Rapport de Projet : Application Planning Poker

Conception agile de projets informatiques

Binôme :
Hossein ABDOLI
Maqrane ABDELAZIZ

M1 Informatique

19 Décembre 2025

Table des matières

1 Introduction et Contexte

Ce projet s'inscrit dans le cadre de l'unité d'enseignement de Conception Agile en Master 1 Informatique. L'objectif était de développer une application de bureau permettant de réaliser des sessions de "Planning Poker", une méthode d'estimation collaborative utilisée dans les équipes Scrum pour évaluer la complexité des tâches.

L'application permet à une équipe de développeurs de charger un backlog de tâches (User Stories), de voter pour estimer leur difficulté à l'aide de cartes virtuelles, et de converger vers un consensus selon des règles précises (unanimité, moyenne, etc.).

1.1 Organisation du travail

Le développement a été réalisé en binôme en suivant les principes du *Pair Programming*. Cette méthode nous a permis de :

- Réduire le nombre de bugs grâce à une revue de code instantanée.
- Partager la connaissance technique sur les bibliothèques utilisées (Tkinter, Firebase).
- Améliorer la conception architecturale en débattant des choix en temps réel.

2 Choix Techniques et Architecture

2.1 Langage et Frameworks

Pour répondre au besoin d'une application de bureau robuste et maintenable, nous avons effectué les choix suivants :

- **Langage : Python.** Ce choix s'est imposé pour sa rapidité de développement et sa syntaxe claire, idéale pour un cycle de développement Agile court. Sa riche bibliothèque standard a facilité la gestion des fichiers JSON et des structures de données.
- **Interface Graphique : Tkinter.** Utilisée via le module `UI.py`, Tkinter est la bibliothèque standard de Python pour les GUI. Elle est légère, ne nécessite pas d'installation complexe chez l'utilisateur final et suffit amplement pour afficher les cartes et les joueurs.
- **Persistance Distante : Firebase.** Pour la gestion des données en temps réel (si applicable) ou la synchronisation, nous avons intégré `firebase_db.py`. Cela offre une flexibilité supérieure à une base de données SQL classique pour des données non structurées comme des sessions de jeu.

2.2 Architecture Logicielle

L'application suit une architecture modulaire séparant la logique métier de l'interface utilisateur (proche du modèle MVC) :

- **Modèle (Métier) :** Les fichiers `card.py`, `deck.py`, `user.py`, `story.py` et `vote.py` encapsulent les entités du domaine. Ils ne contiennent aucune référence à l'interface graphique.
- **Vue et Contrôleur :** Le fichier `UI.py` gère l'affichage et les interactions utilisateurs, faisant le lien avec les objets métiers via `main.py`.

— **Persistance :** Les modules `persistence.py` et `firebase_db.py` isolent la logique de sauvegarde (JSON local et Cloud).

2.3 Modélisation

Le diagramme de classes ci-dessous illustre les relations entre les entités principales : `User`, `Story`, `Vote` et `Deck`.

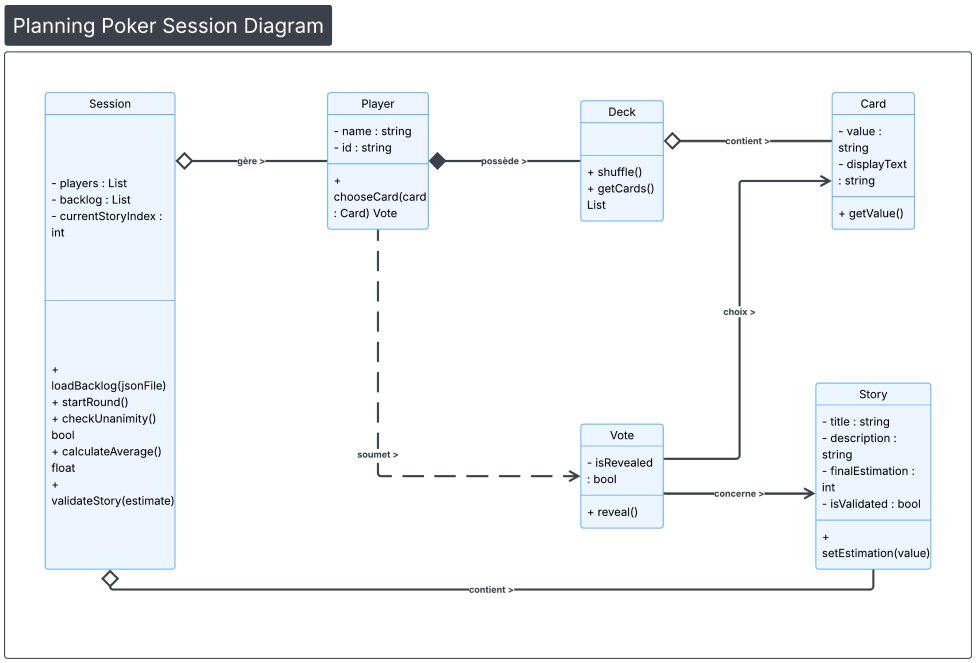

FIGURE 1 – Diagramme de classes simplifié du domaine

2.4 Gestion des données

La persistance est gérée de manière hybride :

1. **Backlog (JSON) :** Le fichier `backlog.json` stocke la liste des tâches à estimer. Structure simple : liste d'objets contenant un ID, une description et un titre.
2. **Sauvegarde Automatique :** L'état de la partie est sauvegardé localement pour permettre la reprise après une interruption (fonctionnalité "Pause Café").

3 Mise en place de l'Intégration Continue (CI)

L'automatisation du cycle de vie logiciel est assurée par **GitHub Actions**, configuré dans le dossier `.github/workflows`. Cette approche garantit que chaque modification du code est vérifiée avant d'être intégrée.

3.1 Workflow

Notre pipeline CI se déclenche à chaque `push` sur la branche `main`. Il comprend les étapes suivantes :

1. **Checkout :** Récupération de la dernière version du code source.

2. **Setup Python :** Installation de l'environnement Python (version 3.x).
3. **Installation des dépendances :** Exécution de `pip install -r requirements.txt` pour installer `pytest`, `firebase-admin`, etc.
4. **Tests Unitaires :** Lancement automatique des tests.
5. **Documentation :** Génération automatique de la documentation technique.

3.2 Tests Unitaires

Les tests sont implémentés avec le framework **pytest** (dossier `tests/`). Ils couvrent la logique métier critique :

— Validation des règles de vote (Unanimité vs Moyenne).
— Calcul des résultats des votes.
— Intégrité des objets `Card` et `Deck`.

La commande exécutée par la CI est :

```
pytest tests/
```

3.3 Génération de Documentation

La documentation technique est générée automatiquement via **Doxygen**, configuré grâce au fichier `Doxyfile` présent à la racine. Elle permet de générer un site statique décrivant les classes et méthodes, facilitant la reprise du projet par d'autres développeurs.

4 Manuel Utilisateur et Fonctionnalités

4.1 Installation et Lancement

Pour exécuter l'application sur un nouvel environnement, les étapes sont les suivantes :

```
# 1. Cloner le d p t
git clone https://github.com/ABDOLI-Hossein/planningPokerAgile.git
cd planningPokerAgile

# 2. Installer les d pendances
pip install -r requirements.txt

# 3. Lancer l'application
python main.py
```

Pour lancer la suite de tests manuellement :

```
pytest tests/
```

4.2 Modes de Jeu

L'application supporte deux modes de calcul pour l'estimation, configurés dans `rules.py` :

— **Mode Strict (Unanimité) :** Tous les joueurs doivent révéler la même carte pour valider l'estimation. En cas de désaccord, un nouveau tour de vote est lancé.
— **Mode Moyenne :** Si l'unanimité n'est pas atteinte après le premier tour, la moyenne des valeurs des cartes est calculée pour proposer une estimation finale.

4.3 Interface du Manager et Gestion de Partie

La figure ci-dessous illustre l'interface principale dédiée au rôle de "Manager" (Scrum Master). Cette vue centralise les outils nécessaires pour configurer et animer une session de vote.

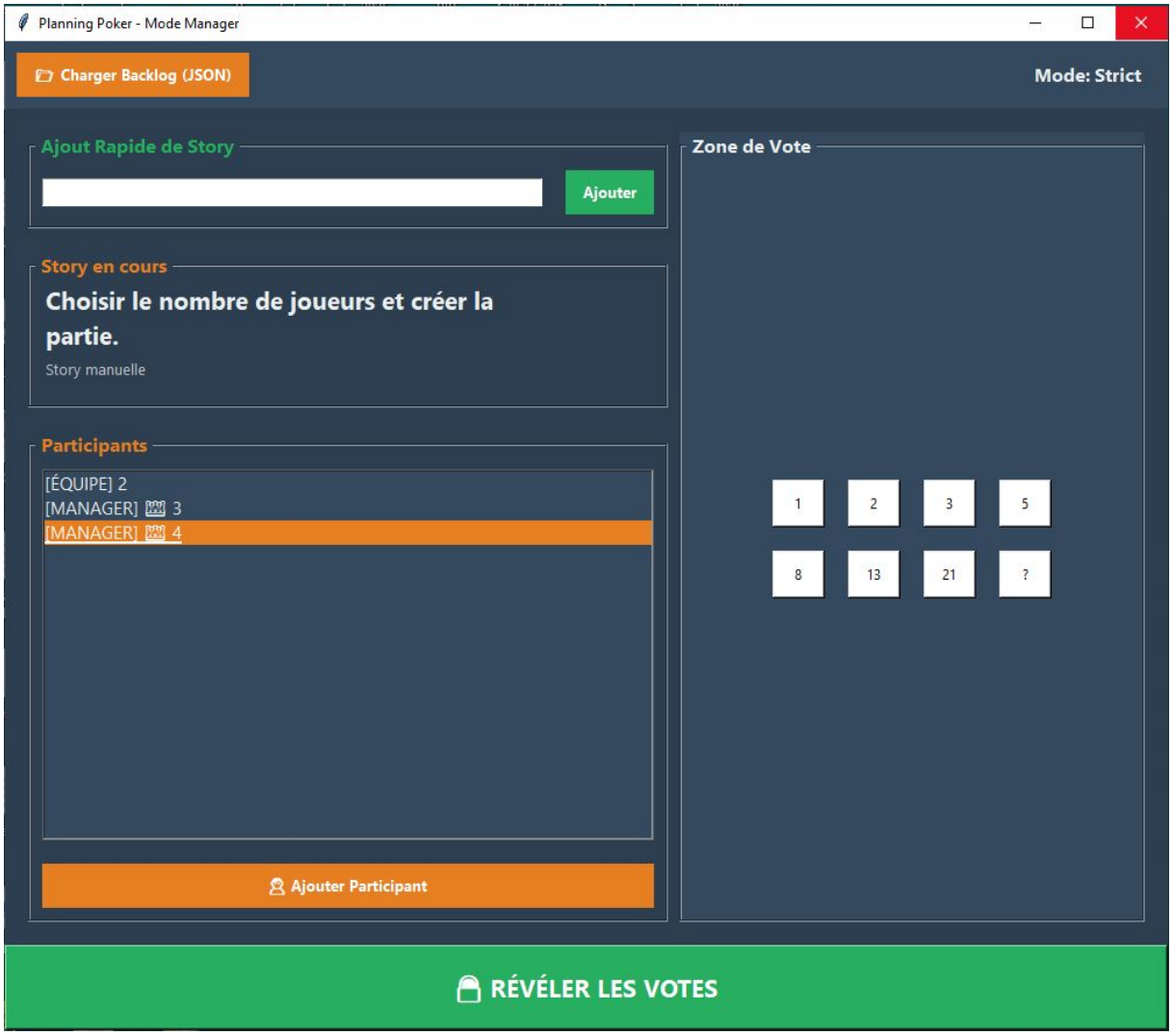

FIGURE 2 – Vue de l'interface Manager en cours de configuration de partie.

Les principaux éléments de cette interface sont :

- **Gestion des Stories :** Un bouton en haut à gauche permet de "Charger un Backlog (JSON)". Une zone "Ajout Rapide de Story" permet également d'insérer une tâche manuellement.
- **Zone de Vote :** Située à droite, elle présente les cartes de vote disponibles pour les participants, suivant ici la suite de Fibonacci modifiée (1, 2, 3, 5, 8, 13, 21, ?).
- **Gestion des Participants :** Le panneau de gauche liste les membres connectés avec leur rôle (ex : [MANAGER], [ÉQUIPE]) et un bouton permet d'ajouter manuellement un participant si nécessaire.
- **Contrôle de la Session :** Le bouton principal en bas, "RÉVÉLER LES VOTES", permet au Manager de clôturer le tour de vote et d'afficher les résultats pour chercher un consensus.

Annexe : Aperçu de la Base de Données (Firebase)

Cette section présente la structure de la base de données NoSQL (Cloud Firestore) utilisée pour l'application "Planning Poker". L'accès à la console d'administration étant restreint, ces captures d'écran attestent de la bonne structuration des données et du fonctionnement du backend en temps réel.

1. Collection `users` (Gestion des utilisateurs)

Cette collection assure la gestion des profils utilisateurs. Chaque document correspond à un participant unique et contient ses informations clés telles que son nom, son rôle (ex : "Manager") et sa date d'inscription.

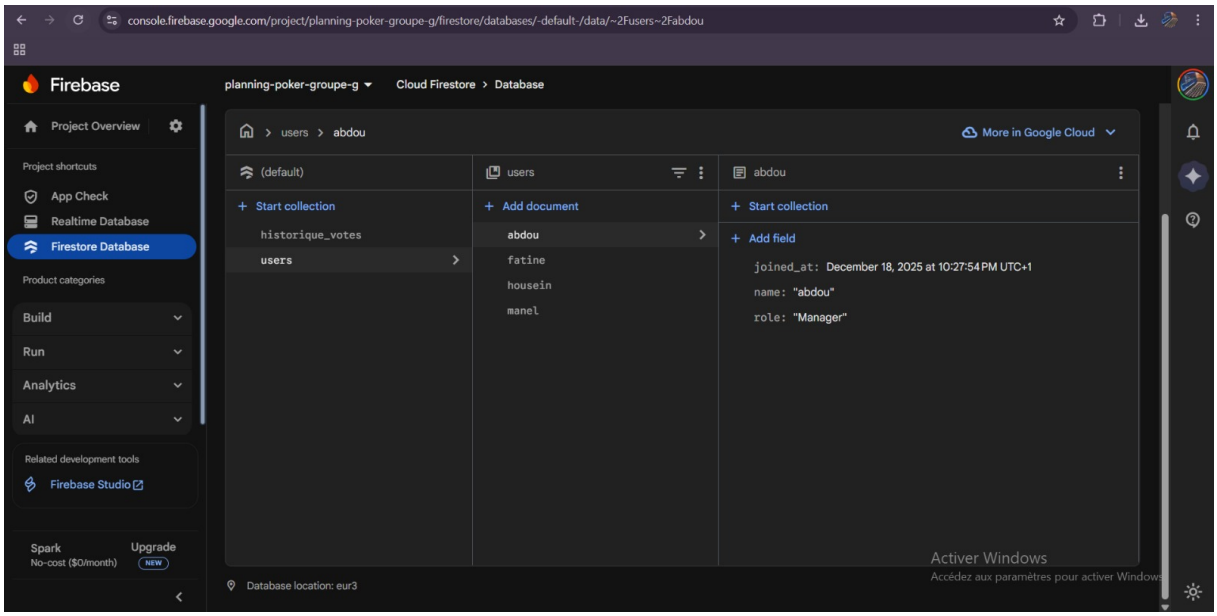

FIGURE 3 – Interface Firestore : Collection `users`. Détail du document de l'utilisateur "abdou".

2. Collection `historique_votes` (Persistance des sessions)

Cette collection stocke l'historique des parties terminées. La structure hiérarchique permet de sauvegarder pour chaque "Story" votée :

- La liste des participants présents ;
- Le détail des votes individuels (via la map `details_votes`) ;
- L'estimation finale validée par l'équipe.

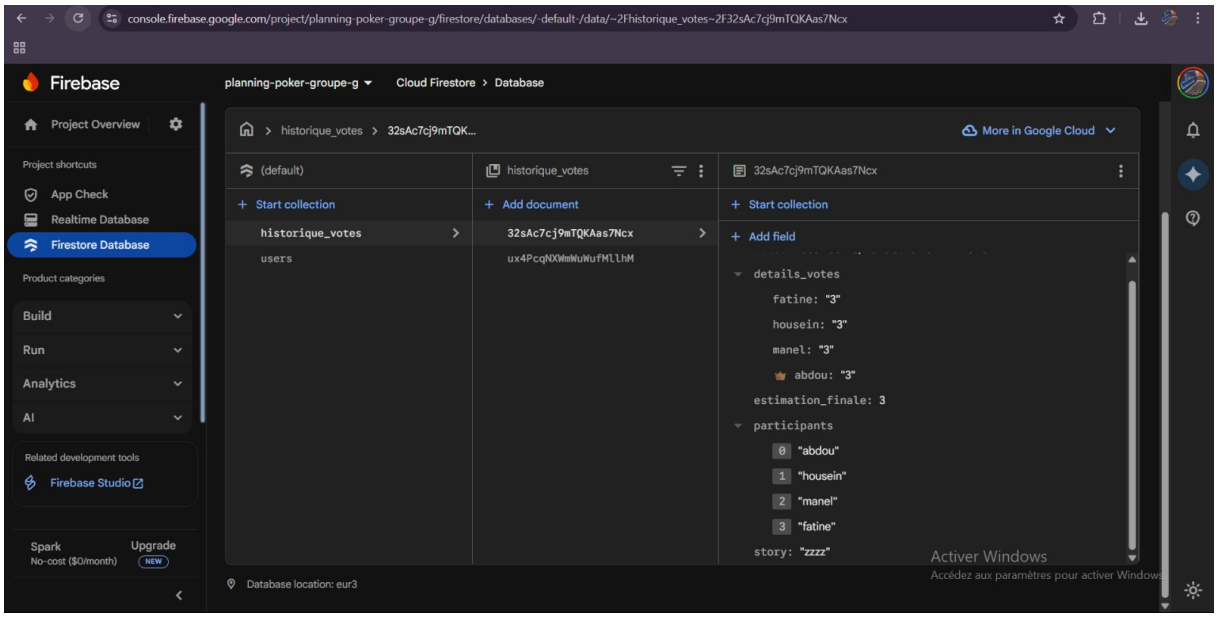

FIGURE 4 – Interface Firestore : Collection `historique_votes`. Vue des votes par membre et de l'estimation finale.