Petit historique du projet

Tableau Trello : <https://trello.com/b/5UZfcxBB/projet-7-grandpy>

Repo Github : <https://github.com/GMBAMorera/Projet_7>

Site : <https://ocgrandpy.herokuapp.com/>

# Mise en place

La première étape de travail auquel je me suis employé a été de découper le développement du projet en différentes user stories, afin de pouvoir estimer la progression du projet et de répartir le travail. En reprenant la progression proposée dans l’intitulé du projet, je l’ai découpé en 5 user stories :

* User story 1 : l’utilisateur se rend sur l’adresse du site, en local. La mise en page du site est en place, le formulaire de question est présent et le bouton du formulaire est inactivé.
* User Story 2 : l’utilisateur se rend sur l’adresse du site, en local. Il peut poser une question, scriptée, et le code javascript lui fournira une réponse scriptée elle aussi.
* User Story 3 : l’utilisateur se rend sur l’adresse du site, en local. Il peut maintenant poser n’importe quelle question, le site enverra une requête au serveur et celui-ci parsera la question, fera des recherches sur les API Google Maps et de Wiki Media et renverra la réponse. Le site publie ensuite la réponse.
* User Story 4 : comme la 3, mais la réponse du site est mise en forme et introduit ses réponses avec des phrases « façon grand-père ». Si l’utilisateur rentre une question dans le but de faire fonctionner du code html, il est débouté (risque XSS). De plus, la couverture de test de l’application est d’au moins 80%.
* User Story 5 : l’utilisateur rentre l’adresse du site (<https://ocgrandpy.herokuapp.com/>) et celui-ci fonctionne comme prévu, notamment sur la responsivité.

Une fois les grandes lignes mises en place, j’ai commencé les cours liés aux langages du web (HTML, CSS et JavaScript pour le web).

# Développement

## User Story 1

J’ai commencé par faire un schéma précis du site voulu, puis j’ai commencé à codé. Ma première vraie difficulté est arrivée quand j’ai cherché à obliger le site à prendre la totalité de la page internet. J’ai testé de nombreuses solutions, ce qui m’a permis de me rendre compte de la complexité et de la subtilité des instructions HTML.

En cherchant à rendre ma page responsive, j’ai implémenté Bootstrap et revu en profondeur la construction de ma page HTML. Ayant déjà codé en HTML pour de petits projets de travail, j’avais commencé à coder ma page sans m’y intéressé, ce que je considère a posteriori comme une erreur. Commencer à coder avec Bootstrap immédiatement m’aurait fait gagner du temps.

Le bout de code avec Flask a été rapide à implémenter, notamment car j’avais déjà utilisé le mini-framework auparavant. Seule la structure en classe aura posé un peu problème.

## US 2

La deuxième partie du projet s’est passé rapidement et sans accroc. J’avais déjà une petite expérience de (Node) JS et la user story tournait surtout autour de l’ajout de la réponse dans un bloc html. Avec l’expérience, les user story 1 et 2 aurait peut-être dû être fusionnées.

## US 3

La user story 3 contenait la majorité des implémentations en python du projet, à savoir le parser et les requêtes vers les API google Maps et WikiMedia.

Concernant le parser, j’ai essayé de mettre en place la méthode Test-Driven Development pour le construire, cette partie du code étant la plus « algorithmique » du projet. Le résultat est mitigé. La difficulté provient de ce que le point de départ du développement n’est pas une structure bien nette des fonctions à construire, mais une idée d’algorithme. L’idée initiale est de construire un algorithme qui découpe une phrase en expressions délimitées par les stop words de la banque fournie dans l’énoncé du projet, puis de conserver les trois dernières expressions pour pouvoir rechercher d’éventuels mots composés.

En particulier, la conception de l’algorithme a montré de nombreux effets de bords difficiles à anticiper avent de tester des exemples concrets. Les fonctions devenaient de fait de plus en plus complexes sans que des tests aient été écris en amont pour vérifier les points qui étaient soulevés. En conclusion de cet essai de code en TDD, je trouve que c’est une manière de programmer utile pour préciser sa pensée, mais qui nécessite une certaine habitude de la programmation et aussi d’avoir soulevé la plupart des risques avant de coder, sous peine de perdre le fil.

L’implémentation des requêtes google map est wikimedia n’a pas posé de problèmes particulier. Le cœur des requêtes vers les API de google maps est un bout de code désigné comme bonne pratique par la documentation de Google, redécoupé pour rendre les tests unitaires plus ciblés et pour l’adapter à plusieurs requêtes vers des API différentes. La requête vers wikimedia a été simplifiée grâce à une librairie python adaptée.

## US 4

La User Story 4 n’a pas posé de soucis particuliers. Elle se compose de finitions disparates, mais sans difficultés particulières.

Le seul soucis aura été de trouver un moyen d’implémenter un mock dans un setup pour les méthodes, les deux n’étant pas initialement compatible… La découverte du déorateur transformant une fonction en fixture et du keywor « autouse » m’a permis de trouver la bonne architecture.

# Déploiement (US 5)

Pour la mise en production, un seul souci aura émergé, car Heroku veut pouvoir fournir l’application sur le port de son choix, tandis que Flask en choisi lui aussi un d’office. Le problème m’a bloqué un certain temps, surtout parce qu’il était difficile de comprendre ce qui se passait. Cela m’aura permis de faire le tour des recommandations de Flask et de Heroku et d’implémenter un certain nombre de bonnes pratiques, comme le serveur waitress, recommendé par Flask.