Afin de planifier notre projet ***Planning game*** nous avons identifié les différents acteurs avec UML unified modeling language. Pour ce blog nous avons identifié deux acteurs: le lecteur et le propriétaire du site que l'on considère comme modérateur du forum et celui qui remplit la database. Le lecteur est en interaction avec le site web et le propriétaire est en interaction avec la base de données.

Nous avons identifié les différentes fonctionnalités des acteurs et avons définit les cas d'utilisation par les diagrammes mais aussi par les description textuelles des cas d'utilisation. Nous avons aussi défini la navigation sur le site web (comment l'utilisateur passe t-il de page en page? les retours ? ect) qui se présente comme étant un menu. Nous nous sommes focalisé sur les fonctionnalités principales et à la fin avons codé des fonctionnalités moins importantes que nous appelons "fonctionnalités annexes".

Ensuite nous avons choisi le serveur sur lequel déployer notre application et avons fait nos schémas de base de donnée et son type (relationnel sous POSTGRESQL).

Nous avons ensuite pensé nos diagrammes de classes. A noter que nous avons implémenté de nouvelles fonctionnalités au fur et à mesure de l'avancement du projet comme le tchat à la fin. Toutes ces étapes ont été timé par un tableau trello (à faire, réalisation, problème, fin, à travailler).

Nous n'avons pas eu à employer de ***métaphore*** car un seul développeur (moi) travaillé dessus. Cependant, si j'avais eu une équipe de développeurs et de non-informaticiens (marketing par exemple) j'aurai employé la fonctionnalité de la librairie par exemple au lieu de dire cv2 j'aurai dit: l'outils pour les images, js: le langage des pages, css: le langage design, POO: dév légo ect.

Dit récemment nous avons au fur et à mesure implémenté sur nos diagrammes de classes de nouvelles fonctionnalités après avoir validé une phase de planification, réalisation et de test (ici le push heroku) pour revenir à la première étape si nécessaire ou après une nouvelle idée. De plus une fois par semaine nous délivrons nos avancé à notre mentor ***Frequent releases*** qui avait pour but de validation (et/ou modification) mais aussi de motivation.

Pour le design, nous avons codé l'essentiel (fond, bouton, js) puis après validations avons fait les href, le js plus poussé (à but esthétique) , fait le css (responsive, hover ect). Par cela, une page pouvait rapidement être modifiée.

Je n'ai pas codé en binôme. Enfin presque ! J'ai reçu de l'aide sur le discord, mais aussi sur les forum stack overflow et developpez.com en plus de l'aide de mon Mentor sous l'angle de la conception. ***Code en binôme***

L'aménagement des horaires était soutenable et avec des temps de pause. Cela permettait de ne pas surcharger le développeur (moi) mais aussi de le garder dans un environnement de travail soutenable. ***Forty-hour-week***

Sur ce projet je ne pouvais pas avoir de retour client. Cependant j'ai fait essayer à quelques proches certaines applications quand ils avaient le temps et avaient de bons retours. ***Cadre opérationnel***

Afin de garder une structure il faut utiliser ***un standard de codage*** dans le code les variables que j'utilise se suivent. Par exemple, si j'utilise une variable "liste" dans une fonction et que cette fonction nécessite une autre fonction, j'utilise une autre variable le même nom de variable. Les fichiers de configuration dispose tout du même nom: CONFIG.py, et les fichiers de database sont tous nommés database.py ect. Le design de chaque template est à peu près le même et respect: fond d'écran, texte, menu, bas de page et fonctionnalité. Chaque page respect la convention suivante: menu latéral, navebarre(qui est une page menu), head et bottom (via un gabarit include) sous un même thème de template. Le framework utilisé est Django.

A chaque finition nous pushons la fonctionnalité sur le serveur qui nous sert d'outils de test. En effet, s'il y a une erreur le mode de debug nous prévient. De cela nous pouvons utiliser une méthode agile: Concevoir, Développer, Tester. Nous validons aussi quelques tests à l'aide de Django et de Pytest pour nous assurer par exemple du statut 200 de la page ou de la validité des url ainsi que des api. ***Intégration continue***

Nous avons bien sûr fait plusieurs étapes de ***refactoring.*** Factorisation du code en plusieurs fonctions, modification du code en cas de non-lisibilité, déplacement de fonction dans différents fichiers en cas de surcharge, création de dossiers en cas de débordement (une application qui a une fonctionnalité annexe par exemple).

Nous avons donc respecté plus ou moins l'xp programming car nous n'avions pas de client et n'étions pas dans une équipe. Nous avons "exploré" la futur application et/ou fonctionnalité, avons itéré puis mis en production. En cas de dysfonctionnement d'une partie du système, le système n'était pas mis en péril et avons pu réparer l'erreur ciblée.