MAIL GPS

Send mails to the right person

Thibault Goutorbe

thibault.goutorbe@aivancity.education

—

1. Préparation et Configuration du Projet
- **1.1 Choisir les outils et les librairies nécessaires :**
  - Choisir les langages de programmation (Python, Node.js, etc.)
  - Identifier les librairies pour la manipulation et l'envoi de mails (par ex. : `smtplib`, `email`, `imaplib`, `flask`, etc.)
  - Installer les dépendances (ex. : `pip install smtplib`, `imaplib`, `requests`, etc.)

- **1.2 Configurer un environnement de développement :**
  - Mise en place de l’environnement virtuel pour isoler les dépendances (ex. : `virtualenv`, `venv`)
  - Gestion des configurations sensibles (par ex., variables d’environnement pour les identifiants de connexion au serveur mail)
  - Créer une structure de projet claire, avec des dossiers pour les tests, la configuration et les fichiers de code principaux.

—

2. Implémentation de l'Envoi de Mails Automatique

- **2.1 Mise en place des scripts d'envoi de mails :**
  - Utiliser la librairie SMTP (`smtplib` pour Python par exemple) pour configurer l'envoi de mails.
  - Structurer le message avec la librairie `email` (créer des objets MIME pour les pièces jointes, le corps du mail, etc.)

- **2.2 Ajouter des fonctionnalités de personnalisation des mails :**
  - Création de modèles de mails en fonction des demandes :
    - Mail pour les demandes de programme scolaire
    - Mail pour les demandes d'horaires
    - Autres demandes types
  - Utiliser des variables pour personnaliser les emails en fonction du contenu (par ex., insérer le nom du destinataire ou des informations spécifiques).

—

3. Connexion avec l'API Mistral (Version Expérimentation)
- **3.1 Étudier l'API Mistral :**
  - Lire la documentation de l'API Mistral pour comprendre les endpoints disponibles, les formats de requête et de réponse.

- **3.2 Implémenter la connexion à l’API Mistral :**
  - Utiliser une librairie HTTP pour effectuer des appels à l'API (ex. : `requests` pour Python).
  - Créer une fonction qui interagit avec l'API pour envoyer des données du mail reçu.
  - Gérer les erreurs de l'API et implémenter des mécanismes de retry si nécessaire.

—

4. Traitement des Mails Entrants et Extraction du Texte
- **4.1 Configurer IMAP pour recevoir les mails :**
  - Utiliser la librairie `imaplib` pour configurer la récupération de mails depuis une boîte de réception (ou une boîte spécifique).
  - Configurer un accès sécurisé (SSL/TLS) pour sécuriser la connexion.

- **4.2 Extraction du contenu des mails :**
  - Extraire le texte et les pièces jointes des mails.
  - Utiliser des librairies comme `email.parser` pour récupérer le corps du texte.
  - Filtrer les mails pertinents (ceux liés aux programmes, horaires, autres demandes).
  - Stocker le texte des mails dans une base de données locale ou dans un fichier pour une analyse future.

—

5. Envoi des Requêtes à l'API et Récupération des Réponses
- **5.1 Envoi des informations extraites vers l'API Mistral :**
  - Envoyer les données pertinentes (contenu du mail) à l’API via des requêtes HTTP POST/GET.
  - Attendre la réponse de l’API pour chaque mail traité.

- **5.2 Traitement des réponses de l'API :**
  - Interpréter les réponses de l’API (formats JSON/XML).
  - En fonction des réponses, générer des mails automatiques pour répondre à l’utilisateur.
  - Ajouter des logs pour surveiller les interactions avec l'API (pour le débogage et le suivi des performances).

—

6. Automatisation de la Réponse
- **6.1 Génération automatique de la réponse à l'utilisateur :**
  - Créer un générateur de réponses en fonction des informations fournies par l’API.

- Structurer le mail de réponse (ex. : "Merci pour votre demande. Voici les horaires de l'école pour cette semaine : ...").

- **6.2 Envoi automatique du mail de réponse :**
  - Utiliser les mêmes scripts d'envoi de mails développés en **2.1** pour répondre à l'utilisateur.
  - Ajouter un système de suivi pour vérifier si l'email de réponse a bien été envoyé et reçu.

—

7. Tests et Validation
- **7.1 Création de mails de test :**
  - Simuler différents scénarios de demande (programme, horaires, questions générales, etc.).
  - Envoyer ces mails à ton système pour valider le bon fonctionnement des différentes étapes (extraction, interaction avec l'API, réponse automatique).

- **7.2 Tests unitaires et intégration continue :**
  - Tester chaque module séparément (extraction de texte, connexion API, envoi de mail).
  - Utiliser un cadre de tests comme `unittest` ou `pytest` pour s'assurer que chaque composant fonctionne correctement.

- Implémenter un processus de CI/CD (par ex., avec GitLab ou GitHub Actions) pour automatiser les tests à chaque modification du code.

—

8. Optimisation et Sécurisation
- **8.1 Sécurisation des informations sensibles :**
  - Chiffrer les informations sensibles (identifiants de messagerie, clés API) avec des outils comme `dotenv` pour gérer les variables d'environnement.
  - Mettre en place des politiques de gestion d’erreurs pour gérer les défaillances API ou les échecs d’envoi de mail.

- **8.2 Amélioration de la gestion des requêtes :**
  - Utiliser des files d’attente (par ex., RabbitMQ ou Celery) pour gérer les demandes de manière asynchrone et éviter les blocages si plusieurs mails sont reçus simultanément.

- **8.3 Suivi et journalisation des activités :**
  - Implémenter des logs pour suivre chaque étape (réception du mail, envoi à l’API, réponse, etc.).
  - Configurer des alertes en cas d'erreurs critiques (en utilisant des outils comme `Sentry` ou des systèmes d'alerte mail).

—

9. Documentation et Livraison
- **9.1 Documenter le projet :**
  - Documenter chaque module avec des commentaires clairs.
  - Rédiger un guide utilisateur expliquant comment configurer et exécuter le programme.

- **9.2 Préparer le déploiement :**
  - Préparer les scripts de déploiement (Docker, configurations pour les serveurs de production).
  - Tester la solution sur un environnement de production simulé.

—