**GrandPy Bot**

Lien du projet : LIEN HEROKU

**Architecture du projet :**

Le fichier run.py permet de lancer l’application en ligne de commande. Le module ocapp contient les tests unitaires, le dossier templates avec les fichiers html et un dossier static pour le css et les images nécessaires. Le fichier main.py est dédié au traitement de la question, c’est-à-dire qu’il la parse, effectue les recherches via les APIs Google Maps et MediaWiki puis renvoie leurs réponses. Ces réponses sont utilisées dans le fichier views.py qui sert simplement à décrire les vues utilisées.

**Partie 1 :** le front end

Le front-end a été développé en ayant d’abord en tête l’apparence visuelle du site. Je souhaitais quelque chose d’épuré, ma première inspiration étant la homepage de Google. Dans une optique d’accès au plus grand monde, je prévois tout de même un bandeau en haut de la page afin d’expliquer brièvement le principe du site.

Le CSS du site a été écrit avec Bootstrap puisque le principe de grille du framework s’adapte très bien au design du site adopté. Il y a donc un header avec un logo et une phrase d’accroche explicative surlignés en orange (le même que celui d’Openclassrooms), le titre de la page est affiché en grand sur la page avec en dessous le formulaire où l’utilisateur pose sa question. Un bouton sur le côté sur les écrans larges, qui se positionne en-dessous de la barre de recherche pour les écrans plus petits, permet de lancer la recherche. Enfin, un footer en bas complète la page en affichant un lien vers le GitHub du projet.

Je n’ai rencontré aucune difficulté concernant la partie html du site, étant donné que le site n’a pas un contenu très étoffé mais la familiarisation avec Bootstrap m’a demandé un peu plus de temps. C’est essentiellement le principe des colonnes qui m’a demandé un temps d’adaptation. J’ai cependant beaucoup apprécié le système de lignes où disposer ses éléments et le fait que le passage au responsive se fasse aussi facilement.

**Partie 2 :** le back-end

J’ai très vite organisé le back-end en trois classes, chacune dédiée à une étape du processus.

La première classe concerne le parsing. Cette partie contenait pour moi une grande inconnue : le parser. Avant d’écrire mes tests et la moindre classe, j’ai donc pris un peu de temps pour comprendre ce qu’il fallait supprimer, garder, et surtout comment le faire. La liste de stop words fournie en énoncé permet de supprimer l’essentiel des mots inutiles issus des questions, mais un problème subsiste assez régulièrement. En effet, la liste des stop-words ne permet pas de détecter les verbes. J’ai donc dû chercher un moyen de régler ce problème. La solution vient de l’API du site ultralingua qui renvoie, pour un mot, un dictionnaire si ce n’est pas un verbe et une liste dans le cas contraire, quelle que soit la conjugaison du mot.

La classe suivante est dédiée à la requête à Google Maps. La seule méthode de la classe permet de créer une requête en prenant en paramètre la question parsée. De la réponse renvoyée par l’API, on extrait la latitude et la longitude du lieu, qui nous serviront à la fois pour afficher la mini map sur le site et une requête lors de la prochaine étape. Cette étape-ci du programme a été facile à réaliser, l’API étant extrêmement bien documentée.

Enfin, j’ai fini par créer une classe dédiée aux requêtes à l’API MediaWiki. Celle-ci étant, il me semble, moins bien documentée j’avais commencé par faire une requête en extrayant du résultat de la requête de Google Maps le nom de la rue à chercher. Cependant, cette méthode en plus de ne pas donner de bons résultats en donne aussi parfois des faussés. Ainsi, si un bâtiment est situé dans une rue avec un nom connu il n’est pas assuré que la page Wikipedia provenant de la requête soit celle de la rue demandée. Pour éviter ce résultat non voulu, j’ai découvert un module de l’API qui prend en paramètre une latitude et une longitude. La réponse contient, dans un rayon précisé par le développeur, tous les lieux avec une page Wikipedia avec l’id de leur page. Il suffit avec une seconde méthode de faire une requête sur l’id de la première page. Le texte renvoyé par l’API contient toutes les balises html, je les supprime grâce à une simple regex.

Afin de développer ces classes, j’ai suivi la méthode du Test Driven Development, même si je me suis vite rendu compte de l’importance de la plannification. En effet, ma classe Parser a évolué avec l’ajout des méthodes pour un deuxième parsing ou encore la regex de la réponse de MediaWiki. J’ai donc dû écrire de nouveaux tests au fur et à mesure du développement et parfois modifier certains anciens tests. J’ai aussi eu quelques difficultés avec les mocks, notamment parce qu’il s’agissait ici de mocker une méthode de classe. Malgré ces problèmes, c’est une méthode que j’ai trouvé très intéressante puisqu’elle oblige à prendre le temps de la réflexion avant de devoir écrire du code.

**Partie 3 :** AJAX

L’échange des données entre le front-end et le back-end se fait comme demandé avec AJAX. J’ai utilisé le framework jQuery pour implémenter cette fonctionnalité. Je l’ai trouvé plus simple à lire et à apprendre que javascript qui est plus verbeux. C’est une partie relativement petite du projet, bien qu’essentielle, et je n’ai donc pas voulu me compliquer la tâche.

L’échange des données se fait donc grâce à la fonction ajax() de jQuery, qui prend en argument la nature de la requête, ici « POST », le contenu du formulaire et l’url où envoyer les données. La question posée par l’utilisateur est traitée dans le fichier « main.py » qui renvoie les données nécessaires à l’affichage de la réponse. À savoir : les coordonnées GPS du lieu, ainsi que l’anecdote tirée de Wikipedia. En cas d’erreur, le programme ne renvoie qu’un message à afficher.

En apparence simple, cette partie du projet est celle qui m’a donné le plus de mal. J’ai en effet eu beaucoup de mal à comprendre comment les données étaient échangées entre le client et le serveur, et surtout comment renvoyer la page.

**Conclusion :**

Ce projet m’a permis de développer mon premier site web ce qui est à mon avis une étape très importante dans le parcours. J’ai donc pu me familiariser avec Flask qui est certes très pratique pour développer rapidement un petit projet mais un peu difficile d’accès pour les débutants. J’ai aussi pu découvrir les bases d’html et css et notamment Bootstrap que j’ai trouvé très pratique pour organiser visuellement une page très facilement.