Rapport de Projet

Conception Agile - Application Planning Poker

M1 Informatique

Binôme: Tadimi - Bekkar

Décembre 2025

Table des matières

1 Introduction et Contexte

1.1 Présentation du Projet

Ce projet consiste en le développement d'une application web de **Planning Poker**, une technique d'estimation agile utilisée dans le développement logiciel pour estimer la complexité des tâches d'un backlog. L'application permet à une équipe de développeurs de voter simultanément sur la difficulté estimée de chaque fonctionnalité, en utilisant des cartes numérotées selon la suite de Fibonacci (0, 1, 2, 3, 5, 8, 13, 21) ainsi que des cartes spéciales (?, café).

L'objectif principal est de faciliter le processus d'estimation collaborative en offrant plusieurs modes de vote (Strict, Moyenne, Médiane, Majorité Absolue, Majorité Relative) et en garantissant que le premier tour nécessite toujours l'unanimité, conformément aux règles du Planning Poker.

1.2 Organisation du Travail

Ce projet a été réalisé en **binôme** (Tadimi - Bekkar) selon une approche de **Pair Programming**. Cette méthode de travail collaborative a permis :

— Une meilleure qualité du code grâce à la revue en temps réel

— Une compréhension partagée de l'architecture et des choix techniques

— Une réduction des erreurs grâce à la détection immédiate des problèmes

— Une meilleure répartition des connaissances sur l'ensemble du codebase

Le développement s'est organisé autour de sessions de programmation en binôme, où chaque membre du binôme a alterné entre le rôle de "driver" (écriture du code) et "navigator" (revue et guidage).

2 Choix Techniques et Architecture

2.1 Langage et Frameworks

2.1.1 Choix de JavaScript/React

Le choix s'est porté sur une **application web** développée en **JavaScript** avec le framework **React** pour plusieurs raisons :

— **Accessibilité** : Une application web ne nécessite pas d'installation spécifique et fonctionne sur tous les systèmes d'exploitation via un navigateur moderne

— **Interactivité** : React permet de créer une interface utilisateur réactive et fluide, essentielle pour une application de vote en temps réel

— **Écosystème riche** : L'écosystème JavaScript offre de nombreuses bibliothèques pour le routage (React Router), les tests (Vitest), et la documentation (JSDoc)

— **Développement rapide** : Les outils modernes comme Vite permettent un développement rapide avec rechargement à chaud (HMR)

2.1.2 Stack Technique

— **React 19.2.0** : Framework UI pour la construction de composants réutilisables

— **React Router DOM 7.9.6** : Gestion de la navigation entre les différentes pages de l'application

— **Vite 7.2.2** : Build tool moderne offrant un développement rapide et une compilation optimisée

— **Tailwind CSS 4.1.17** : Framework CSS utilitaire pour un styling rapide et cohérent

— **Vitest 4.0.8** : Framework de tests unitaires, compatible avec l'écosystème Vite

— **JSDoc 4.0.5** : Génération automatique de documentation à partir des commentaires dans le code

2.2 Architecture Logicielle

2.2.1 Architecture en Composants

L'application suit une **architecture en composants React** organisée selon le pattern **Container/Presentational** :

— **Pages** (`pages/`) : Composants conteneurs gérant la logique métier et la navigation

— **Composants** (`components/`) : Composants présentationnels réutilisables

— **Context** (`context/`) : Gestion centralisée de l'état global via React Context API

2.2.2 Gestion de l'État

L'état global de l'application est géré via le **React Context API** dans `GameContext.jsx`. Ce choix permet :

— Une gestion centralisée de l'état du jeu (joueurs, backlog, votes, mode de jeu)

— L'évitement du "prop drilling" (passage de props à travers plusieurs niveaux)

— Une séparation claire entre la logique métier et les composants UI

Le contexte expose les fonctions suivantes :

— `submitVote` : Enregistre un vote d'un joueur

— `calculateResult` : Calcule le résultat selon le mode de jeu sélectionné

— `checkIfAllVoted` : Vérifie si tous les joueurs ont voté

— `completeFeature` : Marque une fonctionnalité comme complétée

— `exportResults` : Exporte les résultats au format JSON

2.2.3 Modélisation des Données

Structure des Joueurs Chaque joueur est représenté par un objet avec :

```
{
  "id": "player-0",
  "name": "Alice"
}
```

Structure du Backlog Le backlog est un tableau d'objets représentant les fonctionnalités :

```
[
  {
    "name": "Authentification utilisateur",
    "description": "Impl menter un syst me de
       connexion/inscription"
  }
]
```

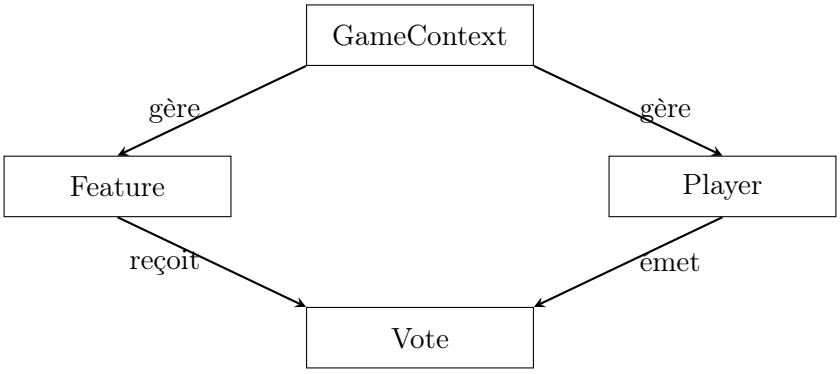
Structure de Sauvegarde La sauvegarde de partie contient :

```
{
  "players": [...],
  "gameMode": "strict",
  "backlog": [...],
  "currentFeatureIndex": 0,
  "completedFeatures": [...],
  "timestamp": "2025-12-18T10:30:00.000Z"
}
```

Structure des Résultats L'export des résultats contient uniquement les fonctionnalités complétées :

```
{
  "gameMode": "strict",
  "completedFeatures": [
    {
      "name": "Fonctionnalit ",
      "description": "...",
      "estimatedDifficulty": 8
    }
  ],
  "timestamp": "2025-12-18T10:30:00.000Z"
}
```

2.2.4 Diagramme de Classes Simplifié



Explication :

— **GameContext** : Gère l'état global du jeu et orchestre les interactions
— **Player** : Représente un joueur avec son identifiant et son nom
— **Feature** : Représente une fonctionnalité du backlog avec son nom et sa description
— **Vote** : Association entre un joueur et une valeur de carte pour une fonctionnalité donnée

2.3 Logique Métier

2.3.1 Calcul du Consensus

La fonction `calculateResult` implémente la logique de calcul selon le mode sélectionné :

1. **Premier tour** : Nécessite toujours l'unanimité (tous les votes identiques)
2. **Tours suivants** : Selon le mode :
   — **Strict** : Unanimité requise à chaque tour
   — **Moyenne** : Calcul de la moyenne arithmétique des votes numériques
   — **Médiane** : Valeur médiane des votes triés
   — **Majorité Absolue** : Plus de 50% des votes pour une même valeur
   — **Majorité Relative** : Valeur la plus fréquente parmi les votes

3 Mise en place de l'Intégration Continue

3.1 Présentation du Pipeline CI

L'intégration continue a été mise en place via **GitHub Actions**, avec deux workflows distincts qui s'exécutent automatiquement à chaque push sur la branche `main` :

— **Tests Unitaires** : Vérification automatique de la logique métier
— **Génération de Documentation** : Création et déploiement de la documentation JSDoc

3.2 Workflow des Tests Unitaires

3.2.1 Configuration

Le workflow `TU_javascript.yml` est déclenché sur :
— Push sur la branche `main`
— Pull Request vers la branche `main`

3.2.2 Étapes du Pipeline

1. **Checkout** : Récupération du code source depuis le dépôt Git

```
uses: actions/checkout@v4
```

2. **Setup Node.js** : Configuration de l'environnement Node.js version 22

```yaml
uses: actions/setup-node@v4
with:
  node-version: '22'
```

3. **Installation des dépendances** : Installation des packages npm depuis le répertoire `ScrumParty`

```yaml
working-directory: ./ScrumParty
run: npm install
```

4. **Exécution des tests** : Lancement de Vitest pour exécuter tous les tests unitaires

```yaml
run: npm run test
```

3.2.3 Couverture des Tests

Les tests unitaires couvrent les aspects suivants :

— **GameContext** : Vérification de l'état initial du contexte (mode de jeu, joueurs, backlog)

— **Composants** : Tests des composants `Card` et `PlayerVoteStatus` pour vérifier le rendu et les interactions

— **Logique métier** : Tests de la logique de vote et de calcul de consensus (via les tests d'intégration dans les pages)

3.3 Génération de Documentation

3.3.1 Configuration JSDoc

JSDoc est configuré via `jsdoc.json` pour :

— Scanner les fichiers `src/**/*.js` et `src/**/*.jsx`

— Exclure les fichiers de test (`*.test.jsx`)

— Générer la documentation HTML dans le répertoire `out/`

3.3.2 Workflow de Documentation

Le workflow `Documentation_javascript.yml` :

1. Effectue le checkout et configure Node.js (identique au workflow de tests)
2. Installe les dépendances
3. Génère la documentation avec la commande `npm run docs`
4. Déploie la documentation sur GitHub Pages via l'action `peaceiris/actions-gh-pages@v3`

La documentation générée est ainsi accessible publiquement via GitHub Pages, permettant une consultation facile de l'API et de la structure du code.

3.4 Avantages de la CI

— **Détection précoce des erreurs** : Les tests s'exécutent automatiquement avant chaque merge

— **Documentation à jour** : La documentation est régénérée à chaque modification du code

— **Confiance dans le code** : Assurance que le code fonctionne sur un environnement propre (Ubuntu)

— **Traçabilité** : Historique des exécutions disponibles dans l'onglet "Actions" de GitHub

4 Manuel Utilisateur et Fonctionnalités

4.1 Installation et Lancement

4.1.1 Prérequis

— **Node.js** : Version 18 ou supérieure (recommandé : Node.js 22)

— **npm** : Inclus avec Node.js

Vérification des versions :

```
node -v
npm -v
```

4.1.2 Installation

1. Cloner le dépôt Git :

```
git clone
   https://github.com/soooofiane/Scrum-Tadimi-Bekkar.git
cd Scrum-Tadimi-Bekkar
```

2. Se placer dans le répertoire du projet :

```
cd ScrumParty
```

3. Installer les dépendances :

```
npm install
```

4.1.3 Lancement de l'Application

Lancer le serveur de développement :

```
npm run dev
```

L'application sera accessible à l'adresse affichée dans le terminal (généralement `http://localhost:5`

4.2 Modes de Jeu

L'application implémente **cinq modes de jeu** :

4.2.1 Mode Strict (Unanimité)

— **Premier tour** : Tous les joueurs doivent voter la même valeur

— **Tours suivants** : L'unanimité reste requise à chaque tour

— **Avantage** : Garantit un consensus total de l'équipe

4.2.2 Mode Moyenne

— **Premier tour** : Unanimité requise

— **Tours suivants** : La valeur retenue est la moyenne arithmétique des votes numériques (arrondie à 1 décimale)

— **Exemple** : Votes [3, 5, 8] → Résultat : 5.3

4.2.3 Mode Médiane

— **Premier tour** : Unanimité requise

— **Tours suivants** : La valeur retenue est la médiane des votes triés

— **Exemple** : Votes [3, 5, 8, 13] → Résultat : 6.5 (moyenne de 5 et 8)

4.2.4 Mode Majorité Absolue

— **Premier tour** : Unanimité requise

— **Tours suivants** : Consensus si une valeur obtient plus de 50% des votes

— **Exemple** : 4 joueurs, votes [5, 5, 8, 13] → Consensus : 5 (2 votes sur 4 = 50%, pas de consensus)

4.2.5 Mode Majorité Relative

— **Premier tour** : Unanimité requise

— **Tours suivants** : La valeur la plus fréquente est retenue

— **Exemple** : Votes [3, 5, 5, 8] → Résultat : 5 (vote le plus fréquent)

4.3 Flux d'Utilisation

4.3.1 1. Page d'Accueil

L'utilisateur arrive sur la page d'accueil qui présente :

— Le titre de l'application

— Un bouton pour démarrer une nouvelle partie

— Un bouton pour charger une partie sauvegardée

FIGURE 1 – Page d'accueil de l'application Planning Poker

4.3.2 2. Configuration de la Partie

Sur la page de configuration (`/setup`), l'utilisateur doit :

1. **Définir les joueurs** :
   - Choisir le nombre de joueurs (2 à 10)
   - Entrer le nom de chaque joueur
2. **Sélectionner le mode de jeu** : Choisir parmi les 5 modes disponibles
3. **Charger le backlog** :
   - Option 1 : Charger un fichier JSON depuis le disque
   - Option 2 : Utiliser l'exemple fourni
   - Option 3 : Saisir manuellement le JSON dans la zone de texte
4. Cliquer sur "Commencer la Partie"

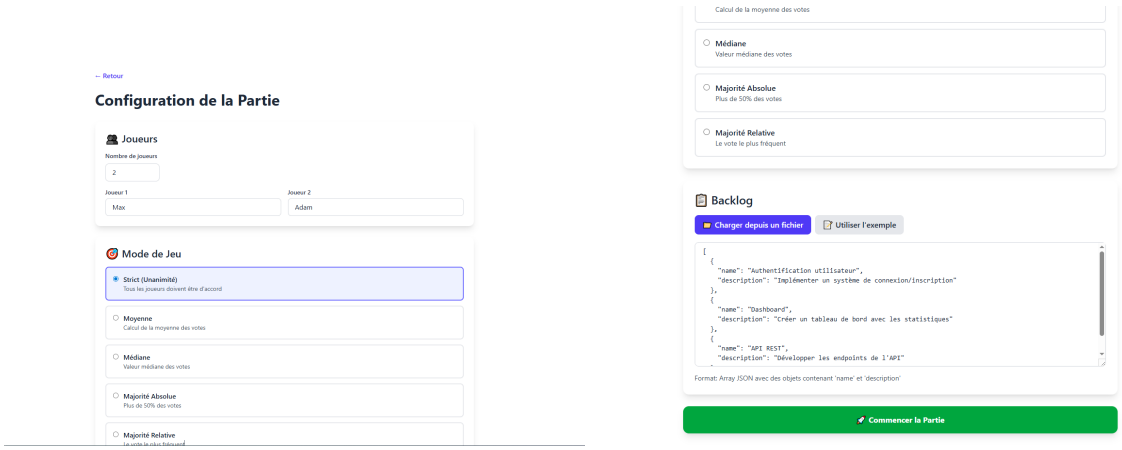


FIGURE 2 – Page de configuration de la partie (vue complète en deux parties)

4.3.3 3. Phase de Vote

Pour chaque fonctionnalité du backlog :

1. Affichage de la fonctionnalité courante avec son nom et sa description
2. Affichage du statut de vote de chaque joueur
3. Le joueur actif sélectionne une carte parmi :
   - Valeurs numériques : 0, 1, 2, 3, 5, 8, 13, 21
   - Cartes spéciales : ? (question), (café/pause)
4. Passage au joueur suivant jusqu'à ce que tous aient voté
5. Affichage des résultats du vote

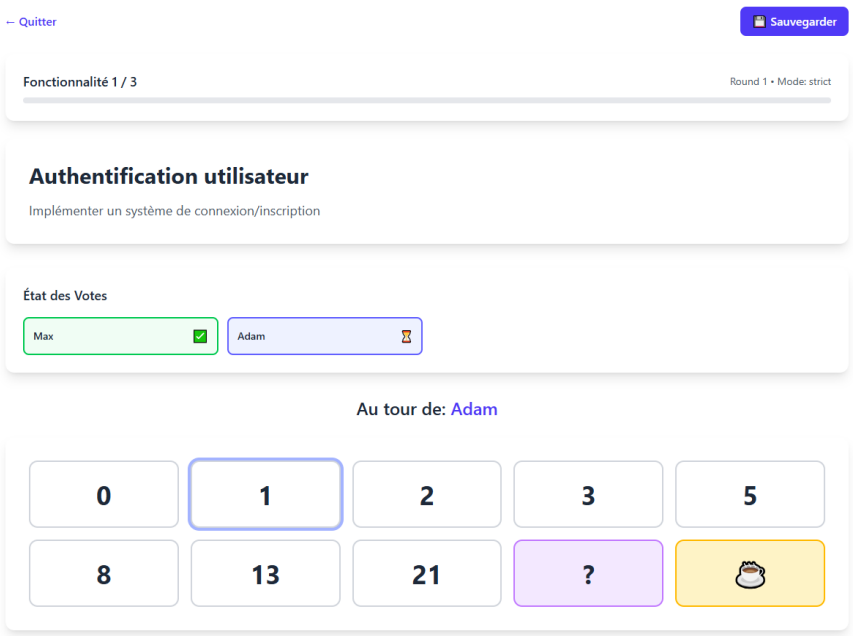


FIGURE 3 – Interface de vote avec les cartes Planning Poker

4.3.4 4. Gestion des Résultats

Après chaque tour de vote :

- Si **consensus atteint** : Possibilité de valider l'estimation et passer à la fonctionnalité suivante
- Si **pas de consensus** : Possibilité de relancer un nouveau tour de vote
- Si **tous les joueurs choisissent "café"** : Proposition de sauvegarder la partie

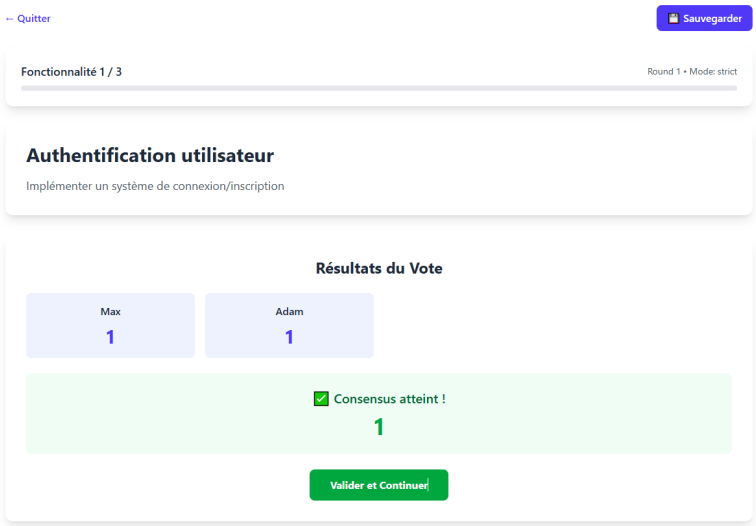


FIGURE 4 – Affichage des résultats d'un tour de vote

4.3.5 5. Page de Résultats

Une fois toutes les fonctionnalités estimées :

— Affichage du résumé : nombre de fonctionnalités estimées, points totaux, mode de jeu

— Liste détaillée de toutes les estimations

— Bouton pour exporter les résultats au format JSON

— Options pour démarrer une nouvelle partie ou retourner à l'accueil

FIGURE 5 – Page de résultats finaux avec le résumé des estimations

4.4 Fonctionnalités Avancées

4.4.1 Sauvegarde et Chargement

— **Sauvegarde** : Possibilité de sauvegarder l'état de la partie à tout moment (format JSON)

— **Chargement** : Reprendre une partie sauvegardée depuis la page d'accueil

— **Export des résultats** : Export uniquement des fonctionnalités complétées (sans l'état du jeu)

4.4.2 Interface Utilisateur

L'interface utilise Tailwind CSS pour un design moderne et responsive :

— Design adaptatif (mobile, tablette, desktop)

— Animations et transitions fluides

— Cartes visuelles pour les votes avec effets au survol

— Indicateurs visuels clairs pour le statut de vote de chaque joueur

5 Conclusion

Ce projet a permis de développer une application web complète de Planning Poker en suivant les principes de l'agilité et des bonnes pratiques de développement. L'utilisation de React et de l'architecture en composants a facilité la création d'une interface utilisateur intuitive, tandis que l'intégration continue garantit la qualité et la maintenabilité du code.

Les différents modes de vote implémentés offrent une flexibilité adaptée aux besoins variés des équipes de développement, tout en respectant la règle fondamentale du Planning Poker : l'unanimité au premier tour.

L'application est prête à être utilisée en production et peut être facilement étendue avec de nouvelles fonctionnalités grâce à son architecture modulaire et sa documentation complète.