Démarche du projet 7 (GrandPy Bot)

Le projet est disponible à l'adresse : https://grandpybot-em.herokuapp.com/

1- Planification du projet :

Pour planifier le projet, j'ai suivi les étapes définies dans la description du projet sur le site d'Openclassrooms.
Ces différentes étapes sont listées dans un tableau Trello à l'adresse suivante : https://trello.com/b/nyPeWNyo/projet-7
J'ai ensuite initialisé un repo GitHub : https://github.com/MINCARELLI13/GrandPy_Bot.git

2- Initialisation de Flask :

Après avoir fini d'étudier l'ensemble des cours, j'ai créé une toute première version de site basique avec uniquement une page d'accueil en HTML et CSS et j'ai vérifié que l'appel à cette page via l'application Flask fonctionné correctement.

3- Interface utilisateur :

Une fois mon site fonctionnel, j'ai créé le formulaire qui allait servir à entrer le texte utilisateur. J'ai alors utilisé Bootstrap afin d'avoir un design moderne et responsive.

J'ai ensuite étudié le fonctionnement de l'API Fetch pour traiter l'envoi des données du formulaire dans Flask. La première version fonctionnelle de l'appel avec Fetch prenait en entrée le message envoyé par l'utilisateur dans le formulaire, et en retour, l'affichait directement dans la zone de chat.

4- Gestion du parser :

Le parser a pour but de sélectionner les mots clés de la phrase entrée dans le formulaire par l'utilisateur afin d'effectuer une recherche pertinente au niveau de l'API Google Places.

Pour cela, j'ai utilisé les expressions régulières pour ne récupérer que les mots composés d'au moins deux lettres, puis j'ai supprimé tous les mots non significatifs de la question (les mots vides).

Cependant, même après avoir nettoyé la phrase entrée par l'utilisateur, les recherches effectuées directement dans Google ne donnaient pas de résultats pertinents et ceci à cause des verbes qui "polluaient" la question.

Une fois supprimés tous les verbes (infinitifs et formes conjugués) contenus dans la question de l'utilisateur, les réponses renvoyées par le moteur de recherche Google ont été positives.

J'ai utilisé le Test Driven Development afin de vérifier que les résultats obtenus correspondaient à mes attentes pour différentes phrases entrées, avec différents cas d'utilisation (signes de ponctuation et apostrophes, phrase constituée uniquement de stopwords...).

5- Utilisation de l'API Media Wiki :

J'ai choisi de commencer par l'implémentation des appels à l'API Media Wiki pour des raisons de simplicité par rapport à l'API Google Places : l'API Media Wiki est ouverte et ne requiert pas de clé pour son installation.
J'ai donc pris connaissance du fonctionnement des requêtes à partir d'un 'titre' (le nom du site recherché) renvoyant un identifiant 'pageid' et grâce auquel une nouvelle requête nous permet d'obtenir l'histoire du site au format JSON - en fait, on obtient plus précisément l'introduction de l'article de Wikipedia.
Sur la version finale de l'application, le 'titre' utilisé pour les requêtes sur l'API Media Wiki est issu de la réponse de l'API Google Places.

6- Utilisation de l'API Google Places :

J'ai ensuite regardé quelle API Google - parmi toutes celles existantes - pouvait me permettre d'obtenir l'adresse du site recherché ***à partir de la question posée*** par l'utilisateur. Au final, il m'a semblé que l'API Google Places était la plus adaptée à mon besoin d'autant plus que la réponse à la requête contenait également le nom du site 'en clair' ce qui me permettait de pouvoir effectuer des recherches 'propres' dans l'API Media Wiki et ainsi éviter un certain nombre d'échecs lors des requêtes avec cette dernière API.

7- Utilisation de l'API Google Maps :

Cette API était indispensable pour l'affichage d'une carte du lieu recherché.
Après avoir rapidement regardé la documentation, j'ai récupéré le code javascript proposé par Google pour l'intégrer directement à mon code.
Tout a été assez simple pour l'intégration du code de Google dans mon document javascript sauf que la carte Google restait figée une fois insérée dans le "chat".
Malgré de nombreuses recherches sur ce problème, je n'ai rien trouvé sur le web. Après plusieurs jours d'essais sur une page web "basique" (sans chat et sans passer par Flask) je me suis aperçu que le problème était que la modification de la zone de "chat" après l'affichage de la carte Google figeait cette carte. Pour résoudre ce problème, j'ai commencé tout d'abord par générer des sous-zones d'affichages à l'intérieur de la zone de "chat" - ce qui a effectivement modifié ma zone de "chat" - mais ce n'est qu'après seulement que j'ai intégré toutes les réponses aux différentes API directement dans les sous-zones - et parmi ces "réponses" il y a évidemment la carte recherchée. Aussi, l'affichage de la carte et des autres informations collectées se fait maintenant directement dans les sous-zones créées précédemment mais sans modification directe de la zone "chat", ce qui évite que la carte Google ne se fige.

8- Mise en ligne sur Heroku :

Une fois toutes ces étapes terminées, lorsque le projet était complet en environnement de développement, je l'ai mis en ligne sur Heroku en suivant la documentation pour l'installation, la création du fichier Procfile et le push du repo en production. Une erreur inattendue est apparue lors de cette mise en ligne et après recherche sur le web il est apparu que la lecture des fichiers textes nécessitait l'utilisation de l'encodage en "latin-1" et non en "utf-8" comme cela est recommandé couramment.