Réalisation d’une application de type RAG

**      **

Certification : Développeur d’application en Intelligence Artificiel

Marseille - Nice

LOMBARDI Joachim

Table des matières

[I.](#_30j0zll) Réaliser la collecte, le stockage et la mise à disposition des données d’un projet en intelligence artificielle 5

[A.](#_1fob9te) Règles de confidentialité 5

[1)](#_3znysh7) Collecte des données 6

[2)](#_2et92p0) Spécification techniques relatifs à l’extraction des données 7

[3)](#_3rdcrjn) Les vues basées sur des fonctions (Function-Based Views - FBV) 13

[4)](#_26in1rg) Les vues basées sur des classes (Class-Based Views - CBV) 13

[5)](#_lnxbz9) Sécurisation 14

[II.](#_35nkun2) Intégrer des modèles et des services d’intelligence artificielle 15

[A.](#_1ksv4uv) Choix moteurs de recherche 15

[A.](#_44sinio) Elasticsearch 15

[B.](#_2jxsxqh) SolR 16

[C.](#_z337ya) FAISS 17

[B. Utilisation d’elasticsearch 18](#_3j2qqm3)

[a)](#_1y810tw) Indexation des documents 18

[*C. Génération de la réponse* 21](#_4i7ojhp)

[a)](#_2xcytpi) Choix du LLM 21

[b)](#_1ci93xb) Choix du moteur d’inférence 22

[c)](#_3whwml4) Utilisation du LLM 23

[D.](#_2bn6wsx) Evaluation du RAG 24

[a)](#_qsh70q) Choix du mode d’évaluation 24

[b)](#_3as4poj) Paramètres du RAG 25

[E.](#_1pxezwc) Tests 28

[a)](#_49x2ik5) Test de scrapping 28

[b)](#_2p2csry) Test d’import en base 28

[c)](#_147n2zr) Test du CRUD 29

[d)](#_3o7alnk) Test du RAG 29

[F.](#_23ckvvd) Monitoring 29

[a)](#_ihv636) Grafana 30

[b)](#_32hioqz) Prometheus 30

[c)](#_1hmsyys) Uptime Kuma 31

[G.](#_41mghml) Logs 32

[H.](#_2grqrue) Déploiement 32

[a)](#_vx1227) Outils de conteneurisation 32

[b)](#_3fwokq0) Conteneurisation 35

[I.](#_1v1yuxt) Outils de versionnement et CI/CD 37

[a)](#_4f1mdlm) Outils de versionnement 37

[b)](#_2u6wntf) CI/CD 38

[J.](#_19c6y18) Conduite agile de projet. 39

[a)](#_3tbugp1) SCRUM 39

[b)](#_28h4qwu) Kanban 40

[c)](#_nmf14n) XP (Extreme Programming) : Méthode Agile Technique. 40

[d)](#_37m2jsg) Trello 41

[e)](#_1mrcu09) Jira 42

[f)](#_46r0co2) Asana : Outil de Gestion de Projet Collaboratif 42

Afin de réaliser mon application de manière sécurisée, efficace et structurée, j’utilise le Framework Django que l’on utilise dans mon entreprise.

Django est un Framework populaire, datant de 2005, qui permet de créer des applications robustes et sécurisées.

Django comprend :

* Un moteur ORM utilisant des objets python plutôt que des requêtes SQL, ce qui simplifie l’interaction avec la base de données.
* Une interface d’administration pour gérer les données. (Notamment le CRUD).
* Un système de routage d'URL séparé de la vue.
* Un système de templates HTML qui permet de séparer la logique métier (dans les vues et modèles) de l’affichage (dans les templates). Il intègre également des variables, des filtres pour transformer les données et des balises pour la logique.
* La sécurité intégrée avec la protection contre les vulnérabilités courantes comme :
* Les attaques XSS (Cross-Site Scripting) : Django échappe automatiquement les variables rendues dans les templates pour empêcher l'exécution de code malveillant.
* Les injections SQL.
* Les attaques CSRF (Cross-Site Request Forgery) ciblent des actions qui modifient les données : Django utilise un système de jetons CSRF pour valider l'origine des requêtes POST, PUT, PATCH et DELETE.
* Le clickjacking : Django inclut un middleware qui ajoute l'en-tête X-Frame-Options pour empêcher l'intégration des pages web dans des cadres (frames).
* Le stockage sécurisé des mots de passe.
* La modularité qui permet de diviser un projet en modules réutilisables.
* Un système de migration de base de données. Les migrations permettent de synchroniser facilement les changements apportés aux modèles Python avec la structure de la base de données.

Le langage utilisé est le Python car c’est celui utilisé pour appli en IA, de plus c’est celui qu’on utilise dans mon entreprise.

L’outil de versioning Git est utilisé pour stocker le script de l’application en ligne.

# Réaliser la collecte, le stockage et la mise à disposition des données d’un projet en intelligence artificielle

Tout média numérique est susceptible d’être utilisé comme source de données, les documents bien évidemment, mais aussi les sites internet, les vidéos, les bandes sonores, les images…

Afin de pouvoir en extraire des données, il faut d’abord connaître les règles de confidentialité.

## Règles de confidentialité

Ces règles servent à protéger les données personnelles comme :

* Le nom, le prénom, l’adresse, le numéro de téléphone.
* Les informations d'identification (numéro de carte d'identité, numéro de sécurité sociale, etc.).
* Les données numériques : adresses IP, cookies, géolocalisation.
* Les données sensibles : informations de santé, opinions politiques, orientation sexuelle, croyances religieuses, etc.

Selon la CNIL, les 7 grands principes des règles de protection des données personnelles sont les suivants :

**Principe 1** : Licéité, limitation des finalités, minimisation des données

Licéité : Les données personnelles ne doivent être collectées que pour des finalités légitimes, explicites et déterminées.

Limitation des finalités : Les données collectées ne peuvent être utilisées que pour les finalités pour lesquelles elles ont été collectées.

Minimisation des données : La quantité de données collectées doit être réduite au minimum nécessaire pour atteindre les finalités déterminées.

En d'autres termes, les entreprises doivent être transparentes sur la raison pour laquelle elles collectent des données et ne collecter que les informations dont elles ont réellement besoin. Les mentions légales sur votre site internet sont là pour les informer.

**Principe 2** : Exactitude

Les données personnelles doivent être exactes et, si nécessaire, mises à jour. Des mesures doivent être prises pour que les données inexactes soient rectifiées ou effacées.

Cela signifie que les entreprises doivent prendre des mesures pour s'assurer que les données personnelles qu'elles détiennent sont exactes et à jour.

**Principe 3** : Limitation du traitement

Les données personnelles ne doivent être conservées que pendant une durée proportionnelle aux finalités pour lesquelles elles sont traitées.

En d'autres termes, les entreprises doivent avoir une politique claire de conservation des données et supprimer les données lorsqu'elles ne sont plus nécessaires.

**Principe 4** : Intégrité et confidentialité

Les données personnelles doivent être traitées de manière à garantir leur sécurité, y compris la protection contre le traitement non autorisé ou illicite et contre la perte, la destruction ou les dommages accidentels, par des mesures techniques ou organisationnelles appropriées.

Cela signifie que les entreprises doivent mettre en place des mesures de sécurité adéquates pour protéger les données personnelles contre les accès non autorisés, les utilisations abusives et les pertes accidentelles.

**Principe 5** : Responsabilisation

Le responsable du traitement est responsable du respect des principes susmentionnés et doit être en mesure de démontrer sa conformité.

Cela signifie que les entreprises doivent désigner un responsable de la protection des données (DPO) qui sera responsable de la mise en œuvre et du respect du RGPD.

**Principe 6** : Respect des droits des personnes

Les personnes ont le droit d'accéder à leurs données personnelles, de les rectifier, de les effacer, de limiter leur traitement, de s'opposer à leur traitement et de les transférer.

En d'autres termes, les entreprises doivent informer leurs clients de leurs droits en matière de protection des données et leur permettre d'exercer ces droits facilement.

**Principe 7** : Protection des données dès la conception et par défaut :

Le responsable du traitement doit mettre en œuvre des mesures techniques et organisationnelles appropriées pour protéger les données personnelles dès la conception du traitement et tout au long des produits et services.

Cela signifie que les entreprises doivent intégrer le traitement des données dans leur stratégie globale d'entreprise.

Ensuite, il existe également des contraintes lors de l’extraction des données :

### Collecte des données

Comme indiqué dans l’introduction, il existe différents types de données outre les différents médias, un même type de source peut avoir plusieurs supports, par exemple ils peuvent être en format texte, JSON, CSV, base de données relationnelles… Ainsi, les méthodes d’extraction sont différentes et peuvent nécessiter du prétraitement comme pour les PDF afin d’avoir des données propres.

Ces méthodes d’extraction peuvent avoir des limites, comme lors du Scraping (extraction de données depuis une page web) où le nombre de requêtes peut être limité, le téléchargement de fichier là aussi peut être limité ou également l’accès à des bases de données protégées par mot de passe. Il peut également y avoir des problèmes de collecte dû au réseau ou bien à la limitation imposée selon la région du monde où se trouve l’utilisateur ou bien les données.

Certaines données sont très volumineuses et imposent des contraintes au niveau de l’espace de stockage et la mémoire, nécessitant des ressources informatiques importantes et des optimisations de traitement.

### Spécification techniques relatifs à l’extraction des données

L’objectif du projet est de réaliser une application utilisant les données médicales du site : pubmed.ncbi.nlm.nih.gov afin de poser des questions techniques sur ces données et d’obtenir des réponses en langage naturel ainsi que les sources ayant servis à produire ces réponses.

#### Pubmed

PubMed (pubmed.ncbi.nlm.nih.gov) est une ressource utilisée pour la recherche biomédicale et scientifique offrant un accès gratuit à une vaste base de données de références et de résumés d'articles scientifiques. Gérée par la National Library of Medicine (NLM) des États-Unis, cette plateforme regroupe des millions de publications provenant de revues spécialisées, couvrant des domaines tels que la médecine, la biologie, la pharmacologie et bien plus encore.

Cette source de donnée est une des sources de données utilisées par l’entreprise dans laquelle j’effectue mon alternance : Data Observer, car elle contient les publications des professionnels et des groupes de professionnels de santé. Un des principaux objectifs de l’entreprise étant justement d’identifier ces professionnels et ces groupes de professionnels par rapport à leurs publications. Si deux professionnels par exemple on les mêmes noms et prénoms on doit pouvoir identifier qui à produit quoi et pour ça, on utilise leurs affiliations (les établissements dans lesquels ils sont travaillés), les postes qu’ils ont occupés, les villes où ils ont travaillé, les personnes avec qui ils ont travaillés…

J’ai également choisi cette source de donnée car dans l’ensemble elle est bien structurée : les titres, abstracts, auteurs et affiliations… sont très souvent identifié par les mêmes balises html ce qui en facilitent grandement l’extraction. De plus, comme précisé dans le premier paragraphe, il s’agit de résumés ce qui simplifie le traitement des données lorsque l’on va vouloir les interroger.

#### Extraction des données

Pour extraire les données depuis le site PubMed, j’ai identifié trois outils :

* **Scrapy** qui est un Framework complet pour le scraping web.
* **Beautiful Soup** qui est une bibliothèque axée sur la manipulation de l’HTML?
* **L’api PubMed** qui permet de récupérer les différentes données, d’effectuer des recherches basées sur des mots-clés, des termes MeSH (Medical Subject Headings), ou des identifiants uniques (comme les PMID). Les résultats sont donnés sous format JSON ou XML.

L’extraction des données d’intérêts étant simple et l’utilisant déjà en entreprise j’ai choisi d’utiliser Beautiful Soup

J’ai choisi d’extraire les données de deux maladies : multiple sclérosis (une maladie qui revient souvent dans les données de l’entreprise) et herpès zoïster (une autre maladie pour alimenter la base de données mais dont le contenu était plus limité) sur la période 2024. Pour ce faire, j’ai scraper les liens de toutes les pages web traitant de ces deux maladies et j’ai scraper le contenu des articles associés à savoir :

* Le titre de la revue dans laquelle un article est publié
* La date de publication de l’article
* Le titre de l’article
* Le résumé de l’article
* L’id pubmed de l’article
* Le doi (Digital Object Identifier) permettant d’identifier et de localiser du contenu numérique.
* Le disclosure indiquant les éventuels conflits d’intérêts.
* Les termes MeSH (Medical Subject Headings) sont un système de vocabulaire contrôlé et hiérarchique développé par la **National Library of Medicine (NLM)** des États-Unis. Ils sont utilisés pour indexer, classer et rechercher des articles scientifiques dans des bases de données biomédicales, comme PubMed et MEDLINE.
* L’URL de l’article.
* Les auteurs de l’article et leurs affiliations.

Pour chaque extraction, on n’extrait les données que si le résumé est présent. Si une donnée est manquante elle prend la valeur None.

##### Initialisation de BeautifulSoup

* Envoi d'une requête GET via *requests.get(url).*
* Vérification du succès de la requête :
  + Si *response.status\_code == 200*, le contenu HTML est parsé avec BeautifulSoup ('html.parser').
  + Sinon, la fonction retourne None.
* Retourne un objet soup prêt pour l'extraction des données.

1. Extraction des URLs des articles PubMed

* Construction de l'URL de recherche :
  + *url = base\_url + "/?term=" + term + "&filter=years." + filter + "-2025"*
* Initialisation du parsing avec init\_soup(url).
* Récupération du nombre total de pages
  + Sélection de l'élément *label.of-total-pages* pour obtenir le nombre total de pages de résultats.
  + Si l'élément n'existe pas, la valeur par défaut est 1.
* Boucle de pagination
  + De 1 à *page\_max*, récupération des blocs d'articles *div.search-results-chunk*.
  + Extraction des 10 premiers liens par article avec *article.find\_all('a', href=True)[:10*].
  + Ajout des liens trouvés à links.
  + Pause de 1 seconde (*time.sleep(1)*) pour éviter d'être bloqué par PubMed.
  + Rechargement de la page suivante en modifiant l'URL *(url+"&page="+str(i))* et répétition du processus.
* Retourne la liste des URLs d'articles trouvés.

##### Extraction des informations d’un article

* Initialisation des variables
  + *articles\_data* : Liste pour stocker les articles scrappés.
  + *suffix = term + "\_" + filter* : Permet d’ajouter un suffixe personnalisé au fichier de sortie.
* Récupération des liens :
  + Si url est fourni, utilise la liste existante.
  + Sinon, appelle *extract\_pubmed\_url()* pour récupérer les liens à partir du base\_url.
* Parcours des articles :
  + Pour chaque link, initialise *soup = init\_soup(link).*
  + Si soup est None, passe à l'article suivant.
* Extraction des informations :
  + title\_review : Sélection de *button.journal-actions-trigger['title'].*
  + date :
    - Extraction via *span.cit*.
    - Nettoyage et formatage avec *format\_date(date).*
    - title : Sélection *h1.heading-title*.
  + abstract : Extraction et nettoyage des paragraphes *div.abstract-content p*.
  + pmid : Extraction du PMID via *span.identifier.pubmed strong.current-id.*
  + doi :
  + Extraction de l'identifiant via *span.identifier.doi a.id-link*.
  + Formatage en URL *https://doi.org/ + doi*.
  + disclosure : Déclaration des conflits d'intérêt dans *div.conflict-of-interest div.statement p*.
  + mesh\_terms :
  + Extraction via *button.keyword-actions-trigger.*
  + Conversion en texte *(", ".join(mesh\_terms)).*
  + url : Création du lien absolu via *get\_absolute\_url(pmid).*
* Extraction des auteurs et affiliations :
  + Sélection des auteurs : *soup.select('.authors-list-item').*
  + Filtrage des doublons avec *seen\_authors* (set pour éviter de scraper plusieurs fois le même auteur).
* Récupération des affiliations :
  + Sélection *author\_tag.select('.affiliation-link').*
  + Récupération des noms via *.get('title', None).*
  + Ajout des noms et affiliations dans *authors\_affiliations*.

#### Merise

##### Modèle Conceptuel de Données (MCD)

Article Authors Affiliations

title\_review name name

date

title

abstract

pmid

doi

disclosure

mesh\_terms

url

term

#### **Entités et Associations**

* **Affiliations**
  + name : Nom de l'affiliation
* **Authors**
  + name : Nom de l'auteur
* **Article**
  + title\_review : Titre de la revue
  + date : Date de publication
  + title : Titre de l'article
  + abstract : Résumé de l'article
  + pmid : Identifiant PubMed
  + doi : DOI de l'article
  + disclosure : Conflits d'intérêts
  + mesh\_terms : Termes MeSH associés
  + url : URL de l'article
  + term : Terme associé

#### **Relations (Cardinalités)**

* **Relation entre Article et Authors (via Authorship) :**
* Un article peut être écrit par plusieurs auteurs.
* Un auteur peut contribuer à plusieurs articles.
* Relation : N:M (Many-to-Many), gérée par le modèle intermédiaire Authorship.
* **Relation entre Authors et Affiliations (via Authorship) :**
* Un auteur peut être associé à plusieurs affiliations.
* Une affiliation peut être liée à plusieurs auteurs.
* Relation: N:M (Many-to-Many), gérée implicitement via Authorship.
* **Relation entre Article et Affiliations (via Authorship) :**
* Un article peut être lié à plusieurs affiliations (via ses auteurs).
* Une affiliation peut apparaître dans plusieurs articles.
* Relation: N:M (Many-to-Many), gérée indirectement via Authorship.

##### Modèle Logique de Données (MLD)

**Affiliations**

* **id\_affiliation** (clé primaire)
* name (nom de l'affiliation)

**Authors**

* **id\_author** (clé primaire)
* name (nom de l'auteur)

**Article**

* **id\_article** (clé primaire)
* title\_review (titre de la revue)
* date (date de publication)
* title (titre de l'article)
* abstract (résumé de l'article)
* pmid (identifiant PubMed)
* doi (identifiant DOI)
* disclosure (déclaration de conflit d'intérêt)
* mesh\_terms (termes MeSH)
* url (URL)
* term (terme associé à l'article)

**Authorship (Table de jonction entre Authors, Article, et Affiliations)**

* **id\_authorship** (clé primaire)
* id\_article (clé étrangère vers Article)
* id\_author (clé étrangère vers Authors)
* id\_affiliation (clé étrangère vers Affiliations)

##### Modèle Physique des Données (MPD)

#### 

|  |  |  |
| --- | --- | --- |
| Attribut | Type | Contrainte |
| id | INTEGER | PRIMARY KEY |
| name | TEXT | NOT NULL |

Table. Affiliations

|  |  |  |
| --- | --- | --- |
| Attribut | Type | Contrainte |
| id | INTEGER | PRIMARY KEY |
| name | VARCHAR | NOT NULL |

Table. Authors

|  |  |  |
| --- | --- | --- |
| Attribut | Type | Contrainte |
| id | INTEGER | PRIMARY KEY |
| title\_review | VARCHAR | NULL |
| date | DATE | NULL |
| title | VARCHAR | NULL |
| abstract | TEXT | NULL |
| pmid | INTEGER | NULL |
| doi | VARCHAR | NULL |
| disclosure | TEXT | NULL |
| mesh\_terms | TEXT | NULL |
| url | VARCHAR(200) | NULL |
| term | VARCHAR | NULL |

Table. Article

|  |  |  |
| --- | --- | --- |
| Attribut | Type | Contrainte |
| id | INTEGER | PRIMARY KEY |
| author\_id | INTEGER | FOREIGN KEY -> Authors |
| article\_id | INTEGER | FOREIGN KEY -> Article |
| affiliation\_id | INTEGER | FOREIGN KEY -> Affiliations |
| UNIQUE (author\_id, article\_id, affiliation\_id) |  |  |

Table. Authorship

#### Insertion des données en base

Les scripts de scraping et d’insertion en base de données s’écrivent dans le fichier business\_logic.py (logique métier) car il s’agit d’un script propre à ce projet, les fonctions courantes sont stockées dans utils.py.

Afin de réaliser l’extraction et l’insertion en base de données, j’ai créé deux commandes dans le fichier commands.py : commands scrap\_article et commands article\_to\_database.

Une fois scrappés, les articles sont stockés dans un JSON ou un CSV, puis de là insérés dans une base de données.

Le stockage dans un JSON permet de séparer les étapes de scraping et d’insertion en base de données. Ce qui est pratique car s’il y a un problème lors de l’insertion des données en base, il faut reprendre le scraping depuis le début, de plus stocker les données à deux endroits différents permet d’éviter la perte des données si la base de données est effacée, cela permet également de pouvoir modifier la structure de la base de données tout en conservant les données ou encore de faire une insertion dans plusieurs bases de données…

La base de données choisie est Postgres, car c’est celle utilisée en entreprise et elle est native avec Django.

Postgres est open-source, elle existe depuis 1986 donc il y a une vaste communauté, beaucoup de modules et de documentation. Elle supporte les bases relationnelles et les objets.

J’ai défini quatre classes :

* *Affiliations* pour stocker les noms des affiliations.
* *Authors* pour stocker les noms des auteurs.
* *Article* pour stocker toutes les informations basiques liées aux articles et la relation Many-to-Many permettant de relier un article avec ces auteurs et pour chaque auteur ses affiliations.
* *Authorship est* la table intermédiaire avec trois clés étrangères pour les articles, les auteurs et les affiliations.

La connexion avec la base de données se fait dans settings.py.

Pour créer la base de données on va créer dans models.py des classes dont chacune correspond à une table de la base de données. On définit ici, le type de variable, si elle peut être nulle, le nom des colonnes... Ensuite on valide les migrations avec la commande makemigrations et on crée la base de données avec migrate. La base de données est créée.

Pour ajouter les données en base si les champs sont ne sont pas des clés étrangères, il se remplissent directement en ajoutant les champs dans l’objet model. Pour chaque article, on a une liste d’auteurs et pour chaque auteur une liste d’affiliations. On itère sur chaque article, puis sur chacun de ses auteurs et chacune de leur affiliation, on les ajoute en base et on récupère les objets de chaque ligne correspondante (les objets auteurs et affiliations) que l’on insère à chaque itération dans l’objet Authorship.

#### CRUD

Maintenant que toutes les données ont été ajoutées à la base, il faut pouvoir mettre à jour la base.

Le CRUD (Create, Read, Update, Delete) se fait dans le fichier views.py car il nécessite un formulaire pour modifier la base de données et un affichage pour visualiser les données.

On peut créer un article, lire l’ensemble des articles, mettre à jour un article ou le supprimer.

Chacune de ces actions est réalisable via une requête HTTP sur l’API FastAPI.

Voici un exemple d’une requête SQL pour récupérer toutes les données d’un article avec ses auteurs et leurs affiliations:

|  |
| --- |
| SELECT a.\*, au.\*, af.\* FROM article art JOIN authorship a ON a.article\_id = art.id JOIN author au ON a.author\_id = au.id JOIN affiliation af ON a.affiliation\_id = af.id WHERE art.id = {pk}; |

On sélectionne toutes les entrées des trois tables puis on indique la première source ici article que l’on va lier à la table intermédiaire via l’id de l’article et article\_id puis lier cette table aux deux tables restantes auteur par l’id de l’auteur et auteur\_id et affiliation par l’id de l’affiliation et affiliation\_id, enfin on précise de quel article il s’agit avec son id et pk (primary key) qui a été récupéré dans l’url via une requête http sur l’article).

Il existe deux types de script pour les vues dans Django :

### Les vues basées sur des fonctions (Function-Based Views - FBV)

Les vues basées sur des fonctions sont des fonctions Python ordinaires qui prennent une requête (Http Request) en paramètre.

Avantages :

* Faciles à lire et à comprendre.
* Idéal pour des besoins simples sans comportement complexe.
* Pas de surcharge avec des concepts avancés.

Inconvénients :

* Deviennent difficiles à maintenir lorsqu'elles deviennent complexes.
* Manquent de modularité et de réutilisabilité pour des comportements similaires.

### Les vues basées sur des classes (Class-Based Views - CBV)

Les vues basées sur des classes sont des classes Python qui héritent des classes utilitaires fournies par Django pour gérer différentes tâches courantes (comme les requêtes HTTP GET ou POST).

Avantages :

* Permet une meilleure organisation pour les projets complexes.
* Plus facile d'ajouter ou de modifier des fonctionnalités spécifiques grâce à l’héritage.
* Supportent des mixins pour réutiliser le code (exemple : LoginRequiredMixin).

Inconvénients :

* Plus complexes à comprendre et à utiliser.
* Nécessitent une bonne connaissance des concepts orientés objet.

Ainsi chaque fonction présentes dans views.py est associée à une URL spécifique présente dans le fichier urls.py dans le dossier de l’application où sont stockées toutes les urls des différentes pages du projet. Pour trouver l’url général de l’api, il faut aller dans le fichier urls.py où se trouve le fichier settings.py. Les urls des différentes pages se concatènent avec l’url générale.

A chaque fonction présente dans views.py est associé un fichier HTML où est défini le design de la page. Les formulaires du CRUD après validation renvoient tous vers la page d'affichage des articles de la base de données.

Les formulaires sont créés dans forms.py en récupérant les modèles et en utilisant leurs attributs, on définit aussi dans forms des méthodes de sauvegarde. Ils sont ensuite importés dans les HTML correspondant.

### Sécurisation

#### HTML

Les formulaires HTML sont protégés contre les attaques CSRF via un jeton placé dans chaque formulaire.

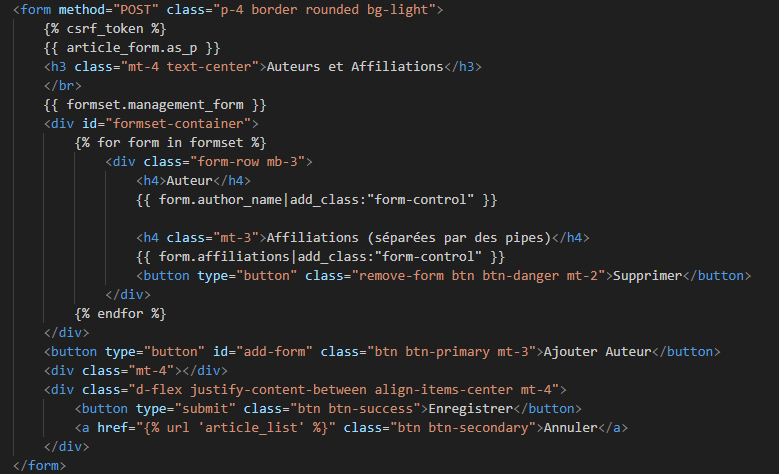


Figure 1. Extrait du HTML create\_update.

Ci-dessus, le jeton CRSF, placé juste en-dessous de la balise d’ouverture form.

#### Ecran d’inscription/connexion

L’accès au site nécessite un login et mot de passe créés lors de l’inscription.

Pour créer un utilisateur, on importe la classe UserCreationForm de Django. La classe UserCreationForm valide automatiquement les champs username, password1, et password2, garantissant qu'ils respectent les exigences de Django, comme l'unicité du nom d'utilisateur et la correspondance des mots de passe. Django utilise l'algorithme de hachage PBKDF2 (Password-Based Key Derivation Function 2) pour hasher les mots de passe. Cet algorithme applique un certain nombre de tours de hachage (par défaut 100 000) pour rendre le hachage plus lent et plus difficile à attaquer par des attaques par force brute. En plus du hachage, un "sel" (un nombre aléatoire unique) est généré pour chaque mot de passe. Le sel est ajouté au mot de passe avant le hachage pour empêcher les attaques cryptographiques comme les attaques par tables de hachage pré-calculées ou par dictionnaire. Le mot de passe haché et le sel sont stockés dans la base de données. Le mot de passe en clair n'est jamais enregistré.

##### Attaque par table de Hashage pré-calculées (RainBow tables)

Les tables de hachage pré-calculées, ou rainbow tables, sont des tables qui contiennent des hachages pour une vaste collection de mots de passe courants des millions voir des milliards. Ces tables permettent à un attaquant de rechercher rapidement un hachage dans la table pour trouver le mot de passe original sans avoir à recalculer le hachage pour chaque mot de passe.

Rôle du salage : L'ajout d'un "sel" unique à chaque mot de passe empêche les attaques par rainbow table. Même si deux utilisateurs ont le même mot de passe, leurs hachages seront différents en raison du sel. Cela rend inutile l'utilisation de rainbow tables, car l'attaquant ne pourra pas prédéfinir tous les hachages possibles.

1. Attaque par dictionnaire

L'attaquant possède un fichier contenant des millions de mots de passe communs (ex : "123456", "password", "qwerty", etc.). Il hache chaque mot de passe avec le même algorithme que celui utilisé par le système cible. Il compare les hachages obtenus avec le hachage qu'il veut casser. Si un hachage correspond, il a trouvé le mot de passe. Ici le salage ne sert à rien car il utilise le même algorithme en revanche, il faut tester les mots de passe un par un ce qui est très coûteux.

Aujourd’hui, la plupart des systèmes utilisent un sel ce qui rend l’attaque par Rainbow tables inefficace. Ces deux techniques montrent l’intérêt de ne pas choisir de mot de passe simple.

Pour se connecter à mon application, l’utilisateur s’inscrit pour se connecter au site en renseignant les champs suivants :

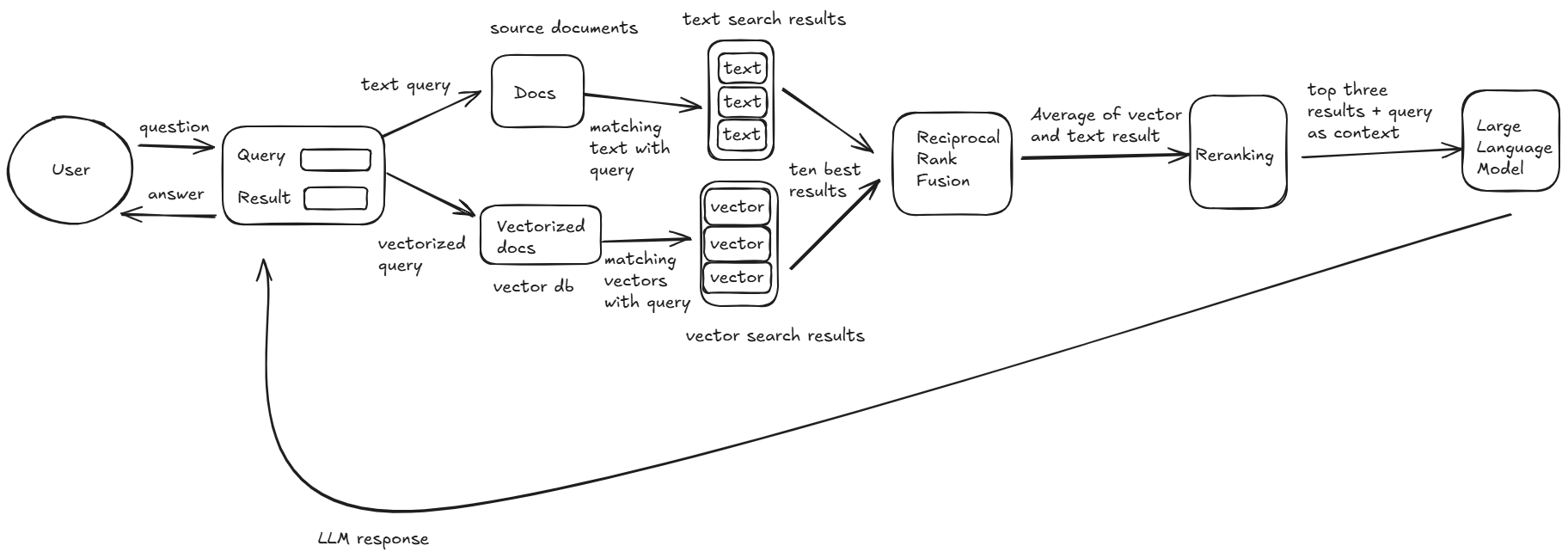
* Nom d’utilisateur
* Email
* Password
* Password (répéter)

Pour se connecter, il renseigne son nom d’utilisateur et son mot de passe. Il a accès à toutes les pages publiques. Les pages privées nécessitent un accès administrateur créé via une commande Django.

Quand un utilisateur se connecte, Django re-hache le mot de passe fourni et compare le résultat avec celui stocké en base de données.

# Intégrer des modèles et des services d’intelligence artificielle

Voici le schéma de mon RAG :



L’objectif est de réaliser un RAG (Retrieval Augmented Generation), à savoir des recherches sur des documents privés (retrieval) et une réponse synthétisée à partir des meilleurs résultats (génération augmentée à l’aide d’un prompt).

Pour ce faire, il faut un moteur de recherche pour parcourir la base de données et un LLM (Large Language Model) pour synthétiser les réponses. Il faut également choisir un moteur d’inférence pour utiliser le LLM en local.

Les trois principaux moteurs de recherche sont Elasticsearch, SolR et Faiss (Facebook AI Similarity Search).

## Choix moteurs de recherche

### Elasticsearch

Elasticsearch est un moteur de recherche distribué et open-source basé sur Apache Lucene. Il permet d'indexer et de rechercher des documents à grande échelle avec des fonctionnalités avancées de recherche en texte intégral.

**Avantages :**

* **Flexibilité et Scalabilité** : Elasticsearch est conçu pour gérer de grandes quantités de données et peut évoluer horizontalement (articles, des images, des vidéos…), ce qui en fait un choix idéal pour des systèmes à grande échelle.
* **Indexation et Recherche rapide** : Grâce à son architecture basée sur **Apache Lucene** (bibliothèque open-source de recherche en texte intégral écrite en Java. Elle fournit des fonctionnalités puissantes et flexibles pour l'indexation et la recherche dans de grands ensembles de données non structurées.), Elasticsearch est extrêmement rapide pour la recherche en texte intégral et peut gérer des requêtes complexes efficacement.
* **Facilité d'Intégration** : Il dispose de nombreuses bibliothèques et plugins qui facilitent son intégration avec des frameworks comme Django, Flask, et d'autres outils populaires dans l'écosystème Python.
* **Recherches Complexes** : Supporte des requêtes complexes, comme des agrégations et des filtres, ce qui permet de personnaliser la recherche pour des cas d'utilisation avancés dans un RAG.
* **Support des Recherches Full-text et Vectorielles** : Elasticsearch a intégré le support des recherches par similarité vectorielle (recherche dense), ce qui le rend compatible avec les systèmes RAG qui utilisent des embeddings comme BERT ou T5.

**Inconvénients :**

* **Complexité de Configuration et Maintenance** : Bien que son installation soit relativement simple, sa configuration et sa maintenance peuvent devenir complexes, notamment lors de l’indexation de très grandes bases de données ou d’optimisation des performances.
* **Gestion des Embeddings** : Bien que le support de la recherche vectorielle ait été amélioré, il reste plus complexe à configurer que dans d'autres moteurs spécialisés comme FAISS.
* **Utilisation des Ressources** : Elasticsearch peut être assez gourmand en ressources (mémoire et processeur) lorsqu’il est utilisé à grande échelle avec des données complexes.

### SolR

Solr est un autre moteur de recherche open-source basé sur Apache Lucene. Il offre des fonctionnalités puissantes pour la recherche en texte intégral et est particulièrement adapté aux environnements d'entreprise.

**Avantages :**

* **Puissant et Flexible** : Solr est un moteur de recherche robuste et flexible qui offre de nombreuses fonctionnalités avancées, comme le faceting (Subdivise les résultats de recherche en facettes, qui sont des catégories ou des critères spécifiques permettant de filtrer ou d’affiner les résultats. Ex : couleur, prix, la marque, catégorie...), la recherche géospatiale, la gestion des données structurées et non structurées, etc.
* **Évolutivité** : Comme Elasticsearch, Solr est conçu pour gérer de grandes quantités de données et peut être déployé sur plusieurs nœuds pour offrir une scalabilité horizontale.
* **Recherches Complexes** : Solr offre un très bon support pour les recherches complexes et permet des personnalisations détaillées des requêtes.
* **Moteur Mature** : Solr est un projet plus ancien que Elasticsearch, donc il est très mature, stable et bien documenté. Il bénéficie également d'une communauté active.

**Inconvénients :**

* **Configuration Complexe** : Solr nécessite une configuration plus approfondie et peut être plus difficile à paramétrer que Elasticsearch. Par exemple, il peut y avoir des défis pour intégrer des embeddings de modèles pré-entraînés pour la recherche par similarité sémantique.
* **Moins Optimisé pour les Embeddings** : Solr est traditionnellement un moteur de recherche basé sur des termes, et bien que des solutions aient été mises en place pour l'intégration des vecteurs, elles ne sont pas aussi simples et intuitives que dans FAISS ou même Elasticsearch.
* **Moins d'Intégration avec Python** : L'intégration avec l'écosystème Python n'est pas aussi fluide qu'avec Elasticsearch, bien qu'il existe des bibliothèques comme pysolr.

### FAISS

FAISS est une bibliothèque développée par Facebook pour la recherche de similarité haute performance. Elle est optimisée pour les recherches dans des espaces vectoriels à grande échelle.

**Avantages :**

* **Spécialisation pour la Recherche de Similarité de Vecteurs** : FAISS est optimisé pour effectuer des recherches dans de grands espaces de vecteurs (embeddings), ce qui est essentiel dans les systèmes RAG où la recherche de passages ou de documents pertinents repose sur des similarités vectorielles.
* **Haute Performance** : FAISS est très rapide et conçu pour gérer de grands volumes de données, avec des index hautement optimisés pour la recherche en temps réel.
* **Prise en charge de la recherche dense** : FAISS est particulièrement bien adapté à la recherche dense basée sur des embeddings, et il propose de nombreuses méthodes pour améliorer la précision et la rapidité des recherches.
* **Support des GPU** : FAISS permet l'utilisation de GPUs pour accélérer la recherche et l'indexation, ce qui est particulièrement utile pour les systèmes RAG traitant de grandes quantités de données.

**Inconvénients :**

* **Moins de Fonctionnalités de Recherche Traditionnelle** : FAISS se concentre exclusivement sur la recherche de similarité vectorielle, donc il ne prend pas en charge la recherche classique (basée sur des termes) comme Elasticsearch et Solr. Pour des systèmes nécessitant des recherches complexes avec des filtres ou des agrégations, il peut être limité.
* **Gestion Complexe des Embeddings** : FAISS nécessite que les utilisateurs gèrent eux-mêmes l'indexation et l'organisation des embeddings. Bien que très performant, il peut exiger un certain niveau d'expertise pour l'implémentation et l'optimisation.
* **Pas de Fonctionnalités de Recherche Textuelle Complètes** : Par rapport à Elasticsearch et Solr, FAISS n'est pas conçu pour effectuer des recherches en texte intégral, ce qui peut limiter sa polyvalence dans des systèmes RAG où les documents sont souvent de grande taille et ont des métadonnées complexes.

En résumé, Elasticsearch gère à la fois les recherches vectorielles (complexe avec SolR) et les recherche textuel (pas conçu pour ça) de plus, il s'intègre bien à python et Django, ses inconvénients sont la maintenance et les ressources ce qui pour le projet qui est limité ne pose donc pas de problème. C’est également ce moteur de recherche que l’on utilise en entreprise.

## B. Utilisation d’elasticsearch

Pour faire des recherches avec elasticsearch, j’utilise un mode de recherche hybride (vectorielle + textuelle) car cela s’avère être la solution la plus efficace.

### Indexation des documents

Lors de la recherche, on classe les documents par index pour les retrouver plus facilement et ne pas chercher dans toute une base de données s’il y a un nombre de documents tel que les temps de recherche pourrait devenir trop important. Dans notre cas, le nombre de documents était faible, un index suffit amplement.

On définit d’une part notre index et d’autre part ce qu’est un document à savoir les informations que le moteur de recherche va pouvoir utiliser pour faire la recherche et pour récupérer les informations sur le document.

Ici, on va effectuer des recherches sur le titre, le résumé et la vectorisation de la concaténation du titre et du résumé. On définit nos champs titre et résumé comme étant ceux du modèle Article et le vecteur comme étant un vecteur dense, c’est-à-dire un vecteur contenant les valeurs significatives du résumé et du titre vectorisés afin de tenir compte du sens du texte.

Pour réaliser la vectorisation, on utilise un modèle capable de donner les résultats les plus significatifs. Le modèle le plus adapté est BERT.

#### BERT (Bidirectional Encoder Representations from Transformers)

BERT, développé par Google AI en 2018, est un modèle de traitement du langage naturel (NLP) basé sur l'architecture Transformer (les Transformers capturent les relations entre tous les mots d'une séquence, indépendamment de leur position.). Contrairement aux modèles unidirectionnels (comme GPT, avant GPT-3), BERT analyse un mot en tenant compte à la fois des mots qui le précèdent et qui le suivent.

**Avantages :**

* **Compréhension contextuelle :** Entraînement bidirectionnel pour une meilleure précision.
* **Performance sur les benchmarks :** BERT a battu des records sur plusieurs benchmarks NLP comme GLUE et SQuAD. (Source : https://arxiv.org/pdf/1810.04805)
* **Polyvalence :** Utilisé dans de nombreuses tâches grâce à son pré-entraînement généraliste.

**Limites :**

* **Taille et complexité :** Les modèles pré-entraînés de BERT sont volumineux, ce qui entraîne des coûts élevés en mémoire et en calcul.
* **Coût énergétique :** Le pré-entraînement et l’utilisation de BERT nécessitent une puissance de calcul significative.
* **Pas adapté aux séquences longues :** Limité à une longueur de séquence de 512 tokens, ce qui peut poser problème pour les documents très longs.

Depuis son lancement BERT a inspiré de nombreuses variantes :

* **RoBERTa :** Optimisation des hyperparamètres pour des performances améliorées.
* **DistilBERT :** Une version plus légère et rapide.
* **BioBERT :** Adapté aux données biomédicales.
* **ALBERT :** Une version compacte de BERT pour réduire les coûts de calcul.

J’utilise une variante entraînée sur les résumés médicaux de PubMed : « microsoft/BiomedNLP-BiomedBERT-base-uncased-abstract »

**Avantages :**

* **Adapté au vocabulaire biomédical :** Le modèle comprend des termes médicaux, pharmacologiques et biologiques mieux qu'un modèle BERT général. Exemple : Il comprend des termes complexes comme *"angiogenèse"*, *"cytokines"*, ou *"miARN"*.
* **Précision dans les tâches biomédicales :** Plus performant que des modèles généralistes comme bert-base-uncased sur des corpus biomédicaux.
* **Flexible :** Peut être ajusté (fine-tuned) pour diverses tâches spécifiques à la recherche biomédicale.

**Limites :**

* **Domaine limité :** Bien qu'excellent pour les textes biomédicaux, il est moins performant sur des textes généraux ou non scientifiques.
* **Données spécifiques nécessaires :** Pour des tâches très spécialisées, un **fine-tuning** supplémentaire sur un sous-ensemble de données biomédicales pertinentes peut être nécessaire.
* **Consommation de ressources :** Comme tout modèle basé sur BERT, il peut nécessiter des ressources significatives pour l'entraînement ou l'inférence.

Ce qui en fait un modèle parfaitement adapté à mon cas car il comprend très bien les terminologies, relations et concepts spécifiques au domaine de la santé et peut donc aider à répondre à des questions techniques dans le domaine de la santé.

#### Création des documents à indexer

La définition du modèle de document à indexer se fait dans document.py.

On crée une instance d’index Elasticsearch basé sur le nom d’index et on réalise le mapping. Ici le projet ayant peu de données, on n’a pas besoin d’exécuter des tâches en parallèles (Shards) et l’accessibilité (replicas) qui correspond à une instance d’elasticsearch, comme il y en a qu’une on peut mettre les répliques à 0. La méthode utilisée pour faire la recherche est le KNN.

Le KNN (k-Nearest Neighbors) est un algorithme qui trouve les **k** éléments les plus proches d'un point donné dans un espace vectoriel, basé sur une métrique de distance comme :

* La distance euclidienne.
* La distance cosinus.
* D'autres distances personnalisées.

On définit ensuite le document :

* + Title
  + Abstract
  + Title\_abstract\_vector

Le document est ensuite rempli dans commands.py avec la table Article qui a une méthode pour réaliser les embeddings et l’indexation est réalisée. Le moteur de recherche peut maintenant être utilisé.

#### Recherche des résultats avec une question en langage naturel.

Il s’agit de la première partie du RAG. La question est posée dans un formulaire, l’appel à la fonction se trouve donc dans views.py et la fonction dans business\_logic.py.

Dans un premier temps, on vectorise la question avec le même modèle que pour les articles.

On définit le type de recherche que l’on souhaite, full-text, sémantique, hybride. Le mode hybride étant payant, j’en ait fait un à la main avec une première recherche full text, puis sémantique et une fonction pour assembler les deux.

D’abord, Elasticsearch va récupérer les 10 résultats les plus pertinents en faisant une recherche KNN parmi 5000 fichiers. Ensuite, il va faire une recherche « multi\_match », c’est-à-dire qu’il va rechercher la correspondance dans les champs titre et abstract. On utilise le type « best fields » pour qu’il garde le score plus élevé parmi les champs titre et abstract.

Ceux-ci sont affectés d’un poids 2 pour le titre et 5 pour le résumé ce qui indique le poids dans le calcul du score en cas de match. On récupère les 10 résultats les plus pertinents. On injecte ces résultats dans une fonction qui va itérer sur chacun de ces résultats pour leur affecter un score en fonction de leur rangs, plus le rang est bas plus le score sera élevé. Ensuite, on additionne les scores de même rang. On les classe dans l’ordre décroissant. Puis on utilise un Cross-Encoder, pour réordonner les résultats. Le Cross-Encoder va capturer toutes les interactions entre les deux textes (ici, la question et le document) pour en évaluer la pertinence.

J’utilise le modèle : ‘cross-encoder/ms-marco-MiniLM-L-6-v2', développé pour traiter des paires (requête, documents) afin de classer les documents par rapport à une requête. On récupère le document et le score correspondant.

Lorsque l’on a nos résultats, on utilise les ids pour récupérer les articles correspondant dans la base de données et les enregistrer dans un dictionnaire.

#### C. Génération de la réponse

### Choix du LLM

#### GPT (Generative Pre-trained Transformer)

GPT est une série de modèles créée par Open AI. Ces modèles sont des réseaux neuronaux avancés conçus pour comprendre et générer du texte naturel.

Ils sont très performants pour la compréhension du langage et la génération de texte mais pas open-source.

#### Mistral

Mistral est une entreprise de pointe spécialisée dans les modèles de langage de grande taille (LLM). Fondée en 2023 par des experts en intelligence artificielle, elle vise à offrir des solutions performantes, ouvertes et transparentes pour les applications basées sur le traitement du langage naturel (NLP).

Mistral propose des modèles open-source équivalents à GPT.

#### Llama

LLaMA est une famille de modèles de langage développée par Meta AI (anciennement Facebook AI) datant de 2023.

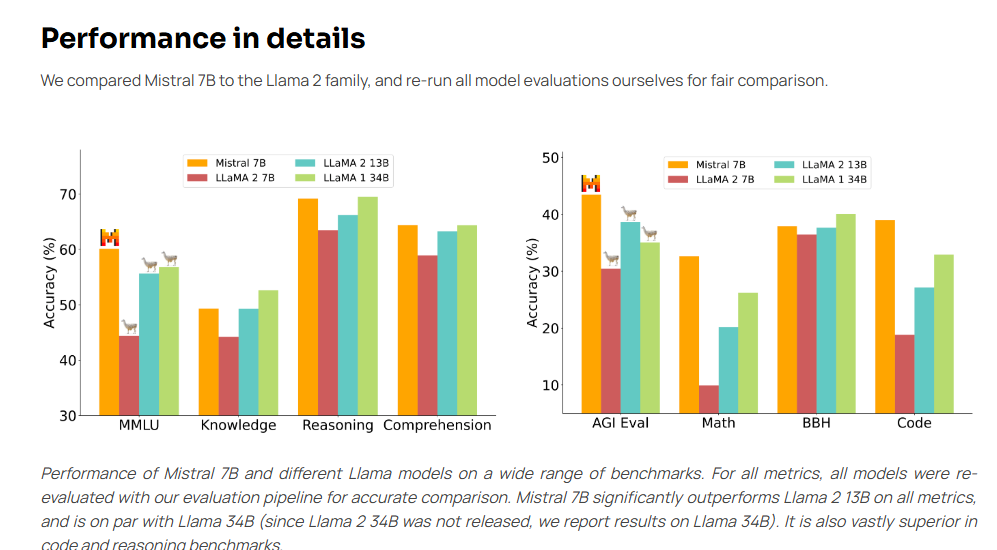


Figure 2. Performance de mistral. (source : <https://mistral.ai/en/news/announcing-mistral-7b>)

Afin de générer une réponse, j’utilise un LLM ici Mistral modèle 7B.

Le modèle Mistral 7b est basé sur la famille des Transformers comme GPT. Il a 7 milliards de paramètres d’où son nom. Ci-dessus une évaluation de ses performances issues du site mistral.

Il est plus performant que d’autres modèles open-source tels que LLama, ce n’est pas le plus performant mais il offre un bon compromis en rapport de performances/consommation de mémoire.

Afin de le faire tourner en local, j’utilise le moteur d’inférence Ollama. Un moteur d'inférence est un composant logiciel ou système spécialisé conçu pour exécuter des modèles d'intelligence artificielle (IA) ou d'apprentissage automatique (ML) afin de produire des résultats ou des prédictions à partir de données d'entrée.

### Choix du moteur d’inférence

Il existe deux principaux moteurs d’inférence Ollama et VLLM.

#### Ollama

Ollama est conçu pour être un framework d'optimisation de modèles de langage qui permet de déployer des modèles LLM sur des serveurs locaux. Il s’adresse aux entreprises cherchant à intégrer des LLM sur leurs serveurs locaux tout en garantissant la sécurité et la confidentialité des données.

L'un des principaux avantages d'Ollama est qu'il permet un contrôle total des données, car les utilisateurs peuvent exécuter des modèles sur leurs propres serveurs locaux. Cela garantit que les données sensibles ne quittent jamais l'infrastructure interne.

#### VLLM

VLLM (ou Very Large Language Models) est une bibliothèque open-source optimisée pour l'exécution et l'inférence de modèles de langage de grande taille (LLMs) tels que GPT-3 et GPT-4.

VLLM est une bibliothèque puissante et flexible pour l'exécution de modèles de langage de grande taille dans des environnements cloud ou de centres de données.

La gestion de la mémoire est optimisée pour traiter de très grands modèles avec des ressources limitées, grâce à des techniques comme la quantification des poids des modèles et l'utilisation de memory-mapped files.

Etant donné que le projet se fait en local et que la taille des données est modérée, Ollama est donc plus adapté au projet.

### Utilisation du LLM

Maintenant que nous avons déterminé le LLM et son moteur d’inférence, nous allons l’augmenter à l’aide d’un prompt. Ce prompt contient les éléments suivants :

* Un persona : représente une configuration ou une personnalité simulée que le modèle incarne pour répondre aux besoins d'un utilisateur spécifique ou d'une application particulière. Cela peut inclure son style de communication, ses connaissances, ou sa perspective.
* Des instructions : Les différentes tâche que le modèle devra réaliser pour arriver aux résultats, et la gestion des cas particuliers (ex : si la réponse ne se trouve pas dans les documents.
* La query : La question posée.
* Le context : Les articles retournés par elasticsearch servant à répondre à la question.
* Le format de la réponse : un dictionnaire avec une clé « réponse » et comme valeur la réponse du LLM sous forme de chaîne de caractère.

Voici le prompt :

“*You are an expert in analysing medical abstract and your are talking to a pannel of medical experts. Your task is to use only provided context to answer at best the query.*

*If you don't know or if the answer is not in the provided context just say: "I can't answer with the provided context".*

*## Instruction:\n*

*1. Read carefully the query and look in all extract for the answer.*

*## Query:\n*

*'"""+query+"""'*

*## Context:\n*

*'"""+context+"""'*

*## Expected Answer:\n*

*{*

*"response": str*

*}*

*You must provid a valid JSON with the key "response".”*

Ensuite, on paramètre le LLM avec les éléments suivants :

* Le modèle : ici mistral pour Mistral 7b
* Le message : Un dictionnaire où l’on passe le rôle à savoir la personne qui envoie le message (assistant si c’est le LLM, system pour définir les comportements généraux du LLM, user si c’est un message envoyé par un utilisateur), ici comme on lui passe un Template avec le message de l’utilisateur, on choisit « user ».
* Stream : si on veut que le message apparaisse au fur et à mesure.
* Le format : pour définir le format de la réponse ici, un JSON.
* Les options : on choisit une valeur fixe de seed pour que le résultat soit toujours les même d’un utilisateur à l’autre et une température qui mesure le niveau de créativité entre 0 et 1. (0 pas de créativité et 1 très imaginatif). Comme nous interrogeons des document la température est à 0.

## Evaluation du RAG

### Choix du mode d’évaluation

Pour évaluer la qualité d’un RAG, il existe différentes solutions en voici quelques-unes :

* La méthode humaine : Un ou des experts évaluent la qualité de la réponse générée et du des résultats de la recherche.
* LLM as a judge : à l’aide d’un prompt, un LLM évalue la qualité de la réponse et du retrieval (résultats de la recherche). Il évalue la cohérence entre la réponse et la question (pertinence), la réponse par rapport aux documents fournis (fact-checking), les documents fournis par rapport à la question. On peut aussi lui demander de comparer les réponses fournies aux réponses attendues…
* L’évaluation automatique nécessite de connaître la réponse attendue. Elle utilise des métriques quantitatives.

Pour la génération :

* + **ROUGE** : Mesure le chevauchement des n-grams entre la réponse générée et la référence.
  + **BLEU** : Évalue la précision des n-grams entre la sortie et la référence (utilisé surtout pour la traduction).
  + **METEOR** : Prend en compte la synonymie et le stemming pour mesurer la similarité.
  + **BERTScore** : Compare les embeddings des réponses générées et des références pour une évaluation sémantique.

Pour le retrieval :

* + **Recall@k** : Taux de récupération des documents pertinents parmi les kkk premiers résultats.
  + **MRR (Mean Reciprocal Rank)** : Mesure la position du premier document pertinent dans les résultats.
  + **NDCG (Normalized Discounted Cumulative Gain)** : Pondère la pertinence des documents récupérés en fonction de leur position.

Ces métriques peuvent être déterminées à l’aide de bibliothèques (Pyserini pour le retrieval et Hugging Face Transformers pour la génération) ou de FrameWork (Elasticsearch, FAISS…) ou des plateformes (DeepAI, Google EvalAI, OpenAI API, LangChain).

Pour faire l’évaluation j’ai choisi LLM as a judge car je ne connais pas les bons résultats.

L’évaluation du RAG se fait à l’aide d’un LLM différent de celui qui fait la génération ici GPT-4-o car il est plus performant, en effet il a été conçu pour travailler sur des tâches complexes là où mistral 7b est adapté à des environnements limités. GPT-4-o est une variante optimisée de GPT-4, développée par OpenAI. Elle a été conçue pour offrir une performance similaire ou supérieure à GPT-4, tout en étant plus rapide et moins coûteuse en termes de consommation de ressources.

### Paramètres du RAG

Afin de réaliser le RAG le plus performant, je fais varier certains paramètres influençant le retrieval et la génération.

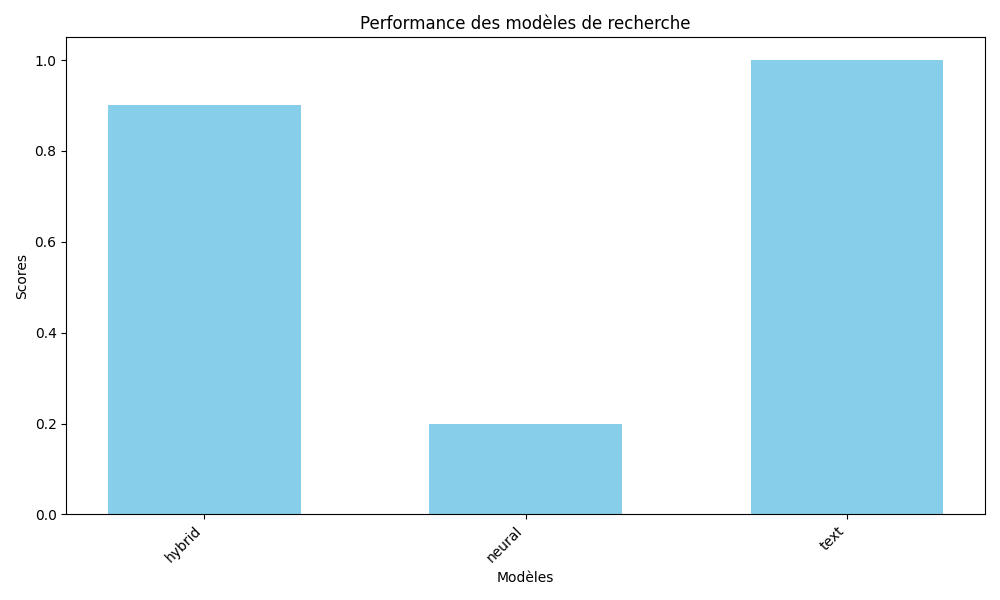
Pour le retrieval :

* Les poids du résumé et du titre
* Le type de recherche : full-text, hybrid, vectorielle.
* Le nombre d’articles que l’on passe pour le re-ranking.
* Le prompt.

Ci-après, le graphique des scores donnés par GPT4-o selon le type de recherche avec le prompt suivant : *« You are an expert in medical abstracts. You will receive two medical abstracts and a query. Your task is to determine which of these abstracts contains the most pertinent information to answer the query.*

*You must return the number of the abstract containing the most relevant information. If the two abstracts contain the same information, return number 1.*

*Please provide a clear reason for your decision.* »



On voit que le type text donne les meilleurs résultats et le neural les pires. En ce qui me concerne, je suis critique sur ces résultats car la recherche neuronale tient compte du contexte pour classer les résultats ce qui est bien plus efficace qu’une simple recherche par mot clé car les questions nécessitent des réponses détaillées et non un élément de réponse. Pareil pour la recherche hybride elle utilise la combinaison des recherches neuronal et textuel en moyennant les scores donc le score devrait être le plus élevé.

Pour la génération :

* Le prompt.
* Le modèle.

Ci-après, le graphique des scores donnés par GPT4-o selon le modèle Mistral avec le prompt suivant: *« Your task is to score the relevance between a generated answer and the query based on the retrieval in the range between 1 and 5, and please also provide the scoring reason.*

*Your primary focus should be on determining whether the generated answer contains sufficient information to address the given query according to the retrieval.*

*If the generated answer fails to provide enough relevant information or contains excessive extraneous information, then you should reduce the score accordingly.*

*If the generated answer contradicts the retrieval, it will receive a low score of 1-2.*

*For example, for query "Is the sky blue?", the retrieval is "the sky is blue." and the generated answer is "No, the sky is not blue.".*

*In this example, the generated answer contradicts the retrieval by stating that the sky is not blue, when in fact it is blue.*

*This inconsistency would result in a low score of 1-2, and the reason for the low score would reflect the contradiction between the generated answer and the retrieval.*

*Please provide a clear reason for the low score, explaining how the generated answer contradicts the retrieval.*

*Labeling standards are as following:*

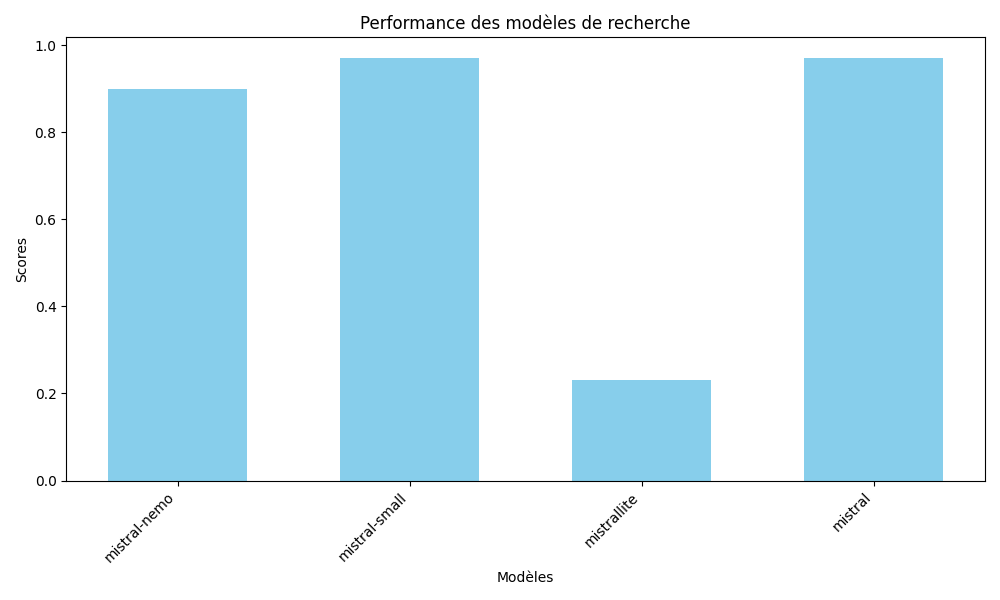
*5 - ideal, should include all information to answer the query comparing to the retrieval， and the generated answer is consistent with the retrieval*

*4 - mostly relevant, although it might be a little too narrow or too broad comparing to the retrieval, and the generated answer is consistent with the retrieval*

*3 - somewhat relevant, might be partly helpful but might be hard to read or contain other irrelevant content comparing to the retrieval, and the generated answer is consistent with the retrieval*

*2 - barely relevant, perhaps shown as a last resort comparing to the retrieval, and the generated answer contradicts with the retrieval*

*1 - completely irrelevant, should never be used for answering this query comparing to the retrieval, and the generated answer contradicts with the retrieval*»



Ici, le meilleur modèle est mistral (7 milliards de paramères) et mistral-small (22 milliards), nemo (12 milliards) et mistralite (7 milliards). Cela correspond à ce à quoi je m’attendais, mais il est certain que la méthode d’évaluation LLM as a judge nécessite un contrôle car elle est bien évidemment sujette à des biais en ce qui concerne le jugement de la réponse.

## Tests

Django fournit ses propres outils de tests. Afin de réaliser les tests, on utilise la classe TestCase qui permet d’écrire des tests unitaires. TestCase est compatible avec les outils de test standard de Python. Django utilise une base de données en mémoire (SQLite) pour accélérer les tests. TestCase fournit un client HTTP intégré pour simuler des requêtes (GET, POST, etc.) et tester les vues et les réponses.

J’utilise aussi la bibliothèque MAgicMock. Conçue pour faciliter la création de faux objets (mocks) dans les tests. Ces objets simulés peuvent être utilisés pour remplacer des fonctions, des méthodes ou des objets réels lors de tests unitaires, afin d'isoler les fonctionnalités et de vérifier les interactions.

### Test de scrapping

Lors de la définition de la fonction de test, le décorateur patch permet de remplacer des objets ou des fonctions par des Mocks, les arguments passés sont fonctions à l’intérieur des décorateurs, respectivement du décorateur le plus bas au plus haut et de gauche à droite pour les arguments. Ici je lui passe la fonction pour initialiser BeautifulSoup (récupération du HTML correspondant à l’URL.) et la fonction pour récupérer toutes les URL liées à mes filtres.

Je crée un objet avec MagiMock qui représente beautifulsoup avec ses attributs. A l’intérieur un deuxième objet MagicMock simule la page html, vérifie la présence de la balise html passé dans lambda et retourne le texte inscrit dans le deuxième MagicMock. L’attribut side\_effect indique ce que le Mock est censé ressortir lors de l’appel.

Ensuite, évaluation du résultat en JSON via les méthodes héritées de TestCase. Ici via les méthodes de type Assert, on vérifie si le JSON est conforme, comme la présence des différentes clés ou bien le fait que le JSON contient une liste.

### Test d’import en base

Ici, on vérifie que les données sont bien présentes dans la base de données temporaire. On fait un appel à l’objet en base et on vérifie que la fonction d’import en base de données ait bien importé l’article en regardant un par un ses attributs.

### Test du CRUD

#### Test du read

L’utilisation du décorateur classmethod permet d’initialiser un objet article pour tous les tests évitant de ré exécuter la fonction pour chaque test. Les tests vérifient que l’objet est lisible pour l’url correspondante et que son affichage utilise le template correspondant

#### Test du create et de l’update

Les tests sont assez similaires avec la vérification de l’URL. L’attribut « refesh\_from\_db » permet de récupérer le dernier objet après la mise à jour de la base de données.

#### Test du delete

On vérifie que l’URL correspondante est bonne et que l’objet a bien été supprimé de la base de données.

-\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### Test du RAG

Dans la méthode de classe on utilise cls car il permet de modifier des attributs pour toutes les instances de classe et non une seule. On définit deux fonctions que l’on va « Mocker » : Search et encode. On attribue une valeur fictive à l’encodeur fictif. Le Mock\_response va créer une réponse fictive pour chaque réponse du moteur de recherche avec l’id, le titre et l’abstract de l’article.

Lorsqu’une recherche est exécutée le Mock\_search va renvoyer la réponse fictive du Mock\_response. Cela revient à simuler une recherche elasticsearch.

RequestFactory permet de créer une requête POST avec la query, l’index et l’URL du RAG. La requête est associée à l’utilisateur car il faut se connecter pour utiliser le RAG. Enfin, la requête est envoyée à la vue du RAG avec la requête fictive et on teste s’il n’y a pas eu d’erreur.

Ici le principe n’est pas réellement de tester si l’application fonctionne parfaitement mais simplement d’effectuer des tests sur certaines parties pour notamment découvrir les Mocks. Le code pourra être amélioré par la suite.

## Monitoring

Afin de réaliser le monitoring de l'application, j'utilise la plateforme Grafana.

### Grafana

Grafana est une plateforme open-source de surveillance et de visualisation de données en temps réel qui permet de collecter, analyser et présenter des métriques issues de diverses sources dans des tableaux de bord interactifs. Il offre une interface de visualisation (graphiques, tableaux, diagrammes) et s’intègre avec des outils comme : Prometheus (surveillance des systèmes), Elasticsearch, InfluxDB (base de données pour séries temporelles), MySQL/PostgreSQL, Cloud (AWS, Google Cloud, Azure). Il permet également de configurer des alertes en fonction de seuils et de les notifier via emails ou messageries.

Avantages

* Interface intuitive : Facile à prendre en main, même pour des utilisateurs non techniques.

Tableaux de bord attractifs et interactifs.

* Open-source et extensible : gratuit dans sa version de base. Communauté active qui développe des plugins et fournit du support.
* Alertes avancées : Détection proactive des problèmes avant qu'ils ne deviennent critiques.
* Flexibilité des sources de données : Prend en charge un large éventail de sources, ce qui en fait une solution polyvalente.

Inconvénients

* Courbe d'apprentissage : Nécessite un peu de temps pour maîtriser les fonctionnalités avancées (alertes, intégrations complexes).
* Dépendance aux sources de données : Ne collecte pas de données directement ; il s'appuie sur des outils comme Prometheus, Elasticsearch, ou d'autres systèmes.
* Gestion des performances : Avec des tableaux de bord très complexes et des sources de données volumineuses, les performances peuvent en souffrir.

Afin de collecter des métriques, j’utilise la plateforme Prometheus.

### Prometheus

Prometheus est une plateforme de monitoring open-source conçue pour collecter, stocker, et analyser des métriques en temps réel.

Chaque métrique est identifiée par un nom et des étiquettes.

Le langage de requête est PromQL, c’est lui qui permet d’extraire et d’analyser les métriques.

Prometheus stocke lui-même ses métriques. Il possède un système d’alerte basé sur les métriques collectées.

Avantages

* Optimisé pour les métriques : Haute performance pour des milliers de points de données par seconde.
* Écosystème riche : Large communauté et support pour Kubernetes et d'autres environnements cloud.
* Alerting natif : Alertmanager permet des alertes complexes sans outils tiers.

Limites

* Pas adapté aux logs ou données textuelles : Prometheus gère uniquement des métriques numériques.
* Rétention des données limitée : Pas conçu pour conserver les métriques sur de longues périodes.
* Scraping pull-based uniquement : Nécessite que les services exposent des endpoints.

Pour surveiller la performance et l’utilisation de mon application RAG, j’ai défini les métriques suivantes :

* *rag\_pipeline\_latency :* Il est appliqué en tant que décorateur de ma fonction RAG et mesure le temps total d’exécution de mon pipeline RAG.
* *search\_latency :* Mesure le temps total d’exécution de la recherche Elasticsearch.
* *llm\_latency :* Mesure le temps total d'exécution du LLM.
* *rag\_requests\_total :* Compte le nombre total de requêtes reçues par le pipeline.
* *rag\_errors\_total :* Compte le nombre total d'erreurs rencontrées dans le pipeline.

Les résultats sont visibles dans Grafana sous forme de graphiques en ajoutant les métriques.

J’utilise uptime kuma pour m’assurer que le conteneur fonctionne bien.

### Uptime Kuma

Uptime Kuma est un outil open-source de surveillance de services.

Voici ses principales caractéristiques :

* Surveillance variée :
  + HTTP(s), TCP, UDP, Ping, DNS, Push, Docker et plus.
* Interface utilisateur moderne : Interface claire et personnalisable.
* Notifications intégrées : Support pour Telegram, Discord, Slack, email, etc.
* Auto-hébergement : Facile à déployer via Docker ou manuellement.
* Tableau de bord complet : Pour voir les temps de réponse et les incidents.

J’ai configuré Uptime Kuma pour qu’il surveille l’état du conteneur django et envoie un mail sur mon adresse s’il y a déconnection du conteneur.

## Logs

Afin de détecter les éventuelles erreurs lors de l’exécution de l’application, j’ai implanté un mécanisme de gestion de ces dernières qui fonctionne de la manière suivante pour l’exécution de chaque fonction :

* Capturer l’erreur : évite de faire planter l’application.
* Journaliser l’erreur : On enregistre le message de l’exception puis la trace complète pour comprendre pourquoi l’erreur s’est produite.
* Envoyer un mail avec les détails de l’erreur : le mail contient le message de l’exception, sa trace.

Afin de réaliser ces étapes je crée un décorateur qui prend la fonction en argument et la retourne si elle fonctionne ou passe l’erreur à la fonction de journalisation si elle échoue.

## Déploiement

Afin de déployer ma solution, j’ai besoin d’encapsuler tout ce dont mon application a besoin pour fonctionner : le code, les bibliothèques, les dépendances et les configurations. Pour réaliser ça, j’utilise des conteneurs. Cela va lui permettre de marcher aussi bien sur un serveur de production qu’en local. Dans un conteneur, l’environnement et les ressources restent les mêmes quelle que soit la machine sur laquelle il est déployé. Les conteneurs permettent également les architectures de micro services ici : Django, Postgres, elasticsearch… où chaque service est exécuté dans son propre conteneur. Voici les principaux outils de conteneurisation.

### Outils de conteneurisation

#### Docker

Docker est une plateforme open-source permettant de créer, de déployer et d'exécuter des applications dans des conteneurs.

Ces principales caractéristiques sont :

* Portabilité :
  + Les conteneurs Docker fonctionnent sur n’importe quel environnement avec Docker installé (Windows, MacOs, Linux).
* Isolation :
  + Chaque conteneur est isolé, avec ses propres ressources (CPU, mémoire, stockage) et ne partage pas ses processus avec d'autres conteneurs.
* Efficacité :
  + Contrairement aux machines virtuelles (VM), Docker utilise le même noyau que l’hôte, ce qui réduit les besoins en ressources.
* Écosystème riche :
  + Docker Hub propose des milliers d’images prêtes à l’emploi (bases de données, serveurs web, environnements de développement).

Les principaux composants de docker sont :

* Les images : elles contiennent tout ce dont une application a besoin pour fonctionner.
* Les conteneurs : Instances exécutables des images Docker. Ils encapsulent l’application avec leurs dépendances dans un environnement isolé.
* Dockerfile : Un fichier texte contenant les instructions pour construire une image Docker.
* Docker Compose : Un outil pour définir et gérer plusieurs conteneurs à l’aide d’un fichier docker-compose.yml.
* Docker Hub : Une plateforme pour partager et télécharger des images Docker.

#### Podman

Comme Docker, Podman est un outil de conteneurisation qui s'adresse aux environnements nécessitant une sécurité renforcée et il nécessite moins de dépendance que Docker.

#### Kubernetes (K8s)

Permet l’orchestration de micro services dans des environnements cloud (AWS, GCP, Azure) et le déploiement d'applications complexes avec plusieurs conteneurs.

J’ai choisi d’utiliser Docker avec le kubernetes d’Azure car j’ai plusieurs conteneurs à déployer sur Azure. Pour docker, je l’utilise en entreprise et je suis plus familier.

Azure est une plateforme de cloud computing développée par Microsoft. Elle offre une large gamme de services pour héberger, gérer et déployer des applications et infrastructures à l’échelle mondiale.

Voici les principales catégories proposées par Azure :

* Informatique et Virtualisation
  + Azure Virtual Machines (VMs) : Machines virtuelles pour exécuter des applications sur Linux ou Windows.
  + Azure Kubernetes Service (AKS) : Service managé pour exécuter des applications en conteneurs avec Kubernetes.
  + Azure Functions : Permet d’exécuter du code sans gérer de serveurs (serverless computing).
* Stockage et Bases de Données
  + Azure Blob Storage : Stockage d’objets évolutif pour les fichiers, images, vidéos, etc.
  + Azure SQL Database : Base de données relationnelle managée (SQL Server dans le cloud).
  + Azure Cosmos DB : Base de données NoSQL haute performance.
* Réseau et Sécurité
  + Azure Virtual Network (VNet) : Création de réseaux privés dans Azure.
  + Azure Firewall : Protection avancée contre les attaques réseau.
  + Azure DDoS Protection : Sécurisation contre les attaques DDoS.
* Intelligence Artificielle et Big Data
  + Azure AI Services : Inclut des services IA comme Cognitive Services (vision, reconnaissance vocale, NLP).
  + Azure Machine Learning : Développement et déploiement de modèles de machine learning.
  + Azure OpenAI Service : Accès aux modèles GPT, comme ceux de ChatGPT.
* Développement et DevOps
  + Azure DevOps : Outils pour CI/CD, gestion de code, et pipelines automatisés.
  + Azure App Service : Déploiement rapide d’applications web et mobiles.
  + Azure Logic Apps : Automatisation des workflows d’entreprise.
* Analyse de Données
  + Azure Synapse Analytics: Plateforme de Big Data et Data Warehousing.
  + Azure Data Factory : Orchestration de flux de données.
  + Azure Databricks : Analytique avancée et Machine Learning basé sur Apache Spark.

Comme indiqué précédemment, j’utilise le kubernet d’Azure. J’ai automatisé le déploiement dans le fichier déployment.yaml.

Ce fichier gère le déploiement et la gestion des Pods. Il assure que le bon nombre de réplicas (instances) tourne et peut redémarrer les Pods en cas d’échec.

ci dessous, pas très lisible en fait, tu penses que tu peux faire un diagramme avec les mêmes infos?

Voici la structure type d’un déploiement Kubernetes

* apiVersion: apps/v1 : Indique la version de l’API Kubernetes utilisée pour le déploiement.
* kind: Deployment : Spécifie que cette ressource est un déploiement.
* metadata.name : Nom unique du déploiement.
* spec.replicas : Nombre d'instances (Pods) à exécuter pour assurer la haute disponibilité.
* spec.selector.matchLabels : Définit l’étiquette utilisée pour associer ce déploiement aux Pods correspondants.

Définition des Pods :

* template.metadata.labels : Définit les labels appliqués aux Pods créés par ce déploiement.
* template.spec.containers : Liste des conteneurs à exécuter dans chaque Pod.
* containers.name : Nom du conteneur.
* containers.image : Image Docker utilisée pour ce conteneur.
* containers.ports.containerPort : Port sur lequel l’application écoute à l'intérieur du conteneur.
* containers.env : Variables d’environnement pour configurer l’application.
* containers.volumeMounts : Monte un volume dans le conteneur pour stocker des fichiers persistants.

Gestion des Volumes :

* volumes.name : Nom du volume défini dans le déploiement.
* volumes.secret.secretName : Utilisation d’un Secret Kubernetes pour stocker des données sensibles (ex: mots de passe).

Afin de faire communiquer les pods entre eux et de rendre accessible le service depuis l’extérieur, j’ai créé le fichier service.yaml. Il distribue aussi le trafic entre les différentes instances d’une application.

Chaque Service suit cette structure :

* apiVersion: v1 : C'est une ressource de base de Kubernetes.
* kind: Service : On définit un Service pour exposer une application.
* metadata.name : Nom unique du Service.
* spec.selector : Permet de lier le Service aux Pods ayant le même label.
* ports :
* port : Port sur lequel le Service est accessible.
* targetPort : Port sur lequel l'application écoute dans le Pod.
* protocol: TCP : Communication en TCP.

### Conteneurisation

Pour préparer mon application au déploiement, j’ai conteneurisé chaque micro-services via le fichier *docker-compose.yaml .*

#### Service django

D’abord, je crée le conteneur principal qui va contenir Django et toutes les dépendances dont j’ai besoin pour exécuter mon application.

Pour construire mon image, j’utilise un Dockerfile : *Dockerfile.django.* Il va faire appel à une autre image que j’ai défini dans Dockerfile.dependencies et que j’ai envoyé sur Docker Hub qui contient toutes les bibliothèques, ainsi que le langage nécessaire pour faire fonctionner mon application. Cela me permet d’éviter de réinstaller toutes les bibliothèques et Python lorsque je reconstruis les conteneurs. Ensuite je crée et je construis un répertoire de travail dans lequel je copie les fichiers de mon application. J’utilise un script Shell pour exécuter toutes les commandes Django nécessaire à l’initialisation de mon application : la création de la base de données, celle du super utilisateur (admin), le remplissage de la base de données et l’indexation des articles dans elasticsearch. Enfin, le script lance le serveur Django.

Après avoir créé l’image django, on définit les ports, on lui passe toutes les variables d’environnements, on monte le code source local dans le conteneur pour synchroniser les modifications et on indique que ce conteneur dépend d’autres conteneurs, notamment elasticsearch, prometheus, db (PostgreSQL), et ollama.

Ci-dessous le lien vers mon docker-compose.

<https://github.com/JoachimLombardi/Projet_chef_d-oeuvre/blob/master/pubmed_analyze/docker/docker-compose.yml>

#### Service elasticsearch

Ce conteneur va fournir Elasticsearch, définir ses ports. Le configurer en mono nœud et définir les options mémoire pour JAVA. Enfin, il va monter un répertoire local. Il va également construire une image personnalisée d’Elasticsearch, via Dockerfile.elasticsearch, qui inclut le plugin Elastiknn, un plugin conçu pour effectuer des recherches de similarité approximative (ANN, Approximate Nearest Neighbors) dans Elasticsearch. Le Dockerfile va ajouter le plugin à Elasticsearch. L’avantage de ANN par rapport à KNN c’est les temps de calculs permettant une vitesse accrue en sacrifiant légèrement la précision.

#### Service elasticsearch\_exporter

Il va exporter les métriques pour Prometheus, définir le port et configurer l’URL d’accès à Elasticsearch.

#### Service db

Fournit la base de données Postgres, définit l’environnement et le port et monte le répertoire.

#### Service postgres\_exporter

Exporte les métriques PostgreSQL pour Prometheus. Définit le port pour les métiques.

#### Service ollama

Service utilisant l’image d’ollama, définit les ports et monte le répertoire.

#### Service prometheus

Service de surveillance qui collecte et stocke les métriques des services comme PostgreSQL ou Elasticsearch. Définit le port pour accéder à l'interface de Prometheus. Monte un fichier de configuration prometheus.yml et un répertoire pour les données.

Le fichier prometheus.yml est une configuration pour Prometheus. Prometheus va interroger toutes les cibles définies dans scrape\_configs toutes les 15 secondes pour récupérer les métriques.

#### Service grafana

Service pour fournir des visualisations des métriques collectées via Prometheus. Définit le port. Il configure un service de rendu d'images pour les graphiques via le service renderer.

#### Service renderer

Service complémentaire pour Grafana permettant de générer des images, des graphiques.

#### Service uptime-kuma

Service de surveillance pour vérifier la disponibilité des autres services. Définit le port. Monte des répertoires locaux pour les données et l'accès au socket Docker. Monter le Docker socket dans le conteneur Uptime Kuma permet à cet outil de surveiller et de gérer les conteneurs Docker directement.

#### Réseau networks

Un seul réseau est défini sous le nom django\_network. Tous les services sont connectés à ce réseau par défaut, ce qui leur permet de communiquer via leurs noms de conteneur.

## Outils de versionnement et CI/CD

### Outils de versionnement

Le versionnement est le processus de gestion des changements dans les fichiers d’un projet, en particulier dans le développement logiciel. Il permet de suivre l'historique des modifications, de collaborer efficacement, et de revenir à des états précédents si nécessaire.

Il existe plusieurs outils de versionnement dont voici les principaux :

#### Git

Git est un outil de versionnement open source qui permet à plusieurs développeurs de collaborer sur un projet, de créer des branches pour des fonctionnalités spécifiques, et de fusionner les modifications dans la branche principale. (Une branche est une copie indépendante du code source, qui permet de travailler sur des fonctionnalités ou des correctifs sans affecter la version principale du projet.). Il fonctionne localement et permet de synchroniser le code local avec un dépôt distant GitHub.

GitHub est une plateforme d’hébergement de code basée sur Git, permettant aux développeurs de :

* Collaborer sur des projets en équipe.
* Gérer le versionnement avec Git.
* Automatiser des workflows avec GitHub Actions.
* Partager des projets open source ou privés.

Il est utilisé par des millions de développeurs dans le monde et est gratuit pour les projets open source.

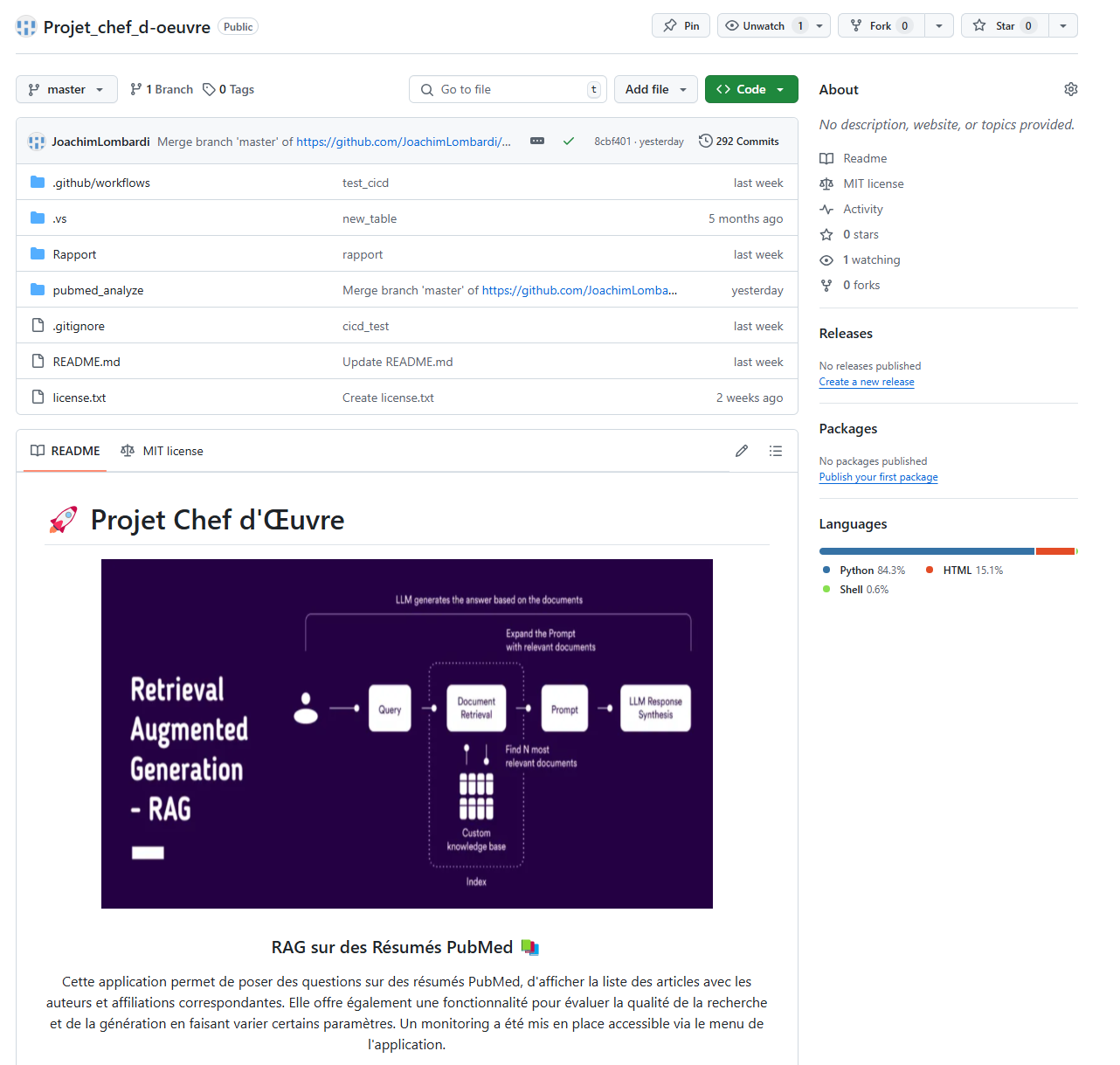


Figure 3. Mon projet sur github.

#### Subversion (SVN)

Subversion est également un système de versionnement mais contrairement à Git, les fichiers sont stockés sur un serveur centralisé, et les développeurs récupèrent une copie locale pour y travailler.

#### Mercurial

Mercurial est un autre système de contrôle de version décentralisé, similaire à Git, connu pour sa simplicité et ses performances. Il est plus intuitif que Git et supporte les grands projets avec des millions de fichiers.

En raison de sa popularité (les plateformes d’hébergement GitHub sont très présentes sur le web) et de mon expérience et du fait que mon entreprise l’utilise j’ai choisi Git.

### CI/CD

Le CI/CD (Continuous Integration/Continuous Delivery ou Continuous Deployment) est une méthodologie de développement logiciel qui vise à automatiser et optimiser les étapes de construction, test, déploiement, et livraison des applications.

* CI (Continuous Integration) :
  + Processus d'intégration continue.
  + Objectif : intégrer régulièrement (quotidiennement ou plus fréquemment) le code des développeurs dans une branche partagée (généralement main ou develop).
* Avantages :
  + Tests automatisés pour détecter rapidement les bugs.
  + Réduction des conflits entre branches.
* CD (Continuous Delivery) :
  + Processus de livraison continue.
  + Objectif : préparer automatiquement le code pour qu'il puisse être déployé à tout moment.
  + Les étapes incluent : tests automatisés, packaging (build), et déploiement dans un environnement de pré-production ou staging.
* CD (Continuous Deployment) :
  + Extension de la livraison continue.
  + Objectif : déployer automatiquement en production après validation des tests, sans intervention humaine.
* Avantages : déploiements fréquents et rapides.

Il existe de nombreux outils pour réaliser le CI/CD (Jenkins, GitHub Actions, GitLab CI/CD…). J’utilise GitHub Actions car mon code est hébergé sur GitHub.

Ici le CI/CD est utilisé pour réaliser les tests lorsque le code est poussé sur GitHub.

Pour réaliser le CI/CD, on utilise GitHub Actions.

GitHub Actions est une plateforme d'intégration et de livraison continue (CI/CD) intégrée directement à GitHub. Elle permet d'automatiser des workflows (processus) en fonction d'événements tels que des push, pull requests, ou des déclencheurs programmés. Avec GitHub Actions, on peut configurer des pipelines pour tester, construire et déployer des applications.

Les instructions pour le CI/CD se font dans le dossier : «.github/workflows » que l’on créer via github.

On définit l’action qui va déclencher le workflow et un environnement de travail ici Ubuntu et on clone le dépôt dans l’environnement d’exécution. On installe docker, on crée les conteneurs et enfin on exécute les tests.

Le résultat apparaît dans GitHub Actions sous forme d’un terminal qui affiche la liste des actions effectuées. Si le workflow est réussi, un macaron vert apparaît sur GitHub à côté du workflow.

## Conduite agile de projet.

Afin de faire de la gestion de projet, il existe plusieurs outils dont voici les principaux :  
Voici les principales méthodes de gestion de projet agile :

### SCRUM

Scrum est une méthodologie Agile qui vise à améliorer la flexibilité, la collaboration et l’efficacité dans le développement de produits, notamment en IT et en Machine Learning. Il repose sur des cycles courts appelés sprints, permettant une adaptation rapide aux changements.

Principaux Rôles dans Scrum :

* Scrum Master → Facilite le processus, élimine les obstacles et assure le respect des principes Scrum.
* Product Owner → Définit les besoins du produit et priorise les tâches dans le backlog.
* Équipe de développement → Développe, teste et livre les fonctionnalités du produit.

Événements Clés de Scrum :

* Sprint Planning → Planification des tâches à accomplir pendant le sprint (1 à 4 semaines).
* Daily Scrum (Stand-up Meeting) → Réunion quotidienne de 15 minutes pour synchroniser l’équipe.
* Sprint Review → Présentation des fonctionnalités développées à la fin du sprint.
* Sprint Retrospective → Analyse des points forts et axes d’amélioration pour le sprint suivant.

Artefacts Scrum :

* Product Backlog → Liste des fonctionnalités à développer.
* Sprint Backlog → Liste des tâches sélectionnées pour un sprint.
* Increment → Version améliorée du produit après chaque sprint.

SCRUM est la méthode agile la plus populaire.

### Kanban

Kanban est une méthode Agile similaire à Scrum mais plus flexible et sans cycles fixes (sprints). Elle se concentre sur la gestion des flux de travail et l'amélioration continue.

Principaux Concepts :

* Tableau Kanban avec trois colonnes principales : À faire, En cours, Terminé.
* Limite de travail en cours (WIP) : Limiter le nombre de tâches dans une colonne pour éviter la surcharge.
* Amélioration continue : Revue régulière des processus pour améliorer le flux de travail.

Avantages :

* Flexibilité et adaptabilité pour les équipes en évolution.
* Suivi visuel des tâches en temps réel.

Outils populaires :

* Trello (qui propose aussi un mode Kanban)
* Jira

### XP (Extreme Programming) : Méthode Agile Technique.

XP (Extreme Programming) se concentre sur l'amélioration de la qualité du code et l'agilité technique. C'est une approche fortement utilisée pour les projets de développement logiciel.

Principaux Concepts :

* Développement itératif : Des cycles de développement courts pour livrer rapidement des versions du produit.
* Code de qualité : Utilisation de pratiques comme la revue de code, les tests automatisés et l'intégration continue.

Avantages :

* Haute qualité du code.
* Adaptation rapide aux besoins du client.

Outils populaires :

* Jenkins (pour l'intégration continue)
* GitLab ou GitHub (gestion du code)

La méthode SCRUM est la méthode la plus pratique lors de travail en équipe car elle permet de visualiser et corriger l’évolution du projet à tout instant, cependant dans ce cas précis le projet à été réalisé en solo à titre individuel ainsi il n’y a jamais eu de présentation au client ni à l’équipe. A noter, que ces méthodes peuvent être complémentaires car Kanban permet de visualiser le SCRUM s’il y en a un. Ci-dessous une présentation des principaux outils de visualisation pour gérer les tâches :D.

### Trello

Trello est un outil de gestion de projet visuel, basé sur la méthodologie Kanban. Il est souvent utilisé pour gérer les sprints Scrum de manière simple et intuitive.

Structure d’un Tableau Trello

* Tableau → Représente un projet.
* Listes → Représentent les étapes du projet (ex: À faire, En cours, Terminé).
* Cartes → Représentent les tâches spécifiques (ex: “Déployer un modèle ML”).
* Étiquettes, deadlines et membres assignés → Facilitent l’organisation des tâches.

Exemple d’Utilisation de Trello dans Scrum

* Product Owner ajoute les tâches dans le Product Backlog.
* L’équipe choisit les tâches pour le sprint et les place dans la liste "À faire".
* Pendant le sprint, les tâches passent par "En cours" puis "Terminé" après validation.

### Jira

Jira, développé par Atlassian, est l'un des outils les plus populaires dans le domaine Agile et Scrum. Il est particulièrement adapté pour des projets logiciels ou des équipes de développement.

Caractéristiques principales :

* Suivi détaillé des sprints et des backlogs.
* Possibilité d’intégrer des tableaux Kanban et Scrum pour la gestion des tâches.
* Outils de gestion de version et de test intégrés.

Avantages :

* Très adapté aux équipes techniques.
* Suivi détaillé des problèmes, tâches et bugs.

Inconvénients :

* Peut être complexe pour des projets non techniques.
* Courbe d’apprentissage.

### Asana : Outil de Gestion de Projet Collaboratif

Asana est une alternative populaire pour la gestion des tâches et la collaboration d’équipe. Il peut être utilisé pour différents types de projets, y compris ceux en Agile et en Scrum.

Caractéristiques principales :

* Suivi des tâches avec des listes de tâches ou des tableaux Kanban.
* Assignation des tâches, priorisation, et suivi des progrès.
* Collaboration en temps réel avec commentaires et pièces jointes.

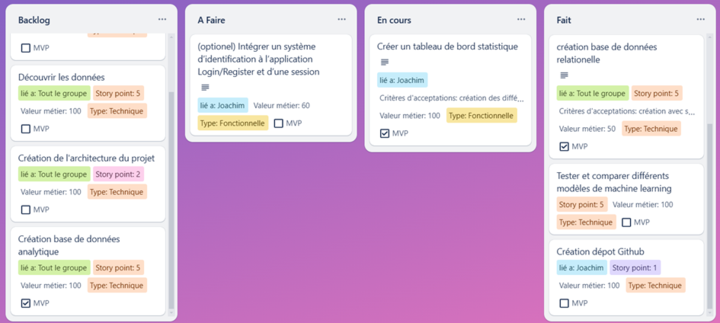
Avantages :

* Interface conviviale et flexible.
* Collaboration fluide pour les équipes.

Inconvénients :

* Moins de fonctionnalités spécifiques pour les équipes de développement logiciel.
* Manque de certaines intégrations avancées.

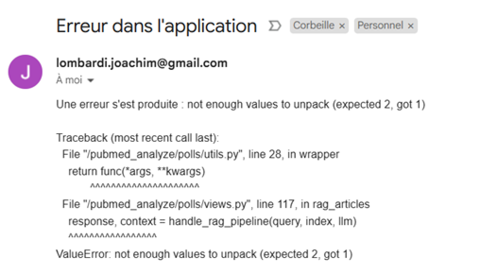
Trello est l’outil le plus populaire, il est simple et complet de plus je l’ai déjà utilisé dans le cadre de ma formation. Voici un Trello réalisé dans un projet antérieur :



Les dates n’y sont pas présentes mais on peut y voir les différentes tâches à effectuer, celui ou ceux qui ont à l’effectuer, les story points représentant l’effort à fournir pour réaliser la tâche et les valeurs métiers désignant l’importance de la tâche pour le client ou l’entreprise. Également, si la tâche est essentielle au MVP (Produit Minimum Viable) à savoir une version contenant les parties essentielles du projet.

Exemple d’une erreur.

Lorsque mon programme a été exécuté, j’ai reçu l’email suivant :



J’ai donc ouvert un ticket sur GitHub. L’erreur indique que deux valeurs sont attendues mais qu’il n’en a eu qu’une. Ainsi, la fonction handle\_rag\_pipeline renvoie une valeur en sortie mais deux variables sont proposées. Après analyse, il y avait deux erreurs, la première entraînant l’exception à savoir le mot « llm » a été passé dans les paramètres de la fonction generation à la place de « model ». Ce qui a entraîné une exception mais cette exception renvoie une réponse JSON et non deux variables.

Ton rapport est très bien et c’est du bon boulot par contre essaie de faire apparaître les compétences qui sont validées au fur et à mesure pour simplifier le travail du jury (pareil pour tes slides)

il manque souvent les spécifications (en amont) technique des projets, on ne comprends pas vraiment comment les apis sont découpées de l’application

ca manque d’image pour le moment pour rendre le contenu plus léger et voir le rendu de l’application et de ses modules (grafana par exemple c’est joli les dashboards)

numéro de page