Application Planning Poker en ligne

1. Présentation Générale :

a. Contexte :

Le scrum poker, est une technique ludique et collaborative utilisée par les équipes Scrum pour estimer la complexité et l'effort nécessaire à des tâches de développement. Cette technique repose sur l'utilisation de cartes pour permettre aux membres de l'équipe d'exprimer leurs estimations de manière anonyme et sans influence. Cependant, la réalisation de sessions de planning poker peut être un défi, en particulier pour des équipes distribuées géographiquement.

Dans ce contexte, le développement d'une application de planning poker accessible en ligne devient pertinent pour permettre à des équipes travaillant à distance ou dans des bureaux différents, de mettre en application cette technique d'estimation de l'effort tout en étant séparés physiquement, sans monopoliser une salle de réunion et sans perte de temps.

b. Objectifs :

Expérimentation de la gestion d'un projet informatique de sa conception à sa réalisation, en respectant le triangle Délai-Coût-Qualité.

Développement de compétences techniques et maîtrise des Outils de Développement. Cet objectif vise à renforcer mes compétences techniques, notamment dans le domaine du développement logiciel. Il s'agit de gagner en expertise avec une variété d'outils et de technologies, en mettant l'accent sur ceux que j'ai choisis pour ce projet.

Gestion équilibrée du temps et allocation des ressources. La gestion judicieuse et équilibrée du temps, une ressource précieuse, notamment en raison de la simultanéité d'autres projets en cours. Il s'agit d'assurer une répartition équitable et efficace du temps alloué à chaque tâche, en mettant en œuvre une planification stratégique et une organisation méticuleuse. Cette approche vise à maximiser la productivité tout en maintenant un haut niveau de qualité sur l'ensemble des projets et examens.

2. Analyse des besoins :

a. Besoins utilisateur :

Collaboration à Distance : Permettre à une équipe répartie géographiquement de réaliser les tâches d'estimation de l'effort et d'échanger.

Simplicité d'Utilisation : Offrir une interface utilisateur intuitive et facile en prise en main , afin que le focus de l'usager soit sur la réalisation de la tâche d'estimation.

Suivi des Résultats : l'utilisateur accède sur le même outil à ses anciennes estimations, ceci peut s'avérer utile pour comparer, convaincre éventuellement un collègue en faisant des analogies avec des fonctionnalités déjà passées en revue.

b. Exigences fonctionnelles :

Les axes critiques à inclure dans le livrable sont :

Authentification et Gestion des Comptes : Les utilisateurs doivent pouvoir se connecter à l'application avec leurs informations d'identification.

Création et Gestion de Parties : Les utilisateurs doivent pouvoir créer une nouvelle partie de planning poker en ligne. Le créateur de la partie doit pouvoir spécifier des caractéristiques à la partie.

Sélection de Cartes : Les participants à la partie doivent avoir accès à un jeu de cartes virtuelles pour attribuer des points d'effort aux tâches. Les utilisateurs doivent pouvoir sélectionner et soumettre leur estimation pour chaque tâche.

Affichage en Temps Réel : Les estimations soumises par les participants doivent être affichées en temps réel pour tous les membres de la partie.
L'application doit afficher le consensus ou la moyenne des estimations lorsque tous les participants ont voté.

Sauvegarde et Historique des résultats : L'application doit enregistrer l'historique des estimations pour chaque tâche et chaque partie. Elle doit également permettre de reprendre une partie entamée à partir des votes partiels effectués.

Choix de plusieurs modes de jeu : La création de la partie doit inclure un choix entre plusieurs modes de jeux. Un mode strict et un mode statistique au moins.

c. Exigences non fonctionnelles :

Extensibilité : L'architecture de l'application doit permettre une extensibilité facile pour prendre en charge de nouvelles fonctionnalités. Et vu dans l'autre sens, la non-réalisation d'un module prévu, ne doit pas affecter le résultat d'un module précédent. Ceci est particulièrement important dans un contexte, ou un blocage potentiel était possible et pouvait avoir comme conséquence une livraison partielle des fonctionnalités imposées.

Compatibilité : L'application doit être compatible avec une gamme de navigateurs Web modernes (Google Chrome, Mozilla Firefox, Safari, etc.).
L'application doit être compatible avec différentes plates-formes, y compris les systèmes d'exploitation Apple et l'environnement linux.
Dans le développement éviter les outils rares, les désuets, les gratuits mais très spécifiques...
Le professeur doit pouvoir disposer d'un fichier « requirements » lui permettant de mettre à jour son environnement sans difficultés.

Évolutivité de la Base de Données : La base de données sous-jacente doit être facilement exportable et compatible, s'il y a lieu de la déplacer sur un serveur distant ou s'il faut l'agrandir.

3. Conception et Architecture :

a. Architecture générale :

En choisissant la librairie python Django pour le développement, opter pour le modèle MVT (Modèle-Template-Vue) semblait incontournable. MVT est une variante plus moderne du bien connu modèle MVC. Le modèle MVT se distingue par sa structure en trois composants clés :

- le Modèle, qui gère la structure de données et la logique métier.

- le Template, qui correspond à la couche de présentation et détermine comment les données sont affichées à l'utilisateur.
- la Vue, qui fait le lien entre les modèles et les templates, agissant comme un contrôleur pour gérer la logique de traitement des requêtes utilisateur.

L'un des principaux avantages du modèle **MVT** est sa forte cohérence avec la philosophie de Django, « le framework pour les perfectionnistes avec des délais ». Cette approche favorise un développement rapide et efficace, avec une séparation claire des préoccupations, facilitant ainsi la maintenance et l'évolutivité du code. De plus, le modèle MVT encourage une architecture propre et une réutilisation du code, ce qui est essentiel pour un projet agile comme l'application de planning poker.

Cependant, le modèle MVT peut présenter des défis, notamment pour les développeurs habitués au modèle MVC traditionnel. La distinction entre « Vue » et « Template » dans MVT peut initialement sembler contre-intuitive, car la « Vue » dans MVT joue un rôle plus proche de celui d'un contrôleur dans MVC, et le « Template » est plus similaire à la vue dans MVC.

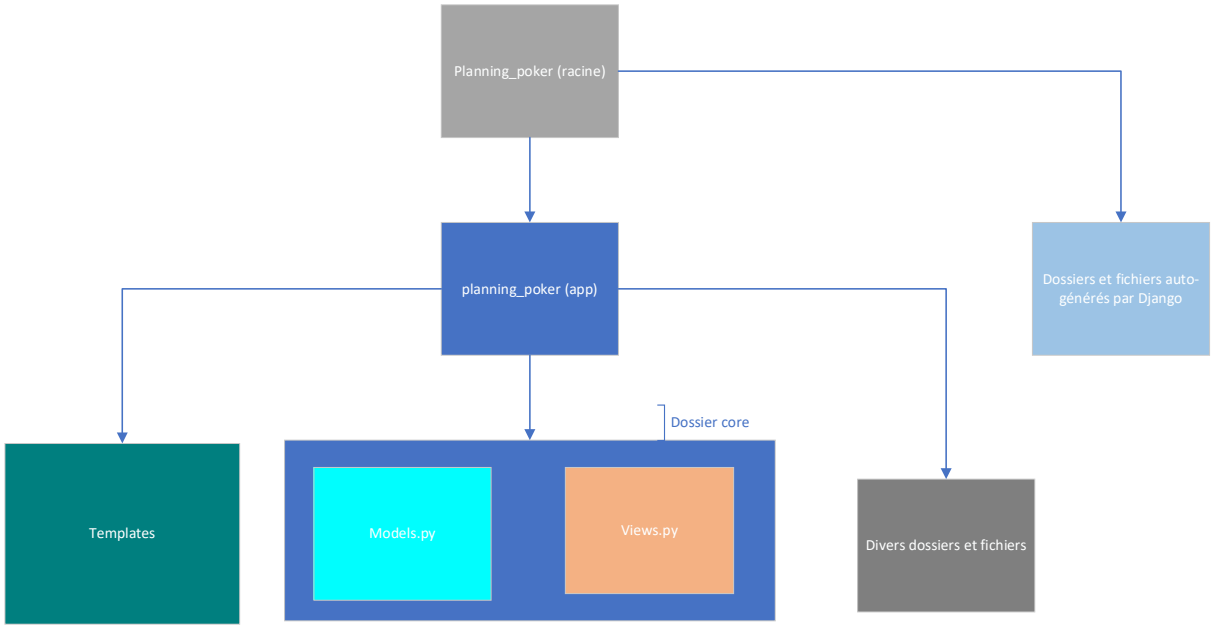

Figure 1: Architecture simplifiée du projet, modèle MVT

b. Conception de la Base de données :

La base de données dans ce projet est locale, ce choix est fait pour éviter tous problèmes de connexion lors du test par l'évaluateur. Elle est stockée dans le fichier db.sqlite3 à la racine de « planning_poker ». Elle est néanmoins parfaitement exportable sur un vrai serveur de base de données et ce de façon simple.

Les classes dans la partie Modèle du projet, en l'occurrence dans le fichier Models.py visible sur le schéma d'architecture, correspondent aux tables dont on a besoin pour faire tourner notre logique. Les attributs de ces classes correspondront aux colonnes des tables dans la base de données.
Ces classes héritent de classes mères mises à disposition par la librairie Django. Certaines classes existent de façon native comme facilité offerte, la classe user ou auth (authentification) en sont deux parfaits exemples.

La mise en œuvre du design pattern **ORM,** Object Relatinal Mapping, a permis de circonscrire les interactions avec la base de données au langage Python. En effet, l'Object Relational Mapping permet de créer, lire, mettre à jour et supprimer des enregistrements dans la base de données en utilisant une syntaxe Pythonique, sans avoir besoin d'écrire des requêtes SQL explicites. Grâce à cette abstraction, le développeur n'a pas besoin de savoir comment écrire des requêtes SQL. Il peut se concentrer sur la logique métier, sans parler de la prévention des risques d'injections et de l'amélioration de la lisibilité.

Le schéma de la base de données est joint en annexe.

c. Conception des fonctionnalités :

La logique propre au jeu est encapsulée dans les vues, qui sont regroupées dans le fichier views.py du dossier « core ». Chaque classe Vue correspond à une fonctionnalité dont on a eu besoin pour faire fonctionner notre jeu.

Ces différentes vues, fonctionnalités, seront étayées dans la documentation, partie API.

Pour faciliter la construction de nos fonctionnalités, il y a eu mise en œuvre du design pattern **decorator.** Par exemple, la vue « home » doit assurer l'affichage d'une page d'accueil pour un utilisateur connecté. C'est l'ajout du décorateur « @login_required » qui assure que seuls les utilisateurs authentifiés peuvent accéder à cette vue.

En utilisant ce design pattern facile et puissant, nous avons pu étendre ou modifier les fonctionnalités d'un objet de manière dynamique à l'exécution sans changer sa structure.

4. Développement :

a. Environnement de développement :

- **Langage de programmation Python :**

Le langage python est lisible, facile à apprendre et facile à débugger. Il offrait donc la sécurité de ne pas bloquer le projet par manque de compétences ou par un bug insolvable.
De plus, étant en voie de spécialisation dans l'IA, il était nécessaire de me poser avec un langage utile et convertible pour mon projet professionnel.

- **IDE Pycharm :**

Ce projet a été réalisé avec Pycharm. Choix subjectif découlant essentiellement de l'esthétique l'éditeur, et d'une aversion envers les outils Microsoft.

- **Librairie Django :**

Un premier essai a été fait avec Tkinter, l'esthétique laissait vraiment à désirer et n'était pas compensée par une facilité particulière d'utilisation ou d'apprentissage, ni même par un usage professionnel potentiel.

Le problème de jouer le jeu à plusieurs a fini par imposer la page web comme solution d' interface utilisateur.

La librairie Django est une librairie de python orientée web, avec laquelle j'ai réalisé un autre projet dans le passé. Elle est richement et lisiblement documentée. Acquérir une compétence dessus est faisable rapidement. De plus elle a l'avantage d'être convertible professionnellement.

- **Gestionnaire de version :**

Git, sur un dépôt distant GitHub.

5. Intégration continue :

L'intégration continue est un élément crucial du développement logiciel moderne, particulièrement dans les environnements agiles. Elle permet de s'assurer que le code est régulièrement construit, testé et intégré, ce qui aide à détecter rapidement les erreurs et à maintenir la qualité du logiciel.

La nature de l'exercice et le temps qui lui est dédié ne permettait pas d'aborder cet aspect confortablement, car l'urgence était de livrer un prototype qui remplit le cahier des charges et qui fonctionne. Néanmoins, un effort signifiant a été déployé afin de réaliser la démarche de l'intégration continue. En effet toute une panoplie de tests unitaires a été rédigée et appliquée. Et un soin particulier a été porté à la documentation.

a. Tests Unitaires

Les tests ont été rédigés essentiellement par fonctionnalité. Il s'agit de définir pour une fonctionnalité des valeurs à contrôler par des assertions. Ces assertions jouent un rôle crucial en vérifiant si les valeurs obtenues lors de l'exécution de la fonctionnalité correspondent aux valeurs attendues. La réussite ou l'échec de ces tests est déterminé par le nombre et la précision des assertions, qui varient en fonction de la complexité et des exigences de chaque fonctionnalité testée.

La librairie « pytest » a été utilisée pour automatiser l'exécution de ces tests et obtenir un rapport d'erreurs le cas échéant. Les tests unitaires sont dans le répertoire « tests » de « core », mais ils sont reconnus automatiquement grâce à leur syntaxe de nomenclature.

Ainsi, à toutes modifications de code, nous pouvons lancer et exécuter tous nos tests par la simple commande « pytest » dans la console (read.me). Et ainsi s'assurer qu'il n'y ait pas de régression ou d'erreurs dues au nouveau code intégré.

Pour aller plus loin, la librairie « coverage » permet d'automatiser l'exécution des tests , et de générer des rapports détaillés dans plusieurs formats dont le html. Ces rapports analysent et mesurent l'étendue du code exécuté pendant les tests. Ce qui revient à avoir une métrique objective de cet axe de l'intégration continue.

Dans notre application le taux de couverture mesuré par « coverage » atteint un niveau de 62%, ce qui est plutôt un bon indicateur de couverture compte tenu des délais et des priorités.

b. Documentation

Un début très compliqué avec Dioxygène et Sphinx, qui s'est avéré non concluant. En effet seul Sphinx a fini par fonctionner sur ma plateforme, et ce pour un résultat « en vrac » de la documentation qui était pour le moins inacceptable esthétiquement.

J'ai choisi de me tourner vers mkdocs pour la création et la gestion de la documentation du projet. mkdocs est un outil de génération de documentation statique extrêmement convivial. La décision d'utiliser mkdocs a été motivée par sa simplicité d'utilisation et sa capacité à produire rapidement une documentation claire et structurée.

Pour commencer, j'ai organisé la documentation en rédigeant des fichiers Markdown, en prenant soin de détailler l'API. L'un des avantages majeurs de mkdocs est sa configuration aisée : avec un fichier de configuration simple (mkdocs.yml), j'ai pu définir la structure de la documentation, personnaliser l'apparence et configurer diverses options.

Le résultat est une documentation complète, sous forme d'un site statique élégant, structuré en plusieurs pages et doté d'un moteur de recherche.

Cette solution me parait, de loin meilleure que toutes ses concurrentes. Je la conseille vivement à mes camarades, car elle a surtout l'avantage de ne pas bloquer ou créer des bugs, et sa compatibilité semble être universelle.

6. Documentation :

Le read.me précise les commandes simples à effectuer pour accéder au site de la documentation.
Il s'agit d'un site de plusieurs pages :

- Une page d'accueil
- Un guide d'installation pour usager type étudiant.
- Un guide d'utilisation : comment créer les parties et comment jouer.
- Un récapitulatif de la structure du projet et de ses différents fichiers.
- Un guide de l'api : détails des Vues et des Modèles.

Pour chaque page, vous disposerez d'une table de contents sur la droite, permettant un accès rapide à la rubrique dont vous aurez besoin.

Vous aurez également un moteur de recherche en haut à droite. Tout le contenu des fichiers Markdown est indexé, le plugin du moteur de recherche trouvera l'emplacement du mot ou de l'expression recherchée et vous y orientera.

7. Conclusion et perspectives d'avenir :

Malgré quelques difficultés, essentiellement dues au manque du temps, ou à des doutes internes d'arbitrage de l'effort avec les autres modules, le livrable a pu être finalisé dans le triangle coût-qualité-délai fixé en amont.

Ce projet a été un excellent exercice d'apprentissage et de mise en pratique des compétences clés en conception et développement de logiciels. Je suis reconnaissant pour les expériences acquises et les compétences nouvelles telles que le développement web avec Django, la gestion de projet agile, et la documentation technique. Je suis confiant que le savoir acquis au cours de ce projet sera un atout pour mes futurs projets et aspirations professionnelles.