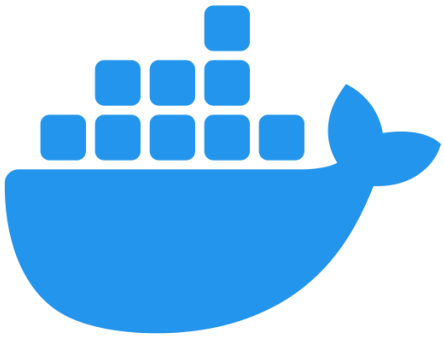
TECHNOmark®
smart traceability

Création et automatisation d'une veille concurrentielle



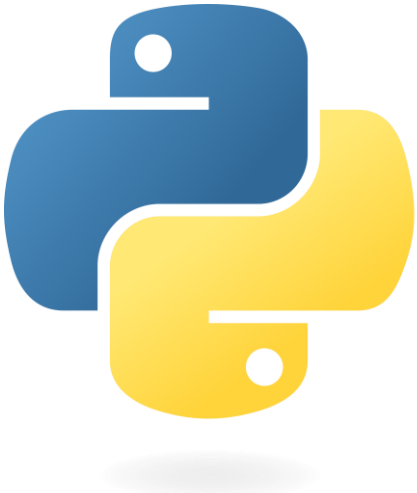


Table des matières

1. Expression des besoins

1.1 Contexte, domaine, processus métier

Dans le cadre de mon BTS SIO, j'ai réalisé mon stage de seconde année à Technomark, une société experte en machines intelligentes de traçabilité et de marquage industrielle. Mon projet principal consiste à créer et automatiser une veille concurrentielle pour l'entreprise.

1.2 Demandeur, acteurs, utilisateurs

1.2.a Demandeur

Le demandeur de ce projet de veille concurrentielle automatisée est Technomark, plus précisément mon tuteur de stage, le responsable SI de l'entreprise. L'objectif est d'automatiser la surveillance du marché et des concurrents afin d'optimiser la prise de décision stratégique.

1.2.b Acteurs

Les principaux acteurs impliqués dans ce projet sont :

- Le tuteur de stage : Responsable de mon encadrement et garant du bon déroulement du projet.
- Le service marketing : Principal bénéficiaire des informations collectées, chargé d'analyser les données issues de la veille.
- Moi-même : Je suis responsable de créer cette veille du début à la fin avec l'aide de mon tuteur.

1.2.c Utilisateurs

Les principaux utilisateurs de cette veille concurrentielle sont l'équipe marketing et l'équipe dirigeante, qui exploitent les données pour ajuster la stratégie commerciale et suivre l'évolution du marché.

1.3 Étude de l'existant, diagnostic

Avant la mise en place de cette veille concurrentielle automatisée, la surveillance du marché était réalisée manuellement par l'équipe marketing à l'aide de recherches sur internet. Cette méthode manuelle posait plusieurs problèmes tels que le temps de mise en place, des probables oublis et ne permettait pas un suivi en temps réel.

1.4 Description de la demande, objectifs, bénéfices attendus

L'objectif de ce projet est de créer un outil automatisé de veille concurrentielle pour collecter et analyser les données issues de différents sites web.

1.4.a Objectifs

- Automatiser la collecte pour réduire le temps de recherche.
- Améliorer la fiabilité des données et limiter les erreurs.
- Assurer un suivi en temps réel avec des mises à jour automatiques.
- Structurer les données pour une analyse plus efficace.

1.4.b Bénéfices attendus

- Gain de temps : Moins de recherches manuelles, plus d'analyse.
- Réactivité : Suivi en temps réel du marché.
- Centralisation : Toutes les données accessibles en un seul endroit.
- Meilleure prise de décision : Données précises et exploitables.

Ce projet vise à rendre la veille concurrentielle plus rapide, fiable et stratégique pour l'entreprise.

1.5 Spécifications fonctionnelles

La solution doit répondre aux besoins des utilisateurs en automatisant la veille concurrentielle. Voici donc les fonctionnalités :

- Récupération des données : En tant qu'utilisateur, je veux que la solution récupère automatiquement les informations concurrentielles.
- Automatisation de la collecte : En tant qu'utilisateur, je veux que les données soient récupérées lors d'une intervention manuelle.
- Affichage des données : En tant qu'utilisateur, je veux consulter les données sur une page web qui sera accessible à toute l'entreprise.
- Gestion des données : En tant qu'administrateur, je veux pouvoir ajouter ou supprimer des articles.
- Droits d'accès : En tant qu'utilisateur ou administrateur, je veux accéder à des fonctionnalités différentes selon mon profil.

1.6 Contraintes ou exigences

Le projet doit respecter plusieurs contraintes :

- Technique : Utilisation de Docker et du langage Python avec des API REST (telles que l'API Mistral AI).

- Matérielles : Utilisation d'un ordinateur présent sur place, comprenant l'abonnement Microsoft 365 (pour un accès à Copilot, avec une adresse e-mail fournie l'entreprise pour la durée de mon stage).
- Délais : Le projet doit être terminé avant la fin de mon stage qui dure 6 semaines.
- Budget : Utilisation de logiciels et d'API gratuites (avec limites d'utilisation).

2. Conception, Spécifications Techniques

2.1 Description de la solution

La solution développée vise à automatiser la veille concurrentielle en collectant, traitant et affichant les données de manière structurée. Elle repose sur un environnement conteneurisé sous Docker.

1. Collecte des données : Récupération des informations pertinentes publiées sur diverses sources via des flux RSS.
2. Automatisation du processus : Programmation d'un script Python permettant une extraction des données avec un clique sur un bouton de lancement.
3. Affichage des résultats : Intégration des données collectées dans une page web HTML avec Flask pour une consultation facile.

L'objectif est d'obtenir un système fiable, automatisé et accessible pour suivre efficacement la concurrence afin de garantir un gain de temps, une centralisation des données et une meilleure réactivité.

2.2 Outils logiciels de la solution

Pour développer et déployer la solution de veille concurrentielle, plusieurs outils et technologies ont été utilisés :

- Python : Langage principal utilisé pour l'automatisation de la collecte et du traitement des données.
- Flask : Framework web léger en Python permettant de créer une interface web et de gérer les routes de l'application.
- BeautifulSoup et newspaper3k : Bibliothèques Python servant à analyser le code HTML des pages web et à extraire automatiquement le contenu des articles.
- API REST : Interfaces permettant d'échanger des données avec des services externes via des requêtes HTTP.

- Postman : Logiciel utilisé pour simuler et tester les requêtes envoyées aux API afin de s'assurer de leur bon fonctionnement avant intégration.
- Docker : Technologie de conteneurisation permettant de déployer l'application de manière isolée et reproductible.
- HTML : Langage de balisage qui permet de structurer les éléments visuels qui composent une page web.
- Visual Studio Code : Environnement de développement utilisé pour écrire et tester le code.

Ces outils assurent une solution performante, sécurisée et facilement déployable.

2.3 Architecture logicielle de la solution

La solution est structurée en plusieurs étapes :

1. Collecte des données
   - Extraction d'articles via des Flux RSS de Google News et DirectIndustry.
2. Traitement et analyse des données
   - Utilisation de bibliothèques Python telles que BeautifulSoup et newspaper3k pour récupérer le contenu des articles extraits.
   - Envoi des données collectées à l'API REST de Mistral AI pour analyser et résumé les données.
3. Affichage des résultats
   - Génération d'une page HTML pour présenter les données traitées via Flask.
4. Sécurisation des clés API
   - La clé API utilisé est stockée dans un fichier .env et chargée de manière sécurisée via la bibliothèque python-dotenv.

2.4 Besoins techniques, ressources

Besoins techniques

- Langages et frameworks : Python, Flask, HTML
- Sources de données : API Mistral AI, Flux RSS, Bibliothèques de scraping
- Conteneurisation et déploiement : Docker
- Outils de développement : Visual Studio Code, Postman (tests API)

Ressources humaines

- Développeur (moi) : Développement, intégration et automatisation du système.
- Tuteur : Suivi du projet et validation des choix techniques.

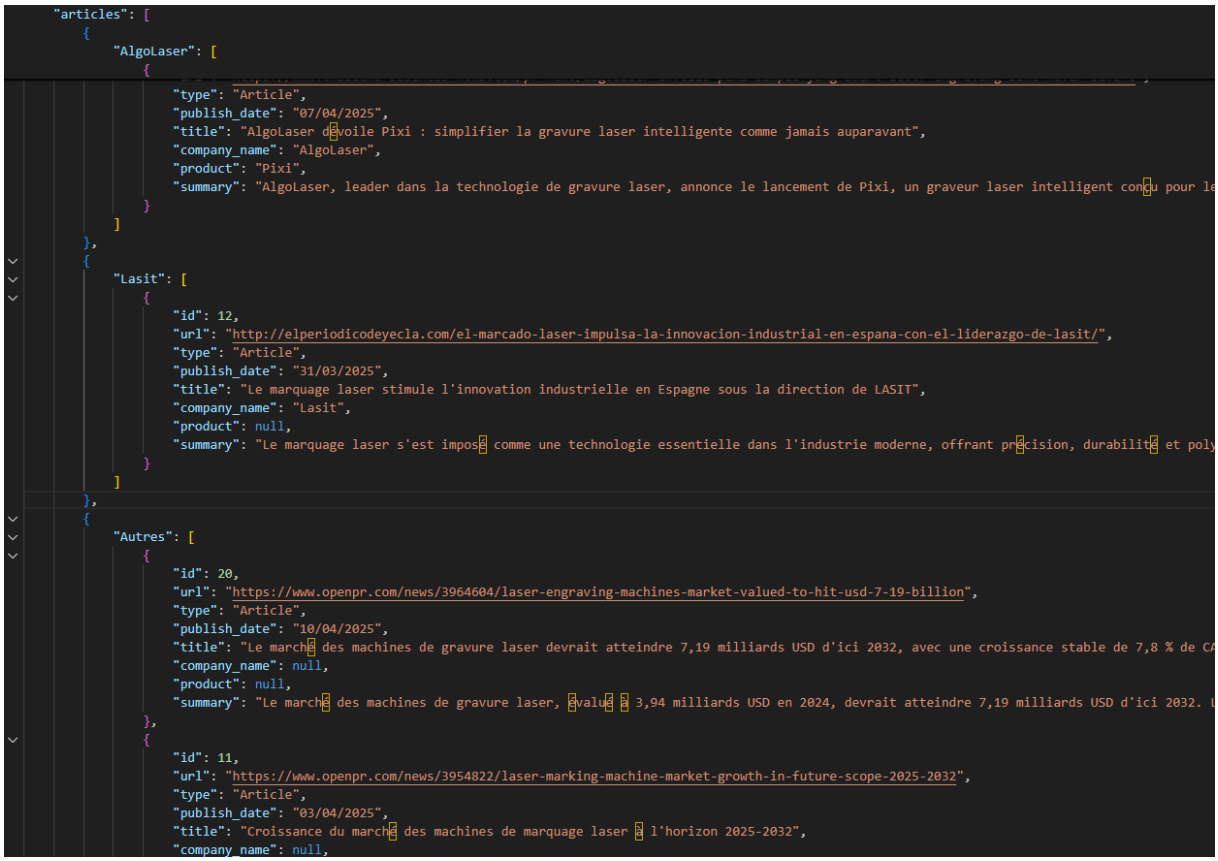
- Service marketing : Vérification des articles retenues et conseils pour améliorer l'extraction.

Ressources matérielles

- Poste de travail : PC de l'entreprise.
- Serveur Flask : Serveur pour l'hébergement de l'application web Flask.

2.5 Analyse des données

Ce projet ne repose pas sur une base de données, mais sur la collecte de données JSON via des APIs et des flux RSS. Les informations collectées incluent des articles RSS, des pages web et des tweets. Voici un extrait ci-dessous :



2.6 IHM (interfaces homme-machine), Maquettage

L'interface utilisateur a été pensée pour rester simple, claire et cohérente avec l'identité visuelle de l'entreprise. La charte graphique utilisée s'inspire directement du site web de Technomark, afin de garantir une continuité visuelle et une intégration naturelle avec les outils internes.

L'interface est générée avec Flask en HTML/CSS et JavaScript.

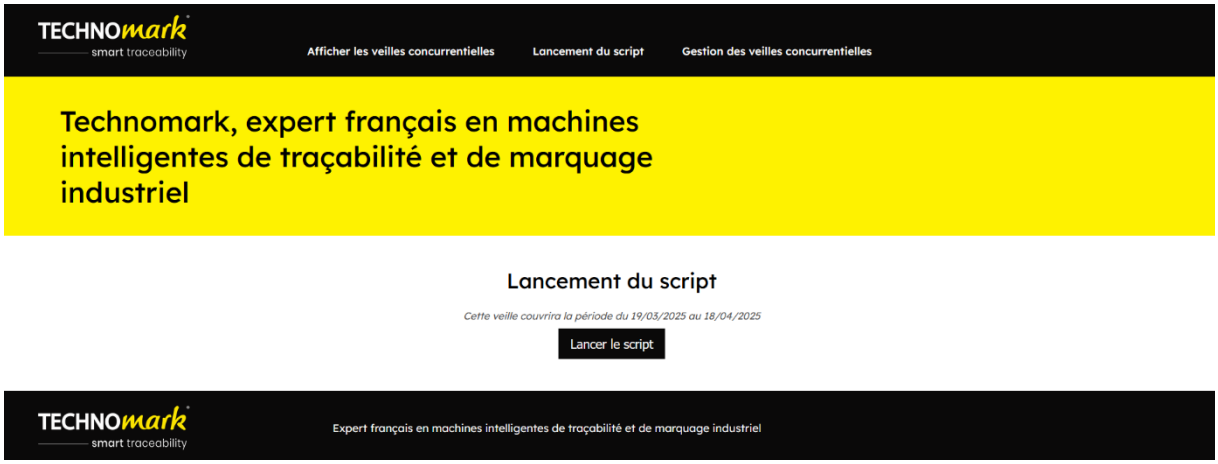
L’application permet :

- Un affichage des pages HTML générées selon le mois sélectionné (disponible en mode admin et utilisateur).

- La gestion des veilles (en mode admin seulement).

- D’afficher les articles collectés et résumés et de les gérer (en mode admin seulement).

- De lancer la récupération des articles via un bouton (en mode admin seulement).



2.7 Conduite de projet : décomposition en tâches, structure équipes, durée

Le projet a été divisé en plusieurs étapes :

1. Mise en place de l'environnement : Configuration de Docker.
2. Collecte des données : Récupération des informations via Flux RSS.
3. Traitement des données : Analyse et structuration avec l'API Mistral AI.

4. Droits d'accès : Gestion d'un mode utilisateur accessible à tout le monde et un mode admin pour pouvoir gérer les veilles, les articles et lancer le script de récupération.
5. Affichage des résultats : Développement de la page HTML via Flask pour visualiser les données.
6. Déploiement et présentation : Mise en place finale et présentation du projet.

Le projet a été réalisé en autonomie dans le cadre du stage, avec un suivi régulier par le tuteur pour valider les avancées et ajuster si nécessaire. Il se déroule sur 6 semaines, avec une progression continue et des validations fréquentes.

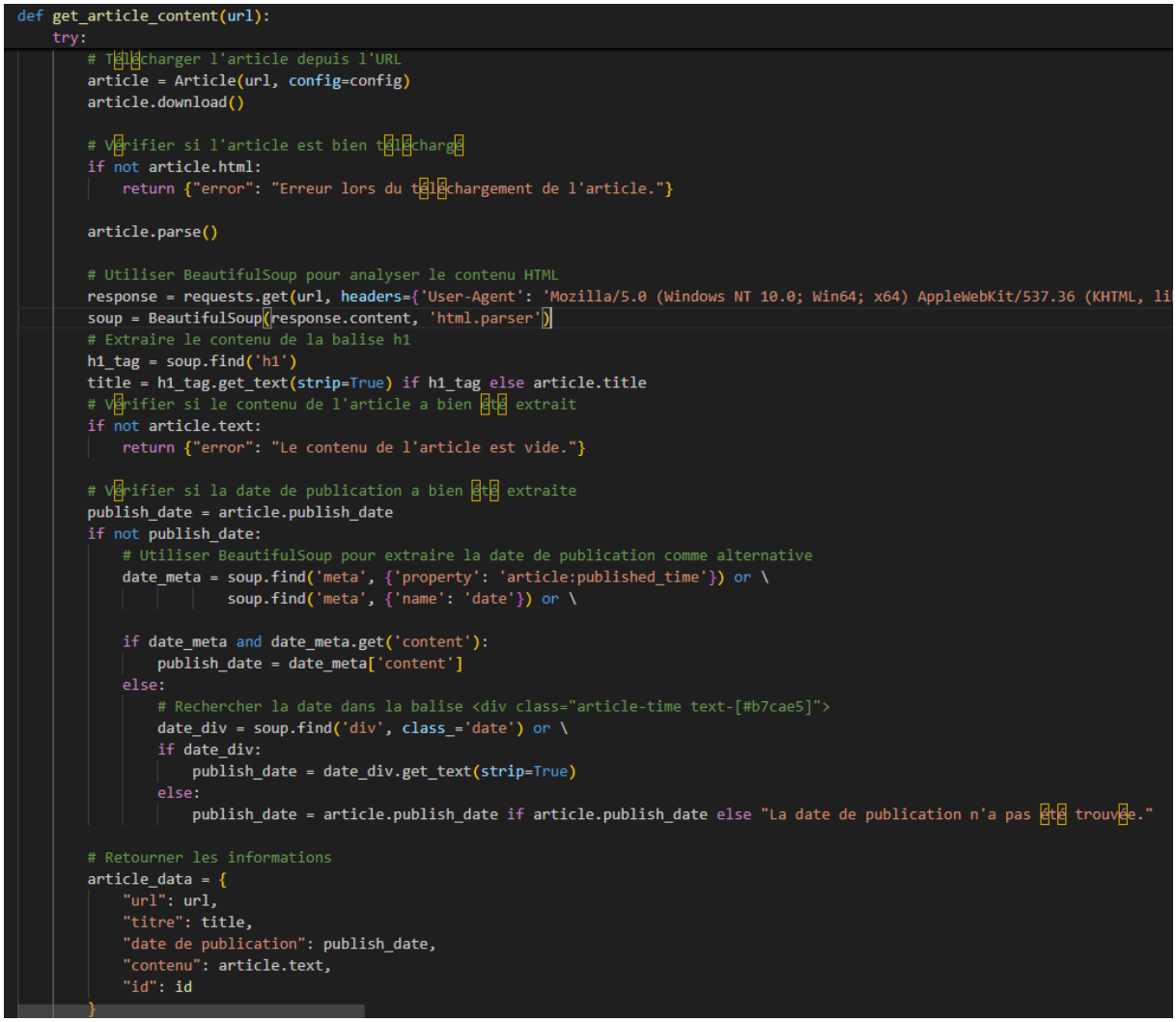
3. Développement

3.1 Réalisation des interfaces et programmes conformes aux spécifications fonctionnelles attendues

- Fonction de récupération des articles via des Flux RSS : Cet extrait montre le script Python utilisé pour récupérer les articles via un flux RSS. Il exploite la bibliothèque Feedparser pour extraire les données et les formater de manière exploitable.

```python
# Fonction pour récupérer et stocke les articles des flux RSS Google News
def fetch_rss_articles(rss_feed_urls, keywords, avoid_words):
    articles = []
    seen_titles = set() # Optimisation des listes
    seen_links = set() # Optimisation des listes

    for url in rss_feed_urls:
        feed = feedparser.parse(url)

        for entry in feed.entries:
            title = entry.title if 'title' in entry else ''
            summary = entry.summary if 'summary' in entry else ''

            # Vérifie si les mots-clés les mots à éviter son présents
            contains_keyword = any(keyword.lower() in (title + summary).lower() for keyword in keywords)
            contains_avoid_word = any(avoid_word.lower() in (title + summary).lower() for avoid_word in avoid_words)
            if contains_keyword and not contains_avoid_word:
                # Vérifie si le titre ou le lien n'ont pas déjà été vus
                if title not in seen_titles and entry.link not in seen_links:
                    # Crée un dictionnaire avec le titre et le lien
                    article = {
                        'title': title,
                        'link': entry.link
                    }
                    articles.append(article)
                    seen_titles.add(title)    # Ajouter le titre à la liste des titres vus
                    seen_links.add(entry.link)  # Ajouter le lien à la liste des liens vus

    return articles

# Récupérer et stocker les articles
articles = fetch_rss_articles(rss_feed_urls, keywords, avoid_words)
```

- Fonction d'extraction des données des articles :
  Cette capture d'écran montre la façon dont sont extraites les données telles que les dates de publications, les titres et le contenu de l'article via les bibliothèques Beautifulsoup et newspaper3k (du contenu a été masqué pour simplifier la lecture).



```python
def get_article_content(url):
    try:
        # Télécharger l'article depuis l'URL
        article = Article(url, config=config)
        article.download()

        # Vérifier si l'article est bien téléchargé
        if not article.html:
            return {"error": "Erreur lors du téléchargement de l'article."}

        article.parse()

        # Utiliser BeautifulSoup pour analyser le contenu HTML
        response = requests.get(url, headers={'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, li
        soup = BeautifulSoup(response.content, 'html.parser')
        # Extraire le contenu de la balise h1
        h1_tag = soup.find('h1')
        title = h1_tag.get_text(strip=True) if h1_tag else article.title
        # Vérifier si le contenu de l'article a bien été extrait
        if not article.text:
            return {"error": "Le contenu de l'article est vide."}

        # Vérifier si la date de publication a bien été extraite
        publish_date = article.publish_date
        if not publish_date:
            # Utiliser BeautifulSoup pour extraire la date de publication comme alternative
            date_meta = soup.find('meta', {'property': 'article:published_time'}) or \
                        soup.find('meta', {'name': 'date'}) or \

            if date_meta and date_meta.get('content'):
                publish_date = date_meta['content']
            else:
                # Rechercher la date dans la balise <div class="article-time text-[#b7cae5]">
                date_div = soup.find('div', class_='date') or \
                if date_div:
                    publish_date = date_div.get_text(strip=True)
                else:
                    publish_date = article.publish_date if article.publish_date else "La date de publication n'a pas été trouvée."

        # Retourner les informations
        article_data = {
            "url": url,
            "titre": title,
            "date de publication": publish_date,
            "contenu": article.text,
            "id": id
        }
```

- Fonction d'analyse des données avec l'API Mistral AI :
  Le morceau de code ci-dessous montre l'envoi des articles collectés à l'API Mistral AI pour analyse et résumé automatique sous forme d'une simple requête textuelle. L'API répond ensuite sous forme de format JSON comprenant un ID, l'URL de la page, le type de texte (article ou fiche produit), date de publication, le titre de la page, la société dont parle la page s'il y en a, et le résumé généré par l'API. Par la suite, un autre appel à l'API de Mistral AI similaire à celui-ci est effectué, mais permettant d'analyser le fichier JSON (créer après le premier appel à l'API) afin de garder que les articles qui parlent des sujets nécessaires pour la veille concurrentielle et de regrouper les objets JSON similaires.

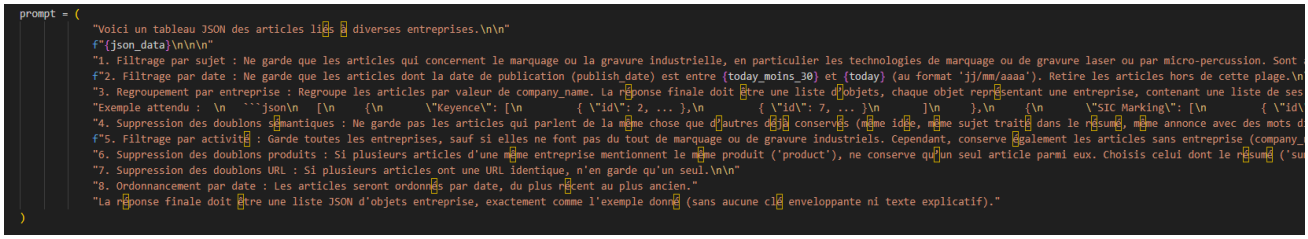
```python
# Fonction pour résumer un texte
def summarize_text(text_to_resume):
    headers = {
        "Authorization": f"Bearer {api_key}",
        "Content-Type": "application/json",
        "X-Request-ID": "request_1"
    }
    data = {
        "model": "mistral-large-latest",
        "messages": [
            {
                "role": "system",
                "content": "Tu es un modèle d'intelligence artificielle spécialisé dans l'analyse d'articles et de fiche
            },
            {
                "role": "user",
                "content": f"J'aimerais que tu renvois ta réponse dans un format JSON avec ces attributs: id ({id}), ur
            }
        ],
        "temperature": 0.7
    }
    response = requests.post(f"{api_mistral_url}/chat/completions", headers=headers, json=data)

    if response.status_code == 200:
        api_response = response.json()['choices'][0]['message']['content']
        # Nettoyage et décodage de la réponse JSON
        try:
            # Vérifier si la réponse est dans un format JSON ou si elle contient un tableau avec une chaîne JSON
            if isinstance(api_response, str) and api_response.startswith('```json'):
                # Extraire la chaîne JSON (enlever les délimiteurs)
                json_string = api_response.strip("```json\n").strip("\n```")
                parsed_response = json.loads(json_string)
            else:
                # Si la réponse est déjà une chaîne JSON valide
                parsed_response = json.loads(api_response)

            # Retourner la réponse sous forme de dictionnaire JSON (pas sous forme de chaîne)
            return parsed_response  # Retourne directement un objet JSON Python

        except json.JSONDecodeError:
            # Inclure la réponse brute dans l'erreur pour mieux comprendre ce qui a échoué
            return {
                "error": "La réponse de l'API ne contient pas un format JSON valide.",
                "raw_response": f"{response.text}"  # Ajouter la réponse brute de l'API
            }
    else:
```

- Code Flask pour l'affichage des articles dans une page HTML générée : À chaque exécution du script de récupération d'articles, la fonction ci-dessous est appelée afin de créer une page HTML automatiquement à partir de la page articles_templates.html où sont affichés les résultats finaux du fichier JSON.

```python
# Fonction de génération de page HTML via la page articles_template.html.
def generate_html(articles, output_file, date_deb=None, date_fin=None):
    rendered = render_template(
        'articles_template.html',
        articles=articles,
        date_deb=date_deb,
        date_fin=date_fin
    )
    with open(output_file, 'w', encoding='utf-8') as f:
        f.write(rendered)

        # Générer le nom du fichier JSON
    json_file = output_file.replace(".html", ".json")

    # Préparer les données JSON
    json_data = {
        "articles": articles,
        "start_date": date_deb,
        "end_date": date_fin
    }

    # Sauvegarder le fichier JSON
    with open(json_file, 'w', encoding='utf-8') as f:
        json.dump(json_data, f, ensure_ascii=False, indent=4)
```

```jinja
{% for group in articles %}
    {% for company, company_articles in group.items() %}
        {% for article in company_articles %}
            <div class="article">
                <h2>{{ article.title or 'Titre inconnu' }}</h2>
                <p class="company">{{ article.company_name or 'Autres' }}</p>
                <p class="date">{{ article.publish_date or 'Date inconnue' }}</p>
                <p class="type">{{ article.type or 'Type inconnu' }}</p>
                <p class="summary">{{ article.summary or 'Résumé inconnu' }}</p>
                <a href="{{ article.url or '#' }}" target="_blank">Lire l'article</a
            </div>
        {% endfor %}
    {% endfor %}
{% endfor %}
```

- Fonction de triage des articles en fonctions de critères avec l'API Mistral AI : Cette fonction permet le triage des articles obtenu grâce aux critères ci-dessous. La réponse de l'API sera renvoyée en format JSON.



```
prompt = (
    "Voici un tableau JSON des articles liés à diverses entreprises.\n\n"
    f"{json_data}\n\n\n"
    "1. Filtrage par sujet : Ne garde que les articles qui concernent le marquage ou la gravure industrielle, en particulier les technologies de marquage ou de gravure laser ou par micro-percussion. Sont
    f"2. Filtrage par date : Ne garde que les articles dont la date de publication (publish_date) est entre {today_moins_30} et {today} (au format 'jj/mm/aaaa'). Retire les articles hors de cette plage.\n
    "3. Regroupement par entreprise : Regroupe les articles par valeur de company_name. La réponse finale doit être une liste d'objets, chaque objet représentant une entreprise, contenant une liste de ses
    "Exemple attendu :  \n  ```json\n  [\n    {\n      \"Keyence\": [\n        { \"id\": 2, ... },\n        { \"id\": 7, ... }\n      ]\n    },\n    {\n      \"SIC Marking\": [\n        { \"id\
    "4. Suppression des doublons sémantiques : Ne garde pas les articles qui parlent de la même chose que d'autres déjà conservés (même idée, même sujet traité dans le résumé, même annonce avec des mots d
    f"5. Filtrage par activité : Garde toutes les entreprises, sauf si elles ne font pas du tout de marquage ou de gravure industriels. Cependant, conserve également les articles sans entreprise (company_
    "6. Suppression des doublons produits : Si plusieurs articles d'une même entreprise mentionnent le même produit ('product'), ne conserve qu'un seul article parmi eux. Choisis celui dont le résumé ('su
    "7. Suppression des doublons URL : Si plusieurs articles ont une URL identique, n'en garde qu'un seul.\n\n"
    "8. Ordonnancement par date : Les articles seront ordonnés par date, du plus récent au plus ancien."
    "La réponse finale doit être une liste JSON d'objets entreprise, exactement comme l'exemple donné (sans aucune clé enveloppante ni texte explicatif)."
)
```

```python
def group_and_order(file_perm):
    data = {
    }

    response = requests.post(f"{api_mistral_url}/chat/completions", headers=headers, json=data)

    if response.status_code == 200:
        json_final = response.json()['choices'][0]['message']['content']

        # Nettoyage et décodage de la réponse JSON
        try:
            if isinstance(json_final, str):
                # Vérifier si la réponse commence par les délimiteurs JSON
                if json_final.startswith('```json'):
                    # Enlever les délimiteurs de début et de fin
                    json_string = json_final[len('```json\n'):].strip('\n```')
                else:
                    json_string = json_final

                # Essayer de parser la chaîne nettoyée en JSON
                try:
                    parsed_response = json.loads(json_string)
                except json.JSONDecodeError as e:
                    return {
                        "error": "Erreur\n",
                        "raw_response": json_final,
                        "\n\nerror_details": str(e)
                    }
            else:
                # Si la réponse n'est pas une chaîne, essayer de la parser directement
                parsed_response = json.loads(json_final)

            # Retourner la réponse sous forme de dictionnaire JSON (pas sous forme de chaîne)
            return parsed_response  # Retourne directement un objet JSON Python

        except json.JSONDecodeError as e:
            # Inclure la réponse brute dans l'erreur pour mieux comprendre ce qui a échoué
            return {
                "error": "La réponse de l'API ne contient pas un format JSON valide.",
                "raw_response": f"{response.text}",
                "error_details": str(e)
            }
```

- Codes de lancement du script de récupération d'articles : Sur une page HTML dédiée au lancement du script, les administrateurs peuvent lancer ou arrêter l'exécution du script (l'arrêt se fait via une autre route Flask).

```python
# Route pour lancer le script
@app.route('/run-script', methods=['POST'])
def run_script():
    print("La fonction runScript est bien appelée !")

    try:
        # Ouvrir le fichier log pour capturer stdout et stderr
        with open("log.txt", "a") as log_file:
            # Lance main.py en arrière-plan avec redirection des sorties
            process = subprocess.Popen(
                ["python", "main.py"],
                stdout=log_file,
                stderr=log_file
            )

            # Sauvegarder le PID dans un fichier
            with open("script.pid", "w") as pid_file:
                pid_file.write(str(process.pid))

        return jsonify({"message": f"Script lancé avec PID {process.pid}"}), 200

    except Exception as e:
        return jsonify({"error": str(e)}), 500
```

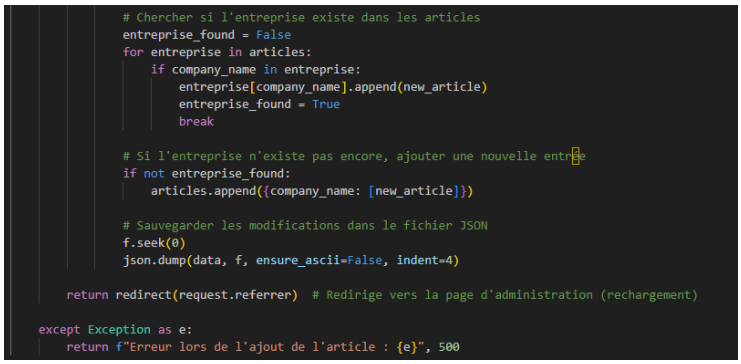
- Codes de gestion des articles : Ci-dessous, les codes permettant d'ajouter ou de supprimer des articles via une page HTML et une route Flask.

```python
@app.route('/supprimer-article', methods=['POST'])
def supprimer_article():
    article_id = request.form.get('article_id')  # Récupérer l'id de l'article
    filename = request.form.get('filename')  # Récupérer le nom du fichier (si besoin)

    if not article_id or not filename:
        return "Erreur : Données manquantes", 400

    json_filename = filename.replace('.html', '.json')
    json_path = os.path.join('veilles', json_filename)

    try:
        # Ouvrir le fichier JSON pour récupérer les articles
        with open(json_path, 'r', encoding='utf-8') as f:
            data = json.load(f)

        # Trouver l'article à supprimer en fonction de l'id
        article_to_remove = None
        for group in data["articles"]:
            for company, articles in group.items():
                for article in articles:
                    if str(article["id"]) == article_id:  # On trouve l'article avec l'id
                        article_to_remove = article
                        break
                if article_to_remove:
                    break
            if article_to_remove:
                break

        if article_to_remove:
            # Supprimer l'article
            for group in data["articles"]:
                for company, articles in group.items():
                    if article_to_remove in articles:
                        articles.remove(article_to_remove)

            # Sauvegarder les modifications dans le fichier JSON
            with open(json_path, 'w', encoding='utf-8') as f:
                json.dump(data, f, ensure_ascii=False, indent=4)

            return redirect(request.referrer)  # Redirige vers la même page

        return "Article non trouvé", 404

    except Exception as e:
        return f"Erreur : {e}", 500
```

Suppression d'article

```python
@app.route('/gestion/ajouter-article', methods=['POST'])
def ajouter_article():
    # Récupérer les données du formulaire
    title = request.form.get('title')
    company_name = request.form.get('company_name')
    publish_date = request.form.get('publish_date')
    article_type = request.form.get('type')
    summary = request.form.get('summary')
    url = request.form.get('url')
    filename = request.form.get('filename')  # Assure-toi que le filename est bien récupéré

    if not all([title, company_name, publish_date, article_type, summary, url]):
        return "Erreur : Tous les champs sont obligatoires.", 400

    # Créer un nouvel article avec un id automatique
    new_article = {
        "id": int(time.time()),  # Utilisation du timestamp comme ID unique
        "url": url,
        "type": article_type,
        "publish_date": publish_date,
        "title": title,
        "company_name": company_name,
        "product": None,  # A ajouter si nécessaire
        "summary": summary
    }
    # Construire le chemin du fichier JSON à partir du nom de la page HTML
    json_filename = filename.replace('.html', '.json')
    json_path = os.path.join('veilles', json_filename)

    # Vérifier si le fichier existe
    if not os.path.exists(json_path):
        return "Erreur : Le fichier JSON n'existe pas.", 400

    try:
        # Ouvrir le fichier JSON en mode lecture/écriture
        with open(json_path, 'r+', encoding='utf-8') as f:
            # Charger les données existantes
            data = json.load(f)

            # Si la structure est correcte, ajouter l'article à la bonne entreprise
            if isinstance(data, dict) and "articles" in data:
                articles = data["articles"]
```

Ajout d'article (1)



Ajout d'article (2)

3.2 Dossier de programmation codes sources documentés et commentés

L'ensemble du code a été structuré en modules distincts pour assurer une bonne lisibilité :

- Un fichier dédié au serveur Flask.
- Un script séparé pour la collecte et le traitement des données.
- Utilisation de fichiers .env pour sécuriser les clés API.
- Commentaires ajoutés dans toutes les parties importantes du code pour assurer la compréhension du fonctionnement.

Ces choix facilitent la maintenance du projet et sa compréhension par un autre développeur si nécessaire.

3.3 Difficultés rencontrées

1) Lors du développement de l'application, j'ai réalisé que la fonctionnalité que je souhaitais réaliser avec l'API Microsoft Graph qui est de résumer le texte d'un article ou d'un tweet avec Copilot est impossible car l'API Microsoft Graph ne comprend pas d'accès à Copilot, j'ai donc changé d'API et j'ai commencé à implanter l'API Mistral IA (IA française avec un accès gratuit avec des limites d'utilisation).

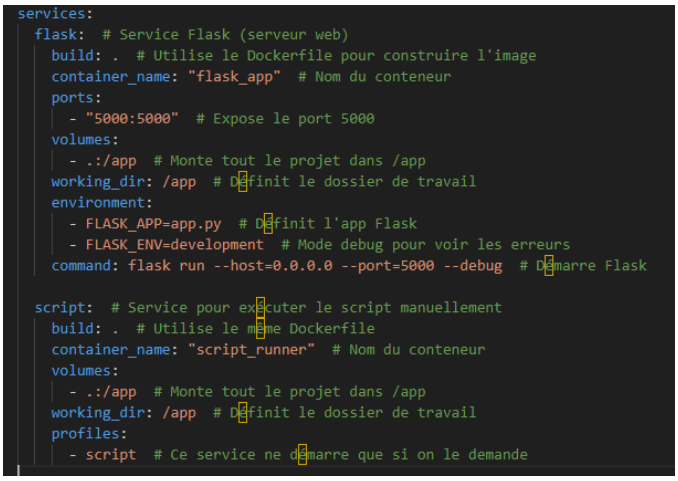
2) L'une des difficultés principales rencontrées a été la mise en fonctionnement du serveur local Flask. Malgré plusieurs tentatives de changement de port dans le code, le serveur ne se lançait toujours pas correctement. Pour résoudre ce problème, j'ai re créé une image Docker, en configurant le port 5000 directement dans le Dockerfile. De plus, à chaque lancement du script, j'ai adapté ma commande Docker pour forcer le lancement du script sur le port 5000.

3) Lors de ma quatrième semaine de stage, j'ai voulu rendre accessible le serveur Flask en permanence et pouvoir exécuter le script de récupération des articles directement depuis l'interface web.

Pour cela, j'ai séparé mon projet en deux images Docker :

- Une dédiée au serveur Flask, qui affiche les articles traités en continu,
- L'autre pour le script de récupération, exécuté uniquement à la demande.

J'ai ensuite mis en place un fichier Docker nommé « docker-compose.yml » afin de spécifier le rôle des deux images :



```yaml
services:
  flask:  # Service Flask (serveur web)
    build: .  # Utilise le Dockerfile pour construire l'image
    container_name: "flask_app"  # Nom du conteneur
    ports:
      - "5000:5000"  # Expose le port 5000
    volumes:
      - .:/app  # Monte tout le projet dans /app
    working_dir: /app  # Définit le dossier de travail
    environment:
      - FLASK_APP=app.py  # Définit l'app Flask
      - FLASK_ENV=development  # Mode debug pour voir les erreurs
    command: flask run --host=0.0.0.0 --port=5000 --debug  # Démarre Flask

  script:  # Service pour exécuter le script manuellement
    build: .  # Utilise le même Dockerfile
    container_name: "script_runner"  # Nom du conteneur
    volumes:
      - .:/app  # Monte tout le projet dans /app
    working_dir: /app  # Définit le dossier de travail
    profiles:
      - script  # Ce service ne démarre que si on le demande
```

Auparavant, je lançais l'ensemble du projet avec cette commande :

```
« docker run -it -p 5000:5000 --rm -v /run/desktop/mnt/host/c/Users/adilena/Documents/projet_veille_concurrentielle:/app veille_concurrentielle:v1.0 /bin/bash »
```

Désormais, le serveur Flask est lancé avec :

```
docker compose up -d
```

Ces changements permettent de séparer le lancement du serveur Flask du script de récupération, ce qui m'a permis par la suite de pouvoir lancer le script directement sur la page web du serveur Flask.

4) Au début, le but du projet était de récupérer les articles et les posts LinkedIn pour suivre les actualités des concurrents sur la micro-percussion et de marquage laser, mais lorsque j'ai procédé à l'ajout de récupération de posts LinkedIn, j'ai voulu en premier lieu utiliser l'API LinkedIn officielle mais ses prix étaient trop élevés. Ensuite avec mon tuteur, on a voulu utiliser une API LinkedIn non officielle, mais on était confronté au même problème. Comme dernier recours j'ai proposer une extraction de posts par Flux RSS LinkedIn via le site RSS.app, mais ayant déjà utiliser ce site lors

de récupération d'articles sur le site DirectInductry, l'essai gratuit prit fin et nous n'avons pu crée des Flux RSS LinkedIn. Nous avons donc décider de ne pas intégrer la récupération de posts LinkedIn au projet.

4. Exploitation, Mise en production

4.1 Test, déploiement, fonctionnement éventuel en double avec l'ancienne procédure.

Des tests ont été effectués tout au long du projet pour valider chaque étape du processus : collecte des données (Flux RSS, scraping), traitement avec Mistral AI, et affichage sur la page web. Ces tests ont permis de vérifier la fiabilité des données et le bon fonctionnement de chaque fonctionnalité au fur et à mesure de leur développement.

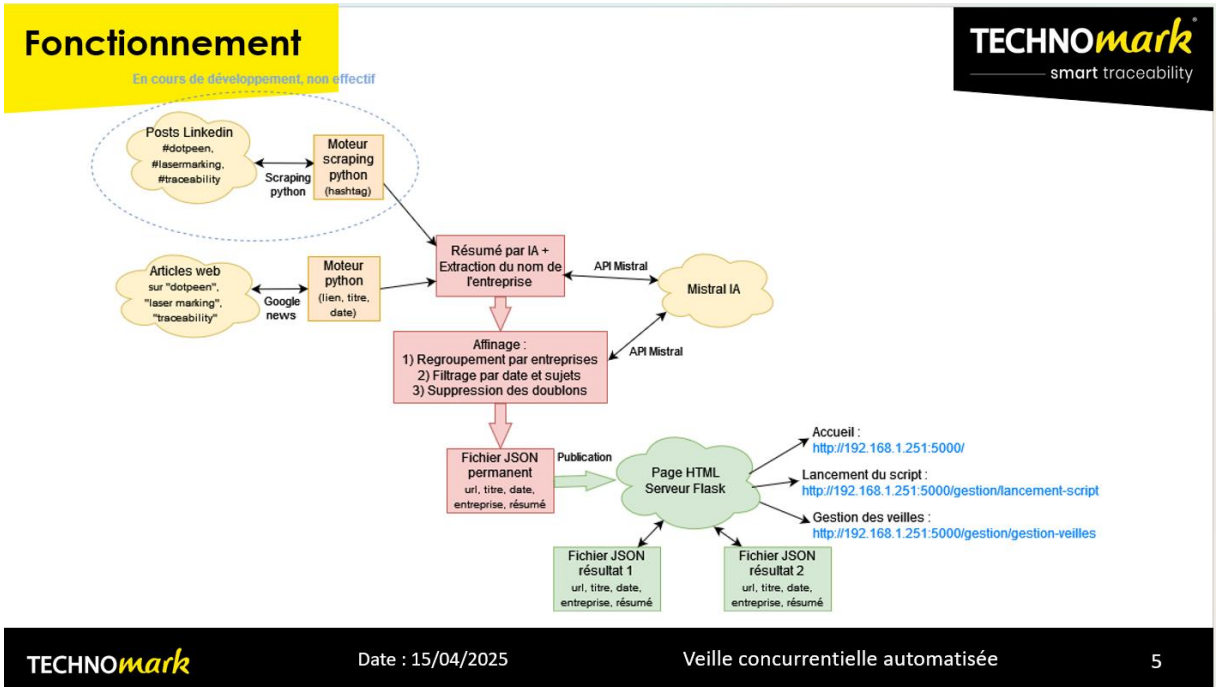
4.2 Rédaction notice utilisateur

Il n'y a pas de notice utilisateur formelle rédigée étant donné que le résultat du projet est un site web basique sans difficulté d'utilisation. Cependant, le code est commenté de manière détaillée pour expliquer son fonctionnement et permettre aux utilisateurs techniques de comprendre rapidement l'application.

4.3 Formation des utilisateurs

Plutôt qu'une formation formelle, j'ai réalisé une présentation à l'équipe marketing lors de ma dernière semaine de stage, afin de :

- Présenter les objectifs de ce projet et ce qu'il va apporter.
- Fournir une explication brève sur le fonctionnement du code (ci-dessous, la diapositive qui présente le fonctionnement).



- Présenter les différents usages de cette veille concurrentielle automatisée.
- Un aperçu du processus de collecte et de traitement des données.
- Une démonstration en direct de l'interface du site web.
- Une session de questions-réponses pour clarifier les points cités si nécessaire.

5. Bilan

Le projet qui m'a été confié pendant ces 6 semaines a été très enrichissant et formateur. J'ai appris à concevoir une solution complète, allant de la récupération automatisée de données à leur affichage dans une interface web via un serveur Flask sur une image Docker.

Ce stage m'a permis de mettre en pratique mes compétences en développement, notamment en Python et dans l'utilisation des API REST, mais aussi de découvrir des outils comme Docker, Flask et plusieurs bibliothèques python.

Enfin, il m'a permis travailler de manière autonome, avec un accompagnement régulier de mon tuteur pour valider les choix techniques et fonctionnels.

Je tiens à remercier chaleureusement toute l'équipe de Technomark pour leur bienveillance tout au long du stage. J'ai été ravie de faire partie de cette équipe et d'avoir pu évoluer dans un environnement aussi motivant.