Projet ORA

Crab Wave

EPITA

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

Réalisé le 23 avril 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 qui comporte 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 **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**

Cette application a pour but de fonctionner sous Windows comme spécifié dans le dossier distribué mais d'être 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 communications et 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*. Ces deux implémentations nous permettent de montrer que grâce à l'API, on peux gérer les fichiers, **Clusters** et **Nodes** de manière totalement différente.

1.2 Division du projet

Comme mentionné dans les documents précédemment remis, nous avons divisé le projet en plusieurs sous-projets. Cette division permet 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 nos propres applications 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 soutenance du 10 mars.

2.1 Authentification et autorisation

Lors de la précédente soutenance nous avons parlé du fait que nous ne souhaitions pas que n'importe qui puisse supprimer n'importe quel **Cluster** par exemple, d'où la nécessité de développer un système d'authentification. Tout d'abord il faut saisir la différence entre l'authentification et l'autorisation. L'authentification est un processus qui permet de s'assurer de l'identité de la personne et l'autorisation est un processus qui vérifie si la personne est autorisée à effectuer une certaine action. Ainsi, 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) 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.

2.2 Tracker

Lors de la rédaction du cahier des charges nous n'avions pas pensé à l'authentification et à l'autorisation mais c'est une grosse partie, importante du projet que nous avons développé depuis la dernière soutenance, son fonctionnement y est décrit dans la partie réservée à cet effet. Nous avons aussi réusiné certaines parties du code avec le recul et l'avancement du projet avec quelques changements architecturaux dû à la création de nouvelles abstractions. Nous avons implémenté la route members qui permet de voir, d'ajouter et de supprimer des membres d'un **Cluster**. Il faut bien entendu avoir l'autorisation pour effectuer certaines actions : cela revient à être propriétaire ou administrateur du **Cluster**. Dans le planning de réalisation nous avions marqué que le **Tracker** devait être réalisé à 90% mais nous n'avions pas pris en compte l'authentification donc nous n'atteignons actuellement pas 90%.

2.3 Daemon

Le **Daemon** a été implémenté en quasi totalité depuis la dernière soutenance où uniquement le projet C# avait été créé, avec les dépendances etc…Mais d'autres choses viennent avec le **Daemon**, que ce soit au niveau du **Core** ou des autres applications.
En effet, il y a désormais une nouvelle implémentation de l'**API** dans le projet, qui gère l'**IPC**. Ainsi, le **Daemon** va pouvoir exposer toutes les fonctionnalités du **Core** à travers des tunnels de communication afin que les applications telles que le **CLI** ou **GUI** puissent les utiliser avec la nouvelle implémentation de l'**API** qui est capable d'intéragir avec ces tunnels.
Ainsi lorsque une application veut appeler Ora pour une quelconque raison (que ce soit pour avoir des informations à propos d'un **Cluster**, ou pour modifier un fichier), elle fera désormais appel au **Daemon**, et cela sans que le développeur ne s'en rende compte, puisque le projet continue d'utiliser l'**API** ORA (mais à "distance"). Cela va donc nous permettre d'avoir le **Daemon** constamment lancé sur le PC de l'utilisateur pour gérer les fichiers, etc…tandis qu'il pourra toujours intéragir avec les fonctionnalités de ORA de manière ponctuelle avec les applications qui utilisent le **Daemon** pour communiquer.

2.4 API (*Application Programming Interface*)

Nous avons implémenté une nouvelle classe dans l'**API** qui est la classe **Member**. Un **Member** est tout simplement un utilisateur de notre application qui est représenté par deux chaînes de caractère : son nome et un identifiant généré par l'intermédiaire d*Identity* qui lui est propre.
Cette dernière est une autre classe que nous avons implémentée qui possède comme attributs deux clés RSA une publique et une privée, générées à partir de la classe RSACipher.
Une interface a été implémentée pour la gestion de **Member** au sein d'un **Cluster**. Pour rappel, un **Cluster** est un groupe de **Nodes** dans lequel ces derniers peuvent échanger et partager leurs fichiers et dossiers. Elle est composée de trois méthodes qui vous seront introduites dans la sous-section *Data Management* de la section **Core**.
Une interface pour la gestion des **Identities** a aussi été rajoutée avec deux méthodes qui seront introduites comme dit un peu plus haut.
Nous avons aussi développé une dernière classe qui est celle des **Nodes**. Elle correspond aux machines que va utiliser le **Member** pour utiliser notre application. Elle a deux attributs qui sont deux chaînes de caractères : un nom et un identifiant.

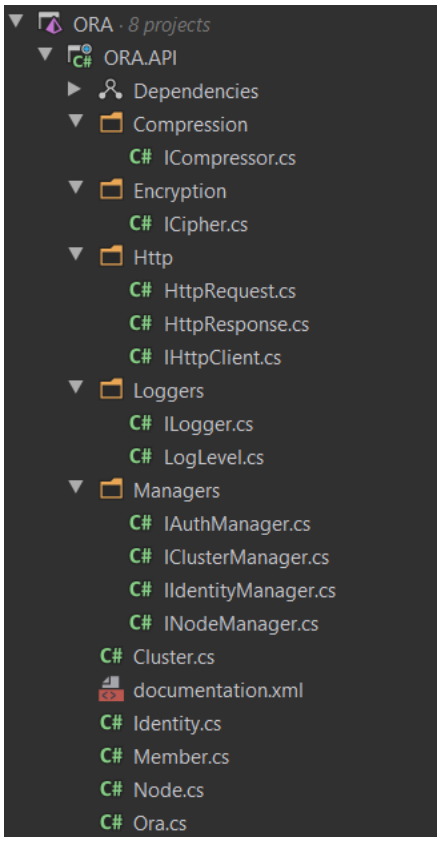

FIGURE 1 – Architecture du projet *API*

2.5 Core

2.5.1 Chiffrement

Nous avons modifié la classe RsaCipher en implémentant un constructeur permettant de créer un nouvel objet RsaCipher à l'aide de clé privée et publique préexistantes, ainsi que deux méthodes permettant à l'utilisateur de récupérer la clé privée et la clé publique.

2.5.2 Data Management

Pour cette soutenance, nous avons introduit deux éléments importants dans le data management : la gestion de **Members** dans un **Cluster** et la gestion d'**Identities**.
Pour la première, trois méthodes ont été implémentées sur une interface :

- *GetMembers* qui retourne la liste de tous les **Members** du **Cluster**;
- *GetMember* qui retourne le **Member** présent dans le **Cluster** correspondant à son identifiant;
- *RemoveMember* qui supprime du **Cluser** le **Member** correspondant à l'identifiant passé en paramètre et retourne un booléen qui est True si la suppression à été succès et False dans le cas contraire.

```
1 usage  1 override  Raffaël MORAISIN
public abstract List<Member> GetMembers();

1 override  Raffaël MORAISIN
public abstract Member GetMember(string identifier);

1 usage  1 override  Raffaël MORAISIN
public abstract bool RemoveMember(string identifier);
```

FIGURE 2 – Interface du Member Management au sein d'un Cluster

Pour la seconde, deux méthodes ont été implémentés sur une interface :
— *GetIdentity* qui retourne l'**Identity** du **Member**;
— *GenerateIdentity* qui génère une **Identity** et la sauvegarde à l'endroit spécifié.

```
namespace ORA.API.Managers
{
    5 usages  1 inheritor  Adamaq01 +1  3 exposing APIs
    public interface IIdentityManager
    {

        2 usages  1 implementation  Adamaq01
        Identity GetIdentity();
        1 usage  1 implementation  Raffaël MORAISIN
        Identity GenerateIdentity(string name);
    }
}
```

FIGURE 3 – Interface de l'Identity Management

2.5.3 Compression

Nous avions tout d'abord prévu d'utiliser la bibliothèque Zstandard, établie par Facebook, car elle est en licence libre et l'algorithme qu'elle implémente offre un excellent ratio taux de compression/vitesse de compression.
Cependant, lors de la réalisation du projet, nous nous sommes rendus compte que la bibliothèque n'était malheureusement pas compatible avec le système d'exploitation Linux, sur lequel nous souhaitons que ORA soit compatible. Nous avons donc décidé de passer à la bibliothèque Zlib, implémentant l'algorithme de compression deflate. Cet algorithme est lui aussi en licence libre et offre un taux de compression semblable à celui proposé par Zstandard, mais la compression et la décompression sont plus lentes.

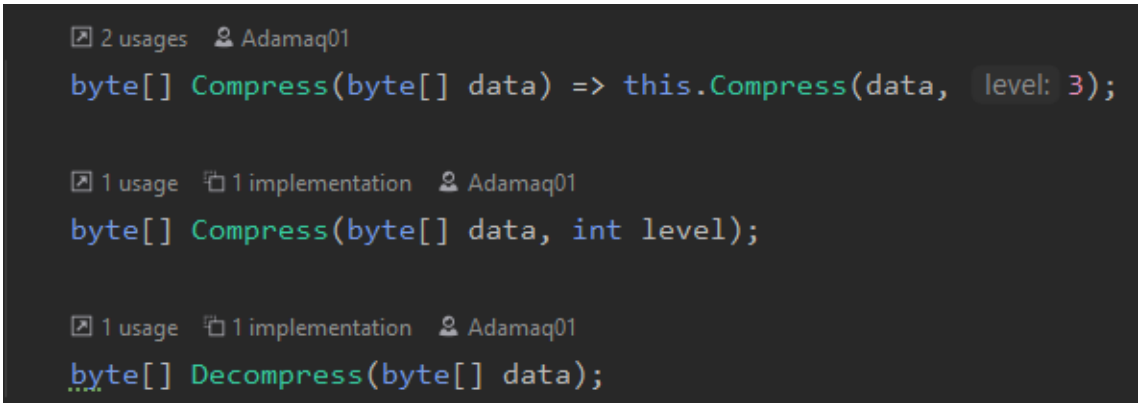

FIGURE 4 – Interface de la compression

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

2.6 CLI

Par rapport à la précédente soutenance, le **CLI**, l'application en ligne de commandes, a également évolué. Désormais, il peut donner la clé publique de l'**Identity** du **Member** courant et générer une nouvelle **Identity**.
De plus, on peut maintenant obtenir la liste des **Members** d'un **Cluster** à partir de l'identifiant de ce dernier.
Enfin, il peut enlever un **Member** d'un **Cluster** à partir de l'identifiant du **Member** voulu et du **Cluster** concerné.

2.7 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 multiplateforme 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.

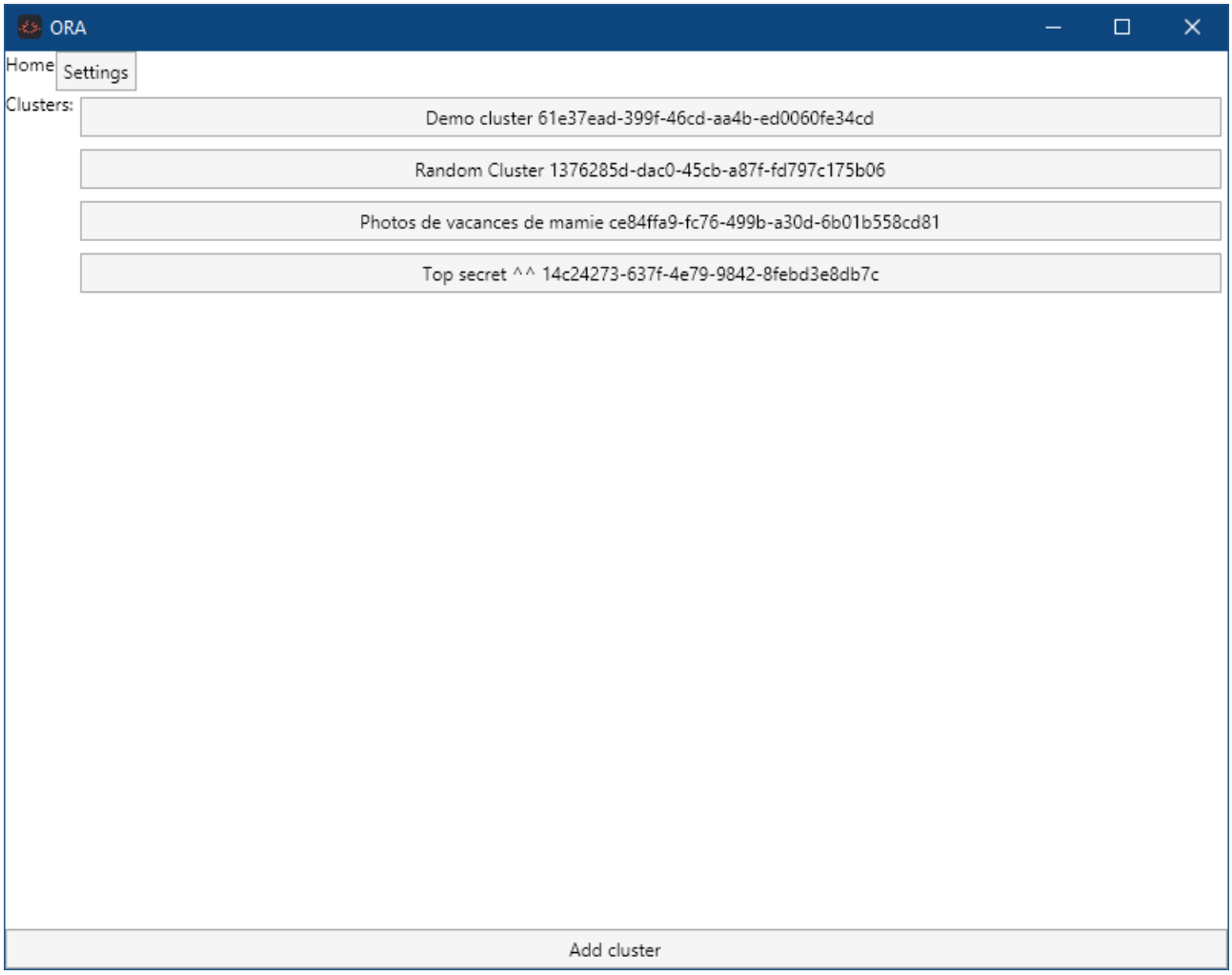

FIGURE 6 – Vue de la page d'accueil avec les Clusters dont l'utilisateur fait partie

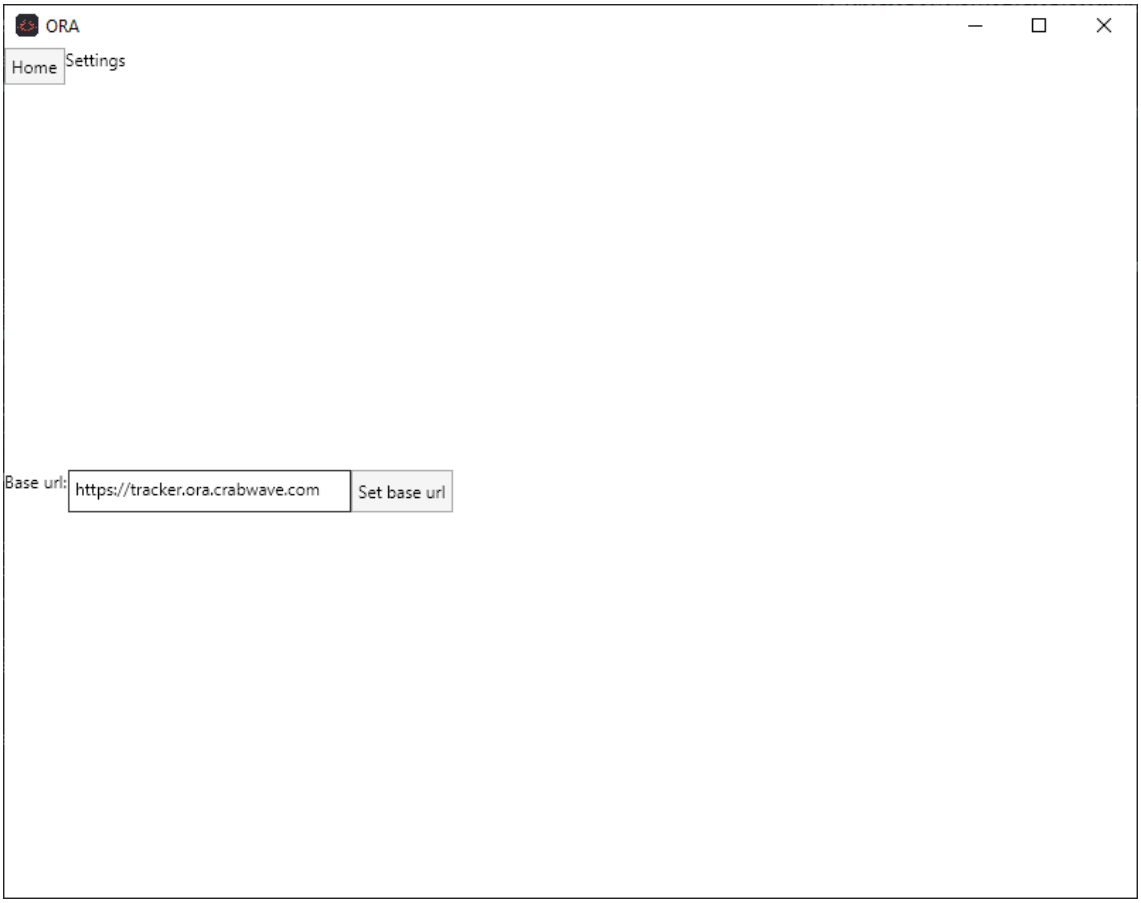

FIGURE 7 – Vue des paramètres de l'application

2.8 Site Web

Comme annoncé à la précédente soutenance nous avions prévu de travailler sur la partie documentation du site web. Cette partie est très importante pour nous puisque c'est ici que les développeurs qui souhaitent créer leur propre application 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.
Cette documentation est générée directement à partir de la documentation du code. Notre code est documenté avec des commentaires XML, la commande dotnet doc permet ensuite de générer un fichier au format XML contenant la documentation de tout le projet.

```
<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 – Un exemple de documentation au format XML

Nous avons développé un script est lancé à la construction du site web et qui permet de récupérer le fichier XML généré à partir des commentaires de documentation du code puis de le transformer au format JSON. Le format JSON (JavaScript Object Notation) est un format de donné qui représente des objets JavaScript il est donc plus facile à manipuler en Javascript que le XML. Notre site web étant dynamique les pages de documentation sont générées dynamiquement à partir de la documentation au format JSON.


```
"ORA.API.Encryption.ICipher": {
  "name": "ICipher",
  "namespace": "ORA.API.Encryption",
  "summary": "An object capable of Encrypting data or Decrypting data that has been previously encrypted with the same algorithm.",
  "example": {
    "text": "The following code\n\nwill print the value true.",
    "code": "ICipher cipher = ...;\nbyte[] data = ...;\nbyte[] encrypted = cipher.Encrypt(data);\n            \nConsole.WriteLine(cipher.Decrypt(encrypted).Equals(data));"
  },
```


FIGURE 9 – Documentation au format JSON générée à partir du XML

Examples

The following code will print the value true.

```
ICipher cipher = ...;
byte[] data = ...;
byte[] encrypted = cipher.Encrypt(data);

Console.WriteLine(cipher.Decrypt(encrypted).Equals(data));
```

FIGURE 10 – La documentation correspondante apparaissant sur le site web

Ainsi grâce à cette initiative, nous obtenons un nombre satisfaisant de pages de documentation qui aideront les développeurs les plus curieux comme mentionné plus haut.En voila un exemple :

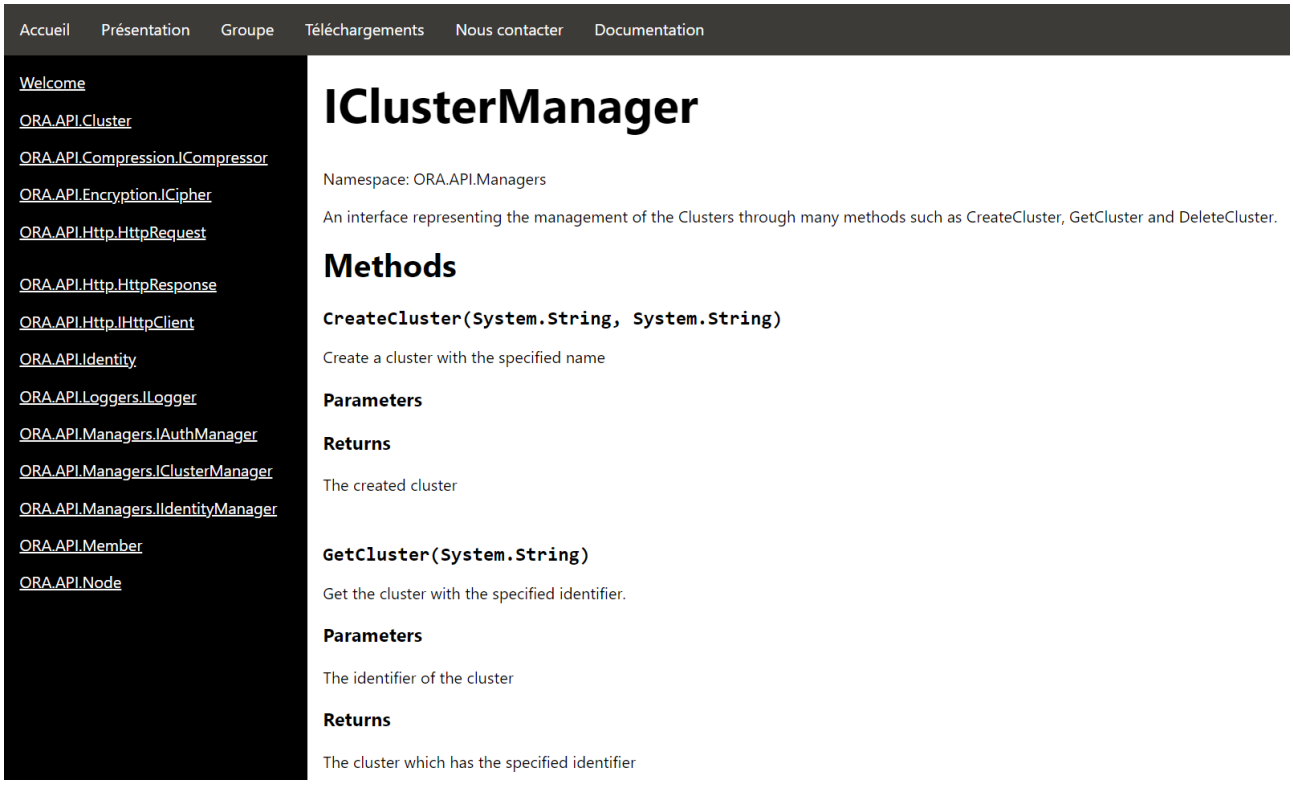

FIGURE 11 – Une page de documentation sur notre site avec à droite le sommaire de cette partie

3 Prochaine Soutenance

3.1 Tracker

Afin de finaliser notre application pour la prochaine soutenance, l'objectif est d'implémenter les dernières routes et modèles de donnée qui permette de faire fonctionner l'application et ses fonctionnalités correctement. Une fois ceci fini, l'objectif sera de réusiner le code et de l'optimiser.

3.2 Daemon

Le **Daemon** n'est pas un programme très complexe, en effet la tâche la plus difficile à été d'implémenter l'**IPC** qui permet à celui ci d'exposer les différentes méthodes de l'**API** d'ORA aux autres applications. Ce qu'il reste à implémenter pour la prochaine soutenance c'est donc le fait qu'il puisse rafraichir le **token** de l'utilisateur automatiquement avant son expiration ainsi que l'installation de celui-ci en tant que service sur le PC de l'utilisateur via un programme d'installation. De plus, le **Daemon** devra se charger de stocker les informations de l'utilisateur (les fichiers trackés, etc…) en mémoire.

3.3 API

L'objectif pour la prochaine soutenance du côté de l'**API** est de finir de l'implémenter. Nous devrons par exemple implémenter la classe des **Files** qui seront les fichiers ou les dossiers qu'échangeront entre eux les utilisateurs de notre application ainsi qu'une interface lié au *File Management*, qui sera en charge de la gestion de ces deniers.

3.4 Core

Pour la prochaine soutenance, nous devrons avoir un **Core** opérationnel. Nous devrons par exemple implémenter les méthodes de l'interface du *File Management* évoqué plus haut dans le partie **API**.

3.5 CLI

Le but de la prochaine soutenance est de finir notre application générée en ligne de commande qui pourra être utilisée comme notre application **GUI** et satisfaire les utilisateurs plus tournée vers une application comme celle-ci.

3.6 GUI

Notre application générale sera totalement prête pour la prochaine soutenance. Elle sera facile d'accès et pratique pour toutes les personnes voulant profiter de la technologie *Peer-to-Peer* afin d'échanger et de partager leur documents.

3.7 Site Web

Actuellement il y a quelques problèmes de formatage et de style sur la page de documentation, pour la prochaine soutenance l'objectif est de résoudre ces problèmes et d'améliorer le style. De plus, nous devrons continuer à documenter l'**API** afin d'avoir une documentation plus complète ainsi que les nouvelles fonctionnalités qui auront été rajoutées.

4 Conclusion

Pour cette soutenance intermédiaire, notre équipe est très contente au vu de l'évolution de notre application ORA. Celle-ci s'est soldée grâce à notre travail d'équipe, notre communication et à notre persévérance à vouloir construire ce projet qui nous tient à coeur. De plus, nous avons tout de même réussi à respecter une très grande partie de notre projection de l'avancement de notre application malgré des petite difficultés que nous avions oublié d'envisager au début de ce projet. Ceci étant, notre projet sera totalement opérationnel pour la soutenance prochaine et nous sommes déjà impatients de voir vos réactions devant notre démonstration finale.

5 Annexe

5.1 Vocabulaire

5.1.1 Paquet NPM

Un Node Package Manager ou paquet NPM est un paquet manager spécialement conçu pour Node.js qui est une plateforme logicielle libre en JavaScript orientée vers les applications réseau. Il vous permet de partager des Packages (aussi nommés modules) pour qu'ils soient accessible publiquement, permettant alors à d'autres utilisateurs de les installer simplement dans leur projet.

5.1.2 Framework

Un framework est un ensemble d'outils constituant les fondations d'un logiciel informatique ou d'une application web.

5.1.3 Wrapper

Un wrapper (ou class-wrapper) est une classe ou un ensemble de classes qui encapsule un ensemble de fonction non orientée objet.