Démarche du projet 7 : Créez GrandPy Bot le papy-robot

Ce document à pour but d'expliquer la démarche et l'algorithme que j'ai utilisé pour réaliser le projet 7 de la formation OpenClassrooms développeur d'application Python.

Voici les liens vers :

- mon code source : https://github.com/micktymoon/P7_GrandPy_Bot
- mon site web : https://grandpybotceline.herokuapp.com/home
- mon Trello : https://trello.com/b/JjwN5vwx/grandpy-bot

Pour commencer, j'ai créé un fichier « routes.py » contenant une application Flask et les différentes routes de mon site web.

La première route s'appelle « /home », elle va me permettre d'afficher la page d'accueil de mon site.

Afin d'afficher cette page, j'ai créé un fichier HTML contenant les différents composants de ma page. Soit une en-tête contenant une image et la phrase d'accroche, un corps de texte contenant le chat et le formulaire permettant de poser une question à GrandPy Bot, et enfin un pied de page contenant mon nom et prénom, ainsi que des icônes redirigeant vers mes pages personnelles des réseaux sociaux. Pour la mise en page j'ai utilisé le modèle CSS bootstrap SOLAR, ainsi qu'un fichier CSS afin d'utiliser Flexbox et de peaufiner les détails de la mise en page.

Une fois la page terminée, je me suis intéressée au parser. Celui-ci va retourner le lieu dont on demande l'adresse. Pour se faire, il prend en paramètre la question posée, vérifie si il y a dans la question les mots « l'adresse » ou « où se trouve », si oui il récupère la suite de la question jusqu'à la prochaine ponctuation et retire les « stepword » afin de ne récupérer que le lieu. Et si les mots ne sont pas présent, alors il retourne « Faux ». J'ai eu quelques soucis lors de la création du parser, notamment dû aux espaces et à la ponctuation. J'ai pu les résoudre en utilisant « re.split » afin de ne récupérer que les mots de la question.

Je me suis ensuite occupée des appels aux API. J'ai commencé par l'API de Google Maps. Pour l'utiliser j'ai dû me créer un compte sur Google Cloud Platform, créer un projet, activer l'API Geocoding et j'ai récupéré une clé me permettant l'accès aux informations de cette API. Puis j'ai créé une fonction qui va récupérer les informations de l'API concernant un lieu donné en paramètre. Pour cela, elle va faire un appel à l'API grâce à l'URL de celle-ci ainsi qu'aux paramètre « address » pour le lieu et « key » pour la clé récupérée précédemment. La fonction va ensuite vérifier que le statut de la réponse de l'API est bon et si c'est le cas elle va retourner l'adresse et l'emplacement du lieu. Sinon elle va retourner « Faux ».

Pour l'API de Wikipédia, je n'ai pas eu besoin de récupérer de clé ou de créer un compte. Mais la récupération des informations de la page d'un lieu donné a dû se faire en deux temps.

D'abord j'ai créé une fonction qui va me trouver, grâce à l'emplacement, une page Wikipédia correspondant au lieu se rapprochant le plus de cet emplacement. Pour cela elle va faire un appel à l'API grâce à l'URL de celle-ci ainsi qu'aux paramètres « format » pour récupérer les informations en format JSON, « action » pour faire une requête à l'API, « list » pour récupérer une liste de page ayant une géolocalisation proche de l'emplacement donné, « gsradius » pour avoir un périmètre de recherche de 1000m autour de l'emplacement et enfin « gscoord » qui contient la latitude et la longitude du lieu. La fonction va ensuite vérifier si la réponse de l'API est bonne et si oui elle va récupérer l'ID de la première page que l'API nous propose. Si non, elle va retourner « Faux ».

Ensuite j'ai créé une deuxième fonction qui va me permettre de récupérer les informations de la page Wikipédia passée en paramètre. Pour cela, elle va faire un appel à l'API grâce à son URL et aux paramètres « format », « action », « prop » pour récupérer les informations de la page, « explaintext » pour récupérer le texte en texte brut et « pageids » pour lui donner l'ID de la page dont on veut récupérer les informations. La fonction va ensuite récupérer les informations de la page. Si la réponse de l'API n'est pas bonne, elle va retourner « Faux ».

Une fois toutes mes fonctions créées, j'ai pu m'intéresser à comment afficher la carte Google Maps sur la page. J'ai donc créé un fichier Javascript contenant une fonction initialisant une carte aux coordonnées données en paramètre, dans un endroit précis de la page. Pour cela j'ai utilisé le code « google.maps.Map » pour créer une carte dans le « div » ayant pour ID « map » dans mon fichier HTML. Et j'y ai ajouté une petite bulle d'information contenant l'adresse et un marqueur pour voir où elle se situe.

Le plus difficile dans ce projet à sûrement été le fait de trouver comment récupérer la question posée dans le formulaire, la traiter et retourner une réponse. La solution que j'ai trouvée a été de créer une nouvelle route, « /Coord », qui contient toutes les informations, les traite et les retourne à la route « /home ». Pour cela, j'ai dû créer une fonction dans mon fichier Javascript qui, grâce à une requête AJAX, va envoyer la question à la route « /Coord ». Cette route va ensuite traiter la question. Elle va d'abord la parser, pour en récupérer uniquement le lieu, puis si la réponse est différente de « Faux », elle va chercher l'emplacement et l'adresse, si la réponse est différente de « Faux », elle va chercher l'ID d'une page Wikipédia ayant une géolocalisation proche de l'emplacement du lieu. Si encore une fois la réponse est différente de « Faux », elle va chercher les informations de la page, et pour finir si la réponse est encore différente de « Faux », alors elle va retourner le lieu, la latitude et la longitude, les informations et l'adresse du lieu. Si une des réponses est fausse, alors elle va retourner une erreur.

Une fois la réponse retournée par la route « /Coord », la fonction Javascript va l'interpréter et retourner un message en fonction de celle-ci. Soit la réponse est bonne et elle retourne un message donnant l'adresse, un message donnant les informations et une carte avec un marqueur pointant sur l'adresse. Soit la réponse possède une erreur et un message s'affiche en fonction de l'erreur trouvée.

Afin de vérifier si mon code se comporte correctement j'ai dû créer un certain nombre de test. J'ai créé plusieurs tests pour chaque fonction, ainsi que pour les deux routes. Pour les fonctions faisant appel à une API et pour les routes, j'ai dû utiliser les « mock » afin de simuler une réponse de l'API ou des fonctions utilisées.

Et pour finir, j'ai mis en ligne mon application avec Heroku, ce qui n'a pas été une mince affaire. Pour cela j'ai du installer Gunicorn et créer un fichier « Procfile » contenant les informations dont Heroku a besoin pour lancer l'application. Soit, que le serveur web utilisé est Gunicorn, que le fichier contenant l'application Flask est « run.py » et la version de Python utilisée.

Mon application étant déjà hébergée sur Git, j'ai dû créer une application sur Heroku et push vers Heroku.