# Projet ORA

## Crab Wave

EPITA

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

## Réalisé le 6 mars 2020

## Table des matières

# 1 Introduction

## 1.1 Présentation de ORA

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. Le *Peer-to-Peer* est un modèle de réseau informatique où, contrairement au modèle client-serveur où il y a un seul serveur, chaque client est aussi un serveur. Cette technologie va donc nous permettre de partager nos fichiers sans passer par un tiers.

Nous avons défini un vocabulaire spécifique à notre application. Afin de bien comprendre la suite ce document, voici la définition des principales notions :

- les utilisateurs seront appelés des **Identities**
- les machines utilisées par ces derniers seront appelées des **Nodes**
- les groupes de Nodes partageant des fichiers seront appelés des **Clusters**

Cette application a pour but de fonctionner sous Windows comme spécifié dans le dossier distribué mais aussi compatible avec Linux car ce système d'exploitation nous tient particulièrement à coeur. Aussi la thématique de la sécurité est importante pour nous, l'application utilise le chiffrement RSA afin de sécuriser les données car des fichiers personnels peuvent être partagés. De plus, si vous n'avez tout de même pas confiance en le Tracker hébergé par notre équipe, vous pouvez tout à fait héberger votre propre instance d'un Tracker ORA et même en modifier le code ! En effet, ORA vise à être totalement open-source à terme, ce qui permet 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* dont le développement commencera après la première soutenance. Ces deux implémentations nous permettent de montrer que grâce à l'API ont peux gérer les fichiers, Clusters et Nodes de manière totalement différente.

## 1.2 Division du projet

Comme mentionné dans le cahier des charges nous avons divisé le projet en plusieurs sous-projets. Cette division permet notamment une séparation entre le Core et les applications CLI et GUI. Cette séparation permet notamment à d'autres développeurs de créer leurs propres applications basées sur l'API de ORA. Étant nous même développeurs, la possibilité de construire notre propre application sur une API de cette manière nous tient particulièrement à coeur.

Le découpage du projet est donc le suivant :

- **Tracker** le programme permettant de connecter 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 …).

## 1.3 Planning de réalisation

Voici le planning de réalisation qui a été présenté dans le cahier des charges :

| 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% |

Nous expliquerons dans la partie **Avancement** ce qui a été fait depuis le début du projet, et dans la partie **Prochaine Soutenance** ce qui sera à faire d'ici la semaine du 20 avril.

### 1.4 Répartition des tâches

Voici le tableau de la répartition des tâches qui a été présenté dans le cahier des charges :

| **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 |

## 2 Avancement

Dans cette partie, nous détaillerons ce qui a été réalisé depuis la validation du cahier des charges.

### 2.1 Tracker

Le Tracker étant le programme qui permet de mettre en connexion les différents Nodes, il nous fallait un serveur utilisant un protocole garantissant la réception des paquets. Ainsi, nous avons choisi de partir sur un protocole basé sur *TCP*. Les requêtes au Tracker étant légères et afin de garder une simplicité d'utilisation (pour les développeurs souhaitant créer une application basée sur le Tracker de ORA) nous avons choisi que le Tracker serait un serveur *HTTP* fonctionnant sous forme d'*API RESTful*. Le serveur est non bloquant c'est-à-dire qu'il peut traiter plusieurs requêtes en même temps sans avoir à attendre la fin du traitement de la requête précédente. En effet, le Tracker aurait été inutilisable si on devait attendre que le traitement de la requête du client précédent soit fini avant de pouvoir espérer recevoir une réponse à sa requête. Pour pouvoir renvoyer des informations sur les Clusters par exemple, il est nécessaire de stocker ces informations. Afin de les stocker, nous avons utilisé la base de donnée *LevelDB* développée par Google, qui est réputée pour sa vitesse et sa faible taille sur le disque. Aussi, l'architecture modulaire que nous avons choisi pour le Tracker nous permet de rajouter des routes et des modèles de données simplement tout en conservant un serveur performant. Les *endpoints* (point d'accès) actuellement implémentés sont :

- `GET /` — renvoie un message de bienvenue.
- `GET /clusters/unIdentifiant` — renvoie les informations associées aux Cluster ayant pour identifiant `unIdentifiant`.
- `POST /clusters?name=unNom` — crée un Cluster dont le nom est `unNom` et renvoi son identifiant.
- `DELETE /clusters/unIdentifiant` — supprime le Cluster associé à l'identifiant `unIdentifiant`.

Sachant que toute requête à un *endpoint* non spécifié ici renverra une réponse dont le code HTTP sera 4XX. Cependant il y a un problème auquel nous n'avions pas pensé lors de la rédaction du cahier des charges : il faut authentifier les routes du Tracker. En effet, nous ne voulons pas que n'importe qui puisse supprimer n'importe quel Cluster. Nous souhaitons que seul le propriétaire du Cluster puisse effectuer cette action. Il faut donc authentifier le Node qui fait la requête au Tracker afin de déterminer si c'est bel et bien le propriétaire.

Enfin, le Tracker est actuellement disponible à l'adresse suivante : https://tracker.ora.crabwave.com.

```
{
    "id": "91c5386b-25c1-4e11-b0c1-0f1fea9a2cbf",
    "name": "Testee",
    "owner": "ef255ef7-dd02-44a3-ad34-70129fdd79a8",
    "members": [],
    "admins": [],
    "files": []
}
```

FIGURE 1 – Exemple de réponse du Tracker à une requête GET /clusters/id

## 2.2 Daemon

Le projet du Daemon à été créé et certaines librairies nécessaires à son fonctionnement ont été importées mais aucun code réel n'a été produit. Le programme n'est donc pour l'instant pas du tout fonctionnel bien que des recherches aient été effectuées.

## 2.3 API (*Application Programming Interface*)

De manière à correctement structurer le projet, nous avons décidé de couper celui-ci en 2 parties distinctes, l'API qui définit l'architecture globale du projet, les méthodes ainsi que les structures qu'elles utilisent (comme les Cluster, les Nodes, etc…), et le Core qui implémente ces méthodes et structures de données. De plus, l'API étant totalement indépendante d'une quelconque implémentation, cela permet à n'importe qui de venir étendre les fonctionnalités d'ORA en implémentant les fonctionnalités voulues mais aussi à n'importe quel développeur de ne pas se soucier du support de différentes plateformes et de se focaliser uniquement sur l'utilisation de cette interface.
Nous avons donc créer plusieurs structures à ce jour :

- *ICipher*, représentant un objet de chiffrement ;
- *ILogger*, permettant de gérer les traces générées par ORA (que ce soit pour du debuggage ou bien des erreurs) ;
- *HttpClient*, représentant un client HTTP basique capable de faire des requêtes GET/POST/DELETE/PUT ;
- *IClusterManager*, permettant de gérer la création/suppression de Clusters ;
- *INodeManager*, permettant de gérer la création/suppression de Nodes ;
- *Les autres structures sont présentes car nécessaire à celles définies plus haut*.

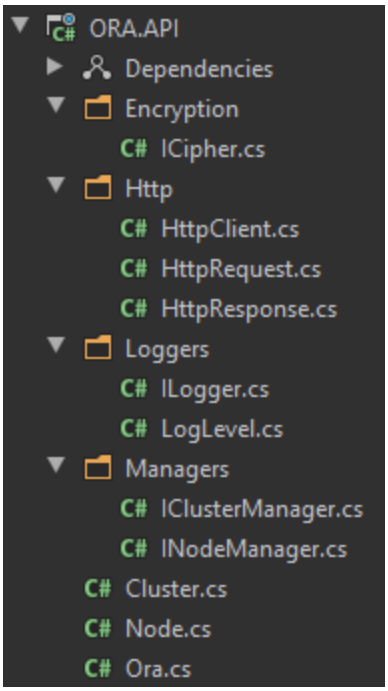


FIGURE 2 – Architecture du projet *API*

## 2.4 Core

### 2.4.1 Client HTTP

Afin de communiquer avec le Tracker, nous avons eu besoin d'un client HTTP fonctionnel implémentant la classe disponible dans l'API. Nous en avons donc implémenté un grâce à la librairie Unirest qui permet de créer des requêtes HTTP assez simplement en quelques lignes seulement.

### 2.4.2 Chiffrement

Afin de garantir la sécurité des échanges entre utilisateurs, nous avons créé une classe permettant de chiffrer des données en utilisant l'algorithme *RSA*.
.NET ayant déjà un espace de nom permettant de chiffrer des données, nommé *System.Security.Cryptography*, nous nous en sommes servis. Nous avons utilisé la classe *RSACryptoServiceProvider* de cet espace de nom.
La classe crée permet à l'utilisateur de choisir la taille de sa clé et comporte 2 méthodes :

- *Encrypt*, qui chiffre les données ;
- *Decrypt*, qui déchiffre les données.

Nous avions dans un premier temps considéré l'utilisation de la classe *CspParameters* afin de stocker les clés dans un conteneur, mais cela aurait rendu l'application incompatible avec Linux.

```
namespace ORA.API.Encryption
{
    6 usages  1 inheritor  Adamaq01  3 exposing APIs
    public interface ICipher
    {
        1 usage  1 implementation  Adamaq01
        byte[] Encrypt(byte[] data);

        1 usage  1 implementation  Adamaq01
        byte[] Decrypt(byte[] data);
    }
}
```

FIGURE 3 – Interface du *Chiffrement*

### 2.4.3 Data Management

Pour pouvoir introduire la notion de *Data Management*, nous avons tout d'abord dû implémenter des Clusters pour réaliser celui-ci.
Ces derniers sont des groupes dans lesquels des identités (utilisateurs) possédant plusieurs *Nodes* (machines) pourront échanger leurs fichiers ou leurs dossiers entre eux grâce à la technologie *Peer-to-Peer*.

Ainsi nous avons codé trois méthodes sur une interface afin de manipuler ces Clusters :
— *CreateCluster* qui permet d'en ajouter un ;
— *GetCluster* qui permet d'en avoir un déjà existant ;
— *DeleteCluster* qui permet d'en supprimer un si cela est possible.

```
namespace ORA.API.Managers
{
    5 usages  1 inheritor  Adamaq01  3 exposing APIs
    public interface IClusterManager
    {
        3 usages  1 implementation  Adamaq01
        Cluster CreateCluster(string name);

        1 usage  1 implementation  Adamaq01
        Cluster GetCluster(string identifier);

        1 usage  1 implementation  Adamaq01
        bool DeleteCluster(string identifier);
    }
}
```

FIGURE 4 – Interface du Cluster Management

## 2.5 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. Cependant, notre site web contiendra aussi la documentation de l'API et du Tracker afin d'aider les développeurs à réaliser leurs applications basées sur ORA.

Afin de réaliser le frontend de ce site nous avons utilisé la bibliothèque React.js pour créer un site web dynamique en Javascript, ainsi que la bibliothèque Sass qui permet, entre autres, d'écrire des feuilles de style de manière plus simple et élégante. Nous n'avons pas développé de backend pour ce site étant donné que nous n'en voyons pas l'utilité. Pour l'hébergement du site web, nous avons utilisé now.sh de ZEIT à la place de Netlify (mentionné dans le cahier des charges) qui est totalement gratuit et qui permet à chaque membre du groupe d'avoir des accès d'administration sur le site. Nous avons aussi acheté le nom de domaine crabwave.com. Ainsi, le site web est disponible à l'adresse suivante : https://ora.crabwave.com.

Le site web de ORA est découpé en deux grosses parties :
— La page principale, qui contient la présentation du projet, la liste des membres du groupe ainsi que les liens de téléchargement du projet ;
— La page de documentation, contenant les présentations de l'API et du Tracker, qui va aider les développeurs à créer leurs propres applications basées sur ORA.

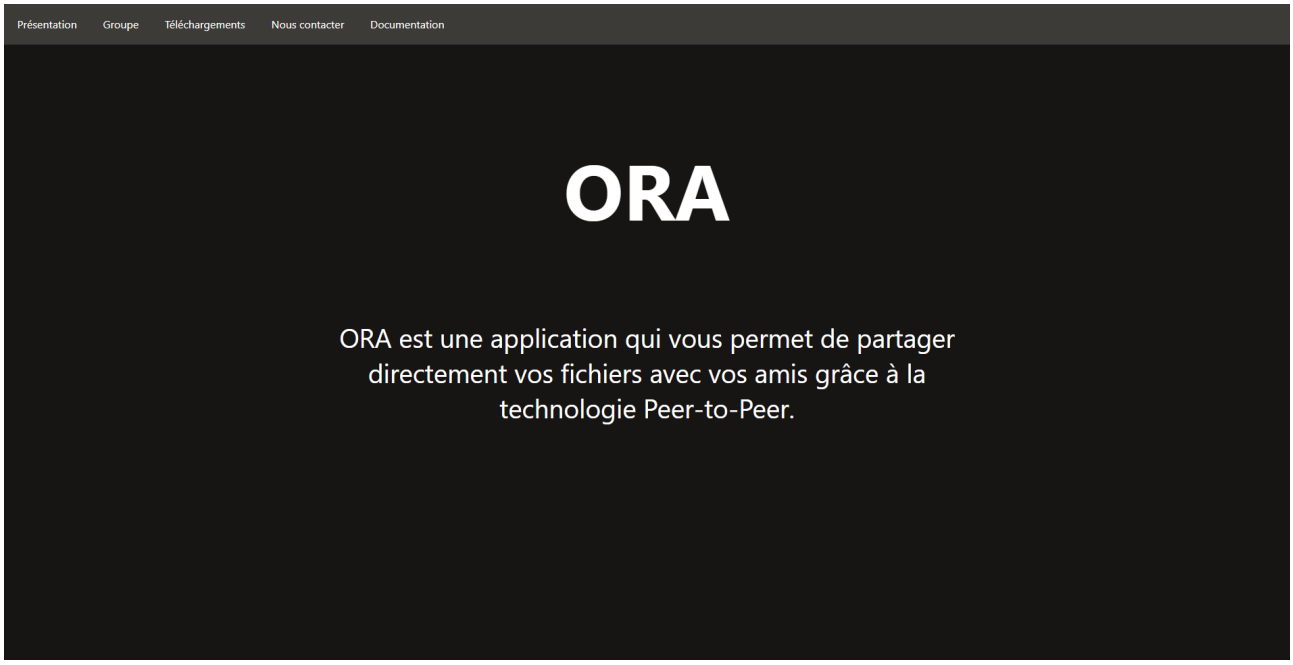


FIGURE 5 – Accueil de la page principale


FIGURE 6 – Présentation du projet

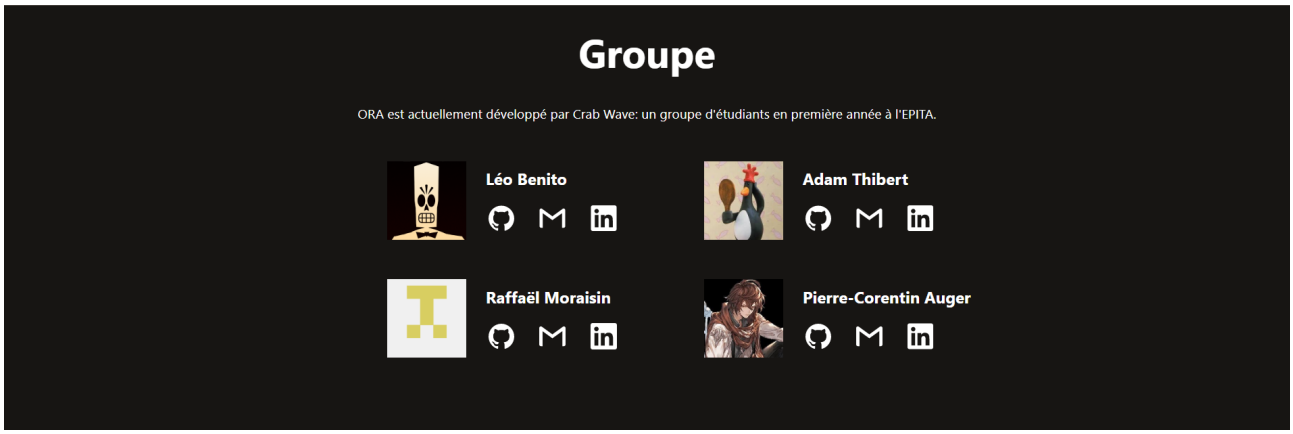


FIGURE 7 – Présentation des membres du groupe

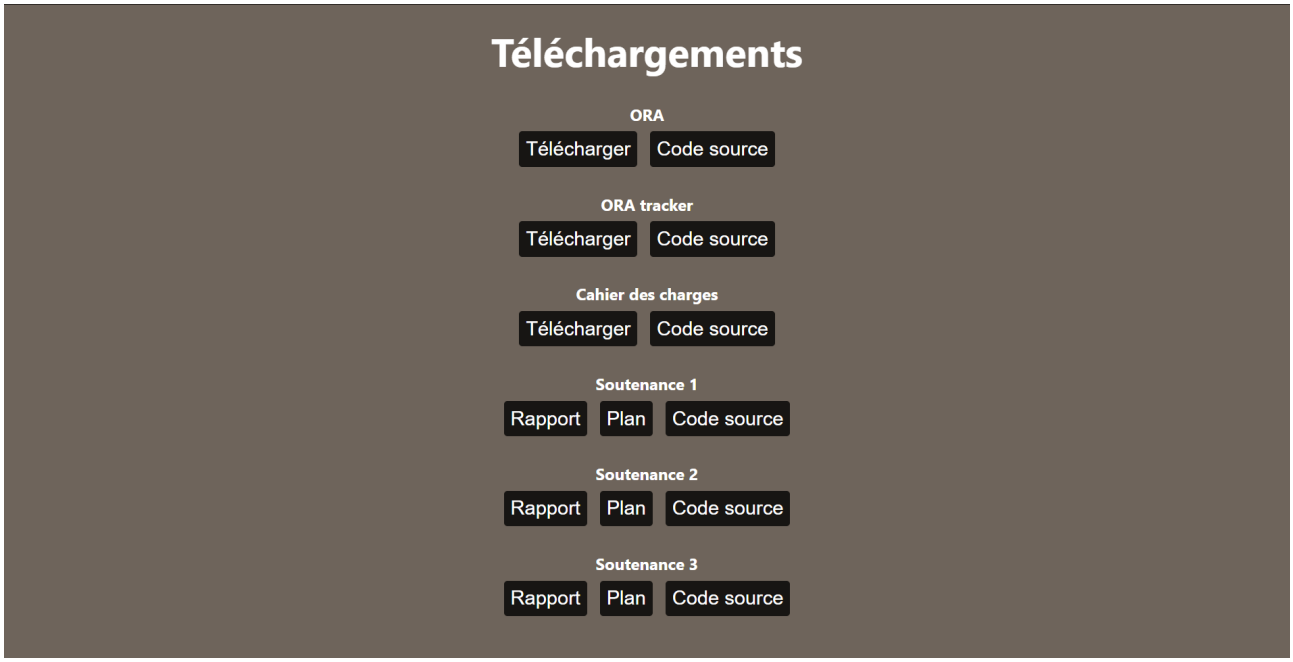


FIGURE 8 – Lien de téléchargement des fichiers du projet

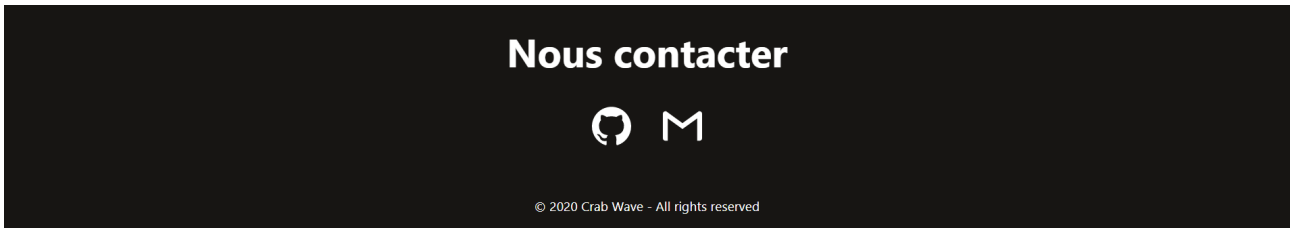


FIGURE 9 – Moyens de contacter le groupe Crab Wave

## 2.6 CLI

CLI signifie application en lignes de commande ou *Command Line Interface* en anglais. Actuellement elle nous permet de gérer nos Clusters, c'est-à-dire de créer, supprimer ou obtenir les informations d'un Cluster. De plus, nous pouvons changer l'URL du Tracker car comme nous l'avons dit plus haut, nous souhaitons que le Tracker puisse être *self-hosted*, ce qui signifie que l'URL du Tracker doit pouvoir être changée.

## 2.7 GUI

Puisque nous n'avions aucunement prévu d'avancer sur le GUI pour la première soutenance, le projet n'a pas été créé mais des recherches ont déjà été effectuées afin de savoir quelles librairies seraient utilisées pour gérer les graphismes de l'application. Le choix se porte donc pour l'instant sur la librairie *AvaloniaUI* bien que ce soit encore sujet à changements puisque nous n'avons pas encore expérimenté avec.

## 2.8 Intégration Continue

Nous avons mis en place une pipeline d'intégration continue avec *Travis CI*. Comme expliqué dans le cahier des charges, la pipeline effectuera tous les tests suivants à chaque modification :

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.

Cette pratique nous a permis de garantir que chaque modification n'introduit aucune régression dans le code, c'est-à-dire qu'elle n'introduit pas défaut. Les principaux avantages de l'intégration continue sont que la détection d'erreurs est plus rapide, ce qui permet dont de les localiser plus facilement.

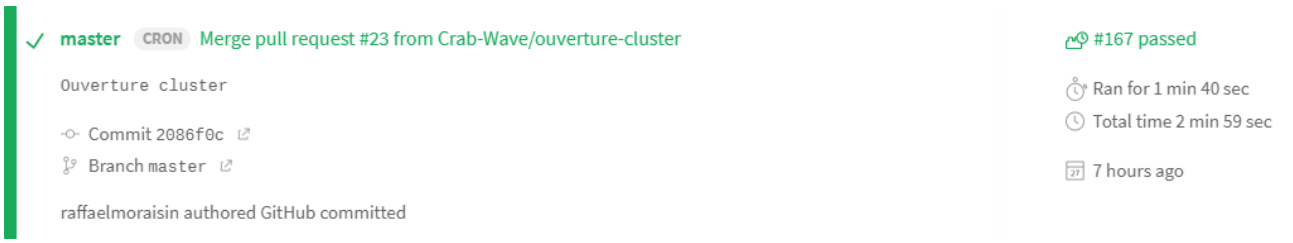


FIGURE 10 – Rapport des travaux de l'intégration continue sur le dernier changement

# 3 Méthodologies utilisées

## 3.1 Avantages

Un des avantages de l'utilisation de l'intégration continue avec *Travis CI* ainsi que l'utilisation des *Pull Requests* et *Issues* de *GitHub* est que la branche principale du dépôt Git reste propre et fonctionnelle quoi qu'il arrive. Cela nous permet de toujours proposer un programme sans bugs majeurs (théoriquement). De plus, grâce à l'intégration continue et aux *tests unitaires* produits avec le *Test Driven Development*, nous pouvons développer les fonctionnalités d'ORA tout en évitant le fameux problème du "ça fonctionne sur mon PC", puisque des tests sont effectués à chaque nouveau commit sur différentes plateformes dans le *Cloud*.

## 3.2 Inconvénients

Bien qu'en théorie les méthodes citées plus haut nous permettent d'être mieux organisé dans notre travail, en réalité c'est un peu plus complexe que cela. En effet, il faut être très rigoureux afin de suivre toutes les "consignes", ce qui peut être un peu compliqué par moment, surtout lorsque nous ne sommes pas à temps plein sur le projet. Par exemple nous n'avons pas réussi à suivre complètement les objectifs fixés lors des réunions pour les Sprints. Le Sprint 3 à ainsi été sauté à cause des périodes de partiels et nous nous sommes attribués des tâches à la volée, sans réelle organisation. Un autre défaut, cette fois-ci dû au fait de travailler grâce aux *Issues* et *Pull Requests* de *Github*, est le fait que devoir faire réviser le code par une autre personne avant d'incorporer la fonctionnalité en question dans le projet peut parfois prendre du temps, ce qui peut ensuite mettre en retard toute une branche du projet. Aussi, le *Test Driven Development* s'est avéré plus difficile à mettre en place que prévu. Étant donné que nous sommes habitués à écrire du code tout de suite, nous ne pensons pas forcément à écrire les tests en premier. De plus, en essayant d'écrire les tests en premier, il est difficile d'imaginer les tests que l'on pourrait faire étant donné que notre connaissance de l'environnement .NET est pour l'instant limitée.

# 4 Prochaine Soutenance

## 4.1 Tracker

Pour la prochaine soutenance, nous devons mettre en place le système d'authentification des Nodes comme mentionné précédemment. De plus, nous devons implémenter d'autres routes comme la route /nodes qui permettra de recevoir les informations d'un Node. Enfin, nous tenterons d'optimiser encore le Tracker en essayant par exemple différentes alternatives à la base de donnée LevelDB comme RocksDB ou Hyper-LevelDB, afin de déterminer laquelle elle est la plus performante, et si ce n'est pas LevelDB, afin d'améliorer la vitesse de réponse du Tracker.

## 4.2 Daemon

Pour la prochaine soutenance, le Daemon aura été complété à 50%, c'est-à-dire qu'il pourra gérer une petite partie des fonctionnalités de ORA (comme créer un Cluster, etc…) ainsi que synchroniser les données avec le Tracker de manière automatique. Pour cela nous utiliserons de l'IPC (Inter-process communication) avec le CLI, qui passera donc par le Daemon pour effectuer des actions. Nous utiliserons la librairie publique https://github.com/jacqueskang/IpcServiceFramework pour celà, bien que ce soit sujet à changement.

## 4.3 API

D'ici la prochaine soutenance, l'API devra être complète à environ 80%, ce qui signifie que la plupart des services et structures de données devront avoir été créées. Cela inclut donc tout ce qui est en rapport avec les Identités et les fichiers.

## 4.4 Core

### 4.4.1 Data Management

Pour la prochaines soutenance, nous implémenterons l'Identity Management, le Node Management et le File Management.
Le premier permet de gérer l'organisation des utilisateurs pouvant avoir accès à un Cluster. En effet, chaque Cluster aura des administrateurs qui pourront enlever ou ajouter des Nodes de celui-ci et un propriétaire qui aura, en plus des droits d'administrateur, celui de pouvoir supprimer celui-ci.
Le second permet à l'utilisateur d'ajouter ou de supprimer une Node (machine de l'utilisateur). Elle permet aussi d'obtenir le nom de celle-ci.
Le dernier permet à l'utilisateur d'ajouter ou de supprimer un fichier ou un dossier sur un Cluster. Elle permet aussi d'obtenir le nom de celui-ci.

### 4.4.2 Compression

Pour la prochaine soutenance, nous implémenterons l'algorithme de compression Zstandard, établi par Facebook. Nous avions choisi cet algorithme car en plus d'être en licence libre, son ratio taux de compression/vitesse le situe parmi les algorithmes de compression les plus efficaces.

| 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 11 – Comparaison de plusieurs algorithmes de compression
Source : https://facebook.github.io/zstd/

## 4.5 CLI

Pour la prochaine soutenance, comme inscrit dans le cahier des charges, nous prévoyons que 80% du CLI soit développer. Cela implique que toutes les méthodes du Data Management créées entre la première et la deuxième soutenance seront implémentées dans celui-ci. Il permettra donc :

- de gérer les Clusters, que ce soit les fichiers qui lui sont attribués ou encore les Nodes qui les possèdent.
- de synchroniser les fichiers de tous les Clusters auxquels nous appartenons.
- et bien d'autre encore.

## 4.6 GUI

Pour la prochaine soutenance 30% du GUI doit être fait. Cela correspond à la création et la mise en place du projet et développement d'un début d'interface permettant de gérer les Clusters ainsi que les Nodes.

## 4.7 Site Web

Bien qu'une grande partie du site web aie été réalisée il manque actuellement du contenu. Il faudrait améliorer la partie présentation c'est-à-dire la reformuler et la développer, ainsi qu'écrire un script ou utiliser un outil qui va générer la page de documentation à partir des commentaires de documentations écrits dans notre code.

## 5 Conclusion

Pour notre première soutenance, nous avons rempli tous les objectifs énoncés dans notre planning de réalisation. Cependant il faut noter que l'évaluation d'un pourcentage d'avancement n'est pas très facile et pas très objective. Il faut donc prendre ces pourcentages dans un intervalle de $\pm 5\%$.
Ainsi nous avons estimer l'avancement des tâches suivantes :

- le Tracker est à 70% opérationnel ;
- le Core est réalisé à 45% ;
- l'API est liée au Core, mais certaines fonctionnalités de ce dernier ne sont pas nécessaires à être importés dans l'API comme le chiffrement ou la compression par exemple ; De ce fait l'API est complétée à 50% ;
- le Daemon est créé et initialisé aux conformes de notre projet sans méthodes pour l'instant. On arrive donc à 10% ;
- pour le CLI, nous avons un début d'interface avec quelques méthodes utilisant toutes les méthodes du Core pouvant servir directement l'utilisateur, en plus du changement d'url. Nous arrivons donc à 20% ;
- le GUI n'a pas encore été commencé ;
- le site web est opérationnel à 50% : on peut voir une présentation du projet et de toute l'équipe mais aussi tous les liens utiles pour les développeurs ayant envie de reprendre une partie de notre code.
- pour ce qui est de LaTeX, nous avons créé notre propre

Nous travaillerons avec le même rythme et la même intensité en vue de la deuxième soutenance pour pouvoir remplir les objectifs mentionnés plus haut.

| 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% |

# 6 Annexe

## 6.1 Vocabulaire

### 6.1.1 API RESTful

Une API compatible REST, ou « RESTful », est une interface de programmation d'application qui fait appel à des requêtes HTTP pour obtenir (GET), placer (PUT), publier (POST) et supprimer (DELETE) des données.

### 6.1.2 Endpoint

Un endpoint (littéralement point d'accès) est un point à partir duquel une API se connecte au logiciel.

### 6.1.3 Issues

Sur GitHub, les Issues permettent d'indiquer quelles sont les tâches à réaliser dans un projet, que ce soit des nouvelles fonctionnalités à implémenter ou bien des bogues à régler.

### 6.1.4 Pull Request

Une pull-request désigne tout simplement l'action qui consiste à demander au détenteur du dépôt original de prendre en compte les modifications que vous avez apportées sur votre fork et que vous souhaitez partager

### 6.1.5 TDD

Le Test-Driven Development (TDD), ou Développements Pilotés par les Tests en français, est une méthode de développement de logiciel qui consiste à écrire chaque test avant d'écrire le code source d'un logiciel, de façon itérative.

### 6.1.6 Travis CI

Travis CI est une plateforme d'intégration continue, elle nous permet de lancer automatiquement des travaux sur chaque *commit*. On rappelle que nous avons réglé notre répertoire Git de sorte qu'il soit impossible de fusionner une branche avec la branche master si les travaux d'intégration continue ne passent pas sur cette branche.