Projet ORA

Crab Wave

EPITA

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

Réalisé le 17 Janvier 2020

Table des matières

1 Introduction

Ce cahier des charges à pour but de présenter le projet ORA. Vous trouverez dans les parties suivant l'introduction une description du fonctionnement, nos objectifs et nos attentes vis à vis de ce projet, ainsi que les outils et les méthodes qui seront utilisés afin de réaliser ce projet.

Avant de commencer la présentation du projet, nous allons commencer par expliquer les origines du projet et de ce groupe.
Ce projet vient de Léo qui cherchait des idées de projet pour le projet de S2. Après plusieurs semaines à trouver des idées non viables (c'est-à-dire une idée de projet étant trop longue ou trop courte à réaliser) pour le projet de S2, il se disait que le *peer-to-peer* était une technologie intéressante et qu'il y avait quelques choses à faire avec. C'est là qu'il eu l'idée du projet ORA : une application de stockage de fichiers en *peer-to-peer*.
Il se mit donc à chercher des membres afin de réaliser ce projet. Cependant beaucoup de personnes préféraient faire un jeu ou désertaient le lendemain après avoir confirmé qu'ils voulaient faire partie du projet. C'est là que Léo fit appelle à l'agent L : l'homme masqué, le mangeur de pizza ; qui le mit en contact avec Raffaël. Après lui avoir parlé du projet, Raffaël accepta mais il manquait encore deux membres afin de former un groupe complet.
Léo proposa à Adam de rejoindre le groupe et Adam répondit : *« Cheh pas »*. Léo lui laissa un peu de temps pour réfléchir mais comme la date de rendu des groupes approchait à grand pas, c'est là que l'agent L : le héros au grand coeur, le sauveur de la nation ; intervint, il réussit à convaincre Adam.
Enfin, il restait encore un membre à trouver ainsi qu'un nom de groupe et un nom de projet. Pour le dernier membre, c'est le membre qui est venu à nous. Pierre-Corentin a envoyé un message à Adam pour savoir s'il avait un groupe et étant donné qu'il restait une place de libre c'est comme ça que Pierre-Corentin rejoignit le groupe.
Le nom de groupe nous est venu en premier, en cherchant des mots qui sonnaient bien ensemble et après un vote on a choisi Crab Wave. De même pour le nom du projet, nous fîmes plusieurs propositions et c'est ORA, trois lettres prisent au hasard, qui fut choisi.

Pour résumer, le projet ORA consiste en une application de partage de fichier en *peer-to-peer* tout en s'inspirant de logiciels de versionnage tels que *git*, de services de cloud tels que *Dropbox* ainsi que du protocole *BitTorrent* qui permet de partager des fichiers sur un réseau *peer-to-peer*.
Les avantages d'une telle solution par rapport à une solution de stockage classique sont que les données ne sont pas stockées chez un tiers. Cependant elles sont dupliquées chez les divers pairs afin d'assurer la stabilité et la reliabilité du stockage. La duplication des données n'étant pas un réel problème puisque le coût du stockage baisse de plus en plus alors que les capacités de stockage augmentent.
Un des autres problèmes lié à une telle méthode de stockage est qu'il faut qu'au moins un pair possédant le fichier désiré soit connecté au réseau afin de le récupérer. Ainsi, ORA apporte une solution à ce problème puisque l'application est hébergeable sur un serveur personnel, créant un noeud avec une disponibilité assez élevée.

2 Présentation du projet

Dans cette partie, le groupe se présentera dans un premier temps, puis nous expliquerons le projet, nous feront un état de l'art et enfin nous parlerons du protocole de transfert de données ainsi que des méthodes de compression et de chiffrement qui seront utilisées.

2.1 Présentation du groupe

— Léo Benito : « Bonjour, je m'appelle Léo, je suis passionné par l'informatique, ainsi que les maths depuis de nombreuses années et je programme depuis 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. »

— 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. »

— 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. »

— 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. »

2.2 Explication du projet

Ce projet est une application de stockage en *peer-to-peer* centrée autour de la notion de partage de fichiers au sein d'un même groupe, appelé *cluster*. Les utilisateurs seront appelés *node*(noeud) ou *peer* (pair). Cette application, devra être compatible avec Windows en priorité comme spécifie dans le dossier distribué ainsi que Linux.

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.

2.3 État de l'art

La technologie de *peer-to-peer* (P2P) est assez récente. En effet, elle vit le jour en 1969, date du lancement du projet *ARPAnet* lancé trois ans auparavant. 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. En théorie, la communication entre utilisateurs finaux ne dépend d'aucun élément central et d'aucune liaison « point-à-point » (liaison entre 2 hôtes inhibant le contrôle avancé du flux). Il est donc par définition possible d'échanger des informations avec n'importe quel ordinateur relié au réseau. A ce jour, cette technologie a évolué et des acteurs du partage de fichiers en *peer-to-peer* ont émergés.

BitTorrent est un protocole créé en avril 2001, son principe est simple : il trouve les utilisateurs qui possèdent les fichiers que d'autres utilisateurs recherchent, puis télécharge simultanément les fichiers de ces utilisateurs. 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.

Soulseek est un logiciel concentré sur le transfert de fichiers musicaux. il utilise 2 serveurs centraux uniquement dédiés à la recherche et au salles de tchat, qui est la fonctionnalité qui démarque Soulseek des autres. De plus, ses clients contiennent une fonctionnalité de bannissement pour lutter contre les freeriders (utilisateurs qui téléchargent des données sans en donner) ou pour rendre le tchat plus accueillant.

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.

2.4 Protocole

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.

2.5 Compression

La compression de données se base sur la théorie de l'information, établie par Claude Shannon, et se base sur deux concepts établis dans cette théorie :

— L'entropie, qui est la quantité d'information minimum nécessaire pour transmettre un message
— La redondance correspond à l'espace superflu utilisé pour transmettre des données.

Pour donner un exemple, prenons la chaîne de caractères « aaaaaaaaaaaaaaa ». Son entropie est nulle, car la probabilité qu'un caractère ne soit pas 'a' est nulle, et la redondance est très forte. Un moyen de compresser ce message serait « 16a », car le caractère 'a' est répété 16 fois. Si l'on se limitait seulement à des paquets de données de petite taille, il serait inutile de mettre en place un système de compression de données. Cependant, au vu de la taille potentielle de certains paquets, il devient nécessaire, afin de gagner en efficacité, d'utiliser un algorithme de compression. Nous utiliserons donc dans ce projet l'algorithme de compression *Zstandard* (abrégé *zstd*) établi par *Facebook* afin de compresser et décompresser les données transmises car après comparaison avec d'autres algorithmes de compression de données, il est un des plus efficace (ratio vitesse/taux de compression) et il possède beaucoup d'implémentations dans plusieurs langages de programmation et notamment le C#.

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

TABLE 1 : Comparaison de plusieurs algorithmes de compression

Source : https://facebook.github.io/zstd/

2.6 Chiffrement

Le Chiffrement de données vise à rendre un message impossible à comprendre à toute personne ne disposant pas d'une clé lui permettant de lire ce message. On distingue deux techniques de chiffrement majeures :

- Le chiffrement dit « symétrique » qui utilise une seule et unique clé afin de chiffrer et déchiffrer un message. Cette technique est utilisée depuis plusieurs siècles, un des algorithmes de chiffrement symétrique le plus connu étant le « Code de César ».
- Le chiffrement dit « asymétrique » qui repose sur une clé publique, connue de tous, servant à chiffrer le message, et sur une clé privée, connue seulement de la personne ayant créé les clés, servant à déchiffrer le message. Cette technique, créée dans les années 70, est très utilisée de nos jours pour sécuriser les connexions en ligne (le protocole *HTTPS* utilise par exemple le chiffrement RSA)

Au cours du projet, nous n'utiliserons uniquement des méthodes de chiffrement asymétriques afin de pouvoir sécuriser les transferts de paquets d'un pair à un autre.

3 Réalisation du projet

3.1 Découpage du projet

Le projet sera découpé en plusieurs sous-projets :

- **Tracker** contenant le programme hébergé sur un serveur distant qui permet d'aider la communication entre les différents pairs.
- **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 …).

Pour le **Tracker**, le **Core** ainsi que le **Daemon**, nous écrirons des tests unitaires, permettant l'application du *Test-Driven Development* qui sera expliqué plus tard dans ce document. Le fait de diviser le projet en plus petits projets ayant un but spécifique permet de séparer le développement de l'application du développement des fonctionnalités du noyaux. Cette division permet aussi aux autres développeurs d'utiliser l'**API**, même si les fonctionnalités ne sont pas encore implémentées ou permet à des développeurs externes au projet de développer leur propres applications.

3.2 Répartition des tâches

Voici le tableau de la réparition des tâches :

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

3.3 Planning

Voici le planning des tâches à effectuer :

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%

Objectifs pour les soutenances :

— Soutenance n°1 : Étant donné que les fonctionnalités du Core ne seront pas finies d'être développées, l'application ne fonctionnera pas totalement. Nous misons donc sur le développement d'une bonne partie du site web pour la première soutenance afin d'avoir du contenu à présenter autre que du code et des recherches.

— Soutenance n°2 : Nous prévoyons d'avoir développé une plus grosse partie du Core pour cette soutenance, le rendant opérable pour une majorité des fonctionnalités prévues, de même, les projets se basant sur le Core tels que le daemon ou le CLI devraient avoir plutôt bien avancé grâce à cela.

— Soutenance n°3 : Les objectifs pour cette soutenance sont d'avoir terminé toutes les fonctionnalités prévues afin d'avoir une application entièrement fonctionnelle.

3.4 Méthodologies 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.

3.5 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

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
- Netlify

4 Conclusion

Pour conclure, le projet ORA est une application de stockage de fichiers en *peer-to-peer*. Ce projet va nous permettre d'apprendre à mieux travailler en équipe ainsi que de s'améliorer en programmation. Le coût du projet n'est pas gratuit bien que la plupart des outils que nous allons utiliser sont open source, il y néanmoins le coût du nom de domaine *crabwave.com* à prendre en compte afin d'avoir une magnifique url pour accéder au site web, le coût d'une machine permettant d'héberger le **Tracker** principal du projet ainsi que le coût de l'impression de ce cahier et des rapports des soutenances à venir.