Bureau d’études

Projet Déploiement de Services et Interopérabilité : E-Mail

Encadré par
Patrice TORGUET

Réalisé par
Fatoumata salia TRAORE | Amina DIDANE

Table des matières

1. Introduction

1.1. Contexte du projet

Le courrier électronique (e-mail) constitue aujourd'hui l'un des principaux moyens de communication sur Internet. Il repose sur un ensemble de protocoles normalisés permettant l'échange de messages entre utilisateurs, indépendamment du matériel, du système d'exploitation ou du fournisseur de messagerie.

Ce bureau d'étude a pour objectif de comprendre, concevoir et implémenter les bases techniques d'un service de messagerie électronique, depuis la transmission des messages jusqu'à leur consultation à distance.

Dans cette première étape du projet, nous allons nous concentrer sur la mise en place d'un serveur SMTP simple, capable de recevoir et de stocker les messages envoyés par un client.

Pour cela , le serveur doit être capable de :

- accepter une connexion d'un client SMTP.
- traiter les commandes ci-dessus dans le bon ordre.
- stocker le message dans un fichier local.

1.2. Description générale du protocole SMTP

Le Simple Mail Transfer Protocol (SMTP), défini par la RFC 5321, est le protocole standard utilisé pour l'envoi et le transfert des courriers électroniques sur Internet.
Il fonctionne selon un modèle client-serveur, où le client envoie des commandes textuelles et le serveur répond avec des codes d'état normalisés.
Le processus d'envoi d'un message SMTP se déroule généralement comme suit :

- Le client établit une connexion TCP avec le serveur sur le port 25 (ou 587 pour une version sécurisée).
- Le client envoie une série de commandes (MAIL FROM, RCPT TO, DATA) afin d'indiquer l'expéditeur, le destinataire et le contenu du message.
- Le serveur accuse réception et enregistre le message.
- Le client termine la session avec la commande QUIT.

2. Analyse et conception

2.1. Spécifications fonctionnelles

Le serveur permet d'envoyer et de consulter des courriers électroniques à l'aide de clients standards. Les fonctionnalités fonctionnelles sont entre autres :

- Connexion client/serveur ;
- Envoi de courrier (SMTP);
- Consultation de courrier (POP3);
- Stockage local des messages ;

- Multiclient;

2.2. Architecture générale du système

Cette partie vise à comprendre comment les différents composants interagissent pour assurer la transmission, la réception et le stockage des courriels dans un environnement local.

Elle se compose de deux entités principales :

- **Côté client :** responsable de l'envoi des courriels.

  Le client (implémenté en Python dans notre cas) se connecte au serveur via le port 25 en utilisant une connexion TCP.
  Il envoie ensuite les commandes textuelles suivantes :

  1. MAIL FROM → indique l'adresse de l'expéditeur.
  2. RCPT TO → précise le destinataire du message.
  3. DATA → envoie le contenu du message jusqu'à la séquence de fin .

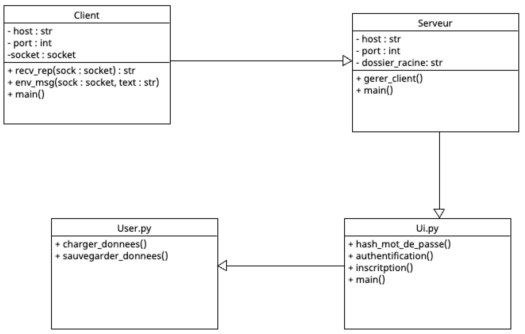
  Chaque commande reçoit une réponse du serveur selon le protocole SMTP.

- **Côté serveur :** chargé de recevoir, traiter et stocker les messages.

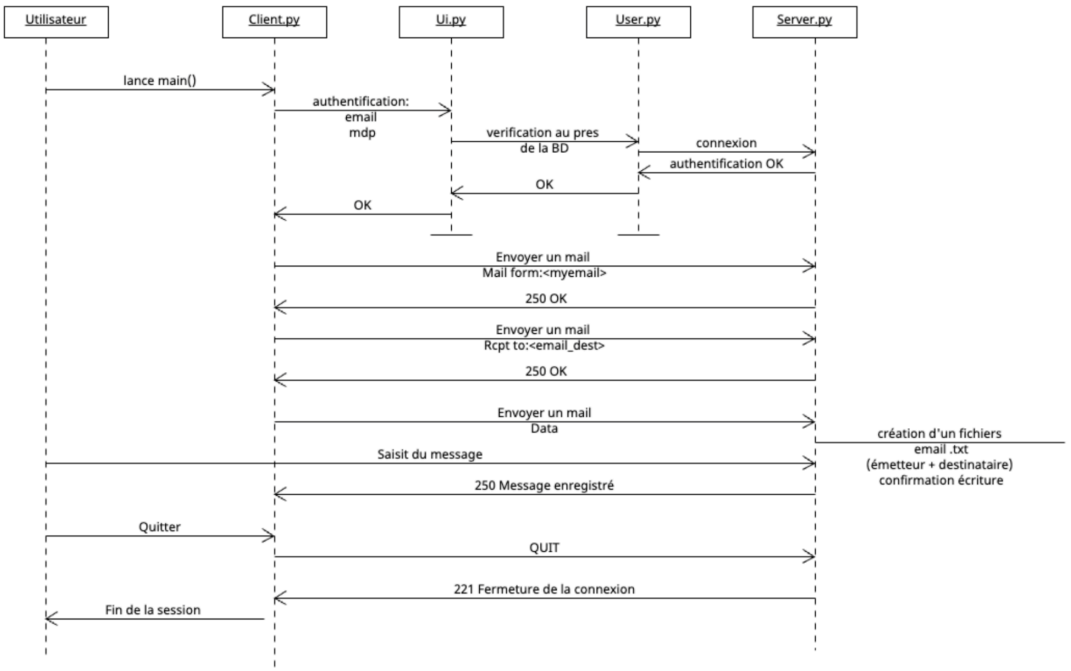
  Il écoute les connexions entrantes, traite les commandes reçues, et stocke les messages localement. Chaque connexion client est prise en charge dans un thread séparé, ce qui rend le serveur multi-utilisateur et capable de recevoir plusieurs courriels simultanément.

Cette architecture permet de bien séparer les rôles dans le système de messagerie.

2.3. Diagramme de classes Version 1



2.4. Diagramme de séquence Version 1



2.5. Description des modules et des fonctions principales

Server.py

gerer_client(connexion, adresse): Gère une connexion avec un client SMTP :

- Envoie le message d'accueil.
- Traite les commandes SMTP reçues (MAIL FROM, RCPT TO, DATA, HELLO, QUIT).
- Enregistre les messages dans des fichiers texte pour l'expéditeur.
- Ferme la connexion proprement lorsque le client se déconnecte.

Bloc principal if __name__ == "__main__":

- Crée le dossier racine des boîtes mail si nécessaire.
- Démarre la socket d'écoute du serveur sur le port configuré.
- Accepte les connexions entrantes et crée un **thread par client** pour gérer simultanément plusieurs clients.
- Vérifie l'inactivité si un timeout est configuré et ferme le serveur après une période sans client (optionnel).

Client.py

recv_rep(sock: socket.socket) -> str: Lit et retourne une réponse du serveur après l'envoi d'une commande SMTP.

- Reçoit les données depuis le socket.
- Décode les octets reçus en chaîne de caractères.
- Retourne une chaîne vide si la connexion est fermée.

env_msg(sock: socket.socket, text: str): Envoie une commande SMTP au serveur via la socket.

- Ajoute automatiquement \r\n si nécessaire.
- Convertit la chaîne en octets et l'envoie avec sendall().

main():

- Établit la connexion au serveur SMTP.
- Affiche le message d'accueil du serveur.
- Affiche le menu d'aide et gère la saisie des commandes par l'utilisateur.
- Traite les commandes SMTP (HELO, MAIL FROM, RCPT TO, DATA, QUIT, etc.).
- Lit et affiche les réponses du serveur après chaque commande.
- Fermer proprement la connexion à la fin de la session.

3. Développement de l'application

3.1 Technologies et outils utilisés

Le projet a été développé en Python ,l'environnement de développement utilisé est Visual Studio Code, pour la rédaction, le test et le débogage du code.
La gestion de version s'effectue avec Git et la plateforme GitHub, qui a servi à héberger le code source, suivre les versions et centraliser les livrables du projet.

Librairies Python utilisées

- **os** : pour gérer les dossiers et fichiers, créer automatiquement les dossiers des boîtes mail et construire les chemins d'accès aux fichiers.
- **socket** : pour créer la socket TCP, écouter les connexions des clients et échanger les commandes SMTP.
- **datetime** : pour générer des horodatages et nommer les fichiers de manière unique.
- **threading** : pour gérer plusieurs clients simultanément, en créant un thread par connexion.

- **socket** : pour se connecter au serveur et envoyer/réceptionner les commandes SMTP.
- **sys** : pour gérer proprement les erreurs et permettre l'arrêt du programme en cas de problème de connexion.

Pour les futures évolutions, on pourra envisager :

- **csv** pour stocker les messages dans un format structuré CSV.
- **ssl** pour sécuriser les échanges avec TLS/SSL

3.2 Gestion du stockage des messages

Le stockage des messages est organisé dans un répertoire local nommé boite_mail/. Chaque utilisateur possède un fichier distinct servant de boîte de réception (ex. : boite_mail/salia@gmail.com).
Ce dossier contient l'ensemble des courriels reçus et envoyés par cet utilisateur.

Lorsqu'un client envoie un message :

- Le serveur identifie le destinataire grâce à la commande RCPT TO.
- Après la commande DATA, il enregistre le contenu du message dans un fichier texte individuel pour l'expéditeur, contenant : expéditeur, destinataire, date et corps du message.
- Si le fichier n'existe pas, il est automatiquement créé.
- Chaque message est stocké dans un fichier séparé, pour plus de simplicité et de lisibilité

Évolution envisagée (version future) :

Le message pourrait être stocké dans un fichier CSV par utilisateur.Chaque ligne du CSV représenterait un message et contiendrait :

  - identifiant unique du message,
  - type du message (message envoyer ou recu)
  - adresse de l'expéditeur
  - adresse du destinataire
  - sujet, date et taille,
  - contenu du message (corps).

Le format CSV permettrait une gestion plus structurée, facilitant le tri, la recherche et le traitement automatique avec Python, tout en conservant une bonne lisibilité.

4. Tests et validation Version 1

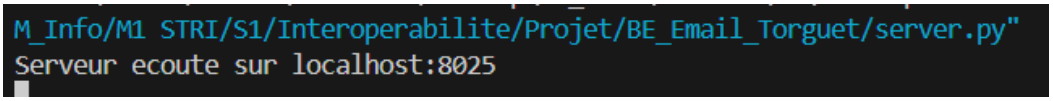
Notre version actuelle du serveur SMTP permet à plusieurs clients de se connecter simultanément et d'envoyer des emails.

Nous avons un système de double stockage :

- Emission : le mail est sauvegardé dans la boîte de l'expéditeur.
- Réception : le mail est également sauvegardé dans la boîte du destinataire.

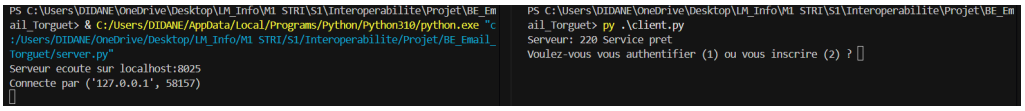
Procédure de test :

1. Lancement du serveur :



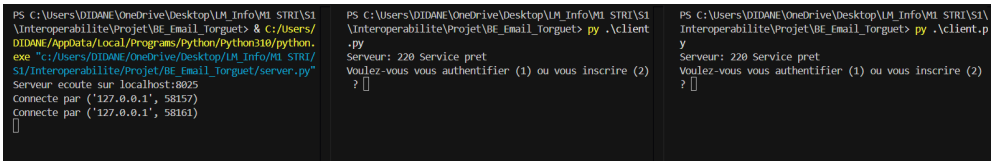
Le serveur écoute sur le port configuré et est prêt à accepter des clients.

2. Lancement du client 1 :



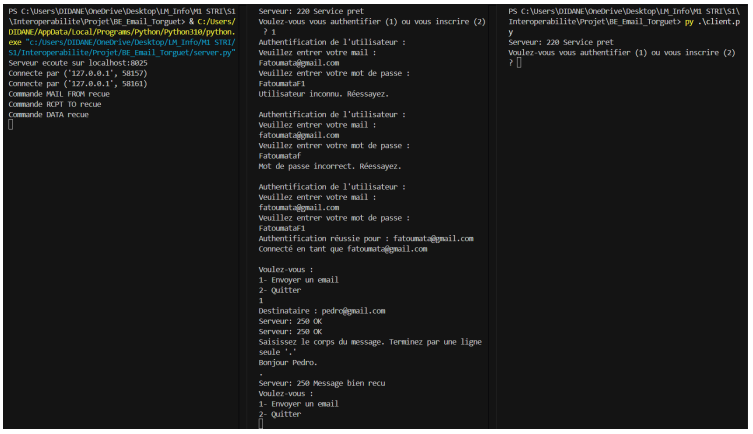
Connexion au serveur et vérification du message d'accueil du serveur.

3. Lancement du client 2 :



Connexion au serveur en parallèle. Les deux clients sont connectés au serveur, et le serveur est à l'écoute pour les deux clients.

4. Envoi des messages :



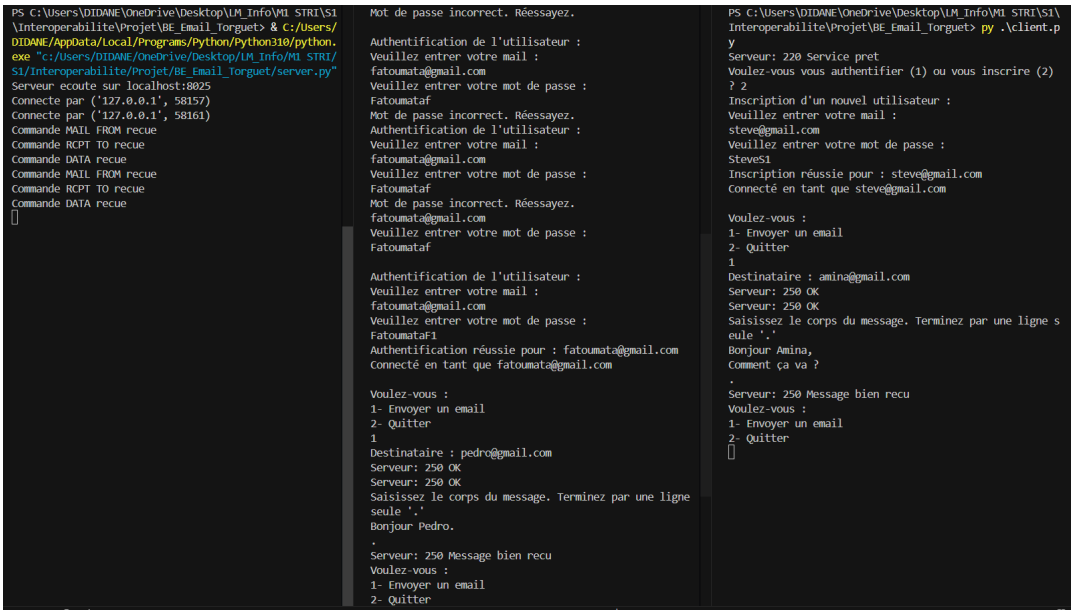
On remarque que L'utilisateur peut se connecter avec une adresse mail qui existe déjà et qui est stockée dans un fichier users.json. Si l'utilisateur essaye de se connecter avec une

adresse qui n'existe pas, un message d'erreur lui indique cela et lui demande de saisir une adresse existante. Si l'utilisateur saisi un mauvais mot de passe, un message d'erreur lui est affiché pour lui demander de saisir une adresse mail et mot de passe valide.

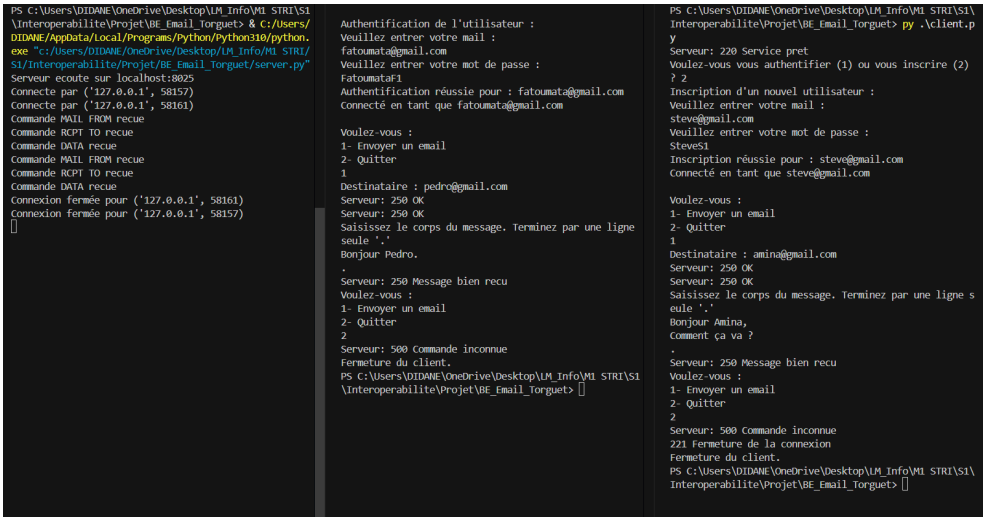
Une fois connecté, un message indique à l'utilisateur qu'il est bien connecté avec son adresse mail, et lui laisse le choix de soit envoyer un mail, soit quitter.

Si l'utilisateur souhaite envoyer un mail à un utilisateur existant, le mail sera ajouté dans le dossier des deux utilisateurs (émetteur et récepteur). L'utilisateur peut envoyer tant de mail qu'il souhaite, et il peut quitter à la fin. On remarque que le serveur a bien reçu les requêtes SMTP :

- "MAIL FROM:" quand l'utilisateur à sélectionner le choix 1 pour envoyer un mail ;
- "RCPT TO:" : quand l'utilisateur a saisi l'adresse du destinataire ;
- "DATA" : quand l'utilisateur a saisi le contenu du message.

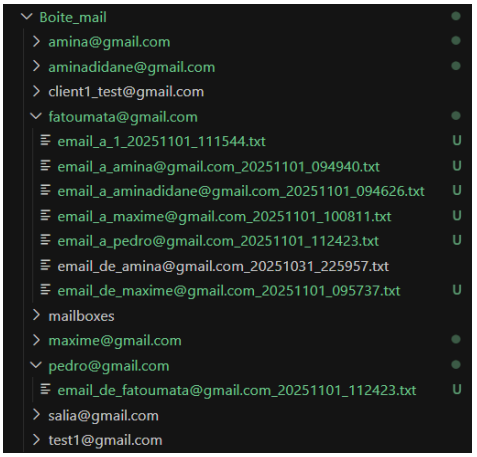


Le deuxième client, peut lui aussi envoyer un mail simultanément, sans que le client 1 quitte la connexion. Un utilisateur peut créer une adresse mail et un mot de passe, et sera connecté directement à cette adresse créée. Il peut ensuite envoyer des mails ou quitter.



Les deux clients peuvent quitter, et le serveur ferme bien la connexion avec ces deux clients.

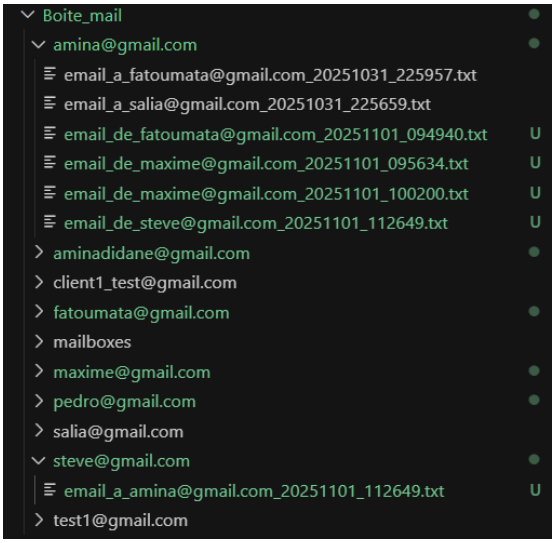
5. État des boîtes mail :

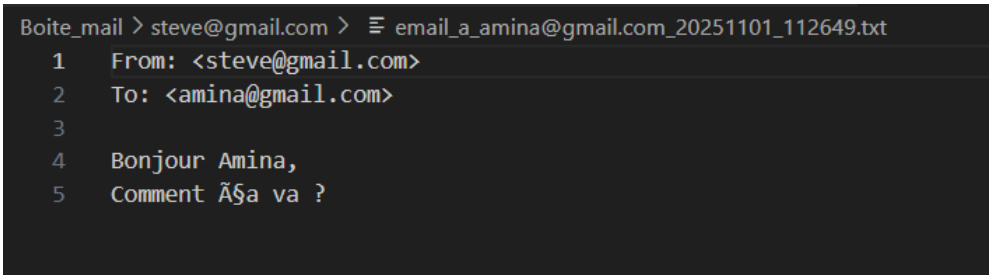


Après que le client 1 connecté en tant que fatoumata@gmail.com est envoyé un mail à pedro@gmail.com, un fichier .txt a été crée dans le dossier fatoumata@gmail.com ayant comme nom email_a_pedro@gmail.com, et un dossier a été créé avec le nom de pedro@gmail.com étant donnée qu'il n'existait pas avant. Ce dossier contient aussi le fichier qui contient le contenu du mail qui a été envoyé de fatoumata à Pedro.

Le client 2 a créé un compte steve@gmail.com, et donc il a été ajouté dans le dossier users.json :

Il a ensuite envoyé un mail à mail à amina@gmail.com, et donc un fichier .txt avec le contenu du mail a été ajouté dans le dossier de amina@gmail.com et de steve@gmail.com :





5. Version 2 : Protocole Hello et Elho

La Version 2 ne fait qu'ajouter la logique (EHLO → 502 → HELO → 250) nécessaire pour débloquer le dialogue initial et permettre au client d'utiliser les fonctionnalités de base (MAIL, RCPT, DATA) implémentées en Version 1.

C'est un mécanisme qui force le client à utiliser le protocole SMTP simple qu'on a déjà coder dans la version 1 de ce projet.

• **EHLO (Extended HELO) :** C'est la solution des clients modernes qui demandent à utiliser les fonctionnalités avancées du SMTP (pièces jointes complexes, sécurité, etc.),.

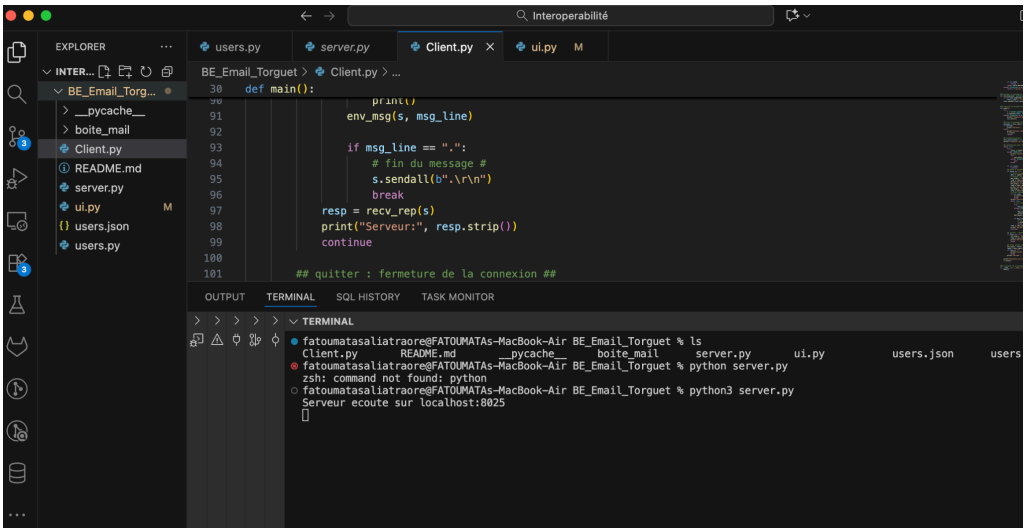
Le serveur rejette la demande en répondant **502 Command not implemented**. Cette réponse dit au client que ce serveur ne connaît pas cette commande, ou qu'elle n'a pas été implémenté.

Suite à cette réponse, le client va basculer sur la commande HELO, permettant au dialogue de continuer avec les fonctions basiques de SMTP.

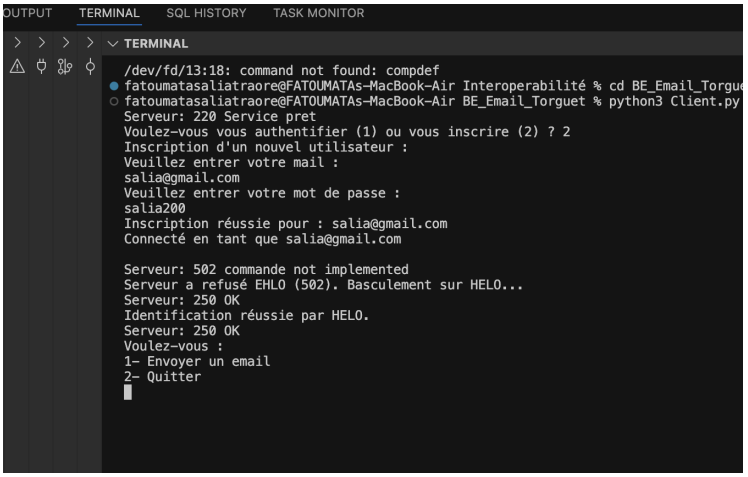
• **HELO (HELLO) :** C'est la salutation basique du protocole SMTP. Après avoir reçu un refus, pour le EHLO, le client va automatiquement réessayer avec HELO. Le serveur doit alors accepter cette simple connexion en répondant à 250 OK.

6. Test Version 2

Pour effectuer le teste après modification de notre code pour implémenter la gestion de ehlo et hello, nous commençons par lancer le serveur :



Ce test vérifie que la nouvelle logique ajoutée au client (le repli EHLO → HELO) fonctionne avec la nouvelle logique de votre serveur.



Nous pouvons contacter sur la capture d'écran ci-dessus que le client dès que il est connecté tente d'envoyer un Ehlo et on remarque que ce dernier a échoué (502) puis le client bascule en hello et l'identification a réussis (250 OK).

7. Protocole POP3 (Post Office Protocol)

Le protocole POP3 permet une synchronisation unidirectionnelle de l'e-mail, ce qui permet aux utilisateurs de télécharger des e-mails d'un serveur vers un client. Le protocole POP3 se distingue par plusieurs caractéristiques spécifiques. Tout d'abord, il effectue un téléchargement local des emails, ce qui signifie que les messages sont transférés du serveur directement vers l'appareil de l'utilisateur. Une fois ce téléchargement effectué, le protocole procède à une suppression automatique des messages du serveur. Ceci entraîne un accès limité aux emails, puisqu'ils ne sont plus accessibles que depuis l'appareil sur lequel ils ont été téléchargés. En revanche, l'avantage est que les messages peuvent être consultés sans connexion Internet, ce qui permet une utilisation hors ligne complète de la messagerie.

Les principales commandes qu'on va implémenter dans notre projet sont :

- QUIT : permet de quitter la session en cours ;
- STAT : indique le nombre de messages et la taille occupée par l'ensemble des messages ;
- LIST : donne une liste des messages ainsi que la taille de chaque message : un numéro suivi de la taille en octets ;
- RETR : numéro_du_message : récupère le message indiqué ;
- DEL : permet d'effacer un message spécifique;

8. IMAP (Internet Message)

Le protocole IMAP permet aux messages d'être stockés dans un serveur distant. Les utilisateurs peuvent se connecter via plusieurs clients e-mail sur des ordinateurs ou des appareils mobiles et lire les mêmes messages. IMAP sert d'intermédiaire entre les serveurs de messagerie et les clients de messagerie, au lieu de télécharger les messages du serveur vers le client de messagerie. Toutes les modifications effectuées dans la boîte aux lettres seront synchronisées sur

plusieurs appareils et les messages ne seront supprimés du serveur que si l'utilisateur efface l'e-mail.Voici un tableau qui illustre la différence entre le protocole IMAP et POP3 :

Caractéristique	IMAP	POP3
Accès aux messages	Les utilisateurs peuvent accéder à leurs e-mails depuis n'importe quel appareil.	Par défaut, les courriels ne sont accessibles qu'à partir de l'appareil sur lequel ils ont été téléchargés.
Gestion du serveur	Le serveur stocke les courriels et IMAP sert d'intermédiaire entre le serveur et le client.	Une fois téléchargés, les e-mails sont supprimés du serveur, sauf configuration contraire.
Accès hors ligne	Les courriels ne sont pas accessibles hors ligne.	Les courriels sont accessibles hors ligne, mais uniquement sur le périphérique de téléchargement.
Vitesse de chargement	Le corps du message n'est télécharger que si l'on clique dessus. L'objet et l'expéditeur apparaissent très rapidement.	Les e-mails sont téléchargés par défaut sur l'appareil. Le chargement initial peut donc prendre plus de temps.
Espace de stockage	Nécessite plus d'espace sur le serveur car les messages y restent stockés.	Préserve l'espace de stockage du serveur car les messages sont automatiquement supprimés après téléchargement.

8.1 Fonctionnement de l’envoie et de la réception des e-mails :

Voici un aperçu du processus d'envoi et de réception d'e-mails avec POP3 :

Envoi d'e-mails

Le protocole SMTP (Simple Mail Transfer Protocol) définit la manière dont les e-mails sont envoyés entre un client de messagerie et un serveur d’envoi.

Dans un premier temps, une connexion TCP est établie entre le client et le serveur SMTP. À l'ouverture de cette connexion, le serveur envoie un message d'accueil indiquant qu'il est prêt à recevoir des commandes. Le client envoie ensuite une séquence de commandes SMTP permettant de transmettre l'e-mail :

- HELO ou EHLO : identification du client auprès du serveur ;
- MAIL FROM : indication de l'adresse de l'expéditeur ;
- RCPT TO : indication de l'adresse du destinataire ;
- DATA : transmission du contenu du message.

Une fois le message entièrement reçu, le serveur SMTP le traite et le stocke sur le serveur. Dans le cadre de ce projet, les e-mails sont enregistrés sous forme de fichiers dans un répertoire correspondant à la boîte aux lettres du destinataire. Cette approche permet de simuler le fonctionnement d'un serveur de messagerie réel sans implémenter les mécanismes complexes de routage DNS et d'enregistrements MX.

Récupération des e-mails

La récupération des e-mails est assurée par le protocole POP3 (Post Office Protocol version 3), qui permet à un client de messagerie d'accéder aux messages stockés sur un serveur.

Lorsqu'un utilisateur souhaite consulter ses e-mails, une connexion TCP est établie entre le client et le serveur POP3. Après l'authentification de l'utilisateur, le client peut interagir avec la boîte aux lettres du serveur à l'aide de commandes POP3 standard.

9. Spécifications techniques : Version finale (3&4)

Cette section marqué l'évolution de notre compréhension du code , dans un premier temps nous listerons les spécification déjà présentes et maintenus des version 1 et 2 puis nous expliciterons les changement/amélioration apportés et leur raison :

A. **Spécifications maintenues (Héritage des Versions 1 & 2)**

Nous avons conservé les fondations solides établies précédemment :

- Architecture Client/Serveur : Utilisation des sockets TCP pour une communication fiable entre le client et les deux services (SMTP et POP3).
- Multitâche : Utilisation de threads pour gérer plusieurs clients simultanément.
- Système de Fichiers (FS) : Une structure arborescente simple pour le stockage des messages en format texte.
- Authentification (Hachage MD5) Pour garantir la confidentialité, les mots de passe sont hachés avant stockage.
- Handshake EHLO/HELO : Implémentation d'une poignée de main avec "fallback" (repli). Si EHL0 échoue, le client tente HEL0, assurant une interopérabilité maximale.
- Intégration du multi tâches : threads.

B. Améliorations majeures et Logiques de la Version 4

La version finale introduit des mécanismes de sécurité et de robustesse conformes aux exigences professionnelles :

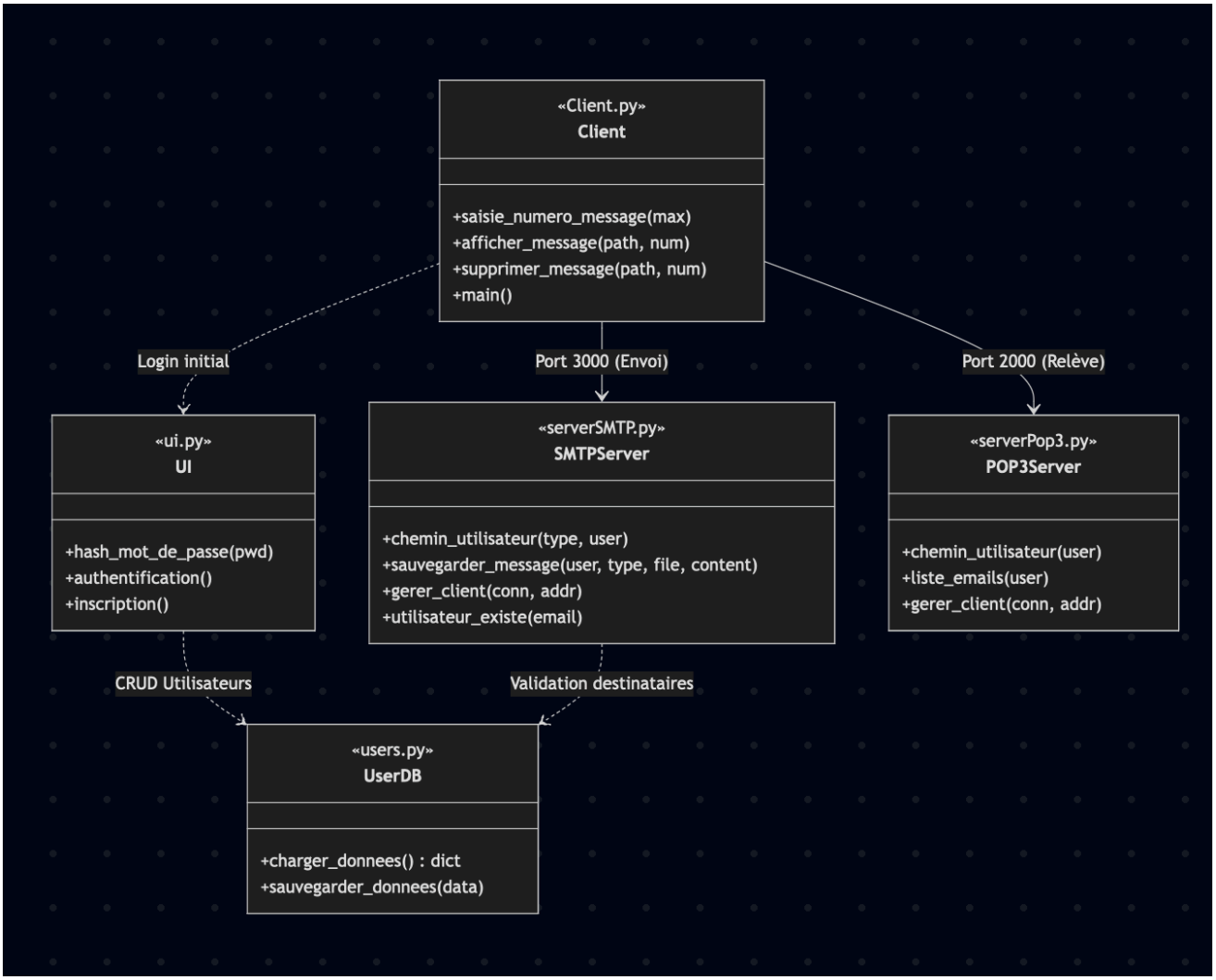
- **Sécurité et Authentification** : Blocage des tentatives : Une logique de limitation à 3 tentatives a été ajoutée.
- **Modularité** : La version finale de notre application repose sur une factorisation généralisée des fonctionnalités majeures. Cette approche de a pour but:
    - Réduction de la Redondance
    - Encapsulation des Responsabilités
    - Facilitation de la Reprise en Main
- **Structuration des données Arborescence des Dossiers:**

  Le choix d'une structure de répertoires hiérarchique a été retenu pour sa simplicité et sa fiabilité :

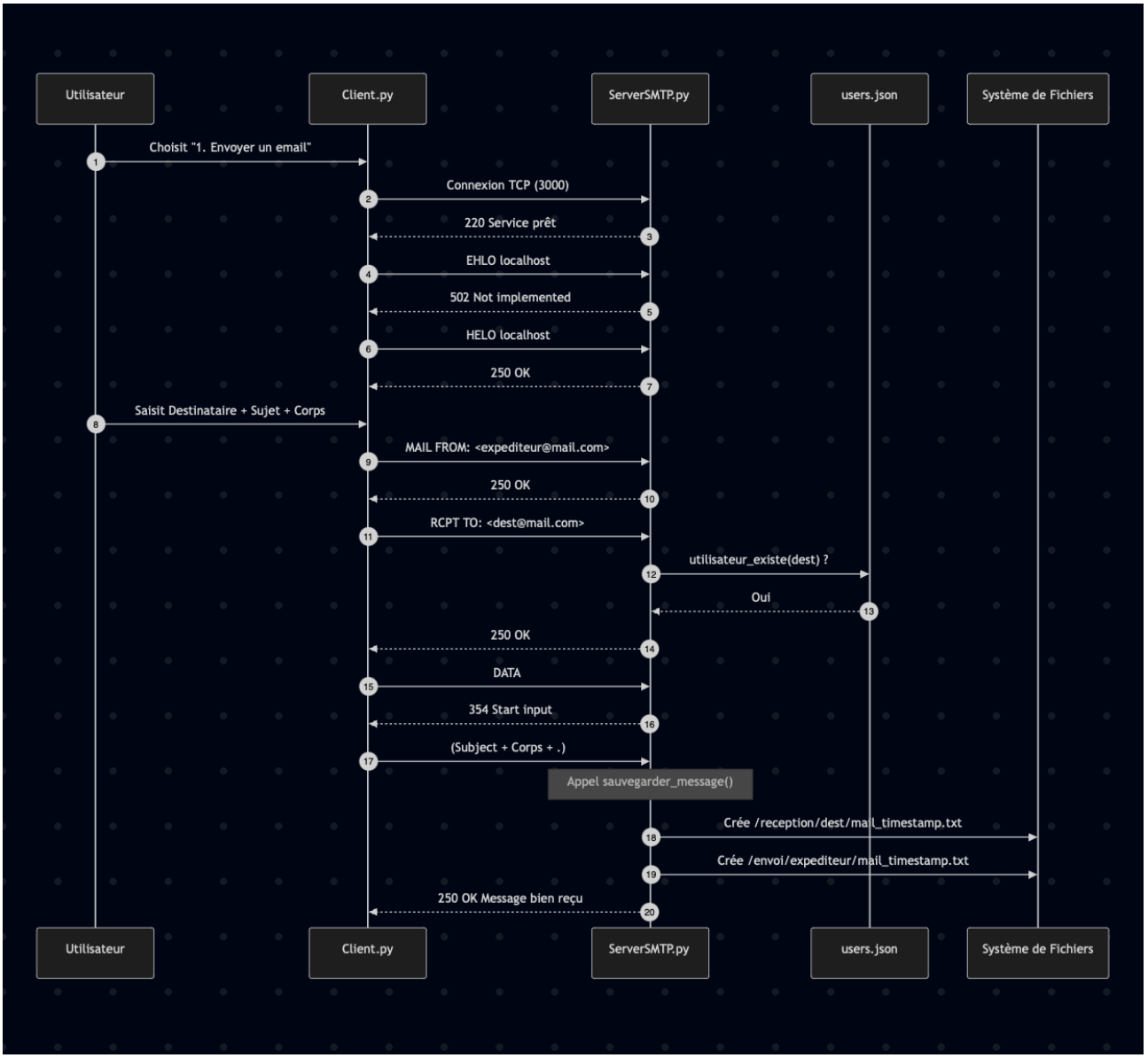
    - **Isolation des flux** : La séparation entre réception et envoi permettant de distinguer clairement les données gérées par le protocole POP3 (flux entrants) de l'historique local (flux sortants).
    - **Passage à l'échelle** : Chaque utilisateur possède son propre sous-dossier, ce qui réduit le nombre de fichiers par répertoire et accélère les opérations de lecture/écriture.

10. Modélisation Définitive

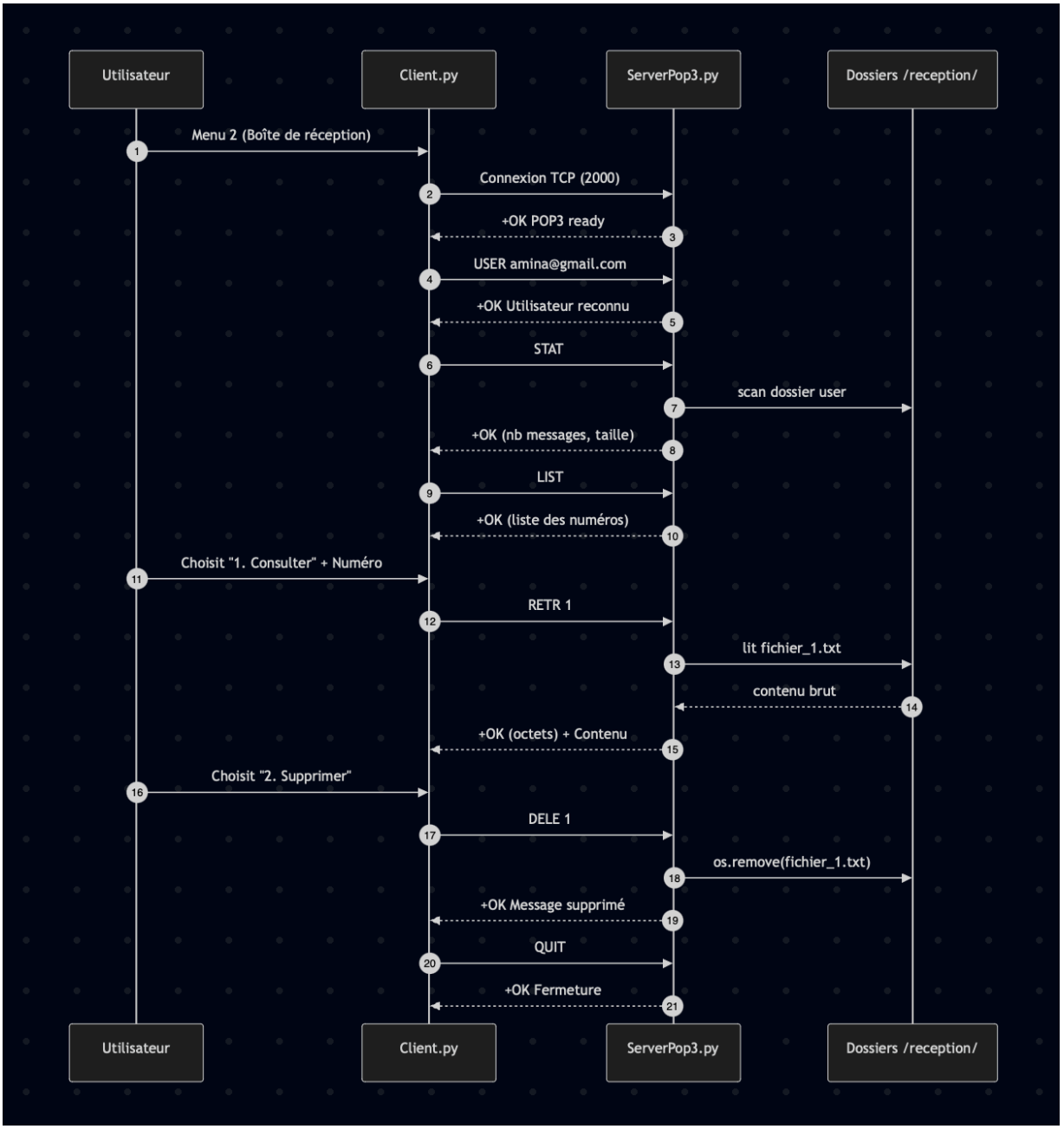
Diagramme de classe :



Diagrammes de séquences :



Flux SMTP



Flux POP3

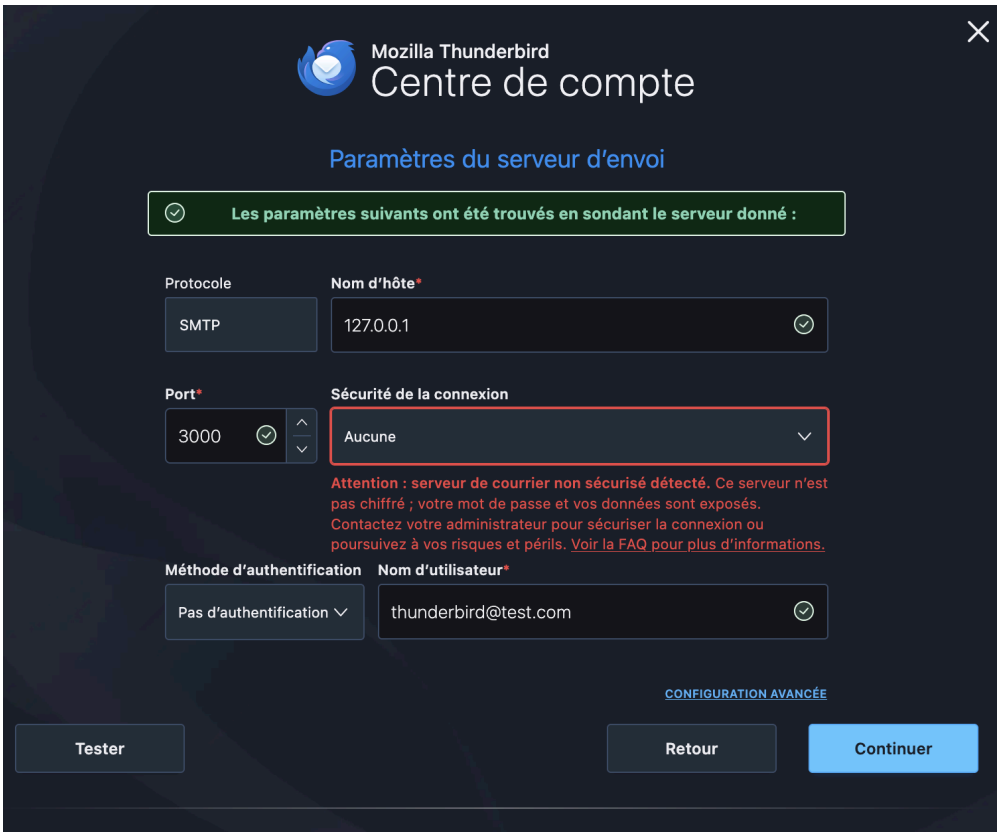
11. Test version finale : Thunderbird

L'objectif de cette étape est de prouver l'interopérabilité de nos serveurs SMTP et POP3 avec un client de messagerie standard du marché. Nous avons choisi Mozilla Thunderbird pour valider que nos scripts respectent les protocoles

1. Configuration manuelle des serveurs

Le compte a été configuré avec l'adresse thunderbird@test.com en utilisant les paramètres locaux suivants :

- Serveur entrant (POP3) : Nom d'hôte 127.0.0.1, Port 2000, Sécurité Aucune.
- Serveur sortant (SMTP) : Nom d'hôte 127.0.0.1, Port 3000, Sécurité Aucune, Méthode Pas d'authentification.



Configuration du serveur SMTP

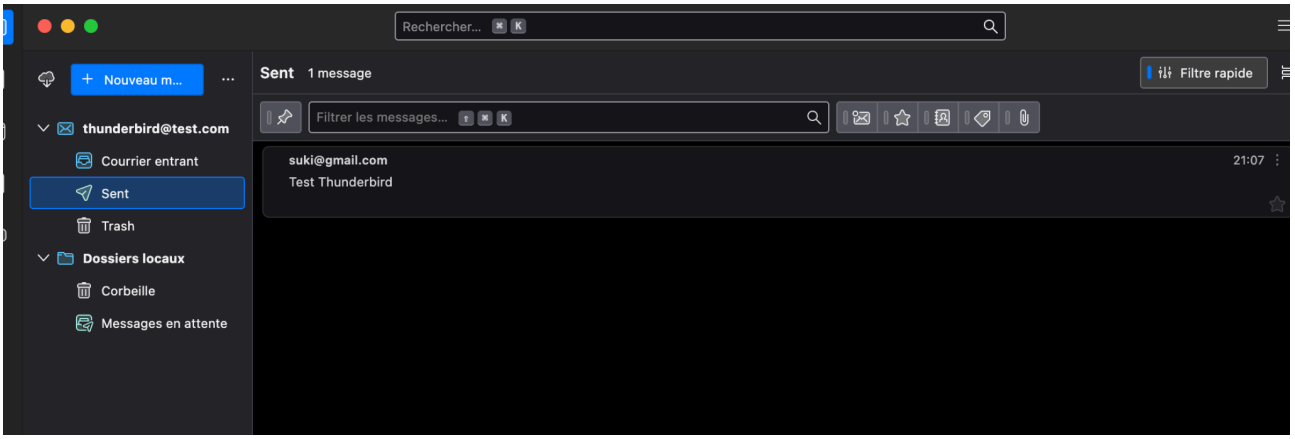
La validation par Thunderbird confirme que les deux serveurs sont à l'écoute et répondent correctement aux sondages initiaux.

2. Scénario de test : Envoi et Réception de bout en bout

Le test consiste à envoyer un mail depuis notre propre client Python et à le réceptionner graphiquement dans Thunderbird.

Étape 1: Envoi via SMTP

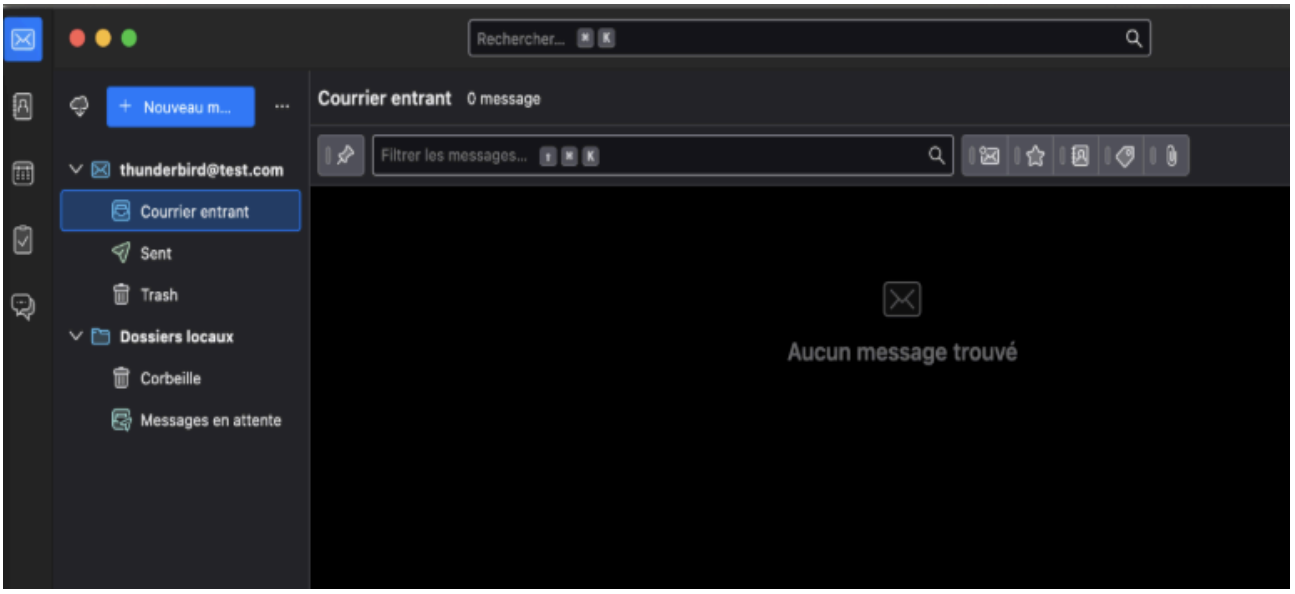
Nous avons utilisé la messagerie du compte thunderbird@test.com pour envoyer un mail depuis l'application à un client de notre serveur smtp. Le terminal confirme que le serveur SMTP a bien reçu et transmis le message. Le fichier est correctement créé sur le disque dans le répertoire de réception.




```
--- Connexion ---
Mail : suki@gmail.com
Mot de passe : test1234

==> Authentification réussie !!

 Connecté en tant que : suki@gmail.com

    --- MENU PRINCIPAL ---
1- Envoyer un email
2- Consulter la Boîte de réception
3- Consulter les Messages envoyés
4- Se deconnecter

Veuillez renseigner une option : 2

================================================================================
============================== BOÎTE DE RÉCEPTION ==============================
N°  | Expéditeur               | Date & Heure         | Taille
--------------------------------------------------------------------------------
1   | thunderbird@test.com     | 09/01/2026 21:07     | 439 octets
================================================================================

Voulez-vous :
1- Consulter un mail
2- Supprimer un mail
=> Appuyer sur m'importe quel autre touche pour revenir a la page d'acceuil
```

```
2- Supprimer un mail
=> Appuyer sur m'importe quel autre touche pour revenir a la page d'acceuil

Veuillez renseigner une option : 1

Numéro du message : 1

                    CONTENU DU MESSAGE

+OK 439 octets
From: thunderbird@test.com
To: suki@gmail.com
Subject: Test Thunderbird
Date: 09/01/2026 21:07:24

Message-ID: <aaad3be9-082f-48fa-82d3-cf929fba3d19@test.com>
Date: Fri, 9 Jan 2026 21:07:24 +0100
MIME-Version: 1.0
User-Agent: Mozilla Thunderbird
Content-Language: fr
To: suki@gmail.com
From: Test Thunderbird <thunderbird@test.com>
Content-Type: text/plain; charset=UTF-8; format=flowed
Content-Transfer-Encoding: 7bit

Test Thunderbird.
```

Malgré la réussite de la configuration des serveurs et la validation de l'envoi SMTP depuis Thunderbird , nous avons rencontré une difficulté majeure lors de la phase finale : la visualisation des messages entrant dans l'interface graphique de Thunderbird via le protocole POP3. Bien que le fichier de données soit physiquement présent dans le répertoire de réception (boite_mail/reception/thunderbird@test.com/) et que son contenu soit techniquement correct , l'interface de Thunderbird affiche "Aucun message trouvé"



En somme, nous n'avons pas pu valider l'affichage visuel dans Thunderbird car, même si nos serveurs tournent et que les mails sont bien enregistrés sur le disque, l'application reste vide. Le problème semble venir d'une exigence de formatage ultra-stricte de Thunderbird.

12. Conclusion

Le développement de ce projet a permis d'améliorer nos compétences en codage et en modélisation logicielle. Le passage par plusieurs révisions successives a favorisé une meilleure communication entre les modules et une gestion de projet plus structurée. Cette approche a facilité notre prise en main technique entre les différentes versions, permettant une compréhension du fonctionnement des protocoles SMTP et POP3 et une logique dans notre implémentation.

Bien que l'application soit fonctionnelle, l'expérience utilisateur peut être optimisée par une gestion plus fine des identifiants et des affichages. La sécurité actuelle présente des marges de progression, notamment sur la gestion des mots de passe trop courts et la vérification des indicatifs d'adresses e-mail. Une amélioration future consisterait à simuler une réelle adresse e-mail propre au serveur pour renforcer le réalisme de l'interface.Entre autre nous avons comme autre amélioration éventuelles :

- **Base de données** : Peaufiner le fichier users.json avec des informations complémentaires pour enrichir le profil utilisateur.
- **Journalisation** : Améliorer le système de logs pour assurer une meilleure traçabilité des événements serveurs.
- **Validation** : Renforcer les contrôles de saisie sur les formats d'adresses et les politiques de sécurité des comptes.

Annexes

- Lien vers la vidéo de démonstration V1 : DIDANE_TRAORE_Mail_v1.mp4
- **Lien vers la vidéo de démonstration Version finale : DIDANE_TRAORE_Mail_v4.mp4**
- Lien vers le dépôt git : lien
- Références (RFC 5321)

Bibliographie

SMTP

Mécanismes SMTP

POP3

SESSION_POP

RFC POP3