BxlContact: a basic app for Secret Communication

Nicolas Gennart, Yiyu Wang, Mohamad Saab, Semilogo Ogungbure

December 2021

Contents

1 Architecture

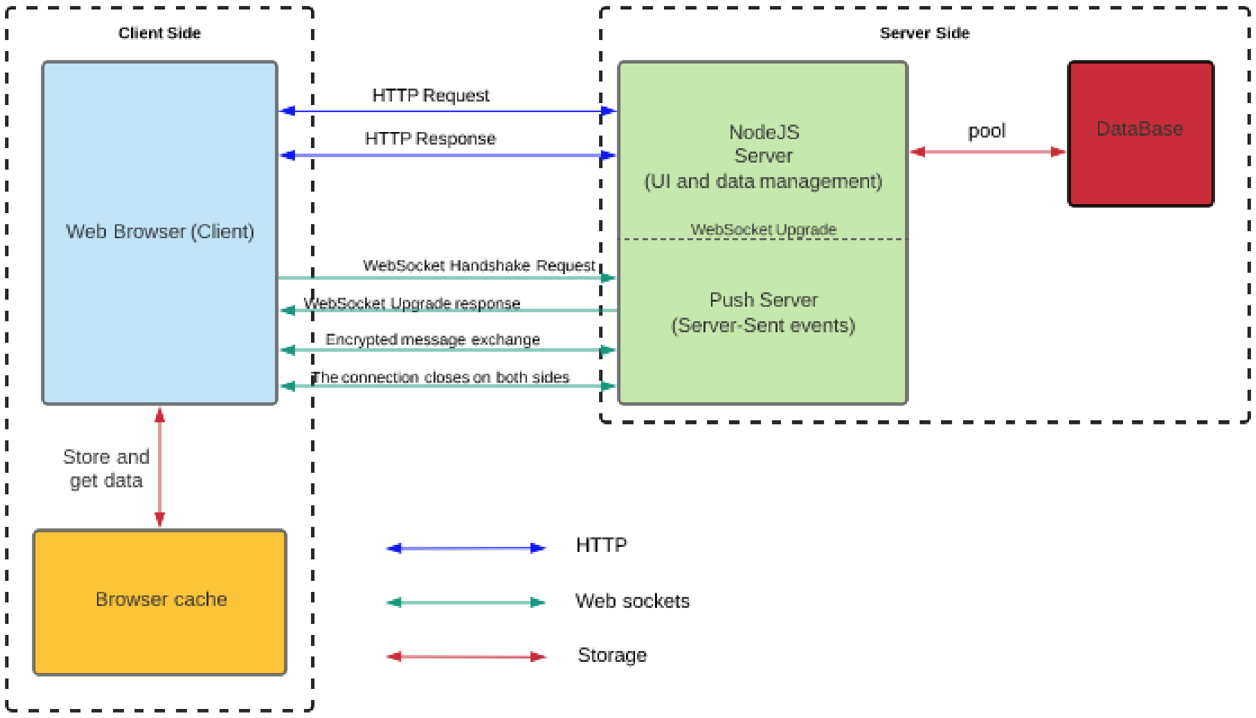

Client Side
Server Side
Web Browser (Client)
HTTP Request
HTTP Response
NodeJS Server (UI and data management)
pool
DataBase
WebSocket Upgrade
WebSocket Handshake Request
WebSocket Upgrade response
Encrypted message exchange
The connection closes on both sides
Push Server (Server-Sent events)
Store and get data
Browser cache
HTTP
Web sockets
Storage

Figure 1: General Architecture

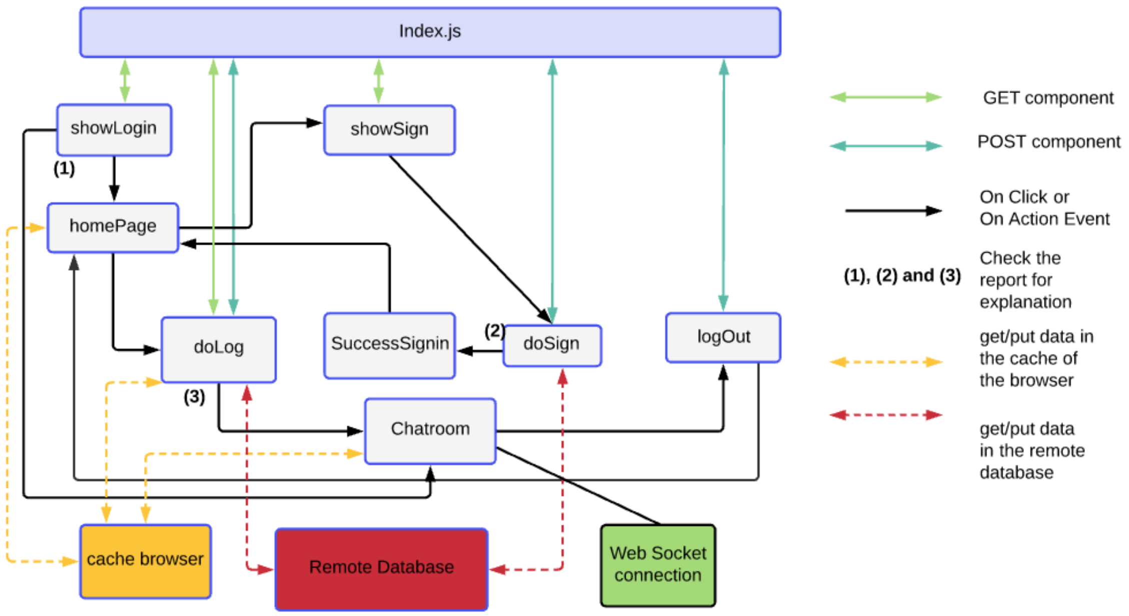


Figure 2: User Controller architecture

(1) When a user wants to connect to the online chat, if he has previously connected with the same browser and has not logged out, he will enter directly into the chatroom (because he will have an existing session store in the server). Otherwise he enter in the homePage and the server will generate the challengecode and store it in the new session created and then sent it to the user who will store it in the browser (in the homePage).

(2) Usernames are unique so the server will look if the user is not already existing if it is not existing the new user will be stored in the remote database and his password will be encrypted with a key stored in the server (this key will be used to encrypt and decrypt all stored passwords).

(3) When the user wants to log in, on the client side we will hash the concatenation of the password and the challengecode (stored in the browser) and then send it to the server. The server will get the hash (send by the client) then the server does the same by retrieving the encrypted password from the remote database, decrypting it and also hashing the concatenation of the password retrieved from the database and the chalengecode stored in the session. If they are equal the user will be able to log in otherwise he will not be able to.

List of decisions

We chose to use node.js because it is easy to learn, to keep things simple and not to complicate them. Node.js is also very popular nowadays and default uses only one thread, so it uses the non-blocking I/O paradigm, which ensures a good performance. (Non-blocking I/O allow a single process to serve multiple requests at the same time.)

We used the websocket, it allows to realize a bi-directional connection between the server and the client. Thanks to this, it will not be necessary to make a TCP connection for each HTTP request, it will only be necessary to make the connection once (the connection will be made during the "websocket upgrade" as shown in the diagram 1). Therefore the server uses the server-sent event mechanism where as soon as an event happens the server can communicate it to the client. This is very efficient for real time applications like our online chat. Server-sent event is the most efficient because unlike short polling where the request is fast but uses a lot of traffic and long polling which doesn't use a lot of traffic but the response is slow, it responds quickly and doesn't use a lot of traffic.

For the database we used MYSQL because it is the most efficient to manipulate the data and it can be multi-threaded and multi-users. So we can add a thread (the pool see figure 1) to listen permanently to the database when there is an access request to write or read data in.

We have also used express incorporated in nodeJS, to be able to use "Router" for a better security (we will explain that latter in the report 2.4) and also to be able to communicate between the server and the client with HTTP requests/response.

2 Innovation and creativity

2.1 Symmetric encryption

For symmetric encryption we have two cases:

The first one when the user is not in secure mode. We encrypt and decrypt the messages with the concatenation of the two communicating usernames as secret key. The messages will be stored in the remote database and the server will not be able to read the messages but it could easily guess the key by xsoring the two usernames, that's why we arrive in the second case which is the secret mode.

The second one is when we are in secret mode. We use the Diffie-Hellmann algorithm to exchange the keys, the procedure is shown bellow.

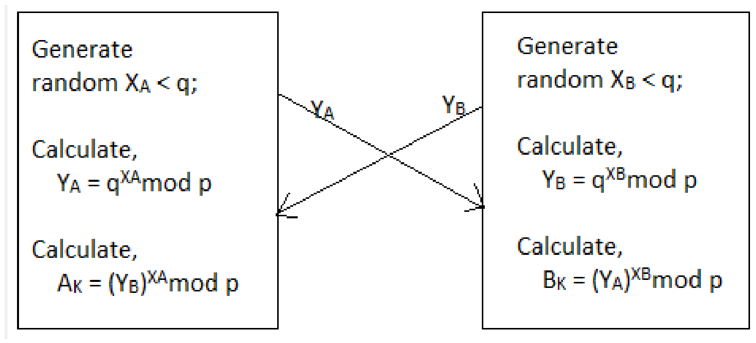

Generate random $X_A < q$;
Calculate, $Y_A = q^{XA} \bmod p$
Calculate, $A_K = (Y_B)^{XA} \bmod p$
Y_A
Y_B
Generate random $X_B < q$;
Calculate, $Y_B = q^{XB} \bmod p$
Calculate, $B_K = (Y_A)^{XB} \bmod p$

Figure 3: Diffie-Hellmann Key Echange (link of the picture)

With the Diffie-Hellmann algorithm the server will never be able to decrypt the messages by guessing the key (at least it will have very little chance), however the messages will not be stored in the database but in the cache browser, it will be explained why in the section 3.3. Unfortunately the attacker will always be able to perform a man in the middle attack, it is possible to counter this by using the libsignal-protocol library but for reasons of difficulty and time we have not been able to do it.

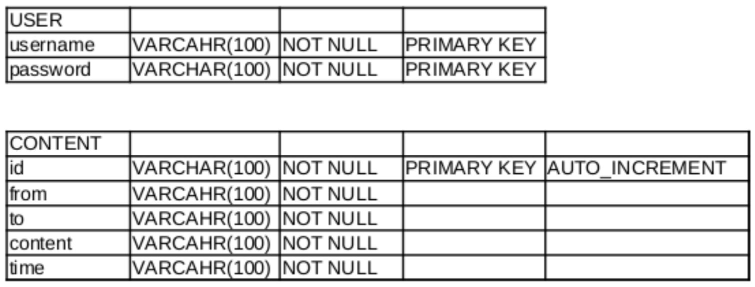
2.2 Remote server

As Yiyu Wang owns an Alibaba server we could put the chat online, so anyone can go and connect to the chat and use it. We put it online by assigning a domain name to the chat, we linked the domain name of the chat with the ip adress of the server with a reverse proxy.

2.3 Remote database

The database is also stored on Yiyu Wang's server. The credentials and messages of users will be stored in the remote database store in the server of Yiyu Wang. As stated above 1 there is a thread that constantly listens to the database in case there is any writing or reading of data.

Below you can see the two tables stored in the MySQL database:



USER			
username	VARCAHR(100)	NOT NULL	PRIMARY KEY
password	VARCHAR(100)	NOT NULL	PRIMARY KEY

CONTENT				
id	VARCHAR(100)	NOT NULL	PRIMARY KEY	AUTO_INCREMENT
from	VARCAHR(100)	NOT NULL		
to	VARCAHR(100)	NOT NULL		
content	VARCAHR(100)	NOT NULL		
time	VARCAHR(100)	NOT NULL		

Figure 4: SQL Tables

2.4 Router (from express)

"Router" will allow us to hide the URL parameters on the client side, the client will not know the structure and information stored on the server. It is almost similar to RESTful. So as shown in the image 2, in the POST methods message will be hidden in the request body. To access POST methods such as doSign or logOut the client will have to click on a button or perform an action and will not be able to access it by putting the path of the url.

2.5 UI

For the graphical interface we used bootstrap and jquery that we incorporated in our project. Bootstrap allows to get a responsive website and a beautiful website. That is to say that the graphical interface will adapt according to the size of the screen. The screen will adapt depending on whether the screen is a computer, an iPad or a phone.

2.6 Web socket

To exchange the encrypted messages, the different keys and to notify all users of the arrival or disconnection of a user, we use a webSocket, which allows to establish only one TCP connection and then to keep it and thus to use the mechanism of "Server-Send Events" as explained above 1.

3 Challenges

3.1 JSON (transform the data)

When the client and the server exchange data through the websocket, this data must be transformed into JSON format. The transformation of the object into JSON format will modify the object and therefore when receiving this object it may not be the same and the object will not be recovered in its original form. This happened in our project for the hashMAP and the decryption of messages.

For the hashMap we managed to solve the problem by implementing a resolve function and a reviver function taken from the the StackOverFlow forum. The resolve function will allow to transform a HashMap into an object and then this object is then put in a correct JSON format and the reviver function will allow to retrieve the HashMap from the received object in JSON format.

For the decryption of the messages we had to access the value of the tuple which was "array of buffer", so that we could only get the array of buffer and not the whole tuple.

3.2 Module crypto in the client side

In order for the client to encrypt a message, decrypt a message and generate its private and public keys, it must have the "crypto" module. We use the crypto module to realize cryptographic algorithms such as AES192 or DiffieHellmann. But the Client can't include this module because it does not include any module from NodeJs.

So to solve this problem we had to use the "browserify" command which is a javascript grouping tool to include the necessary modules in a javascript file so that it compiles on the browser. In our case when we call the browserify command on the chatroom.js file which will be on the client side it will generate the same file except that it will have the crypto module. Therefore this file will be able to call and use the cryptographic function that it needs.

3.3 Store message history

At first we wanted to have the user history (all messages sent between users since their creation) stored in the ram of the server. We noticed that when we did this, the server's ram would quickly become overloaded and the server would be very slow. So we decided to create a database to store all the messages sent between users.

Therefore when creating this database we had to face a second challenge. The messages exchanged between the users are encrypted and cannot be decrypted by the server. The server can store and redistribute the encrypted messages to the different users but it is on the client side that the decryption must be done. Therefore the client must be able to retrieve the symmetric key with which it encrypted the previous messages. But as we wanted the messages to be encrypted with a symmetrical key generated with the Diffie Hellmann algorithm, the user will not be able to retrieve these keys to decrypt these messages sent previously. He will have stored these keys locally in the cache of his browser and they will disappear afterwards. The user will not be able to store these keys in the server's database either, otherwise the server will be able to decrypt these messages.

That's why we decided to implement two encryption modes as explained above 2.1. The first encryption mode is to generate a symmetrical key with the names of the users. This symmetric key can then be easily recalculated by the user and he will be able to retrieve the history of these messages stored in the server database by decrypting them with the recalculated symmetric key. The first method allows the user to recover the history of these messages but the key can be easily guessed. That's why we made a second encryption mode where the messages are not stored in the database but the key will be almost impossible to guess. The user will not be able to retrieve the history of these messages but the messages will be encrypted with a key generated by the Diffie Hellman cryptographic algorithm.