

# Your News Anchor

E1 - Mise en situation

Développeur en Intelligence Artificielle, titre professionnel enregistré au RNCP - École IA  
Microsoft by Simplon



par Vincent Papelard

9 février 2024

# Table des Matières

<b>Introduction</b>	<b>1</b>
<b>1 Présentation des outils, plateformes et langages utilisés</b>	<b>1</b>
<b>2 Gestion du projet</b>	<b>1</b>
<b>3 Les composants du projet</b>	<b>1</b>
3.1 Collecte des données . . . . .	2
3.2 Nettoyage des données . . . . .	3
3.3 Stockage des données . . . . .	4
3.4 Accès aux données et API . . . . .	5
<b>4 Déploiement</b>	<b>5</b>
4.1 Base de données . . . . .	5
4.2 API . . . . .	5
4.3 Script d'extraction des données . . . . .	6
<b>Conclusion</b>	<b>6</b>
<b>Annexes</b>	<b>7</b>

# Introduction

Dans un contexte de diffusion rapide de l'information via Internet et face à la multitude de sources d'informations disponibles, il est parfois difficile de savoir où s'informer. Où trouver les actualités qui nous intéressent vraiment, en allant à l'essentiel ? C'est la problématique à laquelle répond ce projet qui nous est confié par notre client, un gros acteur dans le domaine des médias.

**Your News Anchor** est une application web qui offre à ses utilisateurs un résumé concis des actualités du jour. Ce résumé est automatiquement généré par IA à l'aide d'une liste de sources personnalisable par l'utilisateur, puis présenté par une voix synthétisée comme un vrai journal télévisé. Ce dossier se penche sur toute la partie du projet relative à la collecte, au nettoyage et au stockage des données nécessaires à la génération automatique de résumés. La totalité du code est disponible sur GitHub :

- Collecte/nettoyage de données : [https://github.com/vinpap/your\\_news\\_anchor](https://github.com/vinpap/your_news_anchor)
- API de la base de données : [https://github.com/vinpap/your\\_news\\_anchor\\_db\\_api](https://github.com/vinpap/your_news_anchor_db_api)

Notre client souhaite que le projet soit terminé dans un délai d'un mois. Pour cela, il nous alloue un budget de 4200€. Ce budget couvre uniquement nos honoraires, soit 21 jours travaillés facturés 200€ chacun. Tous les frais réguliers qui devront être payés suite à la livraison du projet, tels que le paiement d'abonnement à des services de cloud, sont à la charge de notre client.

## 1 Présentation des outils, plateformes et langages utilisés

Les outils et plateformes suivants ont été utilisés dans le cadre de ce projet :

- Développement : **Python**
- Base de données : **PostgreSQL**
- Visualisation de la base de données : **DBeaver**
- Plateformes de déploiement : **Microsoft Azure, PythonAnywhere**
- Versionnage et gestion de projet : **GitHub**
- Prise de notes : **Obsidian**
- Rédaction du rendu : **LATeX**
- Diagrammes et schémas : **draw.io**

## 2 Gestion du projet

Le projet a été réalisé au cours de plusieurs sprints d'une semaine chacun. GitHub a été utilisé pour assurer le suivi des tâches et des sprints, ainsi que le versionnage du code.

## 3 Les composants du projet

Le projet est divisé en trois composants présentés dans la figure 2. Ces composants assurent les tâches suivantes :

- la collecte des données
- le nettoyage des données récupérées

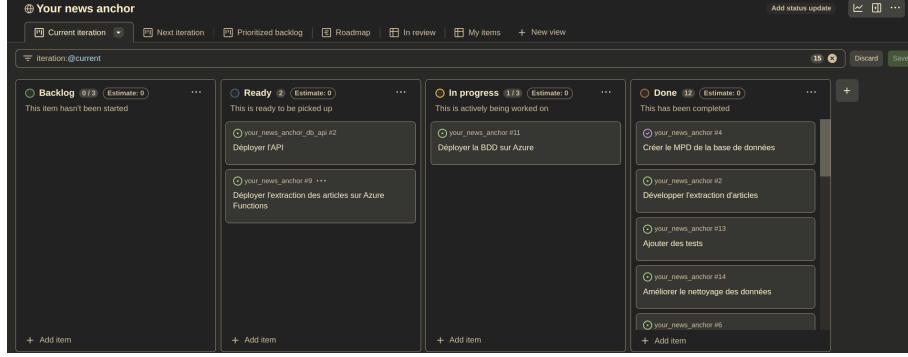


Figure 1: Tableau des tâches de GitHub Projects

- le stockage des données pour que l'application puisse les utiliser

Chaque composant est développé et déployé indépendamment des deux autres, selon une architecture de microservices.



Figure 2: Fonctionnement du script d'extraction des articles

### 3.1 Collecte des données

La collecte des données est réalisée par un script Python destiné à être exécuté à intervalles réguliers (une fois par jour, par exemple) afin que les articles stockés dans la base de données soient à jour. La collecte des données est réalisée en plusieurs étapes. Tout d'abord, le script va récupérer une liste de sources d'informations d'où les articles seront extraits. Chaque source d'information consiste en un flux RSS stocké dans la base de données (cf partie sur le stockage des données pour plus d'informations). Le script va donc commencer par appeler l'API pour récupérer les liens des flux RSS à utiliser. On va ensuite utiliser le module **feedparser** de Python pour parcourir les flux RSS et en extraire une liste d'URLs d'articles avec d'autres informations telles que le titre de chaque article. Dans la mesure où l'objectif final est de réaliser des résumés concis des actualités du jour, on ne récupère pas la totalité des articles de chaque flux RSS. À la place on récupère les n premiers articles de chaque flux (les premiers articles correspondent souvent aux gros titres du jour, et ont donc plus de chance d'être pertinents). Enfin, on récupère le code HTML des pages web de chaque article et on en extrait le texte de l'article ainsi que plusieurs métadonnées utiles (auteurs, date de publication...).

Cette étape représente un défi particulier. En effet, l'approche traditionnelle consiste à identifier dans le code HTML les classes et id qui permettent de localiser sur la page web le contenu que l'on souhaite extraire, en utilisant des outils tels que BeautifulSoup. Or, la structure de chaque page web est différente. Si l'on voulait collecter le contenu des articles de façon fiable, il faudrait techniquement écrire du code propre à chaque site web. Cette approche a plusieurs inconvénients qui la rendent inadaptée à nos besoins :

- Écrire un code différent pour chaque source de données est un travail long et laborieux. De plus certains sites nous compliquent la tâche, par exemple en obfuscant les noms des classes et des id dans leur code source
- L'application prévoit que les utilisateurs puissent eux-mêmes rajouter des sources de données. Cela est impossible si le code ne nous permet de gérer qu'une liste de sites web fixe
- Par ailleurs, cette technique est très peu robuste. Le moindre changement apporté au code source des pages web peut rendre notre code obsolète.

Pour toutes ces raisons, on préférera s'orienter vers d'autres méthodes. L'idéal serait de pouvoir coder un "web scraper universel", qui nous permettrait d'extraire le texte de n'importe quel article publié sur une page web quelconque. Cette tâche est extrêmement difficile à mettre en place en pratique, mais il est néanmoins possible de s'en approcher grâce au package Python [newspaper](#), qui est utilisé dans le cadre de ce projet. Newspaper est un package créé pour extraire et traiter des articles depuis des pages web. Il permet ainsi d'extraire un article depuis une page web en quelques lignes :

```
from newspaper import Article

url = 'https://url-de-votre-article.com'
article = Article(url)
article.download()
article.parse()
text = article.text
```

Newspaper fonctionne en analysant le code des pages web avec BeautifulSoup. Certaines informations (titre, date...) sont retrouvées en cherchant des balises spécifiques telles que la balise <h1> par exemple, où grâce à des expressions régulières. Newspaper extrait aussi d'autres informations telles que les mots-clés d'un article grâce au module de NLP nltk. Dans notre cas, on récupérera les informations suivantes :

- le texte de l'article
- les auteurs
- la date de publication
- l'url de l'image en une de l'article (s'il y en a une)

Il est important de noter que cette solution n'est pas parfaite. Certains types de contenu, tels que les vidéos, ne sont pas exploitables. Newspaper a aussi du mal à traiter certains sites particuliers (il échoue à récupérer les articles du site 'L'Équipe', par exemple). Malgré tout, les résultats obtenus ont été concluants avec la grande majorité des sites d'information testés, en français comme en anglais.

## 3.2 Nettoyage des données

Après avoir collecté les articles, le script Python va passer par une étape de nettoyage des données. Malgré la bonne qualité générale du texte extrait, certains types de texte ne sont pas exploitables pour générer des résumés :

- Les reportages vidéo, qui ne comportent généralement qu'un bref texte de description sous la vidéo
- Les articles réservés aux abonnés, où seul le début du texte est visible
- Les flashes info et autres articles très (trop) courts pour en faire un résumé, de manière générale.

Les noms des auteurs automatiquement extraits par newspaper posent également problème. La liste d'auteurs générée identifie souvent de façon erronée des auteurs dans le corps de l'article, et il n'est pas rare d'obtenir des listes d'auteurs telles que ['Par', 'Julien Dupont', 'Le Président'], par exemple.

Le premier problème peut être résolu simplement : il suffit de filtrer les articles en fonction de leur longueur, pour ne garder que ceux dont le nombre de caractères dépasse une certaine seuil. En ce qui concerne la liste des auteurs, une solution un peu plus complexe a été trouvée : réaliser une détection d'entités dans la liste à l'aide de **Spacy** pour ne garder que les chaînes de caractère qui correspondent à des noms propres qui désignent des personnes. Cela a permis d'éliminer la quasi-totalité des erreurs d'extraction de noms d'auteur.

### 3.3 Stockage des données

Toutes les informations extraites (articles et métadonnées) ainsi que la liste des sources d'information à traiter et toutes les autres données nécessaires au fonctionnement de l'application Your News Anchor sont stockées sur une base de données **PostgreSQL**. Le Modèle Physique de Données qui décrit la structure de notre base de données est présenté dans la figure 3.

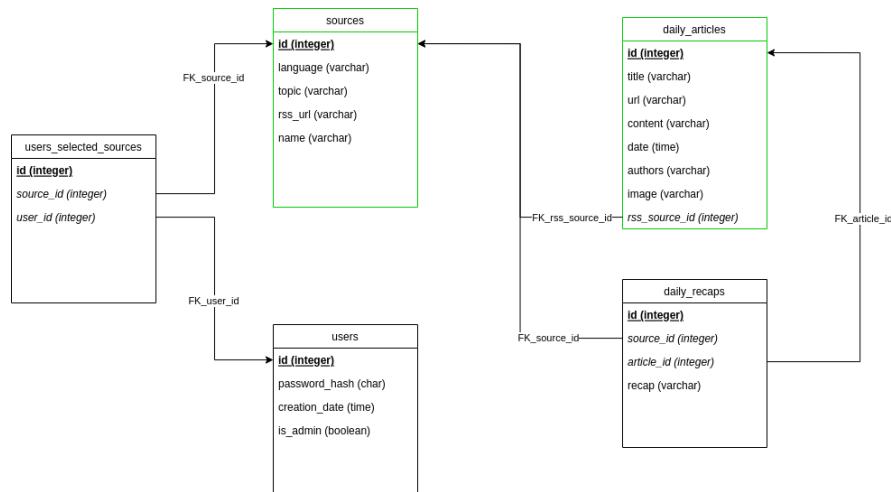


Figure 3: MPD de notre base de données

**Remarque :** seules les tables 'daily\_articles' et 'sources' sont exploitées dans ce dossier. Les autres tables sont utilisées pour gérer d'autres aspects de notre application et sortent du cadre de ce E1.

La base de données est composée des tables suivantes :

- **sources** : contient une liste de sources d'information, avec le lien vers leur flux RSS. Cette liste est définie par les administrateurs de l'application et ne peut pas être modifiée par les utilisateurs. Les flux RSS disponibles dans cette table peuvent être utilisés par n'importe quel utilisateur
- **users\_selected\_sources** : cette table recense les flux RSS ajoutés manuellement par les utilisateurs. Seul l'utilisateur qui a ajouté un flux peut l'incorporer dans sa liste de sources d'informations

- *users* : la table qui contient les informations concernant les utilisateurs (identifiant, hash du mot de passe...)
- *daily\_articles* : contient tous les articles extraits par le script de collecte des données. Le contenu de cette table est renouvelé à intervalles réguliers (par exemple toutes les 24 heures), lorsque le script de collecte des données est exécuté
- *daily\_recaps* : la table où sont stockés les résumés d'articles générés. Ces résumés sont eux aussi renouvelés à intervalles réguliers.

### 3.4 Accès aux données et API

Afin de simplifier l'accès aux données, une API a été développée pour exposer la base de données PostgreSQL. Cela nous évite de devoir gérer nous-mêmes les connexions et requêtes SQL dans notre application, et nous permet de mettre en place des sécurités pour contrôler qui accède à nos données et/ou les modifie. L'API a été développée à l'aide de **FastAPI**, un framework Python destiné à développer rapidement des API REST performantes. Cette API propose deux endpoints pertinents dans le cadre de notre dossier :

- **/feeds**, qui permet de récupérer (mais pas de modifier) le contenu de la table 'sources'
- **/update\_articles**, qui sert à renouveler entièrement le contenu de la table 'daily\_articles'. Cette opération implique d'effacer le contenu existant de la table pour le remplacer par les données envoyées dans un objet JSON joint à la requête.

Dans la mesure où la requête envoyée à ce deuxième endpoint supprime des données, il a fallu sécuriser l'accès à la base de données. Dans ce but, une clé d'API secrète doit être envoyée avec la requête. Cette clé est stockée dans une variable d'environnement afin de ne pas être visible "en clair" sur un dépôt GitHub, par exemple. Plus d'informations sont disponibles dans la section consacrée au déploiement du projet.

## 4 Déploiement

Comme évoqué plus haut, les trois composants de ce projet sont développés et déployés indépendamment les uns des autres.

### 4.1 Base de données

La base de données PostgreSQL a d'abord été mise en place localement pendant la période de développement. Elle a ensuite été exportée via un script SQL sur le service Azure Database for PostgreSQL de Microsoft Azure. Pour plus de sécurité, la base de données a été paramétrée de sorte à n'autoriser l'accès qu'à une adresse IP spécifique, où est déployée notre API.

### 4.2 API

L'API est elle aussi déployée sur Microsoft Azure, plus spécifiquement sur le service **Azure Web App**. Le déploiement est automatisé grâce à GitHub Actions afin que les mises à jour apportées au code soient automatiquement déployées sur Azure. Comme nous l'avons vu, l'API a besoin de définir une clé secrète pour fonctionner. Cependant, il est impossible d'inclure cette clé dans notre code pour des raisons évidentes : celle-ci serait visible de tous sur notre dépôt GitHub ! Pour régler ce problème, on utilisera des secrets GitHub. Cette fonctionnalité de GitHub nous permet de créer des valeurs secrètes liées à notre dépôt mais dont la valeur est cachée. On peut ensuite ajouter les lignes suivantes au fichier YAML qui gère le pipeline de déploiement de GitHub action :

```
- name: Create secret values in environment (APi token and DB password)
run: |
  export API_TOKEN=${{ secrets.SECRET_API_TOKEN }}
  export DB_PWD=${{ secrets.SECRET_DB_PWD }}
```

Lors du déploiement, ces instructions créeront sur le serveur des variables d'environnement qui contiennent nos valeurs secrètes (le mot de passe de la base de données et notre token d'API) sans pour autant exposer leur valeur dans le code.

### 4.3 Script d'extraction des données

Notre script d'extraction d'articles doit être automatiquement exécuté à intervalles réguliers. La solution retenue a été de l'exécuter sur [PythonAnywhere](#). PythonAnywhere est une plateforme basée sur AWS qui permet de déployer des scripts et applications Python avec un minimum d'efforts.

## Conclusion

Ce dossier a couvert toutes les problématiques liées à la mise à disposition des données dans le cadre du projet Your News Anchor. Il a notamment mis en évidence les difficultés liées à la mise en place d'une solution de web scraping universel, et les réponses qui peuvent être apportées à ce problème. L'utilisation de packages tels que newspaper, qui combinent techniques de NLP et analyse de code HTML, en est un exemple.

## Annexes

### Architecture générale du projet

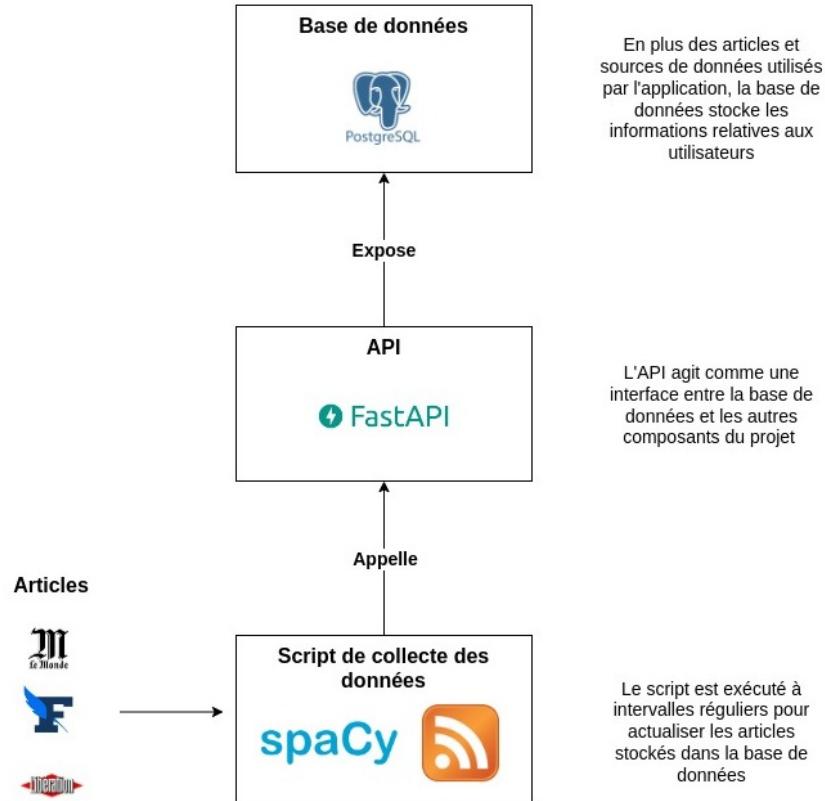


Figure 4: Les différents composants du projet

### Sources de données utilisées pour les tests

Voici la liste des flux RSS avec lesquels l'extraction de données a été testée avec succès pendant le développement :

- <https://www.france24.com/fr/rss>
- <https://www.lemonde.fr/rss/une.xml>
- <https://news.google.com/rss?hl=fr&gl=FR&zceid=FR:fr>
- <https://www.ouest-france.fr/rss/france>
- <https://www.francetvinfo.fr/monde.rss>
- <https://www.20minutes.fr/feeds/rss-monde.xml>
- <https://rmc.bfmtv.com/rss/actualites/>

- <https://feeds.leparisien.fr/leparisien/rss>
- <https://www.ladepeche.fr/rss.xml>

En plus des sources ci-dessus, le script a été testé avec le flux RSS du journal l'Équipe. Les résultats n'ont pas été concluants (Seul les premières phrases de chaque article sont détectées).

Plusieurs flux RSS de sources en anglais ont aussi été testés, avec succès :

- <https://www.independent.co.uk/news/world/rss>
- <https://rss.app/feeds/D0SUH2NYYoOzypYN.xml>
- <https://www.huffpost.com/section/world-news/feed>
- <https://news.google.com/rss>

## Requêtes SQL

- Récupération de la liste des sources, déclenchée par l'endpoint /feeds :

```
SELECT * FROM rss_feeds;
```

- Actualisation des articles enregistrés dans la base de données :

```
- SELECT id FROM daily_articles;
- INSERT INTO daily_articles
  (title, url, content, rss_source_id, authors, date, image)
VALUES('titre', 'url', 'texte', 'id_rss', 'auteurs', 'date', 'lien_image');
- DELETE FROM daily_articles WHERE id IN ('anciens_id');
```

## Captures d'écran

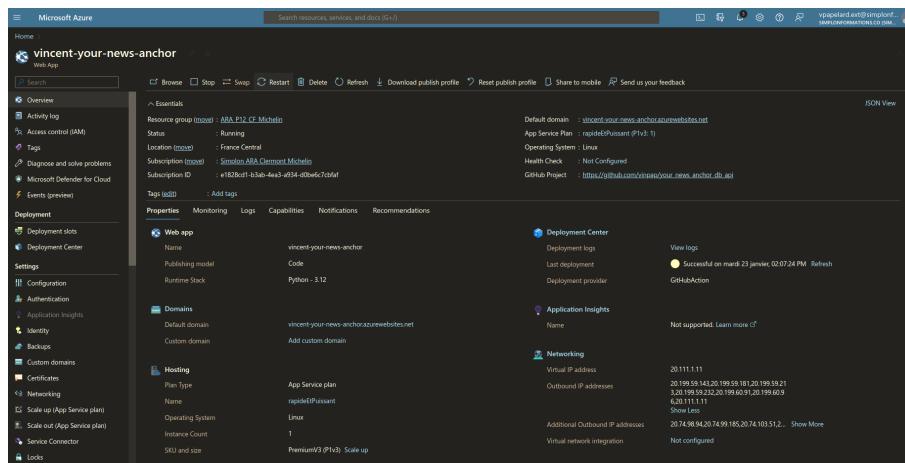


Figure 5: Dashboard de notre API sur Azure

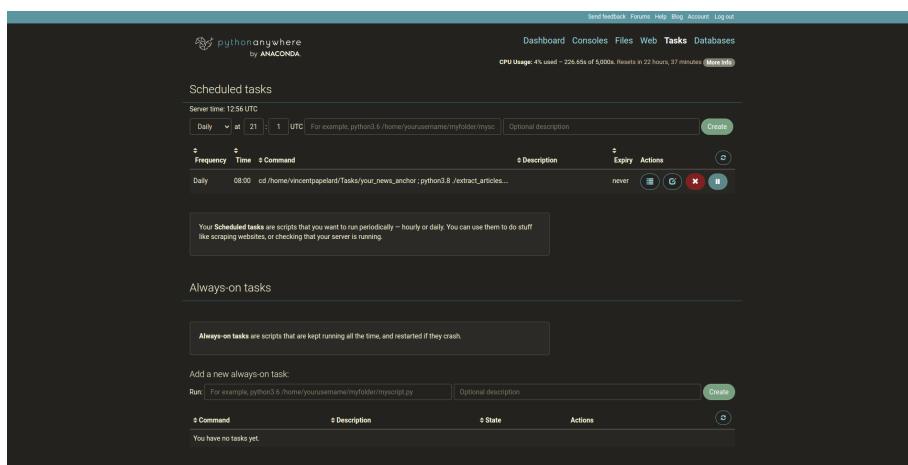


Figure 6: Dashboard de PythonAnywhere. Cette page nous permet de planifier l'exécution de scripts à intervalles réguliers, ici toutes les 24 heures