PROGRAMMATION D'APPLICATION DISTRIBUÉE ET EN RÉSEAU

Projet groupe 5

Fabrice BODSON
Florian NICOLAS
Lucas THITEUX
Thibaut WATRISSE
Thomas COLLIGNON
Justin ROLLAND

Table des matières

Introduction

Dans le cadre de notre projet de programmation d'application distribuée et en réseau, il nous a été demandé de développer une application décentralisée permettant le partage de fichier entre plusieurs clients au travers d'un réseau. Ce système de fichiers repose sur un réseau de machines et chacune peut mettre à disposition du système un ensemble de fichiers dont elle possède une copie. De même, chaque machine du réseau peut demander le téléchargement d'un des fichiers du système de fichiers auprès des machines en disposant une copie.

Afin de concevoir cette application, nous avons commencé par réaliser le cahier des charges avec le client. Une fois celui-ci établit et validé, nous avons lancé une analyse UML de la demande du client au travers de schémas de type Use Case et de User Stories. L'un comme l'autre est expliqué dans ce rapport et ils sont disponibles dans des fichiers séparés de ce rapport. Afin de planifier notre travail, nous nous sommes servis d'un diagramme de Gantt.

Ensuite nous avons suivi la méthodologie 'Test Driven Development' afin de développer l'application. Nous avons également dû effectuer des choix techniques afin de choisir les technologies qui répondent au mieux aux besoins du projet. Durant sa réalisation, nous avons rencontré différents problèmes qui sont expliqués dans ce rapport.

Durant la lecture de ce rapport, vous retrouverez des explications plus détaillées sur notre vision de l'application, sur son architecture, nos choix techniques, des explications sur nos algorithmes, nos difficultés rencontrées ainsi que des informations complémentaires sur ce qu'il est nécessaire de connaître pour pouvoir utiliser l'application.

Cahier des charges

Il nous a été demandé de mettre au point un système de fichiers distribué reposant sur un réseau de machines. Chaque client peut mettre à disposition un ensemble de fichiers qu'il possède et souhaite mettre à disposition des autres clients.
Il faut bien évidemment qu'ils fassent partie du réseau applicatif. Chaque client peut demander le téléchargement des fichiers disponibles sur le réseau applicatif. Voici une liste sous forme de tirets représentant les différentes demandes recensées par le client.

1. **Version python demandée par le client**

➔ 3.10.7

2. **Rejoindre/Quitter le réseau**

   Lorsqu'un nouveau client souhaite rejoindre le réseau, ce dernier doit connaître l'adresse IP de l'une des machines déjà présentes sur le réseau et le numéro de port sur lequel la joindre.

   À tout moment les clients peuvent quitter l'application.

3. **Le listing des fichiers**

   Lorsqu'un client le désire, il peut obtenir la liste complète des fichiers disponibles. Celle-ci est créée dynamiquement par une requête.

4. **Le téléchargement d'un fichier**

   Une fois la liste obtenue, le client peut demander le téléchargement d'un ou de plusieurs fichiers disponibles dans cette liste.

5. **Sélection des fichiers à partager**

   Si plusieurs clients possèdent le même fichier, il faut qu'ils se mettent d'accord sur lequel se charge de l'envoi du fichier pour ce téléchargement. Pour ce faire, on priorise le client le moins occupé au moment de la demande de téléchargement.

   Le téléchargement du fichier est en lien direct entre le client effectuant la demande et le client élu.

6. **Répondre à une demande d'ajout au réseau applicatif**

   Lorsqu'un pair est contacté pour une nouvelle demande, le pair déjà présent envoie les informations nécessaires pour que le nouveau pair s'annonce sur le réseau.

7. **Maintenir sa liste de pairs à jour**

   À chaque nouveau pair sur le réseau, celui-ci contacte tous les pairs déjà présent afin qu'ils s'ajoutent mutuellement. Idem lorsqu'un pair quitte le réseau.

8. **Transfert de fichiers sécurisé**

9. **Numéro de port propre à chaque client**
   Pour éviter les conflits de ports et de communication, chaque client défini le port à utiliser pour le joindre. Un port par défaut est défini.

10. **Fusion de réseaux**
    Si deux réseaux applicatifs différents existent, n'importe quel client peut effectuer une fusion des réseaux afin d'en créer un seul avec tous les clients des deux réseaux applicatifs de base.

11. **Sécurisation des transferts de fichiers**
    Le transfert doit être sécurisé afin que le fichier ne soit pas intercepté ni consulté ni modifié.

Analyse du projet

User-stories

Pour démarrer notre analyse, nous avons commencé par rédiger les user-stories qui sont les exigences ou besoins du point de vue de l'utilisateur lors de l'utilisation de l'application. Elles nous ont permis de mettre sur papier notre interprétation des fonctionnalités de celle-ci.

Nous les avons ensuite présentées au client afin que nous nous mettions en raccord avec ses attentes et ses besoins dans ce projet. À la suite de cette présentation, nous les avons modifiées en fonction des remarques du client. Il nous est également arrivé de les modifier durant le développement de l'application. Voici donc les User Stories :

A) Un utilisateur doit ajouter sa machine sur le réseau de l'application, grâce à l'adresse IP et au port d'une des clients déjà présent sur ce réseau. L'utilisateur doit entrer le port sur lequel il est joignable.

   Tâches :

   - La machine doit faire une découverte dynamique du réseau.
   - La machine doit créer une liste des pairs.
   - La machine doit mettre à jour sa liste locale de pairs.
   - La machine doit mettre à jour la liste des pairs dès qu'une nouvelle communication est reçue.

B) Un utilisateur peut demander à télécharger des fichiers de n'importe quel type et de n'importe quelle taille auprès d'un pair.

   Tâches :

   - Le pair contacté possède une copie des fichiers.
   - Un utilisateur doit pouvoir effectuer autant de téléchargements qu'il le souhaite.
   - Les pairs doivent choisir lequel enverra un fichier au client.
   - Un utilisateur doit avoir accès à la liste complète des fichiers disponibles.
   - L'échange doit être sécurisé.

C) Un utilisateur peut lister ses pairs.
D) Un utilisateur peut enclencher une fusion de réseaux.
E) Un utilisateur peut se déconnecter.

Use-case

Une fois nos user-stories rédigées, nous avons pu schématiser le système avec ses objectifs en un diagramme que l'on appelle use-case. Tout en se détachant de l'aspect technique de l'application, il nous a également permis d'organiser notre travail en le divisant par fonctionnalités.

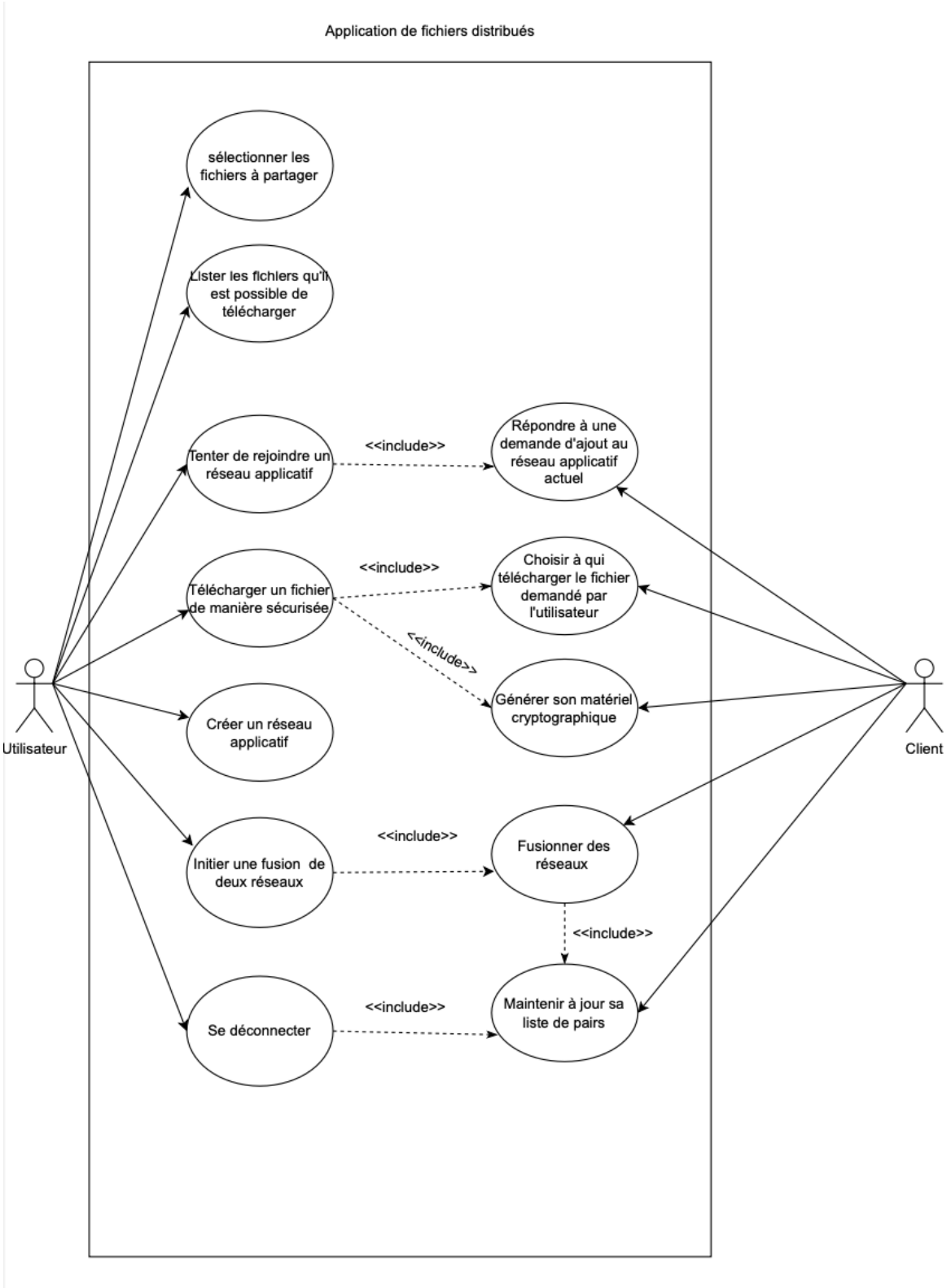

Figure 1- Use Case

Nous avons ensuite synthétisé la description de l'application dans un fichier présent dans la section documents de notre Git.

Types de fichiers pris en compte par l'application :

Afin de vérifier si tous types de fichiers peuvent être transférés par l'intermédiaire de l'application, nous avons effectués des tests. C'est avec succès que les fichiers suivants ont pu être transférés :

- .png
- .pdf
- .dockx
- .mov (gros fichier)

Diagramme de Gantt

À la suite de ces premières analyses, nous avons pu organiser notre travail dans le temps à l'aide d'un diagramme de Gantt jusqu'à la date de remise du projet. À savoir que ce document doit être remis plusieurs fois à jour étant donné les évolutions au cours de son développement. Vous trouverez le rendu final de notre diagramme sous le fichier « gantt.pdf » sur le repository gitlab.

Architecture de l'application

L'architecture de l'application est de type « peer-to-peer ». Il s'agit d'un type de réseau permettant à chaque machine d'agir en tant que client ou serveur. Le but étant de décentraliser le réseau et donc de faire en sorte que toutes les données ne se retrouvent pas sur un serveur central. Dans notre cas, les clients du réseau applicatif peuvent donc communiquer directement entre eux sans passer par l'intermédiaire de quoi que ce soit. Ce type de topologie est dite « fully-meshed » ou « topologie complète ».

Ce type d'architecture correspond donc parfaitement au principe de ce que l'on appelle réseau applicatif.

Vous retrouverez ci-dessous un schéma permettant de visualiser la topologie, représentant la communication entre les différents clients.

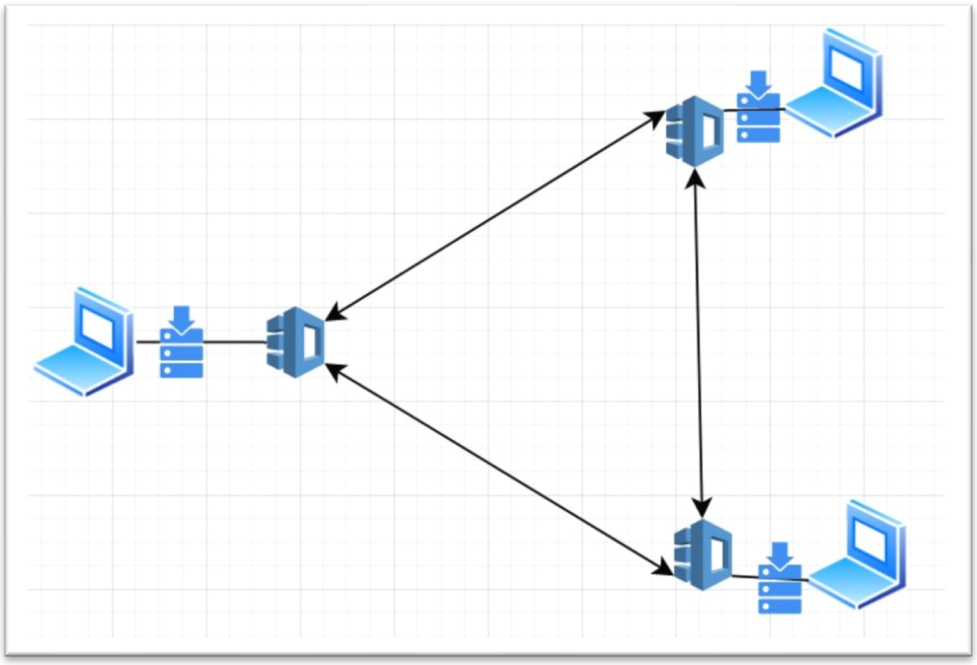

Figure 2 - Topologie du réseau applicatif

Fonctionnement de l'application

Tout d'abord, l'application s'utilise via le terminal et l'interaction s'effectue en sélectionnant des options dans les menus proposés. Ainsi, lorsque l'utilisateur lance l'application, il doit d'abord entrer le numéro de port sur lequel il sera joignable tant que l'application sera exécutée. S'il n'en choisit pas, le port 8000 est celui par défaut.

Ensuite, le menu de démarrage est proposé à l'utilisateur et celui-ci doit choisir s'il crée un réseau, en rejoint un ou bien s'il souhaite quitter l'application :

```
give the port you wanna use for socket connections (default is 8000) : 9000
the port 9000 will be use for socket connections

Press [1] to create a network.
Press [2] to join a network.
Press [q] to quit.
Choice: 1
Waiting for 1st incoming connection
```

Figure 3 - Menu de démarrage

Dans le cas où l'utilisateur souhaite créer un réseau, l'application va se mettre en attente de recevoir une première connexion car l'utilisateur est seul dans son réseau et ne peut donc interagir avec personne.

Dans le cas où l'utilisateur souhaite rejoindre un réseau, il doit entrer l'adresse IP du client qu'il souhaite contacter ainsi que le port sur lequel il est joignable. Une vérification de l'adresse IP fournie est effectuée afin de s'assurer que c'est bien une adresse IPv4.

Comme le montre la figure suivante, le pair contacté envoie un message pour confirmer que le nouveau pair est ajouté à sa liste de pair. En plus du message, il a envoyé sa liste de pairs afin que le nouveau pair ait la liste à jour.

Un travail sur la liste est effectué afin de placer le pair local en premier dans la liste et le pair contacté en deuxième.

Enfin, le nouveau pair prend contact avec chacun des pairs de la liste (excepté les deux premiers) afin qu'ils l'ajoutent à leur liste aussi. Dans le cas présent, un troisième pair est à contacter.

```
Enter peer to contact:
192.168.254.132
192.168.254.132 is a correct IP4 address.
enter the port of the joined peer : 7000

From peer: 'You are added to my peers list'

Actual peers list: ['192.168.254.1:9000', '192.168.254.132:7000', '192.168.254.142:8000']

Reaching out to all the peers...
Peers to contact : ['192.168.254.142:8000']


From peer: 'You are added to my peers list'
```

Figure 4 - Contact du troisième pair

À ce stade-ci, il y a un réseau composé de 3 pairs qui se trouvent tous les trois dans le même menu :

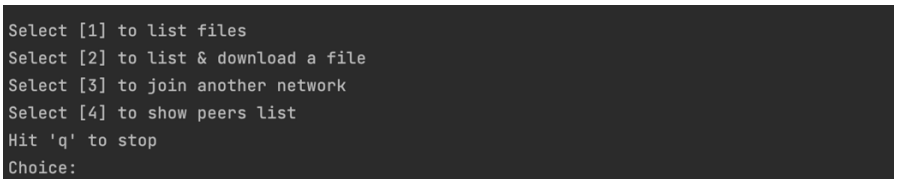

Figure 5 - Menu des trois pairs

Ce menu est le principal, il permet à l'utilisateur de lister les fichiers disponibles sur le réseau, d'en télécharger un, de fusionner avec un autre réseau, d'afficher la liste des pairs ou de quitter l'application.

Listing des fichiers

Le listing des fichiers est assez simple : une lecture du contenu du répertoire nommé 'DistribAppShare' est faite ainsi qu'une requête aux pairs pour qu'ils effectuent la même chose. Les pairs renvoient la liste des fichiers qu'ils ont. Enfin, une liste générale, sans doublons, est affichée à l'utilisateur :

Figure 6 - Aperçu de la fonctionnalité de listing de fichier

Téléchargement d'un fichier

Pour la seconde option, le même processus de listage est effectué. En plus, l'utilisateur doit entrer le numéro du fichier qu'il souhaite télécharger.
Afin de savoir quel pair doit envoyer le fichier demandé (lorsque plusieurs pairs possèdent le même), un algorithme de sélection est appliqué afin de sélectionner au possible un pair inoccupé. Si aucun d'eux n'est disponible, l'un d'eux est choisi de manière aléatoire.
Enfin, un échange de certificats est effectué afin de sécuriser le téléchargement du fichier :

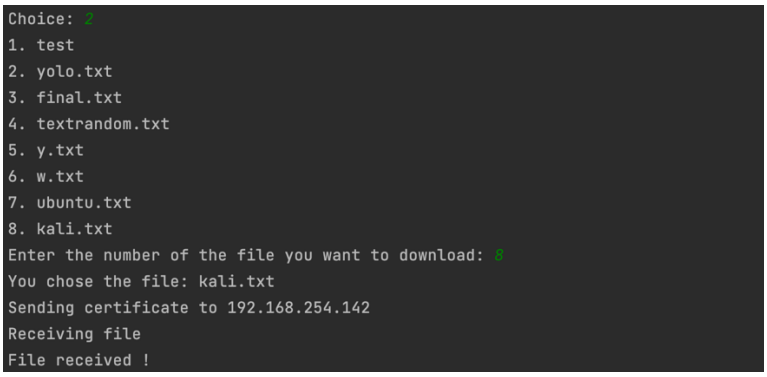

Figure 7 - Aperçu de la fonctionnalité de téléchargement de fichiers

Fusion de deux réseaux

Lorsque l'utilisateur initie une fusion de deux réseaux, il doit entrer les informations nécessaires pour contacter un client de l'autre réseau. Celui-ci envoie sa liste de pairs au client qui l'a contacté.
Le client qui a initié la fusion contacte chaque pair de cette liste reçue afin qu'ils s'ajoutent dans leurs listes.

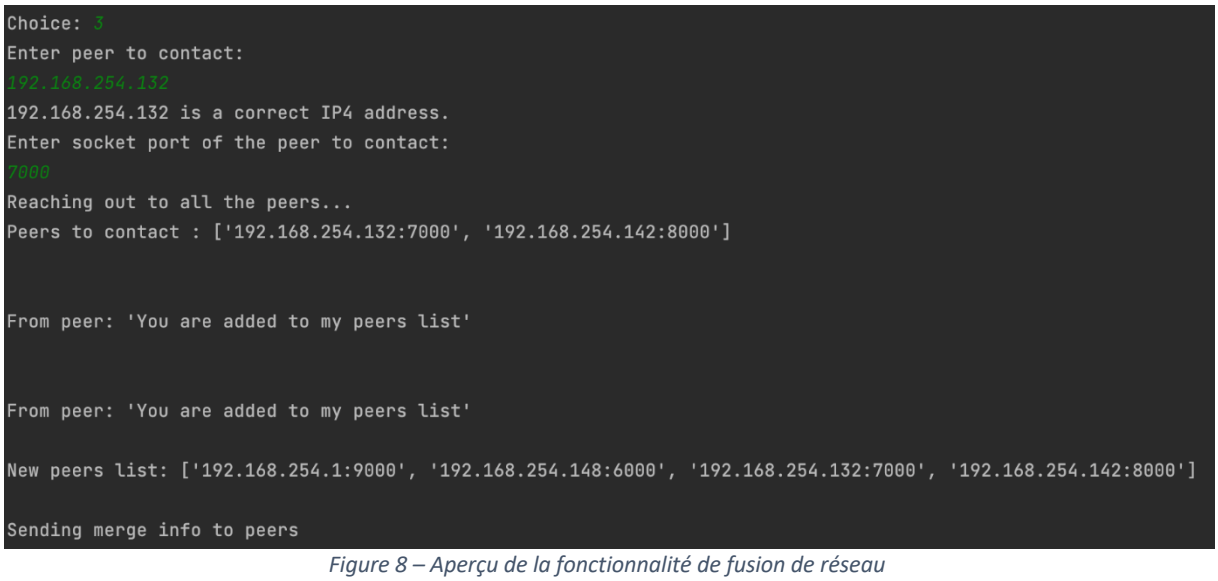

Figure 8 – Aperçu de la fonctionnalité de fusion de réseau

Enfin, il envoie cette nouvelle liste aux pairs présents dans son ancienne liste de pairs afin qu'eux aussi fusionnent.
Sur l'image ci-dessous, le pair reçoit l'informations qu'il faut fusionner les listes et va contacter chacun des pairs de cette nouvelle liste, même les anciens pairs.

```
Choice: Merging to a new network
Peer is already in the peers list

Peer is already in the peers list

Added to peers.
Actual peers list: ['192.168.254.148:6000', '192.168.254.1:9000', '192.168.254.
132:7000']

Added to peers.
Actual peers list: ['192.168.254.148:6000', '192.168.254.1:9000', '192.168.254.
132:7000', '192.168.254.142:8000']

Reaching out to all the peers...
Peers to contact : ['192.168.254.132:7000', '192.168.254.142:8000']
```

Figure 9 – Initiation de la fusion de deux réseaux

Le client contacté pour la fusion reçoit des demandes d'ajout à la liste de pairs comme si un nouveau client s'ajoutait au réseau.

```
Choice: Merge request received
From new peer: Contact request
Added to peers.
Actual peers list: ['192.168.254.132:7000', '192.168.254.142:8000', '192.168.254.1:9000']

From new peer: Contact request
Added to peers.
Actual peers list: ['192.168.254.132:7000', '192.168.254.142:8000', '192.168.254.1:9000', '192.168.254.148:6000']
```

Figure 10 - Phase de réception de la requête de fusion

Les deux dernières options permettent d'afficher la liste de pairs avec leur port à tout moment (le rendu est identique à la dernière ligne de la capture précédente) et de quitter l'application.

Choix techniques

Organisation de notre GitLab

Tout d'abord, nous avons répartis nos tâches en *issues*. Elles sont détaillées et disposent de « todo-list » afin de cocher les tâches ayant été effectuées lors de l'élaboration d'une fonctionnalité. En effet, nous avons créé une *issue* par fonctionnalité. Il est également évident que nous avons créé des *issues* pour la rédaction de documentation et toute autre tâche.

Pour la partie développement du projet, nous avions une première version de celui-ci et nous avons décidé de faire un « refactoring » étant donné que notre base n'était pas tout à fait en accord avec les principes du cours comme le TDD ou même la gestion du Git. Nous avons donc fait une branche « refactoring » afin de mettre celui-ci en place et y effectuer tous les changements nécessaires.

En parallèle de cette branche, une branche *dev* a été créée afin d'ajouter nos nouvelles modifications et de les rendre disponibles à tous les membres du groupe. Cela nous permet également de *push* nos tests avant d'écrire notre code, sans polluer le code fonctionnel de la branche *main*.

Enfin nous avons une branche *main* contenant le code « officiel », celui qui fonctionne et est testé.

Utilisation de librairies python tierces

Outre l'utilisation de time, random, socket, pytest, nous nous sommes tournés sur l'utilisation de librairies que nous n'avons pas forcément utilisés lors de nos différents cours. Nous avons tout de même utilisé la librairie « py-openssl » qu'il faut installer via pip.

Pickle

La librairie pickle est utilisée pour sérialiser et désérialiser des objets Python.
La sérialisation est un processus servant à convertir un objet en une série de bits, qui peuvent être stockés dans un fichier ou envoyés sur un réseau, tandis que la désérialisation est le processus inverse consistant à reconstituer l'objet à partir de cette série de bits.

Dans notre cas, nous nous en servons pour sérialiser nos listes lors d'une communication inter-client.

Par exemple, voici l'échange de la liste des fichiers disponibles pour un pair qui les envoie au pair qui lui a demandé la liste :

```
1.  # Envoi de la liste
2.  self.repository_content = os.listdir(self.repository)
3.  connection.send(pickle.dumps(self.repository_content))

4.  # Réception de la liste
5.  peer_files = pickle.loads(self.socket.recv(2048))
```

Platform

La librairie platform en Python est une bibliothèque qui permet de récupérer des informations sur l'environnement de l'application Python qui l'utilise. Elle fournit des fonctions qui

permettent de déterminer le système d'exploitation, la version de Python, l'architecture de la machine entre autres…

On l'utilise pour adapter la création du répertoire en fonction du système d'exploitation de la machine du client. Voici un exemple :

```
1.  if platform.system() == "Windows":
2.    path = f".\\DistribAppShare\\{filename}"
3.  else:
4.    path = f"./DistribAppShare/{filename}"
```

OS

La librairie os est une bibliothèque fournissant un accès aux fonctionnalités du système d'exploitation depuis l'application Python. Elle permet de réaliser diverses opérations liées au système d'exploitation, telles que la manipulation de fichiers et de répertoires, l'exécution de commandes du système, la gestion des processus, etc.

On utilise cette librairie pour la création et la lecture du répertoire 'DistribAppShare'. Dans l'exemple suivant, nous vérifions si le chemin du répertoire que nous voulons créer exite ou non et nous créons le répertoire s'il n'existe pas.

```
 1. def create_repo():
 2.     rep = ""
 3.     curr_platform = platform.system()
 4.     if curr_platform == "Windows":
 5.         rep = ".\\DistribAppShare"
 6.     elif curr_platform == "Linux" or curr_platform == "Darwin":
 7.         rep = "./DistribAppShare"
 8.
 9.     try:
10.         if not os.path.exists(rep): #if not os.path.exists(f"./{rep}"):
11.             os.mkdir(rep)
12.
13.     except os.error:
14.         print("Error while reading repository. Please check for permission error.")
15.
16.     finally:
17.         return rep
18.
```

SSL et pyOpenSSL.crypto

La librairie OpenSSL.crypto permet d'utiliser les méthodes pour générer des certificats et leurs clés afin de sécuriser les échanges de fichiers.

Avec la librairie SSL, il est possible d'utiliser les certificats afin de manipuler les sockets utilisés pour les sécuriser.

Ainsi, nous avons créé une classe Certificate afin de pouvoir manipuler le certificat facilement. On y crée la clé privée et on l'utilise pour générer le certificat. Les attributs d'initialisation de la classe sont les attributs du certificat. Finalement, le certificat est sauvegardé dans un fichier PEM.

```
1.  class Certificate:
2.  """
        Cette classe représente l'objet Certificat afin qu'il contienne tous les champs
    nécessaires dans un certificat.
        """
3.  def __init__(self, cert_file, key_file, subject):
4.          # Création d'une clé privée de 2048 bits
5.          self.__private_key = crypto.PKey()
6.          self.private_key.generate_key(crypto.TYPE_RSA, 2048)
7.          # Création d'un certificat auto-signé
8.          self.__cert = crypto.X509()
9.          self.cert.get_subject().CN = subject
10.         self.cert.get_subject().O = "My Organization"
11.         self.cert.get_subject().OU = "My Organizational Unit"
12.         self.cert.set_serial_number(1000)
13.         self.cert.gmtime_adj_notBefore(0)
14.         self.cert.gmtime_adj_notAfter(24 * 60 * 60)  # Durée de vie d'un jour
15.         self.cert.set_issuer(self.cert.get_subject())
16.         self.cert.set_pubkey(self.private_key)
17.         self.cert.sign(self.private_key, "sha256")
18.
19.         with open(cert_file, "wt") as f:
20.             f.write(crypto.dump_certificate(crypto.FILETYPE_PEM, self.cert).decode("utf-
    8"))
21.
22.         with open(key_file, "wt") as f:
23.         f.write(crypto.dump_privatekey(crypto.FILETYPE_PEM,self.private_key).decode("utf-
    8"))
24.
```

La librairie SSL permet d'utiliser un socket existant pour le chiffrer à l'aide du certificat généré comme le montre le code suivant :

```
1.      def send_file(self, peer, filename, client_socket):
2.          self.in_transfer("in_transfer", peer)
3.          time.sleep(0.5)
4.
5.          if platform.system() == "Windows":
6.              path = f".\\DistribAppShare\\{filename}"
7.          else:
8.              path = f"./DistribAppShare/{filename}"
9.
10.         # Configurer le contexte SSL/TLS
11.         context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
12.         context.load_cert_chain(certfile=self.cert, keyfile=self.private_key)
13.
14.         context.check_hostname = False
15.         context.verify_mode = ssl.CERT_NONE
16.
17.         # Boucle principale pour accepter les connexions des clients
18.         try:
19.             # Configurer le socket SSL/TLS
20.             ssl_socket = context.wrap_socket(client_socket, server_side=True)
21.
22.             print("Sending file...")
23.             with open(path, "rb") as f:
24.                 fb = f.read(2048)
25.                 while len(fb) != 0:
26.                     ssl_socket.sendall(fb)
27.                     fb = f.read(2048)
28.                 print("File sended !")
29.
30.         except Exception as e:
31.             print("Erreur lors de l'acceptation d'une connexion client :", e)
```

```
32.
33.         time.sleep(0.5)
34.         self.in_transfer("not_in_transfer", peer)
35.         client_socket.close()
```

Ceci est la méthode permettant d'envoyer un fichier. Cette méthode utilise le socket ouvert et le chiffre à l'aide de la librairie SSL et du certificat. Étant donné que les certificats sont des certificats auto-signés et auto-générés, nous avons désactivé la vérification de celui-ci car cela générait des erreurs de vérifications.

Quant à l'envoi de fichiers, voici le procédé suivi pour l'encryption :

Tout d'abord, les certificats sont échangés.

```
1.  # Cert exchange
2.  self.ask_for_cert(selected_peer, int(self.recover_port(selected_peer)), 'send')
3.  self.recv_cert(selected_peer)
```

Ensuite, le contexte SSL est créé et la vérification est bypassée et la connexion à l'envoyeur est faite avec un socket sécurisé :

```
1.  # Configurer le contexte SSL/TLS
2.  context = ssl.create_default_context(ssl.Purpose.SERVER_AUTH)
3.
4.  # Charger le certificat auto-signé pour la vérification
5.  context.check_hostname = False
6.  context.verify_mode = ssl.CERT_NONE
7.
8.  # Connexion au serveur en utilisant SSL/TLS
9.  with context.wrap_socket(message_socket, server_hostname=selected_peer) as ssl_socket:
10.     print("Receiving file")
11.     with open(path, "wb") as f:
12.         fb = ssl_socket.recv(2048)
13.         while len(fb) != 0:
14.             f.write(fb)
15.             fb = ssl_socket.recv(2048)
16.
17.     print("File received !")
```

Décisions concernant le fonctionnement de notre application

Choix du leader

Au début de la conception de notre application, nous avions en tête d'attribuer un rôle de leader afin de gérer l'accès au réseau applicatif, l'envoi et l'orchestration des téléchargements dans le cas où plusieurs clients disposent du même fichier.

Ce concept a été abandonné car il a été jugé plus simple de permettre à tous les clients de partager leur liste de pairs. Ainsi, dans le cas où plusieurs machines disposeraient du même fichier, ils se mettraient d'accord sur lequel envoie le fichier.

Par conséquent, l'implémentation de ce leader ne nous a pas semblé utile, nous avons donc préféré ne pas garder cette idée et d'appliquer une idée plus simple, expliquée au point suivant.

Élection du client qui envoie le fichier doublon

Nous appelons fichier doublon, un même fichier qui serait présent sur deux machines différentes. Lorsqu'un client souhaite télécharger un fichier présent sur le réseau et que ce dernier est présent chez plusieurs clients, nous nous basons sur le nombre de transferts (téléchargements) actifs pour déterminer quel client enverra son fichier.

L'algorithme est le suivant :

```
1.          for peer in owner_list[1:]:
2.              if peer in self.transfer_list:
3.                  pass
4.              elif peer not in self.transfer_list:
5.                  selected_peer = peer
6.          if selected_peer == None:
7.              selected_peer = random.choice(owner_list[1:])
8.
```

Une liste propriétaires du fichier est utilisée pour savoir lequel se trouve dans la liste de transfert. Un pair qui demande à télécharger un fichier est placé dans une liste de transfert :

```
 1. self.in_transfer("in_transfer", peer)
 2. def in_transfer(self, state, ip):
 3. """Cette fonction permet de définir l'état d'un pair comme étant occupé par un transfert de
fichier.
 4.          :param state: [string] En transfert ou non ("in_transfer" ou "not_in_transfer")
 5.          :param ip: [string] Adresse IP de la machine concernée
 6.          :param port: [int] Port utilisé
 7.          :return: L'état de la machine
 8.          """
 9.          # state -> not_in_transfer or in_transfer
10.          array = [state, ip]
11.          if state == "in_transfer":
12.              if ip not in self.transfer_list:
13.                  self.transfer_list.append(ip)
14.
15.          elif state == "not_in_transfer":
16.              if ip in self.transfer_list:
17.                  self.transfer_list.remove(ip)
18.
19.          for peer in self.peers[1::]:
20.              peer_ip = (peer.split(":"))[0]
21.              peer_port = (peer.split(":"))[1]
22.              self.create_socket()
23.              self.socket.connect((peer_ip, int(peer_port)))
24.              self.socket.send(pickle.dumps(array))
25.
```

Sécurisation de l'application

Concernant la sécurisation des échanges de fichiers, comme expliqué lors des explications sur les librairies SSL et pyOpenSSL.crypto, nous avons utilisé des certificats pour assurer la confidentialité et l'authenticité des échanges de fichiers.

Protocoles

Les différents protocoles présents dans ce code sont :

- TCP : qui définit comment établir et maintenir une conversation réseau via laquelle des applications peuvent communiquer.

- IP : qui définit comment des machines s'envoient des paquets de données.

Le port par défaut est le port 8000. Cependant, les utilisateurs peuvent choisir sur quel port ils sont joignable durant l'exécution de l'application. Ce port ne peut pas être changé via l'application, il faut quitter et relancer l'application pour changer de port. Ce port est enregistré comme un entier. La demande du port à utiliser se fait au lancement de l'app via le code suivant :

```
1. def define_port(self):
2.         try:
3.                   tmp_port = int(input("give the port you wanna use for socket connections
(default is 8000) : "))
4.                   print(f"the port {tmp_port} will be use for socket connections")
5.                   self.client.port = tmp_port
6.         except:
7.                   print("invalide port, 8000 will be use as socket port")
```

Si l'utilisateur se trompe dans le port entré alors il sera joignable sur le port par défaut.

Structures de données

Une structure de données est un format spécial destiné à organiser, traiter, extraire et stocker des données. En programmation, une structure de données peut être sélectionnée ou conçue pour stocker des données afin d'être manipulée par des algorithmes.

- Les structures de données sont souvent classées avec leurs caractéristiques qui sont :
    - La linéarité : indique si les éléments sont organisés chronologiquement ou non.
    - L'homogénéité : Les éléments sont du même type ou pas.
    - Statique ou dynamique : Si les emplacements de mémoires au moment de la compilation sont fixe ou non.

- Liste de chaînes de caractères pour les adresses IP des clients. (Structure linéaire, homogène, statique). Le nom de la liste dans notre code est « list_of_peers ».
    - self.peers[1:] ou encore self.peers[2:] :
        - Parfois, on retrouve ce genre de manipulation sur les listes afin d'éviter de prendre la première adresse IP de la liste de pair correspondant à l'adresse IP du pair ou le deuxième qui correspond à l'adresse IP du premier pair qui a précédemment été contacté selon les situations.
- Liste de chaînes de caractères pour les fichiers. (Structure linéaire, homogène, statique). Le nom de la liste dans notre code est « list_files() ».
- Liste de chaînes de caractères pour le transfert des fichiers. (Structure linéaire, homogène, dynamique). Le nom de la liste pour les transferts de fichiers dans notre code est « transfert_list [] ».
- Dictionnaires pour le transfert de fichier. (Structure linéaire, homogène, statique).

Analyse de l’efficacité des algorithmes

Complexité Client.py

get_network_files(self)

La complexité de cette fonction est O(n*t) où n est le nombre de fois que l’on exécute la première boucle et t le nombre de fois que l’on exécute la deuxième boucle à l’intérieur de la première.

list_files(self)

La complexité sera également O(n) où n est le nombre de fois que l’on parcoure la boucle.

select_file_to_download(self)

La complexité est O(n+t) où n est le nombre fois que l’on entre un mauvais choix et t le nombre d’éléments dans le dictionnaire de fichiers.

create_socket(self)

La complexité de cette fonction est constante et est donc de 0(1)

add_peer(self, peer_ip)

La complexité de cette fonction est de O(n), où n est la taille de la liste des pairs (self.peers). Cela est dû à la vérification de présence de l'adresse IP dans la liste à l'aide de l'opérateur "in" (qui a une complexité de O(n)) et à l'ajout de l'adresse IP à la liste (qui a également une complexité de O(n)).

create_network(self)

La complexité est O(n), où n est la taille des données reçues.

join_network(self, ip_to_reach)

Absence de boucle donc complexité constante O(1).

answer_request(self)

La complexité peut être exprimée comme O(n), où n est le nombre d'itérations jusqu'à ce que conn_is_alive soit False.

reach_to_peers(self)

La complexité de cette fonction dépend principalement de la taille de la liste des pairs à contacter (`peers_to_contact`). Si on suppose que la taille de la liste est « n », alors la complexité sera O(n).

disconnect(self)

La complexité de cette fonction est linéaire, c'est-à-dire O(n), où n est le nombre de pairs à contacter dans la liste `peers_to_contact`. Cela est dû à la boucle `for` qui itère sur chaque élément de `peers_to_contact` et exécute des opérations constantes à l'intérieur de la boucle.

In_tansfer(self, state, ip)

O(n-1) où n est le nombre de pairs dans la liste (moins 1 car on n’interprète pas le premier pair).

send_file(self, peer, filename, connection_socket)
La complexité est O(n) où n est la taille du fichier. Puisque l'on lit les fichiers par blocs et on les envoie au socket de connexion (donc est le nombre de blocs qu'on lit en tout).

def recv_file(self, owner_list)
La complexité est O(n+t), où n est le nombre de pairs détenant le fichier recherché, et où t est le nombre de blocs de bits téléchargés pour « générer » le fichier téléchargé.

merge_request(self, ip_to_merge, port_to_merge)
La complexité de cette fonction est O(n) où n est la taille de la liste new_peers.

send_merge(self, ip_to_merge, last_peer_index, port_to_merge)
La complexité de cette fonction est O(n-1), où n est le nombre de pairs dans la liste (-1 car on ne tient pas compte du premier pair).

send_cert(self, peer_ip, peer_cert_port=10000)
La complexité est O(n) où n est la taille du fichier. Puisque l'on lit les fichiers par blocs et on les envoie au socket de connexion (donc est le nombre de blocs qu'on lit en tout).

recv_cert(self, peer_ip, peer_cert_port=10000)
La complexité est O(n) où n est la taille du fichier. Puisque l'on lit les fichiers par blocs et on les envoie au socket de connexion (donc est le nombre de blocs qu'on lit en tout).

ask_for_cert(self, peer_ip, peer_port, text)
La complexité est de O(1).

Complexité Connection.py

input_ip()
La complexité de cette fonction est constante et vaut donc O(1).

manage_files(self, choice)
La complexité de cette fonction est constante et vaut donc O(1).

initialize(self)
La complexité de cette fonction est linéaire, O(n), où n est le nombre d'itérations nécessaires pour obtenir une entrée valide de l'utilisateur.

define_port(self)
La complexité est de O(1)

define_connection(self)
La complexité est O(n)

Explications supplémentaires

Cette partie a pour but d'expliquer les méthodes les plus complexes de notre code.

Join_network :

Cette méthode permet au client de rejoindre un réseau applicatif en insérant l'adresse IP d'un client membre du réseau qu'il veut rejoindre ainsi que le port sur lequel il veut le joindre.

Pour se faire :

- Il envoie d'abord un message au client pour rejoindre.
- Il reçoit la liste de pairs représentant la liste des clients disponibles dans ce réseau.
- L'ancien pair donne sa liste avec l'adresse IP du nouveau pair se situant en dernière place. La liste qui est reçue est donc réorganisée de sorte à ce que l'adresse du nouveau pair se trouve en premier dans la liste et en fait son adresse IP à lui.

Answer_request :

Cette méthode écoute les connexions entrantes. Une fois qu'une connexion a été établie, elle va recevoir des données du client connecté.

- Si ces données décodées contiennent « Join » ou « Contact », la méthode « add_peers » est appelée, un message contenant la liste de pairs agrémentée de celle du client est alors envoyé à ce dernier et la connexion est ensuite interrompue.
- Si ces données décodées contiennent « files », on vérifie que le répertoire existe et on va récupérer la liste des fichiers disponible dans ce répertoire et on la donne au client qui la demande.
- Si ces données décodées contiennent « Ask », la méthode « send_file » est appelée et le fichier demandé est envoyé.
- Si ces données décodées contiennent « Merge », une demande de merge des réseaux est envoyé aux autres pairs.
- Si les données décodées contiennent « New », les méthodes « add_peers » et « reach_to_peers », pour ajouter les nouveaux pairs dans la liste des paires de chaque clients et atteindre ses nouveaux clients.
- Si les données décodées contiennent « in_transfer » ou « not_in_transfer » cela signifie respectivement que l'un des pairs est entrain de transférer un fichier. Ce dernier est alors placé ou non dans une liste qui contient les différents pairs, qui sont entrain de transférer un fichier.
- Si ces données décodées contiennent "s-cert", cela signifie que le code va traiter l'envoi et la réception de certificats.
- Si ces données décodées contiennent "r-cert", cela signifie que le code va traiter uniquement l'envoi de certificats.
- Si ces données décodées contiennent « disconnect », alors le pair qui a envoyé ce message est retiré de la liste de pairs.

Sinon, cette méthode ne fait rien et écoute les connexions entrantes.

Get_network_files :

Cette méthode permet de visualiser les fichiers disponibles sur le réseau applicatif. Pour se faire :

- Le pair va contacter chaque membre de sa liste de pair.
- Il va envoyer une « file request ».
- Il va ensuite récupérer la réponse et la décoder pour ensuite la stocker dans une variable.
- Il va ensuite décortiquer le contenu de la variable afin de faire en sorte d'enregistrer les noms des fichiers dans la liste network_files

Select_file_to_download :

Cette méthode opère en deux temps :

- D'abord elle liste tous les fichiers disponibles sur le réseau et ensuite elle gère le choix de l'utilisateur.

- Comme les indexes des listes démarrent à 0, la méthode opère une soustraction sur le numéro choisit par l'utilisateur.

La méthode retourne le nom du fichier que l'utilisateur a choisi.

Difficultés rencontrées

Nous avons pris beaucoup de temps à nous adapter lors de la mise en place des tests unitaires. En effet, aucun de nous n'avais eu de formation orientée développement. Il nous a donc fallu du temps pour nous habituer au développement et surtout au travail en groupe.

La première partie du projet portait sur l'analyse et nous a donné du fil à retordre car nous n'avions pas l'habitude d'en réaliser, mais aussi car nous n'avions jamais travaillé ensemble auparavant. Cela a créé de nombreux problèmes dans la compréhension du projet ce n'est qu'au bout de plusieurs semaines, que ces divergences ont pu être remarquées. C'est pourquoi nous avons effectué une réunion pour redéfinir les bases du projet et les termes associés afin d'éviter, à nouveau, toutes ambiguïtés et permettre d'avoir des débats plus cohérents. Cela nous a fait perdre beaucoup de temps.

Une fois ce problème réglé, la phase de développement a pu débuter. Afin d'avancer de manière efficace, nous avions décidé qu'après chaque réunion avec le client, nous devions répartir le travail à effectuer en 2 groupes puis se réunir à nouveau le dimanche soir pour expliquer aux autres membres de l'équipe ce qui a été fait, ainsi que les difficultés rencontrées et de définir ce qui était à faire afin de préparer la prochaine réunion client. Cela ne pouvait cependant pas être toujours respecté. En effet, il pouvait arriver à l'un d'entre nous d'avoir un imprévu, ce qui nous obligeais à décaler ces réunions et pouvait alors rendre difficile le travail de préparation en vue de la réunion client qui suivait.

Il pouvait également arriver que nous rencontrions des grosses difficultés dans le développement, que ce soit dans l'implémentation d'une nouvelle fonctionnalité ou bien de rectifier le code proposé jusqu'à présent au client pour rectifier les tests et donc le ce dernier aussi.

En effet, après avoir développé une bonne partie de l'application, nous nous sommes rendus comptes que nous n'avions pas bien respecté les principes du TDD et qu'il était très important de rectifier cela. C'est pourquoi nous avons décidé d'effectuer un exercice similaire à celui de l'analyse, c'est-à-dire de nous réunir et de discuter tous ensemble de ce qui a été fait, de comment résoudre le problème et de nous mettre d'accord sur la façon de développer en respectant les principes de TDD.
C'est ainsi que nous avons recommencé par définir chaque fonctionnalité pour chaque méthode (définir son but, comment elle fonctionne, ce qu'elle produit comme résultat) afin d'avoir une idée claire de ce qui devait être testé (résultats attendus, gestion des résultats inattendus). C'est aussi comme ça que nous avons procédé pour la suite du projet.

Comme nous avions la responsabilité de respecter nos engagements, une dette technique, envers le client en avançant dans le développement de l'application, nous nous sommes répartis en 2 groupes : un groupe qui va s'occuper de ce que nous avons appelé le « refactoring » (pour revoir le code déjà écrit) et un groupe pour le développement des nouvelles fonctionnalités.

Nous nous sommes aussi rendu compte que nous n'avions pas bien géré la gestion du temps, car jusque-là, nous travaillions une journée sur le projet sans compter les heures. Dès lors, nous avons fixé un budget temps de 6h par personne par semaine hors réunions d'équipe en nous basant sur nos estimations du temps de travail qu'on passait auparavant et en les augmentant (étant donné que nous avions autant de travail en plus du « refactoring »).

Conclusion

Ce projet nous a montré la nécessité du respect de la méthodologie de travail lors de l'élaboration de projet en groupe. En effet, l'analyse, la planification et la communication sont fondamentales afin permettre l'avancement du projet. Cela nous a aussi démontré la difficulté de développer une application distribuée dans une équipe réunissant un nombre important de collaborateurs où il a été important de faire preuve de rigueur.

Ces différentes conclusions ont été découvertes à la suite d'expériences rencontrées soit pendant l'application de la méthodologie, soit durant l'écriture du code. Par exemple, nous avons perdu beaucoup de temps lors de l'élaboration de l'analyse, accorder nos visions était assez chronophage. Nous avons par la suite trouvé une solution nous permettant d'avancer de façon plus rapide et efficace. C'était pareil pour la méthodologie TDD. En effet, nous avons mis du temps à l'appliquer correctement, et il nous a même été nécessaire de revenir sur le code qui a déjà été écrit.

La méthodologie TDD nécessite une vision encore plus globale sur chaque fonction que l'on souhaite implémenter. Cela nous appelle à anticiper les cas les plus extrêmes en faisant des tests unitaires sur chaque fonction de la manière la plus complète possible. Cette méthode nous permet donc de proposer un code robuste.

Pour conclure, cela a été pour nous une expérience enrichissante et nous a apporté beaucoup de choses en nous faisant découvrir le fonctionnement du travail d'équipe dans un environnement de développement d'application.

Références

- https://en.wikipedia.org/wiki/Peer-to-peer
- https://docs.python.org/3/library/pickle.html
- https://www.geeksforgeeks.org/os-module-python-examples/
- https://www.lemagit.fr/definition/Structure-de-donnees - --
- https://www.maxicours.com/se/cours/structure-de-donnees-les-tableaux-structures-et-associatifs/
- https://fr.wikipedia.org/wiki/Table_de_hachage#:~:text=Une%20table%20de%20hachage%20est,du%20type%20abstrait%20tableau%20associatif

Table des figures