Zaawansowane Programowanie Webowe

Laboratorium nr 9

Celem laboratorium jest zapoznanie z dalszymi możliwościami JavaScript po stronie serwera poprzez wykorzystanie platformy NodeJS oraz dostarczanych przez nią modułów. W ramach ćwiczenia laboratoryjnego należy rozszerzyć istniejący serwer aplikacyjny wykonany w ramach lab 8 o możliwość obsługi komunikacji asynchronicznej wzbudzanej ze strony serwera.

Tematy zadań:

Na laboratorium wykorzystujemy aplikację sklepu internetowego stworzonego w ramach zajęć laboratorium nr 8. Po zakończeniu lab 8 posiadamy prawie kompletna aplikacje do obsługi sklepu opartą o własnoręcznie zrealizowany serwer aplikacyjny.

Chcielibyśmy jednak rozszerzyć funkcjonalność naszego sklepu o możliwość wprowadzania promocji czasowych dla naszych produktów oraz powiadamiania aktywnych użytkowników o zmianach w ofercie produktów (dodawanie, modyfikacja opisu) wykonanych w tym samym czasie przez admina. Obsługa promocji działała by w ten sposób, że admin mógłby za pomocą modułu zarządzającego określać czasową promocje dla wybranego produktu/produktów. Byłby to rodzaj tzw. last minute.
Opis szczegółowy poniżej.

Scenariusz realizacji wyglądałbym następująco:
Admin zaznacza intersujący go produkt lub produkty. Następnie określa poziom obniżki (procentowy) oraz definiuje czas trwania promocji. Promocje sa z reguły krótkotrwałe np. 5, 10 czy 20 minut. Po zatwierdzeniu promocja powinna od razu obowiązywać. Nowi użytkownicy po wejściu na stronę powinni zobaczyć odpowiednie informacje przy każdym produkcie, który podlega promocji. Problemem są użytkownicy którzy w czasie zatwierdzania promocji byli już użytkownikami naszej aplikacji. Aplikacja jest wykonana w trybie SPA więc nie wymaga przeładowania. Aby również oni mogli skorzystać z promocji powinni być o niej powiadamiani. Powiadamianie musi wyjść ze strony serwera. W tym celu proponuje wykorzystać technologie WebSocket, opisaną w sekcji poniżej. Po zakończeniu czasu promocji na stronie powinna pojawić się informacja o zakończeniu promocji oraz powrót do ceny standardowej.

Podobny mechanizm można wykorzystać również do obsługi wyczerpania stanu magazynowego, któregoś z produktów. Zakup ostatniego produktu powinien skutkować wyświetleniem odpowiedniej informacji (np. brak produktu lub coś podobnego) u wszystkich aktywnych użytkowników.

Wykorzystując Socket.io można rozszerzyć funkcjonalność serwera http o wysyłanie powiadomień o zmianie (nowy produkt lub edycja istniejącego) w bazie produktów

wykonanej przez Admina. Informacja ma być wyświetlana natychmiast u wszystkich aktywnych klientów na ekranie – forma wyświetlania dowolna.

Uwaga. Preferowane jest rozwiązane asynchroniczne oparte na Socket.io. Dużo gorzej punktowane będzie rozwiązania na symulacji asynchronizmu poprzez sekwencyjne odpytywanie o stan serwera poprzez wywołania AJAX tzw. long pooling.

Wykorzystanie Web Socket do powiadamiania (komunikacja natychmiastowa (real-time))

Pakiet NodeJS posiada w swoich zasobach moduł Socket.io. Biblioteka Socket.IO pozwala na tworzenie działających w czasie rzeczywistym aplikacji sieciowych opartych na dwukierunkowej komunikacji między serwerem i klientem. W najprostszej postaci biblioteka Socket.IO ma API bardzo podobne do WebSocket API (http://www.websocket.org/) oraz kilka wbudowanych rozwiązań zapasowych dla starszych przeglądarek internetowych, które nie obsługują nowych funkcji.
Socket.IO zapewnia wygodne API przeznaczone do rozgłaszania, wysyłania wiadomości itd. Oferowane funkcje powodują, że Socket.IO to bardzo popularna biblioteka dla gier działających w przeglądarkach internetowych, aplikacji czatu i aplikacji wykorzystujących strumienie.
Więcej informacji na temat biblioteki Socket.IO dostępne na stronach *http://socket.io/#how-to-use*.
http://socket.io/docs/
https://www.tutorialspoint.com/socket.io/index.htm

Przykładowa aplikacja wykorzystująca Socket.io w wersji minimalistycznej mogłaby np. wyświetlać w aplikacji klienckiej (przeglądarce) czas systemowy z serwera. Można to wykorzystać do obliczania różnicy czas pomiędzy klientem a serwerem.

Aby utworzyć aplikację, w pierwszej kolejności należy zainstalować Socket.IO za pomocą menedżera npm:

```
npm install socket.io
```

Przykład kodu pokazującego mechanizm komunikacji przy użyciu socket.io:

Kod serwera:
app.js

```
var express = require('express' );
var app = express();
var server = require('http' ).createServer(app);

var io = require('socket.io' )(server);
io.on('connection' , function(client) {
        console.log('Client connected...' );
```

```
        client.emit('messages' , { hello: 'test'  });     <- emitent danych

});
app.get('/' , function (req, res) {
    res.sendFile(__dirname + '/index.html' );
});
server.listen(8080);
```

Kod klienta:

Index.html

```
<script src="/socket.io/socket.io.js"></script>
 <script>
     var socket = io.connect('http://localhost:8080' );
     socket.on('messages' , function (data) {     <-  nasłuch emisji
                alert(data.hello);
         });
</script>
```

Uwaga.

`/socket.io/socket.io.js jest dostarczane bezpośrednio przez` Socket.io

Przykład zastosowanie:

Serwer Socket.IO o nazwie clock-server.js nieustannie przekazujący klientowi aktualny czas.

```
var app = require('http').createServer(handler);
var io = require('socket.io').listen(app);          <- Uaktualnienie zwykłego serwera HTTP
                                                       do serwera Socket.IO.
var fs = require('fs');
var html = fs.readFileSync('index.html', 'utf8');
function handler (req, res) {          <- kod serwera udostepnia zawsze kod index.html
 res.setHeader('Content-Type', 'text/html');
 res.setHeader('Content-Length', Buffer.byteLength(html, 'utf8'));
 res.end(html);
}
function tick () {
 var now = new Date().toUTCString();
 io.sockets.send(now);
```

```
}
setInterval(tick, 1000);
app.listen(8080);
```

Klient Socket.IO wyświetlający czas otrzymany z serwera

```
<!DOCTYPE html>
<html>
 <head>
 <script type="text/javascript" src="/socket.io/socket.io.js">
 </script>
 <script type="text/javascript">
 var socket = io.connect();
 socket.on('message', function (time) {
 document.getElementById('time').innerHTML = time;
 });
 </script>
 </head>
 <body>Aktualny czas w serwerze: <b><span id="time"></span></b>
 </body>
</html>
```

Uruchomienie aplikacji - > wywołania node clock-server.js, a zobaczysz komunikat info -socket.io started. Oznacza to, że biblioteka Socket.IO została skonfigurowana i jest gotowa do otrzymywania połączeń. Uruchom więc przeglądarkę internetową i przejdź pod adres URL http://localhost:8080/. Czas będzie uaktualniany co sekundę na podstawie komunikatu otrzymywanego z serwera.

Zadanie do wykonania dla zainteresowanych.

Napisz prosty czat oparty na socket.io pozwalający na komunikację pomiędzy użytkownikami znajdującymi się w tym samym pokoju tematycznym. Zrealizuj zadanie z wykorzystaniem serwera pośredniczącego.

Wymagania końcowe dotyczące aplikacji:

(wymagania częściowo były już definiowane we wcześniejszych materiałach lab – podaje w celu potwierdzenia)

1. Dodawanie nowych produktów powinno umożliwiać dodawanie również zdjęcia/zdjęć poglądowych dodawanego produktu.

2. Sekcja zamówień. Powinna składać się z dwóch części/sekcji – zamówień złożonych i zrealizowanych. Pracownik Obsługi po wyborze zamówienia może zmienić jego stan na zrealizowany i wtedy zamówienie powinno przenieść się do sekcji zrealizowanych.

3. Rozszerzenie funkcjonalności wyszukiwania produktów. Wprowadzenie dodatkowych filtrów:
Wyszukiwanie po cenie (przedziale cenowym);
Wyszukiwanie po fragmencie nazwy.

4. Możliwość wyświetlenia informacji szczegółowych o produkcie. Realizowane np. w formie okna modalnego wyświetlającego np. więcej zdjęć lub poszerzony opis produktu.

5. Extra wymaganie - Logowanie użytkowników. Oprócz użytkowników anonimowych wprowadzenie możliwości rejestracji i logowania użytkownika. Pozwoliłoby to pamiętać stan koszyka zakupów oraz wprowadzić dodatkową funkcjonalność np: przegląd historii zamówień.