P7 – Créez GrandPy Bot,

le papy-robot

Objectifs et contraintes

L'objectif était de créer un site web sous Flask permettant de simuler une petite discussion avec un grand-père sous un modèle bien précis : L'enfant doit poser une question sur un endroit à trouver dans un langage humain correct et le grand-père doit lui répondre en :
1 – Affichant une carte Google Map pour indiquer l'endroit.
2 – Indiquer l'adresse du lieu et afficher un petit texte tiré de wikipedia sur le lieu indiqué.

Un second objectif (mais pas secondaire) était de créer des tests unitaires cohérents. Le développement devant se faire en mode Test Driven Developpement et les tests devant utiliser les mocks pour tester les API.

Les contraintes principales étaient de ne pas recharger la page entre la question et l'affichage des réponses (→ Utilisation d'Ajax et de Javascript) et d'avoir un site responsive utilisable sur smartphone. Outre la contrainte des tests il est évident que les réponses du grand-père doivent être cohérentes par rapport à la question posée.

Le Front-end

J'ai choisi de faire une interface graphique un peu plus riche que celle demandée dans l'énoncé : Une barre de navigation basique, une page de garde, un about très "boogy-woogy" et le choix de l'API d'affichage de la carte par l'utilisateur. En effet, si pour la recherche des coordonnées je me suis basé sur l'API de Google que j'ai trouvée plus simple à utiliser et plus fiable, j'ai choisi de proposer à l'utilisateur d'afficher la map soit par Goole Map soit par OppenStreetMap. Ce choix se fait via un menu déroulant dans la barre de navigation. A noter que le choix de l'un ou l'autre impose un rechargement de la page afin de charger les bonnes libs (le fait de charger les 2 systématiquement ralentissait trop la navigation à mon goût) et utilise pour cela Ajax.
Les 3 grandes parties de la page (header, body et footer) sont 3 templates différents réunis et modifiés grâce aux systèmes d'extends et de blocks de Jinja2 (moteur de templates de Flask).
Pour ce que est de la partie script j'ai opté pour jQuery, le côté responsive étant géré avec Bootstrap.
En passant en mode "Smartphone", le menu About ainsi que la date et l'heure affichée à gauche disparaissent. Il en est de même pour le footer. Les petits personnages passent systématiquement au-dessus de leurs 'input' respectifs et la police est réduite.

Le Back-end

Le backend est bien entendu géré par Flask et python. Il se découpe en 3 grandes parties :
- Les templates : Comme dit plus haut le système des templates est basé sur les extends et le système de blocks de Jinja2. L'utilisation des variables permet de ne pas charger le JS de google map si l'utilisateur souhaite avoir une map OSM et inversement.
- Le "main" : Dans le répertoire papybot du projet les fichiers main.py et __init__.py permettent juste l'initialisation du

projet. Les routes sont-elles dans le fichier “views.py”. Ce fichier contient aussi le traitement des retours Ajax que ce soit pour le traitement de la question posée aou pour le changement de map. Les données renvoyées au front-end sont au format JSON

- Les classes : Dans le répertoire utils : query.py va contenir la Classe Query qui va gérer la question posée (parser). Le fichier api.py va quant à lui contenir 3 classes : 1 classe Api (classe mère) et 2 classes ApiGoogle et ApiWiki qui vont gérer les 2 API.

Algorithmes

- Parser : La classe parser prend pour paramètre la question telle qu’elle est posée. Nous allons dans 1 premier temps, supprimer la ponctuation, les accents et les éventuels caractères “bizarres”. Nous disposons de 2 fichiers json contenant pour l’un une base de stopwords venant d’Internet et pour l’autre une liste de stopwords personnels. La classe retourne la liste des mots qui ne sont dans aucune des 2 listes.

- API : Une classe mère API qui va contenir dans son init la récupération de la query parsée.

    - La class Gooapi (Api Google) : va interroger Google via son api et la fonction “findplacefromtext” le retour (si tout se passe bien) est parsé et seuls les éléments intéressants (nom, latitude, longitude…) sont retournés sous forme de dictionnaire

    - La class WikiApi (Api wiki) : Prend en paramètre le nom renvoyé par Google afin d’avoir une recherche plus fiable. Wiki est interrogé en 2 temps. Dans un 1er temps, l’utilisation de la fonction “list=search” permet de trouver un pageid. C’est ce pageid qui est utilisé pour retrouver un extract des 3 premières phrase de la page qui nous intéresse et qui permet aussi de construire l’URL mise dans la réponse. Les éléments sont là aussi retournés sous forme de dictionnaire

Tests

Les tests ont été effectués avec la librairie Pytest. Les 2 premiers tests permettent de tester si la page d’index renvoi bien un code 200 et si une page quelconque renvoi bien un code 404. Le test suivant consiste à tester le parser en instanciant la classe avec une phrase définie et en validant que la réponse soit bien celle souhaitée. Les tests suivants vont porter sur les 2 api et sur le même principe : La fonction requests.get est mockée grâce à la fixture monkeypatch de pytest.

Déploiement et liens

Le déploiement a été effectué sur la plateforme Heroku.
Lien du site : https://jmlm-p7papybot.herokuapp.com/