RAPPORT DE PROJET

Application de Planning Poker

Lien du repo github : https://github.com/rakotobediana/Projet-Conception-Agile.git

Commande pour démarrer le serveur : `python app.py run`

Rakotobe Diana : 5203849
Do Minh Trang : 5221045

Introduction

Ce rapport présente le développement d'une application de Planning Poker, réalisée dans le cadre d'un projet pédagogique. L'objectif était de concevoir une solution permettant à des joueurs de collaborer sur l'estimation de tâches, en suivant les règles de Planning Poker enseignées en cours. Ce projet a été réalisé en binôme et a impliqué l'application des principes de programmation agile et pair programming. Nous avons également intégré des mécanismes d'intégration continue pour garantir la qualité et la maintenance du projet.

Contexte du projet

Le Planning Poker est une méthode d'estimation collaborative utilisée dans la gestion de projets agiles. Il permet aux membres d'une équipe de donner des estimations sur la complexité ou l'effort requis pour accomplir des tâches. Ce projet vise à reproduire ce processus sous forme numérique, avec des options pour des parties en local et différentes règles de vote.

Pour mener à bien ce projet, nous avons opté pour une approche de programmation modulaire. Cela nous a permis de découvrir de nouvelles méthodologies de développement, d'explorer des outils innovants et d'assurer une mise en place fluide et efficace des liaisons nécessaires.

Choix techniques

Pour ce projet, nous avons choisi le langage **Python**, qui est notre langage de spécialisation. Ce choix nous permettra d'acquérir davantage de compétences et d'expérience, des atouts précieux pour nos futurs projets. En effet, maîtriser Python est essentiel, car c'est un langage polyvalent utilisé dans de nombreux domaines, et il nous permettra de travailler sur des projets variés.

Concernant le framework, nous avons opté pour **Flask.** Nous avons privilégié ce dernier en raison de sa simplicité et de sa légèreté, qui nécessitent peu de configuration pour l'application. Flask permet de définir facilement les routes et de démarrer rapidement un projet. C'est un excellent choix pour un développement rapide et efficace, tout en offrant la possibilité d'ajouter des fonctionnalités au fur et à mesure de l'avancement du projet.

À long terme, nous prévoyons d'apprendre et d'utiliser Django, un framework similaire à Flask mais plus complet, qui offre des fonctionnalités plus avancées et adaptées à des projets plus complexes. Django est particulièrement utile pour des applications plus robustes, avec des besoins spécifiques et des exigences techniques plus larges. C'est la raison pour laquelle nous avons choisi d'utiliser Flask et python . Ils correspondent parfaitement aux besoins et à l'envergure de notre projet actuel mais nous permettent également d'anticiper sur nos futurs projets et expériences.

Outils utilisés

- **Flask** : Pour le développement backend.
- **Jinja** : Pour le rendu des templates HTML.
- **Pylint** : Pour garantir la qualité du code.
- **Doxygen** : Pour la génération automatique de la documentation.
- **Github Actions** : Pour automatiser les tests unitaires et assurer l'intégration continue.

Architecture de l'application

L'application a été développée en Python, en utilisant le framework Flask pour simplifier la gestion des routes et de la configuration de l'application.

- `app.py` : Fichier principal contenant les configurations de l'application et les points d'entrée.

- **`views.py`** : Gère les routes et les blueprints pour une séparation claire des responsabilités.
- **Dossier `models/`** : Contient les classes représentant les principales entités, comme `Joueur`.
- **`backlog.json`** : Fichier contenant les fonctionnalités sous forme de données JSON.
- **Dossier `conf`** : un répertoire dans lequel est stocké le fichier `Doxyfile`
- **Dossier `static`** : gère le flux css et les images
- **Dossier `venv`** : l'environnement virtuel sur lequel on a travaillé tout au long du développement du projet
- **`requirements`** : contient toutes les dépendances
- **Dossier `templates`** : un répertoire contenant la listes des fichiers html

Points clés de l'architecture :

Concernant cette architecture, nous avons adopté une programmation modulaire, ce qui permet une meilleure organisation du code et facilite la compréhension des différentes fonctions et de leur rôle dans l'application.

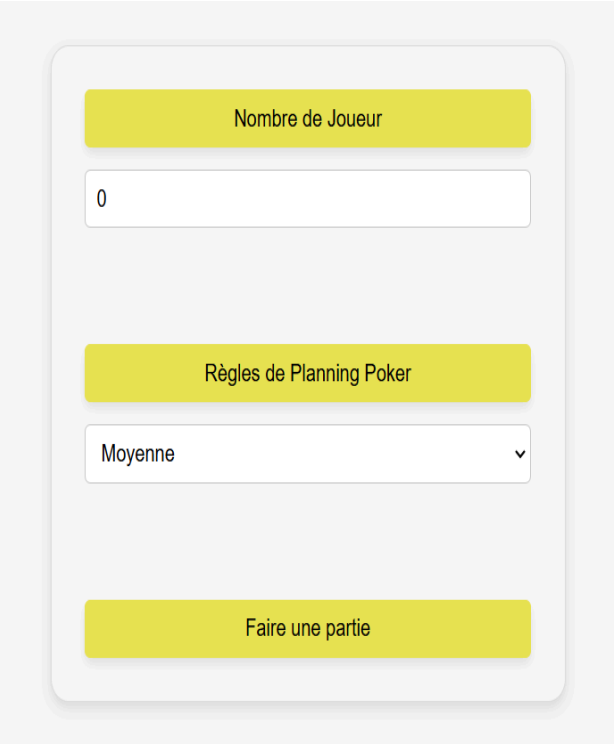
Bibliothèques

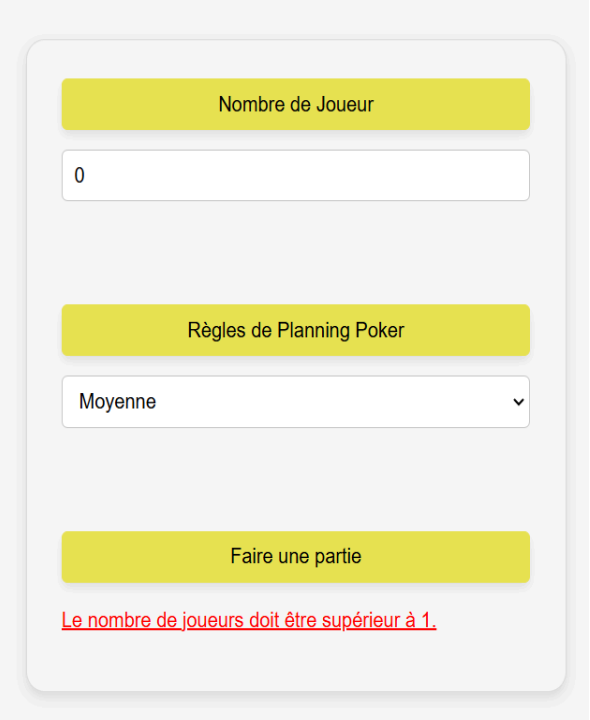
- `request` : Pour gérer les requêtes HTTP.
- `blueprint` : Pour organiser les routes.
- `redirect` : Pour rediriger les utilisateurs entre les pages.
- `json :` Pour l'importation et le traitement du backlog

Fonctionnalités de l'application

Page d'accueil :

La page d'accueil est conçue pour permettre aux utilisateurs de sélectionner le nombre de joueurs et la règle de vote qu'ils souhaitent utiliser. Le champ pour le nombre de joueurs ne peut pas être vide et doit être supérieur à 1. Si l'utilisateur saisit un nombre de joueurs incorrect (inférieur ou égal à 1), il ne pourra pas passer à l'étape suivante et sera invité à entrer un nombre valide. Après avoir sélectionné le nombre de joueurs, l'utilisateur peut choisir la règle de vote qu'il souhaite appliquer pour la partie.





Page de liste de joueurs :

Sur cette page, chaque joueur doit saisir son pseudo dans un champ de texte. Le nombre de champs est défini en fonction du nombre de joueurs sélectionné à l'étape précédente. Par exemple, si 3 joueurs sont choisis, trois champs de saisie apparaîtront. Chaque champ doit être rempli avant que l'utilisateur puisse cliquer sur le bouton "Faire une partie". Ce contrôle garantit que tous les pseudos sont renseignés correctement avant de continuer. Une fois tous les pseudos saisis, l'utilisateur peut valider la saisie en cliquant sur le bouton, ce qui envoie les données en backend pour les enregistrer et les préparer pour l'étape suivante.

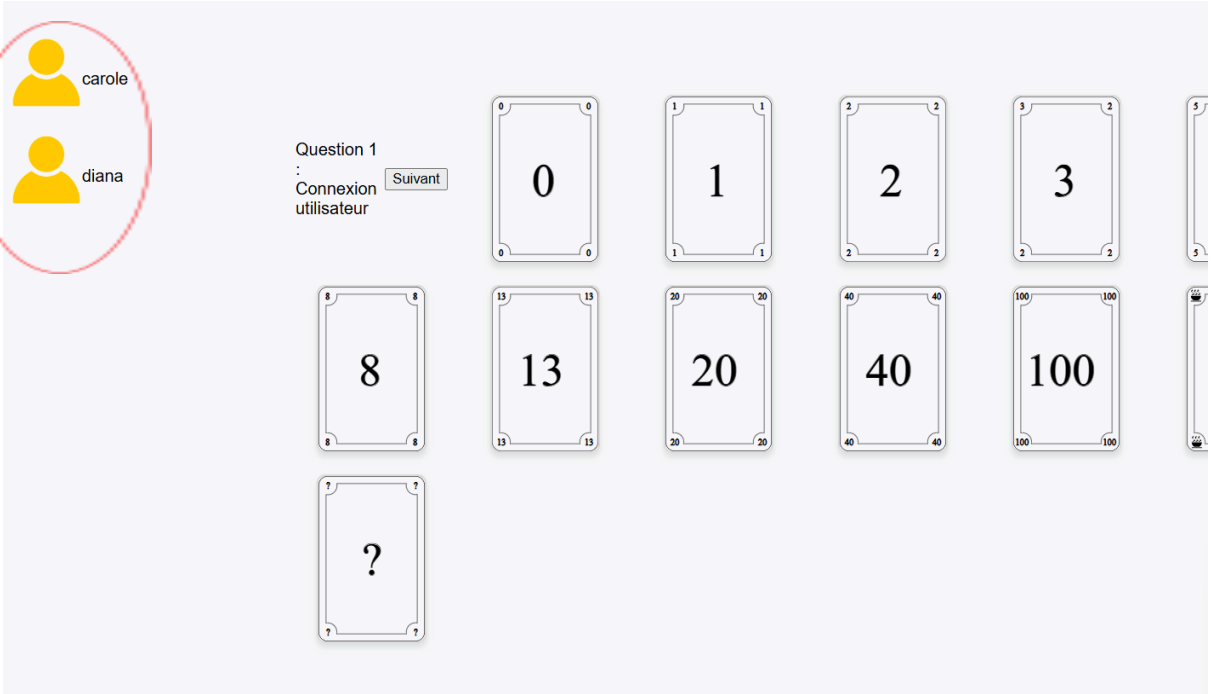
Page de jeu :

Le jeu se déroule dans une interface où les joueurs sélectionnent des cartes représentant des valeurs de complexité ou d'effort d'une fonctionnalité, selon les règles de vote définies précédemment.

Une fois que chaque joueur a sélectionné sa carte d'estimation, le vote est pris en compte. Chaque joueur doit valider son choix en cliquant sur la carte correspondant à son estimation de la tâche. Dès qu'un joueur a voté, sa sélection est enregistrée et il ne pourra plus la modifier avant la fin du tour de vote.

Lorsque tous les joueurs ont voté, le tour de vote est considéré comme terminé. Les résultats sont affichés, et l'estimation de la tâche est enregistrée en fonction des règles de validation choisies (par exemple, unanimité, moyenne, médiane, etc.).

Une fois ce tour de vote terminé, les joueurs peuvent passer à la question suivante, c'est-à-dire à la prochaine fonctionnalité ou tâche à estimer. Le processus de sélection des cartes et de vote est répété pour chaque élément du backlog, jusqu'à ce que toutes les tâches aient été estimées. À chaque tour, les joueurs peuvent interagir de manière similaire pour valider leurs votes et estimer les tâches restantes.



Les joueurs seront présentés en haut à gauche en fonction de leur nombre et de leur nom .

Intégration continue

L'intégration continue (CI) a été mise en place via GitHub Actions pour automatiser les tests, la vérification du style de code, ainsi que la génération de la documentation. Un workflow spécifique a été configuré pour exécuter ces tâches de manière autonome à chaque modification du code. Ce processus garantit une qualité constante du code tout au long du développement.

1. **Exécution des tests unitaires :**

   Un des objectifs principaux de l'intégration continue est de s'assurer que chaque modification n'introduit pas de régressions dans l'application. Les tests unitaires sont exécutés automatiquement à chaque push ou pull request via le workflow. Cela permet de valider que le code fonctionne correctement avant qu'il ne soit fusionné dans la branche principale.

2. **Vérification de la conformité au style de code :**

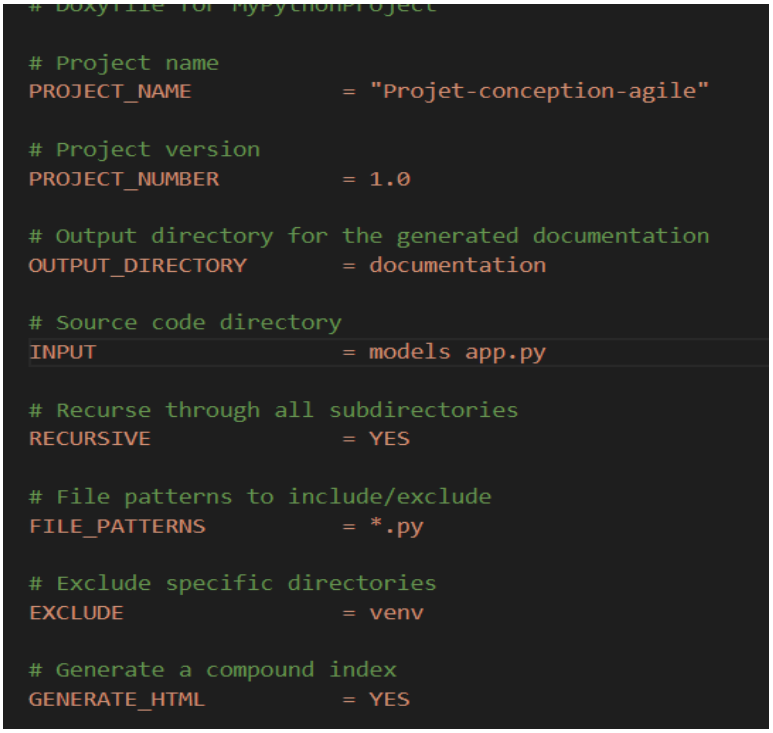
   Afin de maintenir une qualité de code élevée et conforme aux bonnes pratiques Python, des outils comme Pylint et Apycodestyle sont utilisés pour vérifier le style du code. Ces outils analysent le code et vérifient qu'il respecte les conventions et standards établis. Si des écarts sont détectés, une alerte est envoyée, permettant de corriger rapidement les problèmes avant de procéder à l'intégration du code.

3. **Génération automatique de la documentation :**

   La génération de la documentation est un autre aspect essentiel de l'intégration continue. Pour ce faire, j'ai utilisé **Doxygen**, un outil de documentation largement utilisé pour générer automatiquement la documentation à partir des commentaires dans le code source. Doxygen permet de produire une documentation détaillée et structurée, avec des sections telles que la description des classes, des méthodes, et des fichiers, ce qui facilite la compréhension du code par les développeurs.

   - Un dossier `.github/workflows` dans lequel se trouve le fichier `doxygen_documentation.yml`. Ce fichier contient la configuration de mon workflow GitHub Actions pour gérer l'exécution automatique des builds, des tests, ainsi que la génération de la documentation.

   - Le fichier `Doxyfile` a été configuré pour définir les **entrées** et les **sorties** de la documentation. Ce fichier spécifie quels fichiers source doivent être inclus dans la génération de la documentation (entrées), et où la documentation générée doit être stockée (sorties). Le répertoire de sortie est configuré pour contenir la documentation générée sous un format lisible, prêt à être consulté par les développeurs.



```
# Doxyfile for MyPythonProject

# Project name
PROJECT_NAME           = "Projet-conception-agile"

# Project version
PROJECT_NUMBER         = 1.0

# Output directory for the generated documentation
OUTPUT_DIRECTORY       = documentation

# Source code directory
INPUT                  = models app.py

# Recurse through all subdirectories
RECURSIVE              = YES

# File patterns to include/exclude
FILE_PATTERNS          = *.py

# Exclude specific directories
EXCLUDE                = venv

# Generate a compound index
GENERATE_HTML          = YES
```

Gestion de projet et collaboration

Le projet a été géré en appliquant les principes agiles, avec un focus sur le pair programming. Les branches Git ont permis de travailler simultanément sur différentes fonctionnalités tout en limitant les conflits.

Gestion des branches :

Dans ce projet, nous avons appliqué la méthode du **branching** pour gérer efficacement les différentes fonctionnalités et les corrections de bugs. Chaque nouvelle fonctionnalité ou tâche a été développée dans une branche dédiée afin de ne pas interférer avec le code principal.

1. **Création de branches :**

   Pour chaque fonctionnalité, une branche a été créée à partir de la branche principale avec la commande :

   ```
   git checkout -b feature/nouvelle-fonctionnalité
   ```

2. **Résolution des conflits :**

   Lors de l'intégration des modifications, des conflits ont été résolus grâce aux commandes suivantes :

   - ➢ **`git fetch`** pour récupérer les dernières modifications à partir de la branche distante.
   - ➢ **`git merge`** pour fusionner les modifications locales avec la branche principale et gérer les conflits.
   - ➢ **`git checkout`** pour restaurer ou naviguer entre les versions de fichiers en conflit.

3. **Validation continue :**

   Avant de fusionner une branche dans la branche principale, nous avons exécuté les tests unitaires via **GitHub Actions** pour vérifier que les modifications ne causaient pas de régressions. Les tests ont été validés dans le workflow automatisé, garantissant une intégration propre des modifications après chaque validation des tests.

Difficultés rencontrées

Lors du développement du projet, nous avons rencontré plusieurs difficultés, principalement liées à l'analyse et à la récupération des informations provenant du frontend. Ces tâches ont nécessité plus de temps que prévu, ce qui a impacté notre capacité à finaliser certains aspects du projet dans les délais impartis. De plus, l'élaboration de certaines fonctionnalités a demandé des ajustements et des améliorations continues, ce qui a également ralenti le processus.

Il aurait été bénéfique de disposer de plus de temps pour peaufiner ces éléments, améliorer les parties du projet que nous n'avons pas pu finaliser, et approfondir certains aspects sur lesquels nous n'avons pas pu exceller. Une meilleure gestion du temps et des ressources aurait permis de maximiser la qualité et la performance de l'application.

Conclusion

La réalisation de ce projet a été une expérience extrêmement enrichissante pour nous. En travaillant sur cette application de Planning Poker, nous avons non seulement renforcé nos compétences techniques, mais également élargi nos connaissances en développement logiciel. L'apprentissage et l'utilisation du framework **Flask** nous ont permis de découvrir de nouvelles pratiques de développement, tout en facilitant la gestion des routes et de la configuration de l'application.

Nous avons également amélioré notre maîtrise de techniques de développement essentielles telles que la gestion des branches avec **Git**, qui nous a permis de travailler efficacement en équipe et d'éviter les conflits de code. La mise en place de l'**intégration continue** avec **GitHub Actions** a été un point crucial, nous permettant d'automatiser certaines tâches, comme la génération automatique de la documentation via **Doxygen**.

De plus, ce projet nous a appris l'importance de l'utilisation d'un **environnement virtuel** pour gérer les dépendances de manière propre et garantir la réutilisabilité du code, en intégrant un fichier **requirements**. Cette approche a optimisé notre travail en équipe et facilité la configuration de l'environnement de développement pour chacun d'entre nous.

En somme, ce projet nous a permis de renforcer nos compétences en gestion de projets, en développement backend et en gestion de version avec Git. Nous avons appris à organiser efficacement notre code, à collaborer et à automatiser certaines étapes cruciales du développement, ce qui nous servira de base solide pour nos futurs projets.