

Application de Planning Poker

UE : Conception Agile – M1 Informatique

Équipe : FALL Madjiguène, BARRY Aissatou Lamarana , NGUEGANG Berthol Ivan

Encadrant : Valentin Lachand-Pascal

Date de rendu : 19 décembre

1. Introduction et Contexte

Dans le cadre de l'UE *Conception Agile*, nous avons réalisé une application de **Planning Poker**, un outil collaboratif utilisé en méthodes agiles pour estimer la complexité des tâches d'un backlog.

Objectif du projet : permettre à plusieurs joueurs de participer à une session de vote afin d'estimer des fonctionnalités, en respectant les règles du Planning Poker vues en cours. L'application doit gérer différents modes de calcul (strict, moyenne, médiane, etc.), charger un backlog depuis un fichier JSON, sauvegarder l'avancement de la partie et permettre la reprise ultérieure.

Organisation du travail : le projet a été développé en équipe de trois personnes en appliquant le **pair programming** et le travail collaboratif. Fall Madjiguène et NGUEGANG Berthol Ivan ont pris en charge le **frontend**, BARRY Aissatou le **backend**, et l'intégration continue a été mise en place collectivement. GitHub a été utilisé pour le versionnement et le suivi du projet.

2. Choix Techniques et Architecture

2.1 Langage et Frameworks

Nous avons choisi d'implémenter le projet en **Python**, avec le framework **Flask**, complété par **Flask-SocketIO**.

Ce choix est motivé par plusieurs raisons :

- Python est un langage simple, lisible et bien adapté à un projet académique mettant l'accent sur la logique métier.

- Flask est un framework léger, permettant une structuration claire du code sans surcharge inutile.
- Flask-SocketIO permet la gestion du **temps réel**, indispensable pour une application de Planning Poker multi-utilisateurs (soumission des votes, révélation simultanée, synchronisation de l'état de la session).

Côté frontend, nous utilisons :

- **HTML/CSS** pour la structure et le style des pages,
- **JavaScript** pour la logique côté client et les échanges temps réel avec le serveur via SocketIO.

2.2 Architecture Logicielle

L'application suit une architecture proche du **MVC (Modèle – Vue – Contrôleur)** :

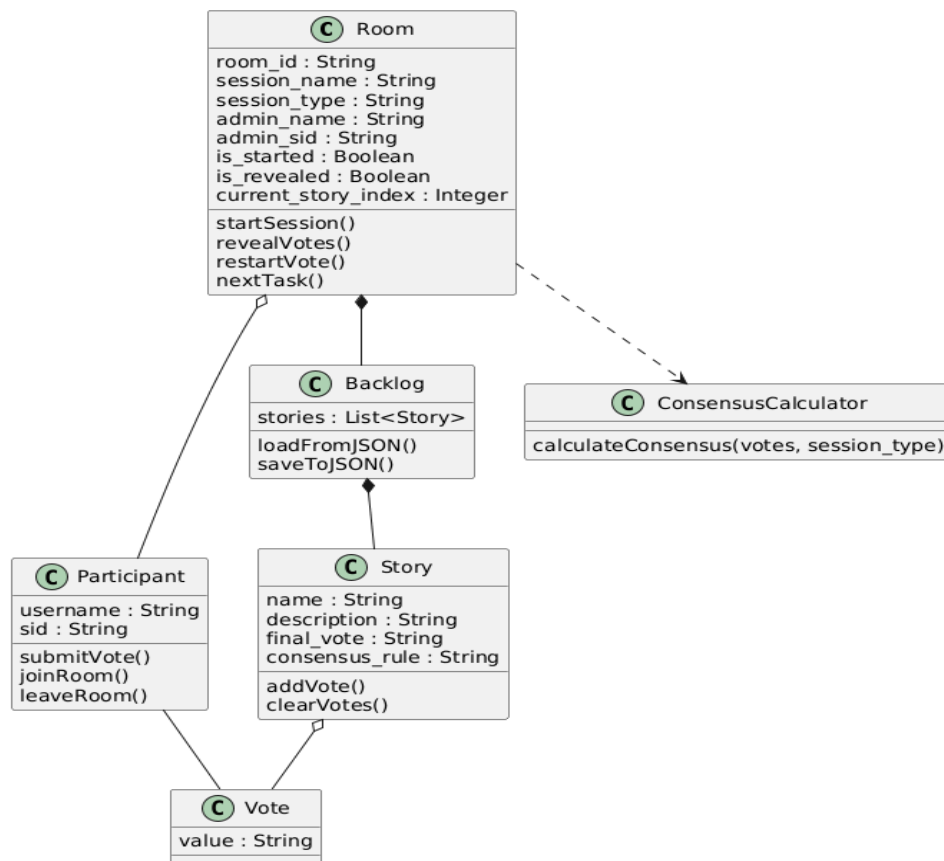
- **Modèle** : les données sont stockées côté serveur dans des structures Python (dictionnaires) représentant les salles, les joueurs, les votes et le backlog.
- **Vue** : les templates HTML (home.html, room.html) définissent l'interface utilisateur.
- **Contrôleur** : app.py gère les routes Flask et les événements SocketIO (création de salle, vote, révélation, changement de tâche).

Cette séparation permet de maintenir un code clair, compréhensible et évolutif.

2.3 Modélisation (Diagrammes)

Le modèle de données repose sur les entités principales suivantes :

- **Salle (Room)** : identifiée par un identifiant unique, elle contient les paramètres de la session (mode de jeu, administrateur), l'état courant et la liste des participants. La salle référence également le backlog des tâches à estimer et utilise le calculateur de consensus pour déterminer les résultats des votes.
- **Joueur (Participant)** : défini par un pseudo et une connexion SocketIO. Chaque joueur peut rejoindre ou quitter une salle et soumettre un vote pour chaque tâche.
- **Backlog** : liste de tâches chargées depuis un fichier JSON. Chaque tâche (*Story*) contient son nom, sa description et l'ensemble des votes soumis par les participants.
- **Vote** : association entre un joueur et une valeur de carte, représentant le choix du participant pour une tâche donnée.
- **ConsensusCalculator** : classe responsable du calcul du résultat final d'un tour de vote. Elle applique les règles de consensus définies pour la session (strict, moyenne, médiane) et retourne le résultat ainsi que les détails du calcul.



2.4 Gestion des Données (JSON)

Les données du backlog sont chargées depuis un fichier JSON fourni par l'utilisateur. Chaque tâche contient :

- un nom,
- une description,
- les votes soumis,
- le résultat final du vote calculé selon la règle de consensus (strict, moyenne, médiane, majorité).

Les votes restent cachés jusqu'à leur révélation, et les participants peuvent les modifier tant que la story n'est pas validée. À la fin de la session, ou en cas de carte café, l'état complet du backlog est sauvegardé dans un fichier JSON, permettant de reprendre la partie ultérieurement.

3. Mise en place de l'Intégration Continue

3.1 Workflow CI

Nous avons mis en place une **intégration continue avec GitHub Actions**. Le pipeline CI est déclenché automatiquement à chaque *push* ou *pull request* sur le dépôt.

Les étapes principales du workflow sont :

1. **Checkout** : récupération du code source depuis le dépôt.
2. **Installation** : installation des dépendances Python listées dans requirements.txt.
3. **Tests** : exécution automatique des tests unitaires.
4. **Documentation** : génération automatique de la documentation.

3.2 Tests Unitaires

Les tests unitaires du projet **Projet_Planning_Poker-Agile** portent principalement sur la logique métier, en isolant les fonctions critiques pour garantir leur fiabilité, indépendamment de l'interface utilisateur.

Plus précisément :

1. **Calcul du consensus en mode strict (unanimité)**
 - Vérifie que tous les votes sont identiques pour déterminer un consensus.
 - Si tous les votes concordent, le consensus est retourné, sinon la fonction signale un échec (NO CONSENSUS).
2. **Calcul de la moyenne (average)**
 - Vérifie le calcul correct de la moyenne des votes numériques.
 - Les tests confirment que la moyenne est calculée correctement et que le détail de la session mentionne « Moyenne simple ».
3. **Calcul de la médiane (median)**
 - Vérifie le calcul de la médiane pour un nombre impair ou pair de votes.
 - Les tests assurent que le résultat correspond à la médiane réelle et que le message détaille correctement la valeur médiane.
4. **Gestion des votes non numériques**
 - Teste la robustesse de la fonction lorsque les votes contiennent des caractères spéciaux ou des symboles.
 - Le système retourne "N/A" et un message explicatif indiquant qu'aucun vote valide n'a été détecté.
5. **Tests SocketIO avec contexte Flask**
 - Vérifie le bon fonctionnement des fonctions liées aux événements SocketIO (on_submit_vote, on_request_backlog_download) dans un contexte Flask simulé (app.test_request_context()).

- Les tests confirment la mise à jour des votes, l'émission des événements et la cohérence des données du backlog pour chaque salle.

Ces tests sont automatisés et exécutables via **pytest**, garantissant que la logique métier reste fiable malgré les modifications futures du code.

3.3 Génération de la Documentation avec Doxygen

La documentation du projet est automatiquement générée à partir des **docstrings Python** présentes dans le code source app.py grâce à **Doxygen**.

Principe de fonctionnement :

1. Configuration de Doxygen

- Le fichier **Doxyfile** définit les paramètres :
 - **INPUT** = . → analyse tous les fichiers Python à la racine et dans les sous-dossiers.
 - **RECURSIVE** = YES → inclut tous les sous-dossiers du projet.
 - **GENERATE_HTML** = YES → produit la documentation au format HTML.
 - **CALL_GRAPH** = YES et **CALLER_GRAPH** = YES → génère les graphes d'appels et de dépendances.
- Les docstrings doivent être présentes dans le code pour que Doxygen puisse les extraire correctement.

2. Génération de la documentation

- La commande **doxygen config/Doxyfile** analyse le code et produit la documentation dans **documentation/html**.
- Les pages HTML incluent :
 - La description des modules et fonctions.
 - Les paramètres et retours des fonctions.
 - Les graphes des appels de fonctions (si applicable).
 - Les variables globales et structures de données importantes.

3. Publication

- La CI/CD peut automatiquement publier la documentation sur **GitHub Pages**, permettant un accès public et permanent.
- Chaque mise à jour du code entraîne une régénération de la documentation, assurant sa **cohérence avec le code source**.

Cette méthode garantit que toute personne qui consulte le projet dispose d'une documentation complète, structurée et toujours à jour, sans nécessiter d'intervention manuelle.

4. Manuel Utilisateur et Fonctionnalités

4.1 Installation et Lancement

Le projet est conçu pour être léger et facilement déployable sur n'importe quelle machine disposant de Python.

Prérequis :

- Git
- Python

Procédure étape par étape :

1. Cloner le projet :

Ouvrez un terminal, placez-vous dans le répertoire où vous souhaitez enregistrer le projet et exécutez :

```
git clone https://github.com/albarry002/Projet\_Planning\_Poker-Agile-.git
```

2. Accéder au répertoire du projet :

```
cd planning-poker
```

3. Installer les dépendances :

```
pip install -r requirements.txt
```

4. Lancer l'application :

```
python app.py
```

5. Accéder à l'application :

Ouvrez un navigateur web et allez à l'adresse : <http://127.0.0.1:5000>

4.2 Modes de Jeu et Règles de Consensus

L'application permet à l'administrateur de définir la règle mathématique qui déterminera l'estimation finale de chaque tâche. Ce choix s'effectue lors de la création de la session.

Modes implémentés :

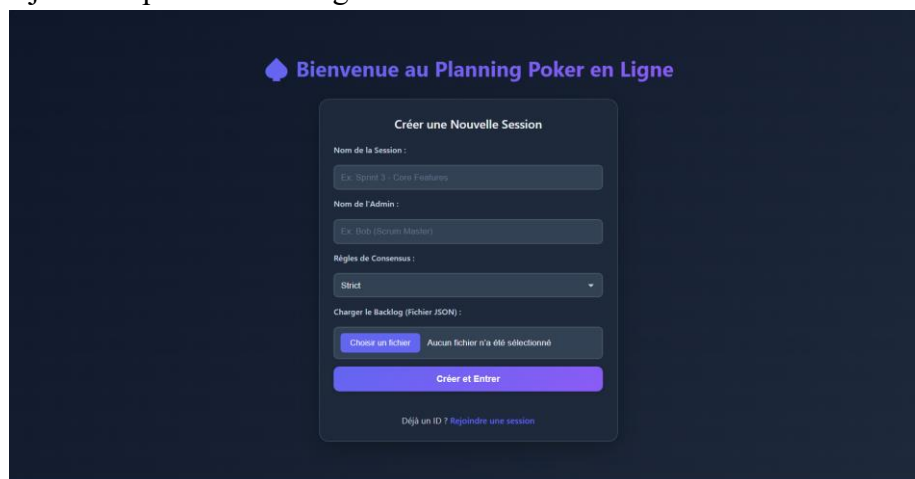
- **Mode Strict (Unanimité) :**
 - Tous les participants doivent voter pour la même valeur pour valider l'estimation.
 - Les votes non numériques comme "Café" ou "?" sont exclus du calcul.
 - En cas de divergence, le système affiche "**NO CONSENSUS**" et l'administrateur relance le vote après discussion.
- **Mode Moyenne :**
 - Le système calcule la moyenne arithmétique des votes numériques.
 - Le résultat est arrondi à une décimale.
- **Mode Médiane :**
 - Les votes numériques sont triés et la valeur centrale est sélectionnée.
 - Les valeurs aberrantes sont ignorées.

4.3 Interface et Flux d'une Partie

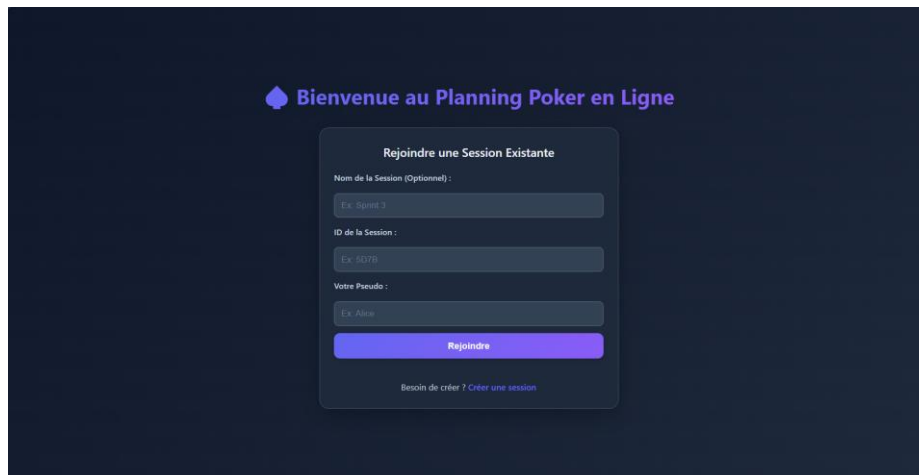
L'interface est ergonomique, avec un thème sombre et réactive grâce aux WebSockets. Le flux d'une partie se déroule en quatre phases :

Phase 1 : Menu Principal et Configuration

- Créer une session : le Scrum Master saisit son pseudo, le nom de la session, choisit le mode de jeu et importe le backlog JSON.



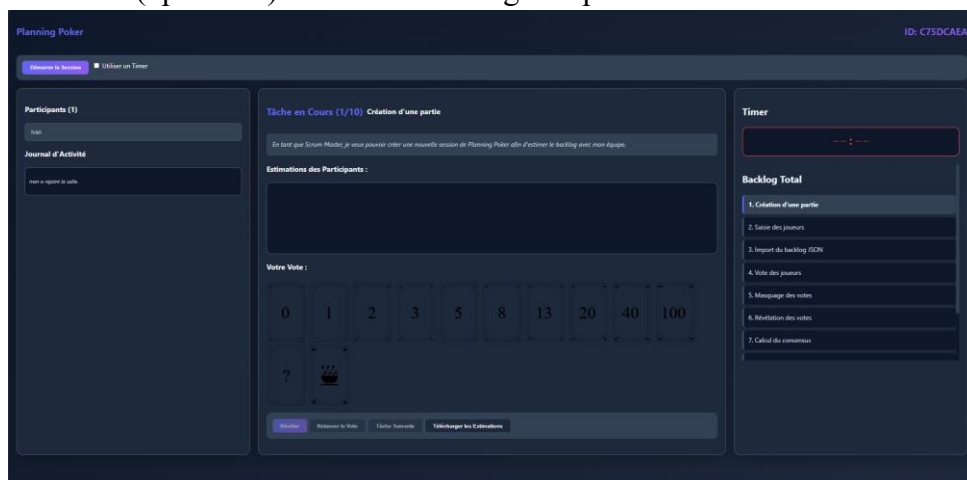
- Rejoindre une session : les joueurs entrent leur pseudo et l'ID de session fourni par l'admin.



Phase 2 : La Salle de Jeu

L'écran est divisé en trois colonnes :

- **Gauche** : Liste des participants et journal d'activité en temps réel.
- **Centre (Zone de jeu)** : Détails de la tâche en cours, tapis de jeu (cartes face cachée ou révélée), deck de vote personnel.
- **Droite** : Timer (optionnel) et liste du backlog complet.



Phase 3 : Déroulement du Vote

- **Vote** : Chaque joueur sélectionne une carte (0, 1, 2, 3, 5, 8, 13, 20, 40, 100, ?, café).
- **Révélation** : L'administrateur clique sur "Révéler" lorsque tous les votes sont soumis.
- **Résultat** : Les cartes se retournent avec animation, le résultat du consensus s'affiche en vert et est sauvegardé.
- **Tâche suivante** : L'administrateur clique sur "Tâche Suivante", réinitialisant le tapis pour la nouvelle tâche.

Phase 4 : Sauvegarde et Export

- À tout moment ou à la fin de la session, l'administrateur peut cliquer sur **"Télécharger les Estimations"**.
- Un fichier JSON est généré avec :
 - Les estimations validées
 - La règle de consensus utilisée
 - L'état des votes

Remarque sur le backlog JSON :

- Chaque tâche contient : nom, description, votes soumis et résultat final du vote.
- En cas de carte **Café**, le fichier JSON est généré pour sauvegarder l'état et permettre la reprise ultérieure.

- Fichier Json avant estimation34

```
{
  "name": "Saisie des joueurs",
  "description": "En tant qu'utilisateur, je veux saisir le nom des joueurs afin qu'ils puissent pa
},
```

- Fichier Json après estimation

```
{
  "name": "Saisie des joueurs",
  "description": "En tant qu'utilisateur, je veux saisir le nom des joueurs afin qu'ils puissent p
  "votes": {},
  "final_vote": "5.0",
  "consensus_rule": "strict",
  "votes_submitted": {
    "Ivan": "5",
    "Fall": "5",
    "Aïssatou": "5"
  }
},
```

Conclusion

Ce projet nous a permis de mettre en pratique les concepts de l'agilité, du travail collaboratif et de l'intégration continue. L'application développée respecte les règles du Planning Poker, est fonctionnelle, extensible et documentée. Des améliorations futures pourraient inclure un chat intégré, une gestion avancée des statistiques ou un déploiement en ligne.