

Rapport laboratoire 4 - SMTP

Auteurs : Anthony David, Stéphane Nascimento Santos

Date : 08.12.2022

1. Introduction

Ce rapport concerne le laboratoire 4 du cours de DAI 2022-2023 de l'HEIG-VD.

Le but général de ce laboratoire est le développement d'une application client TCP en Java qui utilise l'API Socket afin de communiquer avec un serveur SMTP.

Plus précisément, les objectifs sont les suivants :

1. Faire des expérience pratique pour se familiariser avec le protocole SMTP.
2. Comprendre les notions de test double et de serveur mock (serveur fictif).
3. Comprendre le protocole SMTP et être capable d'envoyer des messages électroniques en utilisant en travaillant directement avec l'API Socket.
4. Découvrir la simplicité d'envoi de faux e-mails qui semble avoir été envoyé par une personne autre qu'une personne malveillante.
5. Concevoir une application client orienté objet pour mettre en oeuvre une liste d'exigences spécifiques.

2. Description du projet

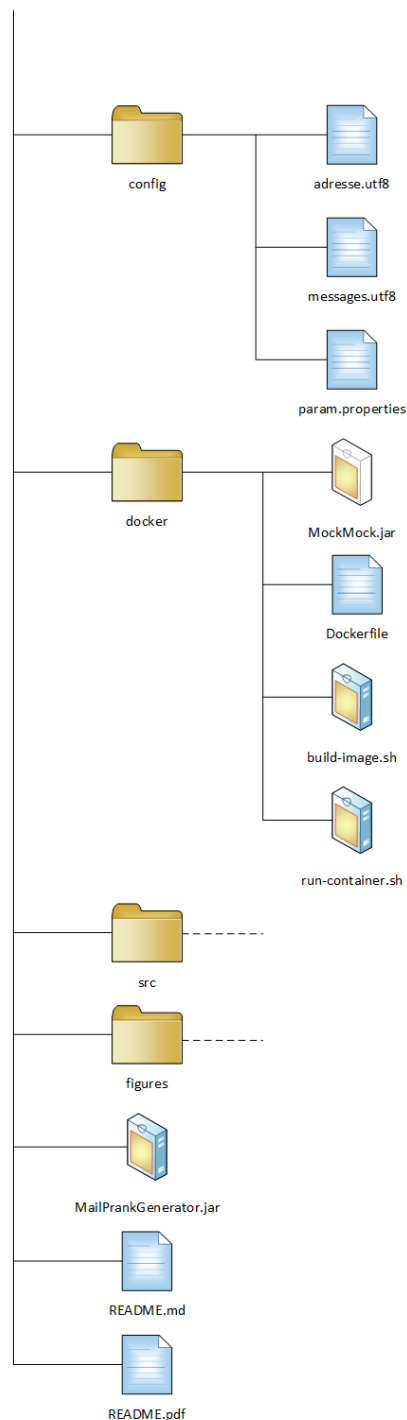
L'entier du projet se trouve sur GitHub à l'adresse suivante :

<https://github.com/AnthonyDavid3110/DAI-2022-SMTP>

Afin de pouvoir les différents outils développés dans le cadre de ce laboratoire, il suffit de forker le repository GitHub susmentionné et de le cloner sur la machine local. Une procédure décrivant ces démarches est disponible dans la documentation de GitHub :

<https://docs.github.com/fr/get-started/quickstart/fork-a-repo>

Une fois le fork et le clone effectué, on trouve sur notre machine un dossier avec l'arborescence suivante:



Les fichiers README.md et README.pdf comprennent le présent rapport respectivement en format Markdown et PDF.

Le fichier pom.xml est le fichier de configuration de Maven. Il est nécessaire au bon fonctionnement de l'application mais il n'est pas nécessaire de s'en préoccuper pour utiliser paramétrer et utiliser l'application.

Le contenu des différents dossiers est exliqué ci-dessous :

2.1 Dossier config

Il contient les différents fichiers de configuration de notre application :

- `messages.utf8` qui contient des mails malicieux.
- `adresses.utf8` qui contient les adresse des personnes à piéger
- `param.properties` qui contient les paramètres de configuration

La modification de la configuration de notre application via la modification des ces fichiers est expliquée au [Point 4.2](#) de ce document.

2.2 Dossier docker

Le dossier `docker` contient les différents fichiers pour faire fonctionner un serveur fictif SMTP "MockMockServer" dans un container docker :

To do : ajouter le schéma de l'arborescence

- `Dockerfile` qui est le fichier de configuration du container docker
- `MockMock-1.4.1-SNAPSHOT-one-jar.jar` qui est l'exécutable java de l'application "MockMockServer".
- `build-image.sh` qui est un script pour la construction de l'image du container docker.
- `run-container.sh` qui est un script permettant de démarrer le container sur docker.

Une procédure d'utilisation complète du serveur fictif SMTP est disponible au [Point 3.3](#) de ce document.

3.3. Exécuter MockMockServer

2.3 Dossier src

Le dossier `src` contient tout le code source de l'application "MailPrankGenerator" développé en Java. Il contient différents sous-dossiers représentant les différents packages de l'application.

Une description précise de l'implémentation du programme est faite au [Point 5](#) de ce document.

2.4 Dossier figures

Le dossier `figures` contient les différents médias (image, diagramme de classe, etc...) présent dans ce document. Le contenu n'est donc significatif à l'utilisation de l'application "PrankMailGenerator" ou au serveur "MockMockServer".

3. MockMock Server

3.1 Qu'est que MockMockServer

MockMockServer est une application multi-plateforme simulant un serveur SMTP développé en Java. L'application permet de tester si des e-mails sortants sont bien envoyés et de voir à quoi ils ressemblent. Il fournit une interface web qui affiche les courriels qui ont été envoyés et vous montre le contenu de ces courriels.

3.2 Pourquoi utiliser un serveur mock

Tester le fonctionnement d'une application sur un véritable serveur SMTP a plusieurs désavantages :

- Concrètement on a pas besoin de recevoir les mails que notre application envoie sur un MUA (Mail User Agent) car ce qu'on va vouloir tester est uniquement l'envoi de nos mails.
- Il peut avoir un délai entre l'envoi et la réception du mail ce qui peut rallonger considérablement le temps de développement de notre application.
- A force d'envoyer bon nombre de mails sur un serveur, il y a des risques que le serveur ne nous permette pas l'envoi de mail pour des raisons de sécurité (black list par exemple).

Du faites des différents inconvénients exposés ci-dessus, il est donc préférable d'utiliser un serveur fictif de type Mock (signifiant simulé en français).

3.3. Exécuter MockMockServer

Le répôt GitHub de l'application MockMockServer est disponible à l'adresse suivante :

<https://github.com/DominiqueComte/MockMock>

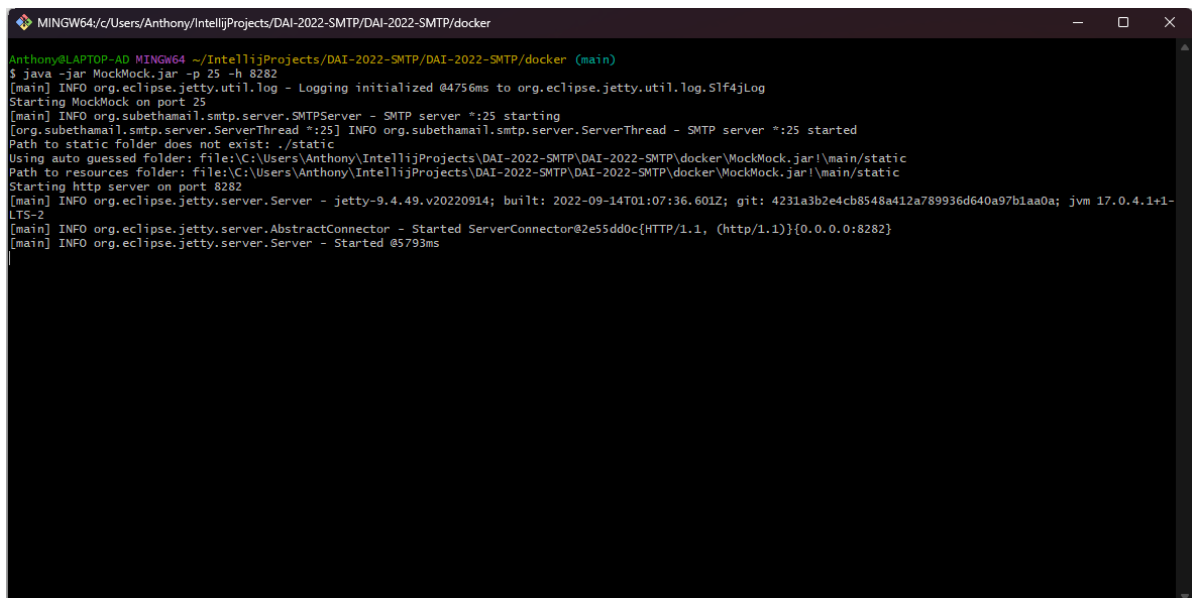
Ce qui permet d'aller voir comment est implémenté le programme. Cependant, un exécutable `.jar` est disonible dans le fork du projet décrit au point 2 de ce document dans le dossier `docker`.

Pour lancer l'application, il suffit d'effectuer la commande suivante dans le dossier docker :

```
$ java -jar MockMock.jar -p 25 -h 8282
```

Dans ce cas, le serveur SMTP aura comme port le 25 et l'application utilisera le port 8282 comme port http pour pouvoir se connecter sur l'interface web.

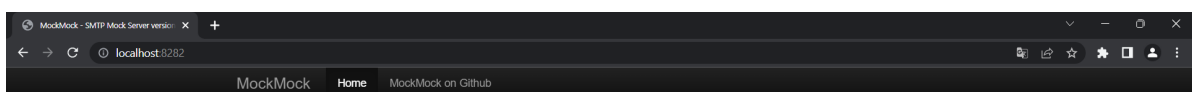
Si le serveur a bien démarré, le contenu de l'invité de commande devrait être similaire à la capture suivante :

A terminal window titled 'MINGW64/c/Users/Anthony/IntelliJProjects/DAI-2022-SMTP/DAI-2022-SMTP/docker' shows the execution of 'java -jar MockMock.jar -p 25 -h 8282'. The output includes logs from org.eclipse.jetty, org.subethamail.smtp.server, and org.eclipse.jetty.server, indicating that the SMTP server is running on port 25 and the HTTP server is running on port 8282. The terminal text is as follows:

```
Anthony@LAPTOP-AD MINGW64 ~/IntelliJProjects/DAI-2022-SMTP/DAI-2022-SMTP/docker (main)
$ java -jar MockMock.jar -p 25 -h 8282
[main] INFO org.eclipse.jetty.util.log - Logging initialized @4756ms to org.eclipse.jetty.util.log.Slf4jLog
Starting MockMock on port 25
[main] INFO org.subethamail.smtp.server.SMTPServer - SMTP server *:25 starting
[org.subethamail.smtp.server.ServerThread *:25] INFO org.subethamail.smtp.server.ServerThread - SMTP server *:25 started
Path to static folder does not exist: ./static
Using auto guessed folder: file:C:\Users\Anthony\IntelliJProjects\DAI-2022-SMTP\DAI-2022-SMTP\docker\MockMock.jar\main\static
Path to resources folder: file:C:\Users\Anthony\IntelliJProjects\DAI-2022-SMTP\DAI-2022-SMTP\docker\MockMock.jar\main\static
Starting http server on port 8282
[main] INFO org.eclipse.jetty.server.Server - jetty-9.4.49.v20220914; built: 2022-09-14T01:07:36.601Z; git: 4231a3b2e4cb8548a412a789936d640a97b1aa0a; jvm 17.0.4.1+1-LTS-2
[main] INFO org.eclipse.jetty.server.AbstractConnector - Started ServerConnector@2e55dd0c{HTTP/1.1, (http/1.1)}{0.0.0.0:8282}
[main] INFO org.eclipse.jetty.server.Server - Started @5793ms
```

Pour se rendre sur l'interface web, l'adresse du serveur sera localhost (comme l'application tourne sur la machine) et le port 8282 comme mentionné ci-dessus :

`localhost:8282`



No emails in queue

A noter que lors d'envoi de mail sur MockMockServer, en plus d'être visible sur l'interface web, il y a également une indication dans l'invité de commande où a été lancé le programme avec l'ajout d'une ligne tel que celle-ci :

```
Email from joseph.lefebvre@heig-vd.ch received.
```

3.4 Exécuter MockMockServer sur un container Docker

Il est également possible et fortement conseillé de faire fonctionner l'application dans un container docker.

Docker doit être installé et fonctionner sur la machine hôte. Si ce n'est pas le cas, une procédure détaillée est disponible ici :

<https://docs.docker.com/get-started/>

Des scripts sont à dispositions afin de faciliter la création de l'image ainsi que son déploiement. Par défaut, les ports utilisés sont les suivants :

SMTP = 25

HTTP = 8282

Pour modifier les ports, il faut modifier le scripte run-container.sh et remplacer les ports susmentionnés par ceux que l'on désire.

1. Lancer la construction de l'image :

```
./build-image.sh
```

Après l'exécution de la commande, l'invité de commande devrait être dans l'état suivant :

```
MINGW64/c:/Users/Anthony/IntelliJProjects/DAI-2022-SMTP/DAI-2022-SMTP/docker
Anthony@LAPTOP-AD MINGW64 ~/IntelliJProjects/DAI-2022-SMTP/DAI-2022-SMTP/docker
(main)
$ ./build-image.sh
#1 [internal] load build definition from Dockerfile
#1 sha256:1c9b2d36a306b560782607f2d6f80ffe1006b0d8723eb002e0eba5e88621e445
#1 transferring dockerfile: 180B 0.0s done
#1 DONE 0.1s

#2 [internal] load .dockerignore
#2 sha256:ef44d7566500c44e8fbd6a6adde7d341a5fbc29c7aa607161964577e2e73f0e
#2 transferring context: 2B done
#2 DONE 0.1s

#3 [internal] load metadata for docker.io/library/openjdk:11
#3 sha256:87dc9b3cee4adf6787fd792601b37fcaad0ed6bd5314a02f15c26446f91634ad
#3 DONE 1.5s

#4 [1/2] FROM docker.io/library/openjdk:11@sha256:99bac5bf83633e3c7399aed725c8415e7b569b54e03e4599e580fc9cdb7c21ab
#4 sha256:8c2fff55dee4aa503dfc85309a0c6a50b5d76e4b584cef86f8babc8f37841c8e
#4 CACHED

#5 [internal] load build context
#5 sha256:981805e1c14d85cf4351e1960c534afa51a1859bc7928c9fb1a44a711222c510
#5 transferring context: 13.09MB 0.1s done
#5 DONE 0.1s

#6 [2/2] ADD MockMock.jar MockMock.jar
#6 sha256:34c64ec0774a67ba6967c62f0036514d3d08708bc37f88f3ac7a8b2365a28d30
#6 DONE 0.1s

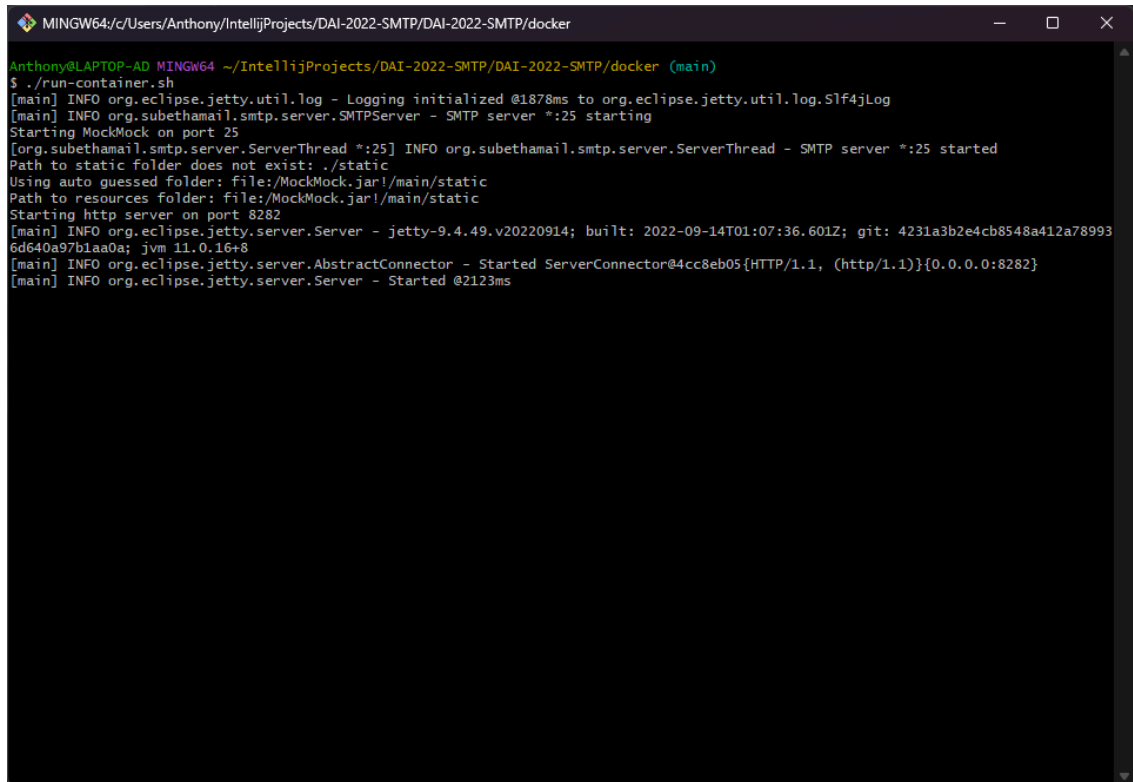
#7 exporting to image
#7 sha256:e8c613e07b0b7ff33893b694f7759a10d42e180f2b4dc349fb57dc6b71dcab00
#7 exporting layers 0.1s done
#7 writing image sha256:b2a1fb3b66e9e96cfff5977ac56eebd01e5aa4b57e0a55f8f0c85d568f12c4e5d done
#7 naming to docker.io/library/mockmockserver done
#7 DONE 0.1s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
Anthony@LAPTOP-AD MINGW64 ~/IntelliJProjects/DAI-2022-SMTP/DAI-2022-SMTP/docker (main)
$
```

2. Lancer le déploiement du container sur Docker:

```
./run-container.sh
```

Après l'exécution, notre invité de commande doit être dans l'état suivant :



```
MINGW64/c/Users/Anthony/IntelliJProjects/DAI-2022-SMTP/DAI-2022-SMTP/docker
Anthony@LAPTOP-AD MINGW64 ~/IntelliJProjects/DAI-2022-SMTP/DAI-2022-SMTP/docker (main)
$ ./run-container.sh
[main] INFO org.eclipse.jetty.util.log - Logging initialized @1878ms to org.eclipse.jetty.util.log.Slf4jLog
[main] INFO org.subethamail.smtp.server.SMTPServer - SMTP server *:25 starting
Starting MockMock on port 25
[org.subethamail.smtp.server.ServerThread *:25] INFO org.subethamail.smtp.server.ServerThread - SMTP server *:25 started
Path to static folder does not exist: ./static
Using auto guessed folder: file:/MockMock.jar!/main/static
Path to resources folder: file:/MockMock.jar!/main/static
Starting http server on port 8282
[main] INFO org.eclipse.jetty.server.Server - jetty-9.4.49.v20220914; built: 2022-09-14T01:07:36.601Z; git: 4231a3b2e4cb8548a412a78993
6d640a97b1aa0a; jvm 11.0.16+8
[main] INFO org.eclipse.jetty.server.AbstractConnector - Started ServerConnector@4cc8eb05{HTTP/1.1, (http/1.1)}{0.0.0.0:8282}
[main] INFO org.eclipse.jetty.server.Server - Started @2123ms
```

A noter qu'une fois l'application en fonctionnement, le comportement est le même qu'au point [3.3](#). C'est-à-dire que lors de chaque mail reçu par le serveur Mock, une ligne sera ajoutée dans l'invité de commande et que l'interface web est disponible via localhost:port.

4. Mail Prank Generator

4.1 L'application Mail Prank Generator

L'application Mail Prank Generator va permettre d'envoyer des emails canulars à une liste de contact prédéfinie.

Pour se faire, nous allons utiliser

4.2 Paramétrage de l'application

Dans le dossier `param` se trouvant à la racine projet, se trouve 3 fichiers de configuration. Par défaut il sont configuration afin de pouvoir utiliser l'application sans les modifier mais il peuvent être adapté au besoin de l'utilisateur.

4.2.1 Configuration des messages à envoyer

Le fichier `param.properties` contient les paramètres de fonctionnement de l'application, soit :

- L'adresse de la machine `host=`

Par défaut `localhost` car prêt pour une utilisation avec [MockMockServer](#).

- Le port SMTP du serveur `port=`

Par défaut configuré sur `25` car prêt pour une utilisation avec [MockMockServer](#) configuré comme ci-dessus.

- Le nombre de groupe de victims `nbggroups=`

Par défaut configuré à `3` groupes de victimes

Les valeurs contenues dans le fichiers peuvent être modifié avant chaque lancement de l'application.

4.2.2 Modifier de liste des victime

Le fichier `adresses.utf8` contient la listes des victimes et des faux expéditeurs de mail.

L'application est configuré pour créer automatiquement le nombre de groupe de victimes configuré dans le fichier `param.properties`

A noter que le nombre minimum d'adresses par groupe est de 3 au minimum (soit une adresse qui sera l'expéditeur et au minium 2 adresse qui recevront le mail). Si le nombre d'adresse n'est pas suffisant, l'application en fonctionnera pas.

A noter également que l'application ajoute automatiquement une signature au mail avec le prénom et le nom de l'expéditeur (le même que l'adresse d'expédition) si, et uniquement si, elle arrive à déterminer le prénom et le nom depuis l'adresse e-mail. L'application peut le faire si l'adresse mail de l'expéditeur est sous le format `prenom.nom@...`. Dans le cas contraire, mail sera envoyé sans signature.

4.2.3 Modifier la liste des victimes

Dernier fichier de configuration, `messages.utf8` contient le sujet et le corps d'e-mails fictifs utilisé par l'application pour envoyer les canulars.

Il est donc possible de modifier / ajouter des e-mails dans ce dossier.

Cependant, il est nécessaire de respcéter les consignes suivantes :

- Chaque mail doit être séparé de la suite de caractère `==`.
- Chaque mail dans commencer par le sujet sous le formt `subject: {sujet du mail}`.
- Une ligne vide doit être laissée entre le sujet et le corps du mail.
- Les mails ne doivent pas être signées au risque de l'être doublement si l'application arriver à déterminer un prénom et un nom à partir de l'adresse mail de l'expéditeur du canular.

4.3 utilisation de l'application

Une fois les différents fichiers de configurations édités aux besoin, on peut trouver un fichier exécutable `MailPrankGenerator.jar` la racine du projet.

Etant donné que l'application retourne dans le terminal un état des lieux des différentes interractions faites avec les serveur, il est fortement recommandé de lancer l'application depuis un terminal afin de pouvoir s'assurer du bon fonctionnement de l'application.

Il suffit donc de lancer l'application avec la commande :

```
java -jar MailPrankGenerator.jar
```

```
MINGW64/c/Users/Anthony/IntelliJProjects/DAI-2022-SMTP/DAI-2022-SMTP
Anthony@LAPTOP-AD MINGW64 ~/IntelliJProjects/DAI-2022-SMTP/DAI-2022-SMTP (main)
$ java -jar MailPrankGenerator.jar
INFO: Beginning of the program.
INFO: Sending mails pranks
INFO: Sending message via SMTP
INFO: 220 23b06c98098b ESMTP MockMock SMTP Server version 1.4
INFO: 250-23b06c98098b
INFO: 250-8BITIME
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 354 End data with <CR><LF>.<CR><LF>
INFO: 250 Ok
INFO: Sending message via SMTP
INFO: 220 23b06c98098b ESMTP MockMock SMTP Server version 1.4
INFO: 250-23b06c98098b
INFO: 250-8BITIME
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 354 End data with <CR><LF>.<CR><LF>
INFO: 250 Ok
INFO: Sending message via SMTP
INFO: 220 23b06c98098b ESMTP MockMock SMTP Server version 1.4
INFO: 250-23b06c98098b
INFO: 250-8BITIME
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 250 Ok
INFO: 354 End data with <CR><LF>.<CR><LF>
INFO: 250 Ok
INFO: End of the program.
Anthony@LAPTOP-AD MINGW64 ~/IntelliJProjects/DAI-2022-SMTP/DAI-2022-SMTP (main)
$
```

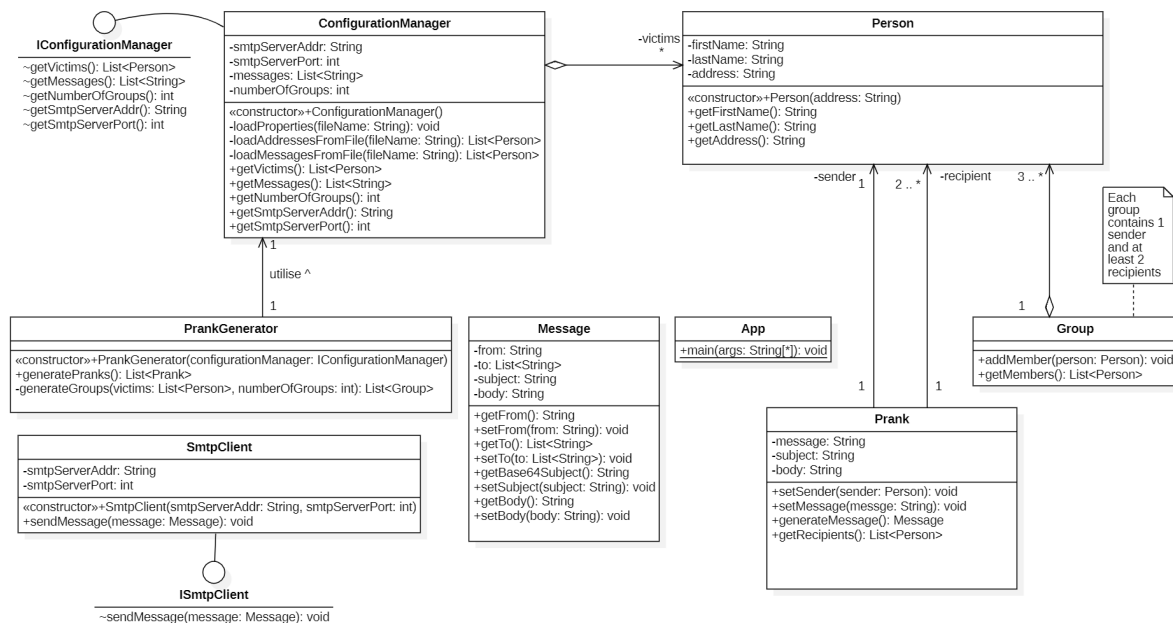
Si on reçoit en retour uniquement les `INFO: { ... }` et que le programme termine par la ligne :

```
INFO: End of the program.
```

C'est que tout s'est passé comme escompté.

5. Description de l'implémentation

5.1 Diagramme de classe



5. 2 Explications sur l'implémentation

On peut découper notre application en 3 grandes parties que sont :

- La gestion des paramètres
- La modélisation des canulars et des informations nécessaires
- L'envoi des canulars via SMTP

Nous avons opté pour une solution où chacun de ces parties était intégré à un package afin de structurer notre de manière optimale et de permettre la réutilisation d'une partie de celui-ci. On pourrait par exemple imaginer réutiliser la gestion des paramètres dans une autre application qui enverrait autre chose que des pranks.

A noter que la conception de l'application se base grandement sur les vidéos du cours de DAI :

<https://www.youtube.com/watch?v=ot-bDyggTtk>

<https://www.youtube.com/watch?v=Nj34XicS6Jm>

<https://www.youtube.com/watch?v=LoFKsT9Rj10>

https://www.youtube.com/watch?v=OrSdRCt_6YQ

La structure de notre programme en est grandement inspiré et des informations complémentaires sur l'implémentation du client peuvent être trouvées sur les vidéos si elle serait manquante ou insuffisamment précise dans ce présent document.

5.2.1 Gestion des paramètres

Comme expliqué précédemment dans ce document, les propriétés, les mails à envoyer ainsi que les adresses sont stockés dans 3 fichiers de configuration qu'il est possible de modifier sans avoir à le faire dans le code Java de l'application.

Afin d'aller lire les paramètres stockés dans ces fichiers, nous avons implémenté un package `config` implémentant une Classe `ConfigurationManager` Permettant d'aller lire et de charger les informations dans les fichiers de configuration.

A noter l'implémentation d'un contrôle de la bonne forme des adresse mail faisant stopper le programme si une adresse mail n'est pas dans un format adéquat.

5.2.2 Modélisation

Pour ce qui est de la médilisation, nous avons décidé de séparer encore en dous sous-package :

mail

Comprenant une classe `Groupe`, une classe `Message`, et une classe `Person` modélisant respectivement un objet du type dont il portent le nom.

A noter l'implémentation d'un getter supplémentaire sur le Subject retournant ce dernier encodé en Base64. Il est nécessaire d'avoir cette encodage du sujet du mail lors de l'envoi.

A noter que lors de la création d'une personne, si le mail est de format `prénom.nom@...` il va récupérer le prénom et le nom de la personne. ce qui lui permettra lors de la création du prank d'ajouter une signature au mail avec le même prénom et le même nom que dans l'adresse mail, ce qui augemente la crédibilité de notre prank.

Prank

Modélisant le canular en lui-même et offrant les méthodes nécessaire à la création de ces derniers.

5.2.3 Partie SMTP

Interface et classe permettant l'envoi de nos pranks via l'API Socket au serveur SMTP.

5.2.5. App

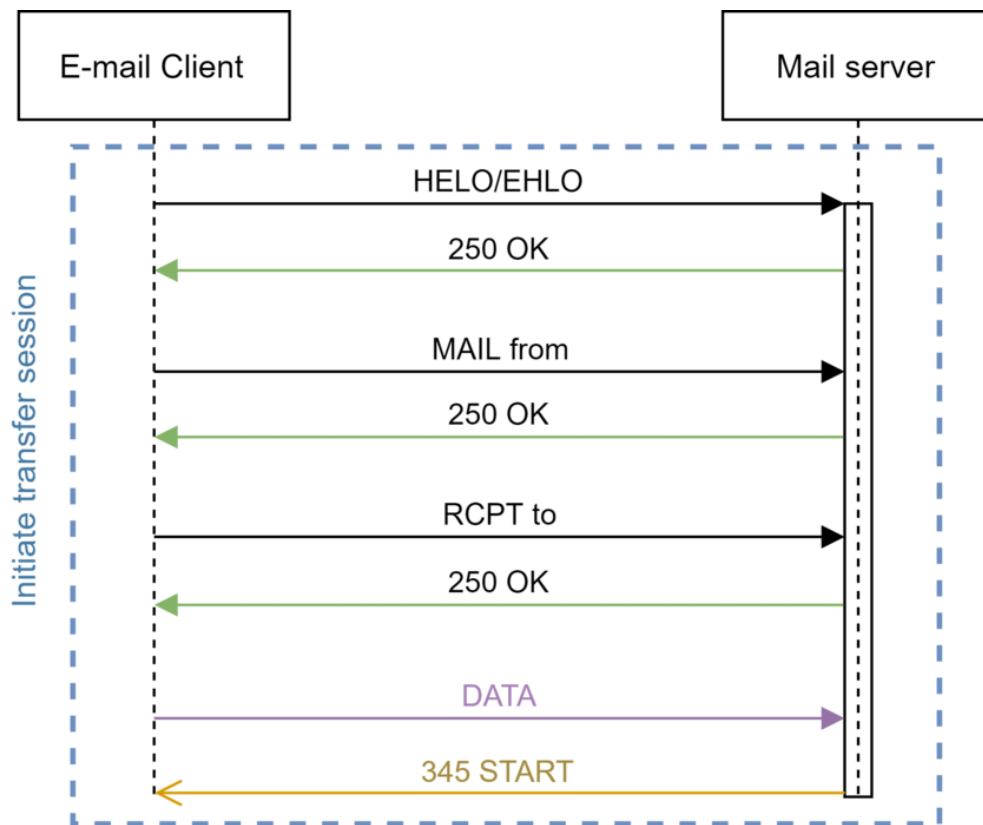
Finalement, nous avons le fichier App.java qui contient la fonctions `main()` principale du programme.

5.3 Communication avec le serveur

Comme évoqué ci-dessus, le client utilise le procole TCP et l'API Socket afin de communiquer avec le serveur SMTP.

La communication avec un serveur SMTP fonctionne de manière relativement simple puisqu'il va répondre à une suite de commande précise que le client va lui envoyer.

Schématiquement la conversation peut vue comme ça :



Les commandes utilisées dans l'application sont les suivantes :

- `EHLO` permet au client de se présenter au serveur
- `MAIL FROM` permet au client d'indiquer l'adresse de l'expéditeur
- `RCPT TO` permet au client d'indiquer l'adresse du/des récepteurs du mail
- `DATA` permet d'envoyer respectivement les en-têtes et le corps du mail

Le serveur répond à chaque fois par un code afin que le client puisse s'assurer que tout s'est bien passé (par exemple `250 OK` si tout est bon pour le serveur).

5.3.1 Exemple de communication client-serveur SMTP

Indications :

- `s` pour server
- `c` pour client
- Communication entre l'application `MailPrankGenerator` et un serveur `MockMockServer` configuré sur le port=25 démarré sur la machine local `host=localhost`.

```
S: 220 23b06c98098b ESMTTP MockMock SMTP Server version 1.4
C: EHLO localhost
S: 250-23b06c98098b
S: 250-8BITMIME
S: 250 OK
C: MAIL FROM: Teresa.Martin@hotmail.com
S: 250 OK
S: RCPT TO: John.Benjamin@yahoo.com
S: 250 OK
S: RCPT TO: Susan.Stevens@gmail.com
S: 250 OK
S: RCPT TO: Debbie.Stewart@yahoo.com
```

```
S: 250 OK
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: Content-Type: text/plain; charset=utf-8
C: From: Teresa.Martin@hotmail.com
C: To: John.Benjamin@yahoo.com
C: , Susan.Stevens@gmail.com
C: , Debbie.Stewart@yahoo.com
C: Subject: =?utf-8?B?
R8Opbm1hbCBjb25jZjZ0IGR1IEF0dW5pYWwgZ3JvdXB1ICJnw6luw6lyaXF1JiIgIQ==?=
C: salut !
    J'espère que tu vas bien !
    Je sais pas si tu as vu mais le fanfare d'Yvonand donne prochaine son concert
    annuel. Ca te dirais de venir avoir moi ?

    A+
C: .
C:
S: 250 OK
C: QUIT
S: 221 Bye
```

6. Conclusion

Après avoir effectué ce laboratoire, nous avons compris le concept de communication avec un serveur SMTP et comment implémenter une solution simple de client pouvant le faire.

Ce laboratoire nous a aussi permis de nous familiariser par la pratique avec l'environnement Docker ce qui a été fort appréciable.