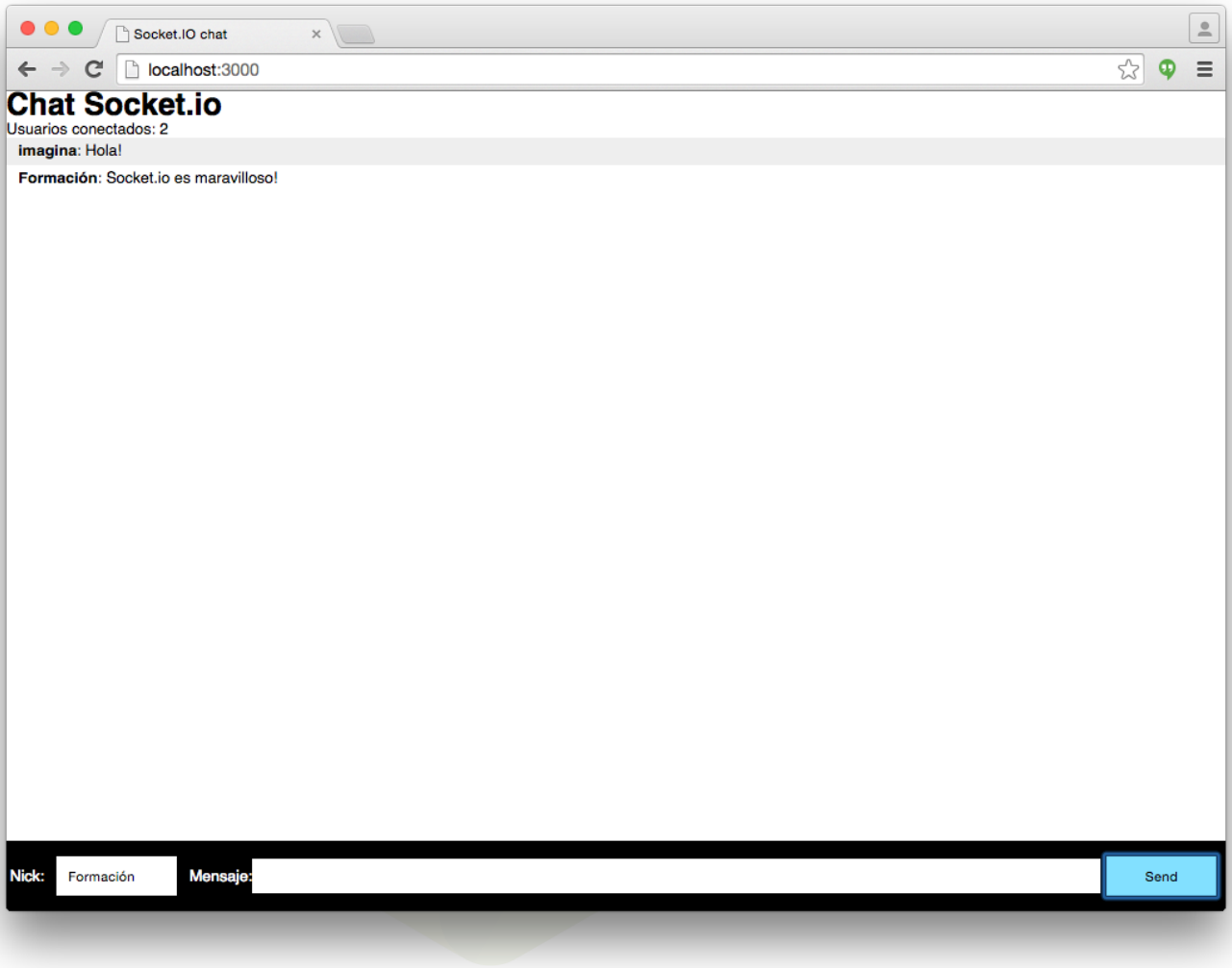
Ejercicios Tema 7

Ejercicio 1

El objetivo de este ejercicio es crear una sala de chat en tiempo real con el siguiente aspecto:



En primer lugar generamos un nuevo proyecto express:

```
$ express chat

   create : chat
   create : chat/package.json
   create : chat/app.js
   create : chat/public
   create : chat/public/javascripts
   create : chat/public/images
   create : chat/public/stylesheets
   create : chat/public/stylesheets/style.css
```

```
   create : chat/routes
   create : chat/routes/index.js
   create : chat/routes/users.js
   create : chat/views
   create : chat/views/index.jade
   create : chat/views/layout.jade
   create : chat/views/error.jade
   create : chat/bin
   create : chat/bin/www

   install dependencies:
     $ cd chat && npm install

   run the app:
     $ DEBUG=chat:* ./bin/www
```

Ejecutamos los comandos necesarios para instalar las dependencias de express y a continuación instalamos socket.io:

```
$ npm install --save socket.io
```

Preparamos el servidor para que escuche conexiones en el script **/bin/www**:

```
#!/usr/bin/env node

/**
 * Module dependencies.
 */

var app = require('../app');
var debug = require('debug')('test:server');
var http = require('http');


/**
 * Get port from environment and store in Express.
 */

var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);
var io = require('socket.io')(server);

/**
 * Listen on provided port, on all network interfaces.
 */
```



```
server.listen(port);
server.on('error', onError);
server.on('listening', onListening);



io.on('connection', function(socket){
  console.log('a user connected');
});

/**
 * Normalize a port into a number, string, or false.
 */

function normalizePort(val) {
  var port = parseInt(val, 10);

  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
    // port number
    return port;
  }

  return false;
}

/**
 * Event listener for HTTP server "error" event.
 */

function onError(error) {
  if (error.syscall !== 'listen') {
    throw error;
  }

  var bind = typeof port === 'string'
    ? 'Pipe ' + port
    : 'Port ' + port

  // handle specific listen errors with friendly messages
  switch (error.code) {
    case 'EACCES':
      console.error(bind + ' requires elevated privileges');
      process.exit(1);
```



```
      break;
    case 'EADDRINUSE':
      console.error(bind + ' is already in use');
      process.exit(1);
      break;
    default:
      throw error;
  }
}



/**
 * Event listener for HTTP server "listening" event.
 */

function onListening() {
  var addr = server.address();
  var bind = typeof addr === 'string'
    ? 'pipe ' + addr
    : 'port ' + addr.port;
  debug('Listening on ' + bind);
}
```

Ahora vamos a configurar el cliente para que lance el evento connection. Para ello modificamos el fichero **layout.jade** incluyendo el script de socket.io y conectamos con el servidor:

```
script(src="/socket.io/socket.io.js")
    script.
      var socket = io();
```

Una vez listas las dos partes, ejecutamos el servidor y abrimos una conexión a localhost:3000 con nuestro navegador.

Si hemos preparado todo bien veremos en la terminal que un usuario se ha conectado.

```
> node ./bin/www

a user connected
```

El siguiente paso es darle funcionalidad al chat así que vamos a crear una interfaz desde la que se pueda escribir y visualizar mensajes.

En **index.jade** creamos una lista y un formulario:

```
extends layout

block content
    h1 Chat Socket.io
    p#num usuarios conectados:0
    ul#messages

    form(action="")
      label#nicklabel(for="nick") Nick:
      input#nick(type="text",autocomplete="off")
      label#mlabel(for="m") Mensaje:
      input#m(type="text", autocomplete="off")
      button Send
```

En **layout.jade** agregamos estilos para especificar el aspecto de la página y también añadimos el script de Jquery que nos hará falta.

```
doctype html
html
  head
    title Socket.IO chat

    style.
      * { margin: 0; padding: 0; box-sizing: border-box;}
      body { font: 13px Helvetica, Arial; }
      form { background: #000; padding: 3px; position: fixed; bottom: 0;
             width: 100%; }
      form input { border: 0; padding: 10px; width: 80%; margin-right: .
             5%; }
      form button { width: 9%; background: rgb(130, 224, 255); border:
             none; padding: 10px; }
      #messages { list-style-type: none; margin-bottom: 100px; padding:
                  0; }
      #messages li { padding: 5px 10px; }
      #messages li:nth-child(odd) { background: #eee; }
      #nick { margin: 10px; width: 10% }
      #nicklabel{color: white;}
      #mlabel{color: white;}
      #m {padding:8px; width:70%;}


  body
    block content
    script(src="/socket.io/socket.io.js")
    script(src="http://code.jquery.com/jquery-1.11.1.js")
    script.
      var socket = io();
```


Si ejecutamos ahora, visualizaremos la interfaz del chat pero no tendrá ningún comportamiento. Para que el chat funcione, tenemos que emitir lo que el usuario teclea hacia el servidor y el servidor deberá difundirlo a todos los sockets.

El código necesario para emitir y recibir mensajes en el cliente:

```
body
    block content
    script(src="/socket.io/socket.io.js")
    script(src="http://code.jquery.com/jquery-1.11.1.js")
    script.
      var socket = io();
      var cont = 0;
      var fnick = $('#nick');
      var fmessage = $('#m');

      $('form').submit(function(){
        socket.emit('chat message', {nick : fnick.val(), message :
          fmessage.val()});
        $('#m').val('');
        return false;
      });
      socket.on('chat message', function(msg){
        $('#messages').append($('<li>').append('<b>'+msg.nick+'</b>: ' +
          msg.message));
      });
      socket.on('chat users', function(msg){
        $('#num').text("Usuarios conectados: "+ msg);
      });
```

Jquery nos permite hacer referencia a elementos del DOM y modificarlos dinámicamente. Hacemos que el submit del formulario lance el evento chat message y pasamos como datos el nick y el texto tecleados.

El resto de código son dos listeners para visualizar los nuevos mensajes y el número de usuarios en el chat.

En el servidor gestionamos los mensajes que llegan difundiéndolos al resto de usuarios:

```
io.on('connection', function(socket){

  console.log('a user connected');
  socket.broadcast.emit('chat message', {nick:"INFO",message:"Nuevo
                                         usuario en la sala"});
  io.emit('chat users',io.engine.clientsCount);
```



```
 socket.on('chat message', function(msg){
     io.emit('chat message', msg);
  });

  socket.on('disconnect', function(){
    console.log('user disconnected');
     io.emit('chat message', {nick:"INFO",message:"Un usuario ha
                              abandonado la sala"});
     io.emit('chat users',io.engine.clientsCount);
  });

});
```

El chat ya debería ser funcional. Podemos probarlo abriendo múltiples ventanas del navegador conectadas a localhost:3000.

Ejercicio 2

Muestra los usuarios que se encuentran online

Ejercicio 3

Habilita chats privados*

*Es una buena idea hacer que el cliente al entrar tenga que elegir su nick, y así al ejecutar el evento `connection` podemos guardar este valor en un array junto a su id de socket ;)

Recuerda. Socket.io tiene dos partes, cliente (en nuestros jade) y servidor (en /bin/www).

Ampliación

Crea salas privadas

