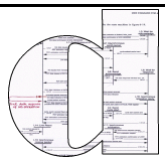
DEPENDABLE SYSTEMS AND SOFTWARE

Data Networks 2015 Dr. Arnd Hartmanns



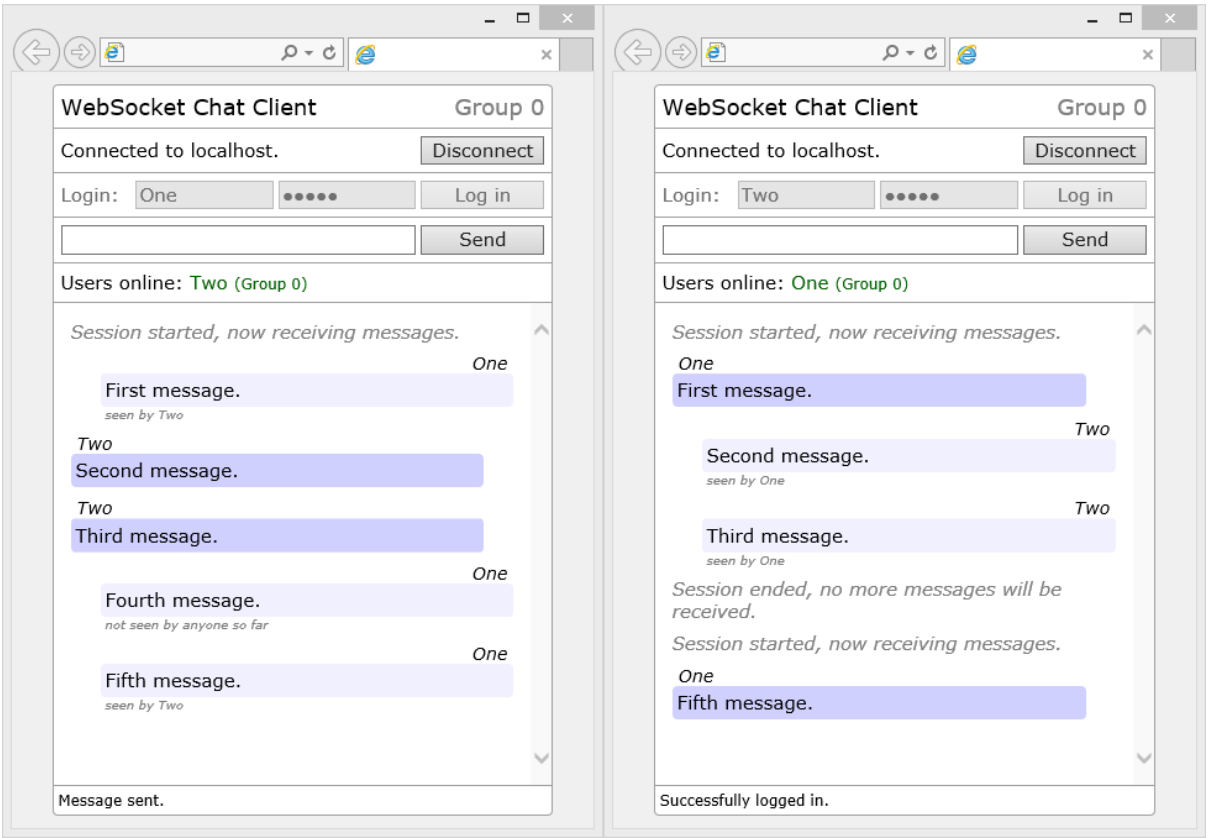
PL1: WebSocket Chat Client

The soft deadline for this programming lab is May 31. The hard deadline is June 07.

Submission: Your submission must be a `.zip` file that contains, at the top level, two files `ChatClient.html` and `ChatClient.js` that implement your solution for this lab. Any extra files that your solution needs must also be included in the `.zip` file. All code you submit must have been written by members of your group, and must not reference any third-party libraries. However, you may use the code template provided in dCMS. Your solution must be easily readable and well-commented.

Your task is to implement a browser-based client, using HTML and JavaScript, for a chat protocol called *dnChat* running on top of the WebSocket protocol (as defined in RFC 6455). By default, *dnChat* servers listen on port 42015, and the protocol does not use a secure connection. To test your implementation, a *dnChat* server that accepts your group password is available most of the time on host `modestchecker.net`.

The browser-based user interface shall provide the ability to connect to a *dnChat* server specified by the user, to log in to the server with a user-specified *name* and *password*, to send user-specified chat messages to all or to individual users connected to the server, and to disconnect from the server. A list of all other users connected to the server shall be shown. Chat messages sent by the local user to other users and chat messages received from other users via the server shall be displayed. Each chat message sent from the local user shall show an indication of whether it has been successfully received by the server, and show a list of the other users that have successfully received the message. An example of such a user interface based on the code template provided in dCMS is shown below. It shows a chat session between two users, where the user called "Two" was disconnected when the fourth chat message was sent.



The *dnChat* protocol uses WebSocket text messages to communicate between client and server. Lines are separated by a pair of a carriage return and a line feed character (`CRLF`, or `\r\n` in JavaScript strings). The first line of each message consists of four capital letters indicating the *command*, followed by a space, followed by a non-empty sequence of digits `0` through `9` not starting with a zero that represents the *number* of the entity *referred to* by the message. Depending on the command, more lines may follow.

Client-initiated communication

A *dnChat* client is in either *disconnected*, *connected* or *authenticated* state. When there is no WebSocket connection to the server, the client is in *disconnected* state. By successfully establishing a WebSocket connection to the server, the client transitions to *connected* state. The client may send the following commands to the server:

`AUTH`
Only valid in *connected* state. Requests to transition to *authenticated* state. The number must be a random number generated by the client, and will subsequently be used to refer to the user. The second line of the message is the *name* chosen by the user, and the third line is the *password*. When authentication is successful, the server sends a message with command `OKAY` and the same number back to the client. Upon receipt of this message, the client transitions to *authenticated* state. Otherwise, the server sends a message with command `FAIL`, the same number, and the second line containing the failure reason back to the client. In this case, the client remains in *connected* state.

`SEND`
Only valid in *authenticated* state. Sends a chat message for other users. The number must be a random number generated by the client, and will subsequently be used to refer to the chat message. The second line is either `*` to indicate that the chat message shall be forwarded to all other users connected to the server, or it is a number referring to the user that the chat message shall be forwarded to. The third line is the chat message text. Upon receipt of the chat message, the server sends a message with command `OKAY` and the same number back to the client. In case the message cannot be accepted by the server, it sends a message with command `FAIL`, the same number, and the second line containing the failure reason back to the client.

`ACKN`
Only valid in *authenticated* state. Acknowledges that a chat message from another user has been received at the client and displayed to the local user. The number refers to the chat message.

Failure indications

The failure indications of the `FAIL` command are as follows:

`NAME`: The specified name is already in use by another client.

`PASSWORD`: The password is not acceptable for authentication on this server.

`NUMBER`: A number is not valid, either because it has already been used for another entity in case it was newly generated by the client, or there is no relevant entity that the number refers to.

`LENGTH`: The chat message text is too long.

Invalid messages

Whenever the server receives a malformed message or a message that is not valid in the current state of the client, it replies with command `INVD` with number zero and closes the WebSocket connection.

Server-initiated communication

In *authenticated* state, the client may at any time receive any of the following messages from the server in addition to the replies described above, and it must process these messages as described below:

`ACKN`
Acknowledges that a chat message previously sent by the client has been received at another client and displayed to another user. The number refers to the chat message. The second line is the number of the other user.

`SEND`
Provides a chat message sent by another user to the local user. The number refers to the chat message. The second line is the number of the other user. The third line is the chat message text. The client must display the message to the local user and send a message with command `ACKN` back to the server as described in the section on client-initiated communication.

`ARRV`
Notifies the client that another user is connected to the server. The number refers to the other user. The second line is the name of the other user. The third line is the description of the other user. The client must show the name and description of the other user to the local user exactly as long as the other user is known to be connected to the server.

`LEFT`
Notifies the client that another user has disconnected from the server. The number refers to the other user.