Assignment 1

WebSocket Chat service

First name: Robert-Oliver

Last name: Kajnak-Misca

Student number: 2614846

VUnet-Id: rka740

Server

The project can be launched, as described in the requirements by supplying npm with 2 parameters: the socket for the web and websocket servers respectively. This is handled in the index.js file. This launches the server by calling the start function from the js file with the same name.

In line with the many-times-repeated sentence from lectures I tried using as few packages as possible, only including that from the provided package.json. For the strict web side, I did manage this, however I had problems with the socket.io (also specified in the package.json). I could manually do the rewdirection to the socket.io.js file, from the installed packates with success (see commented section under <request.url===”/socket.io/socket.io.js”> section, or using the <script src="https://cdn.socket.io/socket.io-1.5.0.js"></script> html solution. But I did not manage to do the rerouting of the get commands nor did I find the relevant documentation pertaining to it (beginning with …?EIO=…).

HTML/server

To simplify the number of variables and POSTS, both the user registration and join/create servers are served on the same page. This also makes it easier to change it into a format where the username and server name are entered at the same time. In this iteration the server fields and buttons are hidden, until the User name is filled in and verified. The verification is done by checking the length and character composition using a regex to abide specifications. No explicit database is used such as MySQL, since the service is not designed to be a long activity with millions of users. With a modest spare RAM of 4GB and considering average typing speed of 300 CMP in ascii would result in roughly 350 bytes/ sec, accounting for some overhead. With 100 concurrent users, nonstop using the service, it would take several days to fill up the server. As such with a more realistic workload, it would take around a month. This also eliminates the possibility of SQL injection (though there isn’t even a password check, so who would bother).

The username is submitted as requested using the sendObjectAsJSON function. The return values from the POST operation are handled using callback functions as parameters. This allows asynchronous data transmission both to the server and receiving it back. Two separate functions can be passed for cases of success and failure. The sendObjectAsJSON function is created in a way that can be utilized both by create chatroom and register user. The structure is general enough to be used by similar functions, should the program be extended. Though this has the drawback that the return code is passed to the functions in the parameters, which is not as logical as being handled inside the main send function.

According to requirement 2.3, a registered user should be able to establish a connection to a websocket using the URI. This implies either cookies or session variables. The problem was solves using a cookie, in which the username and ID is stored. After saving this data, on successful chatroom join, the website redirects itself to the websocket. This however did not function during testing. I decided to shelf the issue and try solving the chat service, assuming a direct link access.

Javascript/server

As mentioned previously, I tried to solve the assignment using only the base functions. I also included md5, since writing the hash function would have been long, hard to read, and I did not see much point in copy-pasting a done implementation, nor did I want to implement it myself.

The get section mainly handles the / and /index requests. For the favicon, an empty page with 404 is returned. For invalid requests a “Nothing to see here” message is returned.

For the post sections, the data is taken in chunks and is checked on chunk concatenation for too long messages that would either block or flood the server. On successful POST request, the two valid options are /user and /rooms. For the /user, the sent string is assumed to be JSON, parsed, than noted in an array, that functions as a database. If the user already exists, the specified error message is sent, otherwise the md5 hash is generated and included in the response.

For the room section, the url is generated and integrated in the response.

HTML/chat

The structure is based on the official example provided on socket.io. It is simplified to make the code shorter and remove the (unnecessary) colors. The username and ID variables are taken from the cookie. These do not always work for some reason. The room ID is marked as <room> which is filled in using the javascript from the index file, which is actually serving this HTML. On field competion and submission, the message is broadcast to the tag ‘send’, and the room provided by the javascript.

Javascript/Websocket

The websockets uses the express framework to make both routing and socket.io usage easier. To circumvent the cookie problem mentioned above a temporary fix was devised, which replaced the value of the room and username directly into the html from the URI.

On socket.io connection the user is expected to use the subscribe tag to register to a room. The username and ID should also be verified should also be verified, however this has not been implemented. Also the messaging worked, when using a single room, however it is currently not functional with the multi-room setup.