**Rapport d’analyse**

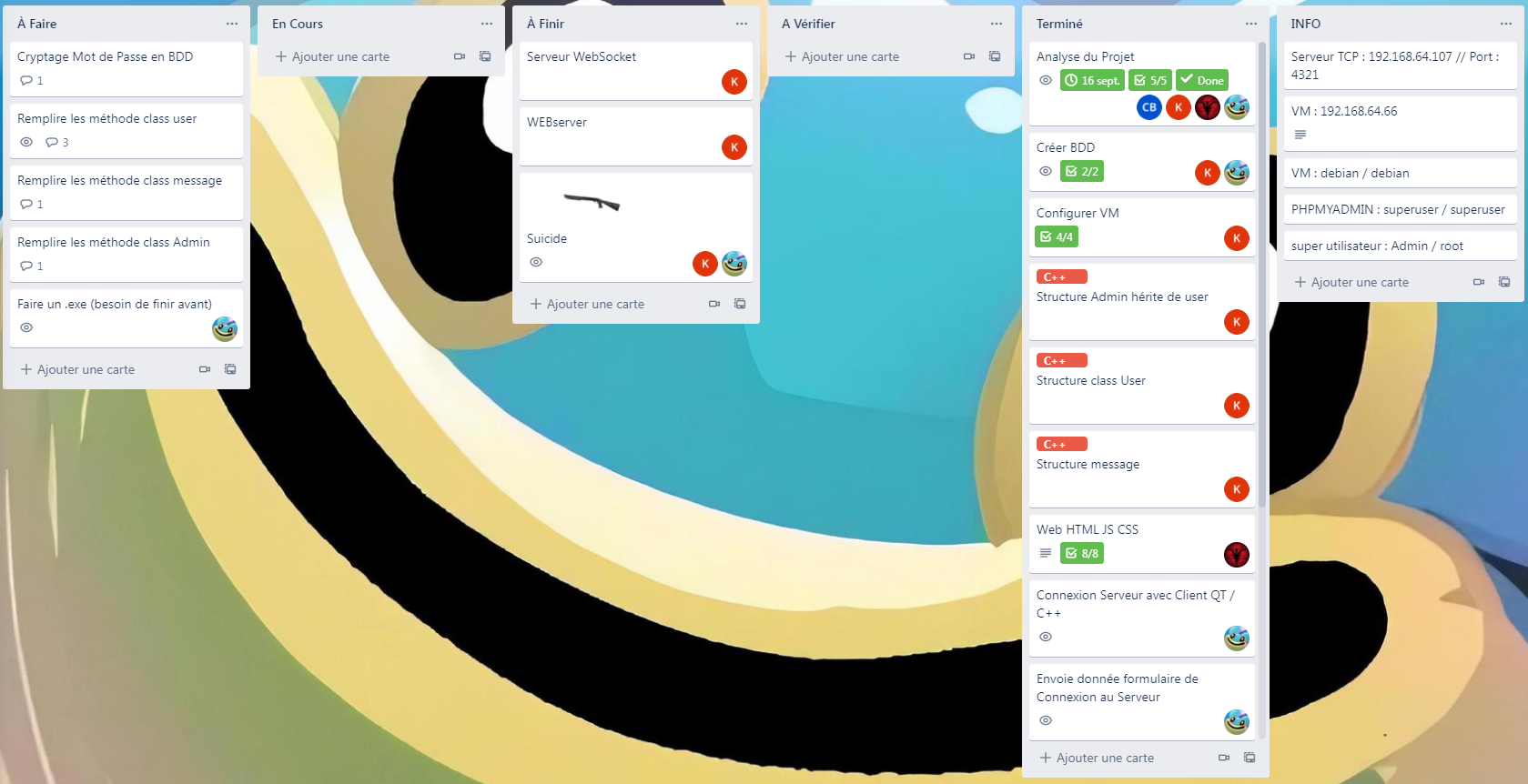
VOLLET Maxence

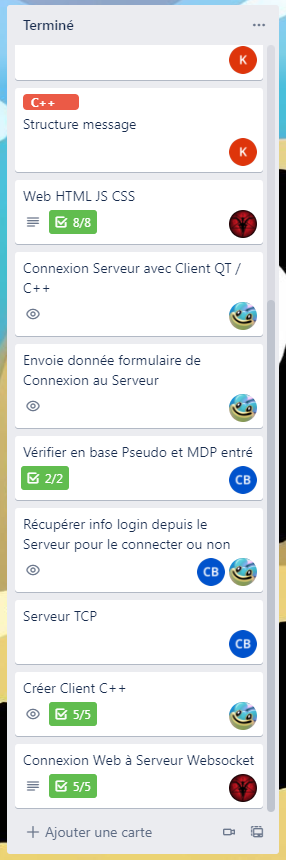
COLSON Baptiste

DUVAL Kylian

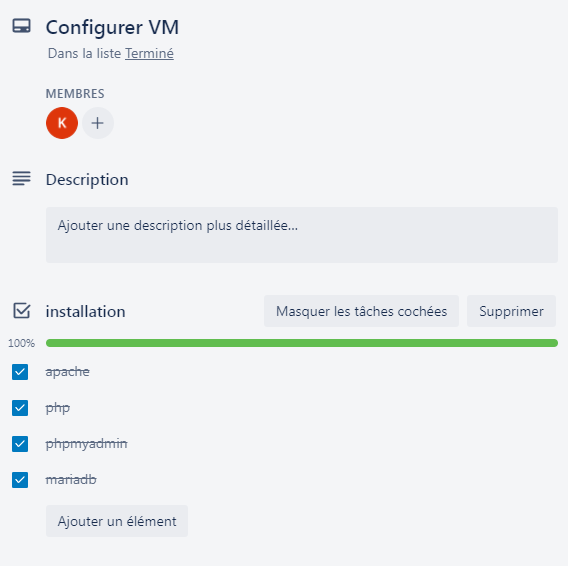
DE ALMEIDA Nicolas

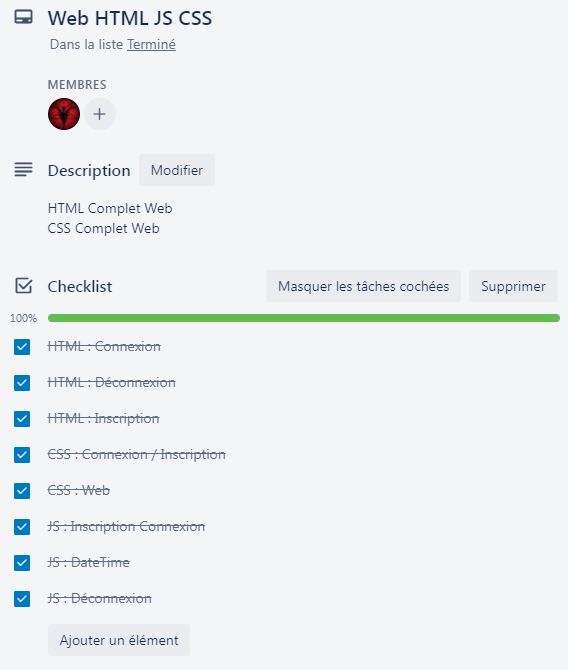
Trello :



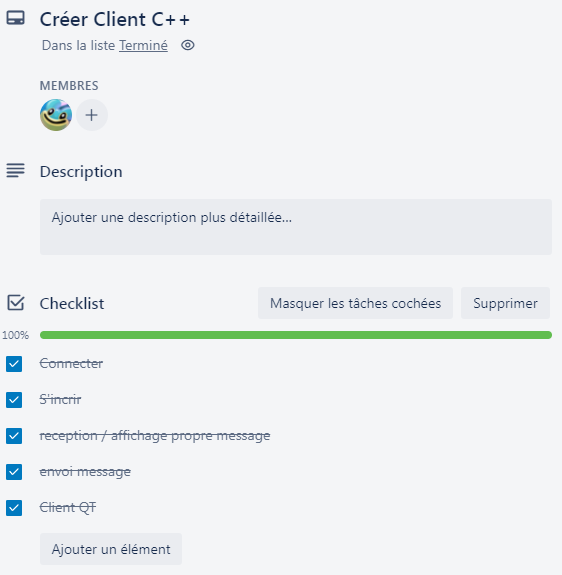












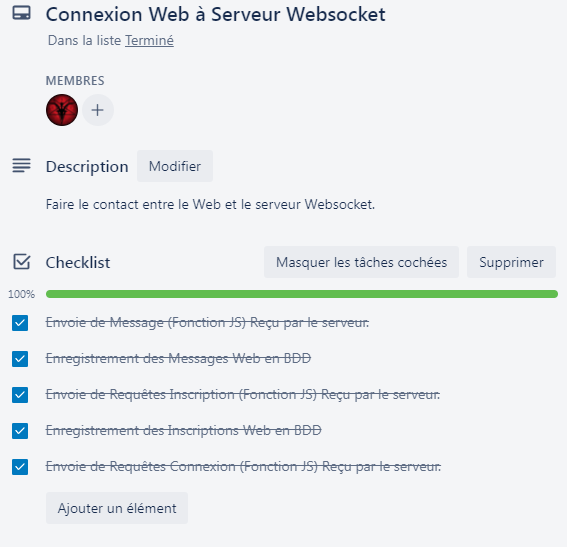


Diagramme de class :

(La personne devant s’occuper des classes n’ayant pas remplis les méthodes des classes et mit uniquement les structures pour Dieu sait quelle raison, toutes les méthodes sont dans la classe de base de QT)

Diagramme de déploiement :

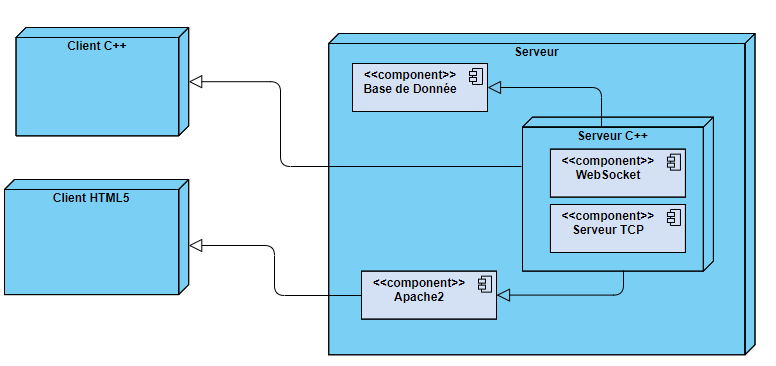
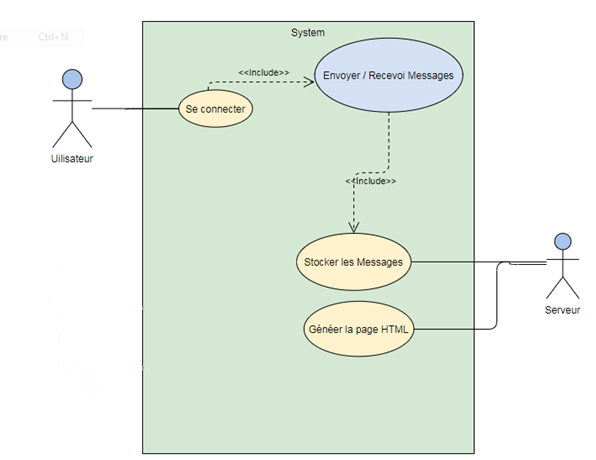


Diagramme des cas d’utilisation :



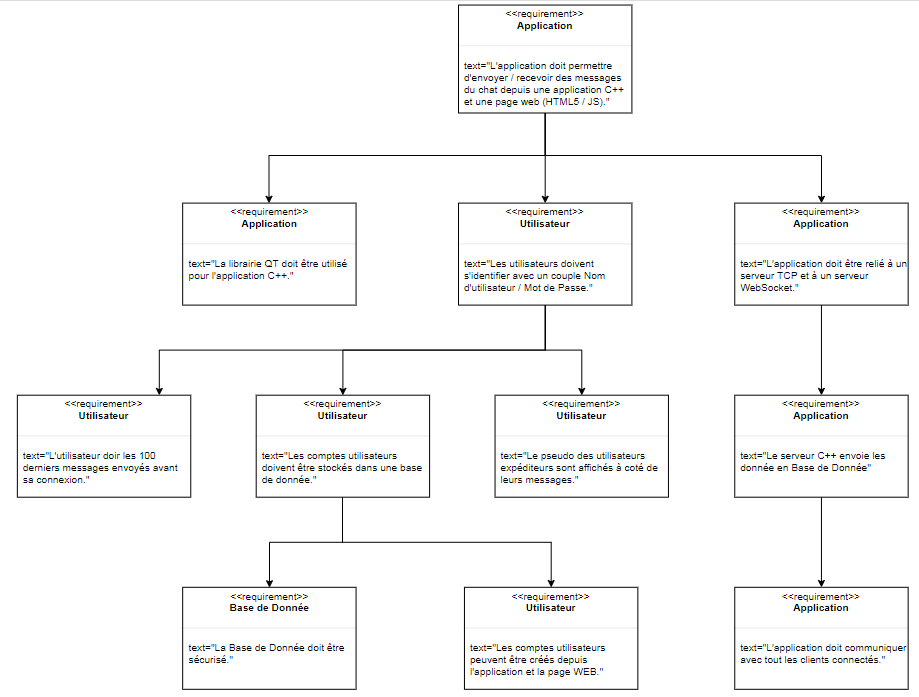
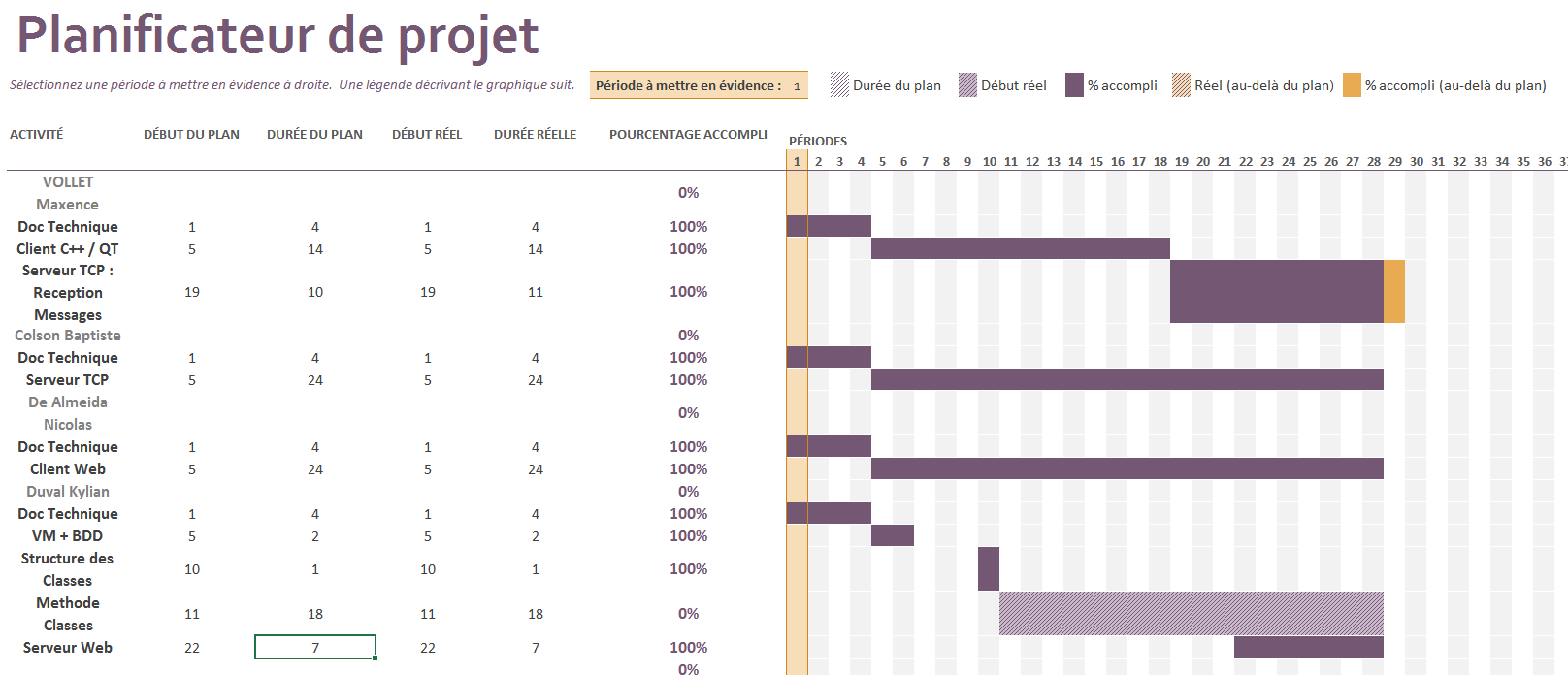
Diagramme d’exigence :

Diagramme de GANTT :



Cahier de recette :

|  |  |  |
| --- | --- | --- |
|  | ID | OK / NOK |
| Bouton connexion Serveur | 1 | OK |
| Formulaire Connexion | 2 | OK |
| Formulaire Inscription | 3 | OK |
| Envoie Message | 4 | OK |
| Déconnexion | 5 | OK |
| Actualisation chat | 6 | OK |

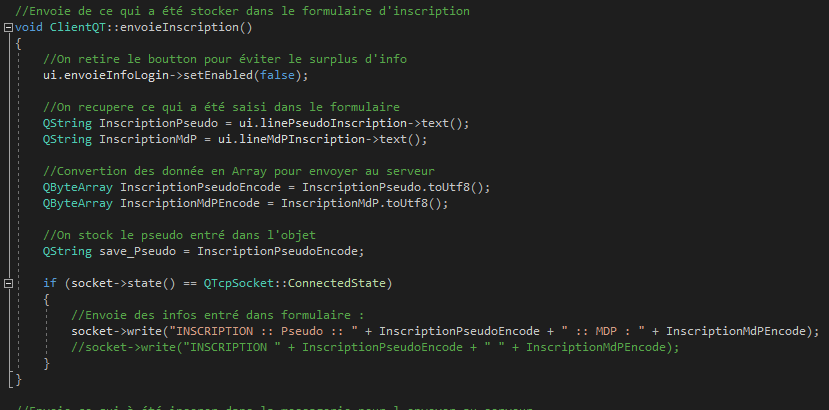
Cahier de test :

|  |  |  |  |
| --- | --- | --- | --- |
|  | Attendu | Commentaire | OK / NOK |
| Lancer le client lorsque le serveur n’est pas encore allumé | Affichage d’un bouton pour se connecter lorsque le serveur sera ouvert | 1 chance sur 2 de se connecter automatiquement | OK |
| Se connecter avec un couple Pseudo et Mot de Passe qui n’existe pas dans la base | Affichage erreur | Impossible de se connecter mais aucune erreur ne s’affiche | NOK |
| Inscription avec un couple Pseudo et Mot de Passe qui existe dans la base | Redirection vers le formulaire de connexion et ajout dans la base |  | OK |
| Inscription avec un couple Pseudo / Mot de Passe qui existe dans la base | Affichage d’une erreur | L’inscription ne s’effectue pas mais aucune erreur ne s’affiche | NOK |
| L’utilisateur peux se déconnecter de la messagerie | Retour vers la page de connexion |  | OK |
| L’utilisateur peux envoyer un message | Envoie du message et mise à jour du chat |  | OK |
| Lorsque l’utilisateur envoie un message, tous les autres clients le reçoive également | Mise à jour du chat pour tous |  | OK |
| Impossible d’envoyer un message vide | Rien ne se produit |  | OK |

**Documentation Technique Client QT**

Par VOLLET Maxence

**Code pour envoyer des données au serveur**



L’exemple choisis ici, mais ceci peut s’appliquer à toutes les données envoyer : il suffit juste de changer le premier mot de ce qui est envoyer par « LOGIN », « INSCRIPTION » ou « MESSAGE » pour pouvoir utiliser ce code partout.

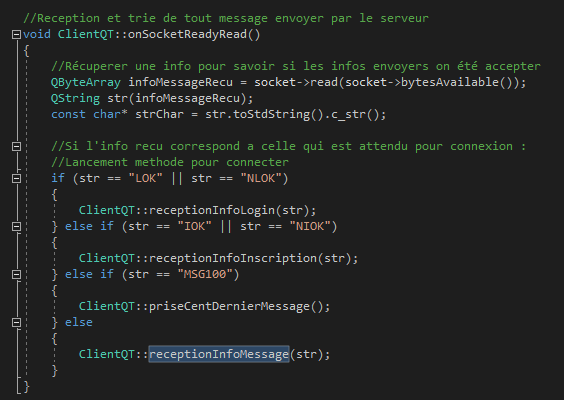
Nous récupérons donc les informations entrées dans le formulaire sous la forme d’un QString que l’on convertit en QByteArray. Si l’on est connecté, le client envoie au serveur les informations entré sous la forme :

*« INSCRIPTION :: Pseudo :: \*PseudoEntrée\* :: MDP : \*MDPEntrée\* »*

Cette phrase est celle qui permet de communiquer avec le serveur pour lui envoyer des messages.

**Code de trie lorsque l’on reçoit un message du serveur**

On convertit le message reçu en QString, puis en une chaine de caractère pour pouvoir les comparé à nos différentes possibilités :



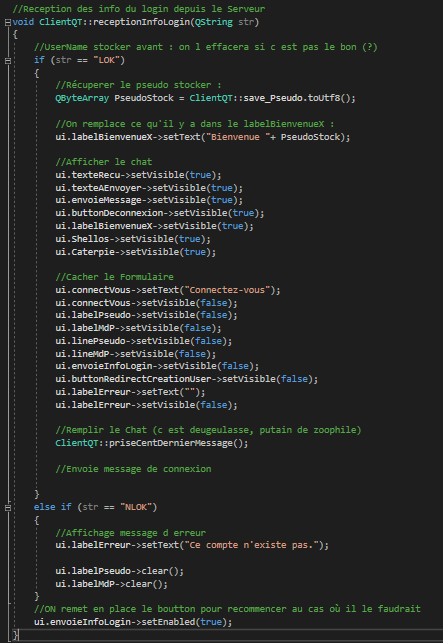
Si le serveur nous envoie une information correspondant à une connexion, c’est-à-dire que l’on reçoit « LOK » ou « NLOK », on envoie le message reçu dans la fonction qui gère les connexions.

Même chose si l’on reçoit une information concernant une inscription, « IOK » ou « NIOK » : on envoie ce que l’on a reçu dans la fonction s’occupant des inscriptions.

Si on reçoit « MSG100 », c’est que le serveur nous demande de lui envoyer un message précis. On s’occupe de cela dans la fonction y correspondant.

Si on reçoit un message autre que ceux-là, c’est que le serveur nous envoie les 100 derniers messages stocker en base qui ont le droit de s’afficher.

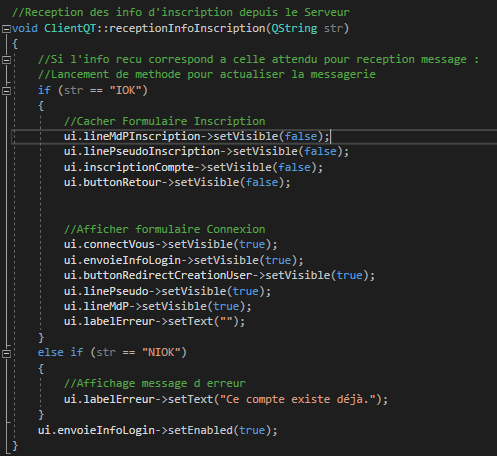
Ceci est la fonction s’occupant des connexions :



Si on reçoit la chaine de caractère indiquant que les infos de connexion sont bonnes, alors on affiche à l’utilisateur les éléments permettant d’utiliser la messagerie : le bouton de déconnexion, le chat, l’endroit où entrer le texte et le bouton pour envoyer le message.

Si le message indique qu’une erreur est survenu, on affiche un message d’erreur et l’utilisateur doit réessayer de se reconnecter. Dans tous les cas, on permet à l’utilisateur d’utiliser à nouveau le bouton d’envoi, qu’on désactive avant pour éviter le spam d’information.

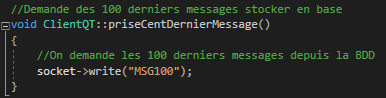
La fonction d’inscription est sensiblement la même chose :



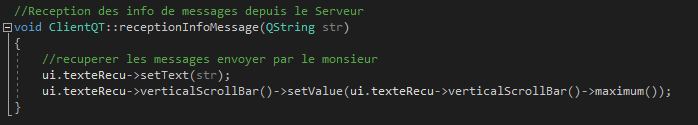
Si le serveur confirme que le compte a été créer, on renvoie l’utilisateur vers la page de connexion pour qu’il puisse s’y connecter.

Si le compte existe déjà, l’on reçoit un message d’erreur et l’utilisateur doit recommencer.

N’ayant pas trouvé comment communiquer autrement que comme cela (puisque les serveurs ont été long à mettre en place), lorsque le client reçoit le « MSG100 », cela signifie que le serveur lui redemande de lui envoyer « MSG100 » pour pouvoir envoyer les 100 derniers message à tous les clients. Cette partit peux très certainement se mettre en place bien mieux, mais faute de temps cela nous a semblé la meilleure façon de faire.



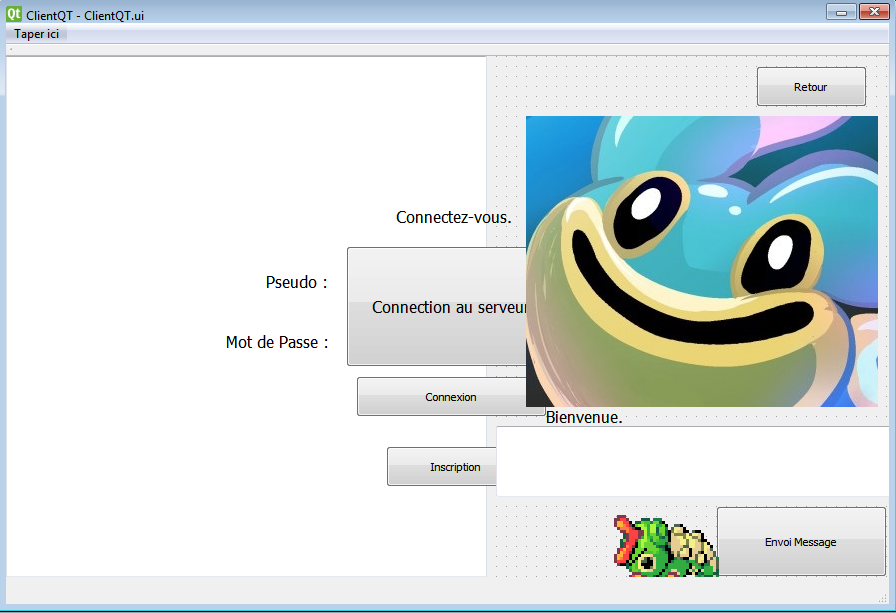
Lorsque le client reçoit les messages stocker en base, ils les inscrit dans la partit servant de chat. Pour pouvoir voir les derniers messages en premier, on fait baisser la Scroll Bar au maximum.



**Problème rencontrer**

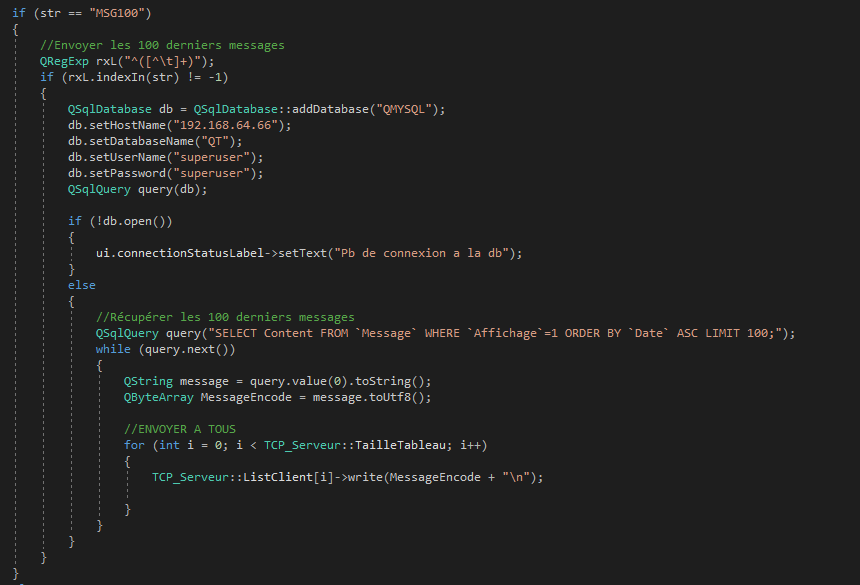
Au niveau du code, avec un peu de recherche, je n’ai pas eu de grosse difficulté. Il n’y a que les options de QT qui m’ont posé des problèmes pour les mettre en place (cacher le texte pour les mots de passe, effacer la zone de texte lorsque le message a été envoyer, …) et la gestion des éléments à cacher ou non.

Pour vous donner une idée, voici à quoi ressemble le client dans l’application QT.

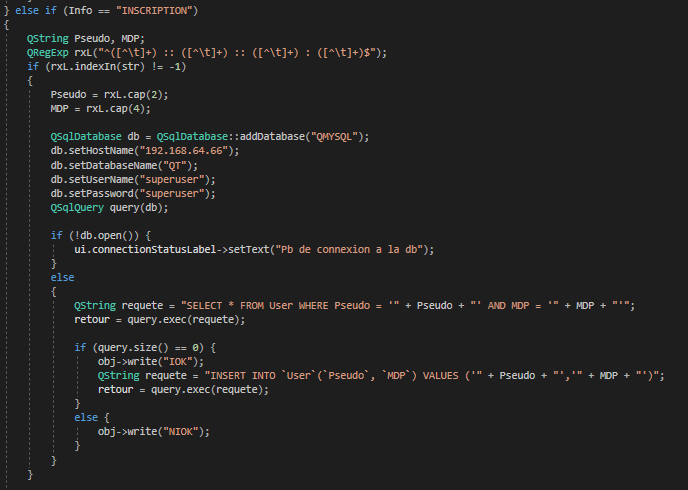


Les vrais problèmes ont été dans la coordination du groupe, et ce malgré le Trello mis en place. Kylian n’a travaillé sur le WebServeur que lors de la dernière semaine, mettant à mal le projet entier, puisque l’on avait besoin d’un serveur fonctionnel pour tester et, si les bases fonctionnaient, pouvoir prendre du temps pour développer d’autres fonctionnalité, comme l’administration ou la gestion du compte utilisateur (photo de profil, changement de mot de passe, …).

**COLSON Baptiste**

Pour ce qui est du serveur TCP, le Client QT doit m’envoyer un login et un mot de passe sous un format précis.

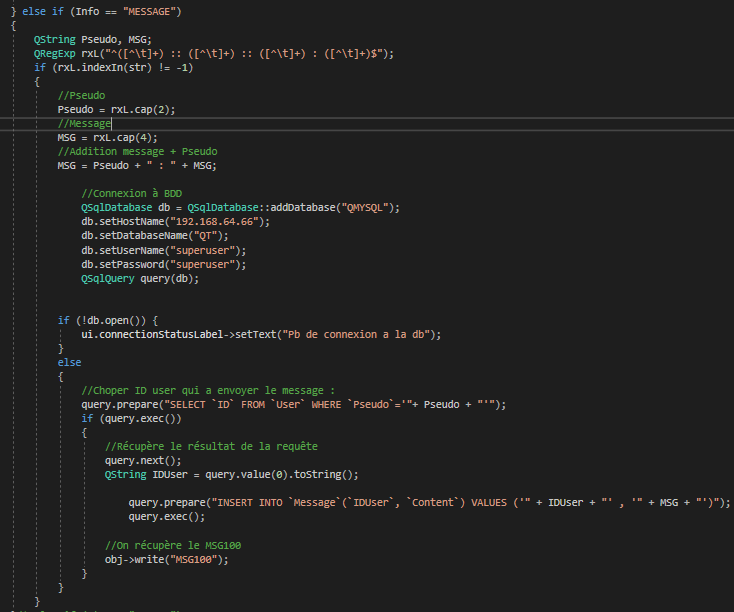
Avec du Regex, j’ai dû identifier le login et le mot de passe avec la requête « QRegExp rxL (« ^([^\t] +) ») ». J’ai dû connecter la BDD pour pouvoir vérifier si le login et le mot de passe correspond à celui de la BDD.

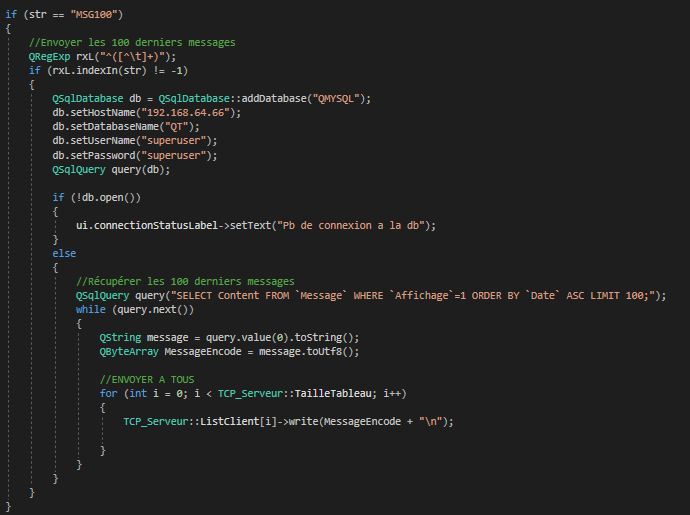


Si le login est bien dans la BDD j’envoie LOK au Client QT pour dire que la connexion est réussie. Si la connexion n’est pas réussie j’envoie NLOK.

Pareil pour l’inscription, je vais avoir un format spécial pour l’inscription que je vais devoir identifier avec une requête Regex. Si l’inscription est réussie, j’envoie IOK au Client QT et si l’inscription n’est pas réussie j’envoie NIOK pour dire que l’inscription n’est pas réussie.

Pour savoir qui envoie le message, je vais associer l’id de l’utilisateur à son message et mettre le message dans la BDD. Puis j’envoie MSG100 au Client QT pour que le Client QT demande au TCP Serveur les 100 derniers messages. On aurait pu utiliser une autre méthode mais nous n’avons pas eu assez de temps.





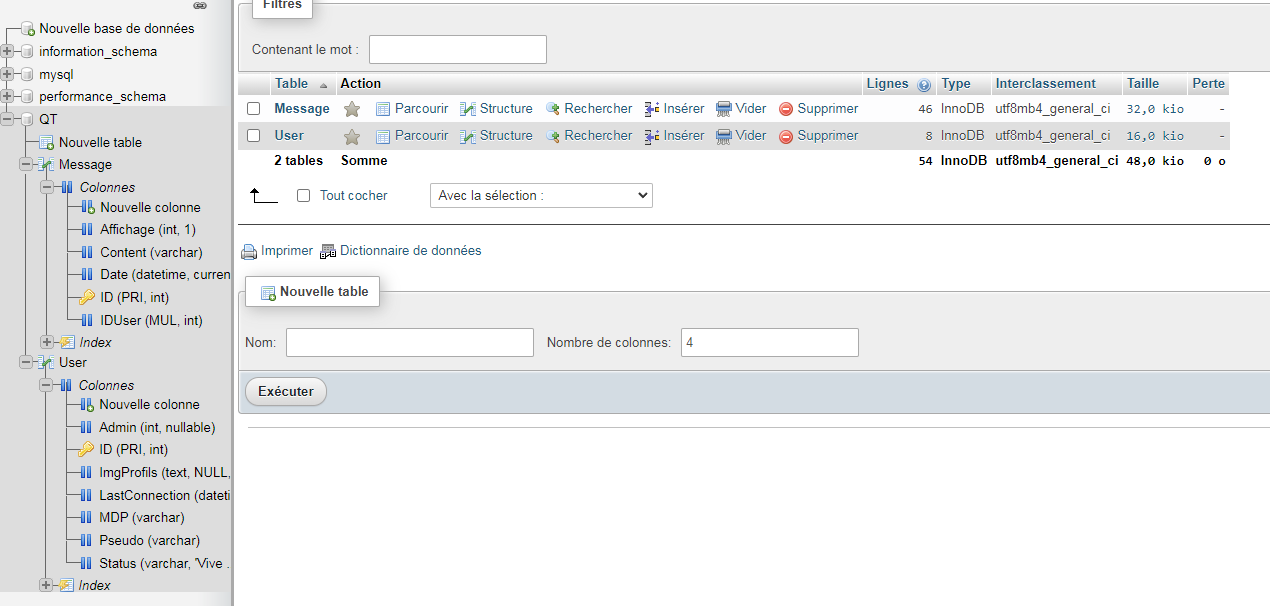
Problématique rencontrer

J’ai rencontré un problème pour identifier le login et le mot de passe mais j’ai pu résoudre ce problème avec une requête Regex.

**Duval Kylian**

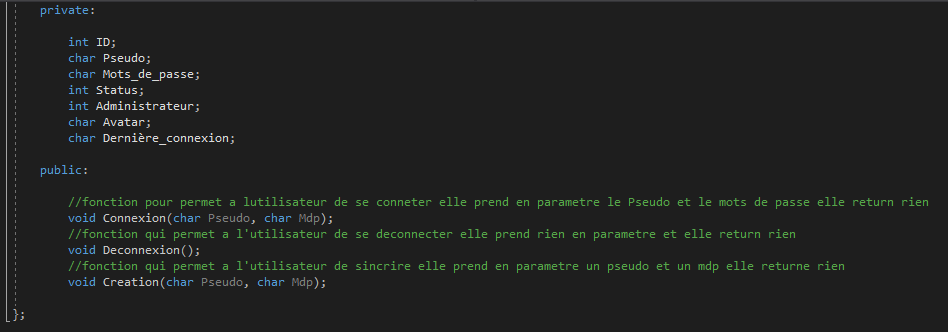
Moi je me suis occuper de configurer la VM de créer la base de donner, de créer les structures class User, Message, Admin et enfin je me suis occuper de faire le websocket Server

La base de donner :

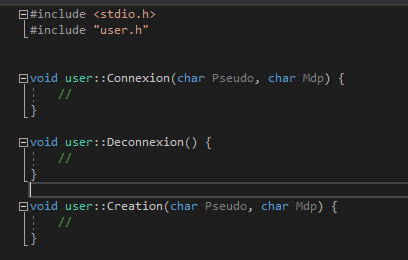


Structures class User :

user.h

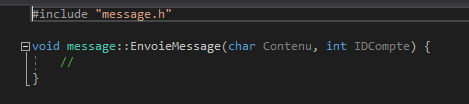


User.cpp

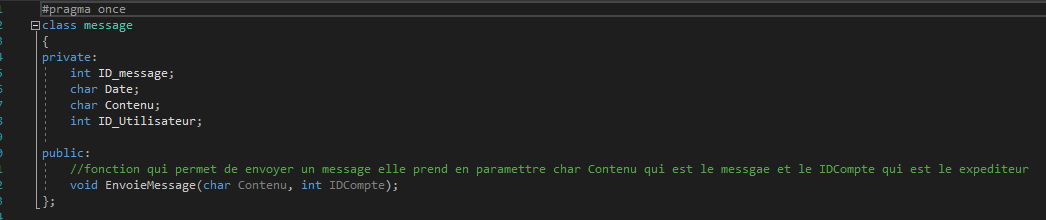


Structures class Message :

message.cpp

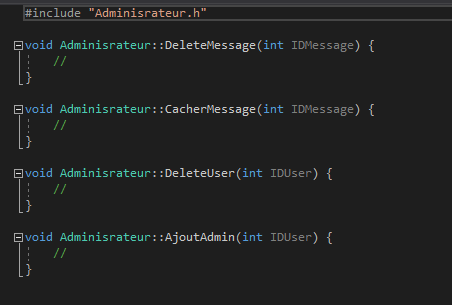


message.h

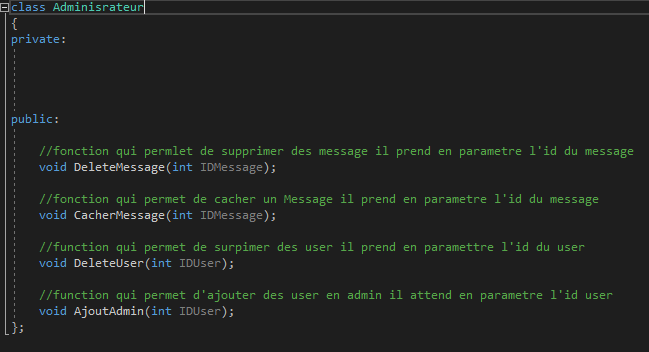


Structure class Administrateur :

Administrateur.cpp



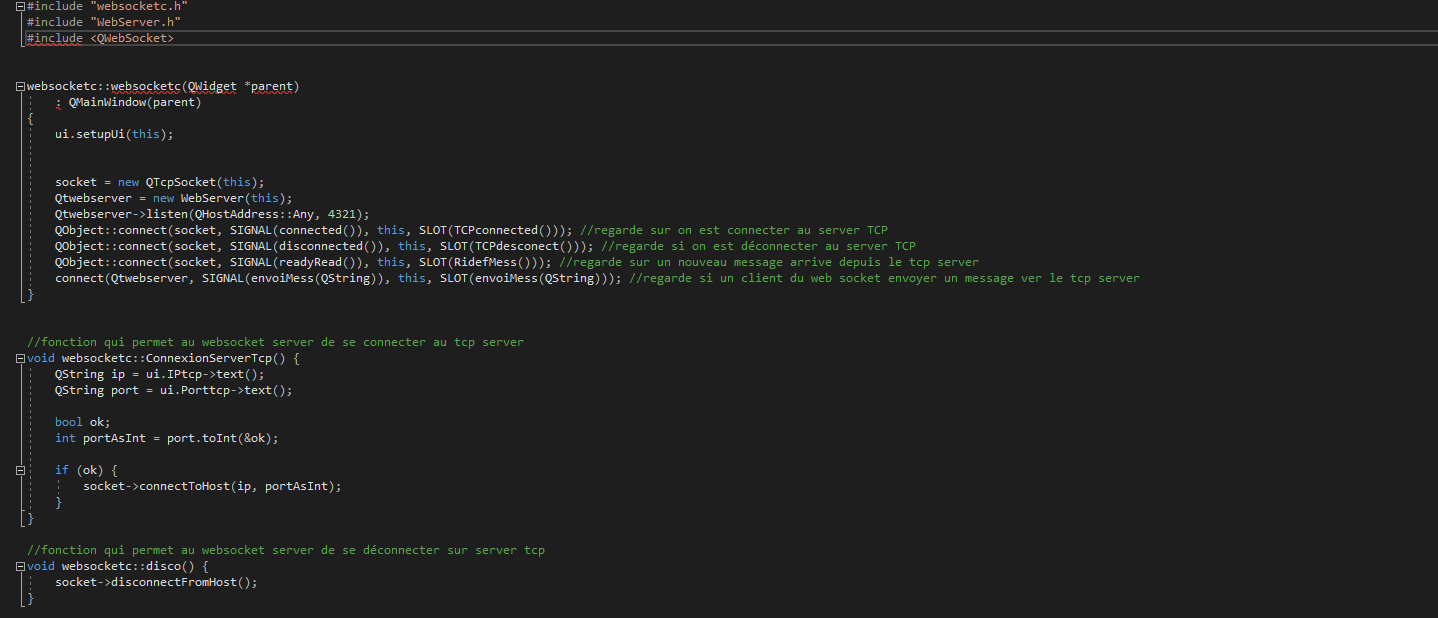
Administrateur.h

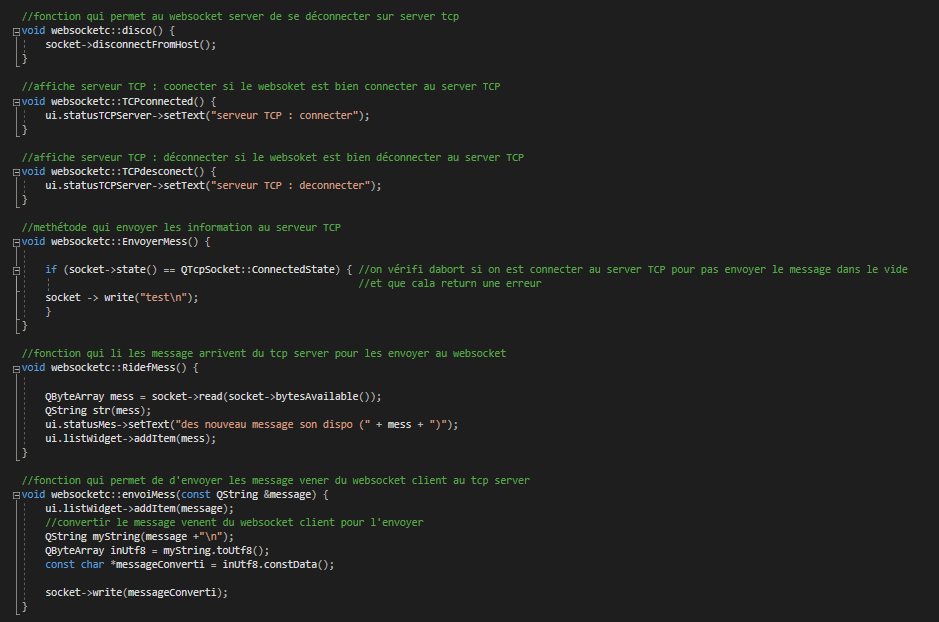


websocket Server:

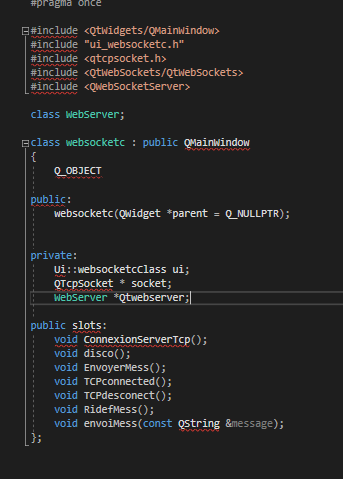
Class websocket:

websocketc.cpp



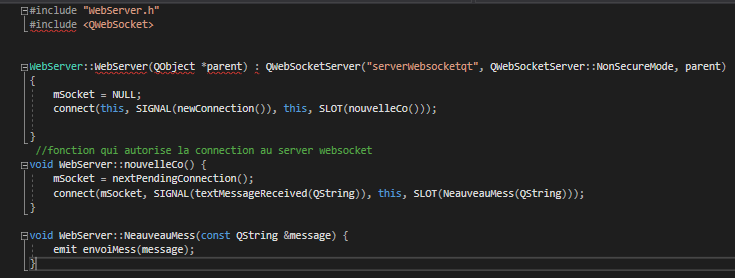


websocketc.h

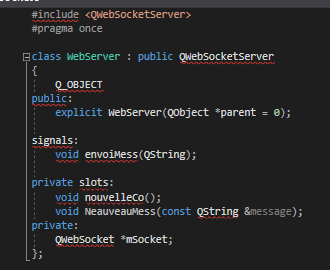


Class WebServer :

WebServer.cpp



WebServer.h



Le problème que j’ai rencontré c’est que je n’arriver pas à renvoyer les message envoyer par le tcp server pour les envoyer au websocket client JavaScript pour résoudre ce problème il faut récuperer les connection des webscoket client avec nextPendingConnection(); ensuite que je stock la connexion du client dans une qlist ensuite que je fasse sendTextMessage sur tous les connexion socket dans le Qlist. Mais je n’ai pas pu le mettre en place mais sinon le client arriver à envoyer des choses au tcp server il manquer juste le renvoi des donner du websercketserver au websocket client

**Pour conclure** le projet ma permit de voir des nouveaux choses comme le websocket server et le websocket client que je ne connais pas avant de faire ce projet se projet ma permit aussi de m’améliorer avec QT et en C++ en général et cala ma aussi permet de voir le JavaScripts car j’ai créé mon propre websocket client pour tester le websocket server

**Documentation technique de la partie Web.**

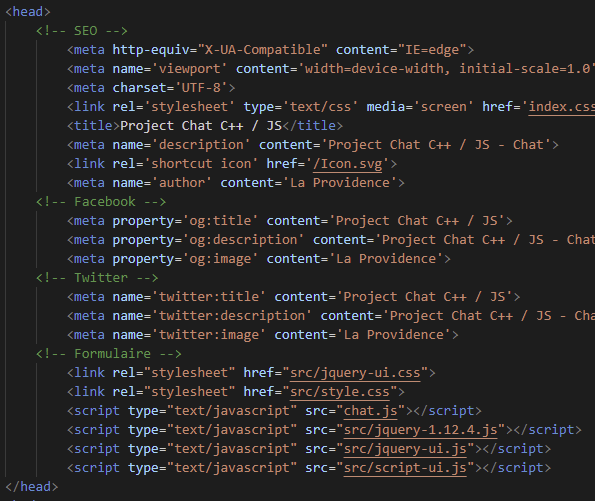
**Par Nicolas de Almeida**

1. **Page HTML Index.html**

Contient l’ensemble du HTML Fixe de la page.

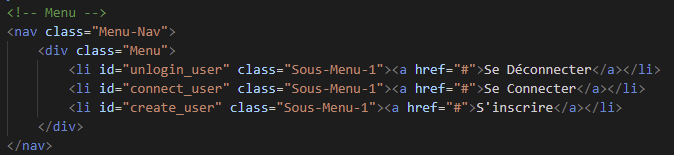
* Le header

Qui me permet de définir les différentes propriété SEO ainsi que d’inclure le Javascript et CSS.



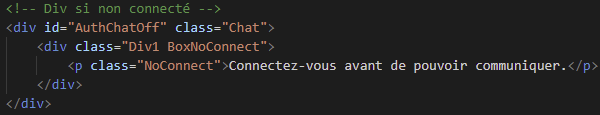
* Mon menu (HTML)

Qui me permet d’afficher un Menu.



* Div si utilisateur non connecté

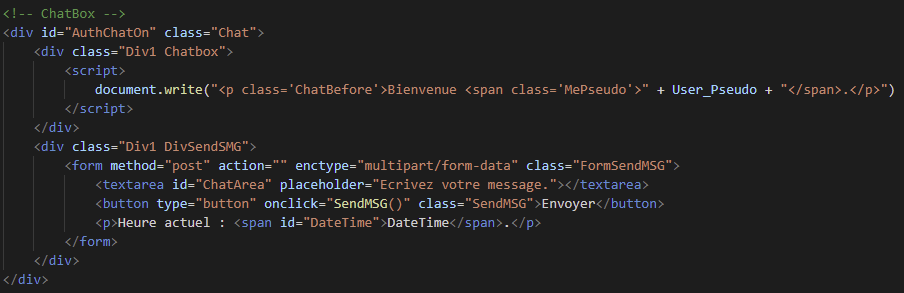
Ce qui est affiché si l’utilisateur n’est pas connecté.



* Ma Chatbox si utilisateur connecté

Chatbox qui permet de communiquer via le JS avec la fonction “SendMSG()”.

Elle contient plus bas le span avec l’id “DateTime” qui contient l’heure via une fonction JS.



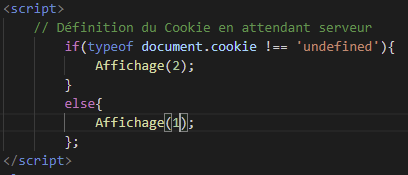
* Formulaire HTML de l’Inscription et Connexion

Contient tout l’HTML des Formulaires.



* Définition de l’affichage.

Si le cookie n’existe pas (et dont que l’utilisateur n’est pas connecté) cela permet d’afficher le code de l'inscription et connexion, et si le cookie existe alors il affichera le chat.

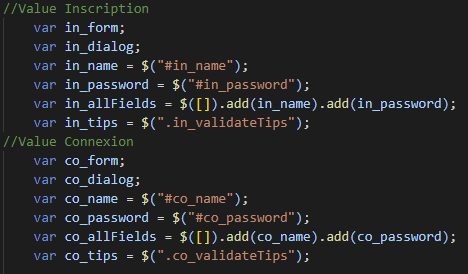


1. **Page JavaScript src\script-ui.js**

Contient l'ensemble du JavaScript concernant l’inscription et la connexion, afin de rendre ça fluide et responsive, mais aussi de vérifier que les informations rentrées correspondent à ce qui est demandé.

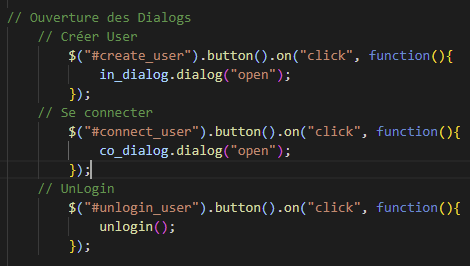
* Initialisation de mes valeurs JS.

Initialisation de mes valeurs JS en lien avec les textareas correspondant aux class et id correspondant.



* Fonction Activant create\_user / connect\_user / unlogin\_user

Activant les fonctions correspondant au clic sur les éléments possédant l’id requis, et permettant ainsi d'afficher les formulaires ou de se déconnecter.



* Fonction permettant de se connecter ou s'inscrire.

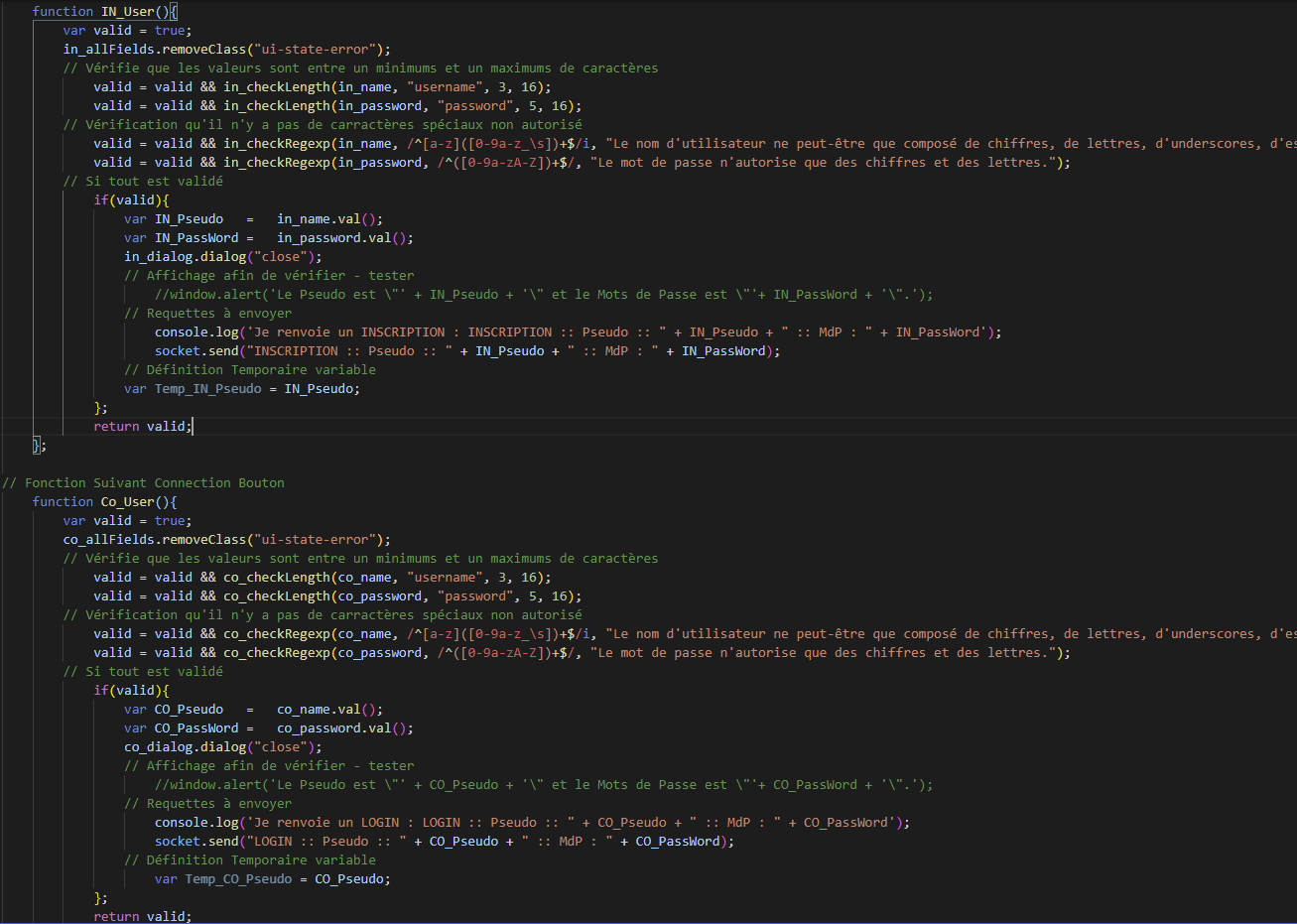
Les deux suivent la même logique et seul les noms de variables sont différents.

En premier lieu on initialise une valeur “valid” qui est par défaut true et l'on retire la class erreur si elle est présente sur l’UI.

Ensuite on vérifie si les informations rentrées sont bien comprises dans l'intervalle de nombre de caractères demandés.

Ensuite on vérifie via une fonction regex si les informations rentrées ne contiennent pas des caractères que spécifique que l’on refuse.

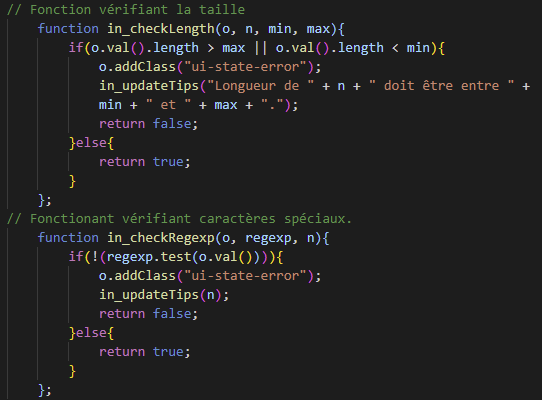
Si cela est encore valid, alors on récupère les valeurs et on les communique au serveur via le web socket et la requête correspondante.



* Fonction Vérifiant la taille et la présence de caractères non requis.

Vérifiant la taille de la chaîne de caractère.

Vérifiant la présence de caractères non désirés.



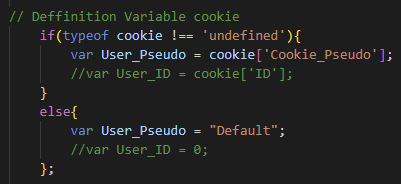


1. **Page JavaScript chat.js**

Contient l'ensemble du JavaScript concernant le chat ainsi que la communication des informations concernant le chat.

* Définition du Pseudo.

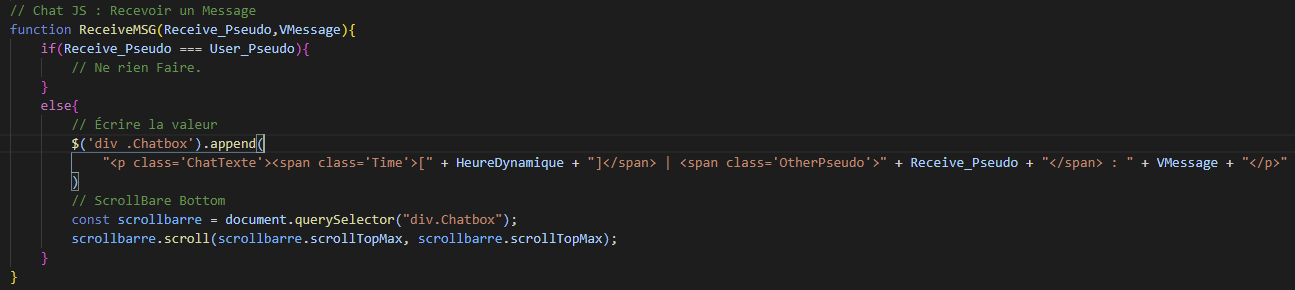
Si un cookie existe, on définit son pseudo en conséquence, sinon on lui attribue un pseudo par défaut.



* Fonction permettant d’ajouter dans le chat les messages reçus.

Ajoute dans le chat les messages reçus des personnes qui n’ont pas le même pseudo que l’utilisateur.

Puis définit la scrollbar en bas.



* Fonction d’envoie de message.

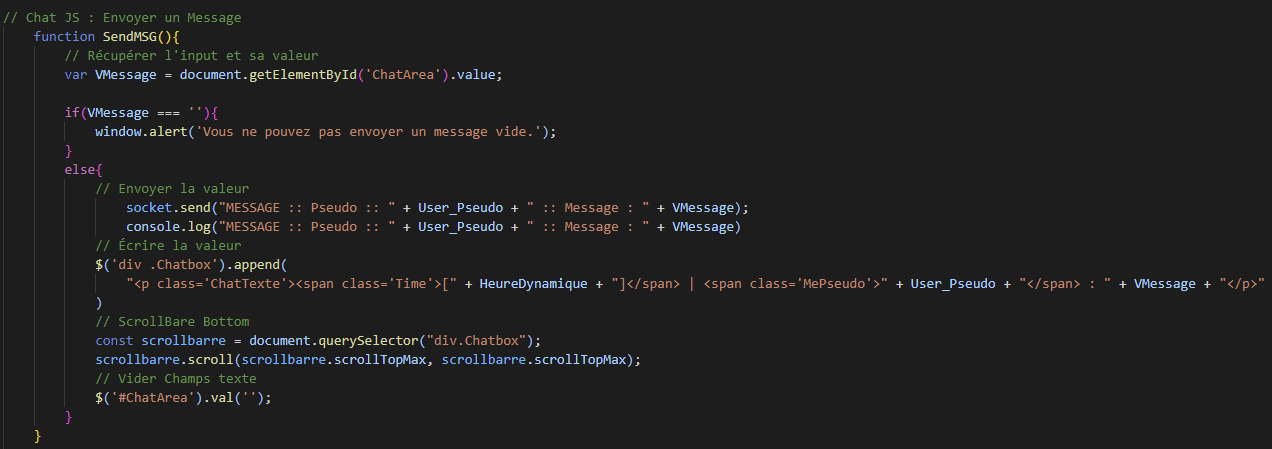
Permet d’envoyer des messages au serveur.

En premier lieu, l'on récupère la valeur du textearea.

Si la valeur est vide on affiche un message refusant l'envoi.

Si la valeur n’est pas vide, on envoie donc cette valeur via webstocket dans une requête qui correspond à ce que demande le serveur.

On affiche ensuite le message en question dans la chatbox avec l’heure et le pseudo, l’on définit la scrollbar en bas, puis l’on vide le textearea.



* Fonction d’affichage

Permet d’afficher ou non les parties qui doivent être affichées ou non, selon l’authentification de l'utilisateur.



* Fonction d’affichage de l’heure de manière dynamique

Fonction affichant l’heure de manière dynamique sur le site, avec un objet Date, et permettant aussi de récupérer la valeur lors de l’envoie et de la réception des messages dans le chat. La fonction se lance seule lors du chargement de la page.



* Fonction de connexion entre le site Web et le serveur Websocket.

On définit l’ip du Websocket qui correspond pour nous connecter.

On lance une écoute et l'on informe le serveur de l’écoute en le saluant.

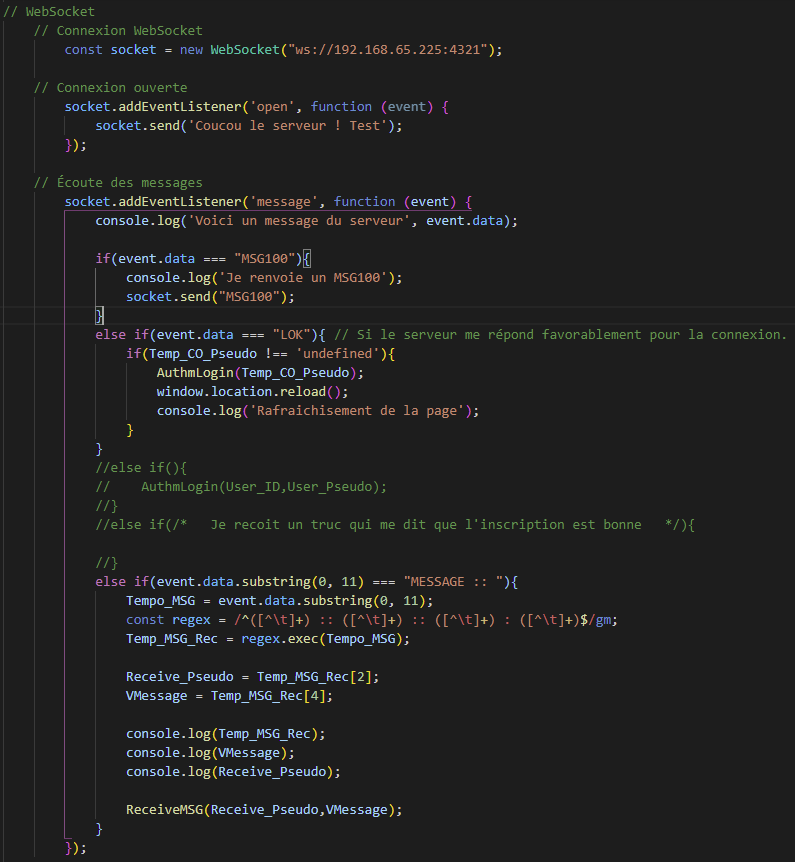
Lorsqu’un message arrive, l’on regarde sa valeur.

si c’est un “MSG100” l’on retourne un “MSG100” pour informer le serveur que le message est bien reçu et envoyé.

Si c’est un “LOK” c’est que le serveur nous confirme que l’authentification est bonne et dans ce cas l’on définit le pseudo temporaire comme pseudo valide et l’on refresh la page en s'authentifiant.

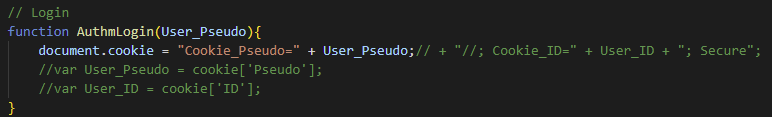
S’il commence par “MESSAGE” c’est un message, et l’on utilise donc une fonction regex pour obtenir la valeur du pseudo et du message.

L’on définit le pseudo et le message puis l’on lance la fonction ReceiveMSG() qui permet de l’afficher dans le chat du web.



* Fonction AuthmLogin

Fonction créant un cookie, utilisé ensuite pour récupérer le pseudo et savoir si l’utilisateur est connecté.



* Fonction Unlogin

Fonction permettant de se déconnecter. Définit le cookie avec une date d'expiration dépassée permettant une auto suppression dudit cookie par le navigateur, l’on relance ensuite la page.

