Projet 7 : GrandPy Robot - Code source - Deploiement sur Heroku

https://github.com/horlas/OC_Project7 https://papyrobotag.herokuapp.com/

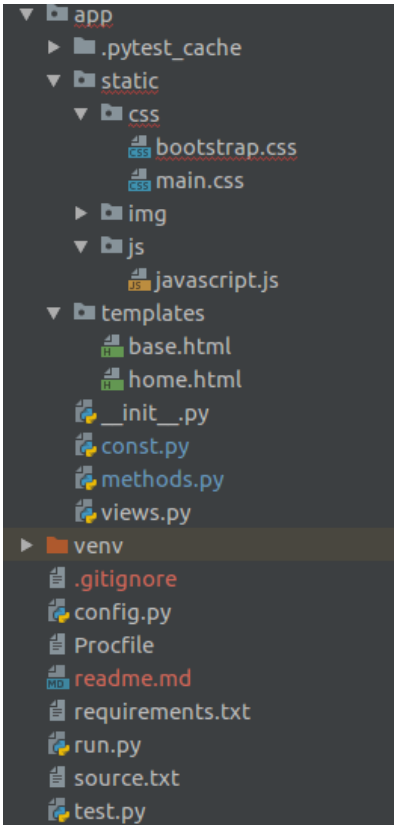
Généralités

Ce projet a été l'occasion pour nous d'apprendre plusieurs technologies : l'utilisation d'un Frame Word python (Flask) , l'utilisation de Javascript et de Jquery (appel Ajax et fonctions gérant le Front End), l'utilisation des API (Google map , geocoding et wikipédia) , l'utilisation de HTML et CSS et Bootstrap, l'utlisation du TDD , tests unitaires et pytest , et le déploiement sur Heroku . La somme de toutes ces nouvelles connaissances nous mènent naturellement à faire une synthèse dans ce document des différents fichiers de script qui composent notre programme en présentant au lecteur une difficulté levée et la solution que nous y avons apporté.

Choix technologiques

L'application est développée en python 3.5 , en utilisant le micro Framework Flask . Les tests unitaires utilisent le module Pytest. Le Frontend est écrit en HTML /CSS/Javascript . Javascripts est importé depuis sa source CDN , ainsi que son Framework Jquery pour faire l'appel AJAX. La version responsive utilise Bootstrap (importé depuis le CDN également) . Un thème Bootstrap est chargé. Nous avons utilisé l'IDE Pycharm. Concernant les API : nous avons utilisé Flask-Google Geocoding pour obtenir les coordonnées géographiques et l'adresse du lieu , l'API Wikipédia pour l'extrait du texte concernant le lieu . La carte est affichée grâce à JavaScript (API JavaScript Google Map) dans le front en récupérant les coordonnées du lieu. Pour les versions voir : requirements.txt :

Arborescence du projet



- **test.py :** contient les tests unitaires. Sont testés les méthodes de classe et fonctions contenues dans le fichier methods.py.
- **Run.py :** lanceur du programme
- **Procfile** : chargement du serveur gunicorn pour déploiement sous Heroku.
- **views.py** : Contenu de Flask , routes du programme : deux routes une qui affiche la page à l'origine , l'autre qui retourne le programme lors de la saisie utilisateur . La mise à jour de la page HTML se fait avec l'appel AJAX(JavaScript.js)
- **methods.py** : fichier contenant la classe ParsePlace et ses methodes , instanciant et traitant la saisie utilisateur . Ainsi que deux fonctions traitant la partie Wikipédia.
- **Const.py** : fichier contenant les constantes, essentiellement les phrases du robot..
- **Static/templates** : base.html permet d'intégrer Flask-bootstrap , les feuilles de styles et les scripts JS. Base.html est le claque par défaut commun à toutes les pages. Home.html est le contenu de la page.
- **Static/css :** feuilles de styles: bootstrap.css theme chargé, main.css : feuille de style personnalisée.
- **Static/js/javascript.js** : fichier regroupant tout le Javascript du programme : l'appel AJAX qui se fait lors de l'envoi par l'utilisateur du formulaire, et les fonctions JavaScript nécessaires au fonctionnement du frontend.

Fonctionnement du Backend

Flask affiche home.html dans lequel apparaît une zone de saisie (formulaire) .
L'utilisateur appuie sur entrer ou clique sur le bouton, à sa convenance.
Cette action a pour effet de déclencher l'appel AJAX. `/$.getJSON(--url, --data, --func).`
L'appel sur l'url est celle contenu dans `@app.route('/_add_datas')`

Elle fait appel à la fonction add_datas :

- Cette fonction recupère la saisie utilisateur `request.args.get(« place »),` Et instancie cette saisie en objet (place) de la classe ParsePlace (contenue dans methods.py) :
- `clean_entry(self)` L'objet « place « est nettoyé : on fait appel pour cela à la librairie python stop_words enrichie de quelques mots et à des regex(expressions regulières) permettant de ne garder que les mots essentiels de la saisie : exemple : user_input « Je veux aller à la piscine olympique de Montpellier » donnera « piscine olympique Montpellier » après traitement.
- `place_for_ggapp(self)` Dans l'optique d'obtenir le maximum de résultat de la part de l'API Google geocoding et de l'API Wikipedia, on transforme l'objet une nouvelle fois en ajoutant des « + » entre les mots . Ex : « piscine olympique Montpellier » devient « piscine+olympique+Montpellier »
- `some_words_about(place)` est la fonction qui fait appel à l'API Wikipedia pour extraire trois phrases du premier article trouvé suite à la soumission de l'objet place. Une première requête est faite en appelant la liste des résultats possibles . Le premier résultat est sélectionné (c'est celui qui a le plus de chance de répondre quelque chose de favorable. C'est avec le titre du premier résultat que la recherche est faite . Cette technique nous permet de répondre quasiment tout le temps quelque chose. En cas de non réponse (par exemple mauvaise saisie de la part de l'utilisateur) , le Robot invite l'utilisateur à vérifier son orthographe et lui parle de sa première voiture.
- `geocoding(self)` Appel à l'APi google geocoding est mis en place en utilisant Python Client for Google Maps Services. L'objet gmaps est instancié en tant qu'objet de la classe googlemaps.Client et prend en paramètre la clé d'API . Ceci permet de faire appel à la methode geocode() et de recupèrer sur iteration de la réponse la latitude , la longitude ainsi que l'adresse du lieu. Nous avons levé l'exception IndexError en cas de mauvaise saisie par l'utilisateur.
- `format_add(self)` Formatage du retour de l'adresse en ajoutant les phrases de GrandPy (constants.py)

Traitement des données en retour par la fonction AJAX (fichier javascript.js)

Les données sont retournées à AJAX sous format json (methode jsonify de flask). Nous retournons les coordonnées géographiques dans le template pour pouvoir afficher la carte grace à l'API Javascript google map. Cependant les coordonnées ne seront pas affichées (display=none des div #lat et #lng) . Le premier message de Grandpy contient le texte de l'adresse et il est ajouté à la zone de réponse.
Le container de la map est initialisé.
L'appel à la fonction InitMap, creation de la carte en utilisant l'API JavaScript Google Map et en récupérant les coordonnées géographiques et en insérant la carte dans le container précédemment créé.
Et enfin le troisième message de Grandpy contient le texte renvoyé.

Affichage de la question de l'utilisateur

Lors de l'appui sur la touche ENTER ou sur le click du bouton d'envoi , le message utilisateur vient s'afficher dans la zone dédiée. Cet affichage est géré en JavaScript par la fonction getMes() : création d'une message utilisateur.
Un click sur la zone de saisie vide cette zone .

Réalisation des tests et utilisation de Pytest

Les deux premiers test s'assurent que les API Google et Wikipedia sont disponibles et retournent un status = 200 . Pour cela nous créons deux fonctions qui se chargent de requêter l'api et de renvoyer le status_code de la réponse. Les tests sont verts si ce code est 200.
Le test suivant simule le module geocode du client python pour GoogleMAp en le patchant , c'est à dire sans avoir la connexion à l'API.
Le test suivant simule le module request de la fonction call_wiki (appel à l'API Wikipédia) en utilisant aussi un mock.
Suivent les tests des méthodes de la class ParsePlace (précedemment décrite) : traitement du texte , geocoding (avec appel à l'API cette fois ci), formatage de la réponse du Robot.
Et enfin le dernier test couvre la fonction d'appel à Wikipedia et d'extraction d'un text sans mock cette fois ci .