



Réalisation d'un utilitaire d'archivage

Réalisé par :

FAILALI Rochdi
EZ-ZAIM Adnane
EL GRIBES Omar

Encadré par :

Prof. RAWAT PRIYANKA

Année 2022/2023

Table des matières

<i>Abstract :</i>	<u>2</u>
<i>I. Choix techniques :</i>	<u>3</u>
<i>II. Structure du projet :</i>	<u>4</u>
<i>III. Fonctionnement :</i>	<u>7</u>

Abstract :

This project consists in archiving an SQL file that the user can retrieve from a server web through a download link in order to place a tgz archive format of the same file in a remote server. In an effort to launch the system untroubledly, all dependencies have to be installed as well as an adaptable crontab file to automate the execution of the scripts. The user can personalise the configuration of the system through configuration file granting them the possibility to enable or disable the historization, modify the expiration date, and change the settings of the mailing process. The project is built with docker, written in python, and could be executed and used whether on a Windows or a Linux based machine.

I. Choix techniques :

Dans la réalisation de notre projet, on s'appuyait sur un ensemble des choix techniques qu'on peut justifier par le tableau ci – dessous :

<i>Choix technique</i>	<i>justification</i>
Docker	<ul style="list-style-type: none">- L'exécution des programmes installés ne dépend pas du OS.- Facilite la manipulation des micro-services.- Gestion travail en équipe- Facilite la coordination des comportements entre les conteneurs.
Nginx	<ul style="list-style-type: none">- En terme de vitesse, Nginx est plus performant que apache- Vu qu'on à déjà travaillé avec apache, on a favorisé Nginx.
FTPS	<ul style="list-style-type: none">- Facile à implémenter- Permet de lire ce qui est dedans.
Postgres	<ul style="list-style-type: none">- Open source- La configuration de Son image sur Docker est plus aisé.
Configuration file .YML	<ul style="list-style-type: none">- Facile à manipuler (lire, commenter, modifier...)- Compatible avec Docker- Facile d'ajouter une couche de validation

Email	<ul style="list-style-type: none"> - On a choisi un serveur SMTPS google afin de pouvoir communiquer avec toutes les boîtes possibles ce qui n'était pas réalisable avec notre serveur local.
-------	--

II. Structure du projet :

Notre projet est constitué par les dossiers et les fichiers suivants :

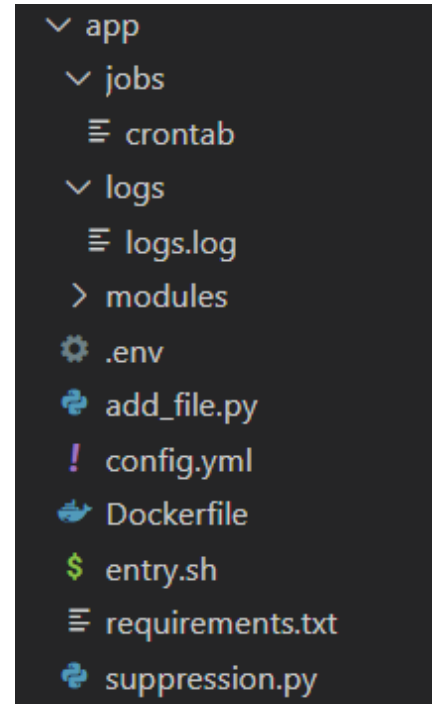
- **Docker-compose.yml** : il s'agit d'un fichier YAML qui nous permet de définir et exécuter des applications Docker à plusieurs conteneurs ainsi que la configuration des services utilisés.
- **.gitignore** : pour considérer un certain nombre des fichiers comme non tracké lorsqu'on veut commiter sur le git.
- **Nginx.conf** : pour configurer notre serveur web.
- **Nginx_doc.conf** : pour configurer notre serveur qui affiche la documentation utilisateur.
- **.env** : contient les variables d'environnement pour définir la configuration de docker.
- **Public** : affiche les fichiers contenus dans le serveur web
- **Doc** : contient les fichiers HTML de la documentation utilisateur générée.
- **Db** : la base de données qui reflète ce qui se passe au niveau du serveur distant en terme d'ajout, suppression, prolongation des fichiers.
- **Start.sh** : Pour lancer la totalité de l'application.
- **Storage** : où on stock les fichiers sur FTPS
- **App** : représente notre application, constituée par :

```

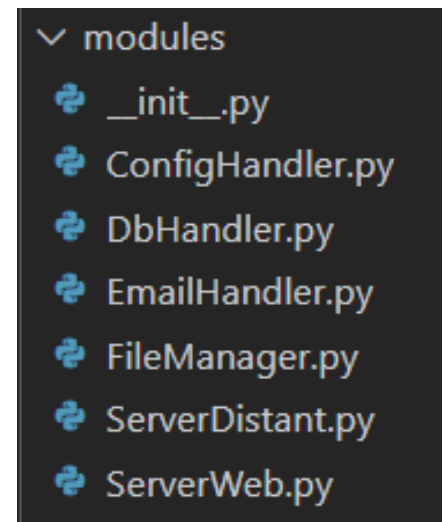
> app
> db
> doc
> public
> storage
⚙ .env
📄 .gitignore
📄 docker-compose.yml
📄 LICENSE
⚙ nginx_doc.conf
⚙ nginx.conf
📄 README.md
$ start.sh

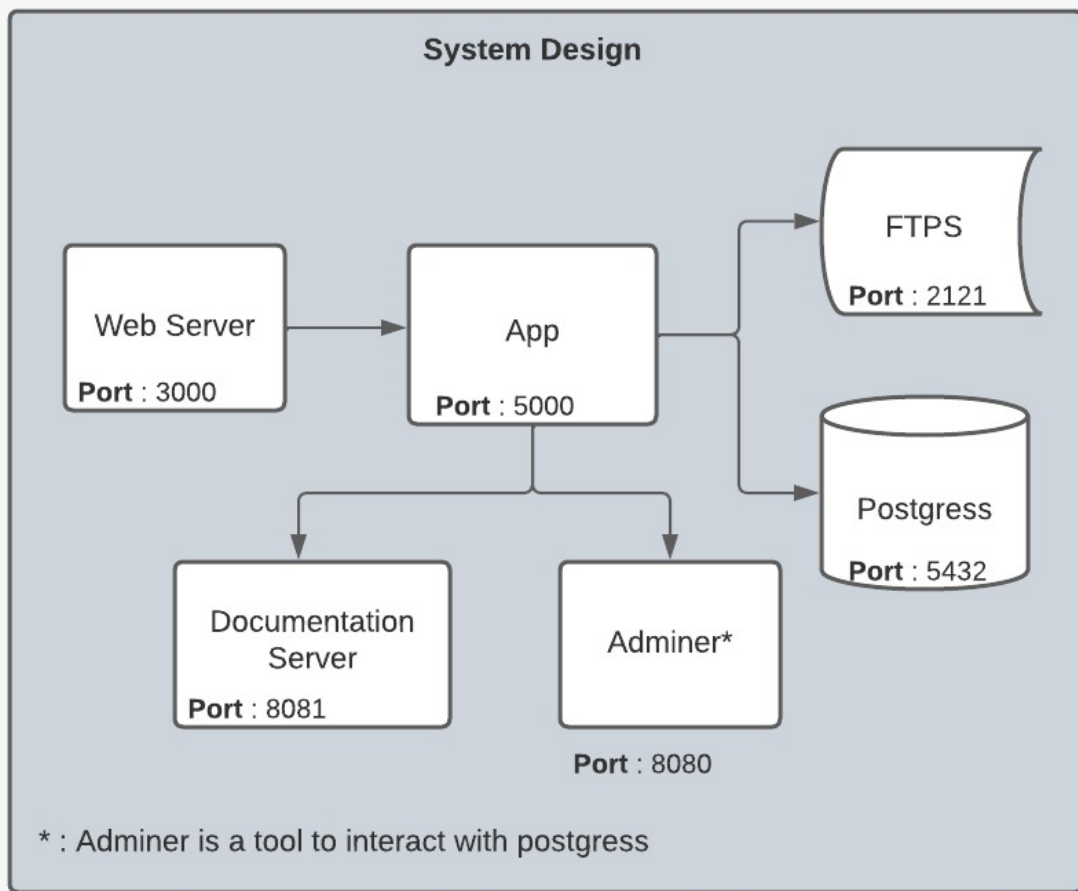
```

- **.env** : contient les variables d'environnement pour définir la configuration de l'application.
- **Config.yml** : permet à l'utilisateur de modifier l'historisation, date d'expiration, les logs et l'email.
- **Crontab** : pour automatiser le programme.
- **Dockerfile** : pour faire les installations nécessaires.
- **Entry.sh** : c'est le script qui permet le lancement du cron et la documentation à chaque fois on exécute notre application.
- **Requirements.txt** : Pour garantir qu'on travaille sur le projet en utilisant le même environnement virtuel, c'est-à-dire les paquets et les versions installées.
- **Suppression.py** : fichier python qui supprime les fichiers invalides.
- **Modules** : c'est l'ensemble des classes python qu'on a utilisé pour gérer l'application, notamment on a :



- *_init_.py : cette fonction s'exécute pour charger l'environnement, la configuration et pour les valider.*
- *ConfigHandler.py : nous permet de gérer la configuration.*
- *DbHandler.py : nous permet de gérer la base de données.*
- *Email.Handler.py : nous permet de gérer les emails.*
- *FileManager.py : nous permet de gérer les fichiers.*
- *ServerDistant.py : nous permet de gérer le serveur distant.*
- *ServerWeb.py : nous permet de gérer le serveur Web*

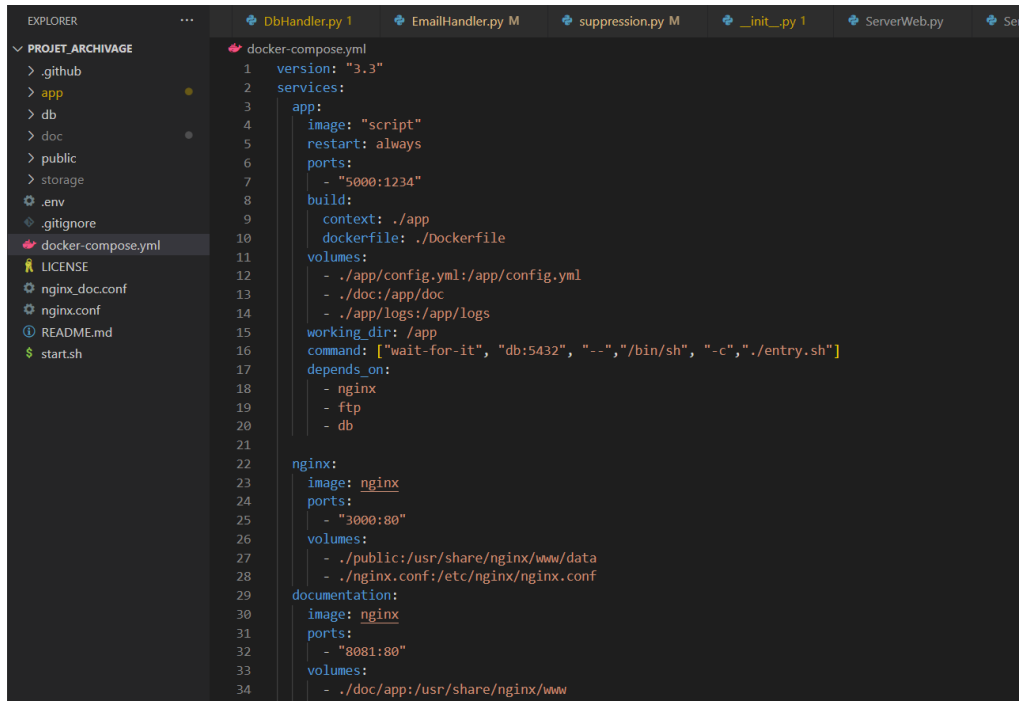




Ce diagramme montre la structure de notre dossier App, l'interaction entre les différentes entités, et les ports d'accès.

III. Fonctionnement :

Le fonctionnement de notre script est séparé par 3 étapes majeures : la construction de notre image app, le lancement des services et l'automatisation de ce processus à l'aide du cron.

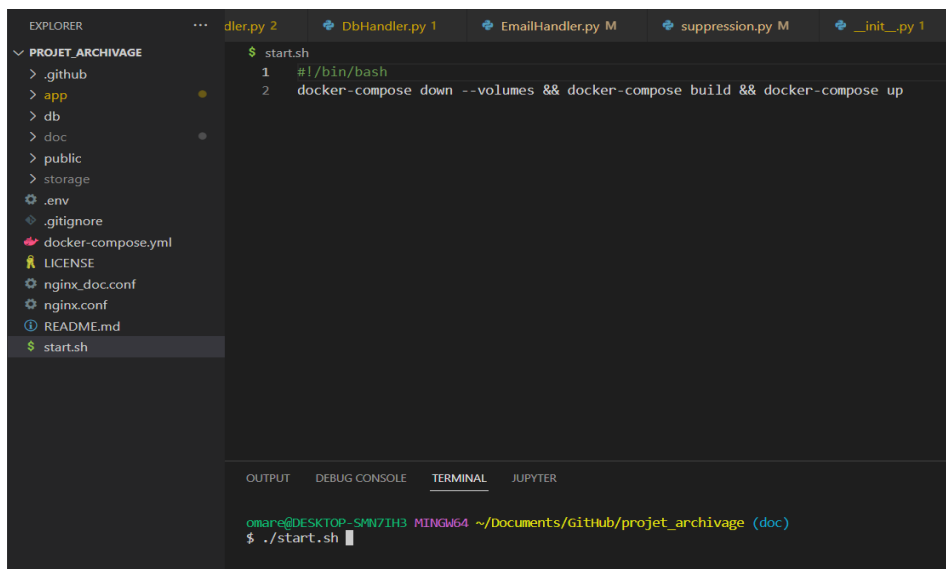


```
1 version: "3.3"
2 services:
3   app:
4     image: "script"
5     restart: always
6     ports:
7       - "5000:1234"
8     build:
9       context: ./app
10      dockerfile: ./Dockerfile
11     volumes:
12       - ./app/config.yml:/app/config.yml
13       - ./doc:/app/doc
14       - ./app/logs:/app/logs
15     working_dir: /app
16     command: ["wait-for-it", "db:5432", "--", "/bin/sh", "-c", "./entry.sh"]
17     depends_on:
18       - nginx
19       - ftp
20       - db
21
22   nginx:
23     image: nginx
24     ports:
25       - "3000:80"
26     volumes:
27       - ./public:/usr/share/nginx/www/data
28       - ./nginx.conf:/etc/nginx/nginx.conf
29   documentation:
30     image: nginx
31     ports:
32       - "8081:80"
33     volumes:
34       - ./doc/app:/usr/share/nginx/www
```

Fig1 . fichier docker-compose.yml qui contient les services

L'utilisateur exécute le script **start.sh** afin de construire l'image app qui contient le code, ainsi que lancer le serveur web, la base de données et le serveur distant.

Fig 2 . Commande pour lancer le script



```
1 #!/bin/bash
2 docker-compose down --volumes && docker-compose build && docker-compose up
```

omare@DESKTOP-SMN7IH3 MINGW64 ~/Documents/GitHub/projet_archivage (doc)
\$./start.sh

Après effectuer le lancement, le serveur Web, le serveur distant, la documentation utilisateur ainsi que la base de données sont construits

Index of /

../		
empty.zip	04-Oct-2022 06:35	150
new.zip	04-Oct-2022 06:35	181
test1.sql.zip	04-Oct-2022 06:35	3818

Fig 3. Serveur web Nginx qui contient nos fichiers zippés.

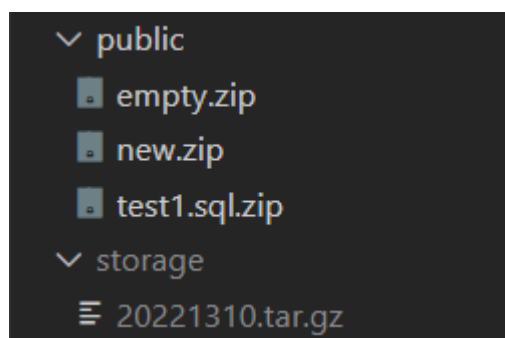
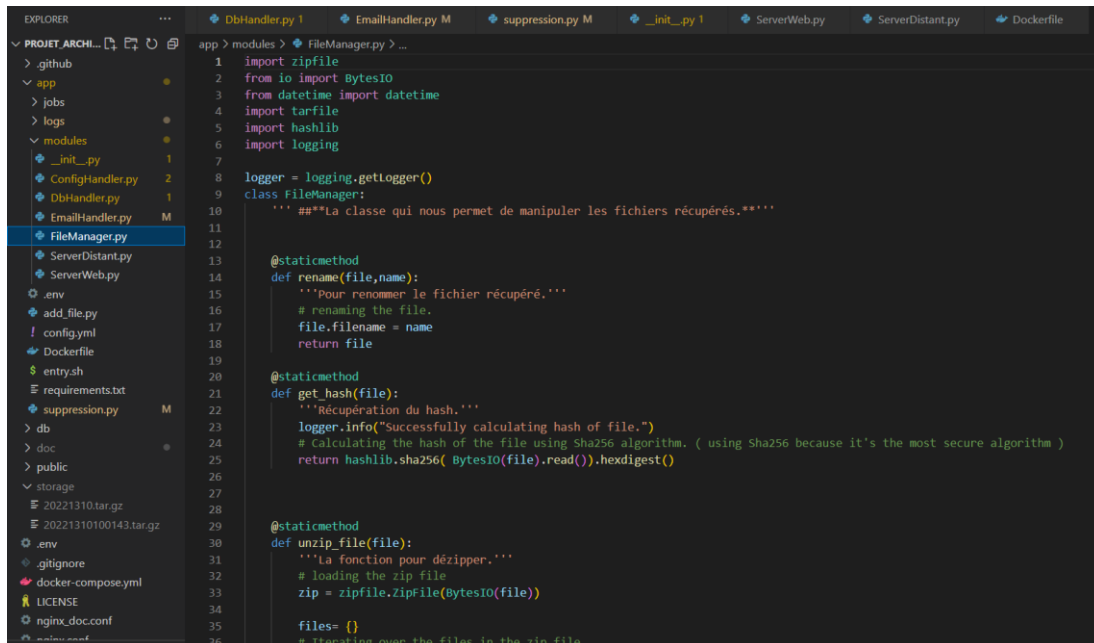


Fig 4. Dossiers storage et Public qui présente respectivement, le contenu du serveur distant et du serveur web

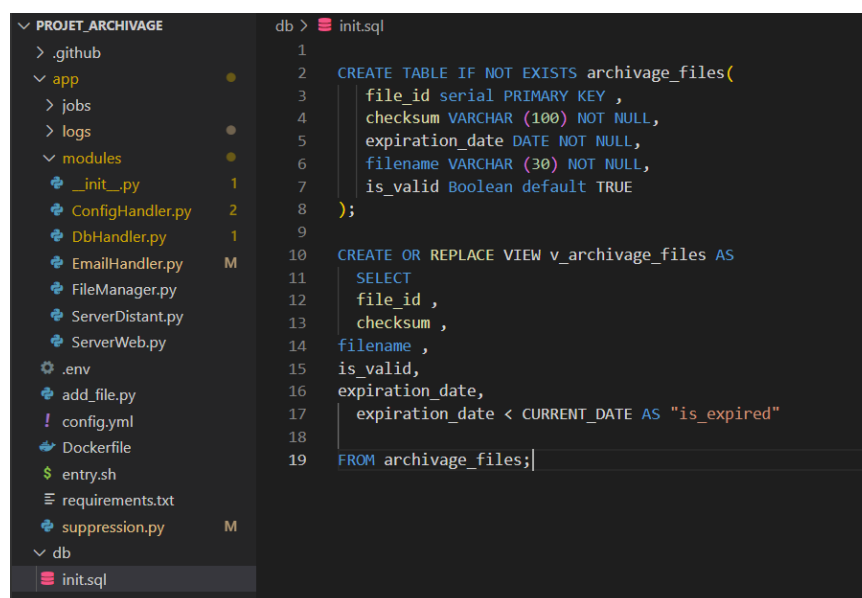
Le lien URL nous permet de télécharger le fichier SQL zippé, ensuite on applique les modifications nécessaires pour le stocker sur le serveur distant à l'aide de la classe FileManager.py



```
1 import zipfile
2 from io import BytesIO
3 from datetime import datetime
4 import tarfile
5 import hashlib
6 import logging
7
8 logger = logging.getLogger()
9 class FileManager:
10     """ ##**La classe qui nous permet de manipuler les fichiers récupérés.**"""
11
12     @staticmethod
13     def rename(file, name):
14         """Pour renommer le fichier récupéré."""
15         # renaming the file.
16         file.filename = name
17         return file
18
19     @staticmethod
20     def get_hash(file):
21         """Récupération du hash."""
22         logger.info("Successfully calculating hash of file.")
23         # Calculating the hash of the file using Sha256 algorithm. ( using Sha256 because it's the most secure algorithm )
24         return hashlib.sha256( BytesIO(file).read()).hexdigest()
25
26     @staticmethod
27     def unzip_file(file):
28         """La fonction pour dézipper."""
29         # loading the zip file
30         zip = zipfile.ZipFile(BytesIO(file))
31
32         files= {}
33         # Iterating over the files in the zip file.
```

Fig 5. Fichier FileManager.py

Pour savoir ce qui se passe au niveau de ce serveur distant, on a créé une base de données pour manipuler ces fichiers qui seront susceptible d'être prolongés ou supprimés selon la configuration que l'utilisateur a la possibilité de modifier.



```
1
2 CREATE TABLE IF NOT EXISTS archivage_files(
3     file_id serial PRIMARY KEY ,
4     checksum VARCHAR (100) NOT NULL,
5     expiration_date DATE NOT NULL,
6     filename VARCHAR (30) NOT NULL,
7     is_valid Boolean default TRUE
8 );
9
10 CREATE OR REPLACE VIEW v_archivage_files AS
11 SELECT
12     file_id ,
13     checksum ,
14     filename ,
15     is_valid,
16     expiration_date,
17     expiration_date < CURRENT_DATE AS "is_expired"
18 FROM archivage_files;
```

Fig 6 . fichier base de données

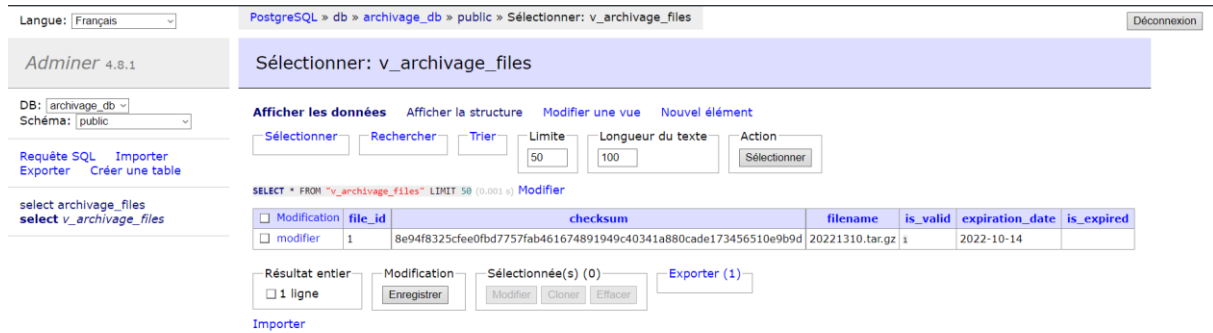


Fig 7. Notre base de données

La base de données reflète ce qui se passe dans le serveur distant, on a utilisé le hash comme outil pour comparer entre les nouveaux fichiers et ceux qui existent déjà.

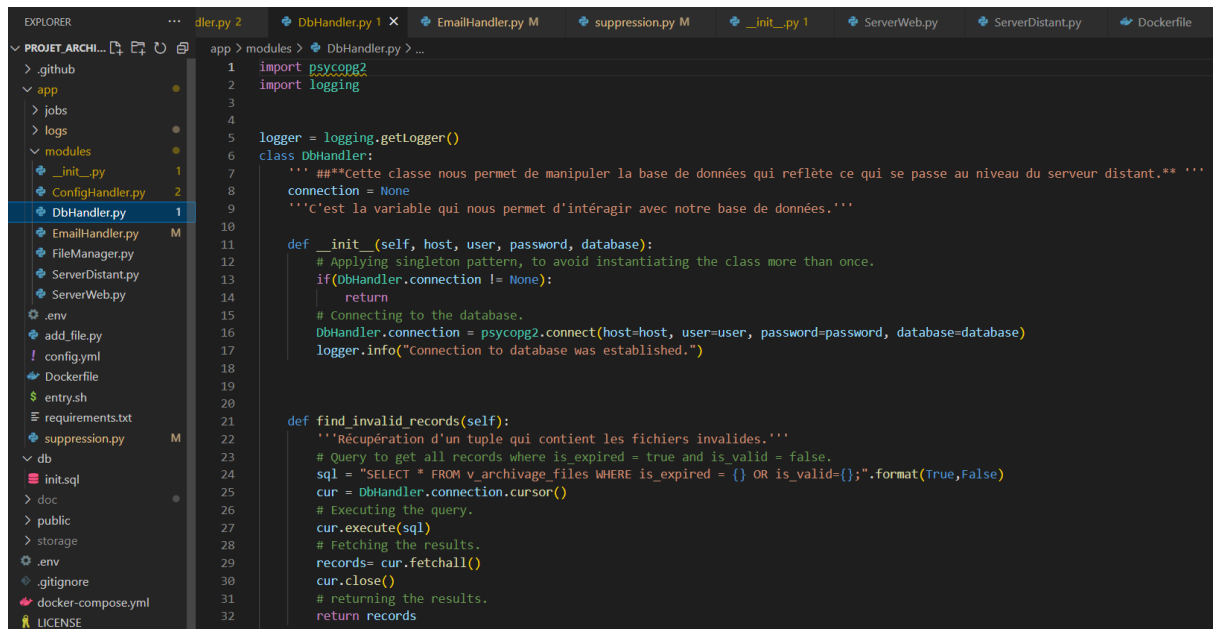


Fig 8. Fichier python pour manipuler cette base de données

Afin d'être notifié pour savoir si le processus a bien fonctionné ou pas, l'utilisateur reçoit un email qui contient les informations nécessaires qui englobe l'intégralité du processus et qui indique les erreurs le cas échéant.

```

1 from datetime import datetime
2 import mimetypes
3 import smtplib,ssl
4 from email.message import EmailMessage
5 import logging
6 from modules.ConfigHandler import ConfigHandler
7
8 logger = logging.getLogger()
9 class EmailHandler:
10
11     ''' ##**Cette classe nous permet de manipuler les emails envoyés à l'utilisateur.** '''
12     server = None
13     ''' cette variable présente notre serveur pour l'envoi du mail. '''
14     mail_sender = None
15     ''' cette variable est l'expéditeur de notre email, c'est le bot automatisé dans notre cas.'''
16     (parameter) smtp_server: Any
17
18     def __init__(self,smtp_server,port,mail_sender, password):
19         # Applying singleton pattern, to avoid instantiating the class more than once.
20         if(EmailHandler.server != None): return
21         # create SSL context to add encrypting while sending mails
22         context = ssl.create_default_context()
23         # connecting to SMTP server
24         EmailHandler.server = smtplib.SMTP_SSL(smtp_server, port, context=context)
25         # login to the SMTP server
26         EmailHandler.server.login(mail_sender, password)
27         EmailHandler.mail_sender = mail_sender
28
29     def message_template(self,mail_receivers,subject,body):
30         ''' La création du message envoyé et tous ses paramètres.'''
31         try:
32             # creating a message object && init config handler
33             config = ConfigHandler()
34             msg = EmailMessage()
35             # Setting props
36             msg['Subject'] = subject
37             msg['To'] = mail_receivers
38             msg['From'] = EmailHandler.mail_sender
39             msg.set_content(body)
40             EmailHandler.server.send_message(msg)
41         except Exception as e:
42             logger.error(e)
43

```

Fig 9. Fichier python pour manipuler les emails

```

1 app 2022-10-13 20:49:01,856 - INFO - Script started.
2 app 2022-10-13 20:49:01,861 - INFO - Connection to database was established.
3 app 2022-10-13 20:49:01,885 - INFO - Connection to distant server was established.
4 app 2022-10-13 20:49:01,892 - INFO - File retrieved successfully.
5 app 2022-10-13 20:49:01,893 - INFO - Successfully calculating hash of file.
6 app 2022-10-13 20:49:01,894 - INFO - Successfully unzipped file.
7 app 2022-10-13 20:49:01,895 - INFO - Successfully tarred file with name 20221310.tar.gz.
8 app 2022-10-13 20:49:01,896 - INFO - Versioning is enabled.
9 app 2022-10-13 20:49:01,900 - INFO - File does exist in FTP.
10 app 2022-10-13 20:49:01,904 - INFO - The expiration date of the file was extended.
11 app 2022-10-13 20:49:01,906 - INFO - Server Distant Connection was closed successfully.
12 app 2022-10-13 20:49:01,907 - INFO - Database Connection was closed successfully.
13 app 2022-10-13 20:49:01,907 - INFO - Script finished.
14

```

Fig 10. Les logs après le lancement du script

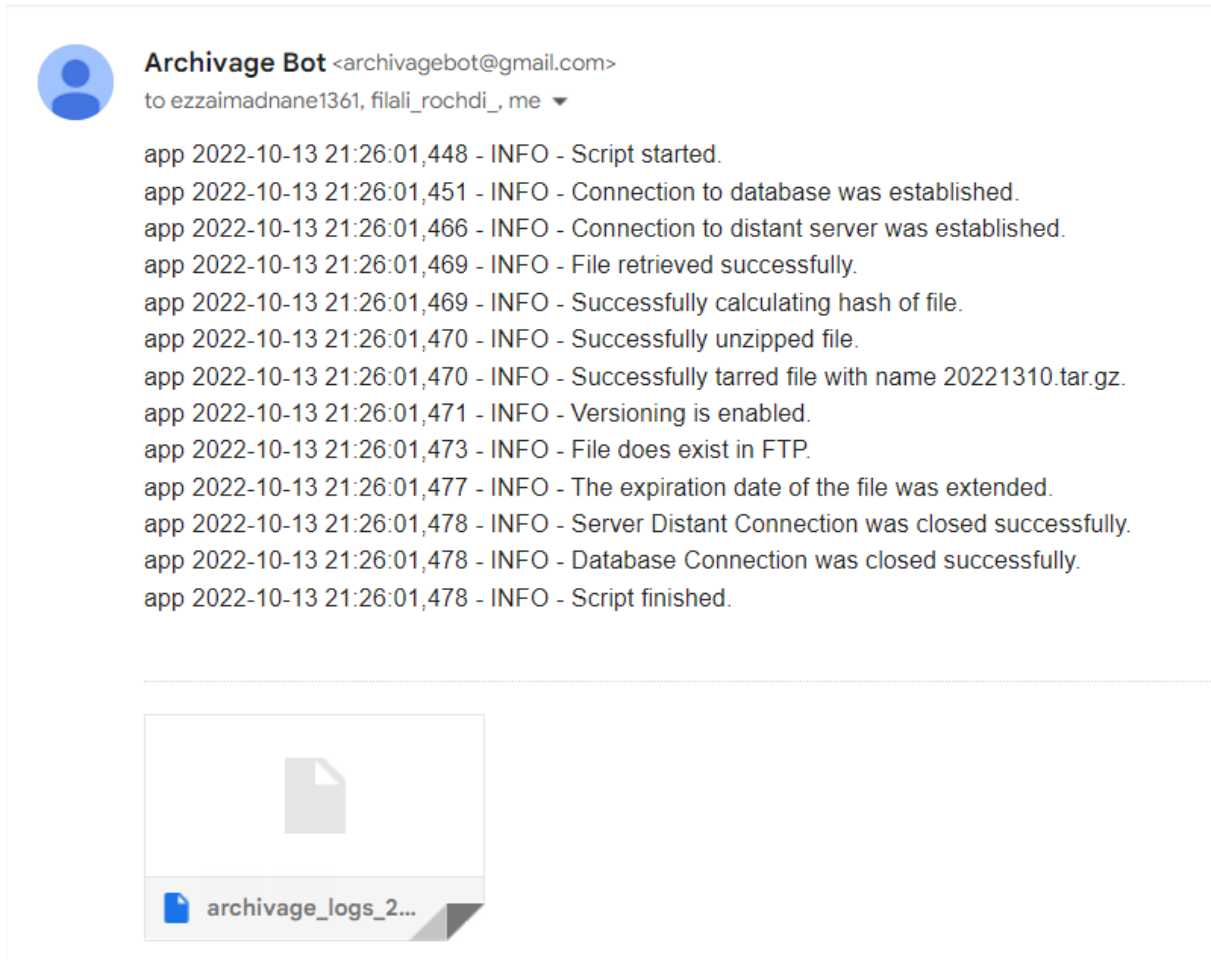


Fig 11. L'email reçu

Le fichier de configuration **config.yml** destiné à l'utilisateur lui permet de modifier la date d'expiration, activer l'historisation ou la désactiver, recevoir les notifications ou pas, inclure les logs dans le mail ou pas.

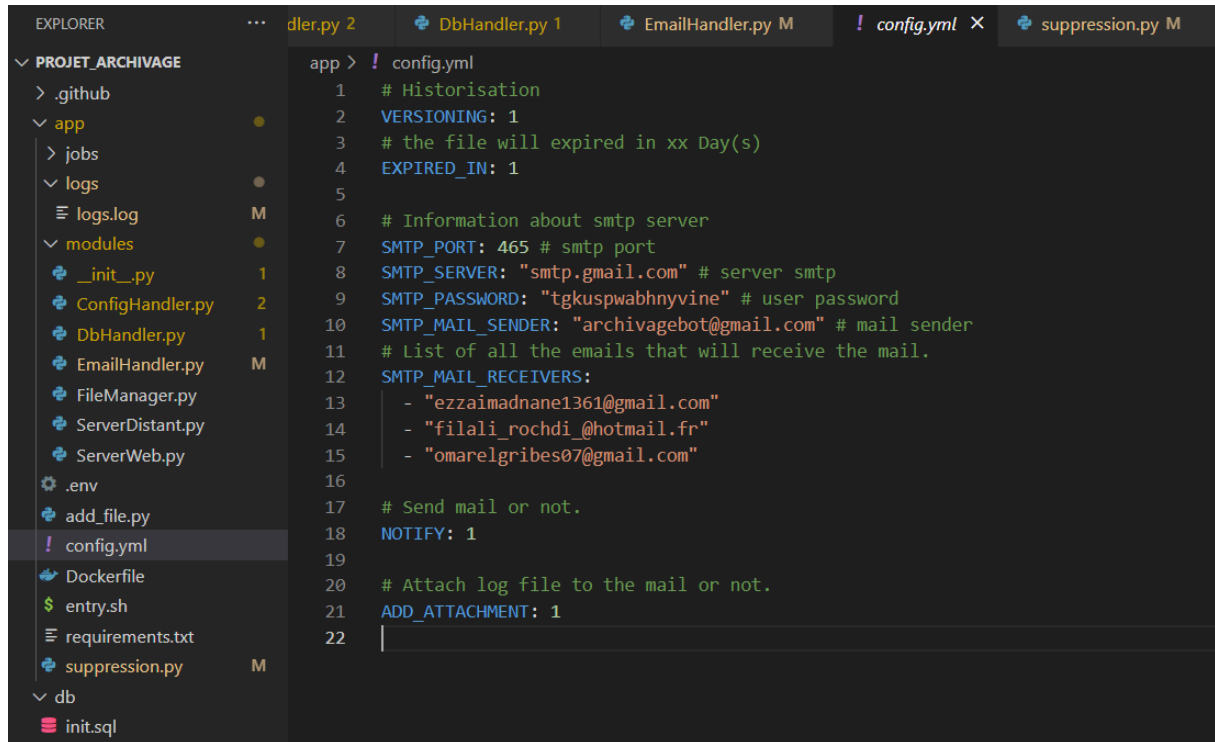


Fig 12. Fichier config.yml à modifier par l'utilisateur.