# PROJET 7

**Lien GitHub:** https://github.com/Tony380/Projet7

**Lien Trello:** https://trello.com/b/EgMK9SG8/projet-7

**Lien Heroku:** https://grandpybot2020.herokuapp.com/

## CADRE DE DEPART:

Après avoir créer un tableau Trello afin de définir les objectifs du projet et les étapes à suivre suivant une méthodologie agile, j'ai créé un repo Git et un environnement de travail pour le développement de l'application.

## FLASK :

La première étape a été la mise en place de Flask en créant dans le projet une arborescence de dossiers qui allaient plus tard contenir tous les fichiers nécessaire au fonctionnement de l'application, ainsi que la création des fichiers views.py, __init__.py et run.py pour lancer cette dernière.

## LE TEST DRIVEN DEVELOPMENT :

Afin de suivre cette méthodologie, qui consistent à écrire les tests de nos requêtes aux différentes API utilisées (Google et Wikipédia) et de notre parser, j'ai créé les fichiers suivants :

- test_parser.py
- test_gmap.py
- test_wiki.py

Ces derniers sont inclus dans le dossier « tests ».

L'utilisation des mocks, pour simuler la réponse de nos requêtes, a été nécessaire.

## LE DEVELOPPEMENT EN PYTHON :

Après avoir développé le parser, à l'aide de Stop_words, j'ai fait la nécessaire acquisition d'une clé API de Google, et écrit les requêtes aux différentes API.

J'ai ensuite écrit le code, dans le fichier views.py, permettant de renvoyer la réponse de nos API (adresse, extrait de la page Wikipédia et son url, ainsi que les coordonnées du lieu recherché par l'utilisateur afin d'afficher la carte de Google Maps. Il est à noter que l'utilisation de deux API de Google a été nécessaire. La première, côte back-end pour rechercher les coordonnées (Geocoding API), la seconde côté front-end, pour afficher la carte (Maps JavaScript API).

## JAVASCRIPT :

Développement du fichier main.js (mis dans le dossier static), qui envoie une requête à notre back-end, et qui reçoit sa réponse au format json (sous forme de dictionnaire).

Pour cela, j'ai utilisé Ajax, ainsi qu'une fonction addEventListener pour déclencher l'opération, quand l'utilisateur cliquera sur le bouton envoyer ou appuiera sur la touche entrée pour valider sa recherche. Ce dernier point sera l'objet du prochain paragraphe.

## HTML – CCS :

La dernière étape quant au développement de notre application, a été la création d'un fichier HTML qui va contenir la structure de notre page web, c'est-à-dire les éléments qui apparaîtront sur l'écran de l'utilisateur (titre de la page, champs de saisi, bouton d'envoi.

Enfin, pour le rendu visuel et le côté responsive de notre page, création d'un fichier CSS lié à nôtre fichier HTML.

## HEROKU :

Pour finir, j'ai créé un compte sur Heroku et lié mon compte GitHub à ce dernier.

J'ai pu mettre en ligne l'application grâce à la commande « git push heroku master », en ligne de commande, à la manière de GitHub.

L'application est donc désormais en ligne.

## DIFFICULTES RENCONTREES :

Trois difficultés ont été rencontrées au cours du développement de cette application :

- prise en main de Flask
- utilisation des mocks pour les tests
- gestion du côté responsive en CSS

Ces difficultés ont été surmontés grâce au cours d'Openclassrooms et de tutoriel sur internet en complément.

## CONCLUSION :

Il est possible d'améliorer ce projet en ajoutant par exemple une base de données afin de sauvegarder les recherches de l'utilisateur ou en y intégrant des polices d'écritures spéciales et des effets visuels sophistiqués. Bien d'autres améliorations sont bien évidemment possibles.

Pour la plus grande partie de ce projet, l'application s'en tient au cahier des charges.