Rapport d'Analyse et de Développement

Agent de Recherche d'Entreprise - Module Python

Présenté par : ANIBA Mouad
Institution : ISGA
Niveau : 2CI

1. Introduction

1.1 Contexte du projet

Le projet présenté dans ce rapport consiste en la création d'un agent de recherche d'entreprise développé en Python. Dans un contexte professionnel et académique où l'accès rapide à des informations fiables sur les entreprises est crucial, cet agent propose une solution automatisée pour collecter, traiter et présenter des données provenant de diverses sources.

1.2 Objectifs

Les objectifs principaux de ce projet sont les suivants :

- Développer un agent capable de collecter des informations pertinentes sur une entreprise donnée
- Intégrer plusieurs sources de données pour obtenir des informations complémentaires
- Proposer une interface utilisateur intuitive et fonctionnelle
- Gérer la persistance des recherches pour faciliter leur consultation ultérieure
- Optimiser les performances des requêtes via l'exécution parallèle

1.3 Méthodologie

La méthodologie adoptée pour ce projet s'appuie sur les principes de la programmation orientée objet, permettant une architecture modulaire et extensible. Le développement a été réalisé en Python, en exploitant diverses bibliothèques spécialisées pour l'extraction et le traitement des données. L'implémentation suit une approche par composants indépendants, communiquant entre eux via des interfaces bien définies.

2. Analyse architecturale

2.1 Structure générale

L'architecture du projet est composée de trois couches principales :

1. **Couche d'acquisition de données** : Représentée par les classes dérivées de DataSource
2. **Couche métier** : Implémentée par la classe CompanyResearchAgent
3. **Couche de présentation** : Gérée par la classe CompanyResearchUI

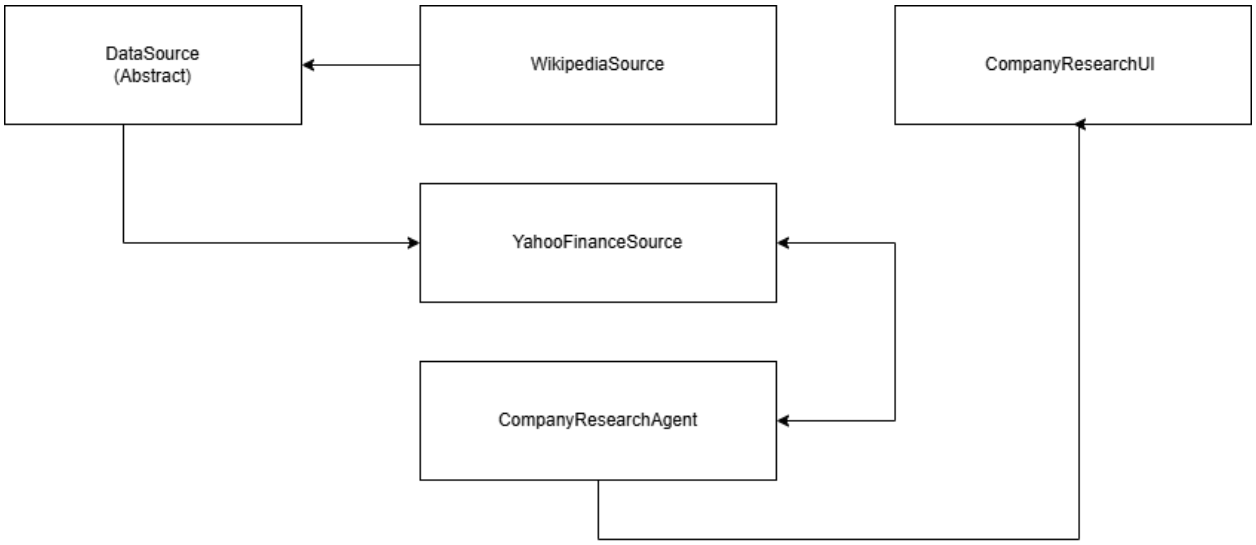
Cette séparation en couches assure une modularité permettant l'évolution indépendante de chaque composant et facilite la maintenance du code.

2.2 Modèle de conception

Le projet implémente plusieurs patrons de conception :

- **Pattern Strategy** : Utilisé pour les différentes sources de données qui partagent une interface commune mais possèdent des implémentations spécifiques.
- **Pattern Facade** : La classe CompanyResearchAgent sert de façade pour simplifier l'accès aux différentes sources de données.
- **Pattern Observer** : Mis en œuvre dans l'interface utilisateur pour réagir aux interactions de l'utilisateur.

2.3 Diagramme de classes



3. Composants principaux

3.1 Sources de données

Les sources de données sont implémentées selon le principe d'héritage à partir d'une classe abstraite DataSource. Chaque implémentation concrète fournit sa propre méthode get_info() pour récupérer des informations spécifiques :

1. **WikipediaSource** : Extrait des descriptions textuelles des entreprises à partir de Wikipedia.
    - Utilise l'API Wikipedia pour récupérer les résumés d'articles
    - Nettoie les données textuelles pour éliminer les balises et le formatage superflu
    - Fournit l'URL de l'article pour référence
2. **YahooFinanceSource** : Collecte des données financières et commerciales.
    - Détermine le symbole boursier de l'entreprise
    - Récupère des informations telles que le secteur d'activité, la capitalisation boursière et le nombre d'employés
    - Fournit des métadonnées comme le site web officiel

Ces classes encapsulent la logique spécifique à chaque source tout en présentant une interface uniforme à l'agent de recherche.

3.2 Agent de recherche

La classe CompanyResearchAgent représente le cœur du système en orchestrant les interactions entre les différentes sources de données et en gérant la persistance des recherches. Ses responsabilités principales sont :

- Initialiser et coordonner les différentes sources de données
- Exécuter les requêtes en parallèle pour optimiser les performances
- Consolider les résultats provenant des différentes sources
- Gérer l'historique des recherches via des opérations de sauvegarde et de chargement
- Mesurer le temps d'exécution des requêtes pour évaluer les performances

3.3 Interface utilisateur

L'interface utilisateur, implémentée par la classe CompanyResearchUI, offre une expérience interactive basée sur les widgets Jupyter. Elle présente les fonctionnalités suivantes :

- Formulaire de saisie pour le nom de l'entreprise
- Sélecteur de langue pour les résultats
- Affichage formaté des résultats de recherche
- Visualisation tabulaire de l'historique des recherches
- Fonctionnalités d'exportation des données au format CSV et JSON
- Navigation par onglets entre les résultats et l'historique

L'interface est conçue selon les principes de conception centrée sur l'utilisateur, avec une attention particulière portée à l'ergonomie et à la clarté de l'information.

4. Analyse technique

4.1 Gestion des requêtes parallèles

Un aspect technique notable du projet est l'utilisation de l'exécution parallèle pour optimiser les performances. Le module ThreadPoolExecutor de Python est employé pour exécuter simultanément les requêtes vers les différentes sources de données :

```
with ThreadPoolExecutor(max_workers=len(self.sources)) as executor:
    future_to_source = {
        executor.submit(source.get_info, company_name): name
        for name, source in self.sources.items()
    }

    for future in future_to_source:
        source_name = future_to_source[future]
        try:
            data = future.result()
            if data:
                results["sources"][source_name] = data
        except Exception as e:
            logger.error(f"Erreur pour la source {source_name}: {e}")
```

Cette approche permet de réduire significativement le temps total d'exécution lorsque plusieurs sources sont consultées, particulièrement pour celles nécessitant des requêtes HTTP avec des temps de latence importants.

4.2 Traitement des erreurs

Le système implémente une gestion robuste des erreurs à plusieurs niveaux :

1. **Niveau des sources de données** : Chaque méthode get_info() intègre des blocs try-except pour capturer les exceptions spécifiques à chaque API.
2. **Niveau de l'agent** : L'exécution parallèle est entourée d'une gestion d'erreurs pour éviter qu'une défaillance d'une source n'affecte les autres.
3. **Journalisation** : Un système de logging est implémenté pour enregistrer les erreurs et faciliter le débogage.

Cette approche multiniveau garantit la résilience du système face aux aléas des services externes et aux entrées utilisateur imprévues.

4.3 Persistance des données

La persistance des données est assurée par un mécanisme de sérialisation/dé sérialisation JSON :

- Les résultats de recherche sont sauvegardés dans un fichier search_history.json
- L'historique est chargé automatiquement au démarrage de l'agent
- Les données sont structurées pour faciliter leur exploitation ultérieure
- L'exportation au format CSV est également proposée pour l'interopérabilité

Cette approche permet de conserver les résultats entre les sessions et d'exploiter l'historique des recherches pour des analyses rétrospectives.

5. Évaluation des performances

5.1 Métriques de performance

Le système intègre une mesure automatique du temps d'exécution pour chaque recherche :

```
start_time = time.time()
# Exécution des requêtes
results["temps_execution"] = f"{time.time() - start_time:.2f} secondes"
```

Cette métrique permet d'évaluer l'efficacité des optimisations et de comparer les performances pour différentes entreprises ou configurations.

5.2 Points forts

Les principaux points forts du système sont :

1. Architecture modulaire : Facilité d'extension avec de nouvelles sources de données
2. Exécution parallèle : Optimisation des performances pour les requêtes multiples
3. Interface intuitive : Interface utilisateur claire et fonctionnelle
4. Multilinguisme : Support de plusieurs langues pour les recherches
5. Persistance des données : Conservation et exploitation de l'historique des recherches

5.3 Limitations

Malgré ses qualités, le système présente certaines limitations :

1. **Dépendance aux API externes** : Vulnérabilité face aux modifications ou indisponibilités des services tiers
2. **Couverture limitée** : Seules deux sources de données sont actuellement implémentées
3. **Absence de cache** : Les requêtes identiques sont réexécutées sans réutilisation des résultats précédents
4. **Interface limitée à Jupyter** : L'interface utilisateur est liée à l'environnement Jupyter Notebook

6. Améliorations potentielles

6.1 Expansion des sources de données

Pour enrichir les résultats, plusieurs sources supplémentaires pourraient être intégrées :

- **LinkedIn** : Pour des informations sur les employés et la culture d'entreprise
- **Bloomberg** : Pour des données financières plus détaillées
- **CrunchBase** : Pour des informations sur les start-ups et les investissements
- **Registre du commerce** : Pour des informations légales et administratives

6.2 Optimisation des performances

Plusieurs améliorations pourraient optimiser les performances :

- **Mise en cache** : Implémentation d'un système de cache pour éviter de répéter les requêtes identiques
- **Requêtes asynchrones** : Utilisation de asyncio pour une meilleure gestion des opérations d'E/S
- **Prioritisation des sources** : Définition d'un ordre de priorité pour afficher les résultats les plus rapides en premier

6.3 Fonctionnalités supplémentaires

Le système pourrait être enrichi par diverses fonctionnalités :

- **Analyse de sentiment** : Évaluation de la perception de l'entreprise dans les médias
- **Visualisation graphique** : Représentation visuelle des données financières
- **Extraction de relations** : Identification des liens entre entreprises (filiales, concurrents, partenaires)
- **Interface web autonome** : Développement d'une interface web indépendante de Jupyter

7. Conclusion

Le projet d'agent de recherche d'entreprise présenté dans ce rapport démontre une implémentation réussie d'un système d'agrégation de données en Python. L'architecture modulaire, la gestion efficace des requêtes parallèles et l'interface intuitive constituent des atouts majeurs de cette réalisation.

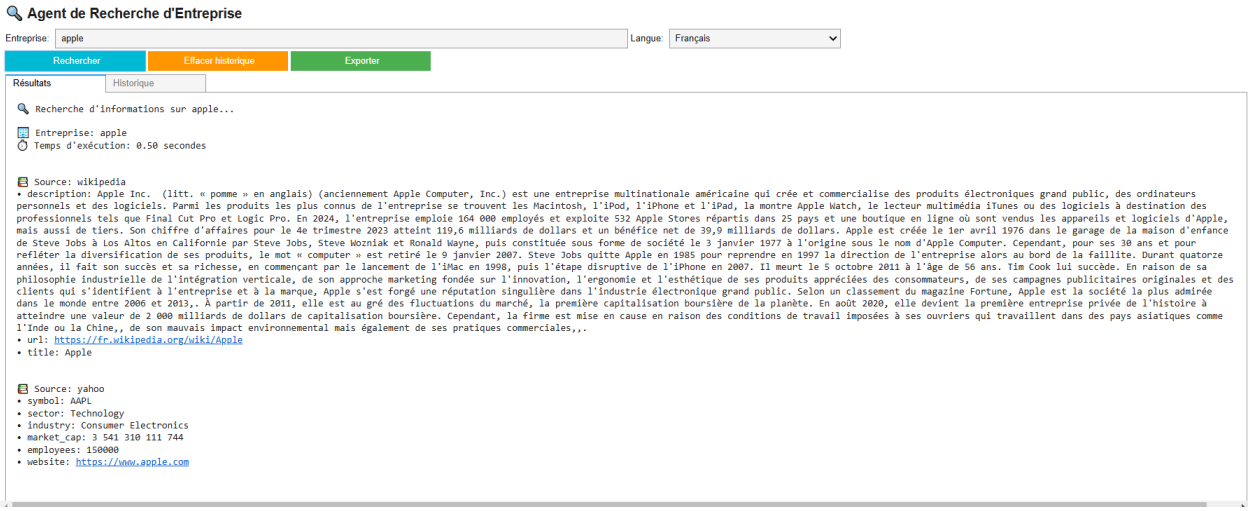
Ce projet illustre l'application de principes de conception avancés en programmation orientée objet et l'utilisation judicieuse des bibliothèques Python pour résoudre un problème concret d'accès à l'information. Malgré quelques limitations, le système offre une base solide pour des développements futurs et des applications dans divers contextes professionnels et académiques.

8. Références

1. Documentation officielle de l'API Wikipedia - https://wikipediaapi.readthedocs.io/
2. Documentation de Yahoo Finance API - https://pypi.org/project/yfinance/
3. Documentation Python ThreadPoolExecutor - https://docs.python.org/3/library/concurrent.futures.html
4. Documentation IPython Widgets - https://ipywidgets.readthedocs.io/
5. Python Logging Cookbook - https://docs.python.org/3/howto/logging-cookbook.htm

9. Annexes

Annexe A: Exemple d'exécution



Annexe B: Structure des fichiers du projet

project/

├── main.py # Script principal

├── historique_recherches.json # Fichier de persistance

└── historique_recherches.csv # Export CSV