1. **Démarche de conception**

Pour bien débuter la conception du projet, nous avons commencé à lister les différentes fonctionnalités demandées. Nous avons écarté neuf propriétés demandées :

|  |  |
| --- | --- |
| **1** | Utilisation de Flask |
| **2** | Utilisation de Ajax |
| **3** | Création d’une page web avec boopstrap |
| **4** | Gestion d’un formulaire |
| **5** | Gestion des demandes de l’utilisateur |
| **6** | Utilisation des Api google et Media Wiki |
| **7** | Mise en page des réponses du bot |
| **8** | Mise en ligne sur Heroku |

Par la suite, nous avons ouvert le projet dans un répository git pour y mettre les différentes ressources du projet et effectuer les différentes sauvegardes dont nous aurons besoin. Ce repository se trouve via [ce lien](https://github.com/ZasshuNeko/OC-Projet-PapyBot).

Pour ce projet, nous avons créé un tableau trello pour la partie organisation agile du projet que vous pouvez consulter sur [ce lien](https://trello.com/b/xy7gLlWE/oc-projet-7)

1. **Etude de l’algorithme**

Nous avons pour ce projet :

Un fichier gérant la page web, créé avec boopstrape, nous avons mis en place un entête avec un logo, un titre. Puis un formulaire avec un corps gérant la discussion avec le bot. Nous avons mis un champ input permettant à l’utilisateur de poser des questions. En bas de page nous avons placé un lien menant au repository git du projet.

Un fichier gérant la partie Ajax, ce fichier permet d’envoyer la question de l’utilisateur vers le code en python. Puis une fois la demande traitée, la réponse est renvoyé vers cette partie pour mettre en place la réponse sans avoir à recharger la page. C’est ce code qui initialise la carte et met en place les markers de positions.

Pour la partie du code en python, ils sont scindés en cinq fichiers distincts.

Le « main » est appelée pour lancer l’application, la route principale de flask génère le template principale.

Une seconde route est mise en place, cette dernière est appelé par jquery lors de l’utilisation de l’input dans le formulaire du template. Quand cette demande est reçue il est envoyé vers le fichier « traitement » qui va gérer la demande, en utilisant la bibliothèque « nltk » nous allons traiter la phrase pour retirer les termes important de la question et les envoyer vers les différentes api.

Chaque api est géré de manière distinct dans deux fichiers, l’un nommé « api\_google » qui va gérer l’envoi du terme important et la réponse de l’api google map. A ce moment nous récupérons les informations utiles pour notre bot, l’adresse et la/les postions de chaque endroit trouvé. Tout cela est concaténé dans un tableau puis renvoyé.

Pour l’API wiki, soit une réponse est trouvé par google et alors nous récupérons deux informations, celle du terme recherché et celle lié à l’adresse récupéré par google map, soit aucune réponse de google et à ce moment-là nous ne cherchons uniquement le terme important pour donner les informations obtenues par l’api wiki.

Les données obtenues des différentes apis sont alors renvoyé vers le fichier « fichierjson » qui va permettre de créer une réponse au format json avec toutes les données importantes pour intégrer les réponses à la template. Une fois les données json effectué, ces dernières sont renvoyées vers le code ajax. A ce moment-là, le code est mis en place dans le template.

A chacune des demandes de l’utilisateur, ce schéma est utilisé, l’application lance la route principale qui génère la page. L’utilisateur demande quelques choses, jquery récupère la demande et la fournie à la seconde route qui va par la suite fournir au fichier qui gère le traitement de la demande, puis aux différentes api, puis au fichier gérant la création du json qui renvois alors au jquery qui fournis la réponse au template.

Pour cela nous avons établi un schéma de travail :

1. **Etablir un plan de travail**
   1. Méthode agile
   2. Déterminer les demandes et les contraintes du projet
   3. Mise en place de Flask
   4. Mise en place de Ajax
   5. Etude des API google et wiki
   6. Etude des résultats api obtenus
   7. Mise en place de la réponse par jquery
2. **Création du programme**
   1. Etablir les différentes classes dont aura besoin le programme
   2. Etablir le fichier jquery
   3. Etablir le fichier template
3. **Création des requêtes API**
4. **Finaliser la logique utilisateur**
   1. Lier les différentes classes
   2. Permettre une interaction utilisateurs dynamique
   3. Mise en ligne du site sur Heroku
5. **Difficultés & solutions**

Nous pouvons scinder les difficultés en trois parties :

Une difficulté de mise en place dût à la méconnaissance des différentes api que nous avons dut étudier.

Mise en place de Flask avec la compréhension des routes et des liens à mettre en place avec le template.

Une fois ces deux difficultés écartés et que nous avons réussis à obtenir une vision globale du projet et ainsi une compréhension de chaque partie qui nous permettra de mettre en place les différentes demandes du projet. La nouvelle problématique est de liéer chaque partie pour que chaque puisse communiquer entre elle. Cette partie fut traitée de manière logique respectant les demandes du projet. Nous avons d’abord mit en place les routes pour Flask, mit en place le template et ainsi tester les routes pour constater si la page web est bien chargé.

Une fois cette mise en place faite, nous créer le fichier jquery qui devra récupérer le input du template puis devra créer un appel « POST » vers le second chemin Flask. Cette première difficulté fut de mettre en place un lien entre la template, jquery et python. Cette problématique fut solutionnée par l’apprentissage et la recherche de différente source d’information pour comprendre au mieux comment cela foncitonne.

La programmation des différents fichiers python ne fut pas problématique, car cela reprend l’apprentissage déjà vu lors des anciens projets. Le renvois des données de python vers ajax pour la mise en place d’une mise à jour du DOM sans actualisation fut un peu plus compliqué, une recherche plus approfondie et un retour sur les api servirent à combler les lacunes qui me manquaient.