save` lub
`npm install express --save-prod`

4.2 Wykorzystanie

Zmień plik `app.js` na:

```
const express = require("express"); //nie korzystam już z http!!
const port = 3000;
const app = express();
//gdy uzytkownik wchodzi na stronę główną
app.get('/', function (req, res) {
    res.send("Hello World");
})
app.listen(port)
```

Częściej stosuje się notację ES5 ze względu na kontekst `this`. Wejdź na stronę:

`http://localhost:3000/`

```
const express = require("express"); //nie korzystam już z http!!
const port = 3000;
const app = express();
//gdy uzytkownik wchodzi na stronę główną
app.get('/', function (req, res) {
    res.send("<h2> Hello World </h2>");
})
//gdy uzytkownik wchodzi na stronę o nas
app.get('/about', function (req, res){
    res.send("My site")
})
app.listen(port, (err) => {
    if (err) {
        return console.log("coś poszło nie tak...:", err)
    }
    console.log("serwer działa na porcie", port)
})
```

Wejdź na stronę: `http://localhost:3000/about`

5. Szablony

5.1 Dodanie plików hbs

Aby wyświetlić stronę internetową stosuje się m.in. szablony Handlebars

```
const express = require("express");
const port = 3000;
const app = express();
//ustawienie, ze moja aplikacja musi korzystac z silnika hbs
app.set("view engine", 'hbs')
//gdy uzytkownik wchodzi na strone
app.get('/', function (req, res) {
    res.render('index')
})

app.get('/about', function (req, res){
    res.send("My site")
})

app.listen(port, (err) => {
    if (err) {
        return console.log("coś poszło nie tak...", err)
    }
    console.log("serwer działa na porcie", port)
})
```

Handlebars domyślnie szuka strony w folderze `views` więc stwórz taki folder i utwórz w nim plik `index.hbs` i stwórz normalną stronę internetową, a następnie uruchom stronę.

Instalacja hbs: `npm install hbs --save` lub `npm install hbs --save-prod`.

Gdyby był błąd `can not find hbs` to rozwiązanie: `sudo npm link hbs`

5.2. Dodanie plików CSS

W katalogu głównym stwórz folder `assets` a w nim `css`.

W pliku `hbs` podepnij style CSS:

```
<link rel="stylesheet" type="text/css" href="../../assets/css/style.css">
```

A w pliku `app.js` dodaj te dwie linijki:

```
// Podpięcie css
const path = require('path')
app.use('/assets', express.static(path.join(__dirname, './assets')));
```

5.3. Dodanie plików JS

W katalogu głównym utwórz folder `js` a w nim plik `func.js` i wpisz do niego:

```
alert("Hello World");
```

W pliku `app.js` dopisz:

```
app.use('/js', express.static(path.join(__dirname, './js')));
```

Pierwszy parametr `js` to nazwa folderu w którym znajduje się plik JS, a drugi `js` to nazwa ścieżki, gdzie plik JS się znajduje.

W pliku `*.hbs` dopisz:

```
<script src="../../js/func.js"></script>
```

Inny przykład:

```
app.use('/js', express.static(__dirname + '../../public/js'));  
<script src="/js/validate.min.js"></script>
```

Bardziej zaawansowany przykład:

w pliku `hbs`:

```
<input type="text" id="example">  
<button id="action">Klik</button>  
<p id="text"></p>
```

w pliku `func.js`

```
const example = () => {  
  const fromInput = document.getElementById("example").value;  
  document.getElementsByTagName("p")[0].innerText = fromInput;  
}  
  
document.getElementById("action").addEventListener("click", example)
```

5.4. Nawigacja między stronami

W pliku `index.hbs` dodaj kod

```
<a href="/subpage">Klik</a>
```

Zauważ, że żadne rozszerzenie pliku nie jest już potrzebne!

5.5. Przekazywanie zmiennych z node.js na stronę internetową

Przykład

W pliku `app.js` zmodyfikuj fragment:

```
app.get('/', function (req, res) {  
  res.render('index', {  
    pageTitle: "Lekcja01"  
  })  
})
```

a z kolei w pliku `hbs` dodaj:

```
<title>{{ pageTitle }}</title>
```

Przykład

Do wcześniej stworzonego pliku `functions.js` dopisz:

```
const sayHello = () => { console.log("Hello World") }  
const add = (a, b) => {  
  console.log(a + b)  
}  
  
const someTitle = "example example" //kod do dopisania  
module.exports = {  
  sayHello,  
  add,  
  someTitle //kod do dopisania  
}
```

W pliku `app.js` dopisz:

```
const fromAnotherFile = require("./functions")  
app.get('/', function (req, res) {  
  res.render('index', {  
    pageTitle: "Lekcja01",  
    subTitle: fromAnotherFile.someTitle  
  })  
})
```

W pliku `index.hbs` dopisz:

```
{{ subTitle }}
```

Przykład

W pliku `app.js` dopisz:

```
const sample = () => 'proba'

app.get('/', function (req, res) {
  res.render('index', {
    anotherTitle: sample()
  })
})
```

Cały plik `app.js`

```

const fromAnotherFile = require("./functions")
const express = require("express");
const app = express();
const port = 3000;
const x = 10;
//kod JS

//ustawienie, ze moja aplikacja musi korzystac z silnika hbs
app.set("view engine", 'hbs')
//wyrenderowanie strony, gdy uzytkownik wchodzi na strone
app.get('/', function (req, res) {

  const sample = () => {
    return 'proba'
  }

  res.render('index', {
    pageTitle: "Lekcja01",
    subTitle: fromAnotherFile.someTitle,
    anotherTitle: sample(),
    subSubTitle: x == 10 ? sample() : null,
  })
})
//wypisanie na stronie tekstu
app.get('/about', function (req, res) {
  res.send("My site")
})
// Podpięcie css
var path = require('path')
app.use('/assets', express.static(path.join(__dirname, "./assets")));
//podpięcie js
app.use('/js', express.static(path.join(__dirname, "./js")));
//server
app.listen(port, (err) => {
  if (err) {
    return console.log("coś poszło nie tak...", err)
  }
  console.log("serwer działa na porcie", port)
})

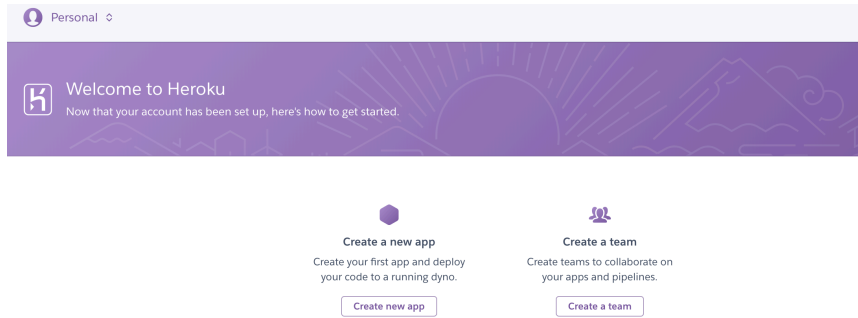
```

6. Deploy

6.1 Zakładanie konta w serwisie Heroku


Aby umieścić stronę w Internecie, utwórz konto na stronie: <https://www.heroku.com/>. Przy rejestracji jako główne środowisko wybierz `node.js`.

Po zatwierdzeniu konta kliknij `create a new app` :






Nadaj nazwę aplikacji:


App name

 
exampleappformystudies is available

Choose a region

 Europe 

 Add to pipeline...



a następnie w prawym górnym rogu kliknij `open .app` - w moim przypadku to:

```
https://exampleappformystudies.herokuapp.com/
```

6.2. Heroku CLI

6.2.1. Instalacja:

Wybierz i zainstaluj odpowiednią wersję: <https://devcenter.heroku.com/articles/heroku-cli>

6.2.2. Zmiana portu

W naszej aplikacji używaliśmy portu 3000. Musimy teraz jednak poinformować naszą aplikację, z jakiego portu korzystamy.

W pliku `app.js` zamień linię

```
const port = 3000;
```

na

```
const port = process.env.PORT || 3000;
```

6.2.3. Uruchamianie skryptu

O ile w terminalu uruchamiamy skrypt poleceniem `node app.js` tak musimy także poinformować, jak Heroku uruchomi naszą aplikację.

W pliku `package.json` dopisz:

```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "start": "node app.js" //chodzi o tę linię kodu  
},
```

6.2.4. Deploy Po instalacji ponownie uruchom terminal i postępuj według instrukcji na stronie: <https://dashboard.heroku.com/apps/exampleappformystudies/deploy/heroku-git>, tj:

```
heroku login  
git init  
heroku git:remote -a twojanazwaaplikacji
```

Deploy app

```
git add .  
git commit -am "make it better"  
git push heroku master
```

A dla już istniejącego repozytorium:

```
heroku git:remote -a exampleappformystudies
```

6.3. GitHub

Aby wydeployować stronę poprzez GitHuba wyślij swoje pliki do repozytorium (wyłączając folder `node_modules`), a następnie z opcji wybierz `github`.

Pojawi się okno z prośbą o wpisanie nazwy repozytorium, które chcesz połączyć. Wpisz je, kliknij `Connect` a po poprawnym połączeniu na dole strony kliknij `deploy manually` - robisz to tylko za pierwszym razem, za każdym następnym dane zostaną uaktualnione automatycznie!

