

Rapport de fin d'année

LockVox

Projet scientifique et Technique de 4^{ème} année



Suiveur

LE BERRE Matthieu

Membres

BOURGEAIS Louis

LAVEAU Hugo

LELIEVRE François

MATTEI Hugues

MONNOT Valentin

Table des matières

Introduction	4
Livrable final	5
Serveur	5
Client	5
 Déroulement du projet	6
Mise en place & répartition des rôles	6
Base de données	6
Interface Utilisateur	7
Protocole réseau applicatif	8
Les rôles (NIY ¹⁸)	8
Communication vocale (NIY)	9
Images	10
Communication textuelle	11
Journalisation d'évènements	12
Sécurité	13
 Problèmes rencontrés	14
Audio	14
Message	14
Découpage de paquets	14
Interface Graphique	15
 Axes d'améliorations	16
 Contributions & Avis des membres du projet	17
BOURGEAIS Louis	17
LAVEAU Hugo	17
LELIEVRE François	18
MONNOT Valentin	19
MATTEI Hugues	20
 Conclusion	21

Annexes	22
ETAT DE L'ART :	27
LOGICIEL DE TELECOMMUNICATION	27
Serveur :	27
Choix :	28
Option et personnalisation de compte	28
Création d'un compte	28
Personnalisation d'un compte	29
Eléments techniques	30
Choix :	32
Salons de communication	32
Qu'est-ce qu'un salon ?	32
Discord :	32
Salons textuels	33
Salon vocal	34
Fonctionnalités	34
Choix	35
Interface graphique	35
Analyse	35
Choix	38
Glossaire	39

Introduction

Face à la situation sanitaire actuelle, l'utilisation de logiciel de télécommunication VoIP¹ a explosé. En effet la plupart d'entre eux malgré leurs qualités esthétiques et fonctionnelles pratiquent la récupération et la revente de données à des fins commerciales.

De plus les principaux logiciels sont privés, il est par conséquent impossible de venir modifier le code source pour y ajouter des fonctionnalités ou des modifications graphiques personnelles.

Soucieux de nos données, et de la possibilité de modification nous souhaitons utiliser un logiciel de télécommunication open source². La seule solution étant Mumble. Néanmoins cette solution commençant à prendre de l'âge et ne propose de solution simple et rapide de déploiement opérationnel. De plus l'interface ne correspond plus aux standards du marché, n'étant pas forcément facile d'accès pour les novices.

C'est dans le but de répondre à ces contraintes que nous avons souhaité développer notre propre logiciel de communication open source et libre de droits qui se nomme LockVox.

Livrable final

Le projet fini sera sous forme de deux programmes. L'un est le serveur qui peut être installé sur la machine souhaitée. Les informations sauvegardées ou transmises entre utilisateurs sont actuellement non chiffrées, mais il est prévu de renforcer la sécurité par la suite. L'autre sera une partie client nécessaire aux utilisateurs pour rejoindre le serveur pour pouvoir profiter des différentes fonctionnalités.

Serveur

Le serveur pourra être installé par n'importe quel utilisateur, ce dernier devra être connecté à une base de données dont le diagramme sera précisé par la suite.

L'arborescence du serveur est définie de la manière suivante :

- Storage
 - Log
 - Ensemble des fichiers de journalisation (à chaque lancement du serveur, un nouveau fichier de journalisation est créé).
 - Private
 - Une liste de répertoires personnels pour chaque utilisateur qui se nomme avec l'UUID³ de l'utilisateur. On y retrouve actuellement un fichier « pp.png » correspondant à la photo de profile de l'utilisateur en question.
 - Public
 - Une liste de répertoires publics pour chaque salon qui se nomme avec l'identifiant du salon contenant un index au format JSON⁴ répertoriant tous les noms de fichiers ainsi leurs positions dans l'ordre chronologique des messages enregistré au format XML⁵ présent dans ce même dossier.
 - Server
 - pp – Répertoires contenant des photos de profil par défaut attribué aux utilisateurs n'ayant pas de photo de profile dans leur répertoire privé.

Client

L'application client sera utilisée par les utilisateurs, il faudra préciser l'adresse ou le nom de domaine du serveur auquel l'utilisateur souhaite se connecter. Par la suite, nous voudrions rendre possible le fait que l'application puisse gérer plusieurs serveurs simultanément.

Afin que vous puissiez mieux comprendre l'architecture des deux applications, nous avons mis à votre disposition un diagramme en annexe qui la détaille plus en profondeur (cf. Figure 1).

Déroulement du projet

Mise en place & répartition des rôles

Nous avons commencé par mettre en place un repository GitHub⁶ afin de pouvoir travailler ensemble et en parallèle sur le projet. Ce Git pourra être utilisé à l'avenir par des utilisateurs voulant prendre part au projet pour des améliorations ou des forks⁷ du projet, de plus les versions officielles seront disponibles sur ce dernier.

Nous avons choisi comme langage de programmation le C++ en combinaisons avec le Framework⁸ Qt. En effet nous avons étudié le C++ à l'école, il y a énormément de documentation en libre accès sur Internet et c'est un langage compatible avec le Framework. Qt a été sélectionné, car c'est l'un des outils de création d'applications graphique le plus réputés, de plus ce dernier est libre de droits pour les projets open source et à but non lucratif. Ce qui est en accord avec l'idée de notre projet même s'il n'est pas open source et qu'il existe une version payante plus complète et non libre de droits pour les programmes privés.

Qt permet de créer des interfaces et d'y intégrer les fonctionnalités de notre code de manière extrêmement simple. De plus, ce dernier propose un outil graphique permettant de réaliser des interfaces utilisateur avec un minima de code.

Nous avons choisi d'utiliser le langage de programmation C++ afin de développer notre application. Ce choix a été fait notamment pour des raisons pédagogiques, le C++ étant un langage de programmation bas niveau, il est nécessaire de comprendre certains concepts comme les sockets⁹ ou les threads¹⁰ plus en profondeur. Heureusement, par sa maturité, la documentation disponible ainsi que la grande communauté permettent d'avancer et d'obtenir de l'aide au besoin.

Base de données



Afin de répondre aux enjeux de sécurité de l'application, nous devons mettre en place un système de salage¹² sur un deuxième serveur distant. Néanmoins le projet étant déjà très conséquent nous avons préféré nous concentrer sur l'aspect fonctionnel en priorité.

Interface utilisateur

Pour créer l'interface utilisateur, nous avons utilisé l'outil de Qt Design Studio qui est un outil de conception et de développement d'interface graphique inclus dans Qt.

Qt design studio fonctionne de la manière suivante, l'outil possède une fenêtre d'interface graphique où l'on peut venir placer des widgets¹³ et un code QML¹⁴ est généré en temps réel.

On peut venir charger des widgets que l'on a créés directement dans Design Studio ce qui nous a permis de pouvoir créer et d'importer toute une palette de widget personnalisés, nous avons utilisé cette méthode pour l'affichage des clients, des messages et des salons.

Design Studio possède aussi un système d'état. On peut venir créer différents états au sein d'un widget. En fonction de l'état dans lequel on est, on peut venir faire des modifications graphiques et modifier différentes propriétés des widgets contenues dans la fenêtre. Avec cette méthode on peut créer différentes fenêtres au sein d'une même fenêtre, et donc avec une seule fenêtre pouvoir afficher une page de connexion, de création de comptes, la fenêtre principale et la fenêtre de paramètre.

Design Studio offre la possibilité d'ancrer des éléments à d'autres éléments (élément parent). On a donc utilisé ce système afin de pouvoir augmenter la taille de notre fenêtre et faire en sorte que tous les éléments de la fenêtre s'adaptent lors du changement de taille.

Nous avons créé des classes C++ pour les clients et les salons afin de pouvoir stocker leurs informations, dans le but de les afficher dans le QML. Nous avons implémenté¹⁵ des classes modèles de listes de clients et de salons. Ces modèles permettent de définir dans le QML la manière dont doivent être lues les classes clients et salons. Nous devons donc par la suite, juste donner au QML nos listes et il pourra venir lire ces listes et les afficher.

L'intérêt d'utiliser le QML est aussi que l'on peut venir récupérer des valeurs présentes dans le C++ et venir les charger directement dans le QML et inversement.

Pour venir chercher des valeurs des variables on a donc créé on utilise des fonctions en C++ qui viennent chercher directement des widgets QML pour les charger dans le code. On peut alors charger et changer l'affichage d'un widget QML en fonction de la valeur d'une variable.

On peut venir faire l'inverse en créant un système de slots et de signaux¹⁶, lors de l'appui sur un bouton dans la fenêtre QML, un signal du widget qui possède une certaine valeur est envoyé vers un slot d'une fonction qui va directement modifier au sein du code la valeur d'une variable.

Cette solution nous a donc permis d'effectuer des changements en temps réel de l'interface par exemple lors de l'envoi de message ou lors du changement de nom d'utilisateurs.

Protocole réseau applicatif

Définition :

La couche d'application représente des données pour l'utilisateur ainsi que du codage et un contrôle du dialogue : des mécanismes de communication offerts aux applications de l'utilisateur.

En d'autres termes, la couche d'application ou le protocole applicatif peut être vu comme le langage utilisé par le client et le serveur afin de communiquer. Les deux applications ont besoin de faire de se poser des questions et d'être capable de comprendre la réponse. C'est à cela que sert le protocole applicatif.

La première étape dans sa réalisation est de se poser la question : « De quelles informations le client et le serveur auront-ils besoin de se communiquer et quelles questions vont-ils pouvoir se poser ? »

Nous avons donc défini une liste des différentes requêtes, qui pourront être émises soit par le client soit par le serveur (cf. Figure 5).

Notre protocole sépare ces requêtes grâce à une action et à un type. Le type définit la catégorie de la requête (Serveur, salons, utilisateur, message, audio, autre) et l'action représente... l'action que l'utilisateur veut faire. Par exemple, lorsqu'un utilisateur souhaite modifier son mot de passe, on renseigne le numéro de type et d'action correspondant, ainsi lorsque le serveur reçoit la requête, il aura la capacité de comprendre l'action que l'utilisateur souhaite réaliser.

Néanmoins, l'action et le type ne suffisent pas à la requête, nous devons aussi renseigner les nouvelles données à changer, ou alors l'utilisateur ou le salon à qui appliquer l'action si cela ne concerne pas un changement de données. Pour cela, nous utilisons la sérialisation¹⁷ et la désérialisation avec le format JSON, car il nous permet de fonctionner avec des objets et qu'il est simple à utiliser.

Afin de nous faciliter la vie, nous avons décidé de sérialiser nos objets de manière générique. C'est-à-dire que lorsque l'on souhaite, par exemple, modifier le mot de passe d'un utilisateur, nous allons alors sérialiser la globalité de l'objet de type « Client », ce qui comprend son nom, sa description, son UUID, etc. Plutôt que de sérialiser uniquement son UUID et son nouveau mot de passe. Encore une fois, nous avons fait ce choix pour la praticité, en effet, cela nous permet de ne pas avoir à gérer la sérialisation au cas par cas en fonction des requêtes effectué et donc d'utiliser la même sérialisation à plusieurs reprises.

Les rôles (NIY¹⁸)

Les rôles définissent un ensemble de droits. Ils sont attribués aux salons et aux utilisateurs. C'est-à-dire qu'un rôle qui donne le droit de parler et d'écrire ne s'appliquera que si l'utilisateur ET le salon dans lequel il est se sont vu attribuer ce rôle.

Les rôles seront gérés par arborescence, c'est-à-dire que chaque rôle pourra avoir un parent et plusieurs enfants. Un enfant hérite par défaut des droits de ses parents dans les salons ou les rôles

Communication vocale (NIY)

La communication vocale pourra s'effectuer par deux moyens, tout d'abord par les salons prévus à cet effet, et deuxièmement par les appels privés entre utilisateurs. À ces appels, seront intégrés différentes fonctionnalités comme pouvoir couper le son d'un utilisateur, gérer indépendamment le volume des différents utilisateurs...

Pour implémenter l'audio, nous avons choisi la bibliothèque¹⁹ EMIPLIB - EDM Media over IP Library.

Cette bibliothèque nous permet plusieurs choses. Premièrement, la gestion en temps réel des trames audio grâce au protocole RTP²⁰. En effet l'audio nécessite du RTP pour éviter les délais et, car cela nécessite de mixer les trames audios des différents utilisateurs qui arrivent en temps réel.

Elle gère aussi nativement les périphériques audios pour Windows, Linux et aussi des bibliothèques externes comme PortAudio.

Cette bibliothèque nous permet aussi d'utiliser le SRTP qui est la version sécurisée du RTP, et donc de sécuriser les communications vocales.

Pour l'utiliser, on crée des chaînes de composants qui vont chacun traiter l'audio pour l'envoyer aux composants suivants et ainsi de suite, voici comment la chaîne que nous avons créée fonctionne :

Client :

- Le client envoie le signal de son entrée micro.
- Le message sous forme d'entier de 16 bits est converti sous forme de float²¹.
- Le message est ensuite filtré afin d'éviter les désagréments liés à un possible effet Larsen²².
- L'audio est encodé avec μ -Law²³
- Encodée spécifiquement pour le protocole RTP puis envoyé au serveur.
- Le client reçoit les différents flux de voix.
- On décode le format RTP des paquets.
- Les messages passent dans un mixeur afin de pouvoir entendre tous les gens dont le micro n'est pas coupé.
- On décode les messages sous forme μ -Law afin de pouvoir les retranscrire sur la sortie audio.

Serveur :

- Le serveur synchronise les messages des différents utilisateurs.
- Il renvoie l'audio aux utilisateurs.

Nous souhaitons implémenter à terme une option qui donnera le rôle au serveur de mixer les différentes sources audios, afin de permettre moins de traitement et une plus faible utilisation de la bande passante côté client.

Images

Nous avons mis en place une gestion d'images sur notre serveur. Dans un premier temps pour les photos de profile des utilisateurs, puis en les intégrant dans les messages. L'intégration des images dans les messages est développée, mais n'est pas encore mise en œuvre.

Nous avons décidé d'utiliser le format Base64²⁴ nous permettant de transformer les images en une chaîne de caractères utilisable pour l'envoi de donnée entre serveurs et clients. Au lancement du serveur, il vérifie si tous les utilisateurs ont une photo de profile dans leur arborescence. Si ce n'est pas le cas, les utilisateurs sans photos de profile se verront attribuer une photo de profile aléatoire parmi celles disponible sur le serveur. La photo de profile aléatoire sera enregistrer dans l'arborescence de l'utilisateur en question et sera utiliser dans le futur.

Lorsqu'un client se connecte, il fait des requêtes auprès du serveur pour récupérer les photos de profile de tous les utilisateurs présents sur le serveur durant son chargement. Le serveur répond en envoyant toutes les photos de profile au format Base64. Le client les reconstruit et les alloue aux l'utilisateur concerné afin de pouvoir les afficher par la suite.

Communication textuelle

Pour la communication textuelle, on a mis en place des salons des textuels, qui permettent d'envoyer des messages.

Nous voulons aussi proposer un système d'envoi de messages privés entre deux utilisateurs. Nous avons développé notre système de messagerie en partant de zéro et l'avons intégré au protocole applicatif.

Nous nous sommes servis du standard XML pour définir l'architecture de nos messages. Chaque message est envoyé par le réseau au format XML avec des champs rappelant l'utilisateur à l'origine du message, le destinataire, la date et l'heure d'envoi, un champ nous permettant de savoir si le message est privé ou destiné à un salon textuel public ainsi qu'un champ image qui peut être vide. Le serveur reçoit le message, vérifie son intégrité, l'enregistre dans son arborescence au format XML, l'envoie à tous les utilisateurs connectés y compris à l'expéditeur (ce qui lui permet d'être sûr que le message a bien été envoyé)

L'enregistrement du message côté serveur est relativement simple, le nom donné au fichier XML correspond au hachage²⁵ en sha256 du contenu XML complet permettant d'être sûr d'avoir un nom de fichier unique et permet de détecter les spams rapides de copier-coller. Les messages sont stockés dans un dossier du nom de salon textuel visé (pas encore de gestion des messages privés) et sont répertoriés dans l'ordre croissant par un fichier d'index au format JSON.

L'insertion d'image dans les messages est implémentée, fonctionnelle, mais n'a pas encore été intégrée à l'interface utilisateur. On traite les images de la même manière que les photos de profil, au format Base64 et insère dans la version XML du message.

Côté client, les messages au format XML sont chargés en mémoire en utilisant des variables dédiées permettant d'accéder à chaque champ des messages indépendamment des besoins.

Journalisation d'évènements

Afin d'avoir une gestion convenable des évènements, nous avons mis en place un système de journalisation d'évènements côté serveur. Comme vue plutôt dans la partie de l'arborescence, les fichiers de journalisation se trouvent dans le dossier log du serveur. Un fichier de journalisation au format texte est généré à chaque lancement du serveur. Dans ce document, nous allons retrouver différentes informations sur quatre niveaux de sévérités.

0 – Journalisation des évènements standard :

- Connexion
- Déconnexion
- Modification du mot de passe
- Modification du pseudo
- Modification de la description
- Modification de l'adresse électronique
- Etc.

1 – Journalisation des avertissements serveur :

- Réception de donnée d'un client inconnu
- Problèmes de paquets, problèmes de désérialisation
- Utilisateur déjà connecté essayant de se reconnecter
- Mot de passe incorrect à la connexion
- Tentative de connexion à un compte non existant
- Tentative de création de comptes avec un mail déjà enregistré
- Paquet invalide
- Tentative de connexion à un salon non existant

2 – Journalisation des erreurs serveur :

- Impossibilité de démarrer le serveur
- Problème avec la base de données
- Problème de création d'utilisateurs
- Paquet de déconnexion alors que le client n'est pas connecté
- Message pour un salon ou un client inconnu
- Impossibilité d'ouverture ou de création de dossiers ou de fichiers
- Impossibilité d'enregistrement de message
- Impossibilité d'ouverture de message ou d'index de message
- Erreur de requête de photo de profile

3 – Journalisation des erreurs de base de données :

- Toutes les erreurs de requêtes ou de connexion

Sécurité

Un logiciel de communication digne de ce nom se doit d'intégrer de la sécurité afin de protéger les informations de ses utilisateurs. En effet, nous devons assurer à tous les utilisateurs que les données qu'ils nous confient ne pourront pas être interceptées par une personne tierce, sans cela, c'est la crédibilité même du logiciel qui serait mise en cause. Ces sécurités garantissent la confidentialité, l'intégrité, la disponibilité des services, l'authenticité ainsi que la non-répudiation des données.

Malheureusement, dû à la contrainte de temps qui nous est imposé pour le rendu du projet, il n'y a actuellement presque aucune sécurité de mise en place. Nous allons néanmoins expliciter les différentes sécurités qui seront mises en place à terme.

Premièrement, nous devons intégrer le chiffrement de bout en bout, autrement dit que les données qui transite entre le client & le serveur soient toutes chiffrées, pour certaines données comme l'audio, l'idéal serait même d'assurer un chiffrement de bout en bout non pas entre le client et le serveur, mais entre un client et un autre client.

Pour intégrer le chiffrement de bout en bout nous comptons utiliser des sockets SSL²⁶. Ce protocole assure le chiffrement des données qui passe par ledit socket. Il se base sur un système de certificats et utilise des paires de clés privées et publiques afin de chiffrer la donnée. Il ne faudra donc pas oublier d'intégrer un système de gestion des certificats côté clients et serveur.

Nous souhaiterions aussi éviter qu'un attaquant puisse avoir accès à des informations de la base de données ou encore du serveur en utilisant les points d'entrées¹ de l'application. Un attaquant utilisera ces points d'entrées pour faire des injections SQL²⁷, etc... Pour éviter cela, il faut protéger les différents points d'entrées.

Il faudrait donc utiliser des Regex (Expression régulière) afin de vérifier le format des données rentrées, et s'assurer que l'utilisateur n'essaye pas d'injecter du code du côté serveur.

Les Regex doivent être implémentés du côté serveur, mais l'on pourra aussi les intégrer côté client afin de complexifier la tâche d'un utilisateur malveillant.

¹ Points d'entrées : Ici, les points d'entrées correspondent aux endroits où l'utilisateur peut venir saisir du texte qui sera envoyé puis traité par le serveur (par exemple les champs courriel et mots de passe de l'authentification).

Problèmes rencontrés

Audio

La partie audio de ce projet a été particulièrement problématique. Tout d'abord nous avons essayé de la développer nous-mêmes depuis la base. Après recherches nous avons réalisé que c'était trop ambitieux au vu du temps restant (Fin décembre). Nous avons alors commencé à rechercher des bibliothèques open source et libres de droits (afin de respecter la problématique du projet). Malheureusement nous n'en avons trouvé qu'un répondant à nos besoins. À partir de là nous avons dû apprendre à utiliser CMake²⁸ et les fonctions d'exportations de bibliothèques de Visual Studio²⁹ afin de pouvoir intégrer la bibliothèque à notre projet.

Cette bibliothèque étant extrêmement complète nous avons passé un certain temps avant d'en extraire les classes pertinentes ainsi qu'organiser le code de manière fonctionnelle.

Message

Dans un premier temps, nous avons essayé d'implémenter les messages avec la bibliothèque QXmpp qui propose une intégration de messagerie baser sur le standard XMPP³⁰.

Problèmes avec QXmpp : bibliothèque développée par une seule personne qui manque de documentation. Open source, mais très conséquente. Difficulté d'appropriation et de connexion convenable à la base de données. Il s'agissait de la seule bibliothèque facile d'accès et open source convenant à notre projet puis qu'elle est la seule permettant d'intégrer et le côté client et côté serveur en C++. Finalement, les difficultés d'intégration à notre projet nous ont poussés à abandonner la bibliothèque et à développer nous-mêmes tout le protocole de messagerie.

Découpage de paquets

Comme nous avons décidé de réaliser notre propre protocole applicatif, nous avons rencontré quelques problèmes dans la gestion des paquets surtout lorsque nous avons été confrontés aux images. En effet, la réception des paquets une lecture complète de toutes les requêtes reçues sur le socket TCP³¹. Il s'avère que nous avons été confrontés à deux états particuliers de celui-ci. Premièrement, lors de la lecture on reçoit plusieurs requêtes en une ce qui peut se produire lorsque le client est occupé par une fonction et que plusieurs requêtes arrivent avant que le client soit libre pour les lire. Deuxièmement, les images peuvent être trop grandes pour un seul paquet et arrive sur client découper. Il a donc fallu mettre en œuvre un analyseur de requêtes permettant de gérer tous ces cas spéciaux. (Découper les paquets contenant de multiples requêtes et mettre en attente dans un tampon les requêtes incomplètes)

Interface graphique

Il y'a eu quelques problèmes qui ont ralenti la création de l'interface utilisateur.

Avant d'utiliser Qt Design Studio nous avons commencé par développer l'interface de LockVox sur Qt Designer qui a quelque similarité avec Qt Design Studio en offrant un environnement graphique de conception d'interface. Dans un premier temps, ne connaissant pas l'outil, nous avons perdu du temps à chercher de la documentation ou des explications pour comprendre son fonctionnement.

Après avoir commencé à créer l'interface, nous n'avons pas réussi à avoir des résultats satisfaisants sur l'affichage des salons, des clients et des messages dans les listes. Aussi, la fonctionnalité permettant de redimensionner la fenêtre, ne fonctionnait pas correctement.

Il était aussi plus compliqué d'utiliser des widgets personnalisés et de les importer directement dans une fenêtre. En découvrant les possibilités qu'offrait design studio et même si nous avons perdu de nouveau du temps à redécouvrir un nouvel outil, il s'agissait du meilleur choix à faire pour la réalisation de notre projet.

Axes d'améliorations

Ce projet étant très conséquent, les axes d'améliorations pour LockVox sont très nombreux, voici une liste exhaustive des différents axes que nous pensions essentiels afin d'avoir une application complète :

- Implémentation de l'audio, c'est la partie la plus essentielle qu'il nous reste à développer.
- Implémentation des messages privés.
- Création d'un deuxième serveur qui servirait de serveur DNS³² / Annuaire afin de pouvoir rechercher des serveurs, ajouter des amis qui se trouvent sur des serveurs différents ...)
- Intégration continue de la sécurité de l'application.
- Intégration continue de l'Interface, celle-ci étant prépondérante à l'application, nous devons toujours essayer de l'améliorer, de l'optimiser, pour la rendre plus simple et agréable d'utilisation.
- Implémentation du partage d'écran et de fichier.
- La gestion multiserveurs côté client (pour le moment, l'application Client n'est capable que de se connecter à un seul et unique serveur, ce qui n'est pas vraiment pratique).

Comme dit précédemment, il reste néanmoins énormément de possibilités pour une application de ce genre, et nous pensons que certaines d'entre elles pourraient apporter une vraie valeur ajoutée à l'application :

- Créer une interface utilisateur modulable (dont les différents composants de la fenêtre puissent être bougés, redimensionner, dont les thèmes de couleurs puissent être modifiés et personnaliser ...)
- Intégration de TeamViewer
- Intégration de Spotify / Deezer
- Intégration YouTube / Twitch
- Développement d'une API³³ LockVox

Contributions & Avis des membres du projet

BOURGEAIS Louis

Au cours de ce projet, j'ai participé à la création et la conception de l'interface utilisateur et l'intégration d'élément fonctionnel à ce dernier en QML et C++ en utilisant Qt Designer puis Qt DesignStudio, principalement la gestion des entrées texte et de bouton dans les menus de connexion et d'inscription. Lors de la première partie du projet, j'ai participé à la création de la base de données avec un MLD³⁴ exporté par la suite sur la base de données MariaDB³⁵.

Ce projet m'a permis de découvrir la création d'un logiciel à partir de zéro, et de comprendre les bonnes actions à faire dès le début pour bien prévoir la suite de ce dernier.

J'ai pu participer à la gestion du projet par rapport à la gestion des tâches, des dates limites et la communication au sein de l'équipe et extérieur. J'ai apprécié ce côté qui m'a permis de me rendre compte qu'il faut une organisation autour du développement.

Malgré quelques erreurs et manques d'implication à certains moments, j'ai apprécié travail sur ce projet, je pense que j'ai appris énormément d'éléments que je pourrai réutiliser dans mes projets futurs.

LAVEAU Hugo

Durant ce projet j'ai notamment participé au développement du protocole applicatif et de la partie audio.

J'ai trouvé la partie protocole applicative très intéressante, car cela m'a permis de mieux comprendre les besoins d'une application de type client-serveur. Autant au niveau de la gestion des différentes actions en tant que telle, qu'au niveau des réponses nécessaires à chacun des utilisateurs de l'application afin de garder une application fonctionnelle et cohérente.

J'ai également travaillé sur la partie audio. Bien que très intéressant, cet aspect du projet était conséquent. Effectivement les problèmes liés à la construction, l'exportation et l'intégration de la bibliothèque (et la recherche documentaire sur cette dernière) m'ont fait perdre un temps considérable ce qui nous a malheureusement contraints à abandonner l'audio sur le livrable.

Néanmoins de manière générale ce projet fut très enrichissant, il m'a permis de confirmer une partie de mes compétences acquises durant 4 ans (notamment la programmation en C/C++), mais également d'acquérir de l'expérience de développement, car je n'avais jamais participé à un projet de programmation aussi conséquent. J'ai également appris sur les chaines de traitement du signal et découvert le protocole RTP qui, je pense, me sera utile dans beaucoup d'autres projets.

LELIEVRE François

Durant le projet j'ai pu participer à la conception et à la création de l'interface graphique sur design studio ainsi que certaines connexions entre des éléments de la base de données vers des widgets de l'interface utilisateur. J'ai aussi participé à la conception de la base de données utilisée sur le livrable.

Dans un premier temps au niveau de la base de données, j'ai participé à la création du MLD de la base de données en utilisant le logiciel looping. Ce logiciel a ensuite pu générer un code SQL que nous avons exporté sur MariaDB.

D'un point de vue graphique, je me suis occupé de réaliser les premières esquisses (concept arts) et créer l'identité visuelle ainsi que le logo du projet.

Au niveau de l'interface utilisateur, je me suis aussi occupé du développement de l'interface graphique en QML, notamment la fenêtre principale, la fenêtre de paramètre, différents widgets comme des boutons le widget des messages. J'ai aussi participé à la connexion de certains boutons ainsi que la récupération de valeurs d'entrées de textes et l'envoi des requêtes au serveur.

Ce projet m'a permis de découvrir de nouvelles technologies en développement à travers Qt et ses outils. Il m'a aussi permis de gagner des compétences en conception et développement logicielles et de réaliser un projet complet qui mélange plusieurs domaines (développement, base de données, interface graphique, réseaux).

J'ai vraiment aimé travailler sur ce projet surtout sur la fin de l'année, car mon envie d'avancer sur le projet a augmenté à partir du moment où l'on a commencé à avoir un résultat graphique. Un point à améliorer serait parfois la charge de travail inégalement répartie que notre groupe a fournie durant l'année. Il aurait peut-être fallu se fixer des dates limites plutôt et s'imposer un rythme.

MONNOT Valentin

Durant ce projet, j'ai participé à la réalisation de plusieurs parties du projet.

Dans un premier temps, je me suis occupé de l'intégration des premières bibliothèques qui ont été intégrées au projet. Je me suis concentré sur leurs compréhensions, leurs compilations et leurs intégrations au projet. La prise en main de CMake était une découverte pour moi, ce qui m'a pris quelque temps d'adaptation et de compréhension. J'ai finalement trouvé mon compte dans l'expérience qui m'en a été apporté.

Dans un second temps, je me suis concentré sur le protocole de messagerie. Comme évoqué précédemment, nous nous sommes d'abord penchés sur la bibliothèque QXmpp qui nous a fait perdre du temps à cause des différents problèmes. Je suis donc partie de zéro pour donner suite à l'abandon de cette bibliothèque et j'ai par conséquent développé l'entièreté du protocole applicatif de messagerie mis en œuvre. Cela m'a permis d'en apprendre beaucoup sur la programmation en C++ et ses variables, notamment dans leurs applications dans leurs versions retouchées par Qt.

J'ai aussi donné ma touche personnelle au protocole applicatif afin de mettre en œuvre les échanges réseau entre le serveur et les clients.

J'ai appris beaucoup sur les standards XML et JSON, ce qui est toujours un plus en sachant que ceux-ci sont libres et mondialement utilisés à travers tout type d'application.

Je me suis ensuite tourné sur les images, et leurs gestions à travers notamment du standard Base64 ainsi que les classes Image propres à Qt, la manière de les charger et de les convertir afin de les envoyer à travers le réseau.

Enfin, j'ai pu découvrir les joies de la gestion de fichiers grâce à l'enregistrement et la lecture des fichiers images, XML, JSON et textes mis en œuvre dans le cadre de la gestion des données côté serveur.

À l'origine ce projet était un projet personnel que nous voulions développer entre amis. Le mettre en place à travers ce projet scientifique et technique était très agréable et a permis une meilleure structuration du projet ainsi qu'une meilleure gestion du planning. Je suis vraiment heureux de voir ce projet se concrétiser. Je savais d'avance que cela ne serait pas chose facile et que de par la conséquence du projet, il aurait été compliqué de le terminer en l'espace d'une année scolaire.

Je compte bien continuer à développer ce projet sur mon temps personnel, peut être le reprendrais-je en tant que projet de fin d'études, et qui sais, peut être trouvera-t-il son public au sein de la communauté open source ! J'espère sincèrement que nous trouverons des utilisateurs et des développeurs aux quatre coins du monde prêt à nous aider à pérenniser ce beau projet !

MATTEI Hugues

Au cours de ce projet, j'ai pu prendre part à plusieurs axes du développement du projet comme notamment le protocole applicatif, l'audio ainsi que l'Interface graphique.

J'ai premièrement travaillé sur le protocole applicatif et notamment la sérialisation des différents objets comme les utilisateurs et les salons. J'ai par la suite continué sur le protocole applicatif en implémentant un certain nombre de requêtes du protocole.

Concernant l'audio, j'ai commencé par essayer de développer les salons audios sans utiliser aucune bibliothèque, mais je me suis vite retrouvé bloqué face à des problèmes de tailles que nous avons cités précédemment.

J'ai donc par la suite continué sur l'audio notamment la compilation et la compréhension de l'audio avec EMIPLIB.

Je me suis aussi occupé du développement de l'interface graphique et plus particulièrement de l'intégration des données du code C++ dans l'interface graphique en QML, notamment les modèles pour les listes, mais aussi les connexions des boutons, la récupération des valeurs des entrées textes et finalement l'envoi des requêtes au serveur.

Le fait d'avoir participé à la majorité des axes du développement m'a permis d'avoir une vue globale sur l'avancement du projet et donc de bien définir sur quel point nous devons axer le développement. Par exemple si l'on avait du retard sur une partie, je pouvais alors adapter la répartition du travail entre les différents membres du groupe.

Ce projet m'a apporté de nombreuses compétences comme les bases de l'architecture logiciel, en effet c'est la première fois que je prends part au développement d'une application de cette envergure, il était donc très intéressant de penser son architecture même si cela n'était pas une tâche aisée et que nous avons dû l'adapter au fur et à mesure de l'avancement du projet.

J'ai aussi beaucoup appris sur le traitement de l'audio dans une application, même si les cours que nous avons eus nous apportent le bagage théorique, il était très intéressant de voir comment l'on met en pratique une chaîne de traitement de l'audio dans une application. Même si cela n'a malheureusement pas été très convaincant et que nous avons dû nous rapatrier sur l'utilisation d'une bibliothèque.

Ayant été la plupart du temps en charge de répartir les différentes tâches entre les membres du groupe. J'ai aussi pris de l'aisance pour gérer un groupe de travail, car il fallait répartir les tâches en fonction de compétences et des préférences de chacun.

J'ai apprécié travailler sur ce projet plus que tous les autres, car ce projet me tenait à cœur. J'ai pris encore plus de plaisir sur la fin, car nous commençons finalement à voir tout l'avancement du projet grâce à l'interface graphique.

Un point négatif du projet serait l'implication continue du groupe tout au long de l'année sur le projet. En effet, la motivation n'a pas été constante tout au long de l'année et cela nous a causé du tort à la fin, car nous avons manqué de temps pour pouvoir implémenter plus de fonctionnalités et donc rendre le projet plus interactif et intéressant.

Je compte continuer le développement du projet, car je pense que ce projet s'inscrit dans une démarche positive dans le monde des logiciels de télécommunication actuelle. J'aimerais, personnellement, beaucoup pouvoir utiliser un logiciel tel que celui-ci dans un futur proche.

Conclusion

L'application est fonctionnelle, la plupart des objectifs sont atteints même si certaines parties sont manquantes, notamment l'audio et la sécurité de l'application.

Néanmoins nous pensons que le projet a été mené à bien. En effet le projet est très conséquent et le plus long a été fait. La base du logiciel est opérationnelle, il suffit de rajouter les fonctionnalités non implémentées par manque de temps.

Nous sommes satisfaits du projet, avant d'être un Projet scientifique et technique ce projet était un projet personnel. Nous souhaitons continuer à le faire évoluer, car nous pensons que c'est un projet intéressant dans le monde numérique actuel et celui de l'open source.

Annexes

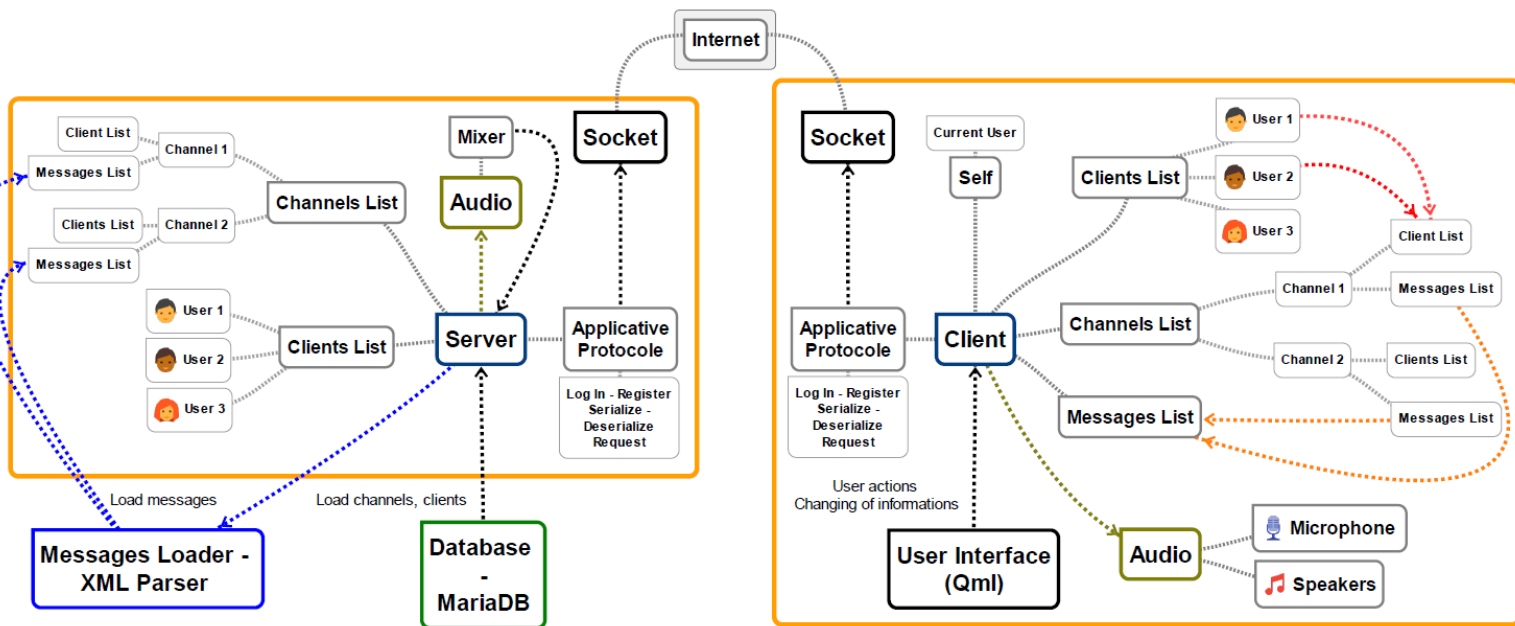


Figure 1 : LockVox Diagramme

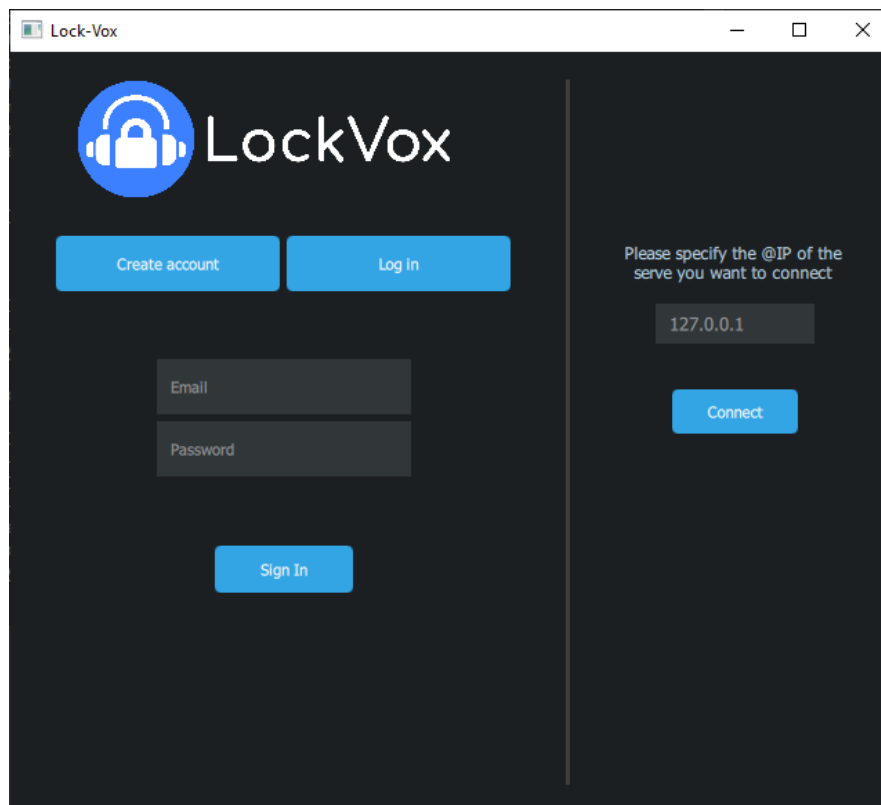


Figure 2 : Fenêtre de connexion

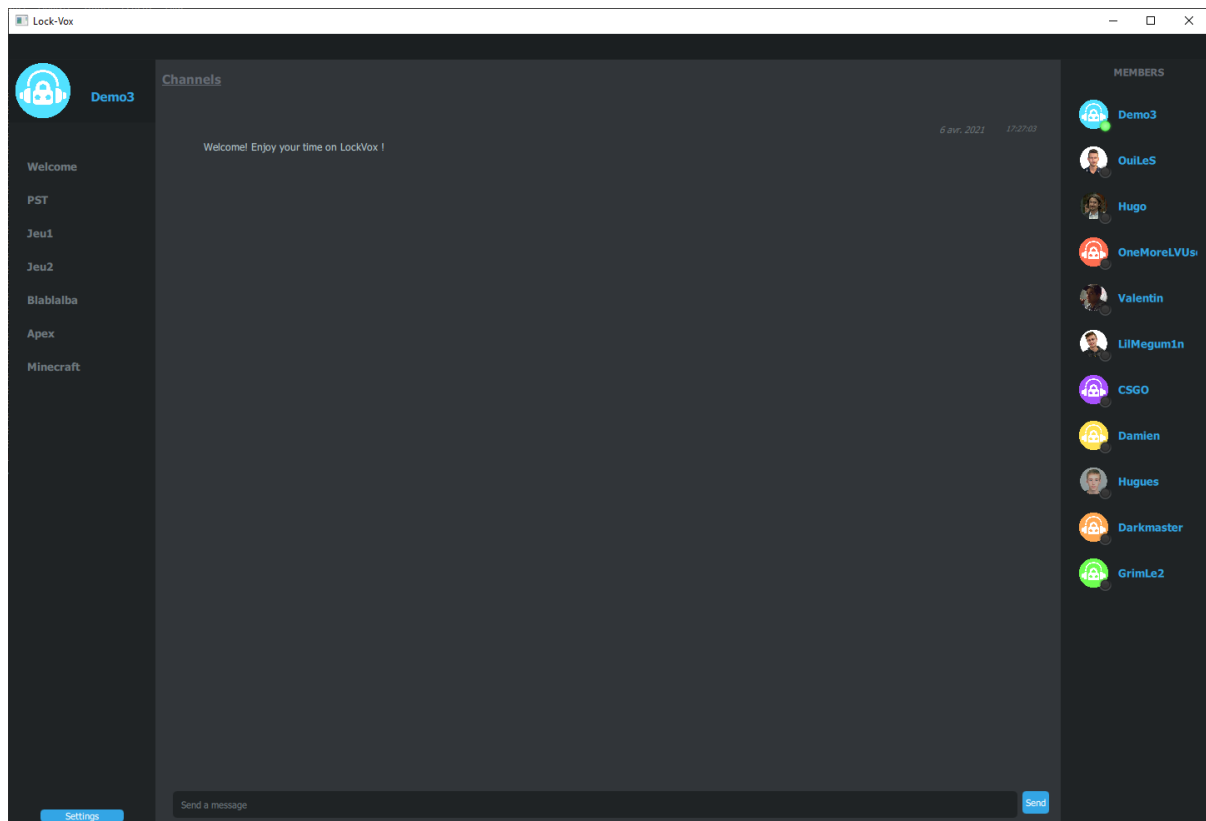


Figure 3 : Fenêtre d'accueil

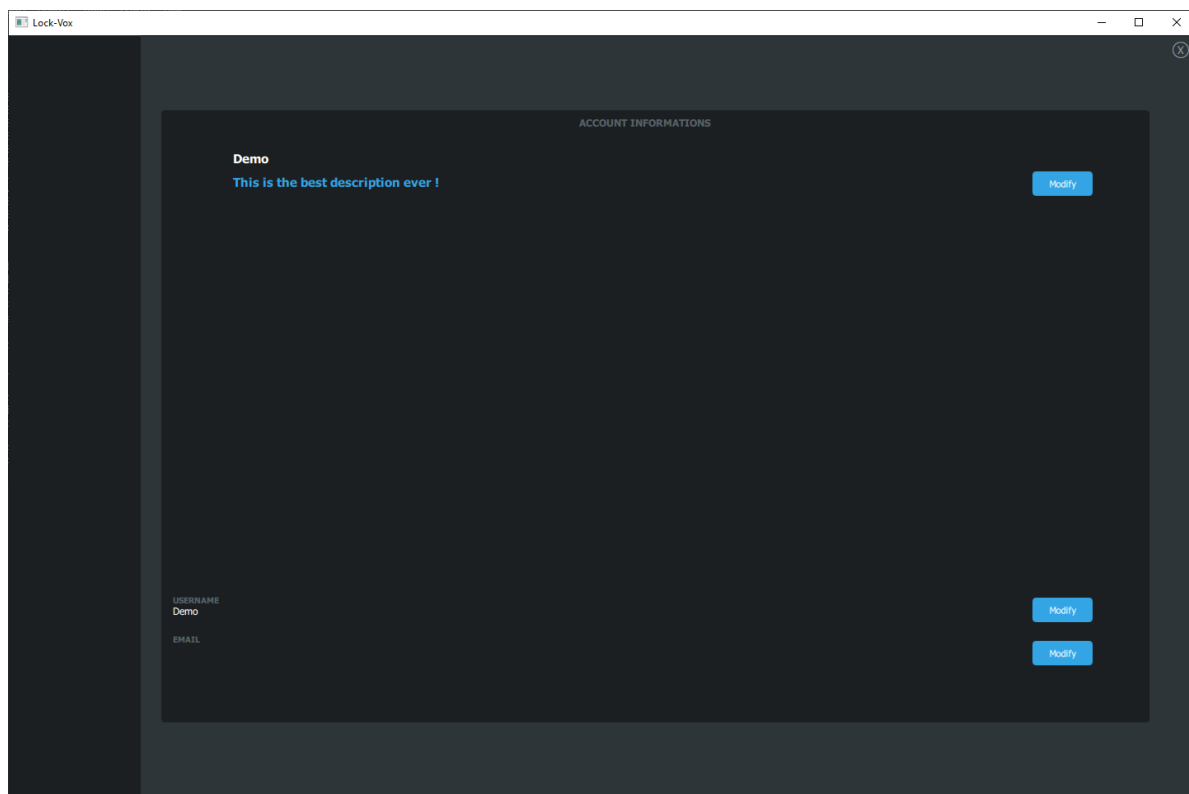


Figure 4 : Fenêtre de paramètres

Côté serveur :

- [Type = 0] Server :
 - [Action = 0] connexion d'un utilisateur
 - [Action = 1] déconnexion d'un utilisateur
 - [Action = 2] mise à jour du pseudo (par un modérateur)
 - [Action = 3] mise à jour de la description
 - [Action = 4] bannissement d'un utilisateur
 - [Action = 5] bannissement d'une adresse IP
 - [Action = 6] retrait du serveur d'un utilisateur (« kick »)
 - [Action = 7] demande d'authentification
 - [Action = 8] demande de création de comptes
- [Type = 1] Salons :
 - [Action = 0] rejoindre un salon
 - [Action = 1] quitter un salon
 - [Action = 2] envoyer un message sur un salon public
 - [Action = 3] Requête de liste de messages d'un salon public
 - [Action = 4] requête de photo de profile
 - [Action = 5] création d'un salon vocal
 - [Action = 6] suppression d'un salon vocal
 - [Action = 7] Renommage d'un salon vocal
 - [Action = 8] Modification du nombre maximum d'utilisateurs d'un salon
 - [Action = 9] Déconnecté un utilisateur d'un salon (« kick »)
 - [Action = 10] couper le micro d'un utilisateur en vocale
 - [Action = 11] création d'un salon textuel
 - [Action = 12] Suppression d'un salon textuel
 - [Action = 13] renommage d'un salon textuel
- [Type = 2] Utilisateurs
 - [Action = 0] couper le son d'un utilisateur (appel privé)
 - [Action = 1] ajouté en ami
 - [Action = 2] supprimer un ami
 - [Action = 3] envoyer un message privé
 - [Action = 4] modifier le pseudo (par l'utilisateur)
 - [Action = 5] changement des rôles (par un modérateur)

Côté Client :

- [Type = 0] Server :
 - [Action = 0] connexion d'un nouvel utilisateur
 - [Action = 1] déconnexion d'un utilisateur
 - [Action = 2] mise à jour d'un pseudo
 - [Action = 3] mise à jour d'une description
 - [Action = 4] bannissement d'un utilisateur
 - [Action = 5] bannissement d'une adresse IP
 - [Action = 6] retrait du serveur d'un utilisateur (« kick »)
 - [Action = 7] demande d'authentification
 - [Action = 8] demande de création de comptes
- [Type = 1] Salons :
 - [Action = 0] un utilisateur a rejoint un salon
 - [Action = 1] un utilisateur a quitté
 - [Action = 2] réception d'une liste de messages sur un salon public
 - [Action = 3] réception d'une erreur d'envoi de message
 - [Action = 4] réception d'un message
 - [Action = 5] création d'un salon vocal
 - [Action = 6] suppression d'un salon vocal
 - [Action = 7] Renommage d'un salon vocal
 - [Action = 8] Modification du nombre maximum d'utilisateurs d'un salon
 - [Action = 9] Déconnecté un utilisateur d'un salon (« kick »)
 - [Action = 10] couper le micro d'un utilisateur en vocale
 - [Action = 11] création d'un salon textuel
 - [Action = 12] Suppression d'un salon textuel
 - [Action = 13] renommage d'un salon textuel
 - [Action = 14] réception d'une photo de profile

- [Type = 2] utilisateurs
 - [Action = 0] couper le son d'un utilisateur (appel privé)
 - [Action = 1] ajouté en ami
 - [Action = 2] supprimer un ami
 - [Action = 3] envoyer un message privé
 - [Action = 4] modifier le pseudo (par l'utilisateur)
 - [Action = 5] changement des rôles (par un modérateur)
 - [Action = 6] réception d'une liste de messages privés

Figure 5 : Liste des actions du protocole applicatif

Cette liste de type et d'action conforme au protocole applicatif contient des actions encore non implémentée, vouée à changer ou inutilisée.

ÉTAT DE L'ART :

LOGICIEL DE TÉLÉCOMMUNICATION

Serveur :

Dans cette partie, nous allons parler des serveurs en général, que sont les différents types d'hébergement, particulier ou professionnel, le référencement, etc. Nous prendrons en exemple Discord et Mumble.

Il existe différentes façons de gérer un serveur. Chez Discord, on retrouve une architecture centralisée. Chaque utilisateur est libre de créer un serveur qui sera hébergé sur les serveurs de Discord en fonction de la région choisie. Cette solution reste la plus simple pour des novices, car il n'y a quasiment aucun paramétrage à faire, simplement un nom a donné et le tour est joué. On retrouve aussi, des systèmes de discussion sans réel serveur, comme Skype par exemple qui ne propose que des discussions de personne à personne ainsi que la création de groupe, c'est un moyen que nous avons mis de côté, car trop limiter en termes de fonctionnalités ainsi que le manque de convivialité que l'on peut avoir sur un serveur de type Discord ou tout le monde peut poster ce qu'il souhaite dans des salons textuels prévus à cet effet, on retrouver un fonctionnement proche d'un réseau social comme Facebook, mais avec une présentation et une approche différente, ainsi qu'un but premier différent. À la différence de Discord, Mumble, ou TeamSpeak qui est basé sur le même fonctionnement, propose aux utilisateurs d'héberger eux-mêmes leurs serveurs. Cela implique un paramétrage un peu plus poussé, de préférence une machine dédiée, et une bonne connexion Internet, ce qui peut être un inconvénient de taille pour certains.

De plus de fait de ne pas avoir d'architecture centralisée peut imposer à l'utilisateur d'avoir un identifiant et un mot de passe par serveur avec un système d'enregistrement propre au serveur. Le fait d'héberger son serveur soit même apporte aussi des bienfaits comme le fait d'avoir le contrôle complet de la gestion, de la mise en place et aussi et surtout, des informations qui transite sur le serveur.

Le problème d'un serveur Discord par exemple, est que l'utilisateur n'a pas tous les accès, ne sait pas comment les données sont stocker, aussi et surtout, toutes les informations, les messages, etc. concernant le serveur sont stockés sur les serveurs de l'entreprise et donc sont potentiellement consultés par l'entreprise a des fins diverses et variées, essentiellement des fins commerciales.

Choix :

Nous aimerions mettre en place un système d'hébergement sur le même principe que celui de Mumble, un serveur centralisé ne colle pas toujours avec l'image de la sécurité. Si l'utilisateur souhaite une discrétion la plus élevée possible, le meilleur moyen reste bien évidemment d'héberger soit même le serveur. Le but étant bien évidemment de pouvoir ouvrir ceci au plus grand public possible, il faut rendre le service le plus simple à mettre en place possible. À l'aide d'une interface simple d'utilisation, de tutoriel officiel, etc. En permettant à l'utilisateur d'avoir plus de contrôle possible sur son serveur, nous espérons pouvoir fournir la meilleure sécurité possible.

Nous pourrions aussi mettre en œuvre une architecture semi-centralisée, permettant avec un seul compte de se connecter à tous les serveurs avec un seul et même compte. Un serveur central fait la liaison avec le serveur en question, lui envoie toutes les informations concernant l'utilisateur puis l'utilisateur se connecte en connexion directe avec le serveur une fois cet échange d'informations effectué, rendant ainsi l'utilisation plus simple tout en ne faisant transiter aucun contenu du serveur sur le serveur central.

Option et personnalisation de compte

Dans cette partie nous aborderons les fonctionnalités et options relatives aux comptes. Nous allons utiliser discord, car ce logiciel contient tous les éléments des autres programmes en plus de nouveau.

Création d'un compte

Lors de la création d'un compte sur le logiciel discord, ce dernier vous demande un nom d'utilisateur, un mot de passe et une adresse mail pour valider votre compte et que celui-ci soit unique. Par la suite vous pouvez le personnaliser ou modifier des options plus techniques.

Personnalisation d'un compte

- Image de profile :

Discord vous permet de modifier votre image de profile à partir d'une image présente sur votre pc. Pour cela une fenêtre de navigation Windows s'ouvre et vous n'avez plus qu'à sélectionner l'image souhaitée. Par la suite vous pouvez la centrer à votre guise, car les photos de profil discord sont contenues dans des cercles.

- Pseudo modifiable :

Chaque compte à un pseudo-unique qui est choisi lors de la création de comptes. Il est possible de modifier son « surnom », les autres utilisateurs voient le pseudo unique si celui-ci n'est pas modifié.

- Définition du statut :

Il est possible d'afficher un statut qui est une petite pastille sur son image de profile, visible dans la liste des utilisateurs du serveur. Ce dernier peut être sélectionné parmi une liste par défaut :

- En ligne, affiche une pastille verte. Indique que vous êtes actuellement en ligne, s'actualise à chaque connexion et déconnexion.
- Inactif, affiche une lune. Indique que vous n'êtes pas actif sur l'application, s'actualise lorsque l'application n'est pas en premier temps pendant un certain temps.
- Invisible, affiche votre image de profil en grisé comme lorsqu'un utilisateur n'est pas connecté pour simuler votre absence. Toutes les fonctionnalités sont néanmoins disponibles.
- Statut personnalisé, vous permet d'afficher un émoji de votre banque d'émoji en plus d'une phrase. Ceux-ci sont affichés au survol de votre vignette pour les autres utilisateurs. Il est possible de choisir le moment précis où ce statut sera supprimé.

- Discord nitro :

Discord comporte un service payant. Celui-ci est un abonnement mensuel qui permet de personnaliser encore plus son compte. Il permet d'avoir des émojis en GIF, d'ajouter plus d'émojis dans votre banque et de pouvoir les utiliser sur n'importe quel serveur. De plus, votre image de profil peut être un GIF et badge nitro est place près de cette image pour « démarquer » ces utilisateurs. Pour finir, cela augmente la taille maximale de téléchargement de fichier de 100Mo et vous permet de diffuser en HD.

- **Connections multicompte :**

Cette option permet à l'utilisateur de lier des comptes d'application comme : Twitch, Reddit, GitHub, Twitter, Steam... Cela rajoute des fonctionnalités au compte (principalement afficher aux autres ses pseudos sur les différentes plateformes) voici quelques exemples :

- Twitch permet de dire si l'utilisateur est en diffusion actuellement, ou quel diffusion est-il en train de regarder, sur Steam permet d'afficher tous les jeux Steam, spot affiche la musique

Éléments techniques

- **Identifiant unique :**

Lors de la création du compte un ID unique de quatre chiffres est généré et placé après le surnom de l'utilisateur. Cela permet de différencier deux comptes avec le même surnom. Lors de l'ajout d'un ami dans sa liste d'amis, l'utilisateur doit impérativement renseigner le « tag » précédé d'un hashtag.

- **Applications autorisées :**


Certaines applications extérieures ou des bots présents sur Discord demandent l'accès aux informations de compte des utilisateurs qui ont accepté et qui l'utilisent. Les permissions sont généralement : sur quel serveur l'utilisateur se situe, qu'elle est le nom et l'avatar de l'utilisateur et son adresse e-mail. Retirer l'autorisation annule tout lien du compte avec le bot ou l'application extérieur, ce qui empêche son fonctionnement.

- **Liste d'amis :**


L'application permet de gérer une liste d'amis privés en plus des amis présents sur les serveurs. On y trouve des options pour limiter les demandes en amis qui sont effectuées. Les options sont : tout le monde peut ajouter l'utilisateur, seulement les amis de ses amis peuvent ajouter l'utilisateur et enfin les membres du serveur peuvent ajouter l'utilisateur en amis.

- Récolte de données :


COMMENT NOUS UTILISONS TES DONNÉES

Utiliser les données pour améliorer Discord 

Ce paramètre nous autorise à utiliser et traiter des informations sur la façon dont tu navigues et utilises Discord à des fins analytiques. Il nous permet, entre autres, de t'inclure dans certaines fonctionnalités expérimentales que nous testons. [En savoir plus.](#)

Utiliser les données pour personnaliser mon expérience Discord 

Ce paramètre nous autorise à utiliser tes informations (comme tes interlocuteurs ou les jeux auxquels tu joues) afin de personnaliser Discord rien que pour toi. [Clique ici pour en savoir plus.](#)

Autoriser Discord à suivre ton utilisation de lecteurs d'écran 

Ce paramètre nous autorise à relever à quels moments tu utilises un lecteur d'écran avec Discord, dans le but d'améliorer l'accessibilité de l'application. [En savoir plus.](#)

Utiliser les données pour que Discord fonctionne

Nous avons besoin de stocker et traiter des données afin de te fournir les fonctionnalités de base de Discord telles que tes messages, les serveurs dans lesquels tu es et tes messages privés. En utilisant Discord, tu nous autorises à fournir ces fonctionnalités de base. Tu peux arrêter tout ça en [désactivant ou supprimant ton compte](#).

Demander toutes mes données

[En savoir plus](#) sur le fonctionnement de l'obtention d'une copie de tes données personnelles.

[Demande de données](#)

- Authentification deux facteurs :

Cette option permet de sécuriser un compte en plus du mot de passe. Il suffit d'ajouter un numéro de téléphone auquel un code d'authentification sera envoyé à chaque demande de connexion. Ce code doit être renseigné en plus du mot de passe. Cela permet aussi d'être prévenu lorsqu'une personne tierce essaye d'accéder à votre compte.

- Filtre des messages :

Discord propose un filtre de contenu explicite (messages et images) pour protéger l'utilisateur. Il peut être appliqué seulement sur les messages privés. On peut le déployer sur tous les messages privés que l'on reçoit, ou seulement les messages ne venant pas d'utilisateur de la liste d'amis, ou encore ne pas analyser du tout.

Choix :

Nous souhaitons utiliser le même système de personnalisation d'image de profil, de statut et de modification du pseudo en plus de rajouter une bannière personnalisée en arrière-plan du profil. Nous comptons mettre en place une gestion de liste d'amis.

Salons de communication

Dans cette partie, nous allons parler des salons textuels. Tout d'abord en définissant ce qu'est un salon textuel, puis en analysant les différentes formes, les différentes fonctionnalités. Afin d'illustrer nos propos, nous utiliserons le logiciel Discord comme référence, car c'est celui qui, à notre goût, est le plus représentatif.

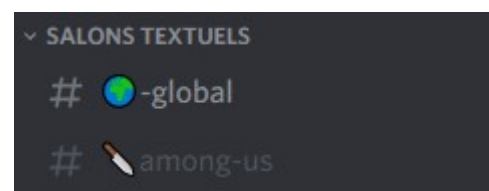
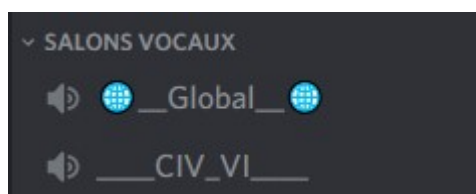
Qu'est-ce qu'un salon ?

Pour bien définir ce qu'est un salon, nous prendrons la définition suivante : « Lieu de réunion, dans une maison où l'on reçoit régulièrement ; la société qui s'y réunit. » Pour nous, on gardera l'idée d'un lieu où les gens peuvent se réunir régulièrement.

Toutes les applications de communication telles que Discord, Teams, possèdent des salons. Ceux-ci permettent aux utilisateurs de se réunir, de manière privée ou publique. On a donc remplacé les salles de réunion par des salons virtuels. Prenons l'exemple de Discord.

Discord :

Sur un serveur Discord, il existe généralement plusieurs salons. On peut y retrouver deux catégories, les salons vocaux ainsi que les salons textuels. Ces salons permettent de délimiter les conversations vocales ainsi que textuelles. Prenons un exemple, si sur un même serveur, plusieurs joueurs jouent à Minecraft. Alors ceux-ci pourraient utiliser un salon dédié aux joueurs Minecraft.



Sur chaque serveur, les salons sont organisés avec des listes. Une liste des salons vocaux et une liste de salons textuels, ils sont regroupés en fonction de leur catégorie.

On remarquera que Discord a fait un choix, les salons ne peuvent pas être textuels et vocaux en même temps. Sur d'autres applications comme Microsoft Teams, les salons sont à la fois vocaux et textuels.

On peut maintenant se demander ce qui définit un salon vocal et un salon textuel. En commençant par les salons textuels puis par les salons vocaux, nous allons essayer de déterminer ce qui fait un bon salon (leurs différentes fonctionnalités).

Salons textuels

Selon Discord, la définition d'un salon textuel est la suivante :

Un **salon textuel** permet de dialoguer à l'écrit avec l'ensemble des membres du serveur.

Le principe de salon est donc respecté, et textuel qui veut dire à l'écrit.

Fonctionnalité :

Nous allons maintenant lister les différentes fonctionnalités d'un salon textuel, à savoir ce qu'il est possible de faire ou non, et nous donnerons notre point de vue à propos de celle-ci.

- Écrire des messages et lire des messages : Fonctionnalités de base disponible sur tous les salons textuels.
- Supprimer les messages : Il est maintenant possible de supprimer des messages sur beaucoup de salons textuels. En règle générale, seul l'utilisateur qui a écrit le message peut le supprimer.
- Modifier les messages : de même, il est intéressant de pouvoir modifier un message même après l'avoir posté.
- Réagir aux messages : permet de donner une « opinion » sur le message.
- Épingler un message : permet de rendre un message plus visible au sein de salon textuel.
- Citation : Permet de donner une forme spéciale à son message (exemple sur Discord : balise pour la mise en forme de code). Nous aimerions intégrer une telle fonctionnalité, néanmoins la contrainte technique est trop grosse pour que cela soit une priorité.
- Dicté le message : permet de dicter son message grâce à la voix. Les contraintes sont encore plus importantes de pour la citation, c'est pourquoi nous avons choisi de ne pas implémenter cette fonctionnalité.

- Notification : Une fonctionnalité importante qui permet d'avertir l'utilisateur lorsqu'un message a été posté. En fonction du temps que nous avons à disposition, nous aimerions pouvoir ajouter des notifications sur les salons textuels.
- Muter le salon : permet de couper les notifications. Cette fonctionnalité vient avec les notifications.
- Partage de fichier : Permet d'envoyer des fichiers sur le salon. Une fonctionnalité très intéressante, mais avec une complexité assez élevée.

Salon vocal

Nous connaissons déjà la définition d'un salon. Et vocal signifie qui est relatif à la voix. Un salon vocal est donc un salon dans lequel il est possible d'interagir grâce à la voix, en d'autres termes, de discuter avec d'autres personnes.

Fonctionnalités

- Rejoindre un salon : chaque utilisateur peut rejoindre un salon vocal.
- Quitter un salon : quitter le salon vocal.
- Se muter : permet à l'utilisateur de couper son micro.
- Couper le son : permet à l'utilisateur de couper le son de salon.
- Nombre de personnes maximum : Il est possible de limiter le nombre de personnes maximum dans un salon. Seuls des droits particuliers permettent de rejoindre le salon quand même.

Nous voudrions implémenter toutes ces fonctionnalités, elles sont présentes sur presque tout le logiciel du même type. On peut donc en déduire que ce sont des fonctionnalités basique et indispensable pour un logiciel agréable à utiliser.

Autres fonctionnalités des salons :

D'autres fonctionnalités telles que la diffusion vidéo sont aussi présentes sur Discord. Comme les autres fonctionnalités, nous aimerions beaucoup que notre logiciel soit capable de diffuser des vidéos, mais encore une fois, cela se révèle assez complexe à mettre en place.

Les salons possèdent aussi des droits qui permettent de filtrer l'accès aux utilisateurs. Cela est géré par l'administrateur du serveur.

Choix

Nous avons décidé de ne pas séparer les salons vocaux et textuels, car nous pensons que cela sera plus agréable à utiliser. Donc à chaque salon sera associés un chat et un salon vocal.

Pour les deux types de salons, nous voudrions implémenter les fonctionnalités de base dans un premier temps.

Après cela, nous essayerons d'implémenter les autres fonctionnalités en fonction de leur importance. Par exemple, le partage de fichier qui est très pratique et très utilisé par beaucoup d'utilisateurs.

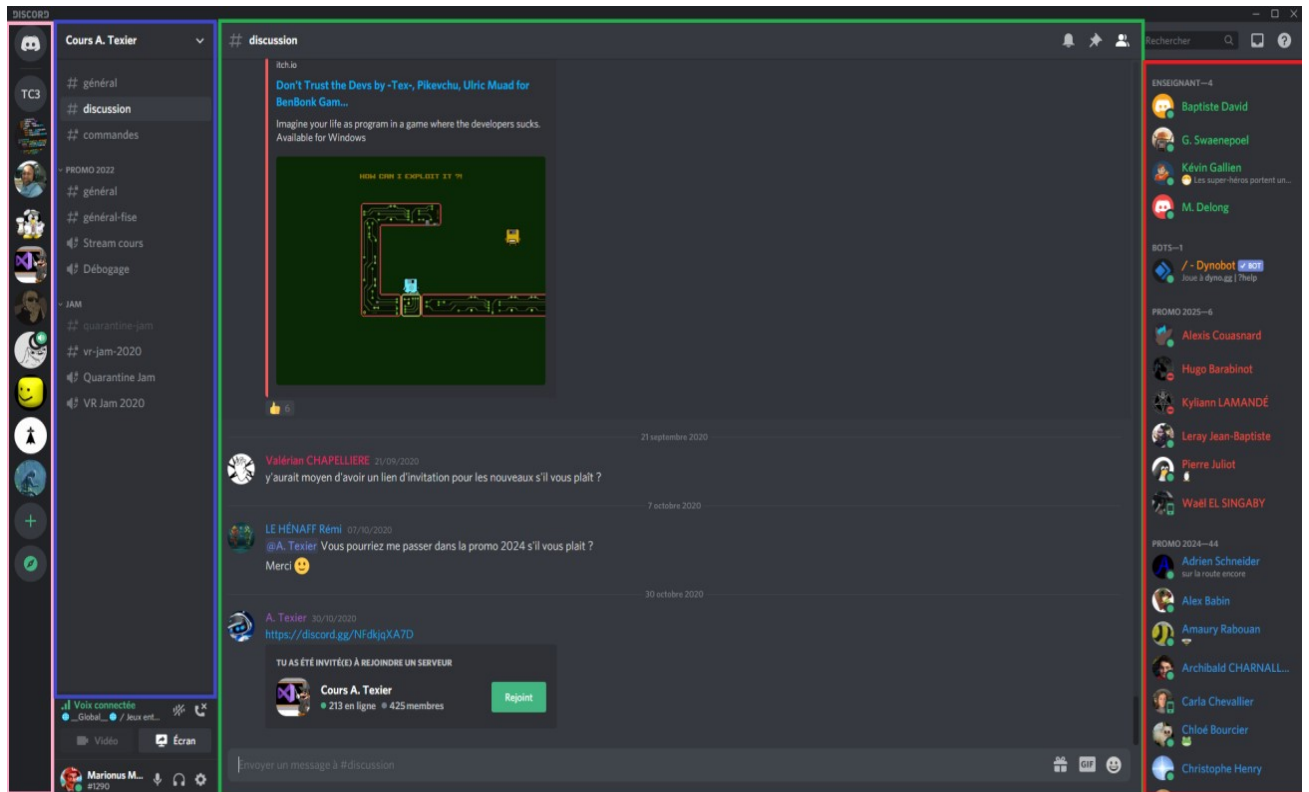
Interface graphique

Analyse

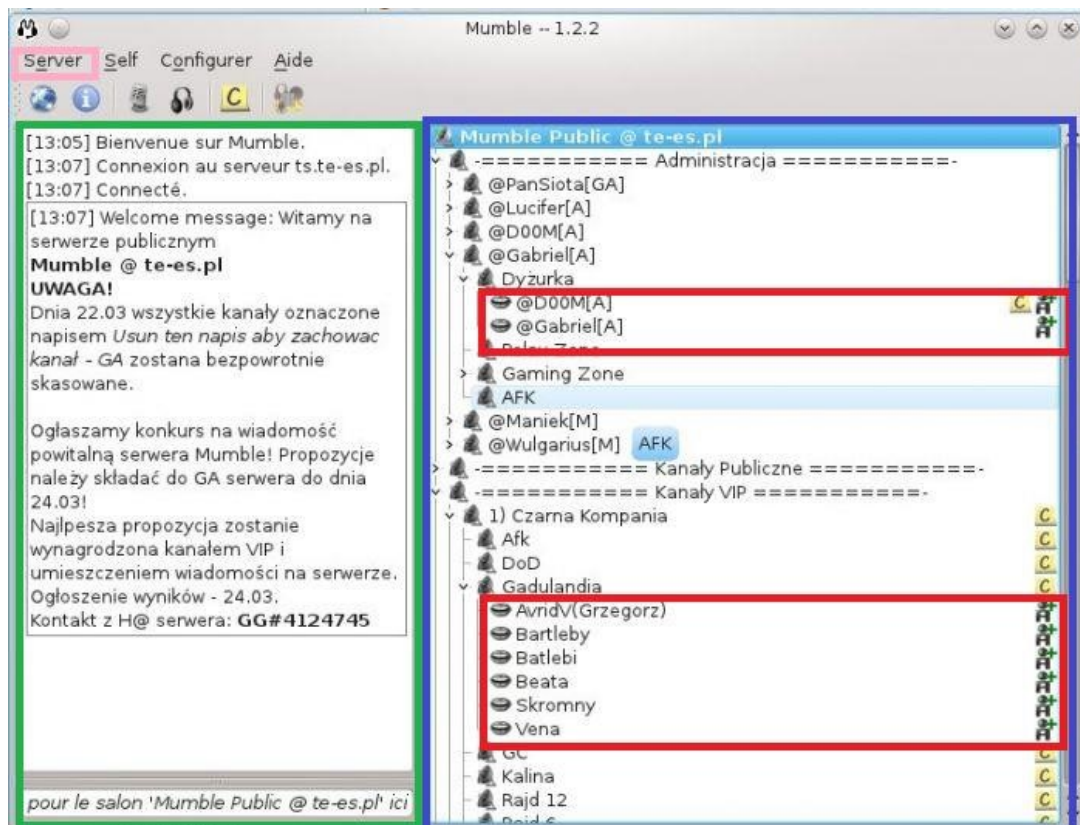
À partir de ces constats, on voit que TeamSpeak et Mumble sont assez proches niveau UI et design, cependant ils manquent de fonctionnalités dans le sens où le format des salons est imposé de manière stricte, là où la séparation du vocal/textuel de Discord permet une gestion plus modulaire des espaces d'échange.

Sur TeamSpeak et Mumble, les utilisateurs sont visibles dans les salons, et les salons sont tous visibles de tout le monde, cela permet d'être sûr de pouvoir connaître l'ensemble des utilisateurs connectés sur le serveur. Discord quant à lui possède la liste complète des utilisateurs triée par rôles et ordre alphabétique. Cependant, il n'est pas possible de savoir si les utilisateurs du serveur sont connectés sur le serveur ou pas, cela est dû au fait que les rôles permettent de rendre des salons visibles seulement pour une partie des membres.

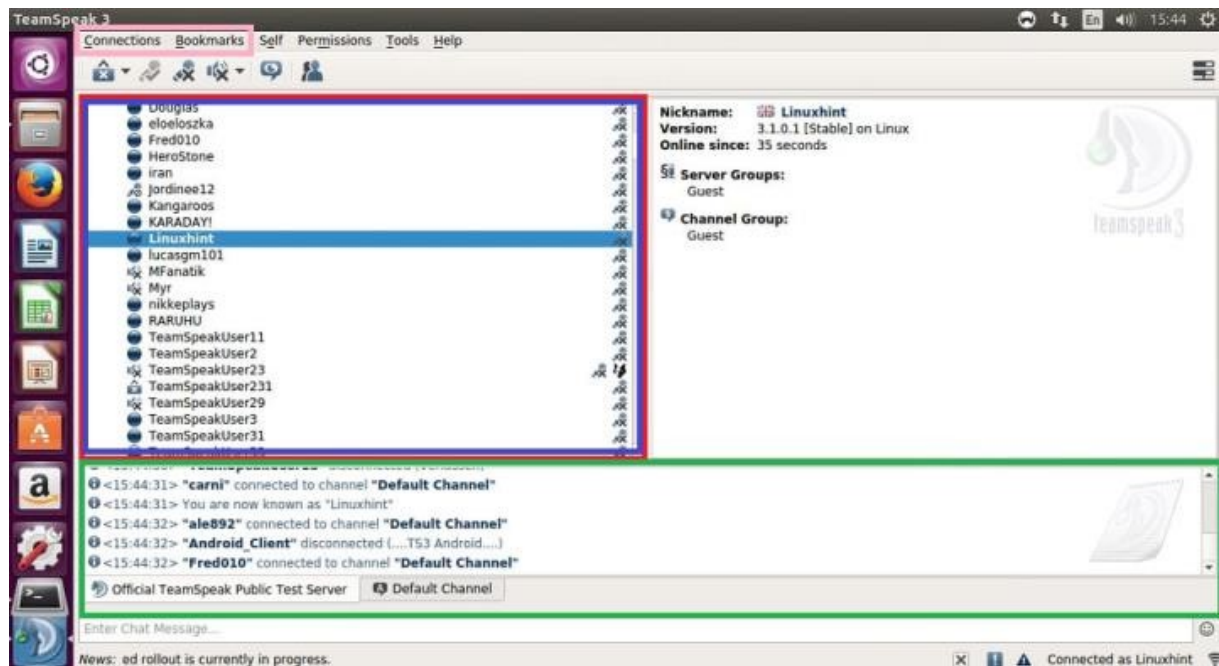
Les serveurs TeamSpeak et Mumble sont des serveurs « privés » dans le sens où la majorité des serveurs sont hébergés par les utilisateurs, ou des services externes à TeamSpeak et Mumble. D'un autre côté, les communications sur Discord transitent toutes par les serveurs de Discord (peut-être même les appels privés).



Interface principale de Discord



Interface principale de Mumble



Interface principale de TeamSpeak3

- Les points communs à « tous » :
 - a. En vert : une zone de chat
 - b. En Rouge : les utilisateurs
 - c. En bleu : Les salons
 - d. En rose : Les serveurs
- Les différences sur ces points :

Le chat :

Discord : Le chat est utilisé dans des salons textuels

TeamSpeak : Un chat par salon vocal

Mumble : Un chat unique, pas de séparation entre les différents salons.

Les utilisateurs :

Discord : Les utilisateurs utilisant le serveur sont tous visibles à droite

TeamSpeak/Mumble : On voit les utilisateurs présents sur le serveur à travers les salons.

Les salons :

Discord : Salon vocal ou textuel

TeamSpeak : Salon vocal et textuel

Mumble : Salon vocal à choisir, textuel « incrusté »

Les serveurs :

Discord : Liste des serveurs auquel on appartient, sur invitation

TeamSpeak/Mumble : Dans un onglet, requiers l'adresse IP + éventuellement un mot de passe

Choix

De manière générale, Discord propose une interface plus ergonomique et une gestion du serveur plus modulaire que ce soit au niveau des moyens de communication ou même des rôles. Le but étant de rendre l'application la plus attractive et complète possible nous orienterons nos choix en nous inspirant majoritairement de Discord.

Néanmoins l'aspect de déploiement de serveurs côté client est très intéressant, dans le sens où cela permet à chacun de posséder son propre serveur, de plus le code étant open source chacun pourra l'adapter à son besoin.

Les fonctionnalités liées à la gestion d'amis sont également très intéressantes, car elles permettent les communications privées, néanmoins elles sont secondaires, car l'objectif de base est surtout d'avoir un moyen de communication sécurisée, ce qui ne nécessite pas d'architecture centralisée (contrairement aux fonctionnalités liées à la gestion d'amis). Le développement de ce point dépendra de notre avancée dans le projet.

Glossaire

1 - VoIP :

Voice over IP : une technologie informatique qui permet de transmettre la voix sur des réseaux compatibles IP, via Internet ou des réseaux privés (intranets) ou publics, qu'ils soient filaires (câble/ADSL/fibre optique) ou non (satellite, WiFi et réseaux mobiles).

2 - Open source :

La désignation open source, ou code source ouvert, s'applique aux logiciels (et s'étend maintenant aux œuvres de l'esprit) dont la licence respecte des critères précisément établis par l'Open Source Initiative, c'est-à-dire les possibilités de libre redistribution, d'accès au code source et de création de travaux dérivés. Mis à la disposition du grand public, ce code source est généralement le résultat d'une collaboration entre programmeurs.

3 - UUID :

Universally Unique Identifier : Une suite de 128 caractères utilisée pour identifier les informations ou les utilisateurs dans les systèmes informatiques. Dans notre cas, nous utilisons la version 4 de l'UUID qui est généré aléatoirement à la création d'un compte. On le retrouve sous la forme suivante : xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx ou x correspond à n'importe quelles lettres de l'alphabet ou chiffre du système décimal.

4 - JSON :

JavaScript Object Notation : un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de représenter de l'information structurée comme le permet XML par exemple.

5 - XML :

Extensible Markup Language : Un langage de balisage extensible, c'est un métalangage informatique de balisage générique qui est un sous-ensemble du Standard Generalized Markup Language (SGML). Sa syntaxe est dite « extensible », car elle permet de définir différents langages avec pour chacun son vocabulaire et sa grammaire.

6 - Repository et GitHub :

GitHub est une plateforme en ligne basée sur le logiciel Git. Git est un logiciel de gestion de versions décentralisé. Il s'agit d'un logiciel libre permettant d'enregistrer les différentes versions du code source d'une application. GitHub est donc une plateforme de stockage basé sur ce logiciel. Un repository est tout simplement une page du site GitHub réserver pour un programme.

7 - Fork :

Dans le domaine de l'ingénierie logicielle, un Fork est une bifurcation de projet. Elle se produit lorsque les développeurs prennent une copie du code source d'un logiciel et commencent à le développer de manière indépendante, créant ainsi un logiciel distinct et séparé. Le terme implique souvent non seulement une branche de développement, mais aussi une scission dans la communauté des développeurs.

8 - Framework :

En programmation informatique, un Framework (appelé aussi infrastructure logicielle) désigne un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel (architecture).

9 - Socket :

Socket (mot anglais qui signifie prise) est un terme informatique qui peut avoir plusieurs significations suivant qu'il est utilisé dans le cadre logiciel ou matériel. Dans le contexte des logiciels, on peut le traduire par « connecteur réseau » ou « interface de connexion ».

10 - Thread :

Un thread ou fil (d'exécution) ou tâche est similaire à un processus, car tous deux représentent l'exécution d'un ensemble d'instructions du langage machine d'un processeur. Du point de vue de l'utilisateur, ces exécutions semblent se dérouler en parallèle.

12 - Salage (de mot de passe) :

Le salage est une méthode permettant de renforcer la sécurité des informations qui sont destinées à être hachées (par exemple des mots de passe) en y ajoutant une donnée supplémentaire afin d'empêcher que deux informations identiques conduisent à la même empreinte (la résultante d'une fonction de hachage).

13 - Widget :

Un widget graphique (également appelé élément de contrôle graphique) dans une interface utilisateur est un élément d'interaction, tel qu'un bouton ou une barre de défilement. Les contrôles sont des composants logiciels avec lesquels un utilisateur peut interagir par une manipulation directe pour lire ou modifier des informations sur une application.

14 - QML :

QML (Qt Modeling Language) est un langage de balisage d'interface utilisateur. Il s'agit d'un langage déclaratif (similaire au CSS et JSON) pour la conception d'applications centrées sur l'interface utilisateur. Il est associé à Qt Quick, le kit de création d'interfaces utilisateur développé à l'origine par Nokia dans le cadre du Framework graphique Qt.

15 - Implémentation :

Mise en place sur un ordinateur d'un système d'exploitation ou d'un logiciel adapté aux besoins et à la configuration informatique de l'utilisateur.

16 - Slots & Signaux :

Les signaux et les slots sont une construction de langage introduite également dans Qt pour la communication entre les objets qui facilite la mise en œuvre du modèle d'observateur tout en évitant le code passepartout. Le concept est que les widgets de l'interface graphique peuvent envoyer des signaux contenant des informations sur les événements qui peuvent être reçus par d'autres widgets en utilisant des fonctions spéciales connues sous le nom de slots.

17 - Sérialisation :

En informatique, la sérialisation (de l'anglais américain serialization) est le codage d'une information sous la forme d'une suite d'informations plus petites pour, par exemple, sa sauvegarde (persistance) ou son transport sur le réseau. La désérialisation représente l'action inverse permettant de rendre l'information utilisable et compréhensible par le programme.

18 - NIY :

Not Implemented Yet : en français « Pas encore mis en œuvre ». Désigne des fonctionnalités du programme qui ont été pensés, voir qui ont déjà commencé à être développés, mais qui ne sont pas encore accessibles par l'utilisateur.

19 - Bibliothèque :

En informatique, une bibliothèque logicielle est une collection de routines, qui peuvent être déjà compilées et prêtes à être utilisées par des programmes. Les bibliothèques sont enregistrées dans des fichiers semblables, voire identiques aux fichiers de programmes, sous la forme d'une collection de fichiers de code objet rassemblés accompagnée d'un index permettant de retrouver facilement chaque routine.

20 - RTP :

Real-Time Transport Protocol : un protocole de communication informatique permettant le transport de données soumises à des contraintes de temps réel, tel que des flux média audio ou vidéo.

21 - Float :

Le float ou « virgule flottante » est une méthode d'écriture de nombres réels fréquemment utilisée dans les ordinateurs.

22 - Effet Larsen :

L'effet Larsen est un phénomène physique de rétroaction acoustique involontaire observé dès les débuts de la téléphonie et décrit par le physicien danois Søren Larsen. Il se désigne en anglais par feedback signifiant plus généralement « rétroaction, réaction, action en retour, bouclage ».

23 - μ -Law :

L'algorithme Loi μ (ou Loi Mu, en anglais μ -law ou mu-law) est un système de quantification logarithmique d'un signal audio. Il est principalement utilisé pour traiter la voix humaine dont il exploite les caractéristiques. Il est principalement utilisé pour les communications téléphoniques. Ce système de codage est utilisé aux États-Unis et au Japon. En Europe, le système équivalent est nommé loi A.

24 - Base64 :

En informatique, Base64 est un codage de l'information utilisant 64 caractères, choisis pour être disponibles sur la majorité des systèmes. Défini en tant qu'encodage MIME, il est principalement utilisé pour la transmission de messages (courrier électronique et forums Usenet) sur Internet.

MIME :

Multipurpose Internet Mail Extensions ou Extensions multifonctions du courrier Internet est un standard Internet qui étend le format de données des courriels pour supporter des textes en différents codages des caractères autres que l'ASCII, des contenus non textuels, des contenus multiples, et des informations d'entête en d'autres codages que l'ASCII.

ASCII :

L'American Standard Code for Information Interchange (Code américain normalisé pour l'échange d'information), plus connue sous l'acronyme ASCII ([aski:]), est une norme informatique de codage de caractères apparus dans les années 1960. C'est la norme de codage de caractères la plus influente à ce jour.

25 - Hachage :

On nomme fonction de hachage, de l'anglais hash function (hash : pagaille, désordre, recouper et mélanger) par analogie avec la cuisine, une fonction particulière qui, à partir d'une donnée fournie en entrée, calcule une empreinte numérique servant à identifier rapidement la donnée initiale, au même titre qu'une signature pour identifier une personne. Les fonctions de hachage sont utilisées en informatique et en cryptographie notamment pour reconnaître rapidement des fichiers ou des mots de passe.

26 - SSL :

Le Transport Layer Security (TLS) ou « Sécurité de la couche de transport », et son prédécesseur le Secure Sockets Layer (SSL) ou « Couche de sockets sécurisée », sont des protocoles de sécurisation des échanges par réseau informatique, notamment par Internet.

27 - SQL :

SQL (Structured Query Language, en français langage de requête structurée) est un langage informatique normalisé servant à exploiter des bases de données relationnelles. La partie langage de manipulation des données de SQL permet de rechercher, d'ajouter, de modifier ou de supprimer des données dans les bases de données relationnelles.

Base de données relationnelles :

En informatique, une base de données relationnelle est une base de données où l'information est organisée dans des tableaux à deux dimensions appelées des relations ou tables.

28 - CMake :

CMake est un système de construction logicielle multiplateforme. Il permet de vérifier les prérequis nécessaires à la construction, de déterminer les dépendances entre les différents composants d'un projet, afin de planifier une construction ordonnée et adaptée à la plateforme.

29 - Visual Studio :

IDE propriétaire appartenant à Microsoft

IDE :

Un environnement de développement intégré (IDE) est une application logicielle qui fournit aux programmeurs informatiques des outils complets pour le développement de logiciels. Un IDE comprend normalement au moins un éditeur de code source, des outils d'automatisation de construction et un débogueur.

30 - XMPP :

Extensible Messaging and Presence Protocol (qu'on peut traduire par « protocole extensible de présence et de messagerie »), souvent abrégé en XMPP, est un ensemble de protocoles standards ouverts de l'Internet Engineering Task Force (IETF) pour la messagerie instantanée, et plus généralement une architecture décentralisée d'échange de données.

31 - TCP :

Transmission Control Protocol (littéralement, « protocole de contrôle de transmissions »), abrégé TCP, est un protocole de transport fiable, en mode connecté.

32 - DNS :

Le Domain Name System, généralement abrégé DNS, qu'on peut traduire en « système de noms de domaine », est le service informatique distribué utilisé pour traduire les noms de domaine Internet en adresse IP ou autres enregistrements. En fournissant dès les premières années d'Internet, autour de 1985, un service distribué de résolution de noms, le DNS a été un composant essentiel du développement du réseau.

33 - API :

En informatique, une interface de programmation d'application ou interface de programmation applicative (souvent désignée par le terme API pour Application Programming Interface) est un ensemble normalisé de classes, de méthodes, de fonctions et de constantes qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels.

34 - MLD :

Modèle logique des données, également appelée dérivation du MCD est un formalisme adapté à une implémentation ultérieure, au niveau physique, sous forme de base de données relationnelle ou réseau, ou autres (ex : simples fichiers). Il s'agit d'un modèle relationnel permettant de prévisualiser l'implémentation d'une base de données relationnel. (Exemple page 5)

MCD :

Le modèle conceptuel des données ou MCD repose sur les notions d'entité et d'association et sur les notions de relations. Le modèle conceptuel des données s'intéresse à décrire la sémantique du domaine (entity/relationship en anglais)

35 - MariaDB :

MariaDB est un système de gestion de base de données éditée. Il s'agit d'un fork communautaire de MySQL : la gouvernance du projet est assurée par la fondation MariaDB, et sa maintenance par la société Monty Program AB, créateur du projet. Cette gouvernance confère au logiciel l'assurance de rester libre.

Source des définitions :

Wikipédia : <https://fr.wikipedia.org>

Larousse : <https://www.larousse.fr>

Définitions réalisées par nos soins