POLYTECH MONTPELLIER

Introduction IA transversale - S5

XADO505A

GenAI labs : trouvez un stage à l'étranger

Étudiante :
Nadia LAHYA

Enseignant :
Arsène FOUGEROUSE

31 Janvier 2026

Table des matières

1 Résumé

Ce projet détaille la mise en place d'un système expert automatisé pour la recherche de stages à l'étranger.

En utilisant n8n* [1], Tavily* et Mistral AI, j'ai conçu un workflow intelligent capable de rechercher des sources multilingues (italien et anglais), à les évaluer via un serveur MCP (Model Context Protocol)* local (`mcp_server.py`), à créer un fichier CSV avec les offres trouvées et à extraire les opportunités les plus intéressantes vers Discord.

Je me suis focalisée sur Milan (priorité absolue) et Genève, en témoignant de mon désir de faire un stage principalement en Italie ou dans un pays qui soit à proximité et dans lequel je pourrait tout de même exercer mon anglais. Le système met en place une stratégie double d'interrogation LLM (Large Language Model)* (agent HTTP et agent IA) dans le but d'optimiser la pertinence des réponses.

2 Justification de mon approche

J'ai choisi une installation locale (non-conteneurisée) du serveur MCP et de n8n pour plusieurs raisons :
- Facilité de développement : tests et adaptations du code en temps réel sans rebuild
- Usage personnel : pas de nécessité de portabilité Docker pour ce cas d'usage
- Rapidité d'itération : modifications instantanées pendant le développement
- Conformité aux exigences : la conteneurisation n'était pas une exigence du projet

3 Déployer le MCP Server de scoring de ville

Le serveur Python (`mcp_server.py`) agit comme un oracle géographique pour le workflow, fournissant des scores de qualité de vie basés sur l'indice Numbeo en utilisant le classement de Kaklauskas et al. 2018.

3.1 Chercher le score d'une ville

3.1.1 Préparer l'environnement Python

Le projet utilise uv* pour la gestion de l'environnement virtuel :

```
$ uv init
$ uv add "mcp[cli]"
```

3.1.2 Récupération des données

Le dataset (`city_scores.json`) suit le classement Numbeo, comme cité précédemment. Ce choix s'est imposé car la qualité de vie est un critère important pour moi, afin de choisir où réaliser mon futur stage.

Exemple des données :

```
{
  "edinburgh": 1,
  "zurich": 2,
}
```

1. Les mots accompagnés d'une * sont définis dans le glossaire à la fin du rapport.

3.1.3 Création du serveur

```python
from typing import Dict, List, Optional
from mcp.server.fastmcp import FastMCP
import json

try:
    with open("city_scores.json", "r") as f:
        data = json.load(f)
except FileNotFoundError:
    data = {}
    print("Fichier city_scores.json non trouvé. Le serveur démarre avec des données vides.")

mcp = FastMCP(name="city-ranker", host="0.0.0.0", port=8001)

@mcp.tool()
def find_city_ranking(city: str) -> Dict:
    city_clean = city.lower().strip()
    rank = data.get(city_clean)
    if rank:
        return {"city": city, "ranking": rank, "message": f"La ville de {city} est classée #{rank} au
        ↪  classement Qualité de Vie (N)." }
    else:
        return {"city": city, "ranking": None, "message": f"Aucune donnée de classement pour {city}."}

@mcp.tool()
def list_cities(limit: int = 10) -> Dict:
    sorted_cities = sorted(data.items(), key=lambda item: item[1])
    subset = sorted_cities[:limit]
    cities_list = [{"city": city.title(), "ranking": rank} for city, rank in subset]
    return {"count": len(cities_list), "cities": cities_list}

def main():
    print("Démarrage du serveur MCP City Ranker...")
    print(f"{len(data)} villes chargées")
    print("Utilisez l'inspecteur MCP pour tester:")
    print("npx @modelcontextprotocol/inspector uv run --python 3.11 mcp_server.py")
    #mcp.run()  #STDIO pour l'inspection MCP
    mcp.run(transport="streamable-http", mount_path="/mcp") # HTTP/StreamableHttp pour n8n

if __name__ == "__main__":
    main()
```

Pour mes tests, lors de l'impléméntataion du serveur, j'utilisais l'inspecteur de MCP qui se base sur `STDIO` comme mode de transport. Ainsi, dans le main, au lieu d'utiliser :

```python
mcp.run(transport="streamable-http", mount_path="/mcp")
```

j'utilisais :

```python
mcp.run()
```

4 Création du workflow de recherche de stage

Le workflow n8n constitue le cœur du système d'automatisation.

4.1 Sourcing multilingue : "Town x Techno" & "Prepare query"

4.1.1 Génération des paires ville-technologie

Le nœud `Town x Techno` produit de manière dynamique les associations entre villes et technologies.

Cette configuration met en avant mes priorités géographiques :

```
const cities = [
  { city: "Milan", country: "IT", languages: ["it", "en"] },
  { city: "Geneva", country: "CH", languages: ["fr", "en"] }
 ];

 const techs = [
 { tech: "DevOps",
 synonyms: ["DevSecOps", "Platform Engineering"] },
 { tech: "Cloud",
 synonyms: ["AWS", "Azure", "GCP", "Kubernetes"] }
 ];

// Génération des paires
 return cities.flatMap(cityObj =>
 techs.map(techObj => ({
 city: cityObj.city,
 country: cityObj.country,
 languages: cityObj.languages,
 technology: techObj.tech,
 synonyms: techObj.synonyms
 }))
```

Pour améliorer le taux de correspondance, j'ai introduit un système de gestion des synonymes afin de prendre en compte les multiples sémantiques employées par les recruteurs.

4.1.2 Gestion multilingue adaptative

Le nœud `Prepare query` est un point clé de mon projet : la gestion dynamique des langues multiples. En raison de mon intention d'effectuer principalement mon stage en Italie (Milan) ou dans un pays dans lequel je peux exercer mon anglais (Genève), j'ai conçu des requêtes ajustées sur le plan linguistique :

— Pour Milan : Recherche en italien ou en anglais
— Pour Genève : Recherche en français ou en anglais

Cette approche permet de découvrir des offres locales que des recherches uniquement en anglais auraient manquées.

Exemple de traduction pour Milan :

```
it: {
  internship: '"stage" OR "tirocinio"',
  context: '("candidati" OR "carriere" OR "lavoro" OR "posizioni aperte")',
```

```
  duration: '("2 mesi" OR "9 settimane" OR "estate" OR "luglio" OR "agosto")',
  exclude: '-affitto -stanza -immobiliare' //exclue les annonces de location
}
```

La requête complète générée devient :

```
"DevOps" stage OR tirocinio "Milan"
(2 mesi OR 9 settimane OR estate OR luglio OR agosto)
candidati OR carriere OR lavoro OR posizioni aperte)
2026 -wiki -pedia -tutorial -affitto -stanza
```

4.2 Double stratégie d'appel LLM

Afin de diversifier les nœuds, j'ai utilisé deux approches pour interroger Mistral Cloud :

— IA Agent : Pour l'analyse profonde et l'extraction de données structurées, comme imagé sur la figure 1.

— Nœud HTTP : Pour la mise en forme du message final Discord, comme illustré sur la figure 2.

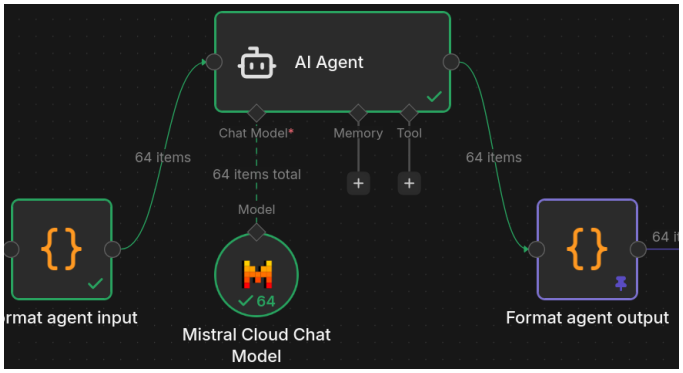

FIGURE 1 – Nœud 'Agent IA' en utilisant Mistral en tant que LLM

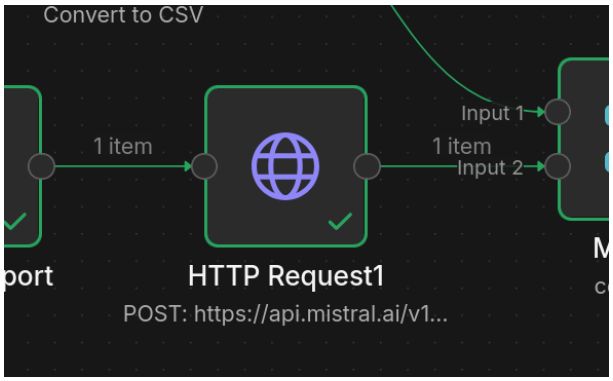

FIGURE 2 – Nœud HTTP faisant appel à l'API Mistral

4.3 Système de classification : TOP / BOF / NON

4.3.1 Justification du système de verdict

Vu volume important de résultats de recherche, j'ai implémenté un système de classification tripartite pour filtrer efficacement les opportunités :

Verdict	Critères d'attribution
TOP	Offre parfaitement alignée : durée 8-10 semaines, technologie correspondante exactement, deadline compatible (après février 2026), rémunération mentionnée, entreprise reconnue
BOF	Offre acceptable avec compromis : durée limite (6-12 semaines), technologie adjacente mais pertinente, localisation secondaire, salaire non précisé mais probable
NON	Offre non pertinente : durée inadéquate (<6 ou >12 semaines), technologie sans rapport, CDI au lieu de stage, deadline passée, localisation géographique erronée

Ce système permet de récupérer automatiquement le Top 3 des meilleures offres pour la notification Discord.

4.3.2 Filtrage et Restructuration des données

Une étape critique a été la restructuration des données entre les nœuds `Format agent input` et `Format agent output`.

Le nœud `Format agent output` effectue plusieurs opérations essentielles :

1. Parsing* sécurisé : Gestion des réponses JSON avec multiples fallback* (ex. score d'une ville a 50 si le score n'est pas connu)
2. Nettoyage : Suppression des marqueurs Markdown (`"'json`)
3. Réorganisation : Mise en ordre des colonnes pour export CSV
4. Fusion des compétences : Transformation de `["Java", "AWS"]` en `"Java, AWS"`
5. Gestion d'erreurs : Capture des offres mal formatées sans interrompre le workflow

4.3.3 Tri et extraction du Top 3

Le nœud `Sort` effectue un tri hiérarchique selon deux critères :

1. Verdict (descendant) : `TOP` > `BOF` > `NON`
2. Ville (descendant) : Milan priorisé alphabétiquement

Cette configuration garantit que Milan apparaît systématiquement en tête des recommandations, conformément à mon souhait n°1 de faire mon stage en Italie.

Le nœud `Filter` supprime ensuite les offres marquées `NON` et les offres de 2025 (périmées), pour ne conserver que les réelles opportunités pour la notification Discord.

4.4 Mon workflow complet

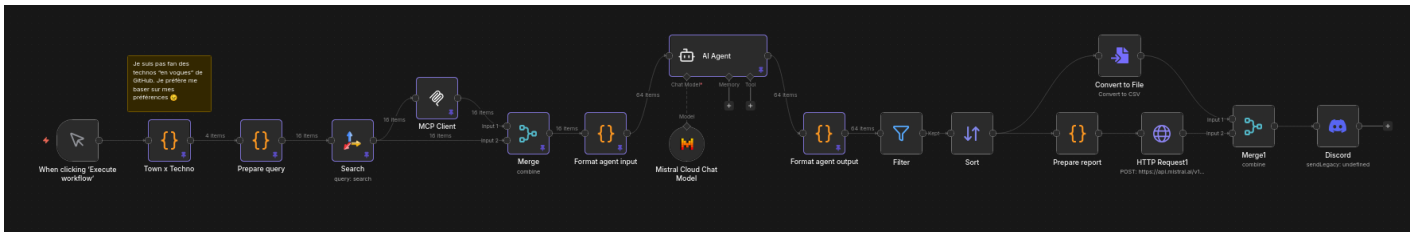

FIGURE 3 – Capture de mon workflow n8n complet

5 Analyse critique et difficultés rencontrées

5.1 Complexité du démarrage

5.1.1 Courbe d'apprentissage abrupte

Le démarrage du projet a été particulièrement laborieu. La principale difficulté résidait dans la compréhension de l'architecture MCP et de son intégration avec n8n :

— Passage stdio → SSE (Server-Sent Events)* : Initialement, ce n'était clair que l'inspecteur et n8n ne communiquaient pas dans le même mode ce qui a fraineé mon démarrage
— Synchronisation des données : Premier verrou technique majeur concernant l'échange de données entre n8n et le serveur local (problèmes de serialization JSON, timeouts, gestion des erreurs de connexion)
— Compréhension du protocole : Temps nécessaire pour maîtriser les concepts de tools, prompts, et ressources dans le contexte MCP

5.2 Les clés d'API

A plusieurs reprises j'ai du changer mon LLM en raison des quotas atteints avec très peu de manipulations.

5.3 Hallucinations et "folies" de l'IA

5.3.1 Problématique majeure

J'ai rencontré des problèmes récurrents et frustrants d'hallucination* où l'IA :
- Cherchait des appartements et logements au lieu de stages professionnels
- Confondait les villes (ex : proposer Gênes au lieu de Genève, ou suggérer des offres en France, suggérer des d'autres ville hors celles demandées)
- Attribuait des scores de ville fantaisistes, ignorant totalement les données fournies par le serveur MCP
- Générait des URLs invalides ou des entreprises inexistantes

Ce comportement imprévisible a nécessité plusieurs modifications dans les prompts ainsi qu'une modification intégrale de la méthode de validation.

5.3.2 Solutions implémentées

Après de multiples itérations, j'ai mis en place les solutions suivantes :

1. Contraintes explicites dans le prompt
2. Validation côté n8n : Filtrage strict des offres avec URLs manquantes, villes absentes du dataset, ou mots-clés suspects (apartment, rent, flat)
3. Exclusions explicites dans les requêtes Tavily : Ajout de l'exclusion `-affitto -stanza -immobiliare -housing -rent -room -flat` pour filtrer les résultats de recherche en amont

5.4 Gestion d'erreurs et robustesse du système

5.4.1 Formats JSON variables

L'IA Agent retournait parfois des formats inconsistants et imprévisibles :
- JSON entouré de `"‘json ... "‘` (marqueurs Markdown)
- Réponse en texte brut au lieu de JSON structuré
- Champs manquants, renommés ou avec types de données incorrects
- Arrays imbriqués de manière non standard

Solution : Implémentation d'un parser robuste avec multiples fallbacks dans `Format agent output`, incluant la détection automatique du format, le nettoyage des marqueurs, et la gestion gracieuse des erreurs.

5.4.2 Timeouts et échecs Tavily

Certaines requêtes Tavily échouaient par timeout (recherches trop complexes ou serveurs distants lents).

Solution : Ajout d'un mécanisme de retry avec délai exponentiel, et limitation stricte à `max_results: 4`.

5.4.3 Synchronisation des nœuds

La restructuration des données entre nœuds nécessitait une gestion d'erreur robuste pour éviter que le workflow ne s'arrête complètement si une seule offre était mal formatée par le LLM.

Solution : Système de try-catch exhaustif avec marquage des erreurs (verdict "ERREUR") permettant au workflow de continuer malgré les échecs partiels.

6 Résultats

6.1 Bilan opérationnel

Tout fonctionne comme attendu. Le workflow est stable, gère correctement les erreurs et remplit parfaitement son rôle malgré quelques folies encore visiles.

6.1.1 Qualité des résultats

Bien que le nombre d'offres réelles soit parfois limité (à cause de la saisonnalité des stages et des spécificités technologiques recherchées), l'efficacité du tri `TOP/BOF/NON` assure un service de haute qualité.

Très peu de propositions inexactes atteignent l'alerte finale Discord, ce qui confirme la performance du système de tri et de gestion des hallucinations.

6.2 Exemple de notification

Le système génère automatiquement un rapport formaté pour Discord, qui comporte :
- un bilan de la recherche qui a été faite
- le TOP 3 des meilleures offres trouvées et pourquoi elles ont été sélectionnées
- un fichier CSV récapitulatif des meilleurs offres trouvées correspondants à ma recherche

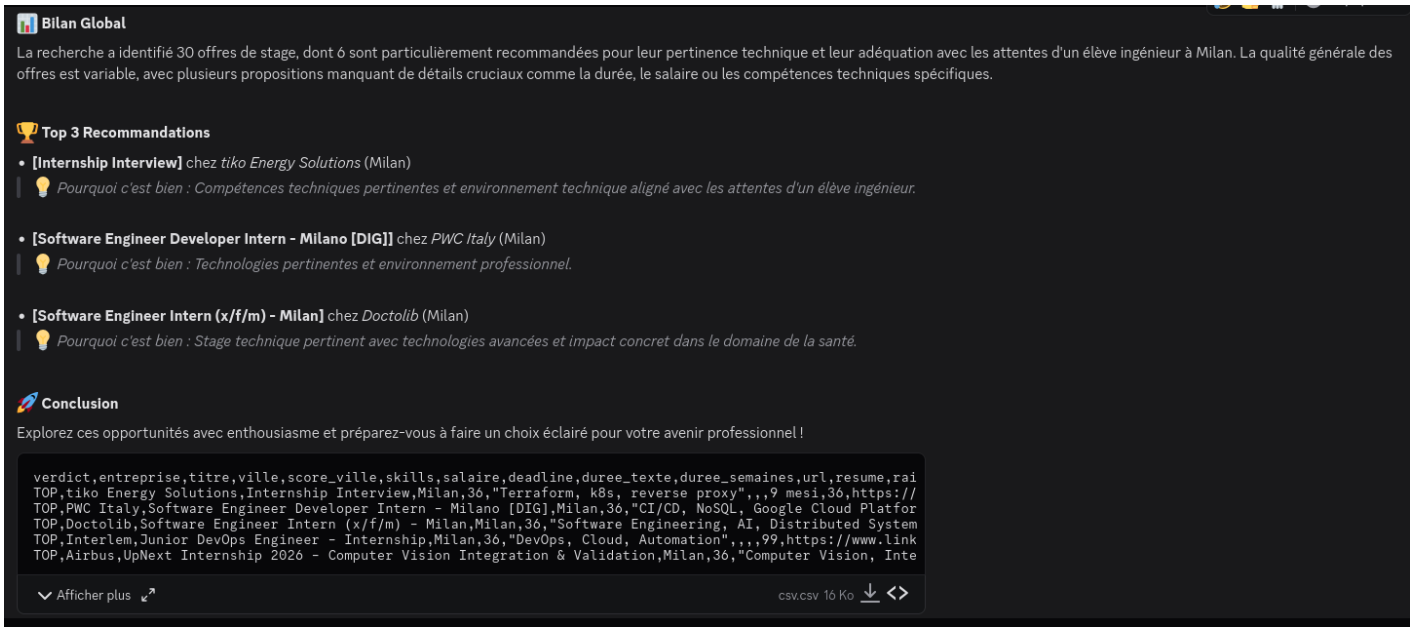

FIGURE 4 – Exemple d'un message discord

7 Conclusion

7.1 Objectifs atteints

Ce projet a permis de développer un système opérationnel et robuste d'automatisation de recherche de stage à l'étranger.

Les objectifs initiaux sont pleinement atteints :

— Gestion multilingue opérationnelle (Italien/Français/Anglais)
— Notation géographique via serveur MCP local (`mcp_server.py`)
— Double approche LLM (Agent IA + Requête HTTP) complémentaire
— Système de filtrage intelligent performant basé sur TOP/BOF/NON
— Alerte automatique des meilleures propositions dans un fichier CSV accompagné d'un message sur Discord (bilan de la recherche + Top 3 des annonces)
— La priorité Milan a été considérée comme le souhait numéro un.
— Gestion d'erreurs solide et flux de travail robuste.
— Gestion maîtrisée de la transformation des données entre les nœuds.

7.2 Compétences techniques acquises

— Maîtrise de n8n : Orchestration de workflows complexes multi-étapes
— Intégration MCP : Connexion de services Python aux LLMs via protocole standardisé
— Prompt Engineering* avancé : Lutte contre les hallucinations et optimisation des résultats
— Manipulation de données : Réorganisation, conversion et vérification
— Multilinguisme : Adaptation dynamique des requêtes selon le contexte linguistique
— APIs REST : Intégration Tavily, Mistral AI, Discord

7.3 Réflexion personnelle

Malgré un début compliqué et frustrant, ce projet m'a donné une vision concrète des problématiques liées à l'automatisation intelligente et des défis associés à l'intégration de différents services d'IA.

Lutter contre les hallucinations de l'IA a été très formateur : j'ai appris l'importance fondamentale de la vérification rigoureuse des données, d'un prompt engineering méticuleux, et de ne jamais accorder une confiance absolue aux résultats d'un LLM.

Est-ce que je l'ulisierai dans ma future recherche de stage ? Certainement, le workwof m'a permis d'avoir des annonces que l'on retouve pas sur les plateformes classiques. Cependant, en raison des limitations des quotas et les problèmes d'hallucinations, je l'utiliserai en complément d'une recherche manuelle traditionnelle. La recherche manuelle me permettra une recherche pointilleuse tandis que le workflow automatisé m'aidera à découvrir des opportunités inattendues.

Choisir une approche locale plutôt que conteneurisée s'est révélé être une décision judicieuse en termes d'apprentissage, qui m'a permis d'effectuer des cycles itératifs rapides. Toutefois, pour une mise en production à long terme, l'utilisation de Docker pour la conteneurisation est indispensable afin de garantir la reproductibilité et la portabilité.

Glossaire

fallback : Solution de secours ou comportement par défaut implémenté dans le code pour éviter une erreur système si une ressource principale (comme une réponse d'IA) fait défaut. 6

hallucination : Phénomène par lequel une intelligence artificielle génère des informations factuellement incorrectes mais présentées de manière convaincante (ex: inventer un salaire ou une localisation). 6

LLM (Large Language Model) : Modèle de langage entraîné sur de vastes quantités de données, capable de comprendre et de générer du texte, utilisé ici pour analyser la pertinence des offres de stage. 2

MCP (Model Context Protocol) : Protocole ouvert développé par Anthropic permettant de connecter des sources de données locales ou des outils personnalisés (comme mon serveur de scoring Python) à des modèles de langage (LLM) de manière standardisée. 2

n8n : Outil d'automatisation de workflow (Low-Code) basé sur les nœuds, permettant d'orchestrer des tâches entre différentes APIs et services. 2

Parsing : Processus d'analyse d'une chaîne de caractères pour la transformer en une structure de données utilisable par un programme (ex: transformer du texte brut en objet JSON). 6

Prompt Engineering : L'art et la science de concevoir, d'affiner et d'optimiser les instructions (prompts) envoyées à une IA pour obtenir les résultats les plus précis et fiables possibles. 8

SSE (Server-Sent Events) : Technologie de push serveur permettant à une application cliente (ici n8n) de recevoir des mises à jour automatiques via une connexion HTTP persistante. Utilisé pour exposer les outils de mon serveur local sur le réseau. 6

Tavily : Moteur de recherche spécifiquement optimisé pour les agents IA, capable de filtrer le "bruit" des pages web pour ne retourner que le contenu textuel pertinent pour un LLM. 2

uv : Gestionnaire de paquets et d'environnements Python ultra-rapide utilisé pour isoler les dépendances du serveur MCP. 2

Références

KAKLAUSKAS, A. et al. (2018). « Quality of city life multiple criteria analysis ». In : *Cities* 72, p. 82-93. ISSN : 0264-2751. DOI : https://doi.org/10.1016/j.cities.2017.08.002. URL : https://www.sciencedirect.com/science/article/pii/S0264275117305905.