REST, Mikroserwisy

Tomasz Zawadzki

```
[student@localhost ~]$ node --version
v9.3.0
[student@localhost ~]$ npm --version
5.5.1
```

Etap 1: Katalog 01_HttpServer

```
[student@localhost take-a-rest]$ npm run 01_HttpServer
```

- Posługując się drugim terminalem przy pomocy komendy `curl` z parametrem `-X` wysłać żądanie wykonania metody GET dla głównego URLa serwisu a następnie dla podstrony `/hello` (należy łączyć się na port 3000).

```
[student@localhost ~]$ curl -X GET "127.0.0.1:3000/hello"
<p>Anonymous message: Oh, Hi Mark!</p>
```

- Zaobserwować wyprowadzane przez serwer komunikaty i przeanalizować kod aplikacji; sprawdzić która część kodu jest odpowiedzialna za raporty i zmienić ją tak, aby raportowała również czas obsługiwanego wywołania (metoda `Date.now()` w JavaScript).

```
// GET /hello -- Show message
app.get("/hello", function(request, response) {
  printReqSummary(request);
  response.send("<p>Anonymous message: Oh, Hi Mark!</p>");
  console.log(Date.now());
});
```

```
Handling GET /hello
1579103422357
Handling GET /hello
1579103423710
Handling GET /hello
1579103424941
```

- Dodać obsługę metody GET dla ścieżki URL `/time`, która zwróci aktualny czas.

```
// GET /time -- Show time
app.get("/time", function(request, response) {
  printReqSummary(request);
  response.send(String(Date.now()));
});
```

```
[student@localhost ~]$ curl -X GET "127.0.0.1:3000/time"
1579103537518
```

Etap 2: Katalog 02_UrlParameters

```
[student@localhost take-a-rest]$ npm run 02_UrlParameters
```

- Wyprowadzić (najlepiej w przeglądarce) i przeanalizować wynik metody GET dla bazowego URLa i innych wariantów wywołań.

http://127.0.0.1:3000/hello/Tomek
Normal message for: Tomek

http://127.0.0.1:3000/hello/Tomek/Zawadzki
Special message for: Tomek Zawadzki (age: undefined years, height: undefined cm)

http://127.0.0.1:3000/hello/Tomek/Zawadzki?age=21
Special message for: Tomek Zawadzki (age: 21 years, height: undefined cm)

http://127.0.0.1:3000/hello/Tomek/Zawadzki?height=182
Special message for: Tomek Zawadzki (age: undefined years, height: 182 cm)

http://127.0.0.1:3000/hello/Tomek/Zawadzki?age=21&height=182
Special message for: Tomek Zawadzki (age: 21 years, height: 182 cm)

- Dodać obsługę metody GET dla ścieżki URLa składającego się z trzech parametrów, która losowo zwraca jedną z części URLa (tj. jeden z parametrów).

```
function getRandomInt(min, max) {
  return Math.floor(Math.random() * (max - min + 1)) + min;
}

// GET /hello/:name -- Show normal message for a named person
app.get("/hello/:name/:surname/:city", function(request, response) {
  printReqSummary(request);
  // Grab URL parameters from `request.params` object
  const params = [request.params.name, request.params.surname, request.params.city];
  const randomParam = params[getRandomInt(0, params.length-1)];
  response.send(`<p>Random param: ${randomParam}</p>`);
});
```

http://127.0.0.1:3000/hello/Tomek/Zawadzki/Krakow
Random param: Zawadzki

Random param: Krakow

Random param: Tomek

Etap 3: Katalog 03_HttpMethods

```
[student@localhost take-a-rest]$ npm run 03_HttpMethods
```

- Wyprowadzić i przeanalizować wynik metody GET dla bazowego URLa i innych wariantów wywołań.

http://127.0.0.1:3000/

HTTP Methods

• Show items (GET /item)

• Add an item (PUT /item/:name)

• Remove an item (DELETE /item/:name)

http://127.0.0.1:3000/item

Available items:

- Zmodyfikować kod, tak aby dodawanie nowego elementu odbywało się przy pomocy metody HTTP POST, a modyfikacja – przy pomocy metody PUT (nową nazwę dla elementu należy podawać przy pomocy kwerendy).

```
/* POST /item/:name -- add (put) new item to the collection */
app.post("/item/:name", function(request, response) {
  printReqSummary(request);
  const itemName = request.params.name;
  /* Is the item in collection? */
  if (items.includes(itemName)) {
    response.send(`<p>Item "${itemName}" already in collection</p>`);
  } else {
    items.push(itemName);
    response.send(`<p>Item "${itemName}" added successfully</p>`);
  }
});

/* PUT /item/:oldName?newName=abc -- rename item existing in the collection */
app.put("/item/:oldName", function(request, response) {
  printReqSummary(request);
  const oldItemName = request.params.name;
  const newItemName = request.query.name;
  if (newItemName === undefined) {
    response.status(400).send(`<p>New item name not specified</p>`);
    return;
  }
  if (!items.includes(oldItemName)) {
    response.status(404).send(`<p>Item "${oldItemName}" doesn't exists</p>`);
  }
  items[items.indexOf(oldItemName)] = newItemName;
  response.send(`<p>Item "${oldItemName}" successfully renamed to "${newItemName}"</p>`);
});
```

Na początku lista przedmiotów jest pusta:

```
[student@localhost ~]$ curl -X GET "127.0.0.1:3000/item"
<p>Available items: </p>
```

Dodanie nowego przedmiotu:

```
[student@localhost ~]$ curl -X POST "127.0.0.1:3000/item/apple"
<p>Item "apple" added successfully</p>
```

Lista przedmiotów:

```
[student@localhost ~]$ curl -X GET "127.0.0.1:3000/item"
<p>Available items: apple</p>
```

Dodanie przedmiotu, który istnieje w kolekcji:

```
[student@localhost ~]$ curl -X POST "127.0.0.1:3000/item/apple"
<p>Item "apple" already in collection</p>
```

Dodanie kolejnego przedmiotu:

```
[student@localhost ~]$ curl -X POST "127.0.0.1:3000/item/banana"
<p>Item "banana" added successfully</p>
```

Lista przedmiotów:

```
[student@localhost ~]$ curl -X GET "127.0.0.1:3000/item"
<p>Available items: apple,banana</p>
```

Dodanie przedmiotu, który istnieje w kolekcji:

```
[student@localhost ~]$ curl -X POST "127.0.0.1:3000/item/banana"
<p>Item "banana" already in collection</p>
```

Aktualizacja nazwy przedmiotu bez podania nowej nazwy:

```
[student@localhost ~]$ curl -X PUT "127.0.0.1:3000/item/banana"
<p>New item name not specified</p>
```

Aktualizacja nazwy przedmiotu z podaniem nowej nazwy:

```
[student@localhost ~]$ curl -X PUT "127.0.0.1:3000/item/banana?newName=pear"
<p>Item "banana" successfully renamed to "pear"</p>
```

Lista przedmiotów:

```
[student@localhost ~]$ curl -X GET "127.0.0.1:3000/item"
<p>Available items: apple,pear</p>
```

Usunięcie przedmiotu, który nie istnieje w kolekcji:

```
[student@localhost ~]$ curl -X DELETE "127.0.0.1:3000/item/lemon"
<p>Item "lemon" doesn’t exists</p>
```

Usunięcie przedmiotu, który znajduje się w kolekcji:

```
[student@localhost ~]$ curl -X DELETE "127.0.0.1:3000/item/apple"
<p>Item "apple" removed successfully</p>
```

Ponowna próba usunięcia tego przedmiotu:

```
[student@localhost ~]$ curl -X DELETE "127.0.0.1:3000/item/apple"
<p>Item "apple" doesn’t exists</p>
```

Lista przedmiotów:

```
[student@localhost ~]$ curl -X GET "127.0.0.1:3000/item"
<p>Available items: pear</p>
```

Etap 4: Katalog 04_RestDatabase

```
[student@localhost take-a-rest]$ npm run 04_RestDatabase
```

- Wyprowadzić i przeanalizować wynik metody GET dla bazowego URLa.

http://127.0.0.1:3000/

REST + Database

• Show all patients (GET /patient)
• Show specific patient (GET /patient/:id)
• Add new patient (POST /patient?name=:NAME&surname=:SURNAME)
• Modify existing patient (PUT /patient/:id?name=:NAME&surname=:SURNAME)
• Remove patient (DELETE /patient/:id)

```
[student@localhost ~]$ curl -X GET "127.0.0.1:3000"
<h1>REST + Database</h1><ul>
<li>Show all patients (GET /patient )</li>
<li>Show specific patient (GET /patient/:id)</li>
<li>Add new patient (POST /patient?name=:NAME&surname=:SURNAME)</li>
<li>Modify existing patient (PUT /patient/:id?name=:NAME&surname=:SURNAME)</li>
<li>Remove patient (DELETE /patient/:id)</li></ul>
```

- Zaobserwować rezultaty dla URLa zawierającego numer (id) pacjenta w zależności od użytej metody HTTP.

Na początku lista pacjentów jest pusta:

```
[student@localhost ~]$ curl -X GET "127.0.0.1:3000/patient"
{"error":"No patients are registered"}
```

Pacjent o identyfikatorze 1 nie istnieje, więc zapytanie o ten zasób zakończy się niepowodzeniem:

```
[student@localhost ~]$ curl -X GET "127.0.0.1:3000/patient/1"
{"error":"No patients are registered"}
```

Również próba aktualizacji tego zasobu zakończy się niepowodzeniem:

```
[student@localhost ~]$ curl -X PUT "127.0.0.1:3000/patient/1"
{"error":"No patient with given id"}
```

Próba usunięcia tego zasobu również zakończy się niepowodzeniem:

```
student@localhost ~]$ curl -X DELETE "127.0.0.1:3000/patient/1"
{"error":"No patient with given id"}
```

Dodanie nowego pacjenta:

```
[student@localhost ~]$ curl -X POST "127.0.0.1:3000/patient?
name=Jan&surname=Kowalski"
{"id":1,"name":"Jan","surname":"Kowalski"}
```

Lista pacjentów:

```
[student@localhost ~]$ curl -X GET "127.0.0.1:3000/patient"
[{"id":1,"name":"Jan","surname":"Kowalski"}]
```

Informacje o pacjencie:

```
[student@localhost ~]$ curl -X GET "127.0.0.1:3000/patient/1"
{"id":1,"name":"Jan","surname":"Kowalski"}
```

Modyfikacja danych pacjenta:

```
[student@localhost ~]$ curl -X PUT "127.0.0.1:3000/patient/1?
name=Jan&surname=Nowak"
{"id":1,"name":"Jan","surname":"Nowak"}
```

Usunięcie pacjenta z listy:

```
[student@localhost ~]$ curl -X DELETE "127.0.0.1:3000/patient/1"
{"message":"Patient removed successfully"}
```

Lista pacjentów:

```
[student@localhost ~]$ curl -X GET "127.0.0.1:3000/patient"
{"error":"No patients are registered"}
```

Próba ponownego usunięcia pacjenta z listy zakończy się niepowodzeniem:

```
[student@localhost ~]$ curl -X DELETE "127.0.0.1:3000/patient/1"
{"error":"No patient with given id"}
```

- Przeanalizować różnice w logice poszczególnych implementacji kodu dla obsługi tych metod.

Należy zwrócić uwagę, że w przeciwieństwie do poprzednich przykładów (01_HttpServer, 02_UrlParameters, 03_HttpMethods), serwer zwraca dane w postaci JSON.

```
response.status(200).send(JSON.stringify(patients));
```

Dodatkowo, wykorzystywane są kody odpowiedzi HTTP, np. **404 Not Found** w przypadku, gdy pacjent o podanym identyfikatorze nie istnieje:

```
response.status(404).send({ error: "No patient with given id" });
```

W przypadku błędnego żądania, serwer zwróci odpowiedź z kodem **400 Bad Request**.

```
response.status(400).send({
  error: "Invalid request - missing queries (name and/or surname)"
});
```

W przypadku powodzenia, serwer zwraca odpowiedź z kodem **200 OK**.

```
response.status(200).send({ message: "Patient removed successfully" });
```

- Uwaga: Zwrócić uwagę na fragment kodu pomiędzy db i średnikiem, który odwołuje się do kodu pakietu lowdb, zainicjowanego na początku.

```
// Load database related modules
const low = require("lowdb");
const FileSync = require("lowdb/adapters/FileSync");
// Store database in db.json file
const adapter = new FileSync("04_RestDatabase/db.json");
const db = low(adapter);
```

Pakiet `lowdb` wraz z adapterem `FileSync` umożliwiają korzystanie z bazy danych przechowywanej w pliku w formacie JSON. Oto przykładowe wywołanie, które jest odpowiedzialne za aktualizację danych pacjenta:

```
db.get("patients")
  .find(patient)
  .assign(updatedPatient)
  .write();
```