Node + Express + Socket.io = chat

Лаба: (β-версия) <https://docs.google.com/document/d/102Ngda6L-BzI65N1ArcHcvh2RgsqFgBLh8aTG1dMyRY/edit>

Архив с первым шагом: <https://github.com/gossoudarev/webteach/blob/master/express-chat/chat-express-1st-step.zip>

<http://kodaktor.ru/lr/node/chat-express-1st-step.zip>

A *socket* is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.

*Сокет* – это конечная точка двусторонней связи между двумя программами, работающими в сети. Сокет привязан к номеру порта, так чтобы слой TCP мог идентифицировать приложение, для которого предназначены данные.

С точки зрения программы сокет – это окошко, через которое виден другой участник взаимодействия. То есть интерфейс к нему.

Событие ‘connection’ возбуждается когда создаётся очередной сокет. В его коллбэк передаётся информация о сокете - объект. Внутри можно использовать этот объект уже для отслеживания этого конкретного соединения, в том числе отсоединения.

|  |
| --- |
| **io.on('connection', function(socket){**  **console.log('a user connected!');**  **socket.on('disconnect', function(){**  **console.log('a user disconnected!');**  **});**  **});** |

Для того, чтобы связь создалась, и это событие возбудилось, нужно, чтобы обратился веб-клиент по соответствующему протоколу. В терминах node/express это означает, что у нас должен быть какой-то маршрут типа /chat/, обращение к которому интерпретируется как обращение веб-клиента чата.

Что такое веб-клиент чата? Это веб-страница, содержащая интерфейс чата и сценарий для взаимодействия с сервером. С ней взаимодействует человеческий клиент чата.

В простейшем виде выглядит так:

|  |
| --- |
| **var** t1 **=** '<script src="/socket.io/socket.io.js"></script><h1>Chat web-client!</h1>',  t2 **=** '**<script>var socket = io();</script>**'; app.get('/chat', **function**(req, res){  res.send(t1**+**t2); }); |

Адрес /socket.io/socket.io.js автоматически генерируется серверной стороной.

выделенная красным строка инициализирует взаимодействие, т.е. посылает запрос, который возбуждает событие **'connection'**.

Но в действительности, конечно, эти строки располагаются в отдельной статической веб-странице.

Далее, мы видим, что и в клиентском, и в серверном сценарии после разных действий появляется переменная с именем socket. На обеих сторонах там содержатся разные по факту объекты, но они оба – сокеты. И функциональность у них одинаковая:

**socket**.**emit**('*имя\_события*', 'текст реплики');

**socket**.**on**('*имя\_события*', function(msg){

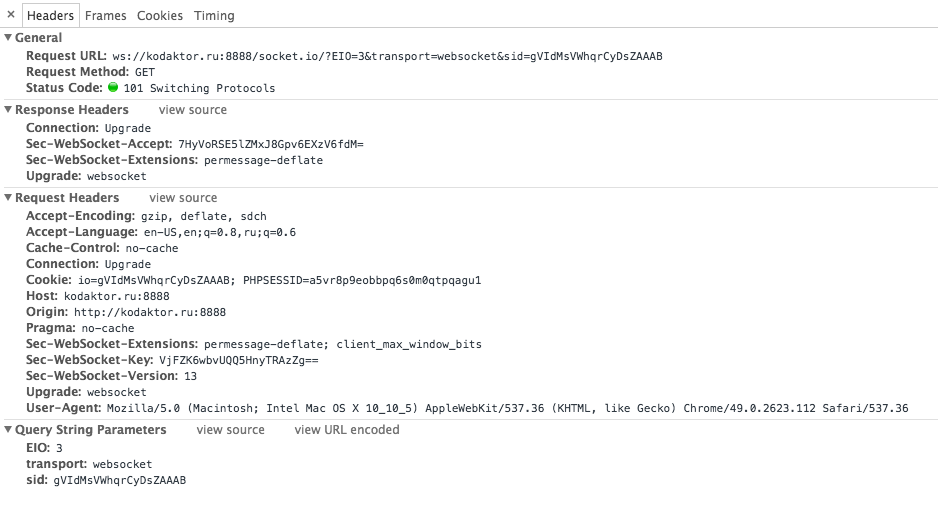
// что делать дальше

});

Итак, если мы даже встроили в каркас нашего приложения маршрут, который генерирует сокет, мы уже получили в первом приближении модель происходящего:

|  |
| --- |
| т.е. это «самодостаточный» серверный js-cценарий |
| **var** PORT **=** 8888; **var** app **=** require('express')(),  http **=** require('http').Server(app),  io **=** require('socket.io')(http);  module.exports **=** (**function**(){  **function** **inner**(){  this.start **=** **function** (whatToDo){      app.get('/page', **function**(req, res){  res.send('<h1>Hey it works!</h1>');  });    io.on('connection', **function**(socket){  **console**.log('a user connected!');  });    **var** t1 **=** '<script src="/socket.io/socket.io.js"></script><h1>Chat web-client!</h1>',   t2 **=** '<script>var socket = io();</script>';  app.get('/chat', **function**(req, res){ res.send(t1**+**t2);  });       http.listen(process.env.port **||** PORT, **function**(){  **console**.log(PORT)  });   };   }  **return** **new** **inner**; })(); |
|  |

При посещении адреса /chat мы получаем запрос по протоколу websocket:



… и возбуждение события сокета 'connection'

Далее нужно вывести клиентский код в отдельную страницу и дополнить работу с сокетами командами посылки сообщений и слушания их.

|  |
| --- |
| **var** PORT **=** 8888; **var express = require('express'),  app = express(),**  http **=** require('http').Server(app),  io **=** require('socket.io')(http);  module.exports **=** (**function**(){  **function** **inner**(){  this.start **=** **function** (whatToDo){  app.use(express.static(\_\_dirname **+** '/public'));     app.get('/page', **function**(req, res){  res.send('<h1>Hey it works!</h1>');  });    io.on('connection', **function**(socket){  **console**.log('a user connected!');  });     **app.get('/chat', function(req, res){   res.redirect('/client.html');    });**       http.listen(process.env.port **||** PORT, **function**(){  **console**.log(PORT)  });   };   }  **return** **new** **inner**; })(); |
| промежуточная версия веб-клиента: <http://kodaktor.ru/express_2e7bd> |

И, в общем-то теперь осталось параллельно добавить в клиентский и серверный код «зеркальные» команды слушания сокета и посылки в него реплики.

см. на следующей странице

|  |
| --- |
| **var** PORT **=** 8888; **var** express **=** require('express'),  app **=** express(),  http **=** require('http').Server(app),  io **=** require('socket.io')(http);  module.exports **=** (**function**(){  **function** **inner**(){  this.start **=** **function** (whatToDo){  app.use(express.static(\_\_dirname **+** '/public'));     app.get('/page', **function**(req, res){  res.send('<h1>Hey it works!</h1>');  });    io.on('connection', **function**(socket){  **console**.log('a user connected!');    socket.broadcast.emit('chat message push', '>> a user connected ');  //everyone except the new one  socket.emit('chat message push', 'Welcome to Ilia Goss chat!');  //only the newcomer   //message from client - recast to others  socket.on('chat message', **function**(msg){  **console**.log('message: ' **+** msg);  socket.broadcast.emit('chat message push', msg);  socket.emit('chat message push', '<strong>me:</strong> ' **+** msg );  });  socket.on('disconnect', **function**(){  **console**.log('a user disconnected!');  socket.broadcast.emit('chat message push', '>> a user disconnected ');  });   });    app.get('/chat', **function**(req, res){   res.redirect('/client.html');    });     http.listen(process.env.port **||** PORT, **function**(){  **console**.log(PORT);  });   };   }  **return** **new** **inner**; })(); |
|  |

Это близкая к финальной версия server.js

На след.странице – аналогично для веб-клиента client.html

|  |
| --- |
| <!DOCTYPE html> <**html**>  <**head**>  <**title**>Express Chat Form</**title**>  <**meta** charset="utf-8"><**style**>**\*** {**font-family**:**sans-serif**}   **span** {**position**:**fixed**; **right**:20**px**; **top**:10**px**;**border**:**double**; **padding**: 15**px**; **border-radius**:20**px**;}   **p** {**border**: **solid** 1**px** **silver**}</**style**>  <**script** src="//ajax.googleapis.com/ajax/libs/jquery/2.2.2/jquery.min.js"></**script**>   <**script** src="/socket.io/socket.io.js"></**script**>   <**script**>  $(**function**(){  **var** backs **=** [{"background":"green"}, {"background":"white"}];  **var** socket **=** io();  socket.on('chat message push', **function**(msg){  $('div').append($('<p>').html(msg));  });     $("#b").on("click", **function**(){  **socket.emit**('chat message', $('#i').val());//послать сообщение серверной части  $('#i').val('').css(backs[0]); //очистить поле ввода сообщения  **setTimeout**(**function**(){ $('#i').css(backs[1]); }, 500);  });  });  </**script**>  </**head**>  <**body**>  <**h1**>Содержимое чата:</**h1**>  <**div**></**div**>  <**span**><**h2**>Введите реплику:</**h2**>  <**input** id="i"><**button** id="b">Написать!</**button**></**span**>  </**body**> </**html**> |
| <http://kodaktor.ru/express_4d4b6> |