Programowanie w JAVASCRIPT

LAB Node.js

Konfiguracja środowiska

Jeśli w systemie brak środowiska to zaczynamy od pobrania Node.js:

https://nodejs.org/en/

I zainstalowania najnowszej wersji.

W pakiecie z Node.js instalowany jest automatycznie menadżer paczek **npm**.

Node.js – przygotowanie projektu.

Jeśli utworzyłeś go na poprzednich zajęciach, możesz przejść do pkt następnego

Utwórz katalog (gdziekolwiek) na projekty (np.: node-projects) a w nim utwórz katalog z laboratorium: ImieNazwiskoNodeLab10.
(Imię i Nazwisko wpisz SWOJE :)

Uruchom **konsolę windows**. To tam będziesz zarządzał projektem.
(możesz ściągnąć i użyć także bardziej rozbudowanej konsoli takiej jak **cmder** lub **PowerShell** albo dowolnej innej - zalecane).

Node.js przy instalacji uzupełnia zmienne środowiskowe o swoją ścieżkę i jest widoczny z każdego poziomu systemu.

Poprzez komendę **cd** (change directory) przejdź do swojego projektu.

Możesz napisać **cd** i skopiować ścieżkę do swojego projektu. Jeśli używasz **cmder** nie zapomnij o przejściu na dysk z projektem poprzez :

```
literadysku:
```

np. d:

Wpisz:

```
npminit
```

i naciskaj **enter** przechodząc kolejne etapy tworzenia pliku konfigurującego. Przede wszystkim wpisz autora projektu. Opcja ta zainicjalizuje projekt i utworzy w nim plik konfiguracyjny w formacie JSON z danymi aplikacji o nazwie *package.json*

Jeśli nie masz, to zainstaluj bardzo wygodne narzędzie do Node.js – *nodemon*:

```
Npm install nodemon-g
```

narzędzie instalujemy globalnie dzięki opcji *-g* , dzięki czemu będzie można je wykorzystać we wszystkich projektach. Uruchamiaj wtedy projekt:

```
nodemon index.js
```

Utwórz w projekcie plik *index.js*

Projekt gotowy do pracy. Otwórz obydwa pliki (package.json i index.js).

Node.js - wywołania zwrotne i nasłuchiwanie zdarzeń

ZADANIE 1. Wywołania zwrotne

Wywołanie zwrotne jest funkcją przekazywaną jako argument funkcji asynchronicznej i opisuje to, co powinno zostać zrobione po zakończeniuoperacji asynchronicznej. W programowaniu Node.js wywołania zwrotne sąwykorzystywane częściej niż emitery zdarzeń, a ponadto są bardzo łatwe wużyciu.

Aby zademonstrować użycie wywołań zwrotnych w aplikacji, utworzymy terazprosty serwer HTTP, który będzie wykonywał następujące zadania:

1. Asynchronicznie pobierał z pliku w formacie JSON jakieś sentencje.
2. Asynchronicznie pobierał prosty szablon HTML.
3. Przygotowywał stronę HTML, łącząc wspomniany szablon z pobranymisentencjami.
4. Gotową stronę HTML wysyłał do użytkownika.

Utwórz plik *titles.json* i uzupełnij go tablicą:

titles.json

```
[
"Początek jest najważniejszą częścią pracy.",
"Jest tylko jedno lekarstwo na duże kłopoty - małe radości.",
"Z uśmiechem na twarzy człowiek podwaja swoje możliwości.",
"Twój czas jest ograniczony, więc nie marnuj go na bycie kimś, kim nie jesteś.",
"Nie licz dni, spraw by dni się liczyły.",
"Nasze życie jest takim, jakim uczyniły je nasze myśli"
]
```

Utwórz szablon *template.html* i uzupełnij kod:

template.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Laboratorium 10</title>
</head>
<body>
  <h1>Lista sentencji</h1>
  <ul><li>%</li></ul>
</body>
</html>
```

Znak % zostanie zastąpiony sentencją z pliku JSON.

Teraz utwórzmy serwer:

index.js

```
var http = require('http');
var fs = require('fs');

http.createServer(function(req, res) {

  if (req.url == '/') {
    fs.readFile('./titles.json', function(err, data) {
      if (err) {
        console.error(err);
        res.end('titles.json error');
      }
      else {
        var titles = JSON.parse(data.toString());
        fs.readFile('./template.html', function(err, data) {
          if (err) {
            console.error(err);
            res.end('template.html error');
          }
          else {
            var tmpl = data.toString();
            var html = tmpl.replace('%', titles.join('</li><li>'));
            res.writeHead(200, {'Content-Type': 'text/html'});
            res.end(html);
          }
        });
      }
    });
  }
}).listen(8000, "127.0.0.1");
```

Wyjaśnienie kodu:

`http.createServer(function(req, res) {…});`- utworzenie serwera HTTP i użycie wywołaniazwrotnego w celu zdefiniowania logiki odpowiedzialnej za udzielanie odpowiedzi.

`fs.readFile('./titles.json', function(err, data) {…});` - odczyt pliku JSON i użyciewywołania zwrotnego do zdefiniowania, co należy zrobić z zawartością odczytanegopliku.

Większość wbudowanych modułów Node.js używa wywołań zwrotnych wraz z dwoma argumentami.Pierwszy jest przeznaczony dla błędu, o ile jakikolwiek

wystąpi, natomiast drugi dla wyniku wywołania.Argument błędu jest zwykle w postaci skrótu `er` lub `err`.

`if (err) {…}` - jeżeli wystąpi błąd, informacje o nim należy zarejestrować w

dzienniku zdarzeń i wyświetlić klientowi odpowiedni komunikat.

`vartitles = JSON.parse(data.toString());` - przetworzenie danych JSON.

`fs.readFile('./template.html', function(err, data) {…});` - odczyt szablonu HTML I użycie wywołania zwrotnego po wczytaniu szablonu.

`let html = tmpl.replace('%', titles.join('</li><li>'));` - przygotowaniestrony HTML zawierającej sentencje.

`res.writeHead(200, {'Content-Type': 'text/html'});` - utworzenie nagłówka odpowiedzi

`res.end(html);` - przekazanie użytkownikowi wygenerowanej strony HTML.

Sprawdź działanie serwera uruchamiając:

http://localhost:8000/

w przeglądarce.

Zadanie zawiera trzy poziomy wywołań zwrotnych:

```
http.createServer(function(req, res) { ...
fs.readFile('./titles.json', function (err, data) { ...
fs.readFile('./template.html', function (err, data) { ...
```

im więcej poziomów wywołań zwrotnych będzie znajdowało się w kodzie, tym bardziej będzie on zagmatwany, co utrudni refaktoring i testowanie.

(*refaktoring* to proces wprowadzania zmian w projekcie/programie, w wyniku których zasadniczo nie zmienia się funkcjonalność. Celem refaktoryzacji jest więc nie

wytwarzanie nowej funkcjonalności, ale utrzymywanie odpowiedniej, wysokiej jakości organizacji systemu.)

Dobrym nawykiem jestograniczanie zagnieżdżania wywołań zwrotnych. Dzięki utworzeniu nazwanychfunkcji obsługujących poszczególne poziomy zagnieżdżenia wywołań zwrotnychtę samą logikę możesz osiągnąć za pomocą mniejszej liczby wierszy kodu, coułatwia jego obsługę, testowanie i refaktoring.

Przetestuj nowy kod:

index.js

```
var http = require('http');
var fs = require('fs');

var server = http.createServer(function (req, res) {
  getTitles(res);
}).listen(8000, "127.0.0.1");

function getTitles(res) {
  fs.readFile('./titles.json', function (err, data) {
    if (err) {
      hadError(err, res);
    }
    else {
      getTemplate(JSON.parse(data.toString()), res);
    }
  });
}

function getTemplate(titles, res) {
  fs.readFile('./template.html', function (err, data) {
    if (err) {
      hadError(err, res);
    }
    else {
      formatHtml(titles, data.toString(), res);
    }
  });
}

function formatHtml(titles, tmpl, res) {
  var html = tmpl.replace('%', titles.join('</li><li>'));
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end(html);
}

function hadError(err, res) {
  console.error(err);
  res.end('Error');
}
```

Wyjaśnienie kodu:

`let server = http.createServer(function (req, res) {` - żądanie klienta początkowopojawia się tutaj.

`getTitles(res);` - kontrola zostaje przekazana do funkcji `getTitles()`.

`Function getTitles(res) {` - funkcja `getTitles()` pobiera sentencje i przekazuje kontrolę funkcji `getTemplate()`.

`Function getTemplate(titles, res) {` - funkcja `getTemplate()` odczytuje plik szablonu I przekazuje kontrolę funkcji `formatHtml()`.

`Function formatHtml(titles, tmpl, res) {` - funkcja `formatHtml()`generuje odpowiedź i przekazuje ją klientowi.

`Function hadError(err, res) {` - jeżeli w trakcie całego procesu wystąpi jakikolwiek błąd, funkcja `hadError()` zarejestruje go i wyświetli klientowi odpowiedni komunikat.

Istnieje możliwość dalszej redukcji zagnieżdżenia spowodowanego przez bloki `if-else`, ale wymaga to użycia innej możliwości programowania w Node.js, jaką jestwczesne zakończenie działania funkcji:

```
function getTitles(res) {
fs.readFile('./titles.json', function (err, data) {
if (err) return hadError(err, res);
getTemplate(JSON.parse(data.toString()), res);
})
});
```

Zamiast tworzyć wiele bloków else, następujezakończenie działania funkcji z powodu wystąpienia błędu. W takim przypadku nie mapotrzeby kontynuacji wykonywania tej funkcji.

ZADANIE: Zmodyfikuj wszystkie funkcje z możliwością ich wczesnego zakończenia.

ZADANIE 2. Emiter zdarzeń.

Emiter zdarzeń wyzwala zdarzenia i ma możliwość ich obsługi, gdy zostanąwywołane. Pewne ważne komponenty API Node.js, na przykład serwery HTTP,TCP i strumienie, są zaimplementowane jako **emitery zdarzeń**. Masz równieżmożliwość tworzenia własnych emiterów zdarzeń.Jak wcześniej wspomniano, zdarzenia są obsługiwane dzięki zastosowaniunasłuchiwania zdarzeń. Wspomniane nasłuchiwanie to po prostu połączeniezdarzenia z funkcją wywołania zwrotnego, która jest wykonywana po każdymwyzwoleniu zdarzenia.

Utworzymy klasę o nazwie `Watcher` przeznaczoną do przetwarzania plikówumieszczonych we wskazanym katalogu systemu plików. Tego rodzaju narzędzie może na przykład zmienićwielkość liter na małe w każdym pliku umieszczonym w katalogu, a późniejskopiować go do zupełnie innego katalogu.Mamy trzy kroki prowadzące do budowy emitera zdarzeń:

1. Utworzenie konstruktora klasy.
2. Dziedziczenie zachowania po emiterze zdarzeń.
3. Rozbudowa zachowania.

Utwórz w projekcie dwa nowe katalogi:

```
watch
done
```

W katalogu `watch` umieść kilka plików tekstowych - ich nazwy napisz **dużymi literami.**

Kod serwera:

serwer.js

```
function Watcher(watchDir, processedDir) {
  this.watchDir = watchDir;
  this.processedDir = processedDir;
}

var events = require('events')
var util = require('util');
util.inherits(Watcher, events.EventEmitter);

var fs = require('fs')
var watchDir = './watch'
var processedDir = './done';

Watcher.prototype.watch = function() {
  var watcher = this;
  fs.readdir(this.watchDir, function(err, files) {
    if (err) throw err;
    for(var index in files) {
      watcher.emit('process', files[index]);
    }
  })
}

Watcher.prototype.start = function() {
  var watcher = this;
  fs.watchFile(watchDir, function() {
    watcher.watch();
  });
}

var watcher = new Watcher(watchDir, processedDir);

watcher.on('process', function process(file) {
  var watchFile = this.watchDir + '/' + file;
  var processedFile = this.processedDir + '/' + file.toLowerCase();
  fs.rename(watchFile, processedFile, function(err) {
    if (err) throw err;
  });
});

watcher.start();
```

Wyjaśnieniekodu:

`functionWatcher(watchDir, processedDir) {` - konstruktordlanaszejklasy Watcher. Jako argumenty konstruktor pobiera nazwę monitorowanegokatalogu oraz nazwę katalogu, w którym mają być umieszczane zmodyfikowanepliki.

```
var events = require('events')
varutil = require('util');
```

`util.inherits(Watcher, events.EventEmitter);-` logika dziedzicząca zachowanie po emiterze zdarzeń. Zwróć uwagę na użycie funkcji `inherits()` stanowiącej część wbudowanego wNode.js modułu `util`. Wymieniona funkcja stanowi elegancki sposób pozwalającyobiektowi na dziedziczenie zachowania po innym obiekcie.Wywołanie funkcji `inherits()` w powyższym fragmencie kodu jest odpowiednikiemnastępującego wywołania w JavaScript:

```
Watcher.prototype = new events.EventEmitter();
```

Po skonfigurowaniu obiektu `Watcher` konieczne jest rozbudowanie metododziedziczonych po klasie `EventEmitter` o dwie nowe: `watch`i`start`.

Metoda `watch()` przeprowadza iterację przez elementy obserwowanego katalogu i przetwarza każdy napotkany plik. Metoda `start()` rozpoczyna monitorowaniekatalogu. Podczas monitorowania wykorzystywana jest funkcja Node o nazwie`fs.watchFile()`. Gdy cokolwiek pojawi się w obserwowanym katalogu, następujewywołanie metody `watch()`, przeprowadzenie iteracji przez katalog iwyemitowanie zdarzenia `process` dla każdego znalezionego pliku.Po zdefiniowaniu klasy `Watcher` wykorzystujemy ją do pracy, tworząc obiekt`Watcher` za pomocą poniższego kodu:

```
var watcher = new Watcher(watchDir, processedDir);
```

Mając nowo utworzony obiekt `Watcher`, używamy metody `on` dziedziczonej poklasie emitera zdarzeń do zdefiniowania logiki odpowiedzialnej za przetwarzanie każdego pliku:

```
watcher.on('process', function process(file) {…
```

Gdy cała logika znajduje się już na swoim miejscu, rozpoczęcie monitorowaniakatalogu następuje za pomocą następującego kodu:

```
watcher.start();
```

Pliki z katalogu `watch` zostaną przeniesione do katalogu `done`, a ich nazwa zostanie zmieniona. Zmień ponownie nazwy plików na duże litery i przerzuć je do katalogu `watch`. Zaobserwuj działanie.

ZZIPUJ PROJEKT I UMIEŚĆ GO NA PLATFORMIE MOODLE.