Procédure Web Scraping du site web de l'EMA

ROUANET Julie
Etudiante en 5ème année pharmacie industrielle
Version 17/07/2025

Sommaire :

A. Introduction :

Le **Web Scraping** (de l'anglais scraping = « gratter/racler ») consiste à **extraire** des **données** de sites Internet et à les **enregistrer** afin de les **analyser** ou de les **utiliser** de toute autre façon. Le scraping permet de collecter des informations de nature bien différente. Il peut par ex. s'agir de coordonnées comme des adresses e-mail ou des numéros de téléphone, mais aussi de mots-clés individuels ou d'URL. Ces informations sont alors rassemblées dans des bases de données locales ou des tableaux. Le Web scraping sélectionne des textes sur les sites Internet afin d'obtenir des informations et de les enregistrer. Ce processus est comparable à un copier-coller automatique (Source : ionos.fr).

Ce document a pour but d'expliquer la logique du programme Python de récupération et mise à jour de manière automatique des fichiers PDF des RCPs des médicaments à usage humain autorisés sur le marché européen à partir du site web de l'EMA dans le but d'en établir une base de données fiable afin d'accélérer l'accès aux informations présentes sur ces RCP. Une extraction des RCP des médicaments retirés du marché européen a également été réalisée dans le but d'étudier la raison de ces retraits.

B. Logique globale du code

À partir du fichier Excel de référence "**Data Medicine Table"** et du **schéma standard d'une URL** pointant vers le PDF du RCP d'un médicament, une première étape de filtrage est réalisée. On ne conserve que les médicaments **à usage humain**, dont le **statut est "autorisé" ou "retiré du marché"**, ainsi que les **colonnes essentielles** : *Nom*, *Numéro de révision* et *Statut*. Ces fichiers (un pour les médicaments autorisés et un pour les retirés) ainsi épuré devient exploitable pour les étapes suivantes.

Ensuite, pour chaque ligne du tableau (chaque ligne représentant un médicament), une **URL est construite dynamiquement** à partir du nom et du statut du médicament, après avoir **nettoyé** ce nom pour qu'il corresponde au format attendu dans l'URL.

Lors du **téléchargement initial**, une requête est envoyée vers cette URL :

- Si le site retourne un **code 200** (accès autorisé), le PDF est téléchargé et stocké dans un dossier correspondant à son statut : ema_authorised_rcp ou ema_withdrawn_rcp.

- Si l'accès est temporairement refusé (code 429, par exemple pour cause de **surcharge** ou de **trop nombreuses requêtes**), les informations du médicament (nom, statut, URL) sont enregistrées dans un fichier CSV failed_urls_authorised.csv ou failed_urls_withdrawn.csv. Ces fichiers seront utilisés pour **relancer automatiquement les téléchargements ultérieurement**, afin d'éviter une saturation du site.

- Si l'URL construite est invalide (code 404, souvent en raison d'un **nettoyage incorrect du nom** ou d'un **format particulier non pris en compte**), le médicament est enregistré dans le fichier not_found_urls.csv. Ces cas seront ensuite **analysés**

manuellement pour ajuster la fonction de nettoyage ou bien **ajouter des exceptions** spécifiques.

Une fois cette phase initiale terminée, les RCP sont classés dans leurs dossiers respectifs. Cependant, ces documents peuvent être régulièrement mis à jour (ajout d'effets indésirables, changements réglementaires, etc.), ce qui se manifeste par un **changement du numéro de révision**.

Lors de chaque **exécution automatique planifiée** (par exemple tous les matins à 9h), le script télécharge la **version actualisée** du fichier de référence. Celui-ci est nettoyé de la même manière que précédemment, puis **comparé avec le fichier de la veille**. Si le numéro de révision d'un médicament a changé, cela signifie que le PDF a été mis à jour :

- L'ancienne version du fichier est renommée [nom_médicament]_old.
- Le nouveau PDF est téléchargé.
- Une fois le nouveau fichier présent, l'ancien est supprimé.

Enfin, le script vérifie qu'aucun RCP n'existe à la fois dans le dossier authorised et dans le dossier withdrawn. Si un même médicament est détecté dans les deux statuts, cela indique un **changement de statut réglementaire** (passage d'"autorisé" à "retiré"). Dans ce cas, le fichier correspondant est **supprimé du dossier des médicaments autorisés** pour maintenir une base de données cohérente et à jour.

C. Aspect technique

1. Structure :

Le code est présent dans un dossier nommé "ema_rcp-main" dans le sous dossier "app" et est divisé en plusieurs dossiers "adapters" et "core" où sont contenus plusieurs fichiers python que le fichier main.py va orchestrer :

- Le fichier **main.py** : fichier orchestre du programme, c'est lui qui est lancé et qui fait appelle à plusieurs autres fonctions dans d'autres fichiers.
- Dossier ***adapters*** : fichier **download_file.py**
  - contient les "adaptateurs" vers l'extérieur (par ex. API, base de données, fichiers, réseau).
- Dossier ***core*** : fichier **update_rcp.py** et **manipulate_df.py**
  - contient la logique métier pure (ce qui ne dépend pas du monde extérieur).

2. Explication

Le fichier main.py

```python
from loguru import logger
from datetime import datetime
import asyncio
from adapters.download_file import download_index, download_files
from core.manipulate_df import simplify_dataframe
from core.update_rcp import rename_update_rcp, update_rcp

# Configurer le logger
today_log: str = datetime.now().strftime("%d-%m-%Y_%H-%M-%S")
today: str = datetime.now().strftime("%d-%m-%Y")
logger.add(f"log/log_{today_log}.log", rotation="500 KB", level="INFO")  # Log

url_index_file: str = (
    "https://www.ema.europa.eu/en/documents/report/medicines-output-medicines-report_en.xlsx"  # noqa:E501
)
index_file_path: str = "index_file.xlsx"
language: str = "en"

# Télécharger le fichier d'index des médicaments
df_authorised, df_withdrawn = asyncio.run(download_index(url_index_file, index_file_path))

# Simplifier le dataframe
df_authorised_light = simplify_dataframe(
    df_authorised,
    path_csv="archives_authorised/simplified_file.csv",
    path_json="list_of_authorised_med.json",
    authorised_names_clean=None)
logger.info("Authorised medicines DataFrame successfully simplified.")

authorised_names_clean = set(df_authorised_light["Name"])

df_withdrawn_light = simplify_dataframe(
    df_withdrawn,
    path_csv="archives_withdrawn/simplified_file.csv",
    path_json="list_of_withdrawn_med.json",
    authorised_names_clean=authorised_names_clean)
logger.info("Withdrawn medicines DataFrame successfully simplified.")

# Renommer les fichiers RCP mis à jour
rename_update_rcp(
    df_authorised_today_path="archives_authorised/simplified_file.csv",
    df_authorised_yesterday_path=f"archives_authorised/simplified_file.csv_{today}.csv"
)
# Mettre à jour les RCP
asyncio.run(update_rcp(
    df_authorised_light,
    language,
    nb_workers=5,
    failed_urls_file="failed_urls_authorised.csv",
    dl_path="ema_authorised_rcp",
    status="Authorised"))

# Télécharger les fichiers PDF authorised
logger.info("Downloading authorised RCP files...")
asyncio.run(download_files(
    language,
    df_authorised_light,
    dl_path="ema_authorised_rcp",
    nb_workers=5,
    failed_urls_file="failed_urls_authorised.csv",
    status="Authorised"))

# Télécharger les fichiers PDF withdrawn
logger.info("Downloading withdrawn RCP files...")
asyncio.run(download_files(
    language,
    df_withdrawn_light,
    dl_path="ema_withdrawn_rcp",
    nb_workers=5,
    failed_urls_file="failed_urls_withdrawn.csv",
    status="Withdrawn"))

logger.info("All tasks completed successfully.")
```

Ce code permet d'orchestrer toutes les autres fonctions et c'est ce fichier qui va être lancé pour débuter le téléchargement et les mises à jour des RCPs. Dans les parties suivantes, nous allons suivre le déroulement du code étape par étape en partant de la lecture du fichier main.py.

Explications étape par étape :

2.1 Importation des outils nécessaires

```
from loguru import logger
from datetime import datetime
import asyncio
from adapters.download_file import download_index, download_files
from core.manipulate_df import simplify_dataframe
from core.update_rcp import rename_update_rcp, update_rcp
```

Ici, on importe tous les outils qui nous seront nécessaires au bon déroulement du fichier main.py comme :

- **loguru** (bibliothèque Python qui sert à écrire des messages dans des fichiers log),
- **datetime** (module standard de Python qui sert à gérer les dates et heures),
- **asyncio** (module standard de Python utilisé pour écrire du code asynchrone : pour pouvoir faire plusieurs choses en même temps sans attendre que chacune soit terminées)
- les **fonctions** que l'on va appeler (présent sur les autres fichiers python).

Tous les autres fichiers python ".py" vont avoir au début de chaque script une liste des outils à utiliser. Certains outils devront être télécharger avant le lancement du script (voir partie Lancement).

2.2. Configurer le logger

```
# Configurer le logger
today_log: str = datetime.now().strftime("%d-%m-%Y_%H-%M-%S")
today: str = datetime.now().strftime("%d-%m-%Y")
logger.add(f"log/log_{today_log}.log", rotation="500 KB", level="INFO")

url_index_file: str = (
    "https://www.ema.europa.eu/en/documents/report/medicines-output-medicines-report_en.xlsx"  # noqa:E501
)
index_file_path: str = "index_file.xlsx"
language: str = "en"
```

Cette partie crée une variable **today_log** correspondant à la date sous le format (jour-mois-année_heure-minute-seconde) afin de dater chaque message d'informations des logs.

Qu'est ce qu'un log ? Un log c'est un message d'information visible sur le terminal* d'un code et dans un fichier dans le dossier **log** permettant de **tracer** toutes les actions du code. Ces messages nous permettent de suivre l'exécution du code et, en cas d'erreur, d'identifier précisément à quel endroit elle s'est produite.

***Terminal** : interface textuelle qui permet à l'utilisateur d'interagir directement avec l'ordinateur en saisissant des commandes, sans passer par l'interface graphique.

Cela se présente de cette manière :

```
2025-06-18 20:10:21.389 | WARNING  | adapters.download_file:download_pdf:102 - Erreur 429 (Too Many Requests) pour Clopidogrel-acino. Nouvelle tentative... (4/5)
2025-06-18 20:10:22.271 | WARNING  | adapters.download_file:download_pdf:102 - Erreur 429 (Too Many Requests) pour Budesonide-formoterol-teva-pharma-bv. Nouvelle tentative... (3/5)
2025-06-18 20:10:22.275 | ERROR    | adapters.download_file:download_pdf:108 - Erreur: Nombre maximum de tentatives (5) atteint pour Optimark
2025-06-18 20:10:22.281 | INFO     | adapters.download_file:download_pdf:77 - T..l..chargement de 81/154 : Vylaer-spiromax
2025-06-18 20:10:22.320 | WARNING  | adapters.download_file:download_pdf:102 - Erreur 429 (Too Many Requests) pour Vylaer-spiromax. Nouvelle tentative... (1/5)
2025-06-18 20:10:23.364 | WARNING  | adapters.download_file:download_pdf:102 - Erreur 429 (Too Many Requests) pour Vylaer-spiromax. Nouvelle tentative... (2/5)
2025-06-18 20:10:28.359 | WARNING  | adapters.download_file:download_pdf:102 - Erreur 429 (Too Many Requests) pour Clopidogrel-teva-pharma. Nouvelle tentative... (4/5)
2025-06-18 20:10:30.151 | WARNING  | adapters.download_file:download_pdf:102 - Erreur 429 (Too Many Requests) pour Biograstim. Nouvelle tentative... (4/5)
2025-06-18 20:10:32.628 | SUCCESS  | adapters.download_file:download_pdf:85 - Succ..s: Vylaer-spiromax
2025-06-18 20:10:32.629 | INFO     | adapters.download_file:download_pdf:77 - T..l..chargement de 82/154 : Celvapan
2025-06-18 20:10:32.692 | WARNING  | adapters.download_file:download_pdf:102 - Erreur 429 (Too Many Requests) pour Celvapan. Nouvelle tentative... (1/5)
2025-06-18 20:10:33.362 | WARNING  | adapters.download_file:download_pdf:102 - Erreur 429 (Too Many Requests) pour Budesonide-formoterol-teva-pharma-bv. Nouvelle tentative... (4/5)
2025-06-18 20:10:34.439 | WARNING  | adapters.download_file:download_pdf:102 - Erreur 429 (Too Many Requests) pour Clopidogrel-acino. Nouvelle tentative... (5/5)
2025-06-18 20:10:36.736 | WARNING  | adapters.download_file:download_pdf:102 - Erreur 429 (Too Many Requests) pour Celvapan. Nouvelle tentative... (2/5)

Date/Time:                Component:
Thread:                   Source:
2025-06-18 20:10:06.605 | ERROR    | adapters.download_file:download_pdf:108 - Erreur: Nombre maximum de tentatives (5) atteint pour Anglox
```

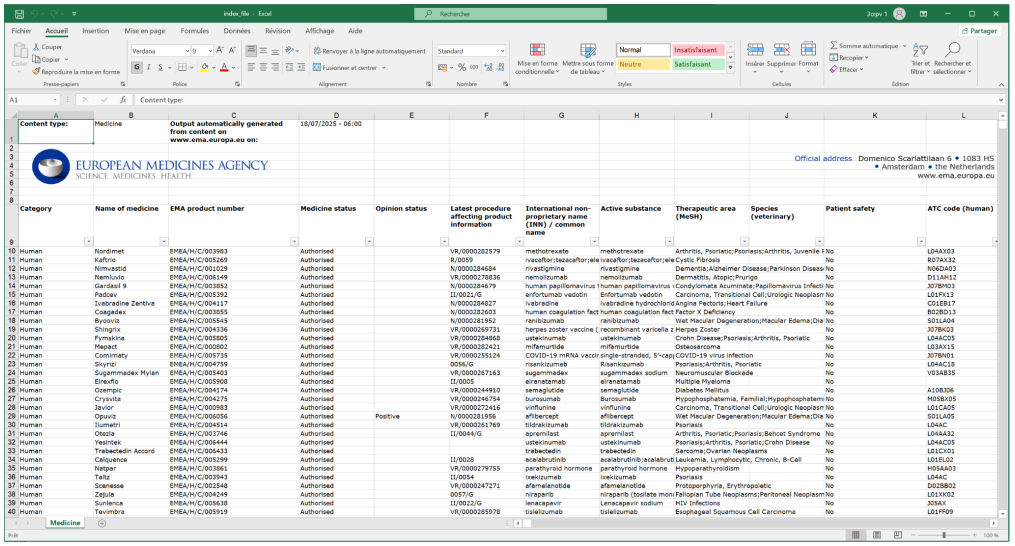
On a aussi établi une date sous format jour-mois-année pour les fichiers archivés que l'on a attribués à la variable **today.** On configure par la suite l'outil log pour pouvoir l'utiliser dans tout le programme.

2.3. Définition des paramètres qui ne varient pas :

- **url_index_file**: on lui attribue la chaîne de caractère correspondant à l'url du fichier excel global (Data Medicine Table)
- **index_file_path** : on lui attribue la chaîne de caractères correspondant au nom du fichier lors du téléchargement soit "index_file.xlsx"
- **language** : on lui attribue la langue de téléchargement des PDF, ici "en" pour anglais.

2.4. Télécharger le fichier index des médicaments

Objectifs : télécharger un fichier de référence "Date Medicine Table" accessible sur le site de l'EMA via un URL.



```
# Télécharger le fichier d'index des médicaments
df_authorised, df_withdrawn = asyncio.run(download_index(url_index_file, index_file_path))
```

df_authorised, df_withdrawn = dataframe* du fichier “data medicine table” téléchargé via la fonction **download_index** avec comme paramètres :

- **url_index_file** = l’URL du fichier
- **index_file_path** = le nom du fichier.

Après cette fonction, on obtient le fichier **index_file.xlsx** qui est le fichier de référence pour la suite.

***DataFrame** : Tableau à 2 dimensions dans le code python que l'on pourra manipuler avec l’outil pandas.

#Zoomons sur cette fonction download_index :

```
async def download_index(
    url_index_file: str,
    index_file_path: str,
) -> pd.DataFrame:
    headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36"}  # noqa: E501
    try:
        response: requests.Response = requests.get(url_index_file, headers=headers)
        while response.status_code == 429:
            logger.info("Error 429 : Too many requests. Retrying in 10 seconds...")
            time.sleep(10)
            response = requests.get(url_index_file, headers=headers)

        if response.status_code == 200:
            with open(index_file_path, "wb") as f:
                f.write(response.content)
                logger.success(f"Download successful from {url_index_file}")
                df: pd.DataFrame = pd.read_excel(index_file_path, skiprows=8)
                df_human: pd.DataFrame = df[df["Category"] == "Human"]
                df_authorised: pd.DataFrame = df_human[df_human["Medicine status"] == "Authorised"]
                df_withdrawn: pd.DataFrame = df_human[df_human["Medicine status"].isin(["Withdrawn", "Withdrawn from rolling review"])]
                return df_authorised, df_withdrawn
        else:
            logger.error(f"Download failed - status {response.status_code} for {url_index_file}")
            raise RuntimeError
    except Exception as exc:
        logger.exception(f"Error: {exc}")
        raise RuntimeError
```

***headers** : Un header est une information envoyée avec une requête internet qui permet de donner des détails au serveur, comme l'identité du navigateur, la langue ou d'autres paramètres techniques afin d’éviter tout refus d’accès.

```
try:
        response: requests.Response = requests.get(url_index_file, headers=headers)
        while response.status_code == 429:
            logger.info("Error 429 : Too many requests. Retrying in 10 seconds...")
            time.sleep(10)
            response = requests.get(url_index_file, headers=headers)
```

```
        if response.status_code == 200:
            with open(index_file_path, "wb") as f:
                f.write(response.content)
                logger.success(f"Download successful from {url_index_file}")
```

Dans cette partie, le code va essayer d'accéder à l'URL du fichier avec la méthode **requests.get**

- Si le code renvoyé par le site est **429**, l'accès est refusé car il y a trop de demandes (il s'agit d'une protection contre les abus), il attend 10 secondes et retente.
- Si le code renvoyé par le site est **200**, le téléchargement peut être effectué, le code va copier-coller le contenu sous forme binaire (wb : write binary) de cet URL dans le dossier sous le nom de **index_file.xlsx.**

```
df: pd.DataFrame = pd.read_excel(index_file_path, skiprows=8)
                df_human: pd.DataFrame = df[df["Category"] == "Human"]
                df_authorised: pd.DataFrame = df_human[df_human["Medicine status"] == "Authorised"]
                df_withdrawn: pd.DataFrame = df_human[df_human["Medicine status"].isin(["Withdrawn",
"Withdrawn from rolling review"])]
                return df_authorised, df_withdrawn
```

Par la suite ce fichier va être lu comme un dataframe nommé "**df**" afin d'être manipulé par l'outil pandas. L'outil pandas va filtrer **df** en ne conservant que que les médicaments à usage humain (filtration "Human") et va renvoyer un dataframe nommé **df_human**.

Ensuite il va filtrer ce **df_human** selon le statut et renvoyer deux dataframes différents :

- **df_authorised** : uniquement les médicaments à usage humain et autorisés sur le marché ("authorised")
- **df_withdrawn** : uniquement les médicaments à usage humain et retirés du marché ("withdrawn" et "withdrawn from rolling review")

On obtient donc **2 dataframes différents filtrés.**

➜La fonction est terminée, on retourne sur le fichier "main".

2.5. Simplification des dataframes

```
# Simplifier les dataframes
df_authorised_light = simplify_dataframe(
    df_authorised,
    path_csv="archives_authorised/simplified_file.csv",
    path_json="list_of_authorised_med.json",
    authorised_names_clean=None)
logger.info("Authorised medicines DataFrame successfully simplified.")

authorised_names_clean = set(df_authorised_light["Name"])

df_withdrawn_light = simplify_dataframe(
    df_withdrawn,
    path_csv="archives_withdrawn/simplified_file.csv",
    path_json="list_of_withdrawn_med.json",
    authorised_names_clean=authorised_names_clean)
logger.info("Withdrawn medicines DataFrame successfully simplified.")
```

Les dataframes téléchargées et filtrées auparavant contiennent beaucoup d'informations et des colonnes inutiles pour notre objectif. La fonction **simplify_dataframe** va simplifier ces deux tableaux et les enregistrer dans un dossier différent.

Dans le dossier : **archives_authorised** pour le **df_authorised**
Dans le dossier **archives_withdrawn** pour le **df_withdrawn**

 # Zoom sur la fonction simplify_dataframe

```
def simplify_dataframe(
    df: pd.DataFrame,
    path_csv: str,
    path_json: str,
    authorised_names_clean: set = None
) -> pd.DataFrame:

    try:
        columns_to_keep: list[str] = [
            "Name of medicine",
            "Revision number",
            "Medicine status",
        ]
        df_light: pd.DataFrame = df.loc[:, columns_to_keep]
        df_light = df_light.rename(
            columns={
                "Name of medicine": "Name",
                "Revision number": "Revision_nb",
                "Medicine status": "Status",
            }
        )
        df_light["Revision_nb"] = df_light["Revision_nb"].fillna(0)
        df_light["Revision_nb"] = df_light["Revision_nb"].astype(
            "int"
        )

        # Correction du nom
        if df_light["Status"].iloc[0] == "Authorised":
            df_light["Name"] = df_light["Name"].apply(clean_name_authorised)
            logger.success("Successfully simplified the Excel file for authorised medicines.")
            df_light.to_json(path_json, orient="records")
        else:
            # On suppose que authorised_names_clean est passé en argument
            def clean_withdrawn_row(name):
                name_clean = clean_name_authorised(name)
                name_edit_authorised = name_clean if authorised_names_clean and name_clean in
authorised_names_clean else ""
                return clean_name_withdrawn(name, name_edit_authorised)
            df_light["Name"] = df_light["Name"].apply(clean_withdrawn_row)
            df_light.to_json(path_json, orient="records")
            logger.success("Successfully simplified the Excel file for withdrawn medicines.")

# Avant d'écraser, archive l'ancien fichier s'il existe
        today: str = datetime.now().strftime("%d-%m-%Y")
        os.makedirs(os.path.dirname(path_csv), exist_ok=True)
        if os.path.exists(path_csv):
            shutil.copy(path_csv, f"{path_csv}_{today}.csv")

        df_light.to_csv(path_csv, index=False)
        logger.success("Successfully archived and saved the new simplified file.")
```

```
        return df_light

    except Exception as exc:
        logger.exception(f"Error: {exc}")
    raise RuntimeError
```

La fonction permet de :

- Garder uniquement les colonnes **“Name of medicine”**, **“Revision number”** et **“Medicine status”**
- Renommer les colonnes en “**Name**” et “**Revision_nb**” et “**Status**”
- De remplir par **0** les cases vides de la colonne “Revision Number” et de les convertir en **nombre entier**
- **Corriger les noms** de la colonne “Name of medicine” pour les noms des médicaments correspondent aux noms présents dans l’URL via les fonctions **clean_name_authorised** et **clean_name_withdrawn**

Il retourne à la fin un **fichier csv** (fichier texte qui stocke des données sous forme de tableau, où chaque ligne correspond à un enregistrement et les colonnes sont séparées par des virgules ou des points-virgules) dans le dossier “archives” nommé : “**simplified_file**”.

#Zoom sur la fonction clean_name_authorised

```
def clean_name_authorised(name: str) -> str:
    # Remplacer les apostrophes typographiques par une apostrophe simple
    name_edit_authorised = name.replace("’", "'").replace("'", "")
    # Supprimer toute la partie entre parenthèses commençant par "in"
    name_edit_authorised = re.sub(r'\(in [^)]+\)', '', name_edit_authorised)
    # Supprimer toute la partie (previously ...)
    name_edit_authorised = re.sub(r'\(previously.*\)', '', name_edit_authorised)
    # Supprimer toutes les parenthèses restantes mais garder leur contenu
    name_edit_authorised = re.sub(r'[()]', '', name_edit_authorised)
    # Remplacer "/" par espace
    name_edit_authorised = name_edit_authorised.replace("/", " ")
    # Supprimer les points
    name_edit_authorised = name_edit_authorised.replace(".", "")
    # Remplacer les virgules, deux-points, points-virgules par des espaces
    name_edit_authorised = re.sub(r"[,;:]", " ", name_edit_authorised)
    # Remplacer plusieurs espaces par un seul espace
    name_edit_authorised = re.sub(r'\s+', ' ', name_edit_authorised).strip()
    # Mettre en majuscule la première lettre de chaque mot
    name_edit_authorised = name_edit_authorised.capitalize()
    # Découper en mots et filtrer petits mots inutiles (optionnel)
    words_to_remove = {'a', 'the', 'of', 'and', 'in', 'on', 'for'}
    words = [w for w in name_edit_authorised.split() if w not in words_to_remove]
    # Remettre en forme avec des tirets
    name_edit_authorised = '-'.join(words)
    return name_edit_authorised
```

Dans cette fonction **simplify_dataframe**, on appelle une autre fonction : **clean_name_authorised**, qui va nettoyer les noms de la colonne “Name of medicine” pour les médicaments ayant le status “Authorised” afin qu’ils correspondent au nom du médicament dans l’URL du PDF.

Exemple : Ibuprofen Gen.Orph doit devenir ibuprofen-genorph pour correspondre à son URL qui est :

https://www.ema.europa.eu/en/documents/product-information/**ibuprofen-genorph-**epar-product-information_en.pdf

En analysant tous les noms des médicaments, on a pu établir cette fonction clean_name qui :

- Remplace les apostrophes typographiques par une apostrophe simple
- Supprime toute la partie entre parenthèses commençant par "in"
- Supprime toute la partie (previously ...)
- Supprime toutes les parenthèses restantes mais garder leur contenu
- Remplace "/" par espace
- Supprime les points
- Remplace les virgules, deux-points, points-virgules par des espaces
- Remplace plusieurs espaces par un seul espace
- Mettre en majuscule la première lettre de chaque mot
- Découpe en mots et filtrer les petits mots inutiles
- Remettre en forme avec des tirets

Elle appelle aussi la fonction **clean_name_withdrawn** pour les médicaments ayant le statut “Withdrawn”

Pourquoi avoir une fonction différente pour le nettoyage des noms de médicaments retirés ?

Il a été observé que certains médicaments retirés du marché portaient le même nom dans la colonne **“Name of medicine”** qu’un médicament toujours autorisé (par exemple : *Liprolog*) et même après application de la même fonction de nettoyage sur les deux (exemple : Vantobra), on obtenait un nom identique pour les fichiers des RCP, alors que l’URL du médicament retiré ne correspondait pas à celle des médicaments autorisés. Cela entraînait un mélange : les fichiers RCP des médicaments autorisés se retrouvaient dans le dossier destiné aux RCP des médicaments retirés.

Voici un exemple illustratif :

	Statut	Nom obtenu avec une seule fonction commune	Nom attendu dans l'URL	Correct ?
Liprolog	Autorisé	liprolog	liprolog	✅
Liprolog	Retiré	liprolog	liprolog-0	❌
Vantobra	Autorisé	vantobra	vantobra	✅
Vantobra (previously Tobramycin PARI)	Retiré	vantobra	vantrobra-previously-tobramycin-pari	❌

Pour éviter ces confusions, une fonction spécifique **clean_name_withdrawn** a été créée. Elle reprend la logique de nettoyage classique avec quelques ajustements supplémentaires :

#Zoom sur la fonction clean_name_withdrawn

```
def clean_name_withdrawn(name: str,
                         name_edit_authorised: str) -> str:
    # Remplacer les apostrophes typographiques par une apostrophe simple
    name_edit_withdrawn = name.replace("’", "'").replace("'", "")
    # Supprimer toute la partie entre parenthèses commençant par "in"
    name_edit_withdrawn = re.sub(r'\(in [^)]+\)', '', name_edit_withdrawn)
    # Supprimer toute la partie (previously ...)
    name_edit_withdrawn = re.sub(r'\(previously.*\)', '', name_edit_withdrawn)
    # Supprimer toutes les parenthèses restantes mais garder leur contenu
    name_edit_withdrawn = re.sub(r'[()]', '', name_edit_withdrawn)
    # Remplacer "/" par espace
    name_edit_withdrawn = name_edit_withdrawn.replace("/", " ")
    # Supprimer les points
    name_edit_withdrawn = name_edit_withdrawn.replace(".", "")
    # Remplacer les virgules, deux-points, points-virgules par des espaces
    name_edit_withdrawn = re.sub(r"[,;:]", " ", name_edit_withdrawn)
    # Remplacer plusieurs espaces par un seul espace
    name_edit_withdrawn = re.sub(r'\s+', ' ', name_edit_withdrawn).strip()
    # Mettre en majuscule la première lettre de chaque mot
    name_edit_withdrawn = name_edit_withdrawn.capitalize()
    # Découper en mots et filtrer petits mots inutiles (optionnel)
    words_to_remove = {'a', 'the', 'of', 'and', 'in', 'on', 'for'}
    words = [w for w in name_edit_withdrawn.split() if w not in words_to_remove]
    # Remettre en forme avec des tirets
    name_edit_withdrawn = '-'.join(words)

    # Si le nom est identique à l'authorised
    if name_edit_withdrawn == name_edit_authorised:
            logger.warning(f"The name {name_edit_withdrawn} is identical for both authorised and
withdrawn medicines.")
        # Chercher la partie (previously ...)
        match = re.search(r'\(previously ([^)]+)\)', name, re.IGNORECASE)
        if match:
            previously = match.group(1)
            # Nettoyer et formater la partie previously
```

```
            previously_clean = '-'.join([w.strip() for w in previously.split()])
            name_edit_withdrawn = f"{name_edit_withdrawn}-{previously_clean.lower()}"
        else:
            name_edit_withdrawn += "-0"
    return name_edit_withdrawn
```

Si le nom est identique à celui qui est autorisé. Le code reprend la partie previously entre parenthèses (si elle existe), enlève les parenthèses et relie les mots par des tirets sinon, s'il n'y a pas de partie "(previously…)", elle ajoute simplement un -0 à la fin du nom.

Pour les exemples précédents, on obtient donc :

	Statut	**Nom obtenu avec 2 fonctions**	**Nom attendu dans l'URL**	**Correct ?**
Liprolog	Autorisé	liprolog	liprolog	✅
Liprolog	Retiré	liprolog-0	liprolog-0	✅
Vantobra	Autorisé	vantobra	vantobra	✅
Vantobra (previously Tobramycin PARI)	Retiré	vantrobra-previously-tobramycin-pari	vantrobra-previously-tobramycin-pari	✅

Ces fonctions peuvent à l'avenir être modifiées, complétées selon les cas observés.

→Retour à la fonction simplify_dataframe

```
# Avant d'écraser, archive l'ancien fichier s'il existe
        today: str = datetime.now().strftime("%d-%m-%Y")
        os.makedirs(os.path.dirname(path_csv), exist_ok=True)
        if os.path.exists(path_csv):
            shutil.copy(path_csv, f"{path_csv}_{today}.csv")

        df_light.to_csv(path_csv, index=False)
        logger.success("Successfully archived and saved the new simplified file.")
        return df_light

    except Exception as exc:
        logger.exception(f"Error: {exc}")
    raise RuntimeError
```

Cette partie de la fonction s'applique lorsqu'on relance le code afin de mettre à jour les RCP et de ne pas écraser le **simplified_file** de la veille puisqu'ils auront le même nom.

En effet cette partie s'assure que s'il existe un fichier **"simplified_file"** dans le dossier correspondant (**archives_withdrawn** ou **archives_authorised**), il copie ce fichier et le renomme **simplified_file.csv_date_du_jour_d'archivage**. On obtient donc 2 fichiers : celui de la **veille** et celui du **jour** afin de les comparer.

En conclusion, cette fonction simplifie et nettoie le tableau data medicine table pour qu'il soit le plus simple d'utilisation pour la suite des fonctions (notamment pour la création de l'URL pour le téléchargement des RCP). Au final, cette fonction simplify_dataframe renvoi un dataframe nommé **df_light.**

De plus, il enregistre aussi des **fichiers json*** correspondant à la liste des médicaments, son numéro de révision et son statut.

***Fichier JSON** : c'est un fichier texte qui organise les données sous forme de paires **clé : valeur**, un peu comme une liste ou un dictionnaire, et qui sert souvent à échanger des données entre programmes ou avec des sites web.

➜ La fonction est terminée, on retourne sur le fichier "main"

2.6. Renommer les fichiers RCP à mettre à jour

```
# Renommer les fichiers RCP mis à jour
rename_update_rcp(
    df_authorised_today_path="archives_authorised/simplified_file.csv",
    df_authorised_yesterday_path=f"archives_authorised/simplified_file.csv_{today}.csv"
)
```

Le fichier main va appeler la fonction **rename_update_rcp** qui a pour objectif de **comparer** les **numéros de révision** entre chaque médicament grâce aux **fichiers simplifiées** créés en amont et de **renommer** les anciens RCPs afin de garder l'ancienne version durant le téléchargement de la nouvelle version. Ceci permet de garantir la sécurité des données et de ne pas perdre l'ancienne version si la mise à jour à échoué.

 # Zoom sur la fonction rename_update_rcp

```
# Fonction pour renommer les fichiers RCP mis à jour
def rename_update_rcp(
        df_authorised_today_path: str = "archives_authorised/simplified_file.csv",
        df_authorised_yesterday_path: str = f"archives_authorised/simplified_file.csv_{today}.csv"
):
    if not df_authorised_yesterday_path or not os.path.exists(df_authorised_yesterday_path):
        logger.error("No file from the previous day found, nothing to compare.")
        return None

    df_today = pd.read_csv(df_authorised_today_path).set_index("Name")
    df_yesterday = pd.read_csv(df_authorised_yesterday_path).set_index("Name")

    for drug_name in df_today.index:
        if drug_name in df_yesterday.index:
            rev_today = df_today.loc[drug_name, "Revision_nb"]
            rev_yesterday = df_yesterday.loc[drug_name, "Revision_nb"]
            if rev_today != rev_yesterday:
                file_path = f"ema_authorised_rcp/{drug_name}.pdf"
                file_old_path = f"ema_authorised_rcp/{drug_name}_old.pdf"
                if os.path.exists(file_path):
                    shutil.move(file_path, file_old_path)
                    logger.info(f"The file {drug_name}.pdf has been renamed to {drug_name}_old.pdf
due to an update.")
```

Cette fonction permet de renommer les fichiers PDF des RCP qui ont été **mis à jour**, en ajoutant le suffixe **"_old"** à l'ancien fichier.

Elle utilise deux paramètres :

- **df_authorised_today_path** : chemin vers le tableau simplifié généré le **jour même**, contenant la liste des médicaments autorisés et son numéro de révision.
- **df_authorised_yesterday_path** : chemin vers le tableau simplifié de **la veille**, pour comparer les données

⚠️ **Cette fonction ne concerne pas les médicaments retirés du marché**, car il n'y a pas de mise à jour dans ces cas.

Déroulement du script :

1. **Vérification de l'existence des fichiers**
   Si le fichier de la veille n'existe pas, un message d'erreur s'affiche et la fonction est ignorée. Il ne peut pas y avoir de comparaison.

2. **Lecture des tableaux**
   Les deux fichiers CSV sont lus avec la bibliothèque Pandas.
   La colonne "Name" est utilisée comme index, c'est-à-dire que chaque médicament est identifié par son nom.

3. **Comparaison des révisions**
   Pour chaque médicament présent dans le tableau du jour :
   a. S'il est aussi présent dans le tableau de la veille
   b. Le numéro de révision d'aujourd'hui et de la veille sont comparés
   c. Si les numéros sont **différents**, cela signifie que le RCP **doit être mis à jour**

4. **Renommer le fichier PDF**
   Si un PDF correspondant au médicament existe, il est renommé en ajoutant le suffixe "_old", ce qui permet de conserver l'ancienne version.

Une fois cette vérification terminée, la fonction se termine et le programme principal peut continuer son exécution dans le fichier "main".

2.7. Mettre à jour les RCP

Si des mises à jour ont été détectées, il faut télécharger les nouveaux RCP correspondants. Pour cela, on utilise la fonction **update_rcp** à l'aide de la commande suivante :

```
# Mettre à jour les RCP
asyncio.run(update_rcp(
    df_authorised_light,
    language,
    nb_workers=5,
    failed_urls_file="failed_urls_authorised.csv",
    dl_path="ema_authorised_rcp",
    status="Authorised"))
```

Paramètres utilisés :

- **df_authorised_light** : Tableau du jour, qui contient les informations des médicaments autorisés
- **language** : Langue dans laquelle les documents seront téléchargés ("en" pour anglais)
- **nb_workers** : Nombre de téléchargements simultanés autorisés (ici fixé à 5)
- **failes_urls_file** : Nom du fichier dans lequel seront stockées les URL des téléchargements échoués (en cas d'erreur 429, trop de requêtes)
- **dl_path** : Dossier dans lequel seront enregistrés les fichiers PDF téléchargés
- **status** : Statut du médicament

 # Zoom sur la fonction update_rcp

```
async def update_rcp(
        df_today: pd.DataFrame,
        language: str = "en",
        nb_workers: int = 5,
        failed_urls_file: str = "failed_urls_authorised.csv",
        dl_path: str = "ema_authorised_rcp",
        status: str = "authorised"
) -> int:

    sem = asyncio.Semaphore(nb_workers)
    nb_updates = 0

    async with aiohttp.ClientSession() as session:
        tasks = []
        for drug_name in df_today["Name"]:
            file_old_path = f"{dl_path}/{drug_name}_old.pdf"
            file_path = f"{dl_path}/{drug_name}.pdf"
            if os.path.exists(file_old_path):
                row = df_today[df_today["Name"] == drug_name].iloc[0]
                tasks.append(
                    download_pdf(
                        language,
                        row,
                        drug_name,
                        len(df_today),
                        dl_path,
                        session,
                        sem,
                        failed_urls_file,
                        status
                    )
                )
                nb_updates += 1
                logger.info(f"Update #{nb_updates} : RCP for {drug_name} added to the download
list.")
        await asyncio.gather(*tasks)

   while await retry_failed_downloads(failed_urls_file,
                                      dl_path,
                                      status,
                                      language,
                                      nb_workers,
                                      "not_found_urls.csv"):
        logger.info("Retrying download of failed files.")
```

```
    for drug_name in df_today["Name"]:
        file_path = f"{dl_path}/{drug_name}.pdf"
        file_old_path = f"{dl_path}/{drug_name}_old.pdf"

        if os.path.exists(file_path) and os.path.exists(file_old_path):
            os.remove(file_old_path)
            logger.info(f"Old RCP file deleted for {drug_name} (new version downloaded
successfully).")

    if nb_updates == 0:
        logger.info("No RCP update was performed.")

    return nb_updates
```

Cette fonction asynchrone permet de gérer le téléchargement des nouveaux RCP mis à jour.

Déroulement :

1. **Mise en place d'une limite de téléchargements simultanés**
   Utilisation d'un **asyncio.Semaphore** pour ne pas dépasser le nombre de téléchargements autorisés en parallèle.

2. **Détection des mises à jour à effectuer**
   Pour chaque médicament présent dans le tableau du jour :

   - Si un fichier **{medoc_name}_old.pdf** existe, cela signifie que ce médicament doit être mis à jour.
   - Dans ce cas, on prépare une **tâche de téléchargement** pour récupérer la nouvelle version du PDF via la fonction **download_pdf.**

3. **Téléchargement des fichiers**
   Ensuite, toutes les tâches de téléchargement sont lancées simultanément à l'aide de **asyncio.gather.**

4. **Gestion des échecs de téléchargement**
   Tant qu'il y a des téléchargements qui ont échoué (stockés dans **failed_urls_file**), le code retente automatiquement de les télécharger jusqu'à ce que le fichier soit **vide** via l'appel de la fonction **retry_failed_download** que l'on abordera par la suite.

5. **Nettoyage des anciens fichiers**
   Une fois les nouveaux PDF correctement téléchargés, les anciens fichiers **{medoc_name}_old.pdf** sont **supprimés** pour éviter les doublons.

6. **Journalisation**
   Des messages d'information (log) sont affichés tout au long du processus, indiquant :
   - Les mises à jour en cours
   - Les tentatives de récupération des erreurs
   - La suppression des anciens fichiers
   - Le cas où aucune mise à jour n'est nécessaire

7. **Retour de la fonction**
   La fonction retourne le nombre total de mises à jour effectuées.

 # Zoom sur la fonction download_pdf

Cette fonction permet de télécharger le fichier PDF du RCP (Résumé des Caractéristiques du Produit) pour un médicament donné depuis le site de l'EMA. Elle vérifie plusieurs conditions avant le téléchargement pour éviter les doublons ou les erreurs inutiles.

9 paramètres :

- **langage** : Langue utilisée pour l'URL de téléchargement ("en" pour anglais)
- **row** : Ligne du tableau contenant les informations du médicament (type : pd.Series)
- **index** : Numéro de la ligne en cours dans la boucle (utile pour suivre la progression)
- **total_count** : Nombre total de médicaments à traiter (permet d'afficher la progression)
- **dl_path** : Chemin du dossier où le PDF doit être enregistré
- **session** : Session HTTP asynchrone pour réaliser les requêtes web (aiohttp.ClientSession)
- **sem** : Sémaphore permettant de limiter le nombre de téléchargements simultanés
- **failed_urls_file** : Fichier où stocker les URL des téléchargements qui ont échoué
- **status** : statut du médicament

```
async def download_pdf(
        language: str,
        row: pd.Series,
        index: int,
        total_count: int,
        dl_path: str,
        session: aiohttp.ClientSession,
        sem: asyncio.Semaphore,
        failed_urls_file: str,
        status: str # "authorised" or "withdrawn"
) -> None:
    nb_retries = 5
    drug_name = row.Name
    echec = False

    if status == "Authorised":
        SPECIAL_CASES = SPECIAL_CASES_AUTHORISED
    else:
        SPECIAL_CASES = SPECIAL_CASES_WITHDRAWN

    if drug_name in SPECIAL_CASES:
        url_path = SPECIAL_CASES[drug_name]
    else:
        url_path = f"{drug_name.replace(' ', '-').lower()}-epar-product-information"

    url =
f"https://www.ema.europa.eu/{language}/documents/product-information/{url_path}_{language}.pdf"
    file_name = f"{drug_name.replace(' ', '-')}.pdf"
    file_path = f"{dl_path}/{file_name}"
    if os.path.exists(file_path):
        logger.info(f"The file {file_path} already exists. Download skipped.")
        return

    not_found_file = "not_found_urls.csv"
```

```
    if os.path.exists(not_found_file):
        df_not_found = pd.read_csv(not_found_file)
        if (df_not_found["Name"] == drug_name).any():
            logger.info(f"{drug_name} already marked as not found. Download skipped.")
            return

    async with sem:
        try:
            logger.info(f"Downloading {index}/{total_count} : {drug_name}")
            retries = 0
            while retries < nb_retries:
                async with session.get(url) as resp:
                    if resp.status == 200:
                        content = await resp.read()
                        with open(file_path, "wb") as f:
                            f.write(content)
                        logger.success(f"Success: {drug_name}")
                        return
                    elif resp.status == 404:
                        logger.error(f"Error 404 for {drug_name}")
                        not_found_file = "not_found_urls.csv"
                        df_404 = pd.DataFrame([[drug_name, status, url]], columns=["Name",
"Status","Url"])
                        echec = False
                        if os.path.exists(not_found_file):
                            df_404_existing = pd.read_csv(not_found_file)
                        else:
                            df_404_existing = pd.DataFrame(columns=["Name", "Status", "Url"])
                        if not (df_404_existing["Name"] == drug_name).any():
                            df_404_final = pd.concat([df_404_existing, df_404], ignore_index=True)
                            df_404_final.to_csv(not_found_file, index=False)
                            logger.info(f"{drug_name} recorded in {not_found_file}")
                        return
                    elif resp.status == 429:
                        logger.warning(f"Error 429 : too many requests for {drug_name}.
Retrying...({retries + 1}/{nb_retries})")
                        sleeptime = random.randint(1, 15)
                        await asyncio.sleep(sleeptime)
                        retries += 1
            if not echec:
                logger.error(f"Error: Maximum number of retries ({nb_retries}) reached for
{drug_name}")
                echec = True
        except Exception as e:
            logger.exception(f"Exception during download of {drug_name} : {e}")
            echec = True

        if echec:
            df_echec = pd.DataFrame([[drug_name, status, url]], columns=["Name", "Status", "Url"])
            if os.path.exists(failed_urls_file):
                df_echec_existing = pd.read_csv(failed_urls_file)
            else:
                df_echec_existing = pd.DataFrame(columns=["Name", "Status", "Url"])
            if not (df_echec_existing["Name"] == drug_name).any():
                df_echec_final = pd.concat([df_echec_existing, df_echec], ignore_index=True)
                df_echec_final.to_csv(failed_urls_file, index=False)
                logger.info(f"{drug_name} recorded in {failed_urls_file}")
```

Allons-y étape par étape :

1. Construction de l'URL

- Si le médicament est un **cas particulier (SPECIAL_CASES)** car la fonction **clean_name_authorised** n'arrive pas à nettoyer correctement son nom pour correspondre au nom dans l'URL ou que l'URL n'est pas la même que les autres, on utilise l'URL spécifique définie dans ce dictionnaire.

```
if status == "Authorised":
        SPECIAL_CASES = SPECIAL_CASES_AUTHORISED
    else:
        SPECIAL_CASES = SPECIAL_CASES_WITHDRAWN

    if drug_name in SPECIAL_CASES:
        url_path = SPECIAL_CASES[drug_name]
    else:
        url_path = f"{drug_name.replace(' ', '-').lower()}-epar-product-information"
```

- Sinon, on construit **l'URL standard** à partir du nom du médicament.

```
url =
f"https://www.ema.europa.eu/{language}/documents/product-information/{url_path}_{language}.p
df"
```

2. Vérifications préalables :

- Si le fichier PDF existe déjà dans le dossier, le téléchargement est **ignoré**.

```
    file_name = f"{drug_name.replace(' ', '-')}.pdf"
    file_path = f"{dl_path}/{file_name}"
    if os.path.exists(file_path):
        logger.info(f"The file {file_path} already exists. Download skipped.")
        return
```

- Si le médicament est déjà marqué comme **introuvable** dans le fichier not_found_urls.csv, on **ignore** le téléchargement.

```
not_found_file = "not_found_urls.csv"
    if os.path.exists(not_found_file):
        df_not_found = pd.read_csv(not_found_file)
        if (df_not_found["Name"] == drug_name).any():
            logger.info(f"{drug_name} already marked as not found. Download skipped.")
            return
```

3. Téléchargement asynchrone

```
async with sem:
        try:
            logger.info(f"Downloading {index}/{total_count} : {drug_name}")
            retries = 0
            while retries < nb_retries:
                async with session.get(url) as resp:
                    if resp.status == 200:
                        content = await resp.read()
                        with open(file_path, "wb") as f:
                            f.write(content)
                        logger.success(f"Success: {drug_name}")
                        return
                    elif resp.status == 404:
                        logger.error(f"Error 404 for {drug_name}")
                        not_found_file = "not_found_urls.csv"
                        df_404 = pd.DataFrame([[drug_name, status, url]], columns=["Name",
"Status","Url"])
                        echec = False
                        if os.path.exists(not_found_file):
                            df_404_existing = pd.read_csv(not_found_file)
                        else:
                            df_404_existing = pd.DataFrame(columns=["Name", "Status", "Url"])
                        if not (df_404_existing["Name"] == drug_name).any():
                            df_404_final = pd.concat([df_404_existing, df_404], ignore_index=True)
                            df_404_final.to_csv(not_found_file, index=False)
                            logger.info(f"{drug_name} recorded in {not_found_file}")
                        return
                    elif resp.status == 429:
                        logger.warning(f"Error 429 : too many requests for {drug_name}.
Retrying...({retries + 1}/{nb_retries})")
                        sleeptime = random.randint(1, 15)
                        await asyncio.sleep(sleeptime)
                        retries += 1
            if not echec:
                logger.error(f"Error: Maximum number of retries ({nb_retries}) reached for
{drug_name}")
                echec = True
        except Exception as e:
            logger.exception(f"Exception during download of {drug_name} : {e}")
            echec = True
```

- La fonction utilise un sémaphore (sem) pour limiter le nombre de téléchargements simultanés.

- Elle tente jusqu'à **5 fois** de télécharger le PDF. Tant que le nombre d'essai est inférieur à 5 elle relance la fonction :
    - Si le téléchargement réussi (**status 200**), le fichier est enregistré localement
    - Si le fichier est introuvable (**status 404**), le médicament est ajouté au fichier **not_found_urls.csv**. Cela peut être dû à un mauvais nettoyage du nom par clean_name où le nom du médicament ne correspond pas au nom dans l'URL ou l'URL est différente
    - Si le serveur renvoie trop de requêtes (**status 429**), la fonction attend un temps aléatoire entre 1 et 15 secondes avant de réessayer.

4. Gestion des erreurs

```
 if echec:
          df_echec = pd.DataFrame([[drug_name, status, url]], columns=["Name", "Status", "Url"])
          if os.path.exists(failed_urls_file):
              df_echec_existing = pd.read_csv(failed_urls_file)
          else:
              df_echec_existing = pd.DataFrame(columns=["Name", "Status", "Url"])
          if not (df_echec_existing["Name"] == drug_name).any():
              df_echec_final = pd.concat([df_echec_existing, df_echec], ignore_index=True)
              df_echec_final.to_csv(failed_urls_file, index=False)
              logger.info(f"{drug_name} recorded in {failed_urls_file}")
```

- Si le téléchargement **échoue** après 5 tentatives ou si une exception est levée, il est considéré comme un échec (Echec = True) et le nom du médicament ainsi que l'URL sont ajoutés au fichier **failed_urls_file** qui va retenter le téléchargement jusqu'à ce qu'il soit vide par la fonction **retry_failed_download.**

Zoom sur retry_failed_download

Cette fonction permet de retenter automatiquement le téléchargement des fichiers PDF qui ont échoué lors des tentatives précédentes.

Elle lit la liste des échecs dans le fichier **failed_urls.csv**, réessaie de télécharger les documents, et met à jour cette liste jusqu'à ce qu'il n'y ait plus rien à télécharger ou que les échecs persistent.

Paramètres :

- **failed_urls_file** : Fichier CSV contenant les médicaments dont le téléchargement à échoué précédemment
- **dl_path** : Dossier où les fichiers PDF doivent être enregistrées
- **status** : Statut du médicament (Authorised ou Withdrawn)
- **langage** : Langue utilisée pour construire les URL (par défaut "en")
- **nb_workers** : Nombre de téléchargements simultanés autorisés (par défaut : 3)
- **not_found_file** : Fichier CSV contenant les médicaments introuvables (erreur 404)

```
async def retry_failed_downloads(
        failed_urls_file: str,
        dl_path: str,
        status: str,
        language: str = "en",
        nb_workers: int = 3,
        not_found_file: str = "not_found_urls.csv",
        ) -> bool:

    if not os.path.exists(failed_urls_file):
        logger.info("No failed_urls.csv file found. No downloads to retry.")
        return False

    df_failed = pd.read_csv(failed_urls_file)
    if df_failed.empty:
        logger.info("The failed_urls.csv file is empty. No downloads to retry.")
```

```
        return False

    total_count = len(df_failed)
    os.makedirs(dl_path, exist_ok=True)

    # Téléchargement des fichiers échoués
    sem = asyncio.Semaphore(nb_workers)
    async with aiohttp.ClientSession() as session:
        tasks = []
        for idx, row in enumerate(df_failed.itertuples(), 1):
            drug_name = row.Name
            file_path = f"{dl_path}/{drug_name}.pdf"
            tasks.append(download_pdf(
                    language,
                    row,
                    idx,
                    total_count,
                    dl_path,
                    session,
                    sem,
                    failed_urls_file,
                    status))
        await asyncio.gather(*tasks)

    if os.path.exists(failed_urls_file):
        df_failed = pd.read_csv(failed_urls_file)

    # Supprimer ceux qui ont été téléchargés
        df_failed = df_failed[~df_failed["Name"].apply(lambda drug_name:
os.path.exists(f"{dl_path}/{drug_name}.pdf"))]

    # Supprimer ceux en 404
    if os.path.exists(not_found_file):
        df_404 = pd.read_csv(not_found_file)
        df_failed = df_failed[~df_failed["Name"].isin(df_404["Name"])]

    # S'il ne reste rien, on supprime le fichier
    if df_failed.shape[0] == 0:
        os.remove(failed_urls_file)
        logger.info("All files have been downloaded, deleting the failed_urls.csv file.")
        return False
    else:
        # Sinon, on met à jour la liste
        df_failed.to_csv(failed_urls_file, index=False)
        logger.info(f"{len(df_failed)} files remain to be downloaded.")
        return True
```

1. Vérifications préalables

- Si le fichier **failed_urls.csv** **n'existe pas** ou est **vide**, la fonction s'arrête et renvoie False (rien à retenter).

```
    if not os.path.exists(failed_urls_file):
        logger.info("No failed_urls.csv file found. No downloads to retry.")
        return False

    df_failed = pd.read_csv(failed_urls_file)
    if df_failed.empty:
```

```
        logger.info("The failed_urls.csv file is empty. No downloads to retry.")
        return False
```

2. Création du dossier de destination

- Si le dossier d'enregistrement des fichiers n'existe pas, il est créé et on défini la variable **total_count** qui est la longueur du tableau **failed_urls_file** qui représente le nombre de téléchargement à retenter (indicateurs)

```
total_count = len(df_failed)
os.makedirs(dl_path, exist_ok=True)
```

3. Retente des téléchargements

- La fonction prépare des tâches de téléchargement pour chaque médicament encore en échec.

```
sem = asyncio.Semaphore(nb_workers)
async with aiohttp.ClientSession() as session:
        tasks = []
```

- Les téléchargements sont gérés en asynchrone avec un sémaphore pour limiter le nombre simultané.

```
        for idx, row in enumerate(df_failed.itertuples(), 1):
            medoc_name = row.Name
            file_path = f"{dl_path}/{medoc_name}.pdf"
            tasks.append(download_pdf(
                    langage,
                    row,
                    idx,
                    total_count,
                    dl_path,
                    file_path,
                    session,
                    sem,
                    failed_urls_file,))
        await asyncio.gather(*tasks)
```

4. Nettoyage de la liste des échecs

- Les fichiers qui ont finalement été téléchargés avec succès sont retirés de la liste.

```
  if os.path.exists(failed_urls_file):
          df_failed = pd.read_csv(failed_urls_file)

# Supprimer ceux qui ont été téléchargés
        df_failed = df_failed[~df_failed["Name"].apply(lambda drug_name:
os.path.exists(f"{dl_path}/{drug_name}.pdf"))]
```

- Les médicaments marqués comme introuvables (not_found_urls.csv) sont également exclus.

```
 # Supprimer ceux en 404
    if os.path.exists(not_found_file):
        df_404 = pd.read_csv(not_found_file)
```

```
        df_failed = df_failed[~df_failed["Name"].isin(df_404["Name"])]
```

5. Mise à jour de la liste ou suppression

- Si la liste des échecs est vide après ces opérations, le fichier **failed_urls.csv** est supprimé.

- Sinon, la liste est mise à jour avec les médicaments encore en échec et la fonction renvoie “True” pour indiquer qu'il reste des tentatives à faire.

```
# S'il ne reste rien, on supprime le fichier
    if df_failed.shape[0] == 0:
        os.remove(failed_urls_file)
        logger.info("All files have been downloaded, deleting the failed_urls.csv file.")
        return False
    else:
        # Sinon, on met à jour la liste
        df_failed.to_csv(failed_urls_file, index=False)
        logger.info(f"{len(df_failed)} files remain to be downloaded.")
        return True
```

➜ Les fichiers sont mis à jour, on retourne sur le fichier “main”.

2.8 Télécharger les fichiers PDF autorisés

```
# Télécharger les fichiers PDF authorised
logger.info("Downloading authorised RCP files...")
asyncio.run(download_files(
    language,
    df_authorised_light,
    dl_path="ema_authorised_rcp",
    nb_workers=5,
    failed_urls_file="failed_urls_authorised.csv",
    status="Authorised"))
```

Lorsqu'il s'agit du **téléchargement initial** des RCP autorisés ou du téléchargement des RCP de **nouveaux médicaments**, c'est la fonction **download_files** qui est appelée.

 # Zoom sur la fonction download_files

6 paramètres :

- **language** : Langue utilisée pour la construction des URL (ex : "en" pour anglais)
- **df_light** : Tableau contenant les informations des médicaments à télécharger = simplified_file
- **dl_path** : Dossier où les fichiers PDF seront enregistrés
- **nb_workers** : Nombre de téléchargements simultanés autorisés
- **failed_urls_file** : Fichier où seront stockées les URL des téléchargements échoués
- **status** : statut du médicament

```
# Fonction principale pour télécharger les fichiers PDF
async def download_files(
    language: str,
    df_light: pd.DataFrame,
```

```
    dl_path: str,
    nb_workers: int,
    failed_urls_file: str,
    status: str
):
    total_count = len(df_light)
    os.makedirs(dl_path, exist_ok=True)

    sem = asyncio.Semaphore(nb_workers)
    async with aiohttp.ClientSession() as session:
        tasks = []
        for idx, row in enumerate(df_light.itertuples(), 1):
            drug_name = row.Name
            file_path = f"{dl_path}/{drug_name}.pdf"
            tasks.append(
                download_pdf(
                    language,
                    row,
                    idx,
                    total_count,
                    dl_path,
                    session,
                    sem,
                    failed_urls_file,
                    status
                )
            )
        await asyncio.gather(*tasks)

    while await retry_failed_downloads(failed_urls_file,
                                       dl_path,
                                       status,
                                       language,
                                       nb_workers,
                                       not_found_file="not_found_urls.csv"):
        logger.info("Retrying failed files...")
```

1. Préparation

- Calcul du nombre total de médicaments à traiter (total_count).
- Création du dossier de destination (ema_rcp_authorised) s'il n'existe pas déjà.

```
total_count = len(df_light)
    os.makedirs(dl_path, exist_ok=True)
```

2. Téléchargement des fichiers

- Mise en place d'un sémaphore (sem) pour limiter le nombre de téléchargements en parallèle.
- Création d'une session HTTP asynchrone* avec **aiohttp**
- Parcours de chaque ligne du tableau df_light.
- Pour chaque médicament :
    - Construction du chemin du fichier PDF.
    - Ajout d'une tâche asynchrone qui appelle la fonction **download_pdf** pour ce médicament.
- Toutes les tâches sont ensuite lancées simultanément avec **asyncio.gather.**

***Session HTTP asynchrone :** c'est comme ouvrir un canal de communication temporaire entre le programme et un site web, qui permet d'envoyer plusieurs demandes sans tout recommencer à chaque fois, ce qui rend les échanges plus rapides et efficaces.

```
sem = asyncio.Semaphore(nb_workers)
    async with aiohttp.ClientSession() as session:
        tasks = []
        for idx, row in enumerate(df_light.itertuples(), 1):
            drug_name = row.Name
            file_path = f"{dl_path}/{drug_name}.pdf"
            tasks.append(
                download_pdf(
                    language,
                    row,
                    idx,
                    total_count,
                    dl_path,
                    session,
                    sem,
                    failed_urls_file,
                    status
                )
            )
        await asyncio.gather(*tasks)
```

3. Gestion des échecs

- Après les premières tentatives, la fonction **retry_failed_downloads** est appelée pour les erreurs 429
- Tant qu'il reste des fichiers dont le téléchargement a échoué, la fonction tente de nouveau automatiquement de les récupérer.
- Un message est affiché à chaque nouvelle tentative.

```
  while await retry_failed_downloads(failed_urls_file,
                                     dl_path,
                                     status,
                                     language,
                                     nb_workers,
                                     not_found_file="not_found_urls.csv"):
        logger.info("Retrying failed files...")
```

Tous les RCP des médicaments autorisés sur le marché sont téléchargé dans le dossier : ema_authorised_rcp

➜ Retour sur le fichier main.

2.9. Téléchargement des fichiers PDF withdrawn

Cette fonction fait exactement la même chose, elle appelle la fonction **download_file** mais pour les RCP des médicaments retirés du marché.

```
# Télécharger les fichiers PDF withdrawn
logger.info("Downloading withdrawn RCP files...")
asyncio.run(download_files(
    language,
    df_withdrawn_light,
    dl_path="ema_withdrawn_rcp",
    nb_workers=5,
    failed_urls_file="failed_urls_withdrawn.csv",
    status="Withdrawn"))
```

2.10. Update des statuts

```
# Supprimer les fichiers RCP ayant un statut "Withdrawn" du dossier des RCP autorisés
logger.info("Removing authorised RCP files that are now withdrawn...")
change_status(df_authorised_light)
change_status(df_withdrawn_light)
```

Un fois tous les RCP téléchargés et mis à jour, si un RCP se trouve à la fois dans le dossier **ema_authorised_rcp** et **ema_withdrawn_rcp**, cela signifie que le médicament a **changé de statut** et a été retiré du marché. Dans ce cas là, le script appelle la fonction change_status et **supprime** le RCP du dossier “ema_authorised_rcp”.

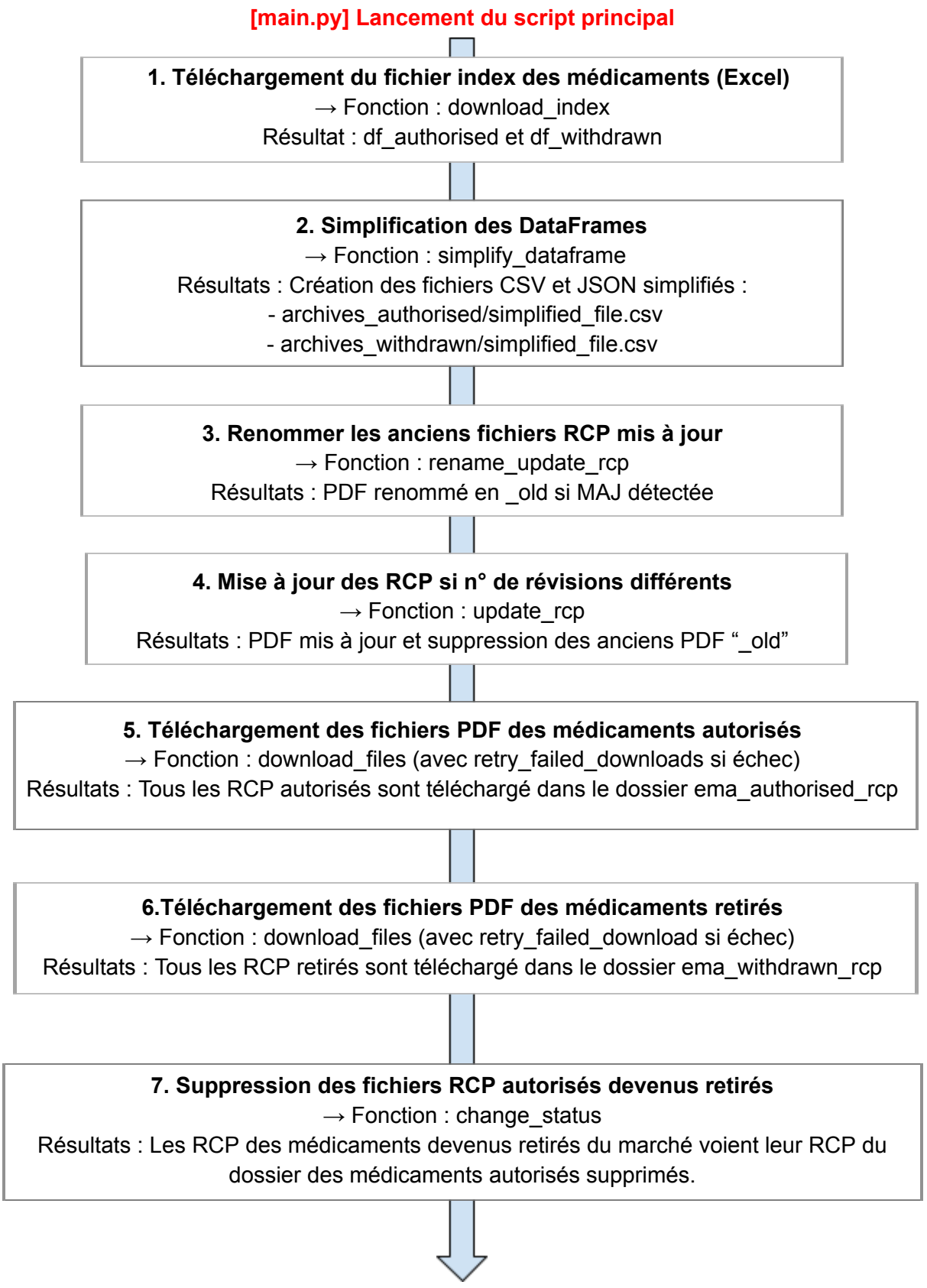
```
def change_status(df_today : pd.DataFrame) -> None:

    for drug_name in df_today["Name"]:
        file_path_authorised = f"ema_authorised_rcp/{drug_name}.pdf"
        file_path_withdrawn = f"ema_withdrawn_rcp/{drug_name}.pdf"
        if os.path.exists (file_path_authorised) and os.path.exists(file_path_withdrawn):
            os.remove(file_path_authorised)
            logger.info(f"Removed {file_path_authorised} because {drug_name} is now withdrawn.")
```

➜ Retour sur le fichier main.

Une fois toutes les tâches accompli, un message s’affiche : “All tasks completed successfully”

```
logger.info("All tasks completed successfully.")
```

3. Schéma récapitulatif :



4. Lancement

Le script doit être lancé une première fois manuellement depuis l'ordinateur ou le serveur qui exécutera automatiquement le projet par la suite (en local ou via une connexion Bureau à distance, selon le cas). Le premier lancement dure environ 5H.

4.1. Type de serveur

Pour que le script Python puisse s'exécuter automatiquement de manière autonome, **le serveur doit être idéalement un serveur exécutable**, c'est-à-dire :

- un **serveur Windows ou Linux disposant d'un système d'exploitation** (et non un simple espace de stockage),
- sur lequel il est **possible d'installer et d'exécuter des programmes** (comme Python) et **d'accéder à internet.**
- et pour lequel on dispose des **droits administrateur** nécessaires à la configuration de tâches planifiées.

Dans le cas contraire, si le serveur ne permet pas l'exécution de code (ex. : simple partage de fichiers type NAS), une alternative consiste à **installer le projet sur un poste de travail local** (ordinateur de référence), souvent allumé et connecté au serveur. Ce poste pourra ainsi **exécuter automatiquement le script** à une fréquence définie (hebdomadaire, mensuelle…) tout en ayant accès aux fichiers partagés.

4.2. Installation Python

Avant de le lancer, il faut installer la dernière version de python ou celle utilisée pour la création de ces fichiers : **Python 3.13.5 (**si possible afin d'éviter les problèmes de compatibilité).

Lien de téléchargement python 3.13.5 : Download Python | Python.org

L'installation se fera soit en se connectant au serveur par la **connexion bureau à distance (RDP)** (sur la barre de recherche windows), soit directement sur l'ordinateur local si le serveur est un simple stockage de fichiers partagés.

Si vous ne disposez pas des droits administrateurs pour installer Python depuis le site officiel, vous pouvez essayer de passer par le Microsoft Store.

Lors de l'installation, si on vous le propose, veillez à **cocher l'option "Add Python to PATH"**, afin de pouvoir l'utiliser directement dans l'invite de commande.

4.3. Récupérer les fichiers python

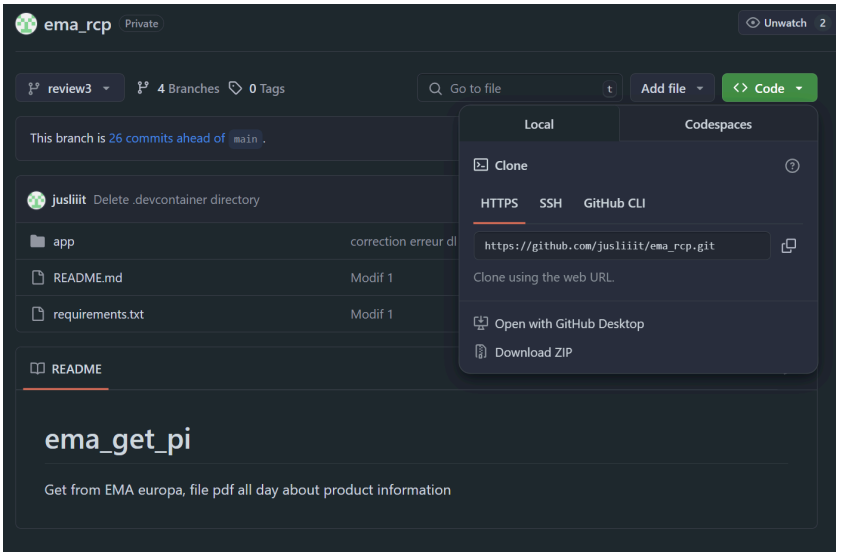
Les fichiers sont disponibles sous format zip dans le serveur PharVig ou sont stockés sur un repository sur github.

Qu'est ce que GitHub ? C'est un **site web** où les développeurs peuvent **stocker**, **partager** et **collaborer** sur du code. (un peu comme un Google Drive du code).

Un repository: c'est un **dépôt** contenant l'ensemble des fichiers d'un projet de développement.

Lien vers le github : [jusliiit/ema_rcp]

En cliquant sur le bouton vert "Code" et "Download ZIP", vous pourrez télécharger les fichiers utiles au bon déroulement du code. Extraire le dossier ZIP là où vous voulez exploiter le projet.



Pour interagir avec GitHub depuis la machine, il faut installer le programme Git localement:

1. **Télécharger le programme** : https://git-scm.com/download/win
2. **Lancer le fichier .exe**
3. **Paramètres d'installation** : Laisser cochée : "Git from the command line and also from 3rd-party software" et choisissez un éditeur (souvent : "Use Visual Studio Code as Git's default editor")

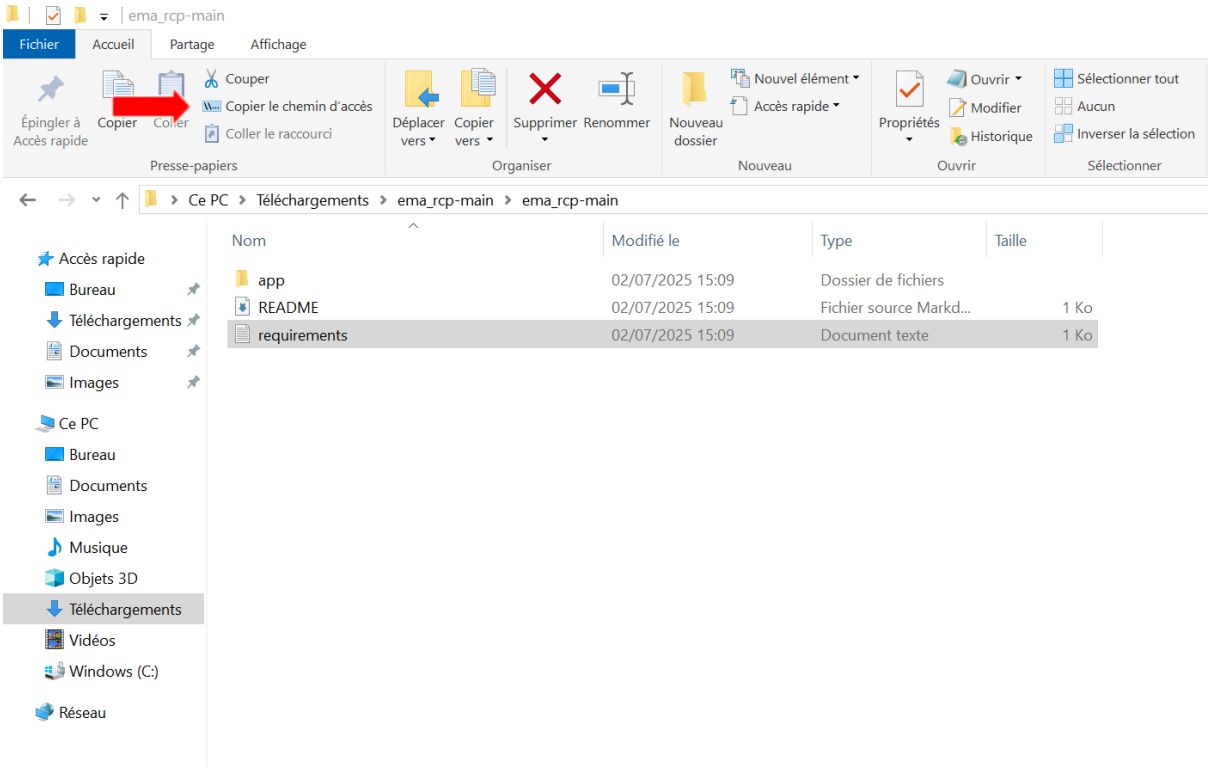
Le fichier **README.md** est un fichier Markdown : un langage très simple qui permet de **mettre en forme du texte**, un peu comme du Word, mais sans logiciel compliqué. Il comporte de façon synthétique les informations pour lancer le code.

Le fichier **requirements.txt** est un fichier texte répertoriant toutes les bibliothèques/ outils utiles au lancement du code (pandas, os,...).

4.4. Installation des modules

Après avoir téléchargé Python et ses fichiers, il faut installer les modules nécessaires au bon fonctionnement du script. Pour ce faire :

1. Copiez le chemin d'accès du fichier requirements.txt



Pour cela, il suffit de sélectionner le fichier et de cliquer sur "Copier le chemin d'accès"

2. Ouvrez l'invite de commande
3. Installez les modules

<u>Première option :</u>
Tout télécharger d'un coup :

- Dans l'invite de commande, tapez : **pip install -r**
- Puis, collez le chemin d'accès du fichier que vous aviez copier

Exemple :
pip install -r "C:\Users\rouajul\Downloads\ema_rcp-main\ema_rcp-main\requirements.txt"

<u>Deuxième option</u> :
Installez chaque module un à un :
Tapez pour chaque module cette commande dans l'invite de commande : **pip install [module name]**

Exemple :
pip install pandas

pip install requests
…

4. Vérifiez l'installation des modules par la commande : **pip list**

Certains modules sont optionnels et utilisés uniquement à des fins de confort ou de test.

Module	A quoi ça sert ?	Indispensable au fonctionnement du script?
pandas	Manipuler les tableaux de données	OUI
requests	Télécharger le fichier Excel	OUI
loguru	Afficher des logs (messages)	OUI
SQLAlchemy	Base de données SQL	NON Optionnel
pytest	Tester automatiquement le code	NON Optionnel
openpyxl	Lire/écrire des fichiers Excel	OUI
aiohttp	Télécharger les PDF en parallèle	OUI
pyright	Vérifications automatiques du code	NON, Utile pour le confort dans un éditeur

Si certains modules n'arrivent pas à se télécharger et qu'ils ne sont pas indispensables, ils peuvent être ignorés.

En cas d'erreur lors de l'installation d'un module, vérifier que l'ordinateur/serveur est bien connecté à Internet, et que pip est à jour en écrivant sur l'invite de commande :

```
python -m pip install --upgrade pip
```

4.5. Lancement initial

Le lancement initial dure environ 5H et permet de télécharger tous les RCP.

- **Manuellement** : ouvrez le terminal de commande sur le pc local ou le serveur et entrez :

```
cmd /c "cd /d {chemin vers le projet} && python app\main.py"
```

Exemple :
cmd /c "cd /d N:\"Scrapping - Julie Rouanet"\ema_rcp-main && python app\main.py"

⚠️**A noter :** Si un **dossier ou un fichier contient des espaces dans son nom**, il est indispensable de l'encadrer de **guillemets** dans la commande (exemple : "Scrapping - Julie Rouanet").

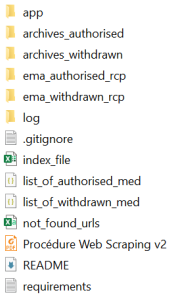
En revanche, si le nom **ne contient pas d'espace** (ex. : Scrapping_Julie_Rouanet), les guillemets ne sont **pas nécessaires**.

Par exemple, la commande suivante est correcte si le chemin ne contient pas d'espaces :
cmd /c "cd /d N:\Scrapping_Julie_Rouanet\ema_rcp-main && python app\main.py"

Le terminal lance le fichier main.py qui est le fichier orchestre de tout le projet.

Le lancement est fait.

Vous devez retrouver dans votre dossier projet ces fichiers ci-contre :



Vérifiez s'il n'y a pas une erreur qui s'est manifestée. Vous pouvez directement voir les messages log sur l'invite de commande ou vous les retrouverez dans le dossier "log".

4.6. Automatisation

L'automatisation permet de mettre à jour les RCP quotidiennement et de télécharger les nouveaux RCP.

1. Ouvrez le planificateur de tâches Windows.

2. Créez une nouvelle tâche, donnez lui un nom.

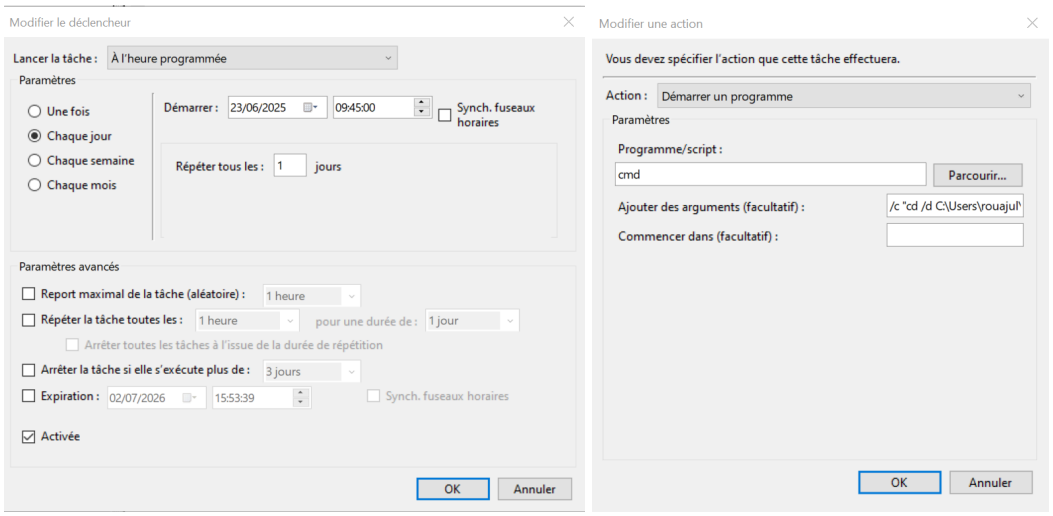
3. Choisissez vos options de sécurité : l'idéal, si vous disposez des droits nécessaires (admin), c'est de cocher les cases **"exécuter même si l'utilisateur n'est pas connecté"** et **"exécuter avec les autorisations maximales"** pour que le script se lance sur l'ordinateur allumé même si personne n'est connecté ou qu'une autre session est ouverte.

   Si le projet est installé sur un **serveur exécutable distant allumé 24h/24**, le script pourra s'exécuter de manière autonome, sans qu'un ordinateur personnel ait besoin d'être allumé ou en veille. Il faudra planifier l'automatisation en se connectant au serveur via la connexion bureau à distance. Pour cela, il faut des droits

administrateurs et un accès au serveur windows et non uniquement à l'espace de stockage partagé.
En revanche, **si le script est exécuté depuis un poste local avec une session dans l'espace de stockage partagé**, il faudra que cet ordinateur reste **allumé** ou en **veille** (avec les bons paramètres de réveil activés).
⚠️ Il ne doit pas être mis en **veille prolongée** (hibernation) ou **éteint**, car cela empêcherait l'exécution automatique du script.

4. Dans l'onglet **Déclencheurs**, cliquez sur "**Nouveau**" et programmer l'heure et la récurrence des lancements.

5. Dans l'onglet "Action",
   - cliquez sur "Nouveau" puis dans "Actions" choisissez **"Démarrer un programme"**,
   - dans Programme/script : écrivez "**cmd**" (sans guillemets),
   - dans "Ajouter des arguments" : écrivez : **/c "cd /d {chemin vers le projet} && python app\main.py".**



Votre planification est prête, lorsque l'heure arrive, une fenêtre noire apparaît (l'invite de commande) et lancera automatiquement le code (fenêtre noire) comme celui du lancement initial et mettra à jour ou téléchargera les RCP si nécessaire.

4.7. Récapitulatif selon le serveur

- Si le script est hébergé sur un **serveur Windows ou Linux exécutable**, toutes les étapes (installation de Python, lancement, automatisation) se font en se connectant au serveur via la **Connexion Bureau à distance (RDP)**.

- Si le projet est stocké sur un **serveur NAS (stockage en réseau)**, l'installation, le lancement et l'automatisation doivent être réalisés depuis un **poste utilisateur** allumé et connecté au NAS.

4.8. Migration de serveur

Il se peut que le projet soit amené à changer de serveur ou de poste, selon les besoins ou l'organisation du réseau. Le processus de transfert dépend du **type de serveur** sur lequel l'installation initiale a été réalisée :

Cas 1 : Installation sur un serveur Windows ou Linux exécutable, qui héberge plusieurs serveurs de partage de fichiers (NAS)

- **Si l'on change de serveur exécutable** (par exemple vers un autre serveur distant capable d'exécuter des scripts), il faudra :
    - réinstaller **Python** et les **modules nécessaires** (via la connexion bureau à distance)
    - reconfigurer le **Planificateur de tâches** (ou équivalent, aussi via la connexion bureau à distance),

- **Si l'on change uniquement de serveur NAS**, c'est-à-dire le répertoire réseau où sont stockés les fichiers :
    - il suffira de **copier-coller** le dossier du projet dans le nouveau NAS,
    - et de **modifier l'argument du Planificateur de tâches** pour pointer vers le nouveau chemin (via la connexion bureau à distance)
    - **Aucune réinstallation de Python ou des modules ne sera nécessaire**, car ceux-ci sont déjà installés sur le serveur exécutable.

Exemple : Migration du serveur NAS N03PharVig au serveur NAS N03PharVig-etud

Chemin du projet dans le serveur initial :
cmd /c "cd /d \\N03PharVig\"Scrapping - Julie Rouanet"\ema_rcp-main && python app\main.py"

- Se connecter au serveur via la **connexion bureau à distance**

- **Copier coller le dossier** dans le serveur entrant N03PharVig-etudiant
  → *Chemin du projet dans le nouveau serveur* : /c "cd /d \\N03PharmVig-etud\"Scrapping - Julie Rouanet"\ema_rcp-main && python app\main.py"

- Modifier l'argument de la tâche planifiée avec le **nouveau chemin d'accès**

Remarque : Si une lettre réseau (ex. : N:) est utilisée dans les deux cas et pointe vers le bon serveur, le chemin peut rester identique. Mais pour éviter toute erreur, le plus fiable est de sélectionner directement le dossier "ema_rcp-main" dans l'explorateur de fichiers et de copier le chemin complet.

Cas 2 : L'installation initiale a été réalisée sur un ordinateur local (poste utilisateur)

Si l'on souhaite changer de **serveur NAS** (notamment en l'absence d'un serveur exécutable accessible), deux situations sont possibles :

1. **Le même PC utilisateur effectue la migration**

Dans ce cas :

- Il suffit de **copier-coller le projet** dans le nouveau NAS,
- Puis de **modifier le chemin d'accès** dans l'onglet **Argument** de la tâche planifiée (dans le Planificateur de tâches Windows),
- Aucun réenregistrement ni réinstallation n'est nécessaire, car l'environnement Python est déjà présent sur ce PC.

2. **Un autre ordinateur souhaite utiliser ou reprendre le projet**

Dans ce cas, le poste doit :

- **disposer de tous les outils nécessaires** : Python, Git (si utilisé), modules installés (pip install ...), accès au réseau, etc.
- Reprendre le projet à partir des **instructions de déploiement** à partir de la **partie 4 : Lancement**
- Adapter les chemins en fonction de la nouvelle localisation du projet

Cela revient à refaire l'installation de l'environnement d'exécution sur un nouvel ordinateur.

5. Maintenance

Il se peut qu'au cours des téléchargements, des cas spéciaux peuvent apparaître. Dans cette partie, nous allons aborder les erreurs que l'on peut rencontrer et comment les réparer ou contourner.

Pour modifier le code, le meilleur moyen est d'installer **Visual Studio Code**, qui permet de :

- naviguer facilement dans les différents fichiers du projet grâce à une interface claire ;
- bénéficier de la coloration syntaxique et de l'autocomplétion pour un meilleur confort de lecture et d'édition du code ;
- exécuter directement des scripts Python via le terminal intégré ;
- utiliser des extensions utiles comme **Python**, **Pylance** ou **Git** pour le versioning ;
- repérer rapidement les erreurs ou les avertissements grâce à l'analyse statique du code.

A chaque modification, aucune erreur doit être détectée avant d'être sauvegardée.

(Facultatif) Le code peut être mis à jour sur le repository Github pour avoir une version téléchargeable à jour disponible pour les autres personnes souhaitant accéder au script.

Pour ce faire, après modifications sur Visual Studio Code, cliquez sur l'onglet Terminal -> New Terminal et entrez les commandes suivantes :

- *git add [le fichier modifié] (exemple : git add app\main.py)*

Appuyez sur entrée

- *git commit -m "commentaires sur la modification que vous venez de faire"*

Appuyez sur entrée

- *git push origin main*

Voilà votre modification est enregistrée et visible sur le repository GitHub.

Pour en savoir plus sur les commandes git : Les commandes git à connaître absolument pour les développeurs

---

5.1. Les headers

Quand le programme envoie une requête à un site web (pour télécharger un fichier), il envoie aussi des **en-têtes**, ou **headers**. Ils permettent de préciser qui envoie la demande et de faire accepter ou mieux traiter la requête par le site. C'est utile car certains sites refusent les requêtes si ces informations ne sont pas présentes ou paraissent suspectes.

Exemple : si le script ne donne pas l'impression d'être un "vrai" navigateur, le site peut bloquer la demande. C'est pour ça qu'on ajoute souvent un header comme le **User-Agent**, qui imite un vrai navigateur.

Il faudra le changer si :
- Le site **refuse** certaines requêtes à cause d'un User-Agent trop **ancien**.
- Il y a besoin de simuler un **autre type** de navigateur ou système.
- Le site détecte les **faux** User-Agent et qu'il faut le rendre plus réaliste.

Dans ces cas là on peut remplacer l'ancien User-Agent par un nouveau

1. Se rendre sur : https://www.whatismybrowser.com/

2. Aller sur le lien en dessous de "Your web browser's unique URL"

3. Copier le texte dans "Their User Agent"

4. Remplacer le texte entre les accolades après les deux points (commençant par Mozilla…) de la variable “headers” dans la fonction download_index dans le fichier download_file.py

```
headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36"}  # noqa: E501
```

5. Sauvegarder le fichier

5.2 Des noms spéciaux

Il se peut que le nom du médicament de l’index_file un fois nettoyé par la fonction clean_name **ne corresponde pas au nom dans l’URL**. Dans ce cas là, le code l'enregistrera en error 404 dans le fichier not_found_file. Si, lors d’une mise à jour, le fichier **not_found_file** est daté du jour, cela signifie qu’un médicament a rencontré un problème d’accès à son URL lors du téléchargement.

Comment y remédier ?

Il y a deux façons de résoudre ce problème :

1. Soit le problème se **répète plus d’une fois** et on peut rajouter une ligne de code qui réglera ce problème dans la fonction clean_name_authorised et clean_name_withdrawn dans le fichier simplify_dataframe (installer visual studio code pour pouvoir le modifier)

Exemple :

- ***Briviact (in Italy: Nubriveo)***

L’URL de ce médicament est : https://www.ema.europa.eu/en/documents/product-information/briviact-epar-product-information_en.pdf

Dans ce cas, le nom à utiliser pour générer l’URL doit être *briviact*, sans inclure le contenu entre parenthèses *(in Italy: Nubriveo)*, car cette mention ne figure pas dans l’URL.

En revanche, pour d’autres médicaments comme *Clopidogrel Teva (hydrogen sulphate)*, le contenu entre parenthèses fait partie du nom de spécialité tel qu’utilisé dans l’URL. Il doit donc être conservé, ce qui donne par exemple : clopidogrel-teva-hydrogen-sulfate.

À ce jour, aucun autre cas n’a été identifié, mais on peut supposer que pour certains médicaments à dénominations nationales variables, la logique restera la même : seules les mentions de type « nom dans un pays » devront être exclues, tandis que les informations chimiques ou galéniques devront être conservées.

On a donc rajouter cette ligne de code :

```
# Supprimer toute la partie entre parenthèses commençant par "in"
    name_edit_authorised = re.sub(r'\(in [^)]+\)', '', name_edit_authorised)
```

- ***Lutetium (177Lu) chloride Billev (previously Illuzyce)***

L'URL de ce médicament est : https://www.ema.europa.eu/en/documents/product-information/lutetium-177lu-chloride-billev-epar-product-information_en.pdf

Dans ce cas, il est nécessaire de conserver l'élément entre parenthèses « (177Lu) », car il fait partie intégrante du nom du médicament. En revanche, les mentions entre parenthèses contenant le mot *previously* doivent être supprimées, car elles ne sont pas utilisées dans la structure de l'URL.

Pour cela, après avoir supprimé les expressions entre parenthèses contenant le mot *previously*, on conserve toutes les autres. Une ligne de code a donc été ajoutée pour appliquer ce traitement sélectif :

```
# Supprimer toutes les parenthèses restantes mais garder leur contenu
   name_edit_authorised = re.sub(r'[()]', '', name_edit_authorised)
```

- ***Bamlanivimab and etesevimab for COVID-19 (statut withdrawn)***

L'URL de la page EMA de ce médicament : https://www.ema.europa.eu/en/medicines/human/EPAR/bamlanivimab-etesevimab-covid-19

Bien que ce médicament ait été retiré du marché et que son RCP ne soit plus disponible sur le site de l'EMA, on observe que l'URL de sa page ne contient pas certaines prépositions comme "and" ou "for".

Afin d'anticiper ce type de situation pour d'autres médicaments dont le nom pourrait inclure des prépositions absentes de l'URL, une ligne de code a été ajoutée pour supprimer automatiquement toutes les prépositions du nom.

```
# Découper en mots et filtrer petits mots inutiles (optionnel)
   words_to_remove = {'a', 'the', 'of', 'and', 'in', 'on', 'for'}
   words = [w for w in name_edit_authorised.split() if w not in words_to_remove]
```

Cette étape permet de filtrer les prépositions sans risquer de tronquer les noms de médicaments valides, comme *Gan**fort***, dont le nom contient une séquence de lettres identique à une préposition mais qui ne doit pas être modifiée.

2. Il se peut que certains noms de médicaments posent problème car **ils ne suivent pas les règles générales de nettoyage** définies dans les fonctions clean_name. Dans ce cas, il s'agit d'une **exception** trop spécifique pour être corrigée par une règle globale, au risque d'impacter négativement d'autres noms. Ces exceptions doivent donc être enregistrées manuellement dans le fichier download_file.py, au sein du dictionnaire prévu à cet effet.

Exemple : ***Budesonide Formoterol Teva Pharma Bv***

Le nom du médicament attendu dans l'URL est : budesonideformoterol-teva

Mais après traitement par la fonction clean_name, le nom devient : Budesonide-formoterol-teva

Ce nom incorrect empêche le téléchargement du fichier correspondant, qui se retrouve alors dans la liste des erreurs 404 (not_found_files).

Pour éviter cela, on n'ajoute pas de règle supplémentaire dans clean_name (comme la suppression des tirets entre les deux premiers mots), car cela pourrait altérer le traitement d'autres noms valides.

À la place, on ajoute ce nom dans le dictionnaire des cas particuliers situé dans le fichier download_file.py, en modifiant le code dans Visual Studio Code.

Pour ajouter un cas, il faut d'abord indiquer le nom généré par la fonction clean_name (par exemple : "Budesonide-formoterol-teva", il est présent dans le fichier not_found_file), suivi de ":", puis du nom correct à utiliser dans l'URL. Ce dernier doit inclure la fin de l'URL permettant un usage direct (par exemple : budesonideformoterol-teva-epar-product-information).

```
SPECIAL_CASES_WITHDRAWN = {
    "Budesonide-formoterol-teva-pharma-bv": "budesonideformoterol-teva-epar-product-information",
    "Pandemic-influenza-vaccine-h5n1-baxter-ag":
"pandemic-influenza-vaccine-h5n1-baxter-epar-product-information",
    "Lamivudine-zidovudine-teva": "lamivudinezidovudine-teva-epar-product-information",
}
```

5.3 Des URL spéciales

Il arrive que certaines erreurs ne proviennent pas du nettoyage du nom du médicament, mais plutôt d'une **structure d'URL non conforme au schéma général** utilisé par l'EMA.

Ces cas seront enregistrés dans le fichier not_found_file du à une erreur **404** (URL introuvable).

Exemple : ***Arikayce-liposomal***

L'URL réelle de son RCP est :

https://www.ema.europa.eu/en/documents/product-information/arikayce-liposomal-product-information_en.pdf

Cette URL **ne suit pas le format standard** attendu, qui est généralement :

https://www.ema.europa.eu/{language}/documents/product-information/{nom}-epar-product-information_{language}.pdf

Dans ce cas précis, l'URL ne contient pas le segment "epar", ce qui empêche le script de la deviner automatiquement.

Comme il est impossible de généraliser ce type d'exception, on ajoute également ces cas particuliers dans la **liste des exceptions** définie dans le fichier download_file.py. On associe alors le nom traité par clean_name à l'URL complète attendue, afin qu'elle soit utilisée directement lors du téléchargement.

```
SPECIAL_CASES_AUTHORISED = {
    "Arikayce-liposomal": "arikayce-liposomal-product-information",
}
```

5.4. Autres

- Si un des fichiers utilisés dans le script (index_file ou not_found_urls) est ouvert sur le poste de travail ; le code ne peut pas l'utiliser, renvoie un message d'erreur et arrête le script. Vérifiez bien que tous les fichiers utilisés par le script soient fermés.

```
2025-07-11 10:10:38.200 | ERROR    | adapters.download_file:download_index:46 - Error: [Errno 13] Permission denied: 'index_file.xlsx'
```

- Lors des tentatives de téléchargement de fichiers PDF depuis le site de l'EMA, le script peut rencontrer une erreur avec le **code HTTP 500**. Cela indique un **problème interne du serveur de l'EMA**, indépendant du script ou de votre configuration. Dans ce cas, il n'y a rien à corriger de votre côté : il suffit **d'attendre que le site redevienne fonctionnel** avant de relancer manuellement le script, ou bien de laisser celui-ci s'exécuter automatiquement lors de la prochaine session planifiée.