

Rapport de projet

Conception Agile - Application Planning Poker

IMANSOUREN Selma ~ LYONNET Andréa

Introduction

Ce projet s'inscrit dans le cadre du module Conception Agile du Master 1 Informatique.

L'objectif est de concevoir et développer une application de Planning Poker, un outil collaboratif utilisé lors des estimations agiles de projet . L'application permet à plusieurs participants de rejoindre une session, de voter pour estimer la complexité d'une User Story, puis de visualiser les résultats de façon collective.

1. Choix Techniques

Nous avons choisi Python pour l'application, Flask pour le framework, HTTP pour la partie serveur et Flask SocketIO pour la communication en temps réel.

1. Python est rapide à prototyper, adapté aux projets agiles et c'est le langage que nous maîtrisons le mieux pour coder une grosse application
2. Flask est léger, flexible grâce à l'utilisation des Sockets plutôt qu'un backend lourd. De plus j'avais (Andréa) déjà une connaissance de FLASK car je l'avais utiliser en stage donc j'étais à l'aise à l'idée de l'utiliser
3. Flask-SocketIO simplifie la gestion des sessions en temps réel (connexion, déconnexion, vote, notification de résultats...)
4. Utilisation de JSON pour stocker les User Stories : un format simple et lisible.

Pour la partie serveur en mode hors réseau local nous avons opté pour un hébergeur, cette partie nous a posé bien des soucis car les options n'était souvent pas compatible avec flask socketIO ((ngrok par exemple). Le choix de l'hébergeur [Railway](#)

nous a permis de lier l'hébergeur à github et d'obtenir une url accessible hors réseau local

2. Architecture

Pour l'architecture notre application suit un modèle qui se rapproche plus au moins du modèle MVC

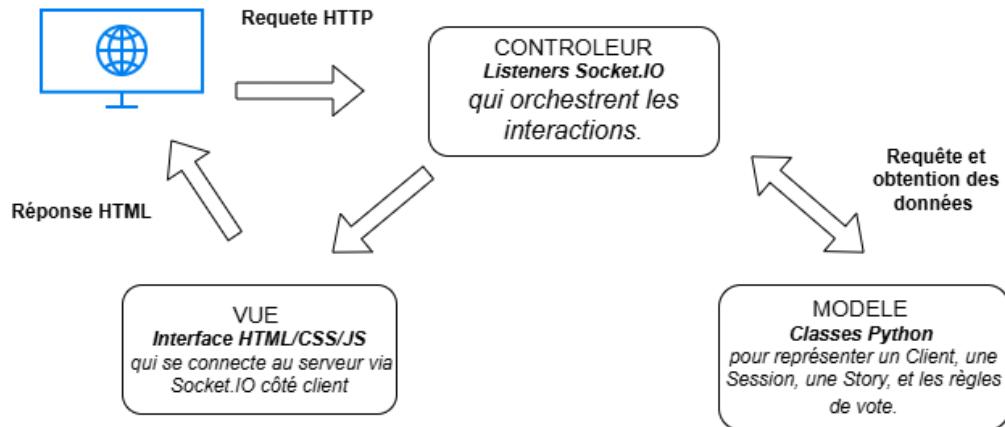


diagramme de classe à faire ici

3. Intégration Continue

Utilisation de GitHub Actions, qui permet d'exécuter automatiquement des workflows à chaque push.

Notre pipeline contient 4 étapes :

1. Checkout
 - Récupérer le code du dépôt.
 - Prépare les fichiers nécessaires au pipeline.
2. Installation des dépendances
 - pip install -r requirements.txt
 - Installation de Flask, Flask-SocketIO, pytest, Sphinx...
3. Exécution des tests unitaires
 - Lancement automatique - pytest.
 - Tests sur :
 - le calcul des moyennes,
 - la validation des votes,
 - les règles d'unanimité selon le mode choisi.
4. Génération de la documentation
 - Générée - Sphinx.
 - Build automatique dans /docs/_build/html.
 - Publication automatique via GitHub Pages.

4. Test Unitaire

Nous utilisons **pytest** comme framework de tests unitaires. Pytest détecte automatiquement les fichiers dont le nom commence par **test_** (ex : test_route.py, test_socket_join.py) la commande pour tester étant **pytest -q**

pytest :

1. importe l'application (ici planning_poker_app),
2. exécute chaque fonction de test,
3. indique le résultat (succès/échec), avec un détail en cas d'erreur.

Fichier **test_route.py** : tests des routes HTTP Flask

Ce fichier valide le bon fonctionnement de la route principale (/) servie par Flask.

1) **test_home_status_code()**

Objectif : vérifier que la page d'accueil est bien accessible.

- on instancie un client HTTP de test via app.test_client(),
- on simule une requête GET vers /,
- on vérifie que le serveur répond avec un code 200 OK.

Pourquoi c'est important :

- cela garantit que l'application démarre correctement,
- et que la route existe et répond sans erreur serveur.

```
import os
import sys

CURRENT_DIR = os.path.dirname(__file__)
PROJECT_ROOT = os.path.dirname(CURRENT_DIR)
if PROJECT_ROOT not in sys.path:
    sys.path.insert(0, PROJECT_ROOT)

from planning_poker_app import app

def test_home_status_code():
    client = app.test_client()
    resp = client.get("/")
    assert resp.status_code == 200

def test_home_contains_title():
    client = app.test_client()
    resp = client.get("/")
    html = resp.data.decode("utf-8")
    assert "Planning Poker" in html
```

2) **test_home_contains_title()**

Objectif : vérifier que le contenu HTML retourné correspond bien à la page attendue.

- on refait un GET sur /,
- on récupère le HTML via resp.data.decode("utf-8"),
- puis on vérifie la présence d'un texte clé : "Planning Poker".

Pourquoi c'est important :

- Un statut 200 ne garantit pas que la bonne page est affichée (on pourrait renvoyer une page vide),
- Ce test valide un élément fonctionnel visible côté utilisateur.

Fichier **test_socket_join.py** : tests Socket.IO (temps réel)

L'application Planning Poker utilise **Socket.IO** pour la communication en temps réel (connexion, join de session, votes...).

Les tests Socket.IO ne passent pas par des routes HTTP classiques, mais par l'envoi/réception **d'évènements**.

1) **test_socket_connects()**

Objectif : vérifier qu'un client peut se connecter au serveur Socket.IO.

- on crée un client de test Socket.IO avec socketio.test_client(app),
- on vérifie qu'il est connecté via client.is_connected(),
- puis on le déconnecte.

Pourquoi c'est important :

- Cela confirme que l'application est correctement configurée (Socket.IO initialisé, namespace par défaut fonctionnel).

```
import os
import sys

CURRENT_DIR = os.path.dirname(__file__)
PROJECT_ROOT = os.path.dirname(CURRENT_DIR)
if PROJECT_ROOT not in sys.path:
    sys.path.insert(0, PROJECT_ROOT)

from planning_poker_app import app, socketio


def test_socket_connects():
    client = socketio.test_client(app)
    assert client.is_connected()
    client.disconnect()

def test_join_session_emits_something():
    client = socketio.test_client(app)
    assert client.is_connected()

    client.emit("join_session", {
        "sessionId": "1234",
        "playerName": "Alice"
    })

    received = client.get_received()
    assert len(received) > 0

    client.disconnect()
```

2) **test_join_session_emits_something()**

Objectif : tester le comportement “connexion à une session” côté Socket.IO (join).

- on connecte un client de test,
- on émet un évènement côté client (ex : "join_session") avec un payload :
 - sessionId
 - playerName

- puis on récupère les messages renvoyés par le serveur via `client.get_received()`,
- et on vérifie qu'au moins une réponse serveur a été reçue.

Pourquoi c'est important :

- cela valide que le serveur reçoit l'évènement,
- exécute le handler correspondant,
- et renvoie une réponse (confirmation, erreur, état de session, etc.).
- c'est une base solide pour éviter les régressions lors des évolutions (modification du format des données, renommage d'évènement, etc.).

Remarque :

- ce test ne vérifie pas encore le contenu exact de la réponse (ex : `success=True`, `playerId`, etc.).
Il garantit déjà l'essentiel : *l'évènement est bien traité et une réponse est produite.*

Résultat

L'exécution pytest confirme que :

- la page d'accueil répond correctement en HTTP (Flask),
- les fonctionnalités temps réel de base fonctionnent (Socket.IO),
- et que les mécanismes critiques (accès à l'application + connexion temps réel + join) sont couverts par des tests reproductibles.

Guide utilisateur – Planning Poker

Pour tester le jeu : lancer le serveur **python planning_poker_app.py** puis ouvrir dans le navigateur : <http://127.0.0.1:5000/>

Pour jouer à plusieurs avec différents membres du groupe qui ne sont pas sur le même réseau local : <https://web-production-8451.up.railway.app/>

Objectif du Jeu

Le Planning Poker est un outil d'estimation agile utilisé en équipe pour évaluer la complexité des user stories.

Chaque joueur choisit une carte représentant son estimation, puis les votes sont révélés simultanément.

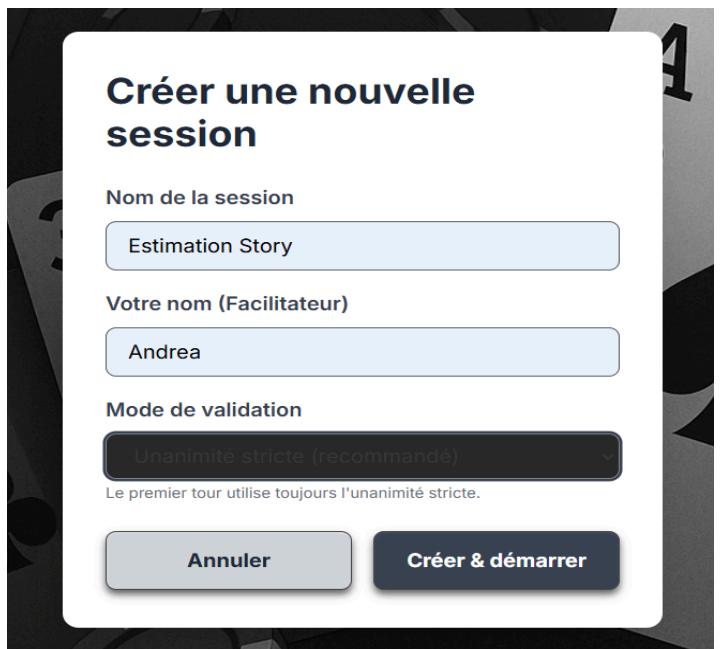
Connexion au Jeu

Cas 1 – Vous êtes le Facilitateur

Vous êtes la personne qui crée la session et gère les votes.

1. Ouvrez le site du Planning Poker (URL locale ou publique).
2. Sur l'écran d'accueil, cliquez sur : « **Créer une Session** »
3. Remplissez :

- Nom de la session
- Votre pseudo (Facilitateur)
- Le mode d'estimation (Unanimité, Moyenne, Médiane)



4. Cliquez sur « **Créer & Démarrer** ».
5. Un code de session à 4 chiffres apparaît (ex : 4821).
 - Ce code doit être communiqué aux joueurs pour qu'ils rejoignent la même session.
6. Vous entrez automatiquement dans l'interface de session.

Cas 2 – Vous êtes un Joueur

Vous rejoignez une session déjà créée.

1. Sur la page d'accueil, cliquez sur :
« Rejoindre une Session »
2. Entrez :
 - Le code de session communiqué par le facilitateur
 - Votre pseudo
3. Cliquez sur « **Rejoindre** ».
4. Vous accédez à la session en temps réel avec les autres joueurs.

Rejoindre une session

Code de session (ID)

6930

Votre pseudo

selma

Annuler Rejoindre

Session Planning Poker : Estimation Story

ID: 6930 • Mode: strict

US-01 - Affichage de la page d'accueil

En tant qu'utilisateur, je veux voir une page d'accueil simple et attrayante.
Priorité : Haute
État : Discussion / préparation

selma a rejoint la session.

Participants (max 5) (2)

Andrea Facilitateur

selma Vous

Déroulement de la Partie

Étape 1 — Choix de la User Story

Le facilitateur sélectionne la story dans le backlog.

Les joueurs voient automatiquement la story affichée.

Étape 2 — Discussion (facultatif)

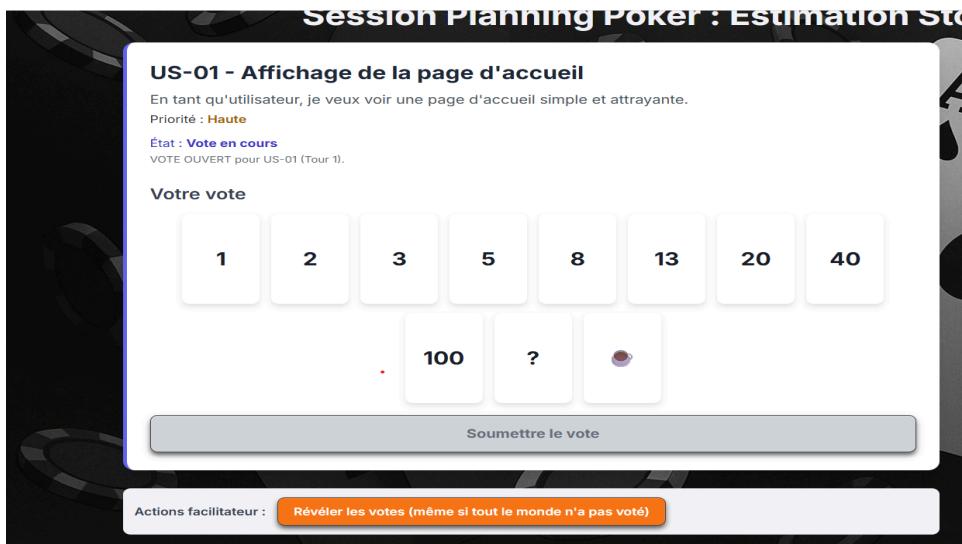
L'équipe peut discuter à voix haute (en appel ou face à face) ou via le chat intégré :

💬 Le chat permet à toute l'équipe d'échanger en temps réel.
Chaque message apparaît instantanément chez tous les joueurs.

Étape 3 — Lancement du Vote

Le facilitateur clique sur « **Ouvrir le vote** »

Les cartes apparaissent maintenant pour chaque joueur.



Étape 4 — Les joueurs votent

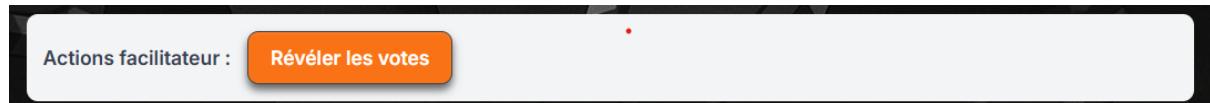
Chaque joueur :

1. Choisit une carte (ex : 3, 5, 8, 13, ?, 🍵)
2. Cliquez sur « **Soumettre le vote** »

Un indicateur A voté confirme la prise en compte.

Étape 5 — Révélation des Votes

Quand tout le monde a voté (ou même avant si nécessaire),
le facilitateur clique sur « **Révéler les votes** »



Tous les votes apparaissent en même temps pour tout le monde.

Si l'équipe est unanime → 🎉 Consensus immédiat
Un effet de confettis peut apparaître

US-01 - Affichage de la page d'accueil

En tant qu'utilisateur, je veux voir une page d'accueil simple et attrayante.

Priorité : **Haute**

État : **Story validée**

CONSENSUS : estimation 2 (mode strict).

Résultats révélés — Pas d'unanimité, discutez ensemble.

Moyenne undefined	Médiane undefined	Min undefined	Max undefined
-----------------------------	-----------------------------	-------------------------	-------------------------

Distribution des votes

Aucun vote.

Étape 6 — Discussion en cas de désaccord

Si les votes diffèrent, une bannière s'affiche :

« 💬 Nous ne sommes pas d'accord. Discutons pour trouver un consensus ! »

L'équipe discute via le chat ou la voix.

Le facilitateur peut ensuite **Saisir & valider l'estimation finale**
en entrant le chiffre retenu après discussion.

Discussion en direct

Andrea 16:14:03
je pense qu'il faut se baser sur la moyenne

selma 16:14:33
oui 50 alors

Écrivez un message... **Envoyer**

Étape 7 — Validation de la Story

Le facilitateur valide l'estimation « **Valider l'estimation** »

La story passe en vert dans le backlog.

La story suivante est automatiquement sélectionnée.

burnoused-jimmie-mussily.ngrok-free.dev indique

Saisir l'estimation finale.

Suggestions: moyenne=54, médiane=54

50

OK

Annuler

Fin de la Session

Une fois toutes les stories estimées, le facilitateur peut cliquer sur « **Clôturer la Session** »

Backlog (5/5 estimés)

- US-01: Affichage de la page d'accueil (Est. 2) Choisir
- US-02: Ajouter au panier (Est. 40) Choisir
- US-03: Filtrage avancé des produits (Est. 40) Choisir
- US-04: Passer au paiement (Est. 3) Choisir
- US-05: Historique des commandes (Est. 13) Courante

Clôturer la session

burnoused-jimmie-mussily.ngrok-free.dev indique

Clôturer la session pour tous les participants ?

OK Annuler

```
"participants": { "d571d173-d547-43cd-8b6a-c2e608fdc235": "Andrea", "4927073f-373c-4a12-8ac6-3506ffbe11a9": "selma" }, "stories_estimated": [ { "id": "US-01", "title": "Affichage de la page d'accueil", "estimate": 2, "rounds": 1 }, { "id": "US-02", "title": "Ajouter au panier", "estimate": 40, "rounds": 1 }, { "id": "US-03", "title": "Filtrage avancé des produits", "estimate": 40, "rounds": 1 }, { "id": "US-04", "title": "Passer au paiement", "estimate": 3, "rounds": 1 }, { "id": "US-05", "title": "Historique des commandes", "estimate": 13, "rounds": 1 } ]
```

Le système génère un fichier JSON exportable contenant :

- Les stories
- Les votes
- Les estimations finales
- Le nombre de tours
- Les participants

Si vous rechargez la page ou perdez la connexion, vous êtes automatiquement reconnecté à la session en cours.

Arborescence du Projet *Planning Poker*

```
planning-poker/
├── .git/           → Données internes Git
├── .github/        → Workflows GitHub Actions
├── .venv/          → Environnement virtuel Python
├── docs/           → Documentation interne (MkDocs)
├── site/           → Version statique générée par MkDocs
├── static/          → Fichiers front-end : JS, CSS, images
├── templates/       → Templates HTML Jinja2
├── tests /          → Test unitaire des routes et des connexions
├── mkdocs.yml       → Configuration du site de documentation
├── planning_poker_app.py   → Application Flask + Socket.IO
├── README.md        → Présentation + instructions d'installation
└── requirements.txt  → Dépendances Python
```

planning_poker_app.py

- Cœur de l'application.
- Gère les sessions, les votes, le chat en temps réel, les stories, les sockets.
- Sert les pages HTML + met à jour les clients en direct.

templates/

- Pages HTML rendues par Flask.
- Inclut la page principale contenant <div id="app"></div> où l'interface dynamique est injectée.

static/

- **js/app.js** : toute la logique du front-end (interface, actions, WebSocket, chat).
- **css/** : styles personnalisés.
- fonds d'écran.

docs/ & mkdocs.yml

- Documentation rédigée avec MkDocs.
- Permet de générer un mini-site explicatif du projet sur github actions

site/

- Version générée automatiquement de la documentation.

README.md

- Présentation du projet
- Comment installer, lancer, tester et contribuer.

requirements.txt

- Liste des librairies Python nécessaires : Flask, Flask-SocketIO, eventlet, etc.

.git/ et .github/

- Contient l'historique Git et les workflows d'automatisation.

CONCLUSION

Ce projet nous a permis de nous mettre dans la peau de véritables chefs de projet en concevant et développant une application de Planning Poker pleinement fonctionnelle, répondant à un besoin réel dans un contexte de gestion de projet agile. L'application développée pourrait concrètement être utilisée dans un cadre professionnel ou académique (*à la place du jeu de carte fait en cours par exemple*).

La réalisation de ce projet nous a permis de découvrir et d'approfondir l'utilisation d'un **serveur web**, dans notre cas avec le framework **Flask**, ainsi que la mise en place de communications en temps réel via Socket.IO. Nous avons également appris à mettre en œuvre des **tests unitaires** avec **pytest** et à générer une **documentation automatique**, contribuant à la qualité et à la maintenabilité du code.

Par ailleurs, le projet nous a confrontés à des problématiques concrètes liées au **réseau et à l'hébergement**, notamment la distinction entre fonctionnement en réseau local et accès hors réseau. Ces aspects ont représenté la principale difficulté du projet, mais ont été particulièrement formateurs.

Nous sommes fiers du travail réalisé et des compétences acquises tout au long de ce projet, et espérons que cette application saura répondre aux attentes et vous plaira.