Université Lumière Lyon 2

Master Informatique

Conception agile de projets informatiques

SOUS LE THÈME

Planning Poker - Application Streamlit

Elaboré par :

- Meksem Hamza
- Charaf-eddine Berhili
- Gourainy Mohamed

Encadrant professionnel :

Valentin Lachand-Pascal

Remerciement

Nous tenons tout d'abord à exprimer notre profonde gratitude à **Monsieur Valentin Lachand-Pascal**, notre enseignant encadrant, pour son accompagnement, ses conseils avisés et son suivi tout au long de la réalisation de ce projet. Grâce à ses orientations pédagogiques et à ses remarques constructives, nous avons pu approfondir notre compréhension des concepts abordés et améliorer la qualité technique de notre application.

Nous remercions également l'ensemble des membres de notre groupe pour l'esprit de collaboration, l'entraide et l'investissement dont chacun a fait preuve durant les différentes phases du projet, depuis l'analyse des besoins jusqu'à l'implémentation finale de la solution.

Enfin, nous adressons nos remerciements à l'ensemble des ressources pédagogiques et documentations techniques consultées, ainsi qu'aux outils d'assistance utilisés durant le développement, qui ont contribué à surmonter certaines difficultés techniques et à mener ce travail à bien.

Table des matières

1. Contexte du projet

Dans le cadre de notre formation, ce projet a été réalisé afin de mettre en pratique les connaissances acquises en développement d'applications informatiques et en gestion de projet. Les méthodes agiles sont aujourd'hui largement adoptées dans les projets informatiques, car elles favorisent la collaboration, la communication et l'adaptation aux changements. Parmi ces pratiques, l'estimation des tâches occupe une place centrale pour assurer une planification réaliste et efficace.

Le *Planning Poker* est une technique d'estimation collaborative couramment utilisée dans les méthodes agiles. Elle permet aux membres d'une équipe de participer activement à l'évaluation des fonctionnalités et d'aboutir à des estimations partagées et plus fiables. Ce projet s'inscrit dans ce contexte et vise à proposer une application facilitant ce type d'estimation.

2. Problématique

L'estimation des tâches dans un projet informatique peut s'avérer complexe, notamment lorsque plusieurs personnes participent au processus. Les différences d'expérience, de compréhension ou de perception de la difficulté d'une fonctionnalité peuvent conduire à des estimations imprécises ou incohérentes. Par ailleurs, l'absence d'outils adaptés peut rendre les sessions d'estimation longues, désorganisées et peu efficaces.

La problématique principale de ce projet est donc la suivante :
comment faciliter et structurer les sessions d'estimation collaborative afin d'améliorer leur efficacité et leur compréhension par l'ensemble des participants ?

3. Objectifs

L'objectif principal de ce projet est de concevoir et de développer une application permettant de réaliser des sessions de *Planning Poker* de manière simple et intuitive.

Les objectifs spécifiques sont les suivants :

- mettre en place une interface claire et facile à utiliser ;
- permettre la participation de plusieurs utilisateurs à une même session d'estimation ;
- faciliter l'organisation des votes et la visualisation de leur état ;
- appliquer les concepts abordés en cours en développement d'applications ;
- utiliser des outils de gestion de versions afin de structurer le travail réalisé.

4. Approche de développement : travail collaboratif et organisation du projet

Le projet a été réalisé dans le cadre d'un **travail en trinôme**, selon une approche collaborative. Chaque membre de l'équipe a participé activement au développement de l'application. Deux membres ont travaillé sur des **branches Git dédiées** afin de développer des fonctionnalités de manière indépendante, tandis qu'un troisième membre a également contribué au développement tout en assurant la **gestion de la branche principale**, la revue du code et l'intégration finale des contributions via des *Pull Requests*.

L'utilisation de **GitHub** a permis d'assurer une bonne coordination du travail, une traçabilité claire des contributions et une organisation proche des pratiques professionnelles. Cette approche collaborative a contribué à la qualité, à la stabilité et à la maintenabilité de l'application développée.

2. Choix Techniques et Architecture

Cette section présente les choix techniques et architecturaux effectués lors du développement de l'application *Planning Poker*. L'objectif n'est pas uniquement de décrire les technologies utilisées, mais surtout de **justifier leur pertinence** par rapport aux besoins du projet et au contexte académique.

2.1 Langage et frameworks

Le langage de programmation choisi pour ce projet est **Python**. Ce choix s'explique par plusieurs raisons. Python est un langage largement utilisé dans le domaine du développement logiciel, reconnu pour sa simplicité syntaxique, sa lisibilité et sa rapidité de prise en main. Il est particulièrement adapté à un projet académique de niveau Master 1, car il permet de se concentrer sur la logique applicative plutôt que sur la complexité du langage.

Le framework **Streamlit** a été utilisé pour le développement de l'interface utilisateur. Streamlit permet de créer rapidement des applications web interactives à partir de code Python, sans nécessiter de connaissances avancées en développement web (HTML, CSS ou JavaScript). Ce choix est pertinent dans le cadre de ce projet, car il facilite la création d'interfaces dynamiques et interactives, tout en réduisant le temps de développement.

Les bibliothèques Python utilisées permettent notamment :

- la gestion de l'interface utilisateur et des interactions (Streamlit),
- la manipulation de données structurées (JSON),
- la gestion du temps et de l'état de l'application.

Au cours **des premières semaines du projet**, l'équipe avait initialement envisagé une approche différente pour le développement de l'application. Toutefois, après plusieurs phases de réflexion, d'expérimentation et d'échanges, il est apparu nécessaire de revoir certains choix techniques afin de mieux répondre aux objectifs du projet. Ce changement d'orientation a conduit à l'adoption du framework **Streamlit**, impliquant une adaptation du code existant et une réorganisation de la structure de l'application. Ce choix s'est révélé pertinent, car Streamlit facilite fortement la création d'interfaces web interactives en Python, sans recourir à des technologies web complexes. Il permet également une gestion simplifiée des interactions utilisateur et de l'état de l'application, ce qui a permis à l'équipe de se concentrer davantage sur la logique métier du Planning Poker, tout en améliorant la productivité, la lisibilité du code et la qualité globale de l'application.

2.2 Architecture logicielle

L'application ne suit pas strictement une architecture MVC complète, mais adopte une organisation **modulaire en couches** afin de séparer clairement les responsabilités. La couche de **présentation** est gérée par Streamlit dans app.py et les éléments de style dans src/ui/, tandis que la **logique métier** (joueurs, fonctionnalités, règles de vote et déroulement de la partie) est regroupée dans src/models/ à travers les classes Player, Feature et Game. La **gestion des données** (chargement du backlog et sauvegarde/chargement d'une partie) est assurée via src/utils/json_handler.py en utilisant le format JSON. Enfin, les constantes (valeurs des cartes, modes de vote, etc.) sont centralisées dans src/utils/constants.py. Cette structuration facilite la lisibilité, la maintenance et l'évolution du projet.

2.3 Modélisation (diagrammes)

Afin de mieux comprendre et expliquer le modèle de données de l'application, des **diagrammes de classes simplifiés** ont été utilisés. Ces diagrammes permettent de représenter les entités principales du système et leurs relations, sans entrer dans le détail complet de l'implémentation.

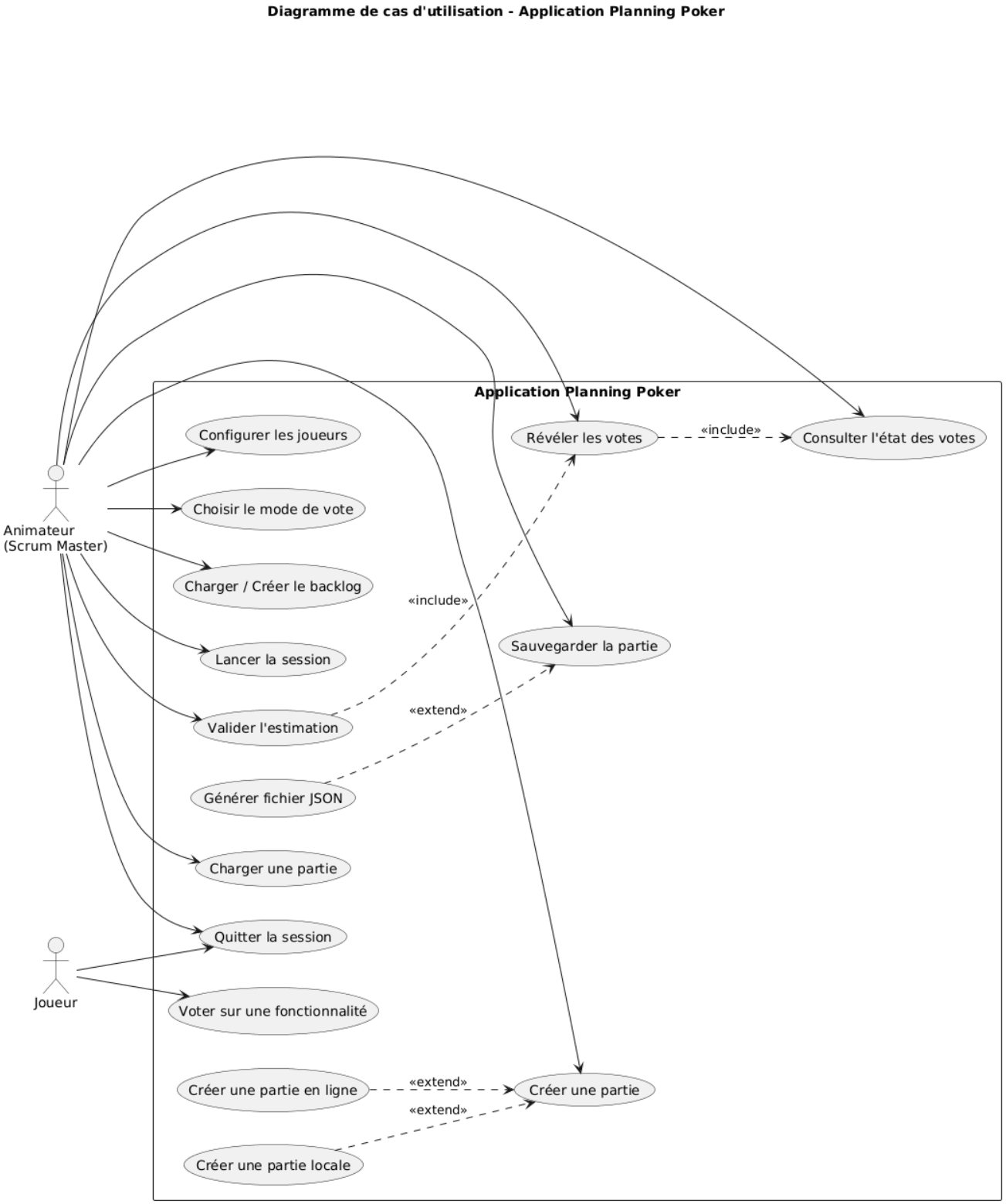
Les classes clés retenues pour la modélisation sont :

- **Joueur (Player)** : représente un participant à la session de Planning Poker ;
- **Backlog / Feature** : représente une fonctionnalité à estimer ;
- **Vote** : représente le choix effectué par un joueur ;
- **Game** : gère le déroulement global de la partie et les règles d'estimation.

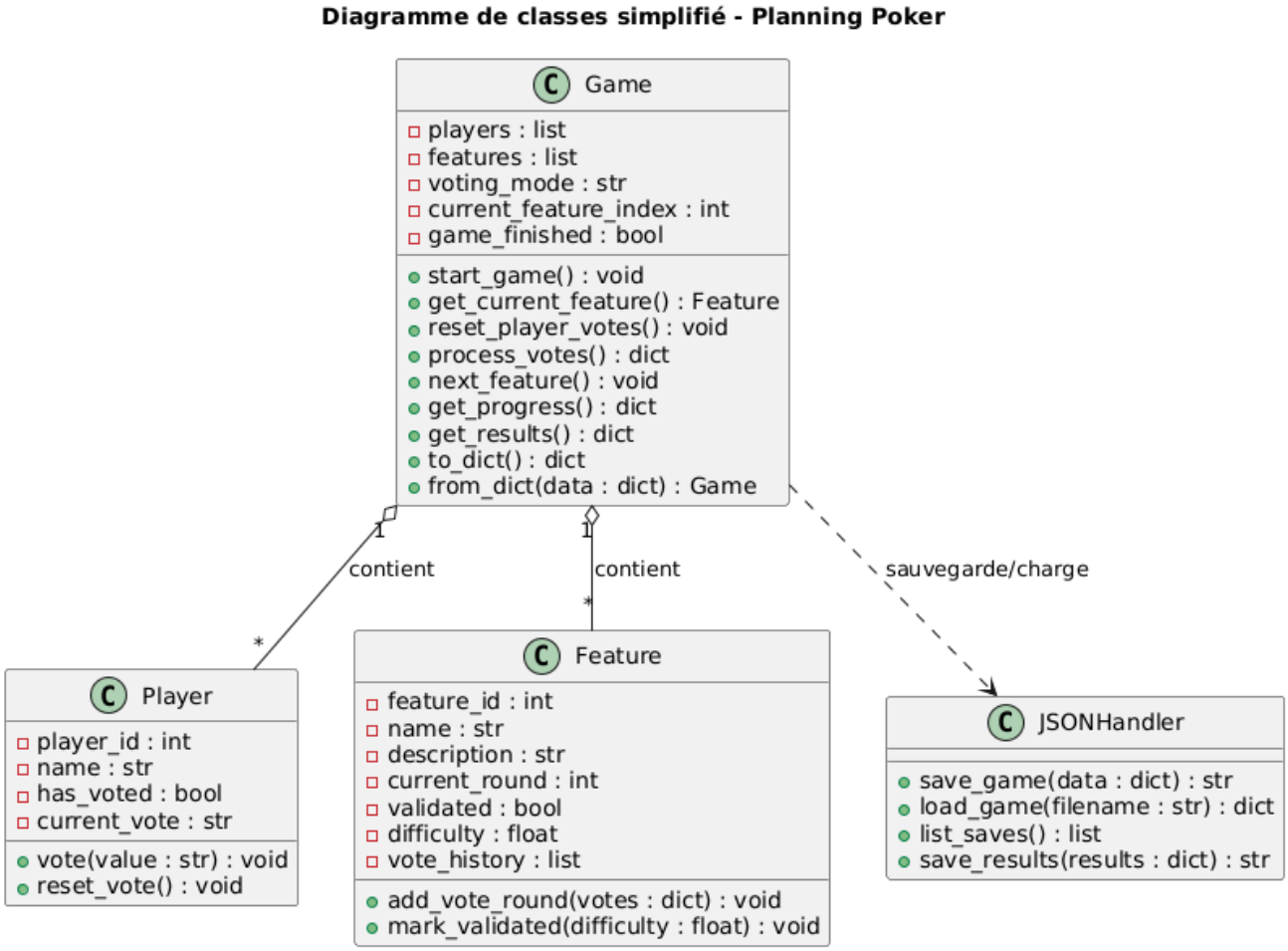
Le choix a été fait de ne représenter que ces classes principales afin d'éviter un diagramme trop complexe et difficile à lire. Cette sélection permet de se concentrer sur les éléments essentiels du fonctionnement de l'application, conformément aux consignes du projet.

Les diagrammes servent ainsi de support à la compréhension de la structure interne du système et de la répartition des responsabilités entre les différentes classes.

- Diagramme de cas d'utilisation :



- Diagramme de classes :



2.4 Gestion des données

La gestion des données repose sur l'utilisation de fichiers JSON, notamment pour :

- le chargement du backlog de fonctionnalités,
- la sauvegarde et le chargement de l'état d'une partie.

Le format JSON a été choisi pour sa simplicité, sa lisibilité et sa compatibilité avec Python. Il permet de stocker les données de manière structurée tout en restant facilement modifiable et compréhensible par un utilisateur.

Les fichiers de backlog contiennent principalement :

- le nom des fonctionnalités,
- leur description.

Les fichiers de sauvegarde de partie permettent de conserver :

- l'état des joueurs,
- les votes effectués,
- la progression de la session.

Ce mode de gestion des données est adapté au contexte du projet, car il ne nécessite pas la mise en place d'une base de données complexe, tout en assurant une persistance suffisante des informations.

3. Manuel Utilisateur et Fonctionnalités

Cette section a pour objectif de décrire le fonctionnement de l'application du point de vue utilisateur. Elle explique comment installer, lancer et utiliser le logiciel, ainsi que les différents modes de jeu et le déroulement d'une session de Planning Poker.

3.1 Modes de jeu

L'application implémente plusieurs **modes de vote**, permettant d'adapter le processus d'estimation aux besoins de l'équipe.

Mode Strict

Le mode Strict exige une unanimité totale entre les joueurs pour valider l'estimation d'une fonctionnalité. Si les votes diffèrent, un nouveau tour de discussion et de vote est nécessaire. Ce mode favorise le consensus et les échanges approfondis entre les participants.

Mode Moyenne / Médiane

En complément du mode strict, l'application propose un mode alternatif basé sur la moyenne ou la médiane des votes (selon le paramétrage choisi).

Une particularité importante est la règle du premier tour à l'unanimité :

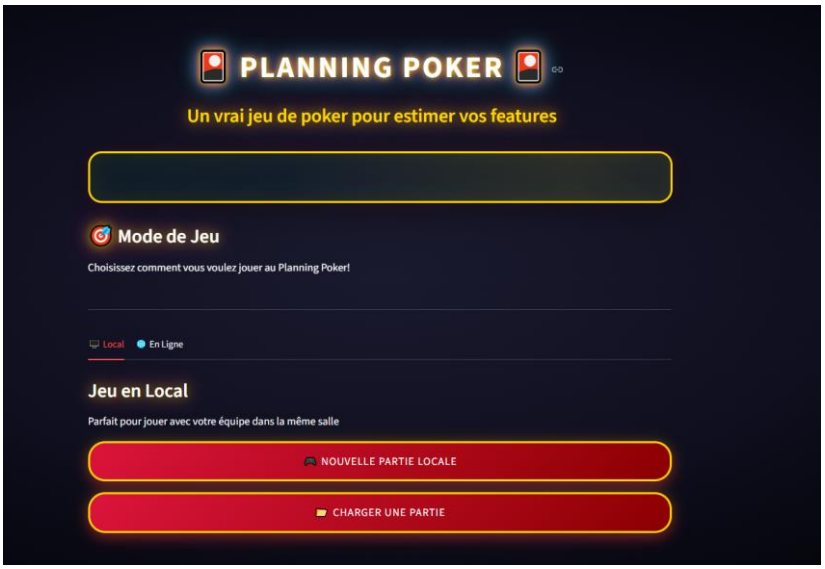
- lors du premier tour, l'unanimité est toujours requise ;
- à partir des tours suivants, si l'unanimité n'est pas atteinte, l'estimation est calculée automatiquement à partir de la moyenne ou de la médiane des votes.

Cette règle permet de conserver l'esprit collaboratif du Planning Poker tout en évitant des blocages excessifs.

3.2 Interface et déroulement :

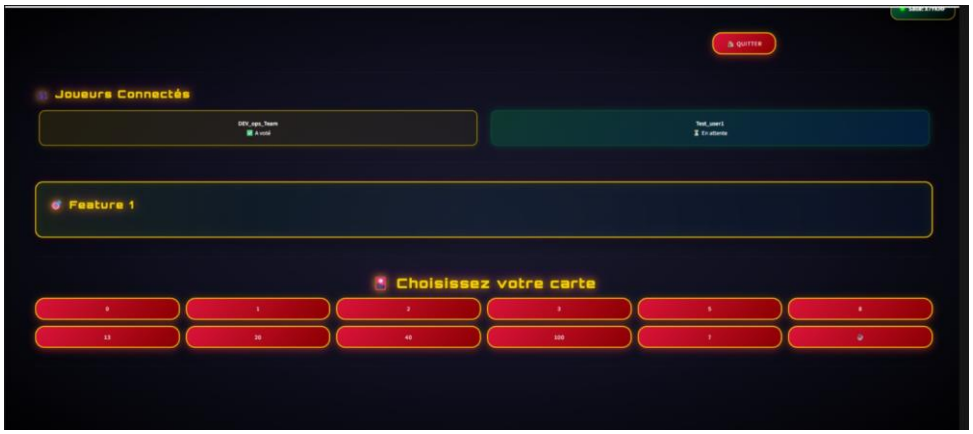
L'interface de l'application a été conçue pour être simple, intuitive et fluide. Le déroulement d'une session de Planning Poker suit les étapes suivantes :

1. Menu principal
   L'utilisateur choisit le mode de jeu (local ou en ligne) et peut démarrer une nouvelle partie ou charger une partie existante.



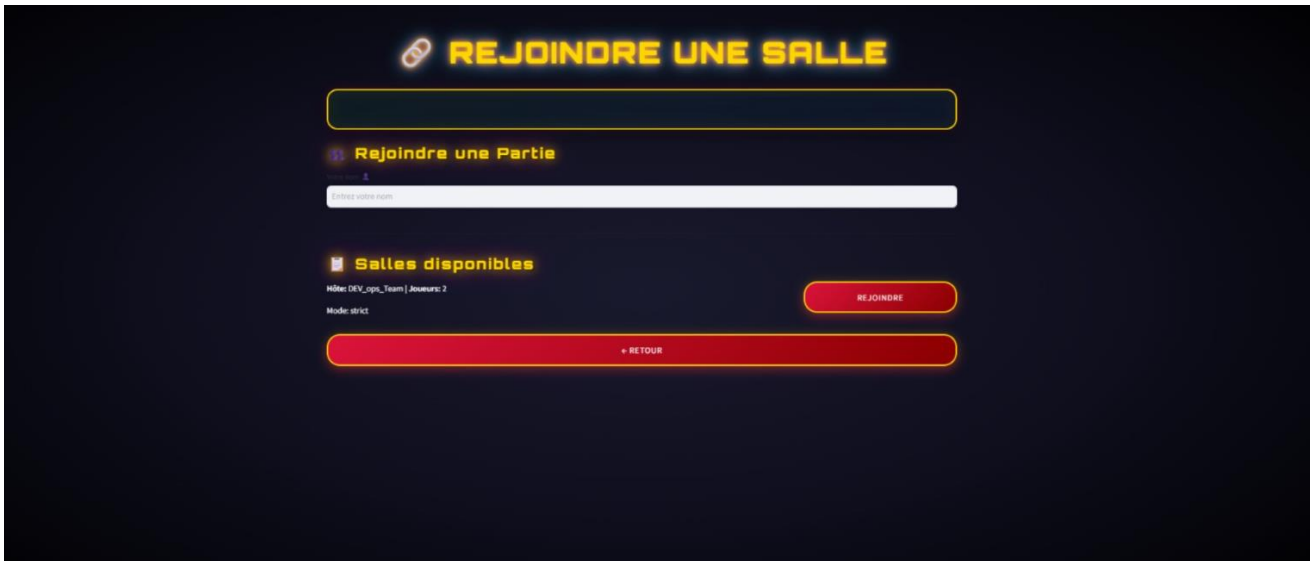
2. **Chargement du backlog**
   Le backlog de fonctionnalités peut être chargé à partir d'un fichier JSON ou créé manuellement via l'interface.

3. **Phase de vote**
   Pour chaque fonctionnalité, les joueurs sélectionnent une carte représentant leur estimation. L'état des votes est affiché en temps réel afin d'indiquer les joueurs ayant déjà voté.



4. **Révélation et validation**
   Une fois les votes complétés, les cartes sont révélées et l'estimation est validée selon le mode de vote choisi.

5. Rejoindre une salle :



4. Intégration Continue (CI)

4.1 Présentation générale du pipeline CI

Dans le cadre de ce projet, une chaîne d'intégration continue (CI) a été mise en place afin d'automatiser les étapes essentielles de vérification du code. Cette intégration continue repose sur **GitHub Actions**, un outil intégré à GitHub permettant d'exécuter automatiquement des workflows à chaque modification du dépôt.

L'objectif principal de ce pipeline est de garantir la **qualité du code**, de **détecter rapidement les erreurs**, et de **maintenir une documentation à jour**, tout en adoptant des pratiques professionnelles proches de celles utilisées en entreprise.

Le pipeline CI est déclenché automatiquement :

- lors d'un **push sur la branche principale (main)** ;
- lors d'une **Pull Request vers la branche main**.

Ainsi, aucune modification ne peut être intégrée sans passer par une vérification automatique.

4.2 Description détaillée du workflow CI

Le pipeline d'intégration continue est défini dans le fichier ci.yml. Il est composé de plusieurs étapes successives, exécutées sur un environnement Linux (Ubuntu). Ces étapes suivent un enchaînement logique classique :

Étape 1 : Récupération du code (Checkout)

La première étape consiste à récupérer le code source du projet à partir du dépôt GitHub. Cette opération est réalisée grâce à l'action actions/checkout, qui permet de cloner automatiquement le dépôt dans l'environnement d'exécution du workflow.

Cette étape est indispensable pour que les étapes suivantes puissent accéder au code, aux tests et à la documentation.

Étape 2 : Installation de l'environnement Python

Une fois le code récupéré, le pipeline configure un environnement Python stable à l'aide de l'action actions/setup-python. La version **Python 3.10** a été choisie afin d'assurer la compatibilité avec les bibliothèques utilisées dans le projet.

Cette étape garantit que le code est exécuté dans un environnement maîtrisé et reproductible.

Étape 3 : Installation des dépendances

Le pipeline installe ensuite l'ensemble des dépendances nécessaires au projet :

- les bibliothèques listées dans le fichier requirements.txt ;
- l'outil de test **pytest** ;
- l'outil de documentation **Sphinx**.

Cette automatisation permet de vérifier que le projet peut être installé et exécuté correctement sur une machine vierge, ce qui est essentiel pour la portabilité du projet.

4.3 Tests unitaires et validation de la logique métier

Une étape clé du pipeline CI est l'exécution automatique des **tests unitaires**. Ceux-ci sont lancés à l'aide de la commande pytest, qui parcourt le dossier tests/.

Les tests unitaires permettent de vérifier le bon fonctionnement de la **logique métier** de l'application Planning Poker, notamment :

- la gestion des votes des joueurs ;
- la validation des règles de vote (par exemple l'unanimité en mode strict) ;
- le calcul des résultats d'estimation selon les règles définies ;
- la cohérence de l'état de la partie (joueurs, votes, tours).

L'exécution automatique de ces tests garantit que toute modification du code ne casse pas les fonctionnalités existantes. Si un test échoue, le pipeline CI est interrompu et la modification est considérée comme invalide.

4.4 Génération automatique de la documentation

En complément des tests, le pipeline CI intègre une étape dédiée à la **génération automatique de la documentation** du projet.

Pour cela, l'outil **Sphinx** a été utilisé. Sphinx permet de générer une documentation HTML structurée à partir de fichiers texte rédigés au format reStructuredText (.rst).

- Le fichier conf.py contient les informations générales du projet, telles que le nom *Planning Poker* et l'auteur (équipe projet).
- Le fichier index.rst constitue la page principale de la documentation et présente le projet ainsi que ses fonctionnalités principales.

Lors de l'exécution du pipeline CI, la commande sphinx-build est lancée afin de générer automatiquement la documentation HTML. Cette étape permet de s'assurer que la documentation est toujours cohérente avec le code et qu'elle peut être générée sans erreur.

4.5 Apports de l'intégration continue au projet

La mise en place de l'intégration continue a apporté plusieurs bénéfices majeurs au projet :

- automatisation des vérifications du code ;
- amélioration de la fiabilité et de la qualité logicielle ;
- détection rapide des erreurs lors des Pull Requests ;
- génération systématique d'une documentation à jour ;
- adoption de pratiques professionnelles utilisées dans le développement logiciel moderne.