OPENCLASSROOMS

Etudiant: Ly Mickael | Mentor Validateur: Aurélien Massé

Soutenance du projet 7: Créez Grandpy Bot, le papy-robot

Lien code source : https://github.com/Goro-Majima/p7-Cr-ez-GrandPy-Bot-le-papy-robot

Pour le projet 7 développé en tdd, l'objectif est de créer une application web en HTML/CSS et Javascript. Il s'agit d'un robot qui renvoie l'adresse, la carte google map et un récit sur un lieu demandé. Des fonctionnalités comme AJAX, les API de Google Maps et Media Wiki sont à respecter dans le cahier des charges. La page web doit aussi respecter une charte graphique et autres contraintes.

Le projet est en programmation orienté objet et la méthode suivie est la méthode agile avec l'outil de gestion de projet Trello. Environnement virtuel créé et suivi des recommandations Pep8 (extension black, pylint)

Le challenge de ce projet est d'apprendre à nettoyer une donnée d'entrée et utiliser des Api avec un serveur de développement simple à utiliser. Il faut apprendre à utiliser le framework FLASK, l'architecture MVT et assembler le tout avec les fonctionnalités développées.

TESTS

Le parser est développé en « TDD » pour créer la fonction par rapport à des réponses prédéfinies. Ainsi, les tests vérifient que la fonction va filtrer la question jusqu'à ne conserver que les mots clés grâce à la liste « stopwords » qui va supprimer les mots contenus dans la question. Les expressions régulières sont utilisées pour éliminer les ponctuations. Les mocks sont utilisés pour tester le comportement des fonctions des classes Googlemap et Mediawiki.

PARSER ET APIS

S'inscrire sur la plateforme google developer afin d'obtenir une clé API pour google maps.

Le(s) mot(s) clé(s) généré(s) par le parser est l'argument de la fonction « http_results » et sera utilisé comme paramètre d'adresse lors de l'appel de l'api. Des datas en json seront parcourus afin d'extraire uniquement les coordonnées gps(latitude et longitude) ainsi que l'adresse postale. Ces coordonnées serviront ensuite d'arguments pour le paramètre « gscoord » lors de l'appel à l'api de mediawiki. Une fois le lieu reconnu par l'api, un deuxième appel est effectué avec comme paramètre cette fois le lieu afin d'extraire les deux premières phrases de l'histoire et l'url du lien wikipedia. Grosse recherche demandée pour trouver les bons paramètres à utiliser pour les extraire.

BACK-END

S'assurer qu'un package contient bien le fichier « __init__ .py » afin que le serveur reconaisse le dossier. Selon l'architecture MVT, deux vues sont nécessaires : home/ pour la page d'accueil, _api/ pour envoyer les résultats désirés sous format JSON. La méthode POST est utilisée pour récupérer la question qui vient d'un formulaire sur la page home. Ces résultats sont obtenus à partir de la question récupérée sous forme d'argument dans la fonction « process_question »(C'est le « modèle» qui va manipuler les données) qui orchestre le parser, les api googlemap et mediawiki. Un dialogue personnalisé de papybot est implémenté par

rapport aux retours des try/except. On démarre le serveur flask avec la commande « python run.py » pour voir le résultat d'une requête.

FRONT-END

Côté HTML : Les fichiers html sont placés dans le dossier « templates » tandis que les fichiers css et js sont placés dans un dossier « static ».

Utilisation de le meta «viewport » pour afficher l'application en responsive design.

Un formulaire est nécessaire pour récupérer l'input de l'utilisateur. Suite à son envoi, un spinner va simuler un état de recherche dans un loader.

Côté JS : fonction initMap pour afficher la map, fonction ajaxPost pour réponse d'un appel asynchrone en ajax. Création et nommage des div en fonction du résultat des données json . Découpage des données json afin de les répartir dans les div créées.

DEPLOIEMENT

Mise en ligne sur Heroku via la console et git. Installation du serveur «Gunicorn » car flask est un serveur de développment. Création du fichier « Procfile » (ne pas oublier la majuscule) qui donne les instructions au démarrage de l'applicaiton, requirements.txt pour les librairies à installer.

Difficultés rencontrées

- Problème d'organisation du projet en package et connexion entre modules.
- Incompréhension de l'utilisation des mocks pour les API.
- Récupération difficile des données de l'api Mediawiki dûe aux paramètres.
- Problème pour exploiter l'input d'un utilisateur car le formulaire n'était pas reconnu dans le module home.js.
- Problème de communication serveur après mise en ligne.

Solutions aux difficultés

- Découpage en fichiers de la partie tests, logique applicative, du code pour la configuration des routes, le front-end.
- Lecture approfondie sur https://www.mediawiki.org/wiki/Extension:TextExtracts#Caveats pour savoir quels paramètres utiliser pour obtenir les infos désirées.
- En javascript, nouvelle variable qui contient l'appel d'un formulaire de données vide auquel on rajoute ce que l'utilisateur a saisi. Cette variable sera l'argument data de la fonction AjaxPost.
- Tuto en ligne et cours pour une mise en ligne réussie : Import de FLASK_CORS et Cross origin pour permettre à AJAX de récupérer les données venant d'autres ressources (CORS, CORB : mesure de sécurité qui empêche l'accès)
- Enfin, l'aide des mentors est la bienvenue sur ce projet qui demandait le suivi de nombreux cours et la compréhension de fonctionnalités différentes.