Mise en situation 1 (E1)

Collecte, stockage et mise à disposition des données d'un projet IA

Les pesticides ont-ils un impact sur l'incidence du cancer en France ?

Formation Développeur en Intelligence Artificielle
RNCP 37827
Promotion 2024-2026

Maximilien Proust

SIMPLON.CO

académie Orléans-Tours
GRETA VAL DE LOIRE
FORMEZ-VOUS À DEMAIN

SOMMAIRE

Github du projet : https://github.com/Mister-proust/Certification-bloc-1

Introduction

Présentation du projet

Le terme pesticide va désigner les substances actives qui vont contrôler ou éliminer des organismes indésirables. Ces organismes indésirables peuvent être des champignons, des animaux, des plantes ou encore des bactéries. Les pesticides vont être utilisés en majorité pour l'agriculture afin de protéger les cultures. Cependant, on va en retrouver également dans l'usage domestique en tout genre tel que le traitement aux mauvaises herbes, l'élimination de nuisibles ou encore les produits antiparasitaires[1].

L'usage des pesticides de synthèse dans l'agriculture s'est développé de manière considérable depuis leur apparition dans les années 1940-1950. Si on prend en compte les données récentes, depuis les années 1990, la quantité de pesticides utilisée dans le monde a augmenté de 80%. En France, certaines régions utilisent énormément de pesticides, dont les régions du nord, du centre Val-de-Loire, de la Garonne ou encore du Rhône. Mais aucune région ne semble épargnée par l'utilisation des pesticides. Pourtant, en 2008, la France a mis en place le plan Ecophyto dont l'objectif était de réduire de 50% l'utilisation des pesticides d'ici à 2018. Force est de constater que ce plan fût un échec, l'utilisation des pesticides augmentant de 25% sur cette période[2]. L'échec de ces mesures montrent l'incapacité de l'État à agir sur l'utilisation des pesticides. Mais quelles peuvent être les conséquences pour la population française ? Il serait impossible de traiter l'ensemble des conséquences, je vais donc me concentrer sur les conséquences pour le cancer humain.

Chaque individu est constitué d'environ 50 000 milliards de cellules répartis en sous-ensembles structurés appelés tissus qui vont eux-mêmes former des organes. Au sein de chaque organe, les cellules vont avoir des fonctions précises qui peuvent être par exemple la multiplication cellulaire. C'est ce phénomène qui va nous intéresser. En effet, lorsque ce processus de multiplication va se dérégler, l'organisme va détruire les cellules dysfonctionnelles. Cependant, ce mécanisme de protection peut ne plus fonctionner et la cellule malformée va alors se multiplier encore et encore et va devenir une tumeur qui peut être bénigne ou maligne. Les tumeurs dites malignes sont ce que l'on va appeler le cancer[3]. En Annexe 1, vous pouvez retrouver un schéma expliquant cela.

[1] Santé gouv, « Pesticides ».
[2] Boitas, « Etat des lieux de l'usage des pesticides en France ».
[3] Fondation ARC, « Qu'est ce qu'un cancer ? »

L'incidence du cancer en France ne cesse d'augmenter depuis 30 ans. En 2023, on estime à 430 000 le nombre de nouveaux cas en France. Le cancer va représenter la première cause de mortalité prématurée. En 2021, le nombre de décès imputable au cancer était de 162 000[4]. Le cancer, son diagnostic et son traitement sont donc un enjeu de santé public majeur. En effet, même si le taux de guérison progresse, il est important de s'attaquer au problème en amont plutôt qu'à la guérison de celui-ci.

Il existe déjà plusieurs études montrant l'impact des pesticides sur la survenue de divers cancers. Ces études sont très bien documentées dans la review « Cancer health effects of pesticides » qui va recenser les découvertes sur le sujet. On s'aperçoit que les pesticides sont plus ou moins corrélés à l'apparition d'une multitude de cancers. Cependant, certaines études font débat et il est important de noter la difficulté d'associer la survenue d'un cancer à l'exposition aux pesticides[5]. L'objectif en mettant en corrélation des jeux de data autour de l'incidence du cancer et de l'achat de substances chimiques considérées comme cancérigènes, c'est d'observer s'il pourrait y avoir un rapport entre la quantité de pesticides utilisée dans certains départements et la survenue de cancers.

Contexte du projet

L'objectif du projet est de réaliser une API qui va exposer les données sur un temps voulu pour savoir s'il pourrait y avoir un lien potentiel entre des départements utilisant beaucoup de pesticides et d'autres départements où la quantité de pesticides est moindre. Pour cela, j'ai utilisé majoritairement du python dans un environnement virtuel avec quelques requêtes SQL et noSQL. A partir de python, j'ai utilisé le notebook pour visualiser mes données. J'ai pu utiliser plusieurs bibliothèques permettant la visualisation ou la mise à disposition des données comme FastAPI, Pandas, Plotly, SQLAlchemy. Mes données ont été stockées dans une base de données Postgres en local pour toutes les données nécessitant une base de données relationnelle. Les données Postgres ont été récupérées via une API pour certaines données et via des CSV pour la majeure partie. Pour les données en base de données non relationnelle, j'ai utilisé une base MongoDB. Les données de ma base non relationnelle sont issues du scrapping. Vous pouvez observer sur la figure 1 le schéma global du projet en visualisant les différentes sources de données utilisées.

En termes de planification, nous avions 6 semaines pour réaliser le projet.

[4] Fondation ARC, « Le cancer en chiffres ».
[5] Bassil et al., « Cancer health effects of pesticides ».

J'ai découpé mes tâches en 3 parties. La première a été de trouver les différentes sources de données selon mes besoins sur les deux premières semaines. Une fois cette tâche accomplie, la suivante a été de regarder le contenu de mes sources, de les nettoyer pour les mettre en lien les unes avec les autres et cela sur une durée de 2 semaines également. Enfin, j'ai exposé ses données via une application FastAPI permettant de visualiser d'éventuels liens sur les deux dernières semaines. L'objectif était de rendre l'application agréable visuellement pour un utilisateur novice.

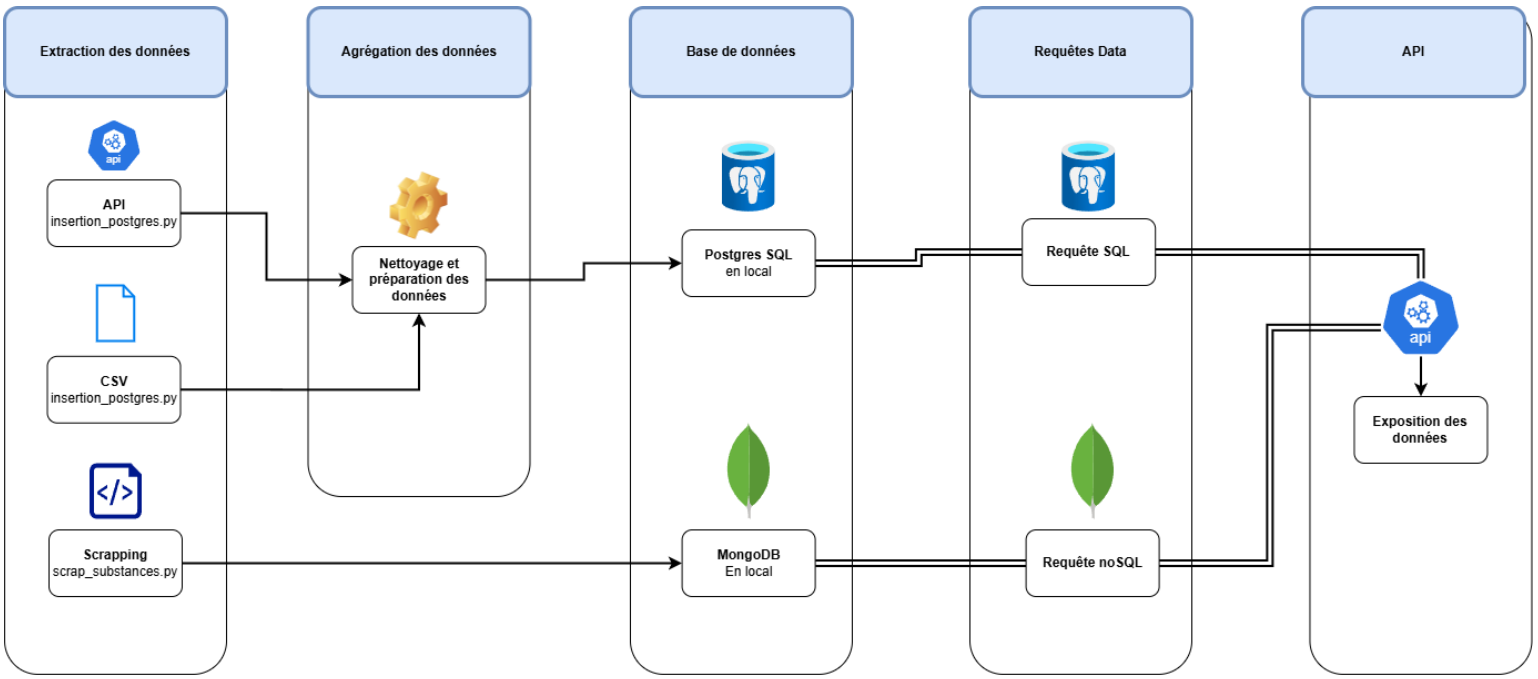


Figure 1 : Schéma de l'extraction à la visualisation des données.

Extraction des données

La première étape du projet a été de trouver des sources qui pourraient permettre d'effectuer un lien entre l'utilisation des pesticides en France et l'incidence des cancers. J'ai pu trouver plusieurs sources intéressantes que je vais présenter ci-dessous.

Données issues d'un service web (API)

L'API (Application Programming Interface) est une interface qui va permettre à deux programmes de communiquer entre eux, notamment pour avoir accès à des données ou à des services en ligne. Ici, j'ai utilisé une API appelée Hubeau (Annexe 2) qui provient du ministère de la transition écologique. Cette source est une source officielle du gouvernement et la qualité du dataset est donc très bonne. De plus, l'API est librement accessible, ce qui est

un atout pour le projet. Les données sont publiques et surtout régies par la licence Etalab. Cette licence signifie qu'on peut réutiliser les données y compris à des fins commerciales à condition de mentionner la source. Dans les conditions d'utilisation, on ne retrouve que quelques points de bonnes pratiques à suivre, tel que l'utilisation de filtre ou la compression des requêtes. Le lien de l'API est le suivant : https://hubeau.eaufrance.fr/page/api-vente-et-achat-de-produits-phytopharmaceutiques.

L'API permet de récupérer les informations par une méthode de pagination. Afin d'éviter une latence élevée, j'ai mis 20 000 lignes par page. En effet, dû à un mécanisme de pagination cumulative, plus il y a de page, plus les données vont être longues à télécharger. Les données récupérées sont des données d'achats de produits phytosanitaires et leurs substances. Je récupère donc 2 jeux de données, l'un pour l'achat de produits et l'autre pour l'achat de substances. J'effectue un premier préfiltre dans lequel je ne récupère que les données départementales. Les bibliothèques python utilisées pour effectuer le nettoyage, la récupération et le stockage en base de données sont Pandas, Request et SQLModel.

Pour la base de données autour des produits phytosanitaires, j'effectue un nettoyage de celui-ci. En effet, après analyse dans un notebook, plusieurs éléments ressortent. Premièrement, il y a beaucoup de départements avec des valeurs égales à 0. Je décide donc de les supprimer, n'ayant pas besoin de 0, représentant surement le total par département. De plus, je convertis les numéros de département dans un format string de façon à n'avoir que des lettres et non des chiffres pour mutualiser l'ensemble de nos données et notamment les départements corses 2A et 2B. Ensuite, je transforme les valeurs AMM (Autorisation de Mise sur le Marché) des produits phytosanitaires de façon à les rendre tous numériques, là aussi dans un souci de cohérence entre les différents jeux de données. Enfin, je reset l'index et j'y ajoute une colonne correspondant à cet index pour en faire ma clé primaire.

Pour le deuxième jeu de donnée autour des substances chimiques, ma première étape a été de récupérer uniquement les valeurs ou la classification était égale à CMR (Cancérogène, Mutagènes ou toxique pour la Reproduction) ou à T, T+, CMR (Nom des CMR avant 2018). En effet, ne travaillant que sur l'incidence des pesticides sur le cancer, les autres substances ne m'intéressent pas. Ensuite, j'ai appliqué les mêmes étapes que pour les produits chimiques à savoir de retirer les 0 des départements, les transformer en string et de transformer en format numérique l'AMM ainsi que d'ajouter comme clé primaire l'index. Avant l'insertion en table, les tables sont créées par SQLModel que vous pouvez retrouver en Annexe 6 tout en sachant qu'il sera explicité plus longuement dans un autre paragraphe. Nous allons donc retrouver les tables "Produits_vente" et "Substance_cmr_vente" dans le schéma "Pollution_cancer". La table "Produits_vente" après renommage va compter 7 colonnes. Dans

l'ordre : "id_produits_vente" qui est la clé primaire, "amm" qui est une clé étrangère, "annee" qui est une clé étrangère, "num_departement" qui est une clé étrangère, "autorise_jardin", "quantite_en_kg", "unite". La table "Substance_cmr_vente" va compter 10 colonnes : "id_substance" qui est la clé primaire, "amm" (clé étrangère), "annee" (clé étrangère), "classification_mention", "code_cas", "code_substance", "num_departement" (clé étrangère), "fonction", "nom_substance" et "quantite_en_kg". Vous pouvez retrouver les détails dans la figure 2 ci-dessous.

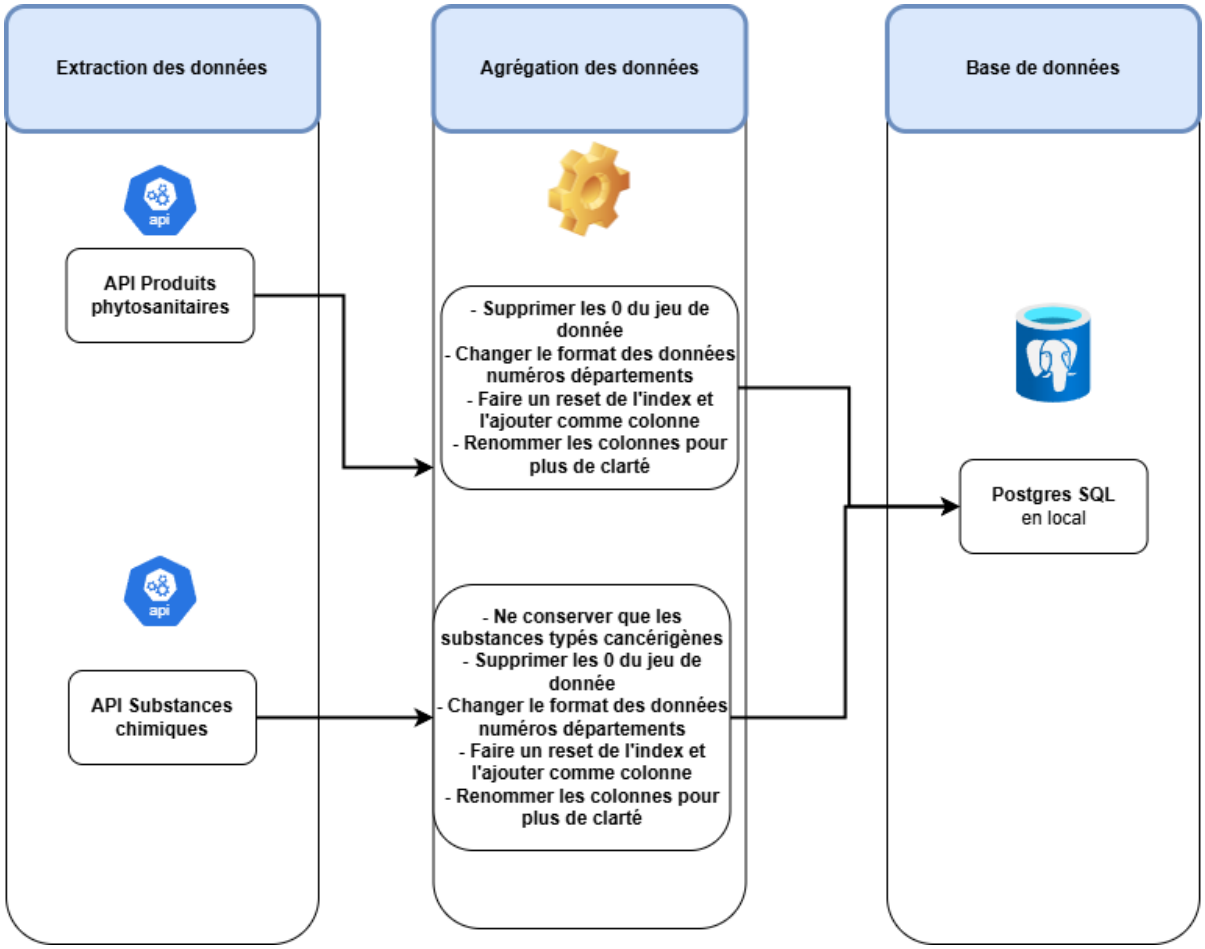


Figure 2: Extraction des données de l'API

Données issues d'une page web

Le scrapping est une méthode permettant de récupérer des balises HTML notamment, avec Beautifulsoup, une bibliothèque open source d'analyse syntaxique de document HTML et XML. Mon choix s'est porté sur un site web comportant des balises HTML intéressantes à capturer, ce site web est sagepesticides.[6]. En effet, il est difficile de trouver des informations supplémentaires sur les molécules que l'on a récupérées avec l'API. Ce site web va répertorier

[6] « https://www.sagepesticides.qc.ca/ ».

les connaissances au sujet de certaines molécules. Le point négatif est le faible nombre de molécules présentes qui est inférieur à 500. J'ai malgré tout choisi ce site web car il autorise la présence de robot comme l'indique le robot.txt qui affiche la mention user : * allow : /. De plus, sur le site on ne retrouve aucune information interdisant la pratique du scrapping. Pour l'extraction, j'ai utilisé un script python avec comme bibliothèques : requests, BeautifulSoup, json, re et logging. L'objectif d'utiliser la bibliothèque json est de permettre un enregistrement des données récupérées dans un fichier .json et ainsi de pouvoir insérer les données dans une base Mongodb. Enfin, j'ai ajouté dans mon script un délai d'une à deux secondes afin de ne pas surcharger le serveur et éviter ainsi les éventuelles saturations.

Au niveau des données récupérées, j'ai choisi de conserver toute la page car selon les analyses futures ou l'exploitation des données, tout pourrait avoir un intérêt pour la compréhension des molécules chimiques et leurs effets néfastes sur l'humain, sur la faune et la flore ainsi que sur la pollution des sols durable. En Annexe 3, vous pouvez retrouver une capture d'écran du site web complet d'une substance recherchée. Le premier bloc, va contenir des informations générales sur la substance. Le reste est composé de tableaux comportant des informations sur le comportement de la substance sur des organismes vivants mais également sur la durée de vie de celle-ci dans les sols et l'eau.

Données issues des fichiers

Les données issues des fichiers seront dans mon dossier uniquement des fichiers CSV. Le fichier CSV est un fichier similaire à un tableur mais les délimiteurs de colonne sont généralement des ";" et les valeurs sont séparés par des ",". Pour insérer et nettoyer ses données, j'ai utilisé 2 bibliothèques : Pandas et SQLModel. Parmi les CSV exploités, je vais les répartir en 3 catégories que je décrirais par la suite. La première catégorie concerne les données démographiques, la deuxième catégorie concerne les données pathologiques françaises et enfin la troisième catégorie concerne les produits phytosanitaires.

Pour la première catégorie, j'ai récupéré les données démographiques sur le catalogue de l'INSEE (Annexe 4) avec le jeu de données "DS_ESTIMATION_POPULATION"[7]. Ce jeu de données va fournir des données sur la population française au 1er janvier de chaque année. Il est couvert par la licence Etalab explicitée plus haut et est donc utilisable. Ce jeu de données s'appelle estimation car les données les plus récentes ne sont pas encore définitives mais provisoires. Cependant, nous n'analyserons pas dans un premier temps ces données récentes étant donné que les autres jeux ne sont pas aussi récents. La première étape a été

[7] « Insee ».

de récupérer dans la colonne “GEO_OBJECT” uniquement les cellules appelées “DEP” car nous ne souhaitons que des données départementales dans le cadre de mon projet. Ensuite, la deuxième étape a été de renommer les colonnes pour que cela soit plus compréhensible dans la base de données. La troisième étape a été d’ajouter un index qui va servir de clé primaire pour cette table. Enfin, la quatrième étape a été de supprimer les 0 devant les départements. En effet, dans un souci d’homogénéisation des données entre les différentes tables, il était indispensable qu’elles aient toutes le même format. Ensuite, les numéros de département sont transformés en string. Après renommage, la table “Effectif_departement” dans le schéma “Pollution_Cancer” va comporter 9 colonnes répartis dans l’ordre suivant : “id_effectif_departement” qui est la clé primaire, “Num_dep” (clé étrangère), “Carac_dep”, “Sexe” (clé étrangère), “Age” (clé étrangère), “Carac_mesure” (clé étrangère), “Chiffre_def” (clé étrangère), “Annee” et “Effectif”.

La deuxième catégorie va correspondre aux données autour des pathologies en France. Ce jeu de données, récupéré via le site d’Ameli[8] (Annexe 5) qui est le site d’assurance maladie en ligne, correspond à des données présentant des informations sur les effectifs de patients pris en charge par l'ensemble des régimes d'assurance maladie. Elles sont disponibles par pathologie, traitement chronique ou épisode de soins, sexe, classe d’âge, région et département de 2015 à 2022. Ce jeu de données est couvert par une licence ODbL qui indique qu’on peut utiliser librement les données à condition de citer les sources de celle-ci. La première étape a été de sélectionner uniquement les colonnes ayant un intérêt pour la suite de mon étude, c’est à dire les années, les numéros de département, les pathologies de niveau 1, 2 et 3 ainsi que le sexe, le nombre de patient et l’effectif total. Une fois ce premier tri effectué, j’ai pris les valeurs indiquant cancer en pathologie 1 et retiré les valeurs nulles sur les pathologies de niveau 2 et 3 correspondants à des totaux que je ne souhaite pas. Pour la troisième étape, j’effectue un changement dans les noms de colonnes, pour améliorer la compréhension de la table. Ensuite, je transforme les effectifs en valeurs numériques. Comme sur les autres tables, j’insère un index qui me servira de clé primaire. Dans un souci d’homogénéisation des données, j’ai modifié les valeurs présentent dans “Sexe” et “Classe_age” de façon à ce que cela soit identique entre toutes les tables et la table de métadonnées dont je parlerais plus tard. Enfin, je retire les 0 devant les départements et je les transforme en string pour éviter les confusions avec les autres tables. Finalement, j’effectue l’insertion en base de données Postgres dans le schéma “Pollution_Cancer” sous le nom de table “Effectif_cancer” avec les noms de colonnes suivant dans l’ordre : “id_effectif_cancer” qui est la clé primaire, “Annee” (clé étrangère), “Pathologie”,

[8] « Pathologies ».

“Type_cancer”, “Suivi_patho”, “Classe_age” (clé étrangère), “Sexe” (clé étrangère), “Departement” (clé étrangère), “Effectif_patients” et “Effectif_total”.

Pour la troisième catégorie, je vais analyser deux tables différentes du jeu de données de l’ANSES concernant les données des produits (produits phytopharmaceutiques, matières fertilisantes et supports de culture, adjuvants, produits mixtes et mélanges) couverts par une autorisation de mise sur le marché (AMM) ou un permis de commerce parallèle (PCP). Ce jeu de données est couvert par la licence Etalab décrite plus haut et on peut donc s’en servir librement. Dans le fichier ZIP, il y a plusieurs tables mais seulement 2 d’entre elles vont nous concerner. Ces 2 fichiers sont le CSV “produits_classe_et_mention_danger” et “produits”. La première table va concerner les produits dangereux et les mentions danger associés ce qui va parfaitement en complément de la table produits qui elle va donner l’ensemble des informations sur les produits phytosanitaires hormis ces mentions dangers qui nous intéresse particulièrement. La première table va subir peu de modifications avec uniquement la suppression d’une colonne nommée “Unnamed: 4”, le renommage des colonnes et l’insertion d’un index pour avoir une clé primaire. Cette table, insérée dans le schéma “Pollution_Cancer” et a comme nom “amm_mention_danger” va contenir 5 colonnes dans l’ordre suivant : “id_amm_danger” comme clé primaire, “amm” (clé étrangère), “Nom_produit”, “Libellé_court” et “Toxicite_produit”. La deuxième table va subir plus de modifications avec dans un premier temps deux colonnes supprimées que sont “type produit” qui ne porte que peu d’intérêt et “Unnamed: 18”. Ensuite, la deuxième étape est le renommage des colonnes dans un souci de compréhension. La troisième étape va modifier le type des colonnes contenant des dates pour les mettre en format date. Enfin, la dernière étape va être de supprimer les amm qui contiennent des valeurs vides ou qui ne sont pas des chiffres (dans de rare cas il va y avoir une combinaison d’amm qui ne nous intéresse pas). Cette table a été insérée dans le schéma “Pollution_Cancer” avec comme nom “amm_produits” et contenant les colonnes suivantes : “amm” qui est la clé primaire, “Nom_produit”, “Seconds_noms_commerciaux”, “titulaire”, “Type_commercial”, “Gamme_usage”, “Mentions_autorisees”, “Restrictions_usage”, “Restrictions_usage_libelle”, “Substances_actives”, “Fonctions”, “Formulations”, “Etat_d_autorisation”, “Date_de_retrait”, “Date_premiere_autorisation”, “Numero_amm_reference” et “Nom_produit_reference”.

Vous pouvez retrouver ci-dessous le schéma représentant les différentes étapes effectuées pour les fichiers CSV.

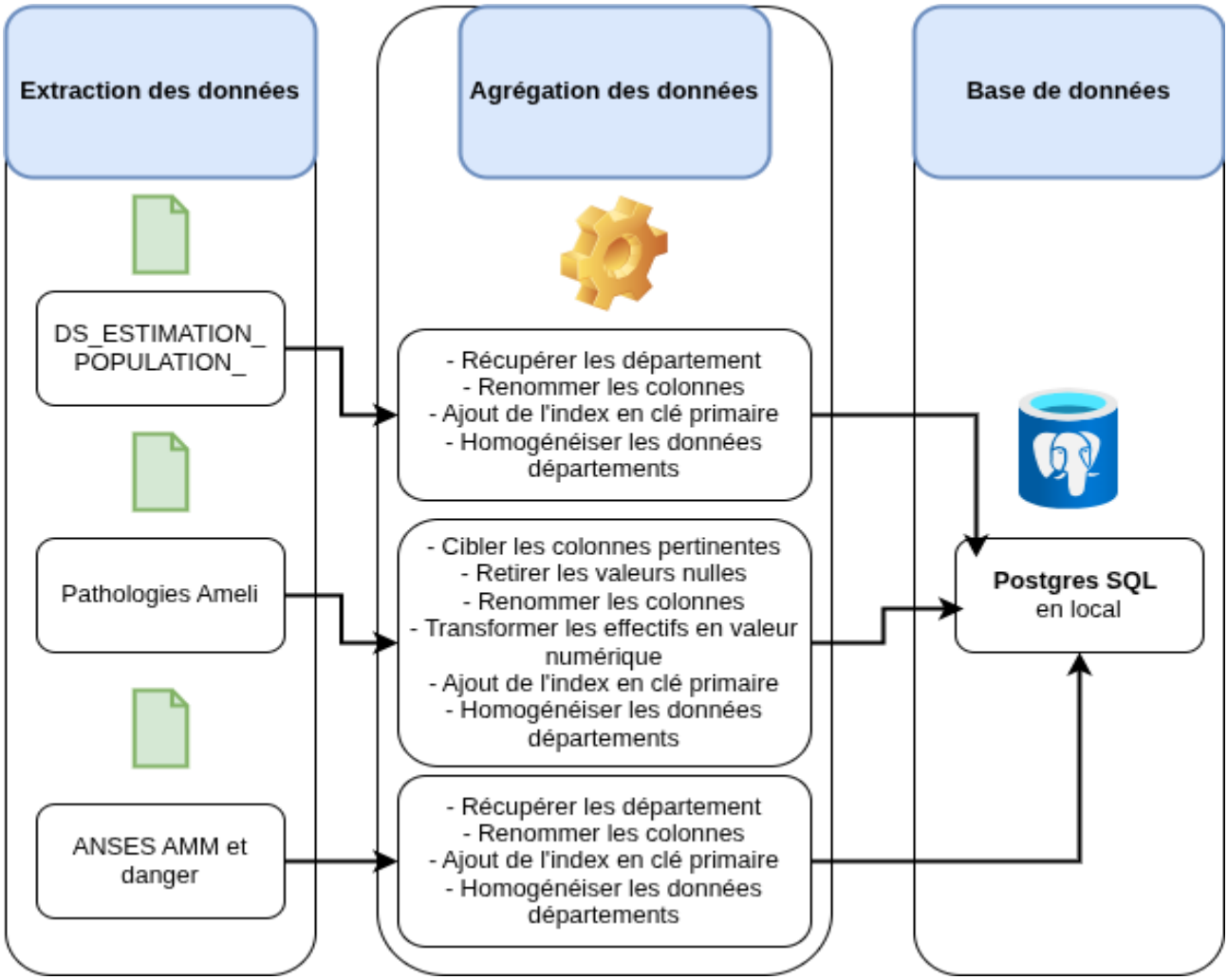


Figure 3: Nettoyage des fichiers CSV avant insertion

Création de la base de données

Pour le projet, j'ai conçu deux types de base de données. Le premier type de base de données est une base de données relationnelle et j'ai choisi Postgres. Le deuxième type de base de données est une base non relationnelle et MongoDB m'a paru une bonne option. Je vais vous présenter mes choix, la construction, le modèle ainsi que la conformité RGPD ci-dessous.

Modélisation des données

J'ai utilisé la méthode Merise afin de structurer mes données et d'organiser les différentes relations entre les entités. Cette méthode a permis de représenter de façon claire les liens entre mes données que je vais exposer ci-dessous :

- Les données démographiques par départements, année, sexe et classe d'âge

- Mis en relation avec les données pathologiques par les 4 composantes présentes également dans cette table.

- Les tables des substances achetées ainsi que des produits achetés sont également reliés par les années et les départements.

- Enfin, il y a une relation entre la table ci-dessus et la table des AMM avec les mentions et dangers du fait de l'AMM.

Bien plus qu'une simple relation, ces tables ont été reliées l'une à l'autre à l'aide de tables que je vais appeler Metadata. Ces Metadata sont des codes permettant d'avoir une homogénéisation parfaite ainsi qu'une compréhension des données par tous. J'ai créé 6 tables de Metadata permettant de faire le lien entre toutes les tables et ainsi assurer la cohérence dans la structure. Ces 6 tables sont les suivantes :

- Metadata_age permettant de relier les tables entre elles avec le même code de classe d'âge.
- Metadata_annee permettant d'aligner les années les unes avec les autres et éviter toute erreur (par exemple insertion de l'année 2050 ne serait pas possible car non présent dans cette table).
- Metadata_departement qui va relier les numéros de département à leurs noms exact pour plus de facilité dans la reconnaissance.
- Metadata_sexe afin de différencier les totaux, des hommes et des femmes.
- Metadata_obs_status permettant de dire si les résultats sont définitifs ou provisoires.
- Metadata_stats pour expliquer les différents codes utilisés dans la table Effectif_departement.

Ces différentes tables reliées les unes aux autres vont définir mon Modèle Conceptuel de données (MCD) et je vais maintenant passer à l'explication de mon modèle physique des données (MPD).

Modèle physique des données (MPD)

Le modèle MDP permet de décrire la structure finale des tables, les types de données, les clés primaires, les clés étrangères ainsi que les contraintes d'intégrité. Un schéma sera bien plus parlant pour exposer ses relations. Vous le retrouverez ci-dessous :

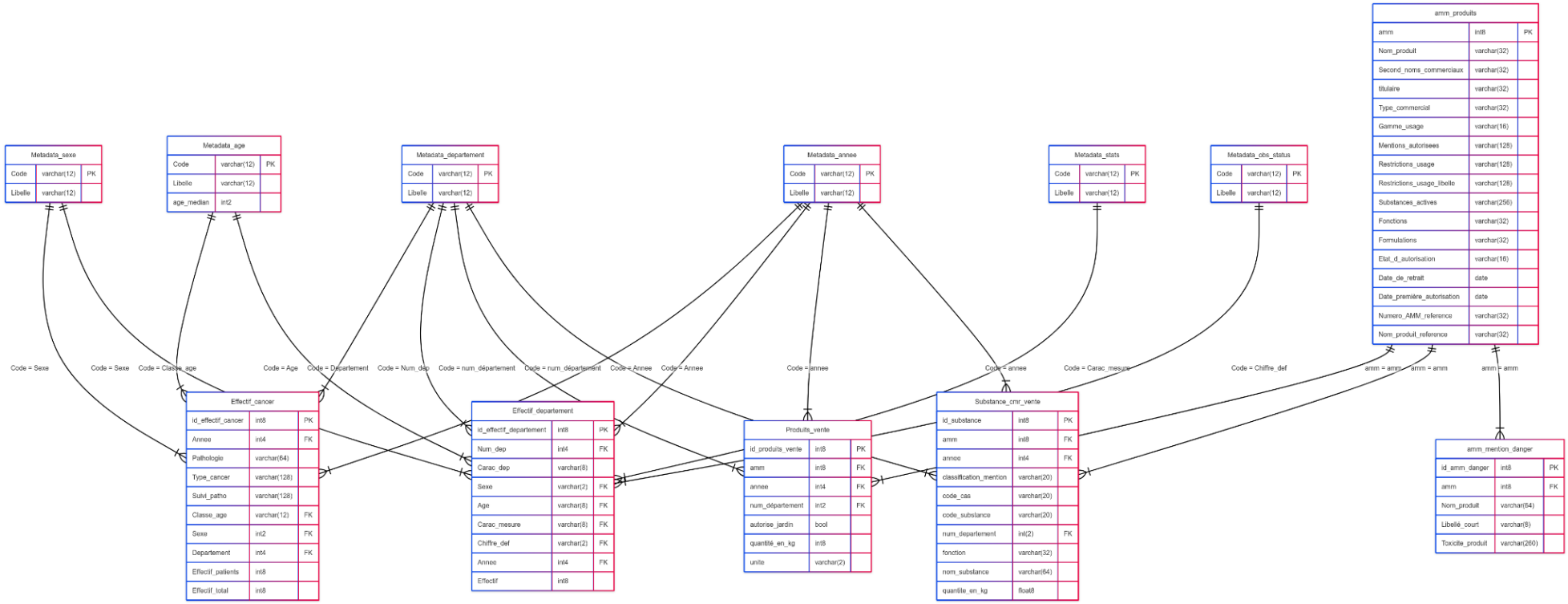


Figure 4 : Représentation des liens entre les tables dans la base de données Postgres

L’ensemble des tables va comporter des clés primaires indispensables à tout système MPD ainsi que des clés étrangères dans certaines tables permettant de les relier les unes aux autres et ainsi d’avoir un intérêt significatif pour la suite du projet.

Pour les données sur les toxicités des substances obtenues par le scraping, j’ai utilisé MongoDB car il permet de stocker facilement les données de documents semi-structurés sous format JSON.

Choix du système de gestion de la base données (SGBD)

Les deux bases de données choisies sont très complémentaires entre elles, ce qui est déjà un point fort.

Tout d’abord, je vais parler de Postgres. Postgres est adapté pour gérer des relations complexes entre les différentes tables, ce qui est le cas pour mon projet (Voir figures présentées précédemment sur le modèle MPD). De plus, nous allons retrouver une certaine facilité de recherche avec l’index, ce qui engendrera des réponses à la base plus rapides. Enfin, Postgres est également connue pour sa très bonne gestion des contraintes, ce qui a permis de rendre robuste le système de table que j’ai créé.

MongoDB est lui réputé pour sa simplicité d’utilisation mais également pour sa grande flexibilité dans la gestion et dans la recherche des différents éléments.

Création de la base de données

La création de la base de données MongoDB a été réalisée à la suite d’un export en fichier json du scrapping puis à la création via l’interface graphique mongoDB de la collection correspondante et de l’insertion du fichier json en locale après création via un Docker.

Pour la base de données Postgres, créée en locale également, elle a été réalisée via un script de mon projet utilisant la bibliothèque SQLModel. Cela permet de déterminer les relations entre les tables, mais également le type des caractères qui vont être insérés dans ces cases et ainsi éviter toute mauvaise manipulation qui entraînera alors une erreur empêchant l’insertion plutôt qu’une erreur dans la base de données. Ci-dessous, je vous mets un exemple de table que j’ai pu insérer dans ma base de données PostgresSQL et vous pouvez retrouver les autres en Annexe 6.

```
class EffectifCancer(SQLModel, table=True):
    __tablename__ = "Effectif_cancer"
    id_effectif_cancer: int = Field(default = None, primary_key=True)
    Annee: int =
Field(foreign_key="Pollution_Cancer.metadata_annee.Code")
    Pathologie: str = Field(max_length = 16)
    Type_cancer: str = Field(max_length = 64)
    Suivi_patho: str = Field(max_length = 64)
    Classe_age: str = Field(max_length = 32,
foreign_key="Pollution_Cancer.metadata_age.Code")
    Sexe: str = Field(max_length = 4,
foreign_key="Pollution_Cancer.metadata_sexe.Code")
    Departement: str =
Field(foreign_key="Pollution_Cancer.metadata_departement.Code",
max_length = 4)
    Effectif_patients: Optional[Decimal] = Field(default=None,
nullable=True)
    Effectif_total: Optional[int] = Field(default=None, nullable=True)
    __table_args__ = {"schema": "Pollution_Cancer"}
```

On observe qu'on peut y mettre le caractère str représentant le type String tout en précisiant la longueur maximale autorisée. On peut y indiquer les clés primaires, ici "id_effectif_cancer" ainsi que de potentielles clés étrangères, ici par exemple "Classe_age".

Conformité RGPD

Les procédures RGPD ont été parfaitement appliquées dans le projet. Les jeux de données peuvent être tous utilisés selon les licences données. De plus, il y a bien une minimisation de l'insertion des données pour n'avoir que celle qui pourrait être nécessaire pour le projet. Les données autour des pathologies sont toutes anonymes et ne permettent aucune identification qu'elle qu'il soit et sont donc conformes aux RGPD.

Concernant le système d'authentification mis en place, les données collectées incluent le nom, le prénom, la date de naissance, un identifiant et un mot de passe. Vous pouvez retrouver en Annexe 7 le script python sur la gestion de l'authentification. Ces données sont considérées comme des données personnelles et bénéficient, à ce titre, d'une protection renforcée :

- Les mots de passes sont hachés et jamais stockés en clair dans la base de données
- L'accès aux données personnelles est restreint et sécurisé

- Les données collectées ne sont utilisées que pour permettre l'accès sécurisé à l'application et ne sont ni partagées, ni revendues à des tiers
- Les utilisateurs peuvent demander la modification ou la suppression de leurs données personnelles via une demande par email à l'administrateur du système
- La conservation des données est limitée dans le temps et adaptée aux besoins fonctionnels du projet (compte supprimé après un an d'inactivité)

Développement de l'API

L'API va permettre de pouvoir exposer les données collectées et traitées afin de les rendre accessibles via des points de terminaisons.

Spécifications fonctionnelles et techniques

L'API a été développé avec le Framework FastAPI basé sur python. J'utilise également SQLAlchemy pour l'authentification avec Passlib et jwt. Pour l'interface MongoDB et l'API, j'utilise Pymongo. Enfin, pour la relation Postgres/API, j'utilise Psycopg2. Ci-dessous, vous trouverez le schéma fonctionnel de l'API avec les différents points de terminaisons ainsi que la sécurisation de ceux-ci via un cadenas. Pour les points de terminaison liés à l'authentification, j'ai choisi un stockage des identifiants en base de données Postgres locale avec mot de passe hashé. L'utilisateur a la possibilité de créer des identifiants via le point de terminaison /inscription et ainsi permettre son accès aux pages suivantes.

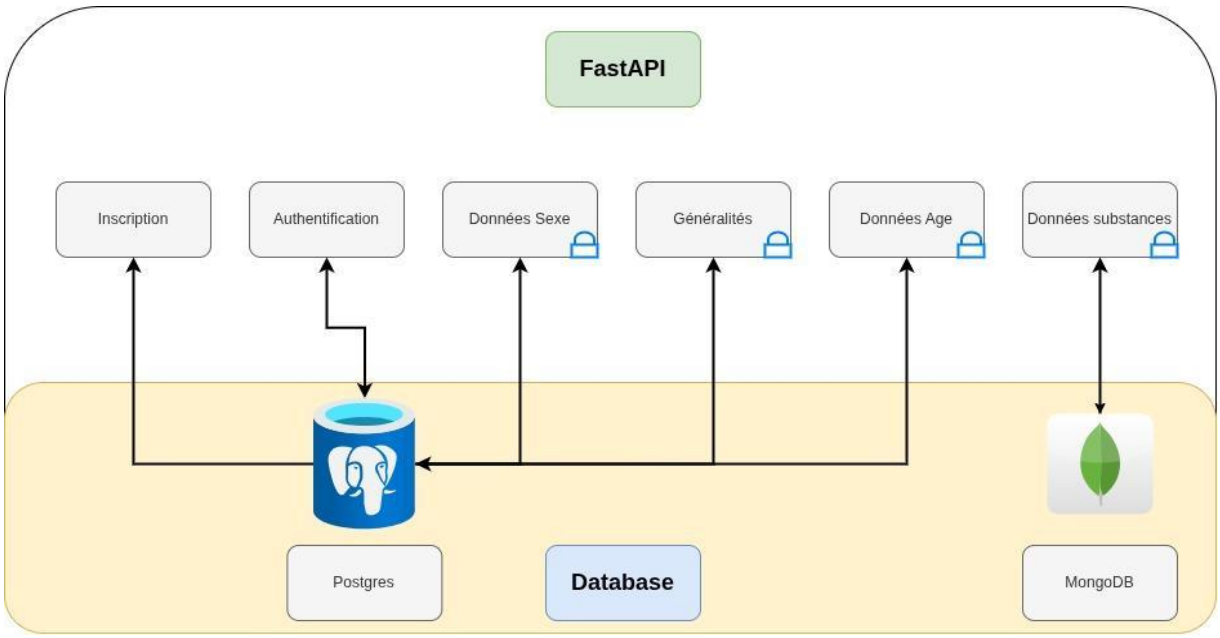


Figure 5 : Schéma fonctionnel de l'API

Conception de l'architecture de l'API

L'API est documentée avec le Swagger que vous pouvez retrouver ci-dessous :

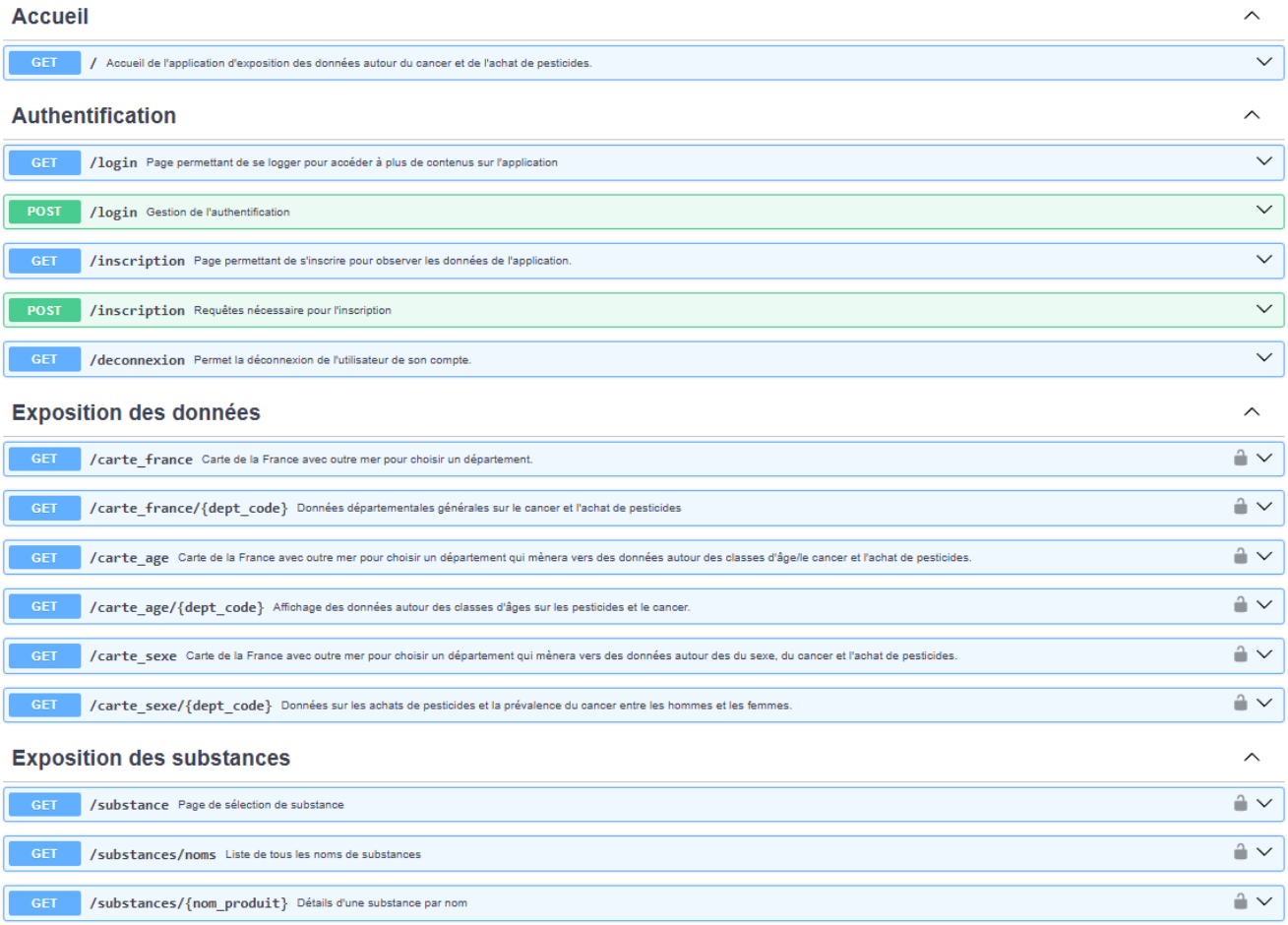


Figure 6 : Représentation du Swagger de l'API

L'API possède 6 points de terminaison. Les deux premiers points de terminaison vont être réservés à l'authentification ou à la création d'un compte et sont donc libres d'accès. Ensuite, on peut répartir en 2 catégories les points de terminaison, les points qui vont accéder à Postgres et le point qui va accéder à MongoDB. L'ensemble de ces points de terminaison ont besoin d'une authentification pour être accessible à l'utilisateur.

Les points de terminaison qui vont accéder aux Données Sexe, Généralités et Données Âge vont permettre à l'utilisateur d'afficher les données en lien avec le point de terminaison correspondant. Ces points de terminaison vont nécessiter l'utilisation de requêtes SQL. Pour cela, un fichier va se connecter à la base de données Postgres. Un autre fichier va appeler cette connexion et dedans on y insère les fonctions qui vont aller chercher dans les tables souhaitées les informations voulues.

Le point de terminaison sur les généralités va mettre en avant les cancers tout âge et tout sexe ainsi que le poids en kilos de substances CMR vendues sur un département donné. Cela permet d'avoir une vision globale et permet de visualiser un potentiel lien entre quantité de substances CMR vendues et la prévalence d'un cancer au sein d'une population d'un département donné. Il est possible de choisir le type de cancer qu'on souhaite observer ainsi que l'année, permettant un ciblage plus spécifique au besoin.

Pour le point de terminaison sur les données Sexe, on va avoir les mêmes observations que pour les généralités, à la différence près qu'on peut choisir le sexe des individus, permettant ainsi d'observer s'il peut y avoir un impact sur la prévalence de certains cancers en fonction du genre.

Enfin, le dernier point de terminaison sur les données Âge va permettre de catégoriser par classe d'âge la prévalence des cancers et ainsi observer si une catégorie d'âge est plus ou moins touchée.

En ce qui concerne le point de terminaison mongoDB, l'utilisation de pymongo va permettre de se connecter à la base mongoDB et ainsi permettre d'effectuer les requêtes via des fonctions de recherche pour récupérer les informations sur les substances voulues. Le point de terminaison va afficher l'ensemble des informations sur une substance donnée si l'utilisateur le souhaite. A terme, cette fonction pourrait être largement améliorée pour être en lien avec les données Postgres. Vous pouvez retrouver en Annexe 8 l'architecture de l'application.

Requêtes de type SQL

Les requêtes SQL pour les 3 points de terminaisons vont avoir le même principe, aller chercher les informations dans le schéma "Pollution_Cancer", puis récupérer des informations directement dans la table "Effectif_cancer" qui correspond à la table majeure. De cette table, on va effectuer une jointure avec la table "Effectif_departement" suivie d'une jointure avec la table "Metadata_departement" et une dernière jointure avec la table "Substance_cmr_vente". Ces 4 jointures vont permettre de récupérer certaines informations qu'on va exposer dans l'API. En plus de cela, un calcul de prévalence va être réalisé à l'aide des tables "Effectif_cancer" et "Effectif_departement". Les seules différences entre les points de terminaison vont se retrouver dans le WHERE ou il y aura des sélections différentes selon les données exposées souhaitées. Afin d'éviter une surcharge dans cette partie, les 3 requêtes SQL ainsi que les premières lignes des résultats sont exposées en annexe 9, 10 et 11.

Requêtes depuis un système big data

Les requêtes dans le cadre de mon API ont été “transformées” en fonction. Cependant, voici les requêtes tel qu’on aurait pu les écrire en langage noSQL :

- Pour la recherche de substance, une première requête type :

{ "nom": "nom de la substance souhaitée" }

- Pour des recherches plus avancées avec par exemple des informations sur le tableau 3, il faudrait conserver la première requête ci-dessus, y ajouter une projection et y écrire dedans :

Filtre :
{ "nom": "nom de la substance souhaitée" }

Projection :
{
  "tableaux.2.structure.Devenir et comportement dans l'environnement.Potentiel de lessivage.Niveau": 1,
  "_id": 0
}

Perspectives et améliorations

L’objectif premier du projet est avant tout de pouvoir observer une corrélation entre l’achat de produits phytosanitaires et la prévalence des cancers. Il y a évidemment plusieurs points d’amélioration techniques que je vais lister ci-dessous :

- Mise en place de test unitaire et d’intégration afin de sécuriser le code et anticiper toutes les erreurs possibles
- Améliorer la gestion des erreurs et des exceptions
- Optimiser les temps de requêtes notamment lorsque les données ont un volume important même si pour l’instant ce n’est pas le cas.
- Améliorer la visualisation des données et augmenter l’interaction des données entre elles qui sont actuellement trop restreinte
- Intégrer des liens entre les informations des substances CMR achetées et les informations produites plus précises

Enfin, il y a plusieurs perspectives intéressantes à mettre en place pour permettre notamment un futur déploiement en application :

- Migrer les données dans une base Cloud pour permettre de déployer l’application

- Conteneuriser l'application avec Docker pour assurer la portabilité
- Intégrer des années supplémentaires, le champ est actuellement trop restreint (2015-2022) mais il n'existe pas de données officielles open source permettant de faire mieux
- Intégrer des données pathologiques plus précise que par département et donc un partenariat avec le ministère est obligatoire car l'anonymisation ne serait plus possible
- Intégrer des données socio-démographiques pour observer quels secteurs sont les plus touchés par le cancer
- Intégration des données autour de la pollution plus large pour améliorer la visualisation ou le lien entre pollution et cancer

Conclusion

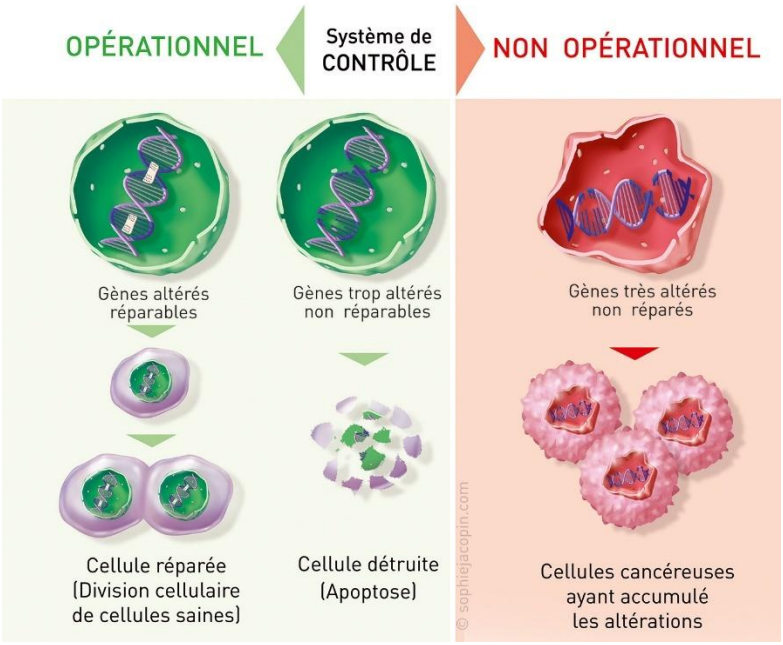
Ce projet a permis de mettre en place une API sécurisée connectée à deux bases de données distinctes. L'objectif initial, affiché dans l'introduction, n'est que partiellement atteint étant donné qu'aucun lien n'a pu être déterminé ni même envisagé à la vue des premières requêtes effectuées.

Il faudrait dans un premier temps développer une application disponible au public ainsi que la création de plusieurs modèles d'Intelligence Artificielle. Le premier modèle qui pourrait être mis en place serait de chercher une corrélation entre une hausse subtile mais visible de la prévalence de certains cancers et de certains produits phytosanitaires. Si certains produits, même non catégorisées CMR venaient à avoir un lien potentiel, cela permettrait de faire évoluer les réglementations en vigueur. En effet, en 2018, le retrait de la vente de plusieurs produits catégorisés CMR s'est immédiatement ressenti dans les différents graphiques observés. Le deuxième modèle pertinent à mettre en place serait une corrélation entre âge et apparition de certains cancers pour déclencher des rendez-vous médicaux de prévention. En effet, plus le cancer est découvert tôt, plus les chances de guérison sont élevées.

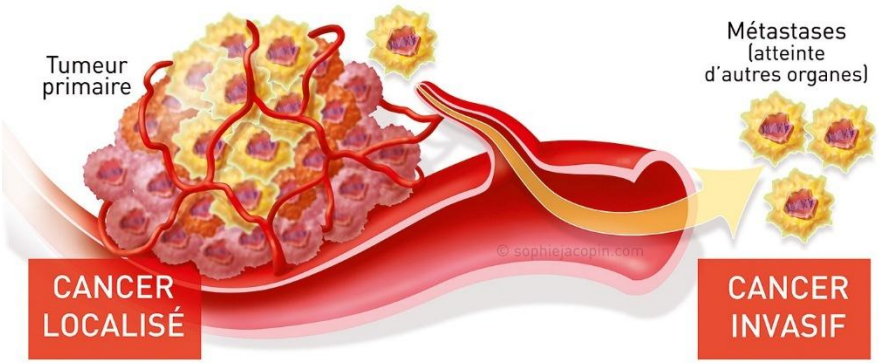
Finalement, le travail accompli pour ce projet n'est que la première étape d'un projet beaucoup plus ambitieux, permettant d'améliorer la santé en France, ainsi que d'avoir un impact non négligeable sur l'écosystème. Les bases de données étant construites, l'étape suivante consistera à approfondir l'analyse des paramètres collectés pour en tirer des conclusions exploitables et potentiellement déterminantes.

Annexes

Annexe 1 : Le cancer expliqué en schéma (via fondation-arc.org)



Ci-dessus, on constate qu'un système de contrôle efficace va empêcher la survenue de cellules tumorale jusqu'à un déréglage de ce système. Ci-dessous, l'évolution d'un cancer localisé à un cancer invasif grâce notamment au microenvironnement qui va permettre aux cellules cancéreuses de circuler.



Annexe 2 : Page web de l’API Hub’eau

Annexe 3 : Visualisation du résultat de la recherche d’une substance chimique sur le site web : https://www.sagepesticides.qc.ca

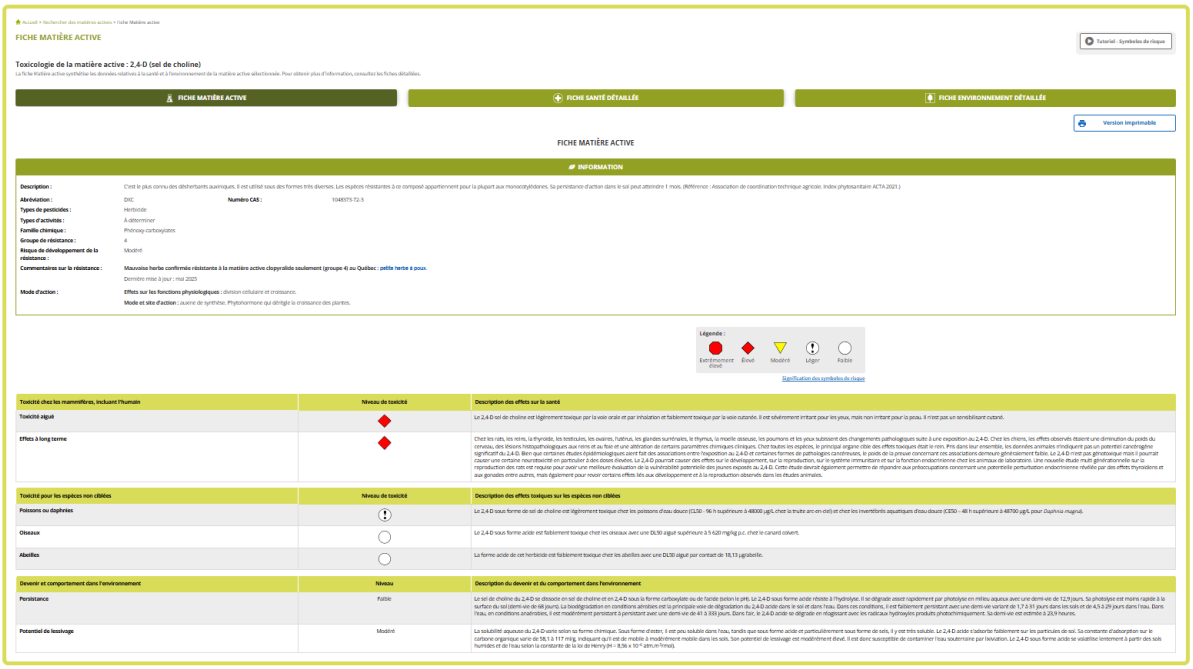


Figure 7 : Visualisation des données obtenues sur la recherche d'une substance

Annexe 4 : Page de l'INSEE de récupération du CSV

Annexe 5 : Page de récupération du CSV sur les données cancer avec Ameli

Annexe 6 : Code de création des tables en base de données relationnelles.

```python
class MetadataAge(SQLModel, table=True):
    __tablename__="metadata_age"
    Code : str = Field(max_length = 32, primary_key=True)
    Libelle : str = Field(max_length=64)
    Age_median : Optional[int] = Field(nullable=True)
    __table_args__ = {"schema": "Pollution_Cancer"}
    def __repr__(self):
        return f"MetadataAge(id={self.code}, libelle={self.Libelle})"

class MetadataAnnee(SQLModel, table=True):
    __tablename__="metadata_annee"
    Code : int = Field(primary_key=True)
    Libelle : str = Field(max_length=64)
    __table_args__ = {"schema": "Pollution_Cancer"}
    def __repr__(self):
        return f"MetadataAnnee(id={self.code}, libelle={self.Libelle})"

class MetadataSexe(SQLModel, table=True):
    __tablename__="metadata_sexe"
    Code : str = Field(max_length = 32, primary_key=True)
    Libelle : str = Field(max_length=64)
    __table_args__ = {"schema": "Pollution_Cancer"}
    def __repr__(self):
        return f"MetadataSexe(id={self.code}, libelle={self.Libelle})"

class MetadataDepartement(SQLModel, table=True):
    __tablename__="metadata_departement"
    Code : str = Field(max_length = 32, primary_key=True)
    Libelle : str = Field(max_length=64)
    __table_args__ = {"schema": "Pollution_Cancer"}
    def __repr__(self):
        return f"MetadataDepartement(id={self.code}, libelle={self.Libelle})"


class MetadataObsStatus(SQLModel, table=True):
    __tablename__="metadata_obs_status"
    Code : str = Field(max_length = 32, primary_key=True)
    Libelle : str = Field(max_length=64)
    __table_args__ = {"schema": "Pollution_Cancer"}
    def __repr__(self):
        return f"MetadataObsStatus(id={self.code}, libelle={self.Libelle})"

class MetadataStats(SQLModel, table=True):
    __tablename__="metadata_stats"
    Code : str = Field(max_length = 32, primary_key=True)
    Libelle : str = Field(max_length=256)
    __table_args__ = {"schema": "Pollution_Cancer"}
    def __repr__(self):
        return f"MetadataStats(id={self.code}, libelle={self.Libelle})"

class EffectifCancer(SQLModel, table=True):
    __tablename__ = "Effectif_cancer"
    id_effectif_cancer: int = Field(default = None, primary_key=True)
    Annee: int = Field(foreign_key="Pollution_Cancer.metadata_annee.Code")
    Pathologie: str = Field(max_length = 16)
    Type_cancer: str = Field(max_length = 64)
    Suivi_patho: str = Field(max_length = 64)
    Classe_age: str = Field(max_length = 32, foreign_key="Pollution_Cancer.metadata_age.Code")
    Sexe: str = Field(max_length = 4, foreign_key="Pollution_Cancer.metadata_sexe.Code")
    Departement: str = Field(foreign_key="Pollution_Cancer.metadata_departement.Code", max_length = 4)
    Effectif_patients: Optional[Decimal] = Field(default=None, nullable=True)
    Effectif_total: Optional[int] = Field(default=None, nullable=True)
    __table_args__ = {"schema": "Pollution_Cancer"}
    def __repr__(self):
        return f"EffectifCancer(id={self.id_effectif_cancer}, Annee={self.Annee}, Departement={self.Departement})"
```

Suite de l'annexe 6 : Code de création des tables en base de données relationnelles.

```python
class EffectifDepartement(SQLModel, table=True):
    __tablename__="Effectif_departement"
    id_effectif_departement : int = Field(default = None, primary_key=True)
    Num_dep : str=Field(foreign_key="Pollution_Cancer.metadata_departement.Code", max_length = 12)
    Carac_dep : Optional[str] = Field(default= None, max_length = 8, nullable=True)
    Sexe : str = Field(max_length = 2, foreign_key="Pollution_Cancer.metadata_sexe.Code")
    Age : str = Field(max_length = 16, foreign_key="Pollution_Cancer.metadata_age.Code")
    Carac_mesure : Optional[str] = Field(default= None, max_length = 32, foreign_key="Pollution_Cancer.metadata_stats.Code", nullable=True)
    Chiffre_def : Optional[str] = Field(default= None, max_length = 16, foreign_key="Pollution_Cancer.metadata_obs_status.Code", nullable=True)
    Annee : int = Field(default= None)
    Effectif : Optional[Decimal] = Field(default= None, nullable=True)
    __table_args__ = {"schema": "Pollution_Cancer"}
    def __repr__(self):
        return f"EffectifDepartement(id={self.id_effectif_departement}, Num_dep={self.Num_dep}, Annee={self.Annee})"


class ProduitsVente(SQLModel, table=True):
    __tablename__="Produits_vente"
    id_produits_vente: int = Field(default = None, primary_key=True)
    amm: int = Field(foreign_key="Pollution_Cancer.amm_produits.amm")
    annee: int = Field(default= None, foreign_key="Pollution_Cancer.metadata_annee.Code")
    num_departement: str=Field(foreign_key="Pollution_Cancer.metadata_departement.Code", max_length = 4)
    autorise_jardin : Optional[str]=Field(default=None, max_length=8, nullable=True)
    quantite_en_kg : Optional[Decimal] = Field(default=None, nullable=True)
    unite: Optional[str] = Field(default= None, max_length = 2, nullable=True)
    __table_args__ = {"schema": "Pollution_Cancer"}
    def __repr__(self):
        return f"Produitsvente(id={self.id_produits_vente}, amm={self.amm}, quantité_en_kg={self.quantité_en_kg})"
class SubstanceCMRVente(SQLModel, table=True):
    __tablename__="Substance_cmr_vente"
    id_substance: int = Field(default = None, primary_key=True)
    amm: int = Field(foreign_key="Pollution_Cancer.amm_produits.amm")
    annee : int = Field(default= None, foreign_key="Pollution_Cancer.metadata_annee.Code")
    classification_mention: Optional[str]=Field(default= None, max_length = 20, nullable=True)
    code_cas: Optional[str]=Field(default= None, max_length = 32, nullable=True)
    code_substance: Optional[int]=Field(default= None, nullable=True)
    num_departement: str=Field(foreign_key="Pollution_Cancer.metadata_departement.Code", max_length = 4)
    fonction: Optional[str]=Field(default= None, max_length = 128, nullable=True)
    nom_substance: Optional[str]=Field(default= None, max_length = 256, nullable=True)
    quantite_en_kg: Optional[Decimal] = Field(default=None, nullable=True)
    __table_args__ = {"schema": "Pollution_Cancer"}
    def __repr__(self):
        return f"Substancecmrvente(id={self.id_substance}, amm={self.amm}, classification_mention={self.classification_mention})"


class AmmMentionDanger(SQLModel, table=True):
    __tablename__="amm_mention_danger"
    id_amm_danger : int = Field(default = None, primary_key=True)
    amm: int = Field(foreign_key="Pollution_Cancer.amm_produits.amm")
    Nom_produit: Optional[str]=Field(default= None, max_length = 64, nullable=True)
    Libellé_court: Optional[str]=Field(default= None, max_length = 36, nullable=True)
    Toxicite_produit: Optional[str]=Field(default= None, max_length = 260, nullable=True)
    __table_args__ = {"schema": "Pollution_Cancer"}
    def __repr__(self):
        return f"Ammmentiondanger(id={self.id_amm_danger}, amm={self.amm}, Toxicite_produit={self.Toxicite_produit})"


class AmmProduits(SQLModel, table=True):
    __tablename__="amm_produits"
    amm: int = Field(primary_key=True)
    Nom_produit: Optional[str]=Field(default= None, max_length = 64, nullable=True)
    Second_noms_commerciaux: Optional[str]=Field(default= None, max_length = 512, nullable=True)
    titulaire: Optional[str]=Field(default= None, max_length = 124, nullable=True)
    Type_commercial: Optional[str]=Field(default= None, max_length = 64, nullable=True)
    Gamme_usage: Optional[str]=Field(default= None, max_length = 128, nullable=True)
    Mentions_autorisees: Optional[str]=Field(default= None, max_length = 128, nullable=True)
    Restrictions_usage: Optional[str]=Field(default= None, max_length = 128, nullable=True)
    Restrictions_usage_libelle: Optional[str]=Field(default= None, max_length = 1024, nullable=True)
    Substances_actives: Optional[str]=Field(default= None, max_length = 512, nullable=True)
    Fonctions: Optional[str]=Field(default= None, max_length = 256, nullable=True)
    Formulations: Optional[str]=Field(default= None, max_length = 128, nullable=True)
    Etat_d_autorisation: Optional[str]=Field(default= None, max_length = 32, nullable=True)
    Date_de_retrait: Optional[date] = Field(default=None, nullable=True)
    Date_première_autorisation: Optional[date] = Field(default=None, nullable=True)
    Numero_AMM_reference:Optional[str]=Field(default= None, max_length = 64, nullable=True)
    Nom_produit_reference: Optional[str]=Field(default= None, max_length = 128, nullable=True)
    __table_args__ = {"schema": "Pollution_Cancer"}
    def __repr__(self):
        return f"Ammproduits(amm={self.amm}, Nom_produit={self.Nom_produit}, Substances_actives={self.Substances_actives})"
```

Annexe 7 : Script sur l'authentification

```
# Initialisation SQLAlchemy
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

class Utilisateurs(Base):
    """
    Modèle de table SQLAlchemy pour les utilisateurs.
    """
    __tablename__ = "Utilisateurs"
    __table_args__ = {"schema": "Authentification"}

    email = Column(String, unique=True,primary_key=True, index=True)
    hashed_password = Column(String)
    nom = Column(String, nullable=False)
    prenom = Column(String, nullable=False)
    date_naissance = Column(Date, nullable=False)
    is_active = Column(Boolean, default=True)
    last_activity = Column(Date, default=datetime.utcnow)

def create_tables():
    """
    Crée les tables dans la base de données si elles n'existent pas.
    """
    Base.metadata.create_all(bind=engine)

# Configuration du gestionnaire de mots de passe avec bcrypt
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

def verify_password(plain_password, hashed_password):
    """
    Vérifie qu'un mot de passe en clair correspond à son hash.

    Args:
        plain_password (str): Mot de passe fourni par l'utilisateur.
        hashed_password (str): Mot de passe hashé stocké en base.

    Retourne:
        bool: True si les mots de passe correspondent, sinon False.
    """
    return pwd_context.verify(plain_password, hashed_password)

def get_password_hash(password):
    """
    Hash un mot de passe en utilisant bcrypt.

    Args:
        password (str): Mot de passe en clair.

    Retourne:
        str: Mot de passe hashé.
    """
    return pwd_context.hash(password)
```

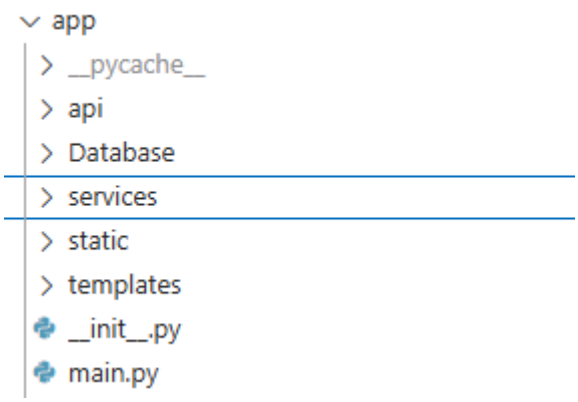
```python
def get_db():
    """
    Fournit une session SQLAlchemy pour interagir avec la base de données.

    Yields:
        Session: Instance de session active.
    """
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

# Configuration OAuth2 avec FastAPI
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/login", auto_error=False)

def get_user(db: Session, email: str):
    """
    Récupère un utilisateur via son email.

    Args:
        db (Session): Session SQLAlchemy.
        email (str): Email de l'utilisateur recherché.

    Retourne:
        Utilisateurs ou None: Instance de l'utilisateur ou None s'il n'existe pas.
    """
    return db.query(Utilisateurs).filter(Utilisateurs.email == email).first()

def authenticate_user(db: Session, email: str, password: str):
    """
    Authentifie un utilisateur en vérifiant son email et son mot de passe.

    Args:
        db (Session): Session SQLAlchemy.
        email (str): Email de l'utilisateur.
        password (str): Mot de passe en clair.

    Retourne:
        Utilisateurs ou None: L'utilisateur authentifié ou None si l'authentification échoue.
    """
    user = get_user(db, email)
    if not user or not verify_password(password, user.hashed_password):
        return None
    user.last_activity = datetime.utcnow()
    db.commit()
    return user
```

```python
def create_access_token(data: dict, expires_delta: Optional[timedelta] = None):
    """
    Crée un token JWT d'accès pour un utilisateur.

    Args:
        data (dict): Données à encoder dans le token.
        expires_delta (timedelta, optionnel): Durée de validité du token.

    Retourne:
        str: Token JWT.
    """
    to_encode = data.copy()
    expire = datetime.utcnow() + (expires_delta or timedelta(minutes=15))
    to_encode.update({"exp": expire})
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
```

```python
async def get_current_user(
    request: Request,
    db: Session = Depends(get_db),
    token: str = Depends(oauth2_scheme)
):
    """
    Récupère l'utilisateur actuellement connecté via le token JWT.

    Args:
        request (Request): Requête FastAPI.
        db (Session): Session SQLAlchemy.
        token (str): Token JWT récupéré via OAuth2 ou les cookies.

    Retourne:
        Utilisateurs: L'utilisateur authentifié.

    Lève:
        HTTPException: Si le token est invalide ou si l'utilisateur n'existe pas.
    """

    credentials_exception = HTTPException(
        status_code=401,
        detail="Identification impossible",
        headers={"WWW-Authenticate": "Bearer"},
    )

    actual_token = token or request.cookies.get("access_token", "").removeprefix("Bearer ")

    if not actual_token:
        raise credentials_exception
```

```
    try:
        payload = jwt.decode(actual_token, SECRET_KEY, algorithms=[ALGORITHM])
        email: str = payload.get("sub")
        if not email:
            raise credentials_exception
    except JWTError:
        raise credentials_exception

    user = get_user(db, email=email)
    if not user:
        raise credentials_exception
    return user

def delete_inactive_users():
    """
    Supprime les utilisateurs inactifs depuis plus d'un an.

    Cette fonction parcourt les utilisateurs et supprime ceux dont la dernière activité
    remonte à plus d'un an.
    """
    db = SessionLocal()
    one_year_ago = datetime.utcnow() - timedelta(days=365)
    inactive_users = db.query(Utilisateurs).filter(Utilisateurs.last_activity < one_year_ago).all()

    for user in inactive_users:
        db.delete(user)
    db.commit()
    db.close()

# Création automatique des tables lors de l'importation du fichier
create_tables()
```

Annexe 8 : Architecture de l'API

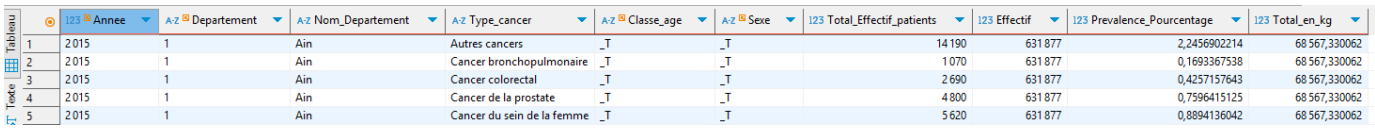


L'API afin d'être plus simple pour un utilisateur novice va avoir une interface graphique et est donc une application. On va retrouver les routes dans le dossier api, les informations de connexion à la base de données dans la Database. Dans les services, on va retrouver les fichiers d'authentifications ainsi que les requêtes MongoDB. Enfin, dans les dossiers static on va retrouver les CSS et dans les templates les pages HTML pour le front.

Annexe 9 : Requête SQL pour la partie généralité

```
SELECT
    ec."Annee",
    ec."Departement",
    md."Libelle"  AS "Nom_Departement",
    ec."Type_cancer",
    ec."Classe_age",
    ec."Sexe",
    SUM(ec."Effectif_patients") AS "Total_Effectif_patients",
    ed."Effectif",
     (CAST(SUM(ec."Effectif_patients") AS DECIMAL) / ed."Effectif") * 100
AS "Prevalence_Pourcentage",
    COALESCE(scv_agg."Total_en_kg_agrege", 0) AS "Total_en_kg"
FROM
    "Pollution_Cancer"."Effectif_cancer" ec
INNER JOIN
    "Pollution_Cancer"."Effectif_departement" ed ON
        ec."Departement" = ed."Num_dep" AND
        ec."Annee" = ed."Annee" AND
        ec."Classe_age" = ed."Age" AND
        ec."Sexe" = ed."Sexe"
INNER JOIN
    "Pollution_Cancer"."metadata_departement" md ON
        ec."Departement" = md."Code"
LEFT JOIN
    (SELECT
        scv.annee,
        scv.num_departement,
        SUM(scv."quantite_en_kg") AS "Total_en_kg_agrege"
    FROM
        "Pollution_Cancer"."Substance_cmr_vente" scv
    GROUP BY
        scv.annee,
        scv.num_departement
    ) AS scv_agg ON
        ec."Departement" = scv_agg.num_departement AND
        ec."Annee" = scv_agg.annee
WHERE
    ec."Annee" BETWEEN 2015 AND 2022
    AND ec."Classe_age" LIKE '\_T' ESCAPE '\'
    AND ec."Sexe" LIKE '\_T' ESCAPE '\'
    AND ec."Departement" = %(dept_code)s
GROUP BY
```

```
    ec."Annee",
    ec."Departement",
    md."Libelle",
    ec."Type_cancer",
    ec."Classe_age",
    ec."Sexe",
    ed."Effectif",
    scv_agg."Total_en_kg_agrege"
ORDER BY
    md."Libelle" ASC,
    ec."Annee" ASC;
```



	Annee	Departement	Nom_Departement	Type_cancer	Classe_age	Sexe	Total_Effectif_patients	Effectif	Prevalence_Pourcentage	Total_en_kg
1	2015	1	Ain	Autres cancers	_T	_T	14 190	631 877	2,2456902214	68 567,330062
2	2015	1	Ain	Cancer bronchopulmonaire	_T	_T	1 070	631 877	0,1693367538	68 567,330062
3	2015	1	Ain	Cancer colorectal	_T	_T	2 690	631 877	0,4257157643	68 567,330062
4	2015	1	Ain	Cancer de la prostate	_T	_T	4 800	631 877	0,7596415125	68 567,330062
5	2015	1	Ain	Cancer du sein de la femme	_T	_T	5 620	631 877	0,8894136042	68 567,330062

Annexe 10 : Requête SQL pour la partie Data_Sexe

```
SELECT
    ec."Annee",
    ec."Departement",
    md."Libelle"  AS "Nom_Departement",
    ec."Type_cancer",
    ec."Classe_age",
    ec."Sexe",
    SUM(ec."Effectif_patients") AS "Total_Effectif_patients",
    ed."Effectif",
     (CAST(SUM(ec."Effectif_patients") AS DECIMAL) / ed."Effectif") * 100
AS "Prevalence_Pourcentage",
    COALESCE(scv_agg."Total_en_kg_agrege", 0) AS "Total_en_kg"
FROM
    "Pollution_Cancer"."Effectif_cancer" ec
INNER JOIN
    "Pollution_Cancer"."Effectif_departement" ed ON
        ec."Departement" = ed."Num_dep" AND
        ec."Annee" = ed."Annee" AND
        ec."Classe_age" = ed."Age" AND
        ec."Sexe" = ed."Sexe"
INNER JOIN
    "Pollution_Cancer"."metadata_departement" md ON
        ec."Departement" = md."Code"
LEFT JOIN
    (SELECT
        scv.annee,
        scv.num_departement,
        SUM(scv."quantite_en_kg") AS "Total_en_kg_agrege"
    FROM
        "Pollution_Cancer"."Substance_cmr_vente" scv
    GROUP BY
        scv.annee,
        scv.num_departement
    ) AS scv_agg ON
        ec."Departement" = scv_agg.num_departement AND
        ec."Annee" = scv_agg.annee
WHERE
    ec."Annee" BETWEEN 2015 AND 2022
    AND ec."Classe_age" LIKE '\_T' ESCAPE '\'
    AND ec."Sexe" IN ('M','F')
    AND ec."Departement" = %(dept_code)s
GROUP BY
```

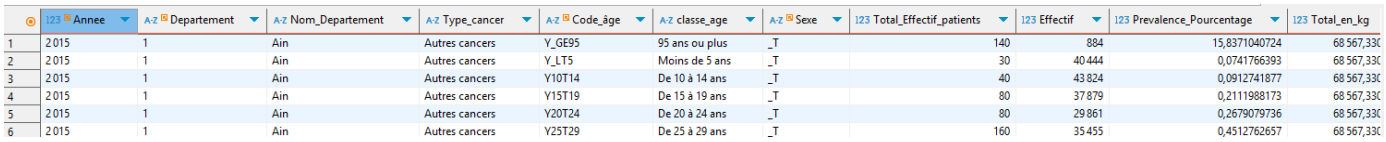
```
    ec."Annee",
    ec."Departement",
    md."Libelle",
    ec."Type_cancer",
    ec."Classe_age",
    ec."Sexe",
    ed."Effectif",
    scv_agg."Total_en_kg_agrege"
ORDER BY
    md."Libelle" ASC,
    ec."Annee" ASC;
```

	Annee	Departement	Nom_Departement	Type_cancer	Classe_age	Sexe	Total_Effectif_patients	Effectif	Prevalence_Pourcentage	Total_en_kg
1	2015	1	Ain	Autres cancers	_T	F	6 760	319 895	2,1131933916	68 567,330062
2	2015	1	Ain	Autres cancers	_T	M	7 430	311 982	2,3815476534	68 567,330062
3	2015	1	Ain	Cancer bronchopulmonaire	_T	F	340	319 895	0,1062848747	68 567,330062
4	2015	1	Ain	Cancer bronchopulmonaire	_T	M	740	311 982	0,2371931714	68 567,330062
5	2015	1	Ain	Cancer colorectal	_T	F	1 240	319 895	0,3876271902	68 567,330062

Annexe 11 : Requête SQL pour la partie Data_Age

```
SELECT
    ec."Annee",
    ec."Departement",
    md."Libelle"  AS "Nom_Departement",
    ec."Type_cancer",
    ec."Classe_age" AS "Code_âge",
    ma."Libelle" AS "classe_age",
    ec."Sexe",
    SUM(ec."Effectif_patients") AS "Total_Effectif_patients",
    ed."Effectif",
     (CAST(SUM(ec."Effectif_patients") AS DECIMAL) / ed."Effectif") * 100
AS "Prevalence_Pourcentage",
    COALESCE(scv_agg."Total_en_kg_agrege", 0) AS "Total_en_kg"
FROM
    "Pollution_Cancer"."Effectif_cancer" ec
INNER JOIN
    "Pollution_Cancer"."Effectif_departement" ed ON
        ec."Departement" = ed."Num_dep" AND
        ec."Annee" = ed."Annee" AND
        ec."Classe_age" = ed."Age" AND
        ec."Sexe" = ed."Sexe"
INNER JOIN
    "Pollution_Cancer"."metadata_departement" md ON
        ec."Departement" = md."Code"
INNER JOIN
    "Pollution_Cancer".metadata_age ma ON
        ec."Classe_age" = ma."Code"
LEFT JOIN
    (SELECT
        scv.annee,
        scv.num_departement,
        SUM(scv."quantite_en_kg") AS "Total_en_kg_agrege"
    FROM
        "Pollution_Cancer"."Substance_cmr_vente" scv
    GROUP BY
        scv.annee,
        scv.num_departement
    ) AS scv_agg ON
        ec."Departement" = scv_agg.num_departement AND
        ec."Annee" = scv_agg.annee
WHERE
    ec."Annee" BETWEEN 2015 AND 2022
```

```
    AND ec."Classe_age" NOT LIKE '\_T' ESCAPE '\'
    AND ec."Sexe" LIKE '\_T' ESCAPE '\'
    AND ec."Departement" = %(dept_code)s
GROUP BY
    ec."Annee",
    ec."Departement",
    md."Libelle",
    ec."Type_cancer",
    ec."Classe_age",
    ma."Code",
    ec."Sexe",
    ed."Effectif",
    scv_agg."Total_en_kg_agrege"
ORDER BY
    md."Libelle" ASC,
    ec."Annee" ASC;
```



	Annee	Departement	Nom_Departement	Type_cancer	Code_âge	classe_age	Sexe	Total_Effectif_patients	Effectif	Prevalence_Pourcentage	Total_en_kg
1	2015	1	Ain	Autres cancers	Y_GE95	95 ans ou plus	_T	140	884	15,8371040724	68 567,33(
2	2015	1	Ain	Autres cancers	Y_LT5	Moins de 5 ans	_T	30	40 444	0,0741766393	68 567,33(
3	2015	1	Ain	Autres cancers	Y10T14	De 10 à 14 ans	_T	40	43 824	0,0912741877	68 567,33(
4	2015	1	Ain	Autres cancers	Y15T19	De 15 à 19 ans	_T	80	37 879	0,2111988173	68 567,33(
5	2015	1	Ain	Autres cancers	Y20T24	De 20 à 24 ans	_T	80	29 861	0,2679079736	68 567,33(
6	2015	1	Ain	Autres cancers	Y25T29	De 25 à 29 ans	_T	160	35 455	0,4512762657	68 567,33(