

Conception agile de projets informatiques

Rapport de projet

Professeur : Valentin Lachand-Pascal

Nicolas Tran et Hubert Geoffray
20/12/2024

Table des matières

Introduction.....	2
Présentation globale	2
Justification des choix techniques	6
Architecture	6
Choix langages	6
Base de données	7
Intégration continue.....	7
Tests unitaires	7
Documentation	8
Axes d'amélioration	9
Expérience utilisateur	9
Expérience développeur	10
Conclusion	10

Introduction

Dans le cadre de ce cours de conception agile de projets informatiques, nous avons développé une application web de planning poker permettant à des groupes d'utilisateurs d'évaluer un ensemble des tâches en fonction de leurs propres critères, tout en les proposant eux-mêmes dans le cadre de projets collaboratifs. Dès le début, nous avons décidé d'implémenter le multijoueur pour permettre aux utilisateurs de participer simultanément. Nous avons opté pour une architecture simple et des choix répondant aux besoins spécifiques du projet.

Présentation globale

La première page de l'application est la page d'inscription qui se présente comme ceci :

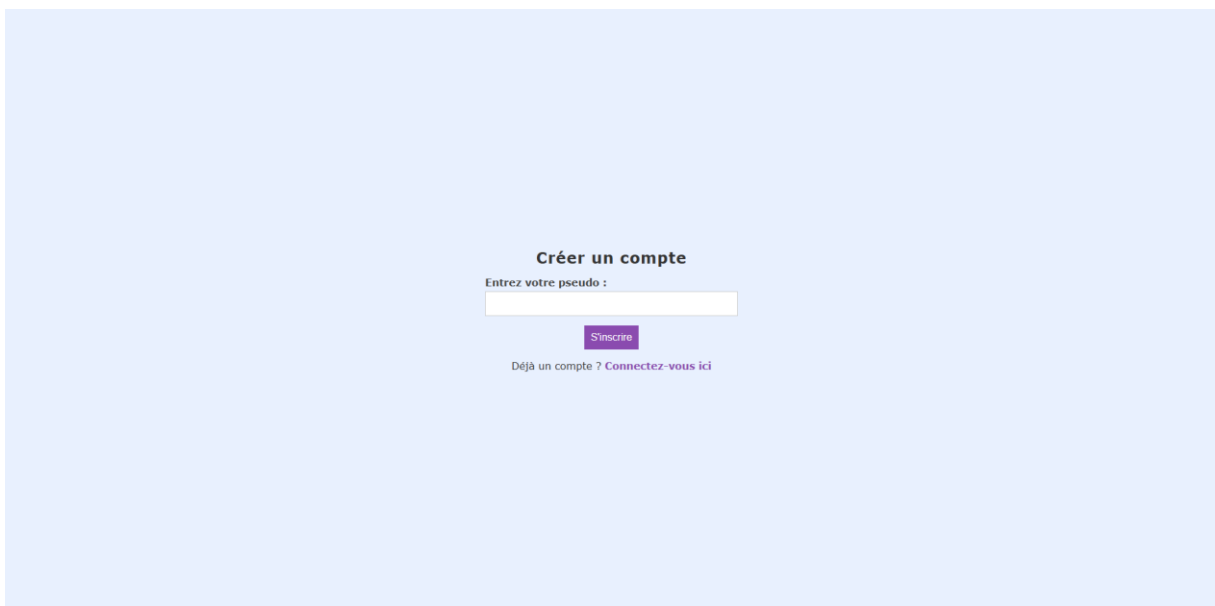


Figure 1: Page d'inscription

L'utilisateur doit simplement entrer un pseudo pour s'inscrire et se connecter automatiquement par la suite. Dans le cas où un utilisateur a déjà un compte, il peut accéder à la page de connexion en cliquant sur le lien « Connectez-vous ici ».

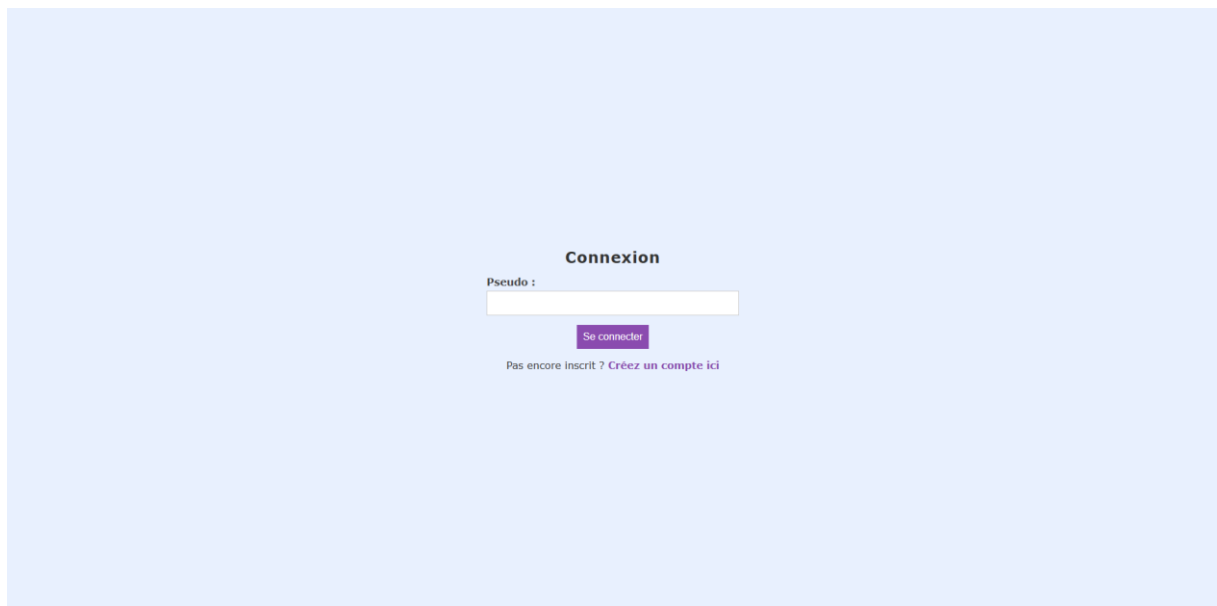


Figure 2 : Page de connexion

La page de connexion est similaire à la page d'inscription. À noter que le système de mot de passe n'a été intégré, car nous souhaitons simplifier l'accès à notre application grâce à un pseudo unique.

Une fois connecté, l'utilisateur est redigéré au dashboard ou le menu principal. C'est sur cette page que l'utilisateur peut créer une partie en choisissant le mode de jeu et le nombre de joueurs. Il peut également rejoindre une partie avec un ID valide ou reprendre une partie à l'aide d'un fichier JSON.

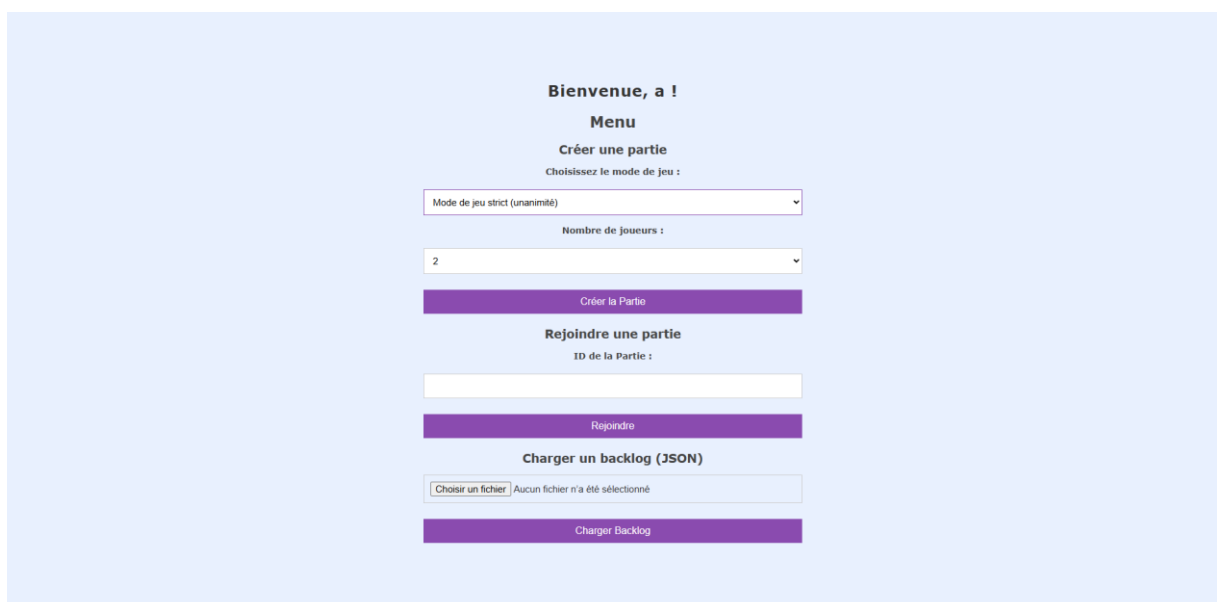


Figure 3 : Menu principal

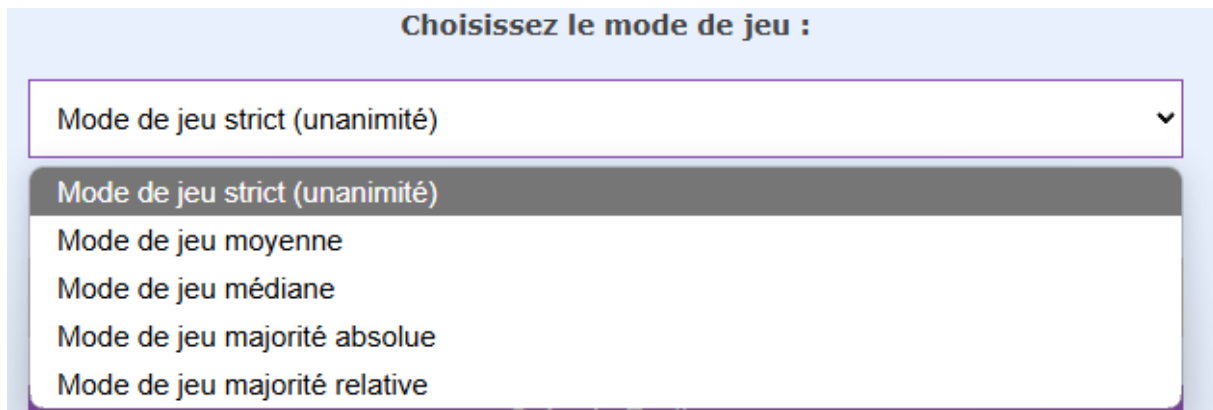


Figure 4 : Liste déroulante choix mode de jeu

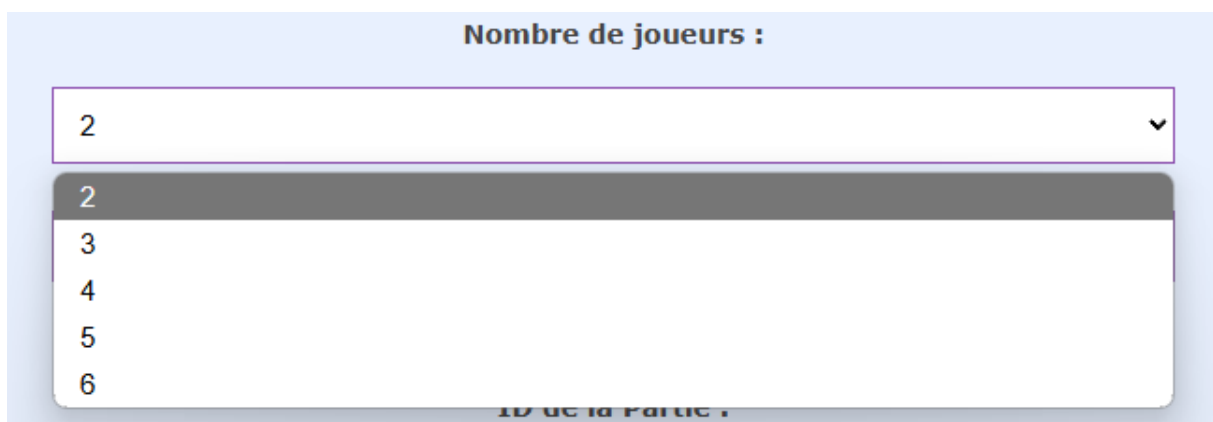


Figure 5 : Liste déroulante choix du nombre de joueurs

La partie est lancée (figure 6), la liste des fonctionnalités de la salle de jeu est la suivante :

- Affichage du mode de jeu (Moyenne)
- Affichage de l'ID de la partie (5040)
- Affichage des joueurs dans la partie en précisant l'host (a (host) et z)
- Affichage de la liste des problèmes et leurs résultats (Construire un pont : résultat moyenne : 20 ; Construire une école : votes en cours a : ? | z : ?)
- Bouton pour ajouter un problème
- Bouton pour voter un problème sélectionné (uniquement l'host)
- Affichage des votes des joueurs en direct (votes masqués)
- Affichage des cartes à voter
- Bouton pour dévoiler les votes (uniquement l'host)
- Affichage des votes suite à la révélation des votes. Si le problème n'a pas encore de résultat (par exemple, si le résultat n'est pas unanime au premier tour : a : 5 et z : 20), l'application demande de revoter une deuxième fois pour calculer la moyenne.
- Bouton pour terminer la partie (uniquement l'host)
- Dans tous les modes de jeu, si la totalité des joueurs votent la carte « café », la partie s'enregistre dans un fichier JSON et termine la partie.

Mode de Jeu : Moyenne

ID de la partie :
5040

Joueurs dans la Partie

a (host) _____

z _____

Problèmes

Voter pour ce problème

Voter pour ce problème

+ Ajouter un problème

Construire un pont

Construire une école

Résultat moyenne : 20

Votes : a : ? | z : ?

Voter pour le problème sélectionné : Construire une école

0

1

2

3

5

8

13

20

40

100

?

☕

Déposer vote

Terminer la partie

Figure 6 : Salle de jeu

Un squelette du site web réalisé sur draw.io montre ses différents chemins et fonctionnalités.

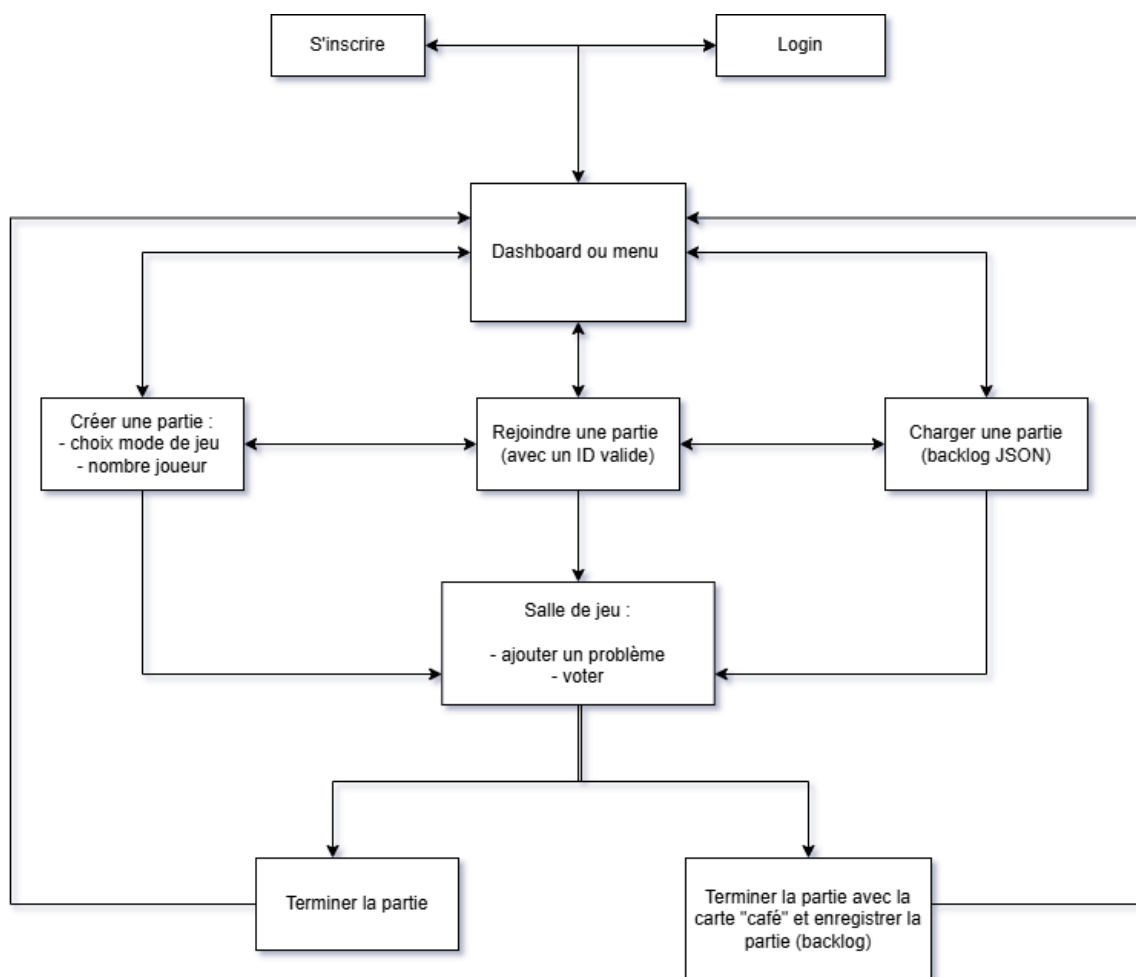


Figure 7: Squelette de l'application

Pour conclure cette présentation globale, cette application se distingue par sa simplicité, offrant une interface claire et épurée qui favorise une expérience utilisateur agréable. L'accès facilité, nécessitant uniquement la saisie d'un pseudo, contribue à une prise en main rapide et intuitive.

Justification des choix techniques

Architecture

L'architecture adoptée pour ce projet est de type client-serveur, qui est une approche commune dans les applications web. Cela permet de séparer la logistique métier, centralisée au niveau du serveur, de l'interface utilisateur, accessible via un navigateur. Pour mettre en œuvre cette architecture, nous avons choisi de travailler avec le Framework *Flask*. Ce choix nous paraissait évident pour plusieurs raisons :

- Une expérience préalable au sein de l'équipe : Au sein de notre binôme, nous étions déjà familiers avec Flask, ce qui nous a permis de nous concentrer rapidement sur la conception et le développement des fonctionnalités principales, optimisant ainsi notre temps de travail
- Sa simplicité : Flask offre une structure légère qui évite la complexité inutile tout en restant assez robuste pour supporter une application web comme la nôtre. Cela nous a permis de personnaliser facilement les composants selon nos besoins spécifiques sans être restreints.
- Un environnement modulaire : Flask dispose de nombreuses extensions qui nous ont facilité la tâche. Parmi elles, Flask-SocketIO nous a permis d'actualiser les pages web de chaque utilisateur en temps réel à chaque modification de celle-ci. Par exemple, dans la page « game_room.html », cette extension actualise automatiquement l'interface de chaque joueur lorsqu'un nouveau problème est soumis ou lorsqu'un nouveau joueur rejoint la partie. Cela garantit ainsi une synchronisation fluide entre les collaborateurs.

Choix langages

En complément de cette architecture backend basée sur Flask, le développement de l'interface utilisateur a été réalisé en utilisant les technologies standards *HTML*, *CSS* et *JavaScript*. Avec l'intégration des templates *Jinja2* dans Flask, nous avons pu générer dynamiquement les pages en fonction des données utilisateur. Par exemple, dans « game_room.html », le numéro de la partie ainsi que son mode de jeu sont générés dynamiquement grâce aux balises « {{ game_id }} » et « {{ mode_label }} ».

Base de données

La gestion des données repose sur une base SQLite qui est seulement utilisée pour stocker les pseudos des utilisateurs grâce à une table définie via SQLAlchemy. Nous avons fait ce choix afin que chaque participant puisse avoir un pseudo lors des parties et par rapport aux consignes du projet, nous avons jugé plus adapté de limiter la base de données à cette unique table. Toutefois, une extension de la structure de la base de données pourrait être envisagée si le projet venait à évoluer.

Concernant l'importation d'un fichier JSON, nous avons conçu une structure JSON adaptée à nos besoins. Ce format est organisé autour de quatre clés principales :

- « partie_id » : identifie la partie
- « mode_de_jeu » : précise le mode de calcul des votes
- « number_player » : indique le nombre maximal de joueurs
- « résultats » : représente la liste des problèmes avec leurs évaluations

Ce format a été conçu pour offrir une structure simple et flexible, facilitant la gestion des données tout en garantissant leur cohérence.

En utilisant ces différentes technologies, nous avons pu concevoir un projet avec une architecture efficace et adaptés aux besoins du projet.

Intégration continue

Dans le cadre du développement de l'application, l'intégration continue a été mise en place pour garantir la qualité du code et faciliter la collaboration. Cette approche inclut l'exécution automatique des tests unitaires et la génération de la documentation.

Tests unitaires

Pour les tests unitaires, nous avons utilisé PyTest pour notre code côté serveur en Python de notre fichier routes.py et Jest pour le frontend JavaScript. Ces tests ont été réalisés uniquement après la finalisation de l'application, car nous n'étions pas familiers avec les concepts d'intégration continue et d'automatisation des tests au début du projet. Par conséquent, nous nous sommes principalement concentrés sur l'exécution manuelle des tests, ce qui nous permettait de détecter des bugs et valider le fonctionnement de notre code sans bénéficier d'une exécution automatisée régulière. Malgré le manque de tests unitaires automatisés, nous les avons réalisés à la fin de notre développement.


```
(venv) hubert-geoffray@Hub:/mnt/c/users/huber/desktop/poker/projet_poker$ pytest -v test/test_routes.py
===== test session starts =====
platform linux -- Python 3.12.3, pytest-8.3.4, pluggy-1.5.0 -- /mnt/c/users/huber/desktop/poker/projet_poker/venv/bin/python3
cachedir: .pytest_cache
rootdir: /mnt/c/users/huber/desktop/poker/projet_poker
plugins: flask-1.3.0
collected 12 items

test/test_routes.py::test_signup_get PASSED [ 8%]
test/test_routes.py::test_signup_post PASSED [ 16%]
test/test_routes.py::test_login_get PASSED [ 25%]
test/test_routes.py::test_login_post_existing_user PASSED [ 33%]
test/test_routes.py::test_dashboard_redirect_if_not_logged_in PASSED [ 41%]
test/test_routes.py::test_dashboard_access PASSED [ 50%]
test/test_routes.py::test_create_game PASSED [ 58%]
test/test_routes.py::test_join_game PASSED [ 66%]
test/test_routes.py::test_game_room_access FAILED [ 75%]
test/test_routes.py::test_game_room_redirect_invalid_game PASSED [ 83%]
test/test_routes.py::test_devoiler_vote_unanimous FAILED [ 91%]
test/test_routes.py::test_devoiler_vote_strict_revote FAILED [100%]
test/test_routes.py::test_devoiler_vote_strict_revote ERROR [100%]
```

Figure 8 : Tests unitaires Python des fonctions de routes.py

```
(venv) hubert-geoffray@Hub:/mnt/c/users/huber/desktop/poker/projet_poker$ npx jest test/test.js
PASS test/test.js (11.541 s)
Tests des fonctions principales
  ✓ addProblemToUI ajoute un problème dans la liste (44 ms)
  ✓ displayVote affiche un vote pour un problème (8 ms)
  ✓ displayVote met à jour un vote existant (7 ms)
  ✓ displayConcludedVote affiche un résultat conclu pour un problème (5 ms)

Test Suites: 1 passed, 1 total
Tests: 4 passed, 4 total
Snapshots: 0 total
Time: 11.895 s
Ran all test suites matching /test\/test.js/i.
```

Figure 9 : Test unitaires JavaScript des fonctions de game_room.html

Documentation

Concernant la documentation, elle a été effectuée avec Doxygen pour Python et JsDoc pour JavaScript. Les deux documentations ont été créées une fois le projet terminé, afin de fournir une vue d'ensemble claire des fonctions. L'absence d'une génération automatique de la documentation durant le développement a nécessité un travail conséquent à la fin du projet. Nos documentations fournissent néanmoins des explications claires des fonctions principales.

```
• devoiler_vote()
doc.devoiler_vote( data )

@brieff Gère les votes pour un problème spécifique dans une partie.

@param data Dictionnaire contenant :
  - "game_id" (str) : l'identifiant de la partie.
  - "problem" (str) : le problème pour lequel les votes sont dévoilés.
  - "compteur" (int) : compteur pour savoir le numéro du tour.

@details
  - Vérifie que la partie et le problème existent.
  - Traite le mode de jeu Strict et les votes unanimes des premiers tours
  - Si tous les joueurs ont voté "safe", la partie est automatiquement sauvegardée.
  - Traite les votes en fonction du mode de jeu (de: strict, moyen, médiane, etc.).
  - Notifie les joueurs du résultat ou demande un nouveau vote si nécessaire.

@exit
  - "error" : en cas de problème avec la partie ou le problème.
  - "unanimous_vote" : si tous les joueurs ont voté la même chose.
  - "revote" : demande aux joueurs de revoter si pas de résultat possible
  - "refresh_ui" : met à jour l'interface utilisateur.

• devoiler_vote_majorite_absolue()
doc.devoiler_vote_majorite_absolue( game_id,
                                   problem,
                                   votes )

@brieff Calcule le vote majoritaire absolu.

@param game_id (str) : l'identifiant de la partie.
@param problem (str) : le problème pour lequel la majorité absolue est calculée.
@param votes (dict) : Un dictionnaire des votes des joueurs.

@details
  - Compte les occurrences de chaque vote.
  - Vérifie si un vote obtient plus de 50% des votes totaux.
  - Si oui, enregistre et met le résultat. Sinon, demande un nouveau vote.

@exit
  - "majority_vote" : Contient le résultat de la majorité absolue.
  - "revote" : Demande un nouveau vote si aucune majorité absolue.
```

Figure 10 : Documentation Doxygen des fonctions routes.py

displayConcludedVote(problem, result)		
Met à jour l'interface pour afficher le résultat d'un vote conclu pour un problème donné.		
Parameters:		
Name	Type	Description
problem	string	Le problème pour lequel le vote a été conclu.
result	string number	Le résultat du vote (peut être une chaîne ou un nombre selon le mode).
Source: doc.js, line 94		
displayVote(problem, player, vote)		
Affiche ou met à jour le vote d'un joueur pour un problème.		
Parameters:		
Name	Type	Description
problem	string	Le problème sélectionné.
player	string	Le pseudo du joueur ayant voté.
vote	string number	La valeur du vote du joueur.
Source: doc.js, line 104		

Figure 11 : Documentation JsDoc pour game_room.html

Axes d'amélioration

Expérience utilisateur

Bien que le projet ait atteint quelques objectifs donnés, plusieurs améliorations pourraient être envisagées pour améliorer l'expérience utilisateur. Voici quelques axes d'amélioration identifiés :

- Le chargement correct d'un backlog JSON serait certainement l'une des premières améliorations à implémenter. Effectivement, actuellement, lorsque l'on souhaite charger un backlog JSON depuis le menu, une partie se crée bien avec les bons paramètres donnés mais en ce qui concerne des problèmes enregistrés, il y a un souci au niveau de l'affichage car il n'affiche pas les évaluations données. Cette fonctionnalité pourrait être corrigée en s'assurant de la bonne transmission des données importées dans l'interface utilisateur, afin que les problèmes et leurs évaluations soient correctement affichés.
- Un enrichissement de la base de données est aussi envisageable pour renforcer ses fonctionnalités et répondre à de potentiels besoins plus complexes. On pourrait tout d'abord améliorer la table « User » en y ajoutant des champs de mot de passe et de gestion de rôle afin d'assurer une certaine sécurité. De plus la création de nouvelles tables comme « Parties » par exemple, permettrait ici d'enregistrer les informations d'une partie directement dans la base de données et de permettre de reprendre une partie sans passer par un backlog JSON. Ces améliorations contribueraient ainsi à rendre la base de données plus fiable.
- Pour l'expérience de jeu, l'ajout d'un tchat servant à communiquer entre les joueurs seraient très utile car notre projet est en multijoueur ce qui aiderait les joueurs à débattre pour les problèmes. L'ajout d'un chronomètre pendant le vote d'un problème ne serait pas de refus car cela inciterait les joueurs à prendre une décision assez rapide, ce qui en fait un jeu fluide.

Expérience développeur

En tant que développeur, le manque d'intégration continue tout au long du développement de l'application nous a permis de comprendre son importance. A l'avenir, nous prévoyons d'approfondir et de mieux appliquer l'intégration continue pour garantir une meilleure qualité du code du début à la fin de nos futurs projets.

Conclusion

Ce projet de planning poker nous a permis d'appliquer les principes de la conception agile dans le cadre d'un développement concret. Grâce à une architecture simple basée sur Flask et des technologies communes comme *HTML*, *CSS* et *JavaScript*, nous avons pu répondre aux objectifs initiaux.

Malgré quelques limitations, le projet offre une solution opérationnelle et conforme aux attentes. Ces limitations ouvrent cependant des perspectives d'amélioration intéressantes, notamment pour enrichir les fonctionnalités, optimiser davantage l'expérience utilisateur et nos méthodes de projets informatiques.