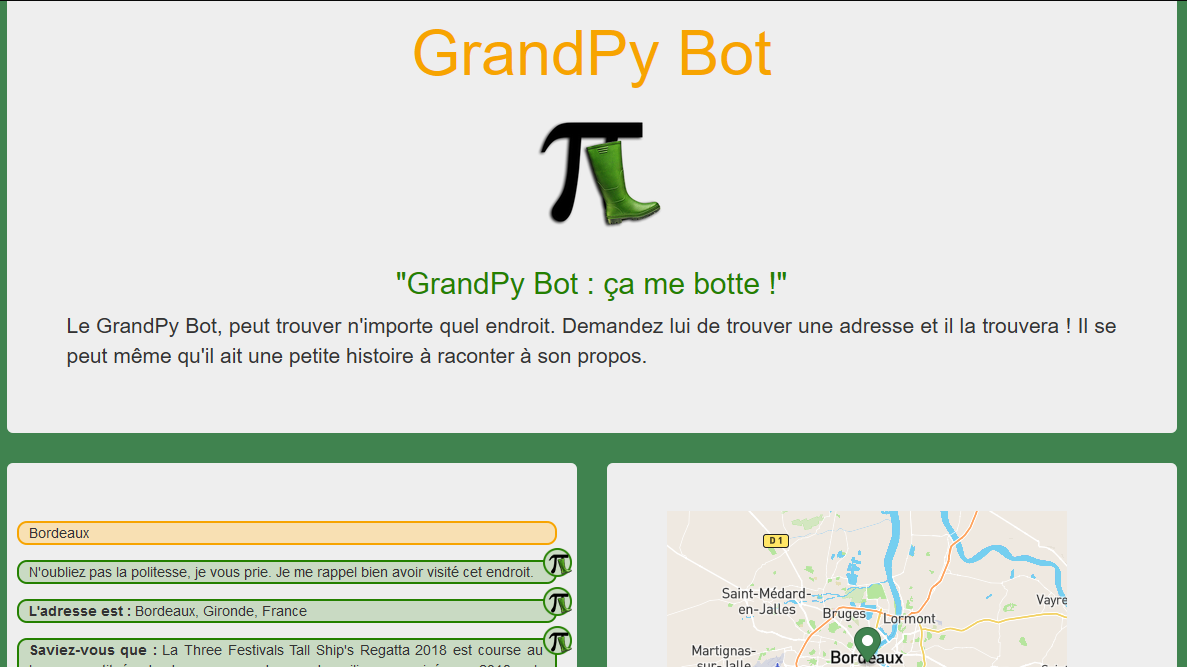
Présentation de la démarche du projet 7 :   
Créez GrandPy Bot, le papy-robot.



Loïc ROMERO

[loic.romero1@gmail.com](mailto:loic.romero1@gmail.com)

Mentor Openclassrooms :

Antonino CANDIDO

Lien vers le code source du projet :

<https://github.com/ROL-1/P7-GrandPy-Bot>

Lien vers le Trello :

<https://trello.com/b/HvdsLlAh/p7-grandpy-bot>

Démarche initiale :

Dans un premier temps, **l'étude du cahier des charges** m'a permis de déterminer que le projet se découperait dans les parties suivantes :

- la création de requêtes pour récupérer les informations de deux apis **API** (Mapbox et Mediawiki).

- la création d'une page web en utilisant HTML 5, Bootstrap pour la partie responsive et Ajax (**jQuery**) pour gérer le formulaire.

- la réalisation d'un programme, avec **Python** 3 et **Flask**, pour : distribuer les actions précédentes, réaliser le filtrage et le 'nettoyage' des données transmises par l’utilisateur et leur analyse pour générer des réponses adaptées.

- la création de **tests** par le biais de pytest.

Sachant que le projet devait être poursuivi selon le **TDD** (Test Driven Development), et que cette méthode nécessite de faire preuve d’anticipation, j’ai estimé nécessaire de réaliser une analyse préliminaire rigoureuse. J'ai découpé le cahier des charges en tâche à l'aide de **Trello**. Ce qui est venu en complément de ma démarche de travail passant par l'utilisation d'un éditeur de texte pour créer un tableau de bord et une **Todo**-liste que je tiens à jour. Liste de laquelle je pars pour écrire d'abord en pseudo-code ce que je souhaite réaliser, avant de coder.

Puis j’ai réalisé un « *dossier de spécifications fonctionnelles et techniques* » en suivant le modèle **UML** des projets précédents. Ce qui m’a permis de bien définir le périmètre fonctionnel. Cela m’a également permis de découvrir des sujets pour lesquels j’avais besoin d’approfondir mes connaissances.

J’ai notamment, j'ai lu la **documentation** **des** **API** de Mapbox et Mediawiki. Et j'ai testé leur utilisation avec Postman, jusqu'à définir les champs que je souhaitais utiliser. Cela sous format JSON, pour préparer l'insertion dans le code et les débugs à venir.

Ensuite, j'ai suivi les **cours** Openclassrooms associés à cette étape du parcours, ayant déjà suivi les cours sur le web, j’ai pu me concentrer sur ceux sur Javascript, jQuery et Flask. Je me suis également appuyé sur le **webinaires** proposé par Thierry Chappuis ([pytest](https://www.youtube.com/watch?v=udW_KcUv6RM&list=PLq7zFUM3vd6NNwb_5v0dj-Q2rt4-Ffr9F&index=11)).

J'ai créé le repo sur **Github** et commencé la création en posant les jalons d'une première architecture du programme.

Création du programme :

L’installation du server Flask m’a permis de créer la page web. J’ai rapidement enrichi la page avec des premiers éléments de CSS permettant d’avoir une structure responsive. Puis j’ai développé la partie dynamique avec jQuery jusqu’à obtenir un retour de la requête passée dans le formulaire.

On retrouve donc dans le module **views**.py les deux routes. L’une adresse la page elle-même en récupérant la clef de l’API MAPBOX dans les variables d’environnement. Et la seconde gère la récupération du formulaire pour passer ses informations au programme avant de retourner un dictionnaire contenant les réponses. Celles-ci s’affichent alors dynamiquement grâce à jQuery sans avoir besoin de recharger la page.

Le cœur du programme se trouve dans le module **main.py**. La classe du même nom, distribue le lancement des différents modules et gère les cas d’échecs par une série de try/except. L’objectif est de toujours proposer une réponse à l’utilisateur.

Une phase de préparation de la « string » récupérée du formulaire passe par le biais des modules **cleaner.py** et **parser.py**. L’objectif est de ne garder que des mots utiles pour la recherche. On va notamment retirer les mots contenus dans la liste mise à disposition sur github : [stopwords](https://github.com/6/stopwords-json/blob/master/dist/fr.json). On va également utiliser **interpreter.py** pour analyser ces mots et générer des réactions. Il s’agira ici de proposer une réponse pré-écrite dans le fichier « bot\_answers.json ».

Ensuite les modules **geocoding.py** et **wikipedia.py** vont successivement lancer des recherches sur les API de Mapbox et MediaWiki. Cependant, dans le cas où Mapbox ne retourne pas de résultat, MediaWiki n’est pas sollicitée., celle-ci attendant les *coordonnées* trouvées par la recherche passée à Mapbox.

Enfin les résultats positifs, ou négatifs sont stockés dans un dictionnaire que retourne la classe views.py par le biais d’un objet « response » par la fonction jsonify de Flask. Alors, le navigateur met à jour la page, faisant cesser l’animation d’attente (**loading.gif**).

Axes d’amélioration du programme :

* Le programme a initialisé l’analyse du des mots transmis par l’utilisateur. Notamment la détection d’une question pour obtenir l’heure, il est possible de créer une réponse adéquate. De même il serait possible de solliciter d’autres APIs pour répondre à d’autres questions (météo, numéro de téléphones, etc...)
* La pertinence des résultats pourrait être améliorée en ajoutant des attributs à passer aux APIs.
* Il serait souhaitable d’ajouter un algorithme pour affiner la recherche dans le cas où les mots passés ne retournent pas de résultat.

Difficultés rencontrées :

* La difficulté principale a été de réussir à travailler en TDD. Cela nécessite de l’anticipation et de maitriser ses connaissances en python quand je suis encore à les consolider. Néanmoins j’ai bien perçu l’intérêt de la démarche, cela aide entre autres choses à créer plus facilement des objets manipulables en Programmation Orientée Objet.