Programowanie Komunikacji Internetowej

Laboratorium 5 **dr hab. inż. Radosław Wajman**

Aplikacje internetowe w Node.JS

Cel zajęć

Do realizacji ćwiczeń potrzebne jest środowisko programistyczne np. Visual Studio Code lub inne wspierające składnię języka JavaScript, oraz serwer Node.JS.
Celem ćwiczenia jest przedstawienie podstawowej technologii platformy Node.JS jako środowiska uruchomieniowego aplikacji typu server-side (back-end) do projektowania dynamicznych stron internetowych w języku JS. We współczesnych aplikacjach JS opartych o szablon projektowy np. Express, Node.js pełni funkcję kontrolera odpowiadając za wstępne przetworzenie żądania, wywołanie logiki biznesowej i wybór strony widoku.

Zadania ćwiczeniowe

Zadanie 1 – Instalacja Node.JS
Zainstaluj środowisko Visual Studio Code (lub inne wspierające składnie języka JS).
Następnie zgodnie z tym poradnikiem zainstaluj środowisko Node JS. Postępuj jedynie do rozdziału „*Korzystanie z NPM*” włącznie.

Program 1 – Instalacja Node.JS
W nowym folderze (tj. dla nowego projektu) postępuj zgodnie z tym poradnikiem. Efektem prac w ramach tego ćwiczenia powinna być aplikacja internetowa zorganizowana w ramach szablonu silnika PUG z włączonym automatycznym restartowaniem serwera po wykrytej modyfikacji plików projektu. Nie jesteś zmuszony do korzystania z PUG. Jednakże dalsze działania opisane w instrukcji na nim się wzorują. Z poradnika zapoznaj się z informacjami dotyczącymi struktury utworzonego projektu, a w szczególności pliki www, app.js oraz definiowanie ścieżek (routes) oraz widoków.
Wykonaj również samodzielnie ostatnie ćwiczenie sprawdzające poziom zrozumienia tematu i utwórz nową ścieżkę w /routes/users.js, która wyświetli tekst „*You are cool*” pod adresem URL /users/cool/.
Uwaga!
Jeśli w trakcie realizacji ćwiczenia napotkasz na błąd związany z niepoprawnym podpisaniem skryptu, przeprowadź działania opisane tutaj.

Program 3 – Odczytanie parametrów żądania otrzymanego z przeglądarki
a) Zmodyfikuj ścieżkę index z punktem „/” wygenerowaną automatycznie razem z szablonem w poprzednim zadaniu, tak, aby wyświetlała ona w oknie przeglądarki informacje żądania poprzez wywołanie metod obiektu req. Wypisz zatem:

- typ żądania (GET, POST, HEAD, itp.),
- adres IP przeglądarki (zdalnego klienta),
- adres domenowy serwera.

- dowolną wartość jednego z pól żądania: „Accept”, „Accept-Language”, „Accept-Encoding”, “Host” i „ User-Agent”. Zobacz też nagłówek żądania.

Przykładowy wynik działania poprawnie wykonanego ćwiczenia przedstawiono poniżej:

```
wynik getMethod(): GET
wynik getRemoteAddr(): 127.0.0.1
wynik getServerName(): localhost
wynik getHeader("Accept"): text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
wynik getHeader("Accept-Language"): pl,en-us;q=0.7,en;q=0.3
wynik getHeader("Accept-Encoding"): gzip, deflate
wynik getHeader("User-Agent"): Mozilla/5.0 (Windows NT 6.0; rv:7.0.1) Gecko/20100101 Firefox/7.0.1
```

b) Zmodyfikuj adres URL wpisany do przeglądarki w ten sposób, aby podawał dwa parametry oraz ich wartości: imie i wiek. Postępuj zgodnie z opisem w rozdziale „Route parameters” tego poradnika.
Dla przykładowego adresu, używanego w niniejszym ćwiczeniu wyglądałoby to następująco:
http://localhost:3000/Radek/30
Następnie, wzorując się na poprzednim ćwiczeniu, wykorzystaj kolejną składową param obiektu req, aby odczytać i wyświetlić wartości parametrów podanych w URL.

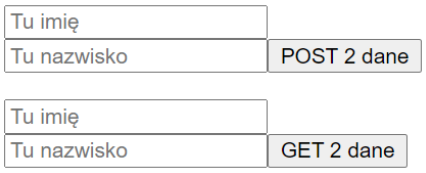
```
{"imie":"Radek","wiek":"30"}
```

c) Zbuduj nową ścieżkę o nazwie np. **dane** (nowy plik *dane.js*), w której zaprojektuj dwa punkty:

- POST (‘/’), w którym wypisz otrzymane parametry składowej req.param
- GET (‘/form’), w którym wypisz otrzymane parametry składowej req.query

Pamiętaj o definicji nowej ścieżki w *app.js*. Postępuj zgodnie z poleceniami w rozdz. „Defining and using seperate routs module” w tym poradniku.
Aby ścieżki działały poprawnie należy utworzyć nowy widok w view. Należy go wyświetlić na żądanie GET(‘/’). Widok niech zawiera dwa formularze (jedne dla post i drugi dla get) np.:

```
block content
  form(id="dane_post" action="/dane", method="post")
    input(type="text", name="imie", value="", placeholder="Tu imię")
    br
    input(type="text", name="nazwisko", value="", placeholder="Tu nazwisko")
    button(type="submit") POST 2 dane
  br
  form(id="dane_get" action="/dane/form", method="get")
    input(type="text", name="imie", value="", placeholder="Tu imię")
    br
    input(type="text", name="nazwisko", value="", placeholder="Tu nazwisko")
    button(type="submit") GET 2 dane
```



Składnia zapisu widoku w ramach pliku PUG jest opisana w dokumentacji pod tym adresem.
Efektem końcowym powinno być takie działanie, kiedy to po naciśnięciu dowolnego z przycisków dane z odpowiedniego formularza zostaną przesłane i prawidłowo wyświetlone.

Program 2 - Zasięg widoczności zmiennych
Utwórz trzy zmienne:

- lokalną o nazwie `lokalna_zmienna` np. w ścieżce *index.js* w miejscu zaraz po deklaracji zmiennej `router`,
- globalną w *app.js* jako pole `app.locals.ii`,
- kolejną globalną w *app.js* jako pole `global.ii`.

Definicje obu zmiennych globalnych najlepiej umieścić zaraz po definicji zmiennej `app`.

Pozostaje teraz modyfikacja w ścieżkach index oraz dane. Przykład realizacji zostanie zademonstrowany na podstawie jednej ze ścieżek. W pierwszej kolejności powinno się zmodyfikować widok dodając paragraf wyświetlający przekazane wartości:

```
p Local (route-scope) variable value is #{local_var}
p Global (app-scope: global.ii) variable value is #{global_var1}
p Global (app-scope: app.local.ii) variable value is #{global_var2}
```

Dodatkowo, wartości tych zmiennych powinny być przekazane ze ścieżki do widoku w momencie wywołania (renderowania) widoku. Proszę pamiętać, aby wcześniej zwiększyć wartości wszystkich zmiennych. Można to zrealizować np.:

```
router.get('/', function(req, res, next) {
  i++;
  req.app.locals.ii++;  //dla zmiennej deklarowanej jako app.locals.ii;
  ii++;                 //dla zmiennej deklarowanej jako global.ii;
  res.render('index', { title: 'Express by RW',
                        local_var: i,
                        global_var1: ii,
                        global_var2: req.app.locals.ii });
});
```

Efekt działania modyfikacji w programie.

Informacja wyświetlana po wielokrotnym wywołaniu żądania na punkt '/' w ścieżce dane.js.	Global (app-scope: global.ii) variable value is 20 Global (app-scope: app.local.ii) variable value is 20
Informacja wyświetlana po wielokrotnym wywołaniu żądania na punkt '/' w ścieżce index.js.	Local (route-scope) variable value is 9 Global (app-scope: global.ii) variable value is 21 Global (app-scope: app.local.ii) variable value is 21

Program 3 – Formularz logowania

Wprowadzimy prosty mechanizm autentykacji użytkownika. Pomijając fakt braku bezpieczeństwa przy przesyłaniu hasła otwartym tekstem, w tym zadaniu nauczymy się tworzyć formularze i przełączać się pomiędzy ścieżkami.

W głównej ścieżce *index.js* zaprojektuj dwa klawisze „Sign In" oraz „Sign Out". Pierwszy z nich powinien przekierować na nową ścieżkę np. *login.js* wraz z widokiem *login.pug*. Strona logowania powinna umożliwić użytkownikowi wprowadzenie nazwy użytkownika i hasła. Możesz wzorować się na tym poradniku. Inne, ciekawe rozwiązanie oparte na porównywaniu wprowadzanych danych z tymi zapisanymi w bazie danych znajduje się w tym poradniku. Ostatecznie, w wyniku poprawnego zwalidowania wprowadzonych danych, aplikacja powinna przejść do strony początkowej i wyświetlić nazwę zalogowanego użytkownika.

Użycie klawisza wylogowania powinno odświeżyć stronę, a nazwę użytkownika pozostawić pustą.

W formularzu logowania podaj błędną nazwę użytkownika albo hasło. Czy po wciśnięciu zaloguj pojawiła się informacja o błędzie? Zamiast `res.render('error');` wypróbuj też innego sposobu wyrzucania błędu korzystając z wygenerowanego mechanizmu Express np.

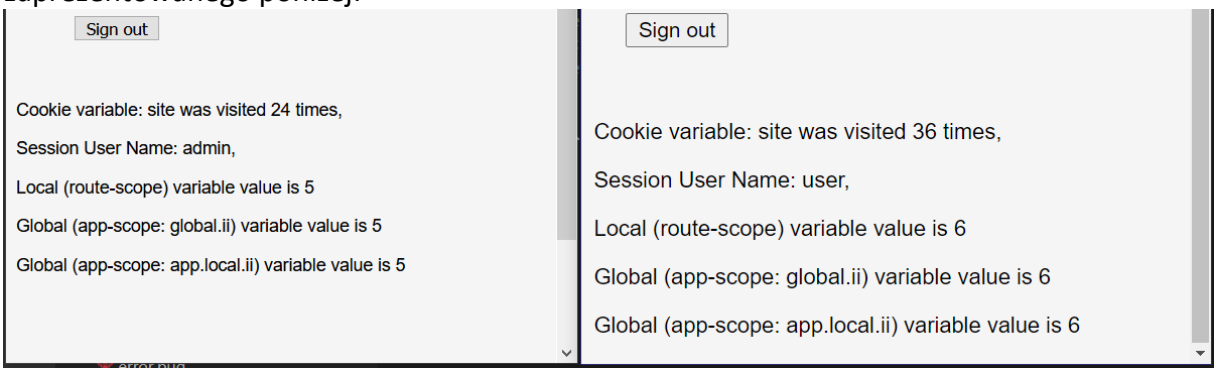
```
next(createError(403));
```

lub

```
throw new Error('Not logged in');
```

Przyjemnym rozwiązaniem byłoby ukrycie klawisza „Sign Out”, gdy żaden użytkownik nie jest zalogowany i podobnie ukrycie klawisza „Sign In”, w przypadku poprawnego zalogowania. Liczę tu na samodzielną realizację.
Czy do przechowywania nazwy zalogowanego użytkownika utworzyłeś zmienną widoczną dla całej ścieżki? Jeśli tak, to może to prowadzić, do ciekawej sytuacji. Chwilowo pozostań zalogowany. Otwórz zupełnie inną przeglądarkę i przywołaj swoją aplikację. Zauważ, że w innej przeglądarce aplikacja wskazuje, że ten sam użytkownik jest wciąż zalogowany. Aplikacja nie rozróżnia sesji użytkownika. Zmienimy to w kolejnym ćwiczeniu.

Program 4 – Sesje użytkowników i ciasteczka
Sesja jest mechanizmem zapewniającym komunikację (przekazywanie obiektów) pomiędzy kolejnymi żądaniami przychodzącymi z tego samego okna przeglądarki. Sesję można traktować jako „worek” skojarzony z konkretnym oknem przeglądarki, do którego wrzucamy, i z którego wyciągamy, obiekty oznaczone odpowiednimi etykietami. Poniższe ćwiczenie demonstruje, jak można użyć mechanizmu sesji do implementacji mechanizmu logowania i wylogowywania oraz do rozróżniania użytkowników w zależności od tego, gdzie uruchamiają naszą aplikacje.
Poniżej dwa ciekawe poradniki realizujące identyfikacje sesji z wykorzystaniem dwóch różnych mechanizmów:
http://expressjs.com/en/resources/middleware/cookie-session.html
https://www.tutorialspoint.com/expressjs/expressjs_sessions.htm
Zachęcam do wypróbowania obu rozwiązań implementując zaproponowane tam przykłady. W ramach swojej aplikacji zmodyfikuj widok index.pug tak, aby wyświetlał np. informacje o krotności odwiedzenia aplikacji w ramach danej sesji oraz spróbuj przyporządkować nazwę zalogowanego użytkownika do sesji. Efekt działania aplikacji powinien zmierzać do tego, zaprezentowanego poniżej:



Pokazany jest tu efekt działania na dwóch różnych przeglądarkach, z których przywoływano tę samą aplikacje równolegle (w tym samym czasie). Wprowadzone uprzednio zmienne tj. lokalna ścieżki oraz dwie globalne wskazują te same wartości (rozbieżność wynika z potrzeby kolejnego odświeżenia strony). Widać natomiast, że nazwa użytkownika sesji jest inna oraz na różnych przeglądarkach strona była odświeżana inną liczbę razy.

Program 5 - Chat

Korzystając z mechanizmów wypracowanych w swojej aplikacji w ramach wcześniejszych zadań zaprojektuj i zaimplementuj usługę prostego czatu między dwiema przeglądarkami, w których są zalogowani różni użytkownicy.

Zaimplementuj mechanizm kolekcji do przechowywania wysłanych komunikatów i ich autorów. Uwzględnij mechanizm auto-odświeżania strony. Wprowadź następującą modyfikację do widoku strony głównej.

```
doctype html
html
  head
    title #{title}
    link(rel="stylesheet" href="/stylesheets/style.css")
    meta(http-equiv="Refresh" CONTENT="10")
  body
    h1= title
```

Program 6 – Wysyłanie i odbieranie plików

Kolejne zadanie wymaga całkowicie własnego pomysłu i implementacji. W tym celu spróbuj wyszukać podobnych rozwiązań.

Zaprogramuj mechanizm wysyłania pliku do serwera oraz jego pobierania. Zweryfikuj, czy otrzymałeś tę samą zawartość pliku.