Travaux Pratiques
API Géographique avec Flask

Alexandre JOUSSET

10 septembre 2025

Introduction

Flask est un micro-framework Python permettant de créer rapidement des APIs et applications web. Dans ce TP, vous allez construire pas à pas une **API géographique** capable de :

— Retourner un message de bienvenue.
— Obtenir les coordonnées GPS d'une ville via OpenStreetMap.
— Calculer la distance entre deux villes avec OpenRouteService.
— (Bonus) Conserver l'historique des recherches.

Objectif

À la fin du TP, vous aurez développé une API REST complète en Flask, intégrant des données externes et appliquant les bonnes pratiques de structuration et de documentation.

Matériel requis

— Python 3 installé.
— Bibliothèques : `flask`, `requests`.
— Un éditeur de code (VSCode, PyCharm, etc.).

Consignes générales

— Travaillez en binôme ou seul.
— Testez vos endpoints avec `curl`, Postman ou un navigateur.
— Gardez un code clair et commenté.

Exercices

Exercice 1 : Installation et Hello World

1. Créez un dossier `api-geo`.
2. Installez Flask.

3. Créez un fichier `app.py` avec un endpoint `/` qui retourne un message de bienvenue.
4. Lancez le serveur et testez l'URL dans un navigateur.

Question : Pourquoi le mode `debug=True` est-il utile pendant le développement ?

Exercice 2 : Coordonnées d'une ville

1. Installez la bibliothèque `requests`.
2. Ajoutez un endpoint `/coords/<city>` qui utilise l'API Nominatim d'OpenStreetMap pour retourner la latitude et la longitude de la ville passée en paramètre.
3. Testez avec plusieurs villes (`/coords/Paris`, `/coords/Londres`, etc.).

Question : Pourquoi faut-il préciser `format=json` dans la requête à Nominatim ?

Exercice 3 : Calculer une distance entre deux villes

1. Créez un compte sur `https://openrouteservice.org/` et récupérez une clé API gratuite.
2. Créez un endpoint `/distance` qui prend en paramètres GET `from` et `to`.
3. Utilisez Nominatim pour récupérer les coordonnées des deux villes.
4. Utilisez OpenRouteService pour calculer la distance routière entre elles.
5. Retournez la distance en kilomètres dans la réponse JSON.

Question : Pourquoi est-il important de limiter le nombre d'appels ?

Exercice 4 : Améliorer la structure

— Déplacez la fonction qui récupère les coordonnées dans un fichier `utils.py`.
— Importez-la dans `app.py` pour éviter de répéter le code.

Question : Pourquoi est-il important de factoriser le code ?

Exercice 5 (Bonus) : Historique des recherches

1. Ajoutez une structure (liste ou autre) pour stocker chaque appel à `/distance` (ville de départ, ville d'arrivée, distance).
2. Créez un endpoint `/history` qui retourne cet historique au format JSON.
3. (Optionnel) Permettez de vider l'historique via un endpoint `DELETE`.

Question : Comment rendre cet historique persistant même après redémarrage du serveur ?

Questions de réflexion

1. Quels sont les avantages de Flask par rapport à un framework plus lourd comme Django pour ce type de projet ?
2. Quelles bonnes pratiques appliquer pour nommer et documenter ses endpoints ?
3. Pourquoi limiter le nombre d'appels à des APIs externes ?
4. Comment sécuriser une clé API dans un projet collaboratif ?

Livrables

— Le code source complet de l'API.
— Un fichier `README.md` expliquant comment installer et utiliser l'API.
— Les réponses aux questions posées dans chaque exercice.
— La documentation des endpoints.

Date de rendu

Le TP doit être rendu avant le **17 septembre 2025** à 18h.