Projet ORA

Crab Wave

EPITA

leo.benito • raffael.moraisin • adam.thibert • pierre-corentin.auger

Réalisé le 26 mai 2020

Table des matières

1 Introduction 4

2 Présentation du groupe 6
- 2.1 Léo Benito . . . 6
- 2.2 Raffaël Moraisin . . . 6
- 2.3 Adam Thibert . . . 6
- 2.4 Pierre-Corentin Auger . . . 6

3 Présentation du projet 7
- 3.1 Inspirations . . . 7
- 3.2 Explications . . . 7
- 3.3 Répartition des tâches . . . 10
- 3.4 Planning . . . 10

4 Division du projet 12
- 4.1 Introduction . . . 12
- 4.2 Tracker . . . 12
- 4.3 Daemon . . . 15
- 4.4 Application Programming Interface (API) . . . 15
- 4.5 Core . . . 16
  - 4.5.1 Protocole . . . 16
  - 4.5.2 Compression . . . 17
  - 4.5.3 Chiffrement . . . 18
- 4.6 Command Line Interface (CLI) . . . 19
- 4.7 Graphical User Interface (GUI) . . . 20
- 4.8 Site Web . . . 24
  - 4.8.1 Page principale . . . 24
  - 4.8.2 Documentation . . . 25

5 Outils et Méthodes utilisées 28
- 5.1 Outils utilisés . . . 30

6 Déroulement du projet 31
- 6.1 L'avant cahier des charges . . . 31
- 6.2 L'après cahier des charges . . . 31
- 6.3 L'après première soutenance . . . 32
- 6.4 L'après seconde soutenance . . . 32

7 Expériences personnelles 33
7.1 Léo Benito 33
7.2 Raffaël Moraisin 33
7.3 Adam Thibert 34
7.4 Pierre-Corentin Auger 35

8 Défauts 37

9 Conclusion 38

10 Annexes 39
10.1 Vocabulaire 39

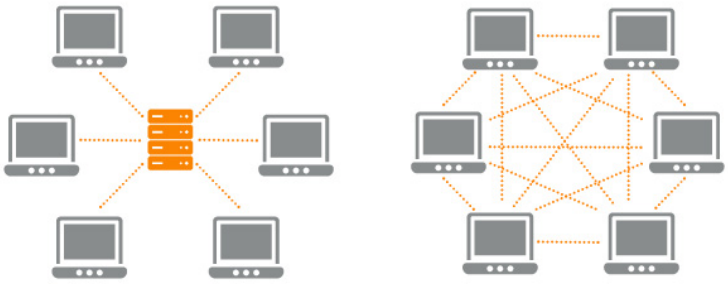
1 Introduction

Le *Peer-to-Peer* (P2P) est un modèle de réseau informatique où contrairement au modèle client-serveur qui comporte un seul serveur, chaque client est aussi un serveur. Le *Peer-to-Peer* est assez récent, il vit le jour en 1969 date du lancement du projet *ARPAnet* qui était un précurseur de l'Internet utilisant le P2P. Ce réseau pose les bases d'un réseau d'interconnexions non hiérarchique et surtout décentralisé. Il est notamment le premier réseau à transfert de paquets, qui deviendra la base du transfert de données sur Internet grâce au concept de commutation de paquets. Mais, c'est surtout les systèmes de partage de fichiers qui ont permis de populariser le *Peer-to-Peer* comme *Napster* en 1999.

En avril 2001, le protocole *BitTorrent* est créé, c'est un protocole de partage de fichier en P2P. Son principe est simple : il trouve les utilisateurs qui possèdent les fichiers que l'utilisateur recherche, puis il télécharge simultanément auprès des différents utilisateurs qui possèdent le fichier. De ce fait, les taux de transmission sont plus rapides qu'avec *HTTP* et *FTP* (protocoles de transfert de données plus classiques), qui téléchargent les fichiers de façon séquentielle à partir d'une seule source. Par rapport aux autres systèmes P2P, *BitTorrent* a l'avantage de créer une sorte de cercle vertueux lors du partage de fichiers. En effet, celui qui donne plus peut recevoir plus alors qu'à l'inverse celui qui donne peu ou pas recevra moins d'autrui.

Enfin, eMule est un logiciel *open-source* de partage de fichiers en P2P. De ce fait, des programmeurs distribuent des versions améliorées ou corrigées d'eMule en partant de son code source, ce qui a popularisé encore plus le logiciel. De plus, ce dernier possède un système de cercle vertueux d'accès au fichier désiré comme *BitTorrent*, plus compréhensible car il est basé sur un multiplicateur dépendant de 2 ratios qu'on peut facilement calculer. Ainsi plus ce multiplicateur est grand, moins le client a de temps à perdre dans les files d'attentes pour accéder au fichier qu'il souhaite. Enfin, eMule possède un client *HighID* permettant aux *LowID*, ordinateurs derrière un pare-feu ou avec une adresse IP se terminant par 0, de télécharger ou d'envoyer des données. De ce fait, eMule fonctionne dans tous les pays contrairement aux autres.

ORA est une application de stockage de fichiers en *Peer-to-Peer*, centrée autour de la notion de partage de fichiers au sein d'un même groupe. Nous n'avons pas utilisé le protocole *BitTorrent*, nous avons défini notre propre protocole. Notre application permet donc de partager des fichiers sans passer par un tiers, ce qui offre aux utilisateurs un meilleur contrôle sur leurs données. Cependant, les fichiers vont être dupliqué ce qui est un problème du *Peer-to-Peer* car les différents pairs vont stocker le fichier afin de bénéficier du fichier mais aussi d'assurer la reliabilité du stockage. En réalité, ce n'est pas un réel problème au niveau du stockage puisque le coût du stockage baisse de plus en plus alors que les capacités de stockage augmentent.



Pour cette application, nous avons établi un vocabulaire spécifique correspondant à différentes abstraction. Nous vous invitons à vous référer à la partie Annexe pour accéder à la définition de ces termes et ainsi pouvoir comprendre la totalité de ce rapport.

2 Présentation du groupe

2.1 Léo Benito

« Bonjour, je m'appelle Léo, je suis passionné par l'informatique, ainsi que les maths depuis de nombreuses années et j'ai commencé à programmer il y a quelques années maintenant. Pour moi ce projet va être de nouveau l'occasion d'améliorer ma capacité à travailler en groupe. Aussi, il va me permettre de découvrir des outils avec lesquels je n'ai jamais travaillé et de pratiquer des méthodes de programmation agiles comme le *TDD* ou l'intégration continue. »

2.2 Raffaël Moraisin

« Je m'appelle Raffaël Moraisin et cela fait maintenant 2 ans que j'étudie l'informatique. Je l'ai découvert grâce à l'option ISN à mon lycée et depuis je suis toujours intéressé par cette matière qui met en relation apprentissage d'une langue et rigueur d'une logique mathématique. Ce projet est tout d'abord le premier projet informatique auquel je participe. De ce fait, celui-ci va m'apprendre à fournir un code lisible et idéal pour moi et les autres, à utiliser plusieurs outils comme LaTeX ou *Overleaf* mais aussi à décomposer une idée sous forme de fonctions algorithmiques. Enfin, ce projet est destiné au stockage en *peer-to-peer*, technique dont j'ignorais l'existence et que j'apprendrai a perfectionner à travers ce projet. »

2.3 Adam Thibert

« Je m'appelle Adam Thibert et je suis passionné d'informatique depuis l'école primaire, ayant un oncle développeur, j'ai été poussé à commencer la programmation assez tôt, commençant par des cartes programmables pour finir sur du développement software. J'ai tout le temps envie d'apprendre de nouvelles technologies et je suis assez curieux pour aller assez loin avant de me faire un avis. Ainsi, bien qu'ayant déjà quelques années d'expérience en programmation, je n'ai quasiment jamais utilisé la plupart des technologies et méthodologies utilisées dans ce projet, que ce soit le *TDD*, le *peer-to-peer*, LaTeX ou encore tout simplement le C#. En plus d'apprendre à correctement utiliser ces technologies, le projet me permettra de m'améliorer dans beaucoup de points, que ce soit au niveau de la documentation du code, du travail en équipe ou même de l'organisation. »

2.4 Pierre-Corentin Auger

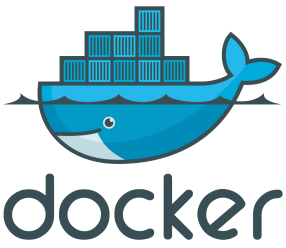
« Bonjour, je m'appelle Pierre-Corentin et j'ai 17 ans (bientôt 18). Bien que m'étant intéressé à l'informatique assez récemment, j'arrive à me débrouiller. Ce projet étant le premier de ce genre auquel je prends part, j'attends à ce qu'il me fournisse l'expérience nécessaire pour pouvoir aborder les projets des années supérieures. A l'issue du projet, j'espère pouvoir mieux comprendre comment marche le partage d'information en *peer-to-peer*, dont j'avais déjà entendu parler mais que je n'ai jamais utilisé . J'attends aussi à ce que ce projet me permette d'apprendre à utiliser LaTeX, à améliorer la lisibilité de mon code et à optimiser au mieux mes fonctions. »

3 Présentation du projet

Dans cette partie, nous vous présenterons le projet plus en détail puis nous rappellerons la répartition des tâches ainsi que le planning des tâches.

3.1 Inspirations

Comme expliqué dans l'introduction de ce document, ce projet est beaucoup inspiré des méthodes de partage de fichiers en *Peer-to-Peer* et notamment des torrents et du protocole *BitTorrent*. Mais nous nous sommes aussi inspirés de programmes créés auparavant pour des projets personnels afin de désigner l'architecture globale du code ou bien encore du design de l'application en *CLI* et en *GUI* avec notamment une forte inspiration du *CLI* de *Docker*.

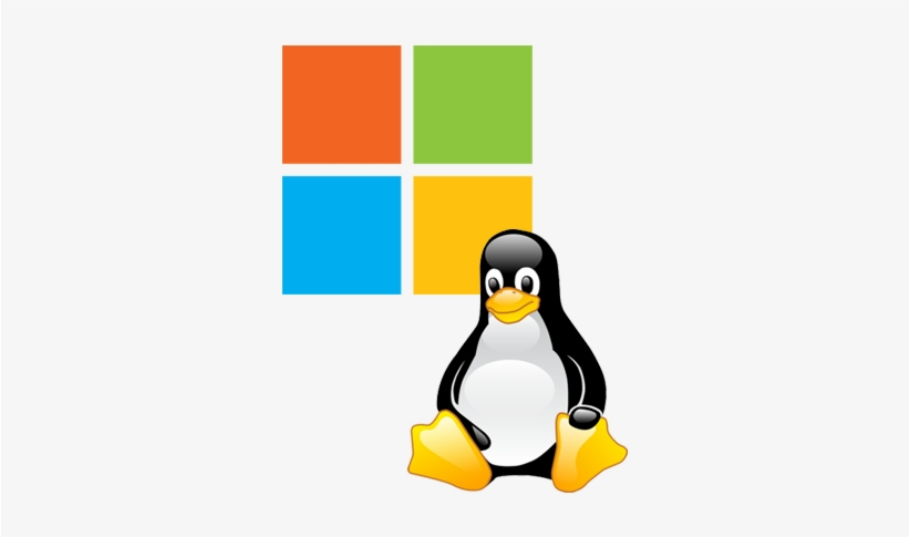


3.2 Explications

Notre système fonctionne avec un programme qui tourne en tâche de fond nommé le *Daemon* qui va exposer toutes les fonctionnalités du *Core* via des tunnels *IPC*. Le protocole *IPC* (Inter-Process Communication) permet à différents processus de communiquer entre eux. Pour fonctionner, le *CLI* et *GUI* vont demander au Daemon d'éxécuter ces tâches spécifiques au lieu d'utiliser les fonctionnalités du *Core* directement. Nous avons créé notre application de cette manière puisqu'il nous faut un programme qui tourne en fond pour pouvoir demander des informations au *Tracker* régulièrement mais aussi de pouvoir distribuer des fichiers si un *Node* le demande sans que l'application soit ouverte.

Dans l'introduction nous avons dit que notre application permettait de partager des fichiers sans passer par un tiers, ceci n'est pas exactement vrai puisqu'il y a un programme nommé le *Tracker* qui contient les informations de tous les *Clusters* et permet aux *Nodes* de se connecter entre eux. Cependant, si vous n'avez pas confiance en celui hébergé par notre groupe, vous avez la possibilité de héberger un *Tracker* vous-même puisqu'il est totalement auto-hébergeable.
Aussi, le *Tracker* ne stocke pas directement les fichiers, il stocke leurs hash, leurs taille ce ne sont pas des informations confidentielles.

Notre application a pour but de fonctionner sous Windows comme spécifié dans le dossier distribué mais aussi d'être compatible avec Linux car ce système d'exploitation nous tient particulièrement à coeur. Nous avons donc mis en place des tests sous Linux dans notre pipeline afin de garantir que notre programme fonctionne sous Linux.



Aussi la thématique de la sécurité est importante pour nous, l'application utilise le chiffrement RSA afin de sécuriser les communications et les données car des fichiers personnels peuvent être partagés.

À terme, nous souhaitons que ORA soit totalement open-source, ce qui permettrait à n'importe qui d'en créer une version modifiée et de la redistribuer. Notre projet contient deux implémentations différentes de l'*API* qui auraient pu être créées par des développeurs utilisant celle de ORA. La première implémentation est le *CLI*, l'application en lignes de commandes dont le développement a débuté afin de fournir un aperçu de l'interaction avec le **Tracker**. La deuxième est le *GUI*, l'application graphique qui est plus *end-user friendly*. Ces deux implémentations nous permettent de montrer que grâce à l'API, on peux gérer les fichiers, *Clusters* et *Nodes* de manière totalement différente.

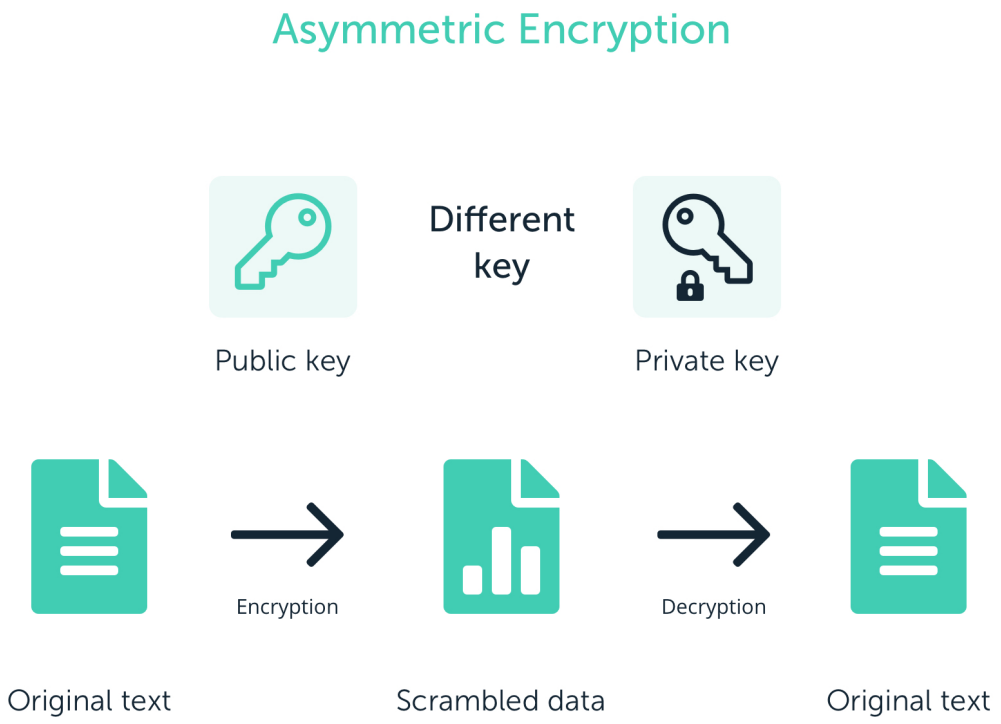
Le protocole désigne une spécification de plusieurs règles afin d'établir une manière spécifique d'encoder et de décoder des données sous forme de paquets qui seront par la suite transférés via un réseau informatique. Pour ce projet, nous avons décider de définir notre propre protocole pour les transferts de données entre différentes *Nodes*, sur-couche du protocole *TCP* (*Transfer Control Protocol*). De plus, nous utiliserons le protocole *HTTP* (*HyperText Transfer Protocol*) afin de communiquer avec le *Tracker*, programme hébergé sur un serveur et qui permettra aux pairs de connaître l'état actuel du réseau ORA, c'est-à-dire les *Nodes* actuellement connectés, les *Cluster* disponibles, etc…

Un *Node* est identifié par un couple de clé privé/publique tandis qu'un *Cluster* est identifié par un un *UUID (Universal Unique IDentifier)* généré lors de sa création.

Un *Node* ne possède pas forcément tous les fichiers disponibles sur le *Cluster*. Afin de synchroniser les fichiers qu'il lui manque, il doit passer par un *Tracker* qui lui renverra les informations d'un ou plusieurs *Node* possédant le fichier. Ces informations lui permettront ensuite de se connecter en *Peer-to-Peer* au(x) *Node(s)* possédant le fichier à jour dans le but de le télécharger.

Cette approche évite de stocker des fichiers chez une entreprise dans laquelle on n'a pas forcément confiance. Le but étant d'avoir un système sécurisé, il doit être impossible pour un *Node* extérieur à un *Cluster* de lire les fichiers qui lui sont associés. Nous utiliserons donc un système d'authentification asymétrique afin d'éviter cela. De plus, les fichiers sur le disque seront compressés et chiffrés, ce qui évitera les problèmes d'accès physique sans permission. Enfin, les paquets transmis via le réseau *Peer-to-Peer* seront aussi compressés et chiffrés de manière asymétrique.

Tout cela (création de *Cluster*, ajout de *Nodes*, etc …) sera gérable depuis une application en ligne de commandes ainsi qu'une interface graphique plus limitée. On devra aussi développer un site web afin de présenter le projet à l'utilisateur.



3.3 Répartition des tâches

Lors de la rédaction du cahier des charges, nous avions décidé de répartir les tâches de la manière suivante. Étant donné que tout le monde a touché à plusieurs parties du projet la répartition n'a pas été vraiment respectée, mais nous avions déjà signalé que la répartition n'avait pas grand sens pour nous.

Tâche	**Responsable**	**Suppléant**
Tracker	Léo	Raffaël
API	Léo	Adam
Core - Networking	Adam	Léo
Core - Chiffrement	Pierre-Corentin	Raffaël
Core - Compression	Pierre-Corentin	Adam
Core - File Management	Raffaël	Léo
Daemon	Adam	Raffaël
Application - CLI	Raffaël	Pierre-Corentin
Application - GUI	Raffaël	Léo
Site Web	Léo	Pierre-Corentin
LaTeX	Adam	Pierre-Corentin

3.4 Planning

Voici le planning des tâches que nous avons déterminé au début de l'année. Pour la première soutenance, l'avancement a bien été respecté. Pour la seconde soutenance, le planning a été moins respecté avec certaines parties en avances et d'autres en retard. Enfin, pour la dernière soutenance tout a été fait à 100%.

Tâches	1	2	3
Tracker	70%	90%	100%
API	50%	80%	100%
Core	40%	70%	100%
Daemon	10%	50%	100%
CLI	20%	80%	100%
GUI	0%	30%	100%
Site Web	40%	70%	100%
LaTeX	40%	70%	100%

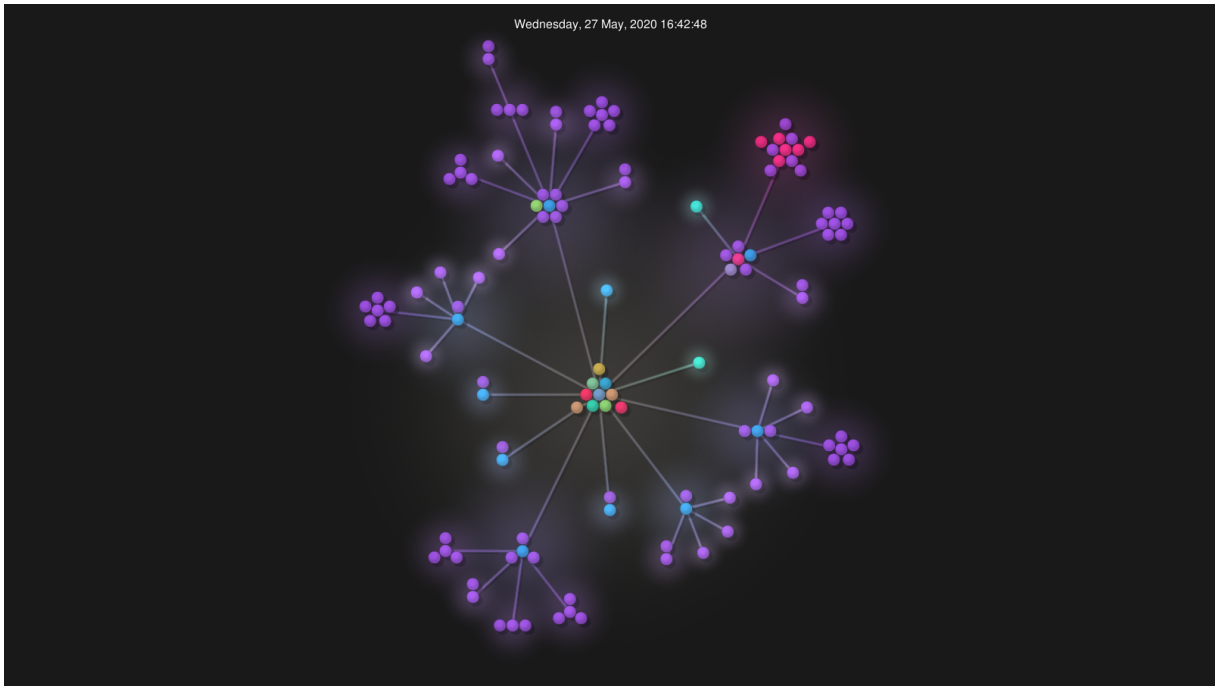


FIGURE 1 – Visualisation du projet sous forme d'arbre grâce au logiciel Gource

4 Division du projet

4.1 Introduction

Nous avons divisé le projet en plusieurs sous-projets ce qui nous a permis de le structurer. Cette division permet notamment une séparation entre le *Core* et les applications *CLI* et *GUI* qui permet à d'autres développeurs de créer leurs propres applications basées sur l'**API** de ORA. Étant nous même développeurs, la possibilité de construire nos propres applications sur une *API* de cette manière nous tient particulièrement à coeur.

Le découpage du projet est donc le suivant :

- **Tracker** un serveur qui centralise les informations des *Clusters* et permet aux *Nodes* de se connecter entre eux afin de partager des fichiers.
- **API** (*Application Programming Interface*) contenant un ensemble normalisé de classes, de méthodes, de fonctions et de constantes servant de façade aux applications voulant utiliser ORA (CLI, GUI, etc …).
- **Core** contenant toutes les implémentations des classes, méthodes et fonctions définies dans l'**API**.
- **Daemon** contenant l'application principale s'exécutant en arrière-plan, se basant sur les fonctionnalités du **Core**.
- **Application CLI** contenant l'outil en lignes de commandes servant à interagir avec les fonctionnalités du programme.
- **Application GUI** contenant l'outil en interface graphique servant à interagir avec les fonctionnalités du programme.
- **Site Web** nécessaire pour la communication avec les utilisateurs (téléchargement du programme, présentation du projet, etc …).

4.2 Tracker

Comme expliqué précédemment, le *Tracker* est le programme qui permet aux différents *Nodes* se connecter entre eux pour partager des fichiers, mais il stocke aussi les informations nécessaires des *Clusters* comme l'ID des membres. Pour le *Tracker* nous avons choisi d'utiliser le protocole HTTP (HyperText Transport Protocol) qui est un protocole de la couche Application (du modèle OSI) c'est le protocole utilisé pour le Web. Il est basé sur le protocole TCP (Transmission Control Protocol) un protocole de la couche Transport (du modèle OSI) qui permet un transport fiable en mode connecté, c'est-à-dire qu'il garantit qu'aucun paquet ne soit perdu et il fonctionne sur une connexion qui est établie.

Nous avons choisi d'utiliser le HTTP pour sa simplicité d'utilisation et la lisibilité de ses requêtes. Cependant, nous nous sommes rendu compte lors du développement de ORA que ce choix n'était pas très bon étant donné qu'avec le protocole HTTP une connexion TCP est ouverte puis refermé lors de la réception de la réponse. Si nous voulons garder une connexion ouverte avec le *Tracker* lorsque le client est connecté pour le notifier de diverses événements, ceci est impossible.

Pour remédier à ce problème, nous aurions dû utiliser le protocole TCP directement et définir notre protocole de couche Application afin de pouvoir garder une connexion ouverte et d'avoir moins de contrainte de le futur si nous avons besoin. Cependant, nous nous sommes rendu compte de ce problème trop tard et étant donné qu'il fallait réécrire tout le *Tracker* et le *Core* nous avons décidé de rester sur le protocole HTTP.

Le *Tracker* permet aux *Nodes* d'obtenir des informations sur les *Clusters* mais aussi de se connecter entre eux pour échanger des fichiers. Comme expliqué à la première soutenance nous avons réalisé qu'il fallait qu'on effectue de l'authentification du côté du *Tracker* pour éviter que n'importe qui puisse supprimer n'importe quel *Cluster* sans en être le propriétaire. L'authentification fonctionne de la manière suivante.

Lors du premier accès aux services ORA (par exemple après un redémarrage ou lors de la 1ère connexion), un utilisateur devra s'authentifier afin de recevoir un token (ou jeton en français) privé qui sera utilisé dans toutes les requêtes qui requiert l'autorisation pour ne pas avoir à effectuer le processus d'authentification (qui est lourd) à chaque fois. Afin de s'authentifier le *Member* envoie la clé publique de son *Identity*, le *Tracker* va générer un token et le chiffrer avec la clé publique reçue ce qui garantit que seul le détenteur de la clé privée pourra déchiffrer le token et donc l'utiliser.

Le *token* obtenu expire au bout de quelques minutes si l'utilisateur n'a effectué aucune requête durant cette période. Chaque requête effectué avec ce token permet de le rafraîchir donc de repousser sa date d'expiration ce qui évite que les utilisateurs actifs aient à régénérer un token à chaque fois. Mais l'utilisateur ne va pas mémoriser son token, c'est donc le programme qui s'en chargera pour lui, de même que pour le rafraîchir. C'est donc au rôle du *Daemon* d'effectuer ces opérations, afin de pouvoir continuer à partager les fichiers.

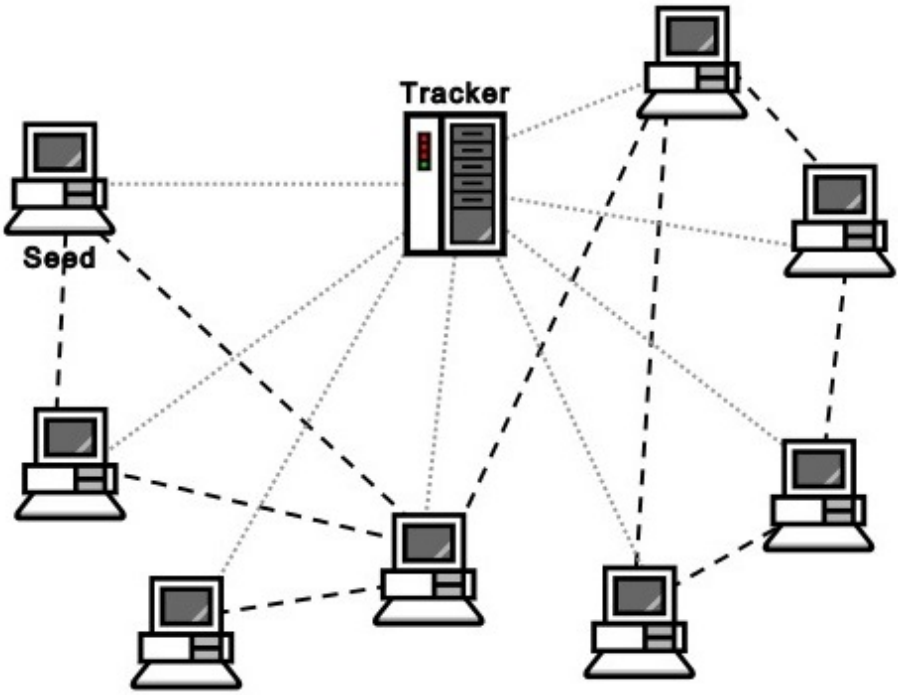


FIGURE 2 – Schéma d'un tracker

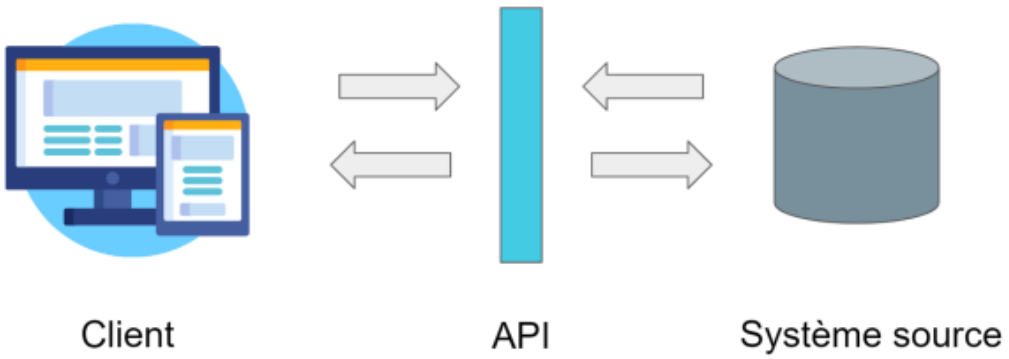
4.3 Daemon

Le **Daemon** est le service qui tourne sur le PC de l'utilisateur même lorsqu'il n'utilise pas explicitement ORA, c'est lui qui va gérer toute les tâches que l'utilisateur n'aura pas besoin de faire lors de son utilisation des applications, que ce soit l'authentification, la synchronisation des fichiers, etc….

Pour cela, le *Tracker* va pouvoir communiquer avec les applications au travers d'une version spécifique du Core qui implémente de l'*IPC* (Communication inter-processus) grâce à la bibliothèque *IpcServiceFramework*. Ainsi, lorsqu'une application voudra accéder à quelconque donnée d'ORA, que ce soit la liste des clusters, l'URL du **Tracker** ou bien le contenu d'un fichier, c'est le **Daemon** qui effectuera ces actions et qui renverra la réponse.

4.4 Application Programming Interface (API)

L'**API** permet en quelques sortes de définir les objets composants ORA sans pour autant fournir d'implémentation à ceux-ci. C'est en quelque sortes une collection de lignes directrices que doit suivre un programmeur travaillant avec ORA. De plus, la présence d'une API facilite fortement la généricité du code mais aussi sa compréhension pour les nouveaux développeurs voulant programmer une application utilisant ORA. En effet, le développeur n'a pas à se soucier de l'implémentation derrière, il a uniquement accès aux noms des méthodes et à ce qu'elles renvoies.



4.5 Core

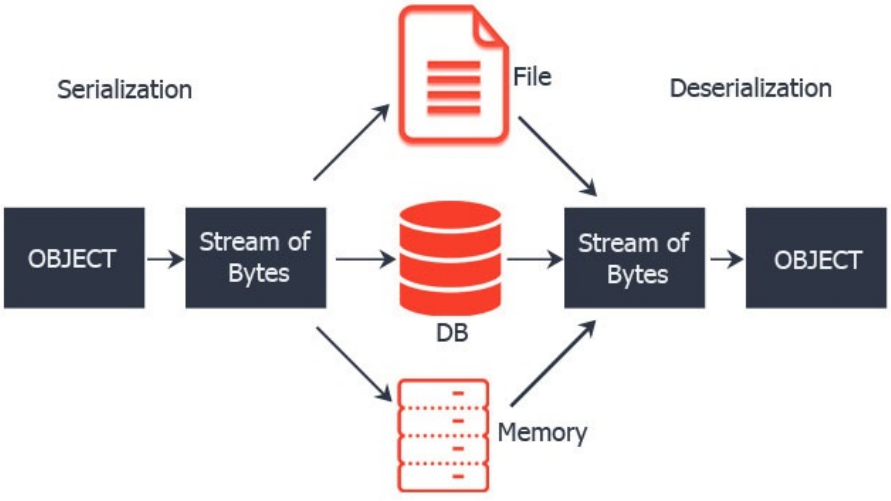
Le *Core* représente l'implémentation par défaut de l'*API* d'ORA. C'est grâce à lui que tout fonctionne et que les fonctions décrites dans l'*API* renvoient le bon résultat. Ainsi, il est découpé de la même manière que l'*API*, avec des *Managers* permettant de gérer les modèles de données tels que les *Clusters*, les *Files*, etc…

4.5.1 Protocole

Afin de transmettre les fichiers d'un pair à un autre, il nous a fallu écrire un petit protocole de transmission de paquets au dessus du protocole *TCP*. Ce protocole est très simple puisqu'il s'agit d'un bête paquet TCP contenant les données identifiant le type de paquet (demande de fichier ou transfert de fichier) ainsi que les données du paquet elles mêmes. Voici un tableau récapitulatif de ce protocole.

Packets	FileRequest	File
Packet ID	0	1
Data Length	Taille du champ de données	Taille du champ de données
Data	Cluster Hash du fichier	Cluster Hash du fichier Contenu du fichier

Ainsi, on peut voir que ce système permettra l'ajout de potentiels futurs paquets gérant possiblement d'autres fonctionnalités. On remarque aussi que le champs contenant les données peut contenir n'importe quoi, tant que la manière pour encoder et décoder ces données sont correctement décrites dans la classe représentant le Packet avec les méthodes abstraites *Serialize() -> byte[]* et *Deserialize(byte[]) -> void*.



4.5.2 Compression

La compression, implémentée dans le Core, est une composante majeure du partage de fichier, car elle permet de réduire de manière substantielle la taille des fichiers. Nous avions choisi à la base d'utiliser la bibliothèque Zstandard, car elle offre un des meilleurs Ratio compression/vitesse de compression dans des conditions optimales, et est de plus en licence libre. Cependant, nous nous sommes rendu compte lors du développement que la bibliothèque Zstandard n'était pas compatible avec Linux, système d'exploitation sur lequel nous voulons que notre application soit utilisable, et avons décidé de passer à la bibliothèque Zlib qui offre elle aussi un très bon ratio en plus d'être aussi en licence libre.

Zlib implémente le format de compression de données Deflate, qui couple deux algorithmes de compression : LZ77 et le codage de Huffman. LZ77 va venir remplacer les occurrences de données par une référence à la première apparition de la donnée. Le codage de Huffman va utiliser un arbre binaire dans lequel les caractères sont stockés dans les feuilles de manière à ce que les caractères les moins fréquents soient dans les feuilles les plus profondes de l'arbre, puis encoder les caractères selon l'occurence de la feuille, ce qui permet d'en faire tenir plusieurs, chacun initialement codé sur un octet, sur un seul octet.

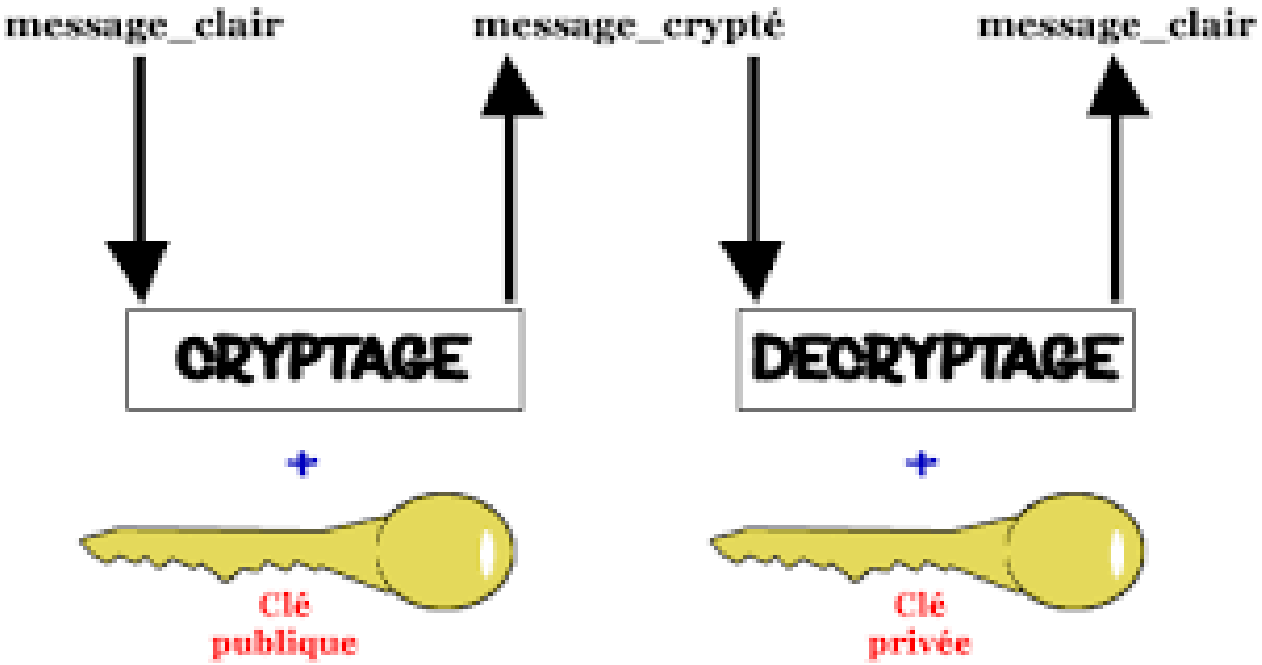
Compressor name	Ratio	Compression	Decompress
zstd 1.3.4 -1	2.877	470 MB/s	1380 MB/s
zlib 1.2.11 -1	2.743	110 MB/s	400 MB/s
brotli 1.0.2 -0	2.701	410 MB/s	430 MB/s
quicklz 1.5.0 -1	2.238	550 MB/s	710 MB/s
lzo1x 2.09 -1	2.108	650 MB/s	830 MB/s
lz4 1.8.1	2.101	750 MB/s	3700 MB/s
snappy 1.1.4	2.091	530 MB/s	1800 MB/s
lzf 3.6 -1	2.077	400 MB/s	860 MB/s

FIGURE 3 – Comparaison de plusieurs algorithmes de compression
Source : https://facebook.github.io/zstd/

4.5.3 Chiffrement

Implémenté dans le core, le chiffrement des données est ce qui permet de s'assurer de la sécurité du stockage des fichiers sur l'application. Nous avions choisi au début du projet d'utiliser le chiffrement RSA, un algorithme de chiffrement asymétrique que nous avons choisi pour sa sécurité. En effet, il est aujourd'hui impossible de trouver les clés grâce à la force brute sans des moyens matériels conséquents, et que la taille maximale de la clé trouvée par cette attaque est de 759 bits.

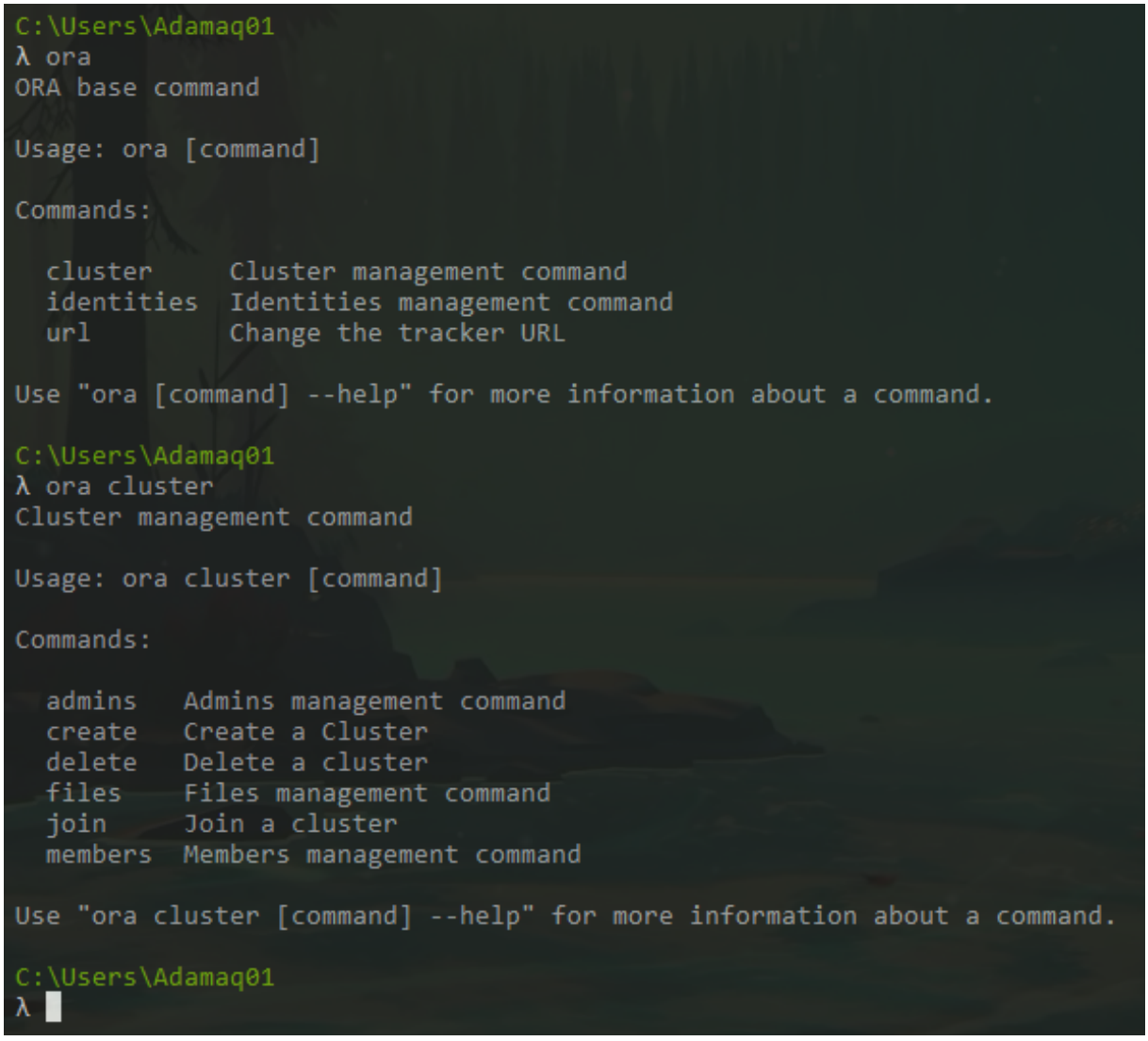
Cet algorithme se base sur un coefficient **n** obtenu en multipliant deux nombres premiers distincts **p** et **q**, d'un entier naturel **e** premier avec (p-1)(q-1), et de **d** l'inverse de e sur (p-1)(q-1). Le message encodé est le modulo n de l'exponentielle du message original par e, et on retrouve le message original en faisant le modulo n de l'exponentielle du message encodé par d.



4.6 Command Line Interface (CLI)

Le *CLI* représente la façon la plus simple pour quelqu'un qui y est habitué d'intéragir avec ORA. Toutes les fonctionnalités y sont implémentées via un système de commandes et d'arguments, permettant à l'utilisateur de gérer ses clusters, ses identités, ses fichiers, etc... Bien que compliqué à prendre en main au début, ce sera sans doute votre meilleur ami lorsque vous aurez des soucis avec les autres moyens d'intéragir avec ORA.

De plus, le CLI reste constamment à jour avec l'*API* et le *Core* puisque dès qu'une nouvelle fonctionnalité y est implémentée il est très facile de l'intégrer grâce à la bibliothèque CommandDotNet.



4.7 Graphical User Interface (GUI)

Une Interface Graphique est nécessaire à toute application, car à tout moment elle pourrait être utilisée par des personnes n'ayant peu, voir jamais utilisé d'interface en ligne de Commande. Comme le CLI, elle implémente toute les fonctionnalités en plus de permettre à l'utilisateur de les visualiser sur une fenêtre. Il pourra ainsi dans l'onglet *Home* visualiser les clusters dont il est membre et en ajouter un, où bien dans l'onglet *Settings* de changer l'URL du Tracker où son pseudonyme.

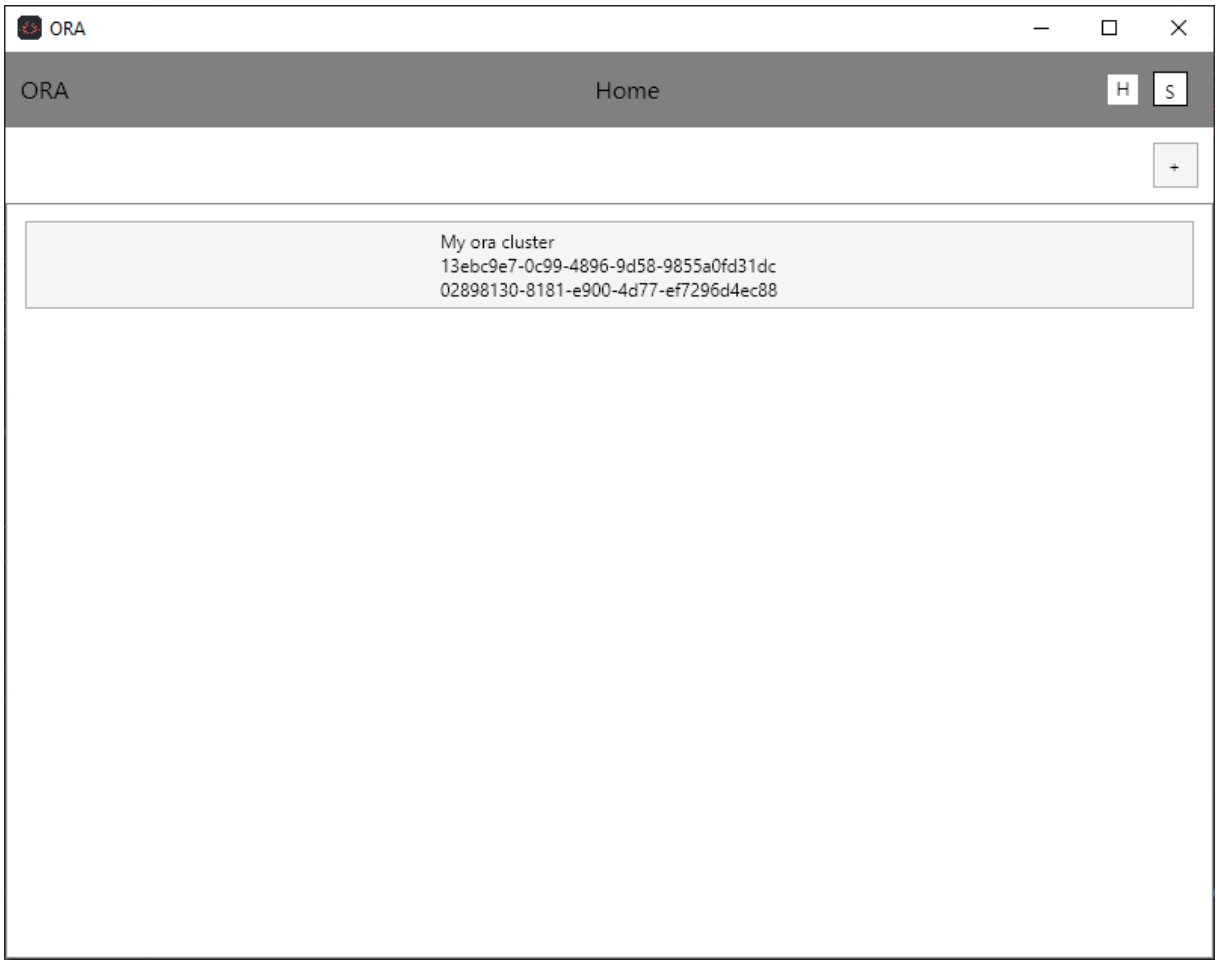


FIGURE 4 – Page d'accueil du GUI

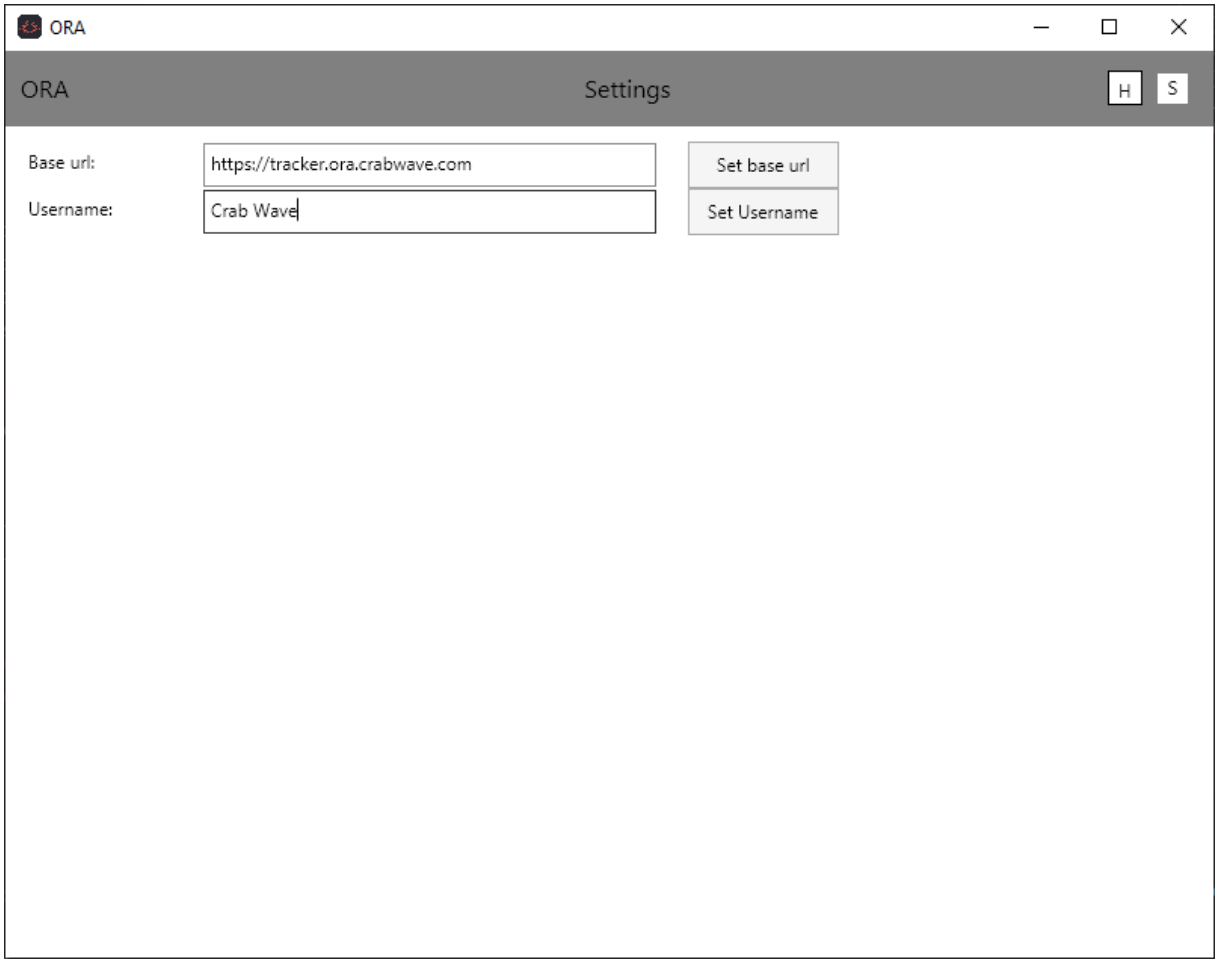


FIGURE 5 – Paramètres du GUI

Nous avons rencontré quelques problèmes lors du développement de l'application *GUI*. Au début du projet, nous avions prévu d'utiliser la bibliothèque *AvaloniaUI* afin de développer l'application. *AvaloniaUI* utilise le patron de conception Modèle-Vue-Controlleur qui permet de séparer le code afin de garantir une meilleure maintenabilité et testabilité. Cette bibliothèque est multi-plateforme et est compatible avec *.NET Core* ce qui satisfaisait nos contraintes. Cependant la documentation de cette bibliothèque et les exemples disponibles étant très légers, nous n'avons pas réussi à faire fonctionner le routing, qui est le processus nous permettant de changer de vue.

Nous nous sommes donc tourné vers une autre librairie qui satisfaisait aussi nos contraintes : *Electron.NET*. *Electron* est un framework qui permet de créer des applications de bureau en *HTML*, *JavaScript* et *CSS*. *Electron.NET* est un wrapper de *Electron* utilisant l'*IPC* qui permet d'écrire les controlleurs en C# et les vues en CSHTML. Tout d'abord, la construction de l'application était impossible, nous avons finalement découvert que c'était un *paquet NPM* dont l'installation était incomplète lors de la construction du projet qui causait ce problème. Une fois les problèmes de construction de l'application résolus, un autre problème est apparu : il était impossible d'ouvrir une fenêtre.

Après de nombreuses tentatives de résolution de ce problème, nous avons décider de revenir à l'utilisation de *AvaloniaUI*. Le *GUI* était à un stade assez précoce, il comportait une seule vue étant donné que nous avons des problèmes avec le routing comme mentionné précédemment. Cependant, la veille du rendu nous avons réussi à faire fonctionner le routing donc l'application bénéficie de plusieurs vues pour la démonstration. Actuellement, l'application nous permet de créer de changer l'*URL* de base du *Tracker*, de créer ainsi que de voir le noms des *Clusters* auquel l'utilisateur a accès.

Bien qu'*AvaloniaUI* soit plutôt simple à prendre en main, nous avons eu à de nombreuses reprises des difficultés à l'utiliser de par une documentation manquant souvent d'explications, mais aussi à cause du peu d'exemples d'utilisation disponibles. Nous avons donc eu dû mal à faire tout ce que nous souhaitions avec cette bibliothèque.

FIGURE 6 – Logo d'Avalonia

4.8 Site Web

Aujourd'hui un site *Web* est un élément indispensable à tout projet afin de le promouvoir ou de le présenter d'une manière claire et précise. Nous avons fait le choisi de développer un site *Web* avec :

- une page principale contenant la description du projet, des liens vers les membres du groupe et les liens de téléchargement du projet
- une page de documentation de l'*API* de ORA pour permettre aux développeurs de créer leurs propres applications basées de notre *API*

Le site Web que nous avons développé est dynamique, c'est-à-dire que les pages sont générées à la volée. Cela permet par exemple pour la page de documentation d'écrire une unique page qui va être remplie dynamiquement avec les informations de documentation de la classe de l'API à laquelle correspond la page actuelle. Le site comporte un frontend qui a été écrit en *JavaScript* à l'aide de la bibliothèque React.js qui permet d'écrire des sites Web dynamique. Aussi, ne comporte pas de backend étant donné que nous n'en avons pas l'utilité. Pour ce qui est de l'hébergement, nous avions prévu au début d'utiliser *Netlify* mais nous avons migré vers now.sh de ZEIT qui est gratuit comme Netlify mais qui permet à plusieurs membres d'avoir des accès d'administration pour le site. Nous avons aussi acheté le nom de domaine crabwave.com et le site est actuellement disponible à l'adresse suivante : https://ora.crabwave.com. Étant donné que pour la première soutenance nous n'allions pas avoir beaucoup de choses à montrer nous avons misé sur l'avancement du site Web pour avoir des choses à montrer graphiquement.

4.8.1 Page principale

Le design du site *Web* est très sobre avec quelques nuances de gris et de noir. Le style a été écrit en Sass, une extension du *CSS* qui a des fonctionnalités bien pratique comme le namespacing. La page principale contient une barre de navigation permettant de naviguer entre les différentes sections et de se rendre sur la page de documentation. La page principale contient plusieurs sections :

- **Accueil** contenant le nom du projet et une phrase d'accroche présentant en quelques mots le projet
- **Présentation** contenant une description plus fournies du projet pour les utilisateurs souhaitant en savoir un peu plus
- **Groupe** contenant les noms de tous les membres du groupe ainsi que leurs *GitHub*, adresse mail et *LinkedIn*
- **Téléchargements** référençant les liens de téléchargement du projet avec les documents des soutenances, le code source du projet et le téléchargement du projet
- **Nous contacter** contenant les liens de contact du groupe Crab Wave

La barre de navigation a été écrite sous la forme d'un composant *React.js* ce qui permet de la réutiliser d'écrire une seule fois le code et de la réutiliser pour la page de Documentation qui va avoir besoin de cette même barre de navigation.

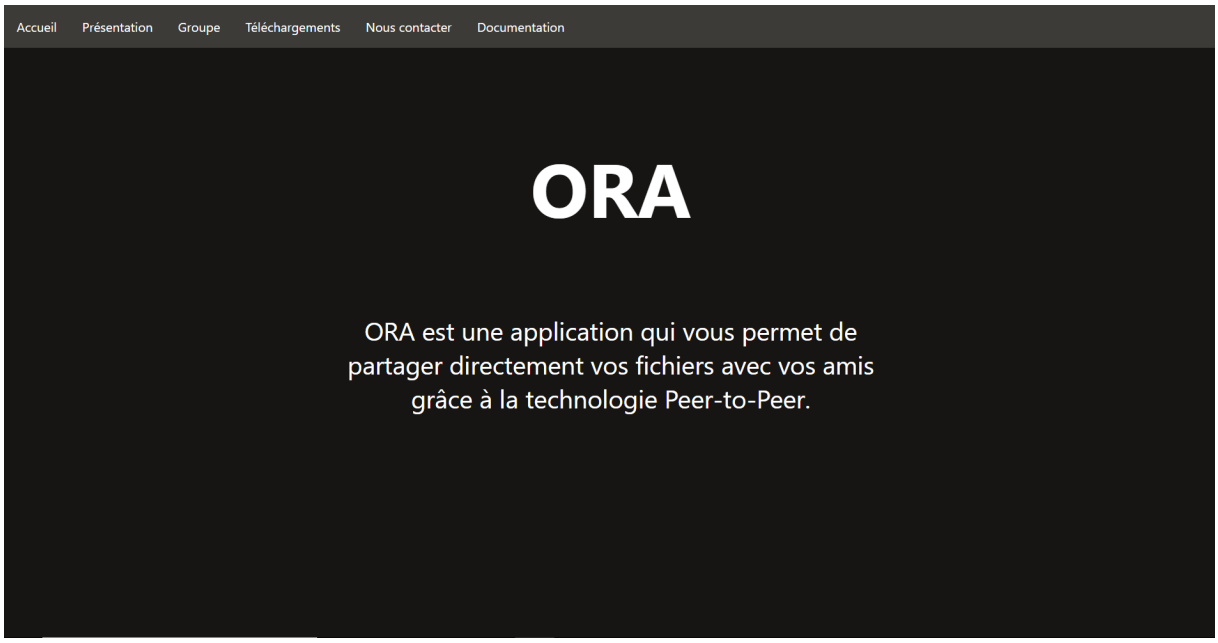


FIGURE 7 – Page d'accueil du site internet

4.8.2 Documentation

La page de documentation est très importante pour nous puisque c'est ici que les développeurs souhaitant créer leur propres applications sur l'**API** d'ORA ou contribuer à nos applications pourront avoir toutes les informations et les détails de chaque classe et chaque méthode de l'**API** qui constituent la base de notre projet. La page de documentation contient une page principale permettant d'introduire l'API à l'utilisateur et les pages de documentations de chaque classe de l'API. Chaque page contient une barre de navigation dont nous avons déjà parlé dans la partie précédente, et un bandeau sur le côté gauche permettant de naviguer entre les différentes pages de documentation.

La page de documentation est générée dynamiquement grâce à la bibliothèque React.js et grâce à un document JSON contenant la documentation de toutes l'API. Tout le code de l'API est documenté avec des commentaires au format XML. La commande dotnet doc permet de générer un fichier contenant la documentation au format XML que nous avons écrits dans le code. Nous avons développé un script en JavaScript qui permet de récupérer ce fichier de documentation au format XML et de le convertir au format JSON.

Le format JSON (JavaScript Object Notation) est une représentation sous forme de texte des objets JavaScript ce qui est très pratique car il suffit simplement de parser le fichier avec une méthode déjà présente dans le langage pour avoir accès à l'objet correspondant.

```
<example>
    The following code
    <code>
        ICipher cipher = ...;
        byte[] data = ...;
        byte[] encrypted = cipher.Encrypt(data);

        Console.WriteLine(cipher.Decrypt(encrypted).Equals(data));
    </code>
    will print the value true.
</example>
```

FIGURE 8 – Documentation du chiffrement au format XML

En transformant le fichier XML en JSON nous créons un champ correspondant à chaque classe de l'*API* et ce champ contient il y a la liste des méthodes ainsi que leurs prototypes, documentations, etc…

Le fait de formater de cette manière permet que lorsque l'utilisateur souhaite accéder à la documentation d'une classe de notre côté nous avons juste accéder au champ correspondant à la classe.

Grâce à cette méthode il nous suffit d'écrire simplement la documentation des classes et des méthodes dans l'API et la page correspondante sera générée dynamiquement.

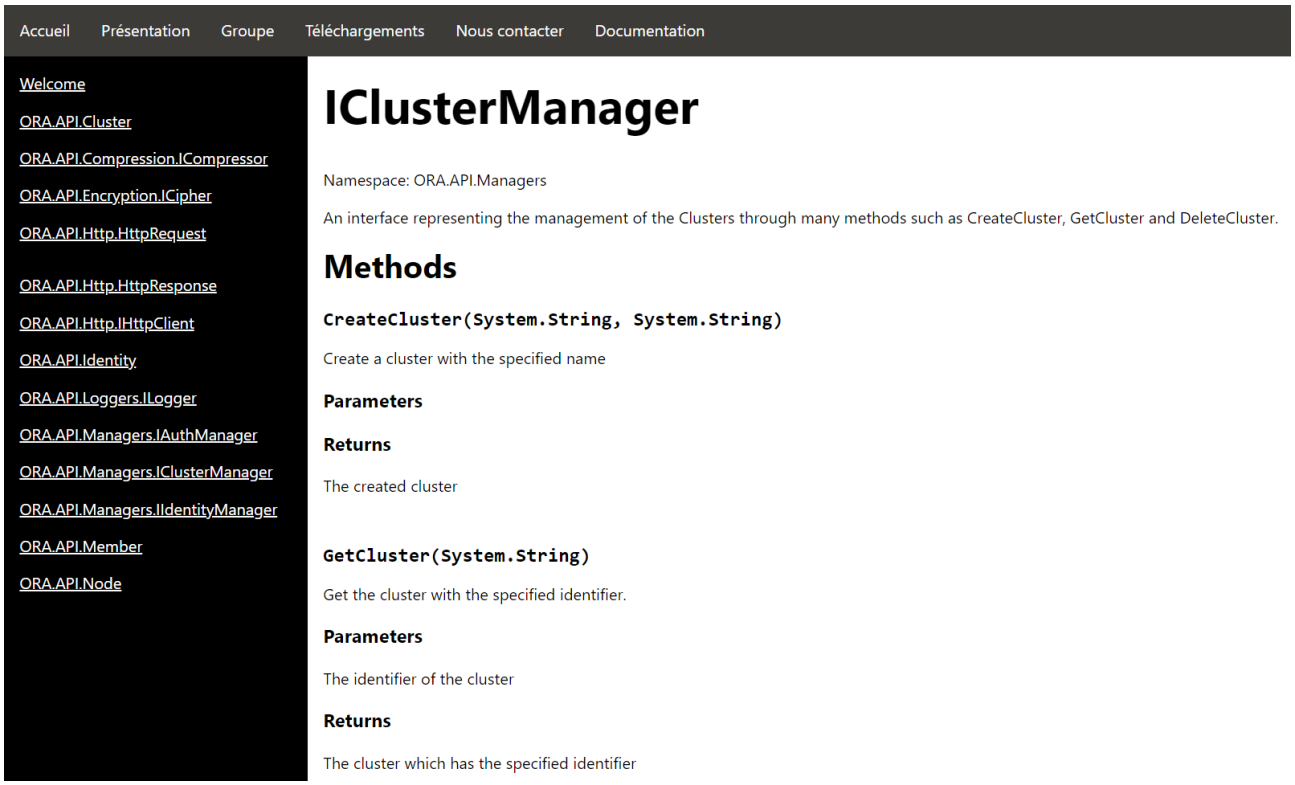


FIGURE 9 – Documentation générée par le fichier XML sur le site web

5 Outils et Méthodes utilisées

Dans cette partie, nous allons parler de toutes les méthodologies qui seront utilisées afin de mener à bien le développement du projet ORA.

Nous utiliserons dans ce projet le ***Test-Driven Development*** (TDD) qui est une méthode de développement logiciel qui consiste à écrire les tests unitaires avant d'écrire le code source. Un test unitaire est une procédure vérifiant le bon fonctionnement d'un composant (appelé « unité ») du programme. On peut ainsi résumer le TDD en trois phases distinctes :

1. Écrire un test qui échoue (puisqu'il n'y a pas de code couvrant ce test).
2. Écrire le minimum de code afin de faire passer le test.
3. Réusiner le code afin de retirer la duplication de code et de le rendre plus lisible.

Le TDD va donc nous permettre d'avoir un code plus lisible et plus simple ainsi que de réduire le nombre de bugs et la duplication de code.

Le *workflow* que nous avons choisi pour la gestion du dépôt *git* est le ***GitHub Flow***. Ce *workflow* est plus léger que le classique *git-flow* et repose sur quelques règles simples :

1. Tout ce qui est présent sur la branche *master* est toujours déployable en production, ainsi *master* reste une branche stable.
2. Pour développer une fonctionnalité il faut créer une branche avec un nom explicite à partir de *master*.
3. Pour fusionner une branche avec *master* il faut ouvrir une *pull-request* permettant aux développeurs de discuter des modifications ce qui garantie une meilleur communication au sein du groupe.
4. Une fois que les développeurs se sont mis d'accord sur les modifications, ils doivent réaliser une revue de *pull-request* ce qui facilite la détection d'erreurs.
5. Utiliser les *issues* afin de lister les problèmes rencontrés et pouvoir assigner leurs résolution à quelqu'un.

De plus, nous utiliserons l'**intégration continue** afin de réaliser ce projet. Cette pratique consiste à vérifier que chaque modification de code ne produit pas de régression, c'est-à-dire qu'il n'introduit pas de défaut. Les principaux avantages de l'intégration continue sont la détection d'erreurs plus rapide, ce qui permet dont de les localiser plus facilement.

A chaque modification, la pipeline d'intégration continue que nous allons créer effectuera les tests suivants :

1. Test de compilation, vérifie que l'application compile.
2. Test unitaire, vérifie que l'entièreté des tests unitaires soient satisfaits (ces tests seront produit grâce au TDD).
3. Test de convention de code, vérifie que le code respecte les conventions définies au préalable par le groupe.

Pour garantir que la branche *master* est stable, nous allons paramétrer le dépôt *GitHub* pour qu'il soit impossible de fusionner la branche *master* avec une quelconque autre branche si la pipeline d'intégration continue échoue.

De plus, nous allons découper le développement du projet en plage de temps de 2 semaines nommées **sprints**. Cette durée a été choisi du fait que la durée entre chaque soutenance est d'un mois et demi et que l'on ne travaille pas à temps plein sur le projet. A chaque début de sprint chacun choisit les taches qu'il souhaite embarquer. Ces taches sont définies par des ***User Stories*** et leur difficulté est estimée par des ***Story Points***.

Une *User Story* est la description d'un besoin qui permet de déterminer les fonctionnalités a développer. Elle est souvent présentée sous la forme : « En tant que X, je veux Y afin de Z ». Tandis que les *Story Points* permettent d'estimer l'effort à fournir pour réaliser une tache. C'est l'équipe qui défini ensemble ce qu'est une *User Story* de taille moyenne, qui permet après de définir les *Story Points* associés aux autres. Cela évite donc les estimations en jours-hommes qui dépendent d'une personne à une autre. Ainsi, les *User Stories* sont pondérées par les *Story Points* qui leur sont associés.

Pour finir, nous avons décidé de mettre en place une réunion durant une heure tous les lundis afin de faire un bilan sur la semaine passée, de prendre conscience des tâches qu'il reste et de planifier ce qui est à faire pour la semaine à venir.

5.1 Outils utilisés

Afin de développer l'application et de mettre en place les méthodologies choisies, nous utiliserons les outils suivants :

- C#
- git
- GitHub
- LaTeX
- Travis CI

Pour le CLI nous avons utilisé la bibliothèque CommandDotNet et pour le GUI nous avons utilisé la bibliothèque Avalonia UI.

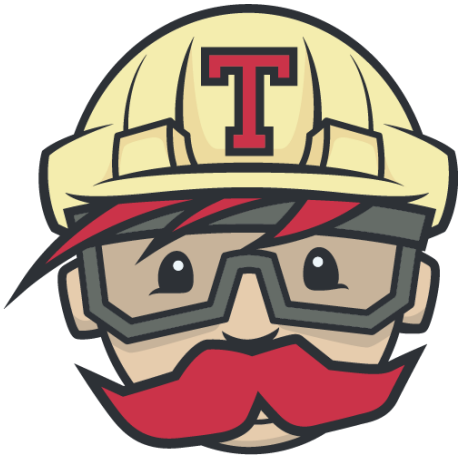


FIGURE 10 – Logo de Travis CI

De plus, nous avons choisi de développer un site Web dynamique. Ainsi, les outils suivants s'ajoutent à la liste, en plus de ceux précédemment cités :

- React.js
- Sass
- Vercel (précédemment appellé ZEIT)

6 Déroulement du projet

Dans cette partie, nous détaillerons comment s'est passé le déroulement global du projet au long de cette année, à partir du moment où nous avons eu l'idée d'ORA et que le groupe était formé.

6.1 L'avant cahier des charges

Avant même d'avoir commencé à rédiger le cahier des charges, nous avons commencé à réfléchir à différentes manières de structurer le projet. Que ce soit au niveau de l'architecture du code, de l'hébergement du site, du tracker, du nom de domaine, etc…

Nous nous sommes donc orientés assez vite vers des plateformes gratuites de préférence et dont certains d'entre nous avaient l'habitude d'utiliser. Nous avons donc opté pour héberger le dépôt git sur Github dans une Organisation, d'utiliser Travis-CI afin de faire des tests quotidiens.
Pour ce qui est de l'hébergement du Tracker, certains services gratuits auraient pû faire entièrement l'affaire, mais Adam possède un VPS (*Virtual Private Server*) et a donc décidé de l'héberger dessus puisque ce serait plus simple à maintenir pour lui. A ce moment là, notre choix se portait vers Netlify pour l'hébergement du site internet, Léo s'étant déjà familiarisé avec cet environnement.

Nous avons donc passé une bonne semaine des vacances de Noël à mettre tout celà en place, que ce soit les dépôts git, les projets C-Sharp, l'automatisation des tests, la création du logo ainsi que du template LaTeX que nous avons utilisé jusque là fortement inspiré des classes open-source AwesomeCV disponibles sur github.
Suite à cela, nous avons planifié plusieurs réunions afin que l'ont soit tous bien au courant de la manière dont le projet était géré, etc….

6.2 L'après cahier des charges

Après la validation de notre cahier des charges par Mr. Boullay, nous nous sommes tout de suite attaqué au projet. Léo avait commencé la structure du **Tracker**, Adam de l'**API** et Pierre-Corentin et Raffaël commençaient à implémenter les fonctionnalités décrites par l'**API** dans le **Core**.

Ainsi, Pierre-Corentin a pû mettre en place le chiffrement dans le **Core** et Raffaël les débuts de la gestion des **Clusters** tandis que Léo travaillait sur le **Tracker** et le **Site Internet**. Adam a mis en place la mise à jour automatique de tout ceci sur son VPS, afin de garder un *Tracker* toujours au même niveau que la branche master sur le dépôt Github.

6.3 L'après première soutenance

Après la première soutenance auprès de Mr. Ternier, nous avons chacun pris une pause dans la réalisation du projet afin de nous adapter au travail au début du confinement, ainsi que pour travailler sur un TP de Programmation assez compliqué.
Adam s'est chargé de continuer le **Daemon**, l'**IPC**, et a refactoriser le chiffrement, tandis que Raffael a implémenté les Membres dans le **Core** et a continué l'implémentation du **CLI**. Pierre-Corentin avait déjà commencé à écrire en partie la **Compression** avant la première soutenance, mais a dû réécrire cette partie suite au changement de bibliothèque. Léo s'est chargé de continuer le **Tracker** et le **Site Internet**, ainsi que de mettre en place la base du **GUI**.
Enfin, tout le monde s'est chargé de documenter le code déjà écrit, que l'on retrouve sur le Site.

6.4 L'après seconde soutenance

Après l'envoi de la vidéo de seconde soutenance à Mr. Ternier, nous avons repris assez rapidement le travail, tout en continuant de travailler les cours en ligne.
Léo, Raffael et Pierre-Corentin ont travaillé le **GUI** afin d'y implémenter les méthodes présentes dans le CLI ainsi que pour lui donner un peu de style. De plus, Léo à de son côté fini le **Tracker**. Enfin, Adam s'est chargé d'implémenter le **File Management** et a implémenté ces méthodes dans le **CLI**.

7 Expériences personnelles

7.1 Léo Benito

Je n'ai pas l'habitude de travailler sur des projets aussi gros et long donc ce projet a été pour moi une expérience très enrichissante. C'était une des premières fois que j'étais chef de projet et pour le coup gérer un gros projet sur son temps libre à côté des cours n'est pas facile.

Je suis conscient que je ne suis pas le meilleur chef de projet et que j'ai fait beaucoup d'erreurs mais ça m'a permis d'apprendre pleins de choses. Ce projet m'a permis de beaucoup m'améliorer en programmation orienté objet dont je ne connaissais que les bases.

J'ai aussi appris qu'il fallait passer plus de temps à réfléchir en amont avant de développer un projet pour éviter de refactoriser le code tout le temps surtout sur une base de code de cette taille (qui commence à être un peu grosse). Le fait d'avoir définis des conventions de codes et une pipeline de test était très intéressant pour avoir un code normalisé et qui fonctionne.

Je pense que nous aurions du définir des conventions pour le nommage des branches sur git et définir une guideline pour que le code soit plus normalisé parce qu'il est actuellement normalisé au niveau de la syntaxe mais pas de la sémantique. Nous avions prévu de faire du TDD ce que nous avons plus ou moins abandonné, le TDD est vraiment une autre manière d'aborder le problème très intéressante mais dont nous ne sommes pas trop habitué et nous manquons un peu de rigueur actuellement.

De mon côté j'ai continué à écrire des tests et j'ai dû apprendre à rendre en mon code testable en utilisant certains patrons de conception. Pour ce qui est des sprints et de l'organisation nous l'avons suivi jusqu'à l'arrivé du virus en France malheureusement.

Je suis aussi très content de mon groupe, ils étaient tous de bonne volontée et n'ont jamais refusé le travail ils ont su s'adapter malgré la difficulté du projet.

7.2 Raffaël Moraisin

ORA étant mon premier projet de groupe de cette envergure, je pense qu'il m'a été très bénéfique. En effet, cela m'a apporté de l'expérience dans le travail en équipe ainsi que dans le travail personnel. Cela se ressent également en dehors du projet, lorsque j'ai un problème que je n'arrive pas à résoudre, je suis plus enclin à en discuter avec mes camarades ce qui me permet fréquemment de trouver des solutions.
L'expérience que m'a apporté ce projet se voit lorsque je veux faire un bonus à un TP de

programmation. Par le passé, j'étais cantonné dans ce que je savais faire, alors qu'aujourd'hui je peux passer des heures dans la documentation pour réussir à faire ce que je veux faire. J'ai vraiment beaucoup aimé travailler sur le projet.

En effet, comme le fait de créer une application et de la voir grandir de nos yeux est très ludique, celui-ci me créait une coupure avec les cours classiques et me permettait donc de faire des pauses sans perdre mon temps. Le fait d'être responsable du GUI était également très plaisant. Cela m'a permis d'apprendre à créer une interface d'une application. Enfin, je voudrais remercier mes partenaires de projet : Adam, Léo et Pierre-Corentin.

En effet, ils n'ont jamais dit non au travail et je pense que si ORA est ce qu'il est, c'est avant tout grâce à eux. De plus, le projet s'est déroulé dans la bonne ambiance tout du long, et je pense que j'en garderai un bon souvenir.

7.3 Adam Thibert

Pour ma part, j'ai beaucoup apprécié travailler sur ce projet, j'ai pu apprendre beaucoup de choses, notamment au niveau du Peer-to-Peer qui est une technologie que j'utilisais mais dont je n'avais jamais vraiment étudié le fonctionnement.

Je n'avais aussi jamais fais d'IPC, ou du moins pas d'une telle manière. Ainsi ce projet qui paraît assez simpliste en amont m'a en fait posé beaucoup de problèmes au niveau de la conception et je pense que c'est là dessus que j'ai le plus appris. En effet, lorsque je réfléchissais à une manière d'implémenter quelque chose, il me fallait toujours plusieurs tentatives afin d'avoir un résultat qui fonctionnait correctement et dont j'avais réussi à résoudre toutes les problématiques qui se sont posées avec les tentatives précédentes.

Au delà de ça, je me rend aussi compte que certains choix de technologies n'étaient pas forcément les bons, et que si nous avions réfléchis et posé l'architecture du programme dès le début en essayant de réfléchir à un maximum de problématiques, nous aurions éviter certains problèmes qui sont maintenant inhérents à ORA.

Au delà des contraintes techniques, j'ai toujours eu du mal à expliquer correctement mes propos aux personnes avec qui je travaille, et ça a bien entendu été le cas lors de ce projet aussi où je n'arrivais pas forcément à faire comprendre mes idées aux autres membres du groupe. Mais je pense que cela à évolué positivement puisque travailler avec des débutants en programmation comme Pierre-Corentin ou Raffaël m'a aidé à devenir un peu plus pédagogue dans ma façon d'expliquer certaines notions.

Enfin, au delà de tout ça, ce projet reste une expérience très enrichissante, que ce soit au niveau technique ou humain, ce qui est toujours bon à prendre car je pense que c'est par le biais d'expériences comme celles-ci que l'on apprend le plus.

7.4 Pierre-Corentin Auger

Bien que ça ne soit pas la première fois que je travaille sur un projet de groupe aussi conséquent, c'était la toute première fois que je faisais un projet de programmation en groupe.
Au début, je me sentais assez stressé, car je suis celui dans le groupe qui avait le moins de connaissances en programmation en entrant à EPITA, mais cela a fini par se dissiper après avoir commencé à attaquer le projet.
La période entre la remise du cahier des charges a été assez difficile au début. N'ayant jamais utilisé GitHub, ni travaillé, comme précisé plus haut, sur un projet de programmation en groupe, je n'avais aucune idée de comment utiliser un repository commun, ni de comment créer une branche, la fusionner ou travailler dessus.

J'ai cependant rapidement appris après avoir demandé de l'aide à mon groupe, puis l'avancement s'est déroulé sans accroc à partir de ce point là. J'étais principalement chargé d'implémenter les méthodes de chiffrement et de déchiffrement des données, et pour ce faire, je me suis renseigné sur la documentation .Net Core. J'ai ainsi découvert qu'il implémentait le namespace RSACryptoServiceProvider, permettant de chiffrer les données grâce à cet algorithme.

Pour écrire les premières versions, je n'ai pas eu trop de mal, mais cela s'est compliqué lorsque les tests de Travis CI m'ont indiqué qu'il était impossible de stocker la clé dans un conteneur, ce que j'avais envisagé au début. J'ai donc réécris mon code afin d'enlever toute allusion au conteneur, puis ensuite pour des modifications mineures.

La période entre la première et la seconde soutenance aura été la plus difficile pour moi. J'ai eu au début beaucoup de mal à me réadapter au rythme avec le confinement, ce qui m'a causé quelques soucis pour me remettre au travail au début, puis après m'y être remis, j'ai commencé à travailler sur la compression.

Après avoir fini la première version de la compression, j'ai rapidement découvert un problème majeur : la bibliothèque Zstandard n'est pas compatible avec le système d'exploitation Linux. Nous avons du donc changer pour la bibliothèque Zlib.
J'ai eu des difficultés pour réécrire les algorithmes de compression et de décompression, car là où les méthodes Compress et Decompress étaient très simples à utiliser avec Zstandard, les méthodes Deflate et Inflate de Zlib étaient beaucoup plus compliquées à utiliser, et j'ai eu beaucoup de mal à trouver des exemples sur internet pour voir comment les utiliser correctement, mais ai pu en trouver grâce aux membres du groupes.

La période avant la soutenance finale est au final celle où j'ai eu le moins de mal. Bien que la fatigue de fin d'année se fasse sentir, j'ai eu relativement peu de mal à comprendre et utiliser le XAML pour créer le GUI avec Avalonia.
Ce projet m'a permis d'acquérir beaucoup d'expériences, tout d'abord cela m'a permis de me familiariser avec des outils comme GitHub où les tests unitaires ; ou encore à me faire découvrir de nouveaux langages comme le XAML.

J'ai globalement beaucoup apprécié travailler sur ce projet. Tout d'abord, il y a la satisfaction de voir le projet avancer et grandir, mais aussi j'ai eu la chance d'avoir d'excellents partenaires de projets qui ont pu m'aider lorsque je m'étais retrouvé dans des impasses.

8 Défauts

Lors de ces derniers jours de code intensif, nous nous sommes rendu compte que plusieurs choix que l'on avait fait au cours du développement de l'application n'étaient pas forcément les meilleurs et nous ont amenés plus de problèmes qu'autre chose. C'est notamment le cas du choix d'effectuer la communication entre le **Tracker** et les clients via des requêtes *HTTP* effectuées par les clients.

Malheureusement, bien que ce choix peut paraître assez insignifiant, cela signifie que le **Tracker** ne peut pas communiquer de lui même avec les clients. En effet, nous avons donc une communication quasi-unidirectionnelle. La seule manière pour le **Tracker** d'envoyer une information à un client et d'attendre que le client fasse la requête pour cette information.

Cela nous a posé quelques soucis lors du développement global de l'application et encore plus lors du développement du partage de fichiers entre les pairs. Ainsi, avec notre solution actuelle, chaque pair est un serveur TCP qui écoute les connections entrantes sur le port 5000, ce qui signifie bien évidemment que ce port doit être ouvert pour que ORA fonctionne.

Si on avait implémenté le **Tracker** différemment, on aurait pu envoyer des informations de manière bidirectionnelle entre le **Tracker** et les clients, ce qui aurait pu éviter ce soucis de serveur TCP ouvert constamment sur un port statique. En effet, une solution aurait été d'utiliser le protocole UDP ainsi qu'une technique de hole-punching afin d'effectuer la connexion entre deux **Nodes**.

Enfin, bien que ce soit le soucis le plus contraignant que nous avons rencontré, beaucoup d'autres sont apparus lors de la conception de l'application, mais nous sommes tout de même entièrement satisfaits de ce que nous avons produit et particulièrement de la méthode d'authentification qui marche parfaitement et qui nous a demandé beaucoup de réflexion.

9 Conclusion

Au final, nous avons réussi la grande majorité de nos objectifs et sommes plutôt fiers de ce que nous avons fait. Si tout nos objectifs, c'est parce que tout d'abord, certaines approches que nous avons prises marchent, mais ne sont pas optimales, comme c'est le cas pour le Tracker, que nous aurions pu implémenter afin d'envoyer les informations de manière bidirectionnelle. De plus, le temps nous a manqué pour l'implémentation de certaines fonctionnalités, tel que pour le GUI, où bien que la majorité des méthodes implémentées dans le GUI soient présentes, il en manque quelques unes.

Cependant, nous savions depuis le début que tout ne serait pas parfait, mais malgré ça nous sommes content de ce que nous avons réalisé. Ce fut un projet de groupe très instructif, qui nous a permis de nous habituer à travailler et à nous coordonner sur un projet de programmation à plusieurs, de nous apprendre à utiliser de nouveaux langages comme XAML, et plus généralement de nous améliorer en programmation.

10 Annexes

10.1 Vocabulaire

— les utilisateurs seront appelés des **Members** au sein d'un **Cluster**
— les machines utilisées par ces derniers seront appelées des **Nodes**
— les groupes de **Nodes** partageant des fichiers seront appelés des **Clusters**