Komunikacja w aplikacjach internetowych z wykorzystaniem web socketów na przykładzie node.JS

Nowoczesne aplikacje internetowe charakteryzują się komunikacją w czasie rzeczywistym. Takie rozwiązanie było niemożliwe do osiągnięcia znanymi dotychczas metodami komunikacji w architekturze klient – serwer. Zazwyczaj to klient odpytuje serwer w stałych odstępach czasu o potrzebne informacje. Powoduje to dodatkowy narzut związany z częstą komunikacją, a w rzeczywistości tylko zbliża o aplikacji czasu rzeczywistego.

Nowe podejście do tego problemy to komunikacja z wykorzystaniem web socketów. Podczas startu aplikacji nawiązywane jest dodatkowe połączenie TCP, które jest wykorzystywane do komunikacji dwukierunkowej, a więc to serwer może również wysłać informacje do klienta bez uprzedniego żądania od niego.

Poniżej został opisany sposób wykorzystania web sobcetów z wykorzystaniem biblioteki socket.IO oraz node.JS.

Stworzenie projektu w node.JS

Aby rozpocząć pracę nad projektem w node.JS należy z linii komend wpisać komendę npm init i uzupełnić informację opisujące projekt (nazwa, wersja, autor, itp.). W wyniku działania powyższej komendy powstanie plik package.json. Należy do niego dodać zależność do biblioteki socket.IO która będzie wykorzystana do komunikacji. Finalna wersja powinna być zbliżona do tej poniżej:

```
{
  "name": "socket",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "dependencies": {
    "express": "~4.10.2",
    "socket.io": "~1.3.7"
  },
  "author": "",
  "license": "ISC"
}
```

Kolejny krok to stworzenie serwera w pliku o nazwie wskazanej w sekcji main w package.json, czyli index.js

```
1  var express = require('express');
2  var app = express();
3  var http = require('http').Server(app);
4  var io = require('socket.io')(http);
5
6
7  io.on('connection', function(socket){
8    socket.on('eventName1', function(msg) {
9      console.log('Receive eventName with data: ' + msg);
10   })
11 });
12
13 setTimeout(function(){
14   io.emit('eventName2', "Message");
15 }, 5000);
16
17 app.get('/', function(req, res){
18   res.sendfile('index.html');
19 });
20
21 http.listen(3000, function(){
22   console.log('listening on *:3000');
23 });
```

Jak można zauważyć plik jest bardzo krótki. W linii numer 8 serwer nasłuchuje na zdarzenie o nazwie „eventName1” i w reakcji na jego otrzymanie wyświetla przesłaną wiadomość na konsoli. W linii numer 13 serwer po 5 sekundach wysyła wiadomość do klienta o treści „Message”.

Stworzenie części klienckiej

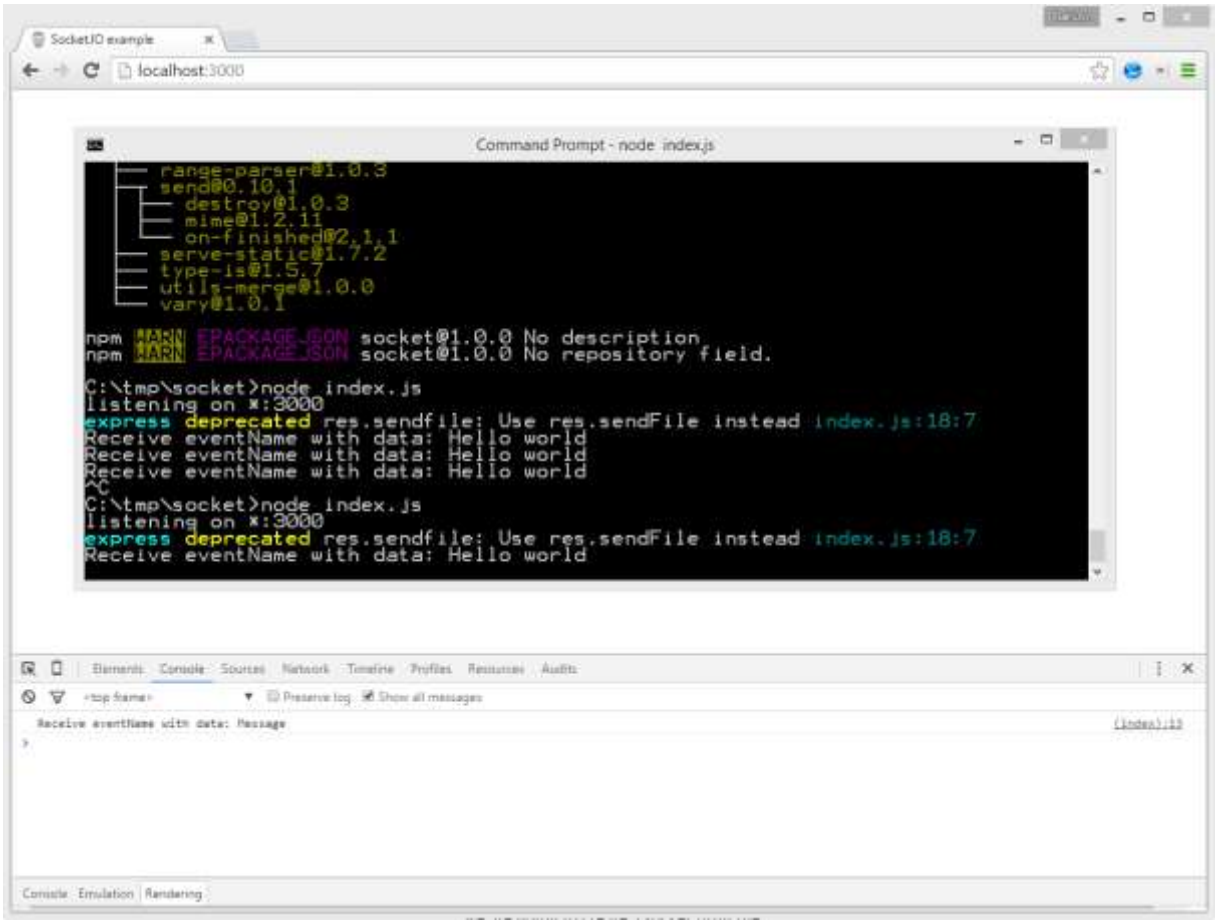
Klient używany do komunikacji z serwerem jest równie prost jak sam serwer. Należy jedynie obłużyć nadejście zdarzenia lub je wyemitować.

```
1  <!doctype html>
2  <html ng-app='WebTalk'>
3    <head>
4      <title>Socket.IO example</title>
5
6      <script src="https://cdn.socket.io/socket.io-1.2.0.js"></script>
7
8      <script>
9        var socket = io();
10       socket.emit('eventName1', "Hello world");
11
12       socket.on('eventName2', function(msg){
13         console.log('Receive eventName with data: ' + msg);
14       });
15     </script>
16   </head>
17   <body>
18   </body>
19 </html>
```

W linii 10 emitowane jest zdarzenie do serwera o treści „Hello world”, zaś w linii numer 12 klient nasłuchuje na zdarzenie „eventName2” i w przypadku jego nadejścia wyświetla jego treści na konsoli przeglądarki.

Podsumowanie

W kilkudziesięciu liniach kodu udało się stworzyć prosty serwer wraz z klientem i niezależną dwustronną komunikacją. Efekt końcowy został przedstawiony poniżej. Klient wysyła wiadomość do serwera o treści „Hello world”, zaś serwer do klienta „Message”.


SocketIO example
localhost:3000
Command Prompt - node index.js
range-parser@1.0.3
send@0.10.1
destroy@1.0.3
mime@1.2.11
on-finished@2.1.1
serve-static@1.7.2
type-is@1.5.7
utils-merge@1.0.0
vary@1.0.1
npm WARN EPACKAGE.JSON socket@1.0.0 No description
npm WARN EPACKAGE.JSON socket@1.0.0 No repository field.
C:\tmp\socket>node index.js
listening on *:3000
express deprecated res.sendfile: Use res.sendFile instead index.js:18:7
Receive eventName with data: Hello world
Receive eventName with data: Hello world
Receive eventName with data: Hello world
^C
C:\tmp\socket>node index.js
listening on *:3000
express deprecated res.sendfile: Use res.sendFile instead index.js:18:7
Receive eventName with data: Hello world
Elements Console Sources Network Timeline Profiles Resources Audits
Receive eventName with data: Message
(index):13
Console Emulation Rendering