Dossier de programmation

Post Office Protocol 3

Polytech Lyon

Jérémy DOLLE – Mélanie PETITCUENOT

Sommaire

Ce document présente l’application client/serveur conforme à la norme RCF 1939 « Post Office Protocol 3 ». Vous y trouverez les différentes fonctionnalités de l’application ainsi que les éventuelles erreurs que vous pourrez rencontrer lors de son utilisation.

[I. Application Serveur 2](#_Toc478401554)

[a. Fonctionnalités 2](#_Toc478401555)

[b. Démarrage 3](#_Toc478401556)

[c. Gestion de la commande reçue 3](#_Toc478401557)

[II. Application Client 5](#_Toc478401558)

[a. Fonctionnalités 5](#_Toc478401559)

[b. Erreurs possibles 7](#_Toc478401560)

[c. Initialisation 8](#_Toc478401561)

[d. Envoi d’une requête 8](#_Toc478401562)

[e. Lire la réponse 9](#_Toc478401563)

[III. Connexion sécurisée 9](#_Toc478401564)

Table des illustrations

[Figure 1 : Fenêtre de lancement du serveur 2](file:///C:\Users\jeremy\IdeaProjects\Server_Client_POP3\Compte-rendu\CR.docx#_Toc478401661)

[Figure 2 : Connexion TCP établie avec un client, en attente de l’authentification 2](file:///C:\Users\jeremy\IdeaProjects\Server_Client_POP3\Compte-rendu\CR.docx#_Toc478401662)

[Figure 3 : Fichier utilisateur 3](file:///C:\Users\jeremy\IdeaProjects\Server_Client_POP3\Compte-rendu\CR.docx#_Toc478401663)

[Figure 4 : Fonction run() lancée après connexion d'un client 3](file:///C:\Users\jeremy\IdeaProjects\Server_Client_POP3\Compte-rendu\CR.docx#_Toc478401664)

[Figure 5 : Liste contenant les commandes connues du serveur 4](file:///C:\Users\jeremy\IdeaProjects\Server_Client_POP3\Compte-rendu\CR.docx#_Toc478401665)

[Figure 6 : Méthode answerCommand() qui gère la redirection de la fonction answerCommand() 4](file:///C:\Users\jeremy\IdeaProjects\Server_Client_POP3\Compte-rendu\CR.docx#_Toc478401666)

[Figure 7 : Définition de l'adresse IP du serveur 5](file:///C:\Users\jeremy\IdeaProjects\Server_Client_POP3\Compte-rendu\CR.docx#_Toc478401667)

[Figure 8 : Connexion TCP établie avec le serveur, lancement de la commande pour s'authentifier 5](file:///C:\Users\jeremy\IdeaProjects\Server_Client_POP3\Compte-rendu\CR.docx#_Toc478401668)

[Figure 9 : Exemple d'échange du point de vue du serveur 6](file:///C:\Users\jeremy\IdeaProjects\Server_Client_POP3\Compte-rendu\CR.docx#_Toc478401669)

[Figure 10 : Exemple d'échange du point de vue du client 6](file:///C:\Users\jeremy\IdeaProjects\Server_Client_POP3\Compte-rendu\CR.docx#_Toc478401670)

[Figure 11 : Exemple d'erreur côté client, le message demandé n'existe pas 7](file:///C:\Users\jeremy\IdeaProjects\Server_Client_POP3\Compte-rendu\CR.docx#_Toc478401671)

[Figure 12 : Exemple d'erreur côté serveur, le message demandé n'existe pas 7](file:///C:\Users\jeremy\IdeaProjects\Server_Client_POP3\Compte-rendu\CR.docx#_Toc478401672)

[Figure 13 : Initialisation du client 8](file:///C:\Users\jeremy\IdeaProjects\Server_Client_POP3\Compte-rendu\CR.docx#_Toc478401673)

[Figure 14 : Méthode d'envoi de requêtes du client 8](file:///C:\Users\jeremy\IdeaProjects\Server_Client_POP3\Compte-rendu\CR.docx#_Toc478401674)

[Figure 15 : Méthode readStream() qui permet de récupérer la réponse à une commande 9](file:///C:\Users\jeremy\IdeaProjects\Server_Client_POP3\Compte-rendu\CR.docx#_Toc478401675)

[Figure 16 : Échange non-sécurisé 9](file:///C:\Users\jeremy\IdeaProjects\Server_Client_POP3\Compte-rendu\CR.docx#_Toc478401676)

[Figure 17 : Échange sécurisé 9](file:///C:\Users\jeremy\IdeaProjects\Server_Client_POP3\Compte-rendu\CR.docx#_Toc478401677)

# Application Serveur

### Fonctionnalités

L’application serveur est très facile d’utilisation et l’interface est simple. Pour le lancer, il suffit d’appuyer sur le bouton. Le serveur passe alors en mode écoute et attend une connexion TCP de la part d’un client.

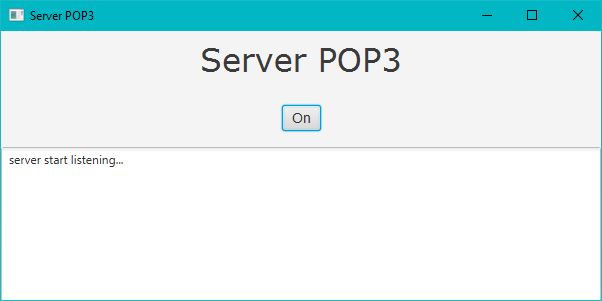


Figure : Fenêtre de lancement du serveur

Dès qu’un client veut joindre le serveur, une connexion TCP est automatiquement établie. Par la suite, le client peut envoyer différentes commandes. Cependant il doit commencer par s’identifier avec un nom d’utilisateur et un mot de passe en utilisant la commande APOP.

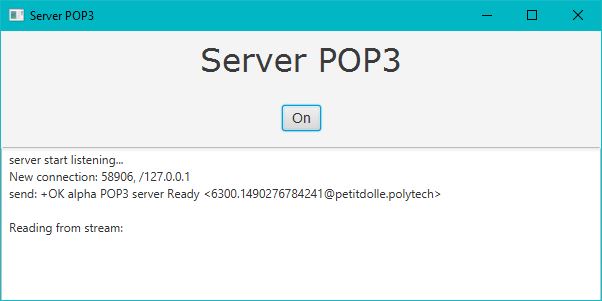


Figure : Connexion TCP établie avec un client, en attente de l’authentification

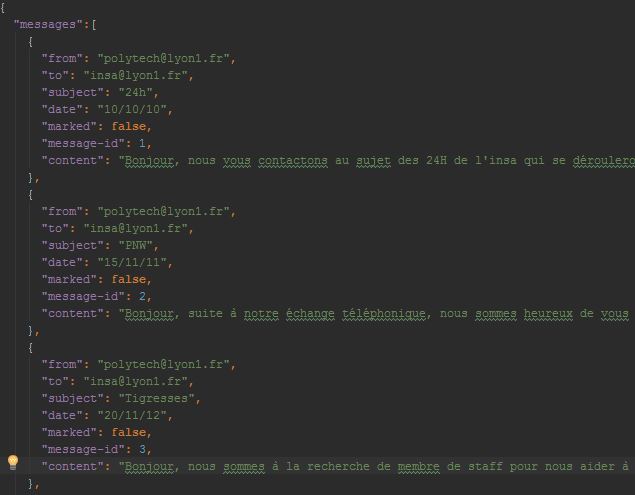
 Seuls les clients ayant un identifiant et un mot de passe enregistrés dans le fichier utilisateurs du serveur peuvent se connecter et avoir accès aux différentes fonctionnalités. Chaque utilisateur possède un fichier JSON dans lequel tous les messages qu’il a reçus sont enregistrés.

Figure : Fichier utilisateur

### Démarrage

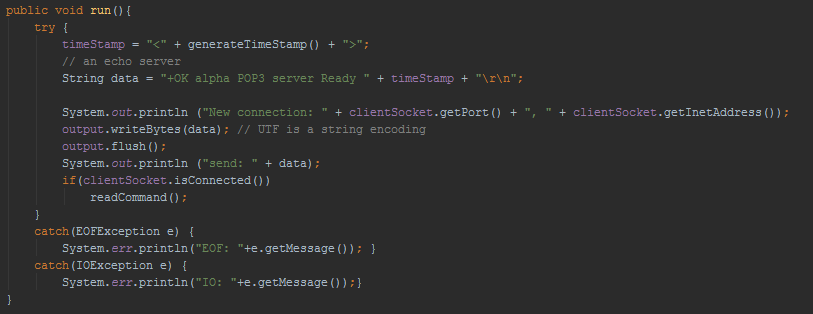
Lorsque qu’un client se connecte au serveur la fonction run() est lancée. Dans un premier temps, le serveur génère son message de bienvenu et l’envoi au client. Après cela, si le client est connecté, on attend un réponse de sa part grâce à la méthode readCommand(). Qui va lire la ligne envoyé pas le client et lancer une fonction answerCommand() qui va se charger de répondre en fonction de la commande envoyée.

Figure : Fonction run() lancée après connexion d'un client

### Gestion de la commande reçue

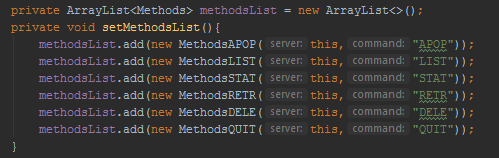
Nous avons fait le choix de créer une liste contenant la liste des différentes méthodes connues par notre Serveur. Chaque méthode est associée à une Classe qui contient les méthodes associées à la commande.

Figure : Liste contenant les commandes connues du serveur

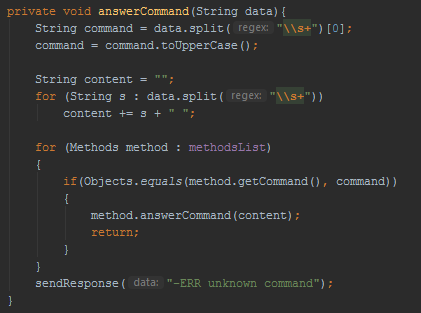
De cette manière, nous avons pu nous affranchir d’un switch case en optant pour un code plus générique en récupérant le premier mot de la ligne envoyée par le client qui est en fait la commande. Une fois cette commande récupérée il suffis de la mettre en majuscule, de parcourir la liste des commandes connues par le serveur et si la commande est la même que celle précédemment récupérée. Si cette commande est connue on lance la méthode answerCommand().contenue dans chacune des classes des commandes. Si la commande est inconnue Le serveur le fait savoir à son client en lui envoyant : « -ERR unknow command ».

Figure : Méthode answerCommand() qui gère la redirection de la fonction answerCommand()

# Application Client

### Fonctionnalités

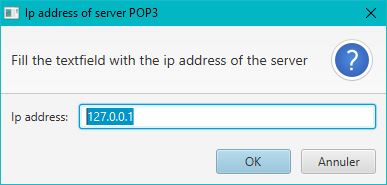
 Lors du lancement de l’application, le client commence par renseigner l’adresse IP du serveur qu’il souhaite joindre, le port étant renseigné directement par l’application. Suite à cela, l’application cliente se connecte au serveur via une connexion TCP.

Figure : Définition de l'adresse IP du serveur

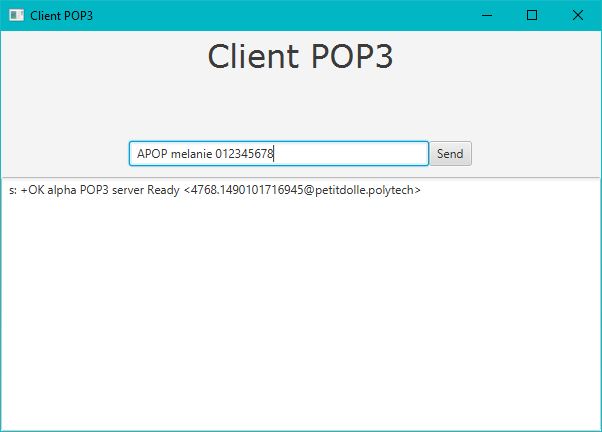
 Le client doit ensuite se connecter en utilisant la commande APOP et en renseignant son identifiant et son mot de passe. L’utilisation de toutes autres commandes sera refusée par le serveur et retournera une erreur. De plus, si le client échoue plus de trois fois à se connecter, la connexion TCP est coupé par le serveur et le client doit relancer l’application pour retenter de se connecter.

Figure : Connexion TCP établie avec le serveur, lancement de la commande pour s'authentifier

Une fois connecté, le client a à sa disposition plusieurs commandes afin d’obtenir des informations sur ses mails :

* Supprimer (DELE) + numéro d’un message : permet de marquer un message qui sera supprimer lors de la déconnexion ;
* Lister (LIST) : renvoie le nombre de messages avec la taille totale et une liste avec le numéro de chaque message (non-marqué pour être supprimé) et sa taille en octet ;
* Quitter (QUIT) : supprime les messages marqués puis déconnecte le client et ferme la connexion TCP ;
* Lecture (RETR + numéro d’un message) : renvoie le détail du message ciblé avec l’expéditeur, le receveur, le sujet, la date et le corps du message ;
* Statistique (STAT) : renvoie le nombre de message sur la boite mail ainsi que l’espace occupé en octet.

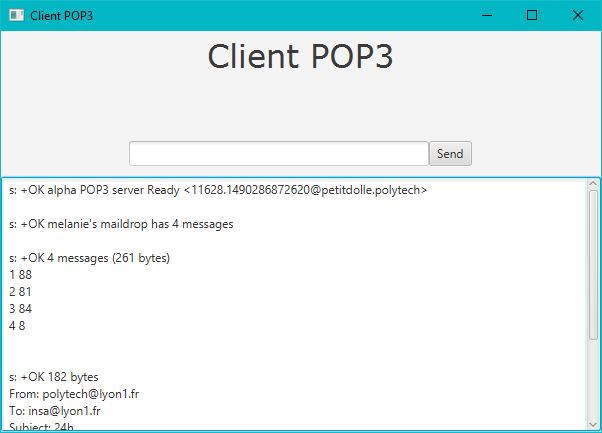


Figure : Exemple d'échange du point de vue du serveur

Figure : Exemple d'échange du point de vue du client

### Erreurs possibles

Le client peut rencontrer différentes erreurs durant l’utilisation de sa boite mail. Les erreurs lui seront retournés et explicités. Voici les cas les plus fréquents.

**Echec de connexion :**

Au bout de trois essais infructueux de tentative de connexion, le client est déconnecté. Les échecs des tentatives peuvent être dues à une erreur dans l’écriture de l’identifiant ou du mot de passe.

**Numéro de message invalide :**

Les commandes *Supprimer* et *Lecture* oblige le client à préciser le message auquel il souhaite les appliquer. Si le client ne renseigne pas le numéro d’un message ou en renseigne un qui n’existe pas, une erreur lui sera renvoyé.

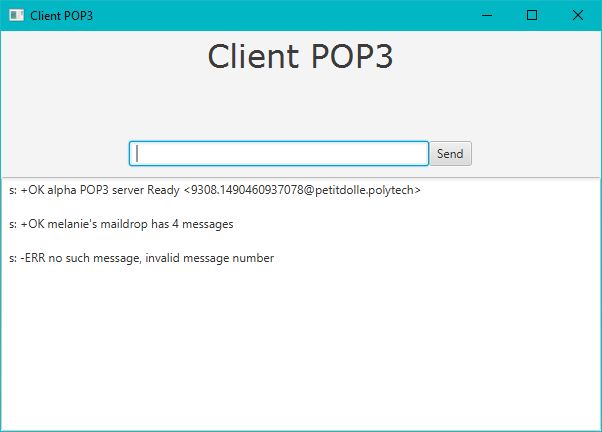


Figure : Exemple d'erreur côté client, le message demandé n'existe pas

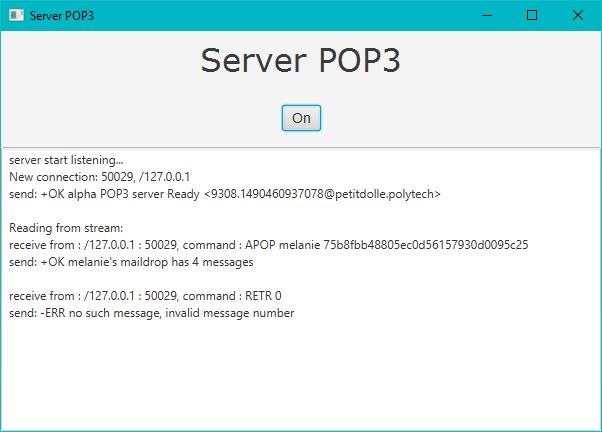


Figure : Exemple d'erreur côté serveur, le message demandé n'existe pas

### Initialisation

Lors de la création d’un client, nous initialisons notre socket ainsi que nos flux. Ensuite, grâce à la méthode gettimeStamp() le client va attendre la message de bienvenue et stocker le timestamp.

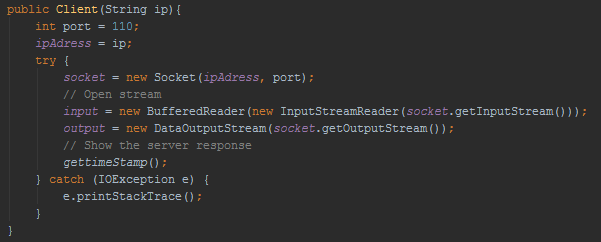


Figure : Initialisation du client

### Envoi d’une requête

Lorsque le client souhaite exécuter une commande, la méthode sendRequest(COMMANDE) est lancée. Si la commande est APOP on encode le mot de passe en MD5 avec le timeStamp récupéré juste avant pour envoyer le tout au serveur. Sinon on envoie simplement la commande saisie par le client. Pour ensuite passer dans un état d’écoute grâce aux méthodes readStream() et readStreamRetr() qui sont les même fonction que pour le serveur : le but est d’attendre une réponse terminée par **\r\n**.

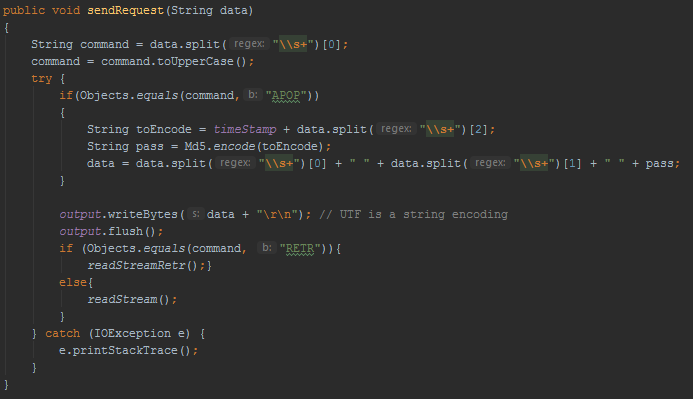


Figure : Méthode d'envoi de requêtes du client

### Lire la réponse

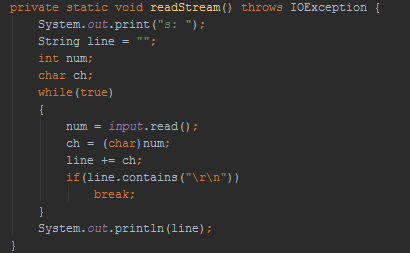
Ici, on lit caractère par caractère jusqu’à ce que la ligne contienne \r\n et on l’affiche une fois ceci terminé

Figure : Méthode readStream() qui permet de récupérer la réponse à une commande

# Connexion sécurisée

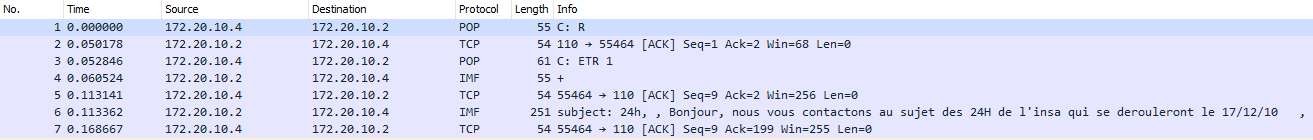
 La connexion entre le client et le serveur est sécurisé en utilisant le protocole POP3S. Alors, le mot de passe de l’utilisateur est chiffré lors de son envoie au serveur, tout comme les mails en provenance du serveur. Ainsi, un éventuel pirate ne peut pas intercepter les mots de passes pour usurper l’identité d’un utilisateur ou lire les messages qui transitent.

Figure : Échange non-sécurisé

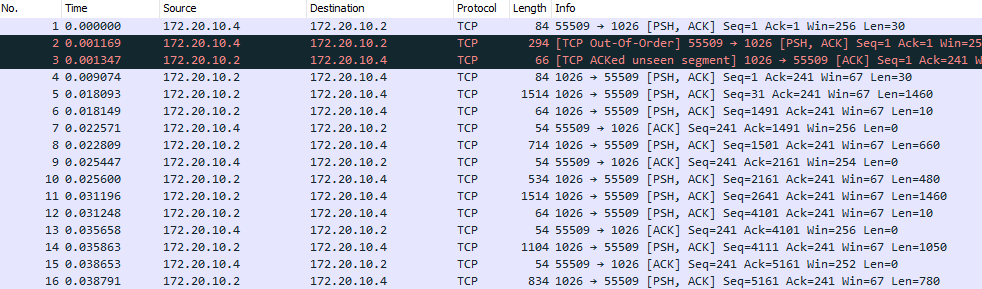


Figure : Échange sécurisé