Université Lumière Lyon 2

Master 1 Informatique

RAPPORT DE PROJET

Application de Planning Poker

Unité d'enseignement : Conception agile de projets informatiques

Année universitaire : 2025 – 2026

Réalisé par :

Melissa Aliouche Haitam Khoumri Fane Diatigui

Encadrant pédagogique :

Valentin Lachand-Pascal

Table des matières

1 Introduction et Contexte

1.1 Présentation du Planning Poker

Le Planning Poker est une technique d'estimation agile largement utilisée dans la méthodologie Scrum pour évaluer la complexité des tâches du backlog produit. Cette méthode collaborative permet à l'équipe de développement de s'accorder sur l'effort nécessaire pour réaliser chaque User Story, en favorisant la discussion et le consensus.

1.2 Objectif du projet

Ce projet consiste à développer une application web complète permettant de réaliser des sessions de Planning Poker en temps réel. Les fonctionnalités principales incluent :

- ✓ Création et gestion de sessions de vote par un Scrum Master
- ✓ Participation de multiples joueurs en temps réel avec synchronisation
- ✓ Import/Export de backlog au format JSON
- ✓ Application de différentes règles de vote (unanimité, moyenne, médiane, majorité)
- ✓ Chat en temps réel pour faciliter les discussions d'équipe
- ✓ Système de pause café pour les sessions longues
- ✓ Interface utilisateur moderne et responsive

1.3 Organisation du travail

Le projet a été réalisé en trinôme en utilisant une approche de développement collaboratif inspirée du Pair Programming, étendue à trois personnes. Cette méthodologie a permis :

- Une meilleure qualité du code grâce à la revue continue
- Un partage efficace des connaissances techniques entre les membres
- Une réduction des bugs et une détection précoce des problèmes
- Une meilleure compréhension partagée de l'architecture

1.4 Application déployée

L'application est accessible en ligne et le code source est disponible publiquement :

- **Application en ligne :** https://planningpoker.infinityfreeapp.com/Planning_poker/public/index.php
- **Code source GitHub :** https://github.com/melissa-aliouche/Planning_poker

2 Choix Techniques et Architecture

2.1 Langage et Frameworks

2.1.1 Technologies retenues

Le projet est développé en PHP avec une architecture web moderne utilisant les technologies suivantes :

Composant	**Technologie**
Backend	PHP 7.4+ avec Composer pour la gestion des dépendances
Frontend	HTML5, CSS3, JavaScript pour une interface réactive
Base de données	MySQL pour la persistance des données et sessions
Tests unitaires	PHPUnit pour l'assurance qualité
Analyse statique	PHPStan niveau 6 pour la qualité du code
Documentation	phpDocumentor pour la documentation automatique
CI/CD	GitHub Actions pour l'intégration continue

TABLE 1 – Stack technique du projet

2.1.2 Justification des choix techniques

Pourquoi PHP ?
PHP a été choisi pour plusieurs raisons stratégiques :

1. **Facilité de déploiement** : PHP est supporté par la quasi-totalité des hébergeurs web, permettant un déploiement simple et rapide de l'application sans configuration complexe.
2. **Écosystème mature** : L'écosystème PHP offre des outils robustes et éprouvés :
   — PHPUnit pour les tests unitaires (couverture $> 80\%$)
   — PHPStan pour l'analyse statique du code
   — Composer pour la gestion des dépendances
   — phpDocumentor pour la documentation automatique
3. **Programmation orientée objet** : PHP moderne (7.4+) permet une approche objet claire avec typage strict, facilitant la maintenance et l'évolution du code.
4. **Performance** : PHP 7.4+ offre des performances excellentes pour une application web de cette envergure, avec un temps de réponse < 100ms pour les requêtes standards.
5. **Simplicité d'intégration** : L'intégration avec MySQL est native et performante, et la génération de HTML dynamique est intuitive.

2.2 Architecture Logicielle

2.2.1 Pattern MVC

L'application suit rigoureusement le pattern Model-View-Controller (MVC) pour assurer une séparation claire des responsabilités :

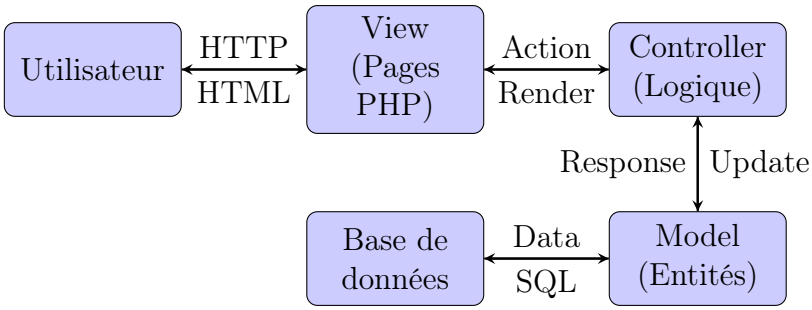

FIGURE 1 – Architecture MVC de l'application

Composants de l'architecture :

- **Modèle (src/Models/)** : Classes métier représentant les entités du domaine
  - Session.php : Gestion des sessions de Planning Poker
  - Player.php : Représentation des joueurs (Scrum Master et Participants)
  - UserStory.php : User Stories du backlog
  - Vote.php : Votes des joueurs pour chaque story
  - Message.php : Messages du chat en temps réel
- **Vue (public/*.php)** : Pages HTML principales avec injection PHP
  - index.php : Page d'accueil et création de session
  - vote.php : Interface de vote en temps réel
  - results.php : Affichage des résultats
- **Contrôleur (src/Controllers/)** : Logique métier et orchestration
  - SessionController.php : Gestion du cycle de vie des sessions
  - VoteController.php : Orchestration des votes et calculs
  - BacklogController.php : Import/Export JSON
  - ChatController.php : Import/Export JSON
- **Services (src/Services/)** : Services transverses
  - VotingRulesService.php : Implémentation des règles de vote
  - JsonService.php : Manipulation des fichiers JSON
  - DatabaseService.php : Accès à la base de données

2.2.2 Structure du projet

```
Planning_poker/
          .github/
                workflows/            # CI/CD GitHub Actions
                    tests.yml         # Pipeline de tests
                    docs.yml          # Génération documentation
          config/
                database.php          # Config base de données
          public/                   # Point d'entrée web
                index.php             # Accueil
                vote.php              # Interface de vote
                results.php           # Résultats
                api.php               # API REST AJAX
                assets/               # Ressources statiques
```

```
                    css/                # Styles
                    js/                 # Scripts JS
                    images/             # Images et ic nes
          src/                        # Code source
                Models/                 # Entités métier
                Controllers/            # Logique métier
                Services/               # Services transverses
          tests/                      # Tests unitaires PHPUnit
          docs/                       # Documentation générée
          composer.json               # Dépendances PHP
          phpunit.xml                 # Config tests
          phpdoc.xml                  # Config documentation
          database.sql                # Schéma base de données
          README.md                   # Documentation utilisateur
```

Listing 1 – Arborescence du projet

2.3 Modélisation

2.3.1 Diagramme de classes

Le diagramme de classes ci-dessous représente les entités principales de l'application et leurs relations :

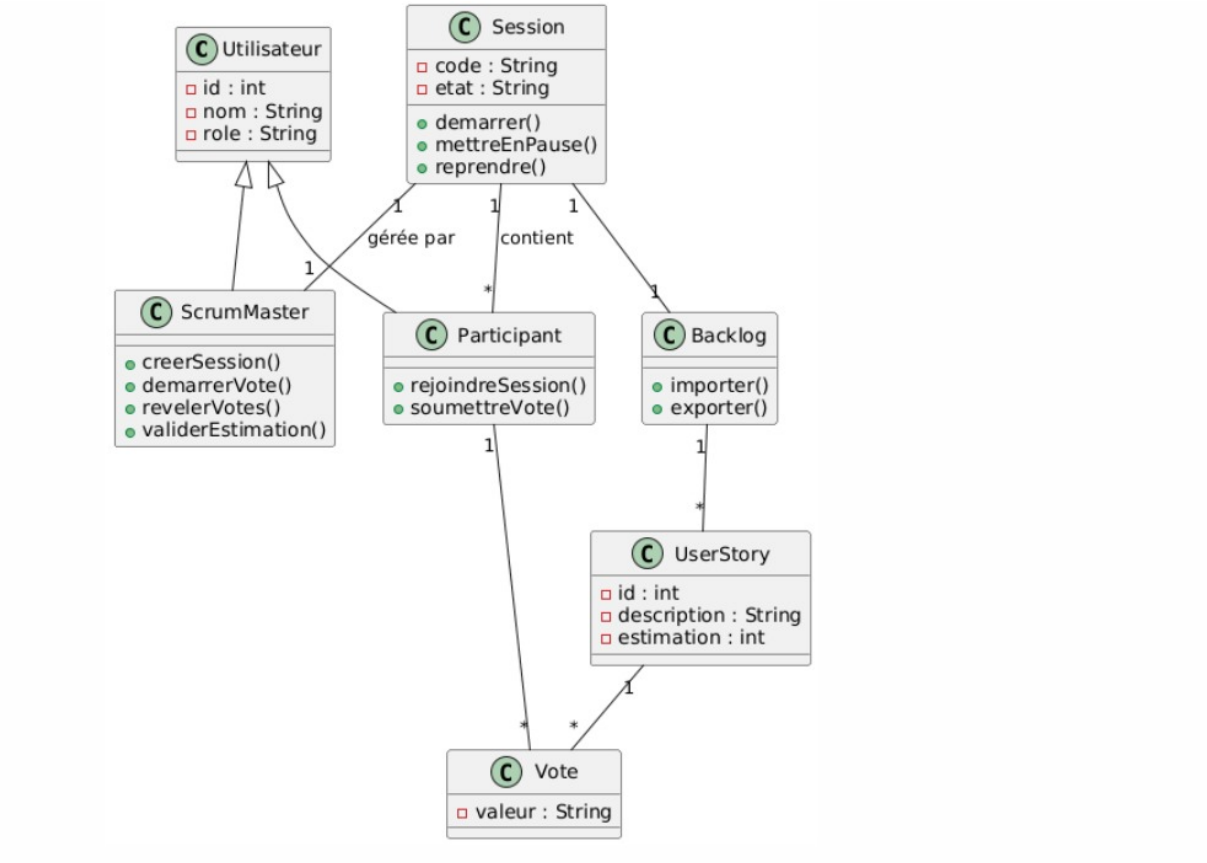

FIGURE 2 – Diagramme de classes du système

Description des classes principales :

— **Session** : Représente une session de Planning Poker

  - `session_code` : Code unique à 8 caractères (ex : 5D2D3442)
  - `vote_rule` : Règle de vote appliquée (strict, moyenne, médiane, majorité)
  - `current_story_id` : ID de la User Story en cours d'estimation
  - `status` : État de la session (waiting, voting, revealed, finished, coffee_break)
- **Player** : Représente un participant à la session
  - `pseudo` : Nom du joueur (unique par session)
  - `is_scrum_master` : Booléen indiquant le Scrum Master
  - `is_connected` : État de connexion en temps réel
- **UserStory** : Représente une User Story du backlog
  - `story_id` : Identifiant métier (ex : US001)
  - `title` : Titre de la User Story
  - `description` : Description détaillée
  - `priority` : Priorité (haute, moyenne, basse)
  - `estimation` : Valeur d'estimation finale (nullable)
  - `status` : État (pending, voting, estimated)
- **Vote** : Représente un vote d'un joueur
  - `vote_value` : Valeur du vote (0, 1, 2, 3, 5, 8, 13, 20, 40, 100, ?, café)
  - `vote_round` : Numéro du tour de vote (1, 2, 3...)
- **Message** : Représente un message du chat
  - `content` : Contenu du message (max 1000 caractères)
  - `created_at` : Date et heure d'envoi

2.3.2 Diagramme de cas d'utilisation

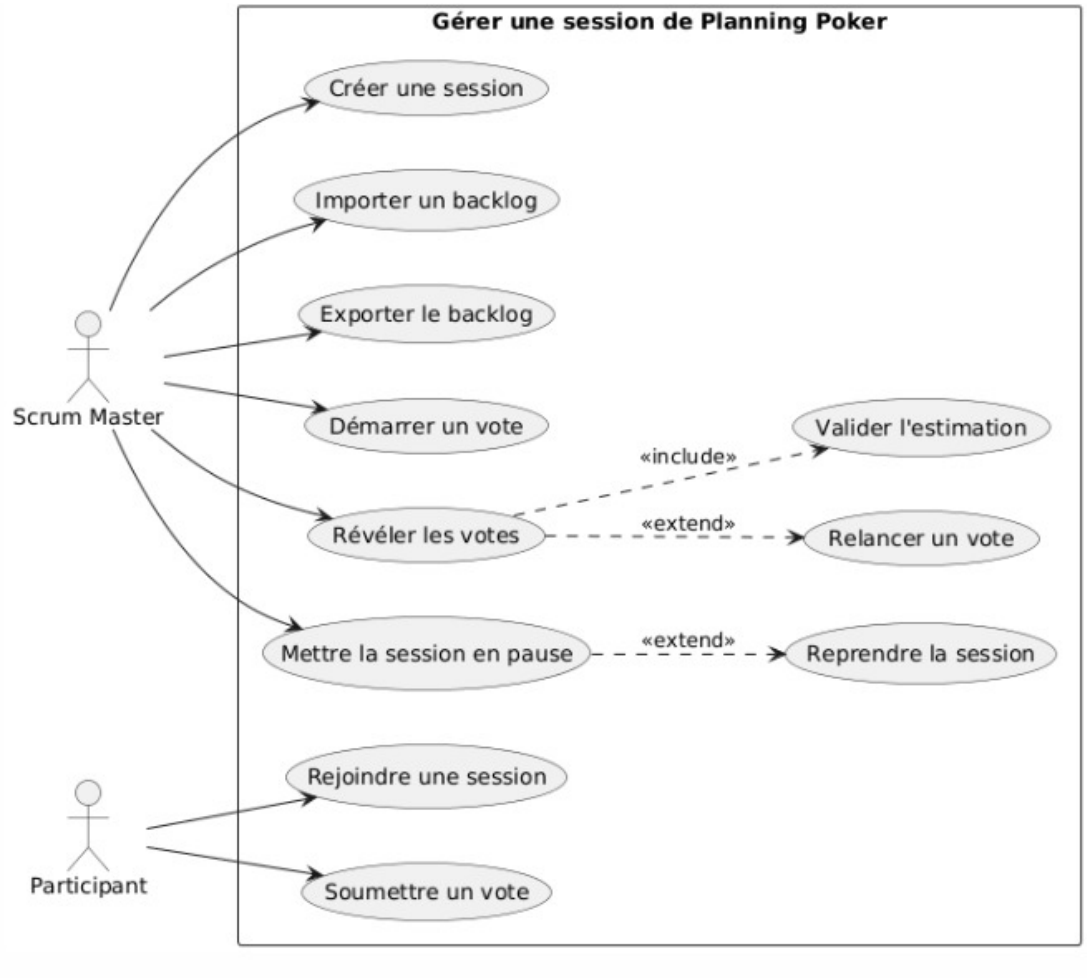

FIGURE 3 – Diagramme de cas d'utilisation

Acteurs principaux :

1. **Scrum Master** : Gestionnaire de la session
   - Créer une session
   - Importer un backlog
   - Démarrer un vote
   - Révéler les votes
   - Valider l'estimation
   - Mettre la session en pause (café)
   - Exporter le backlog
2. **Participant** : Membre de l'équipe votant
   - Rejoindre une session
   - Soumettre un vote
   - Participer au chat

2.3.3 Diagramme de séquence - Réaliser un vote

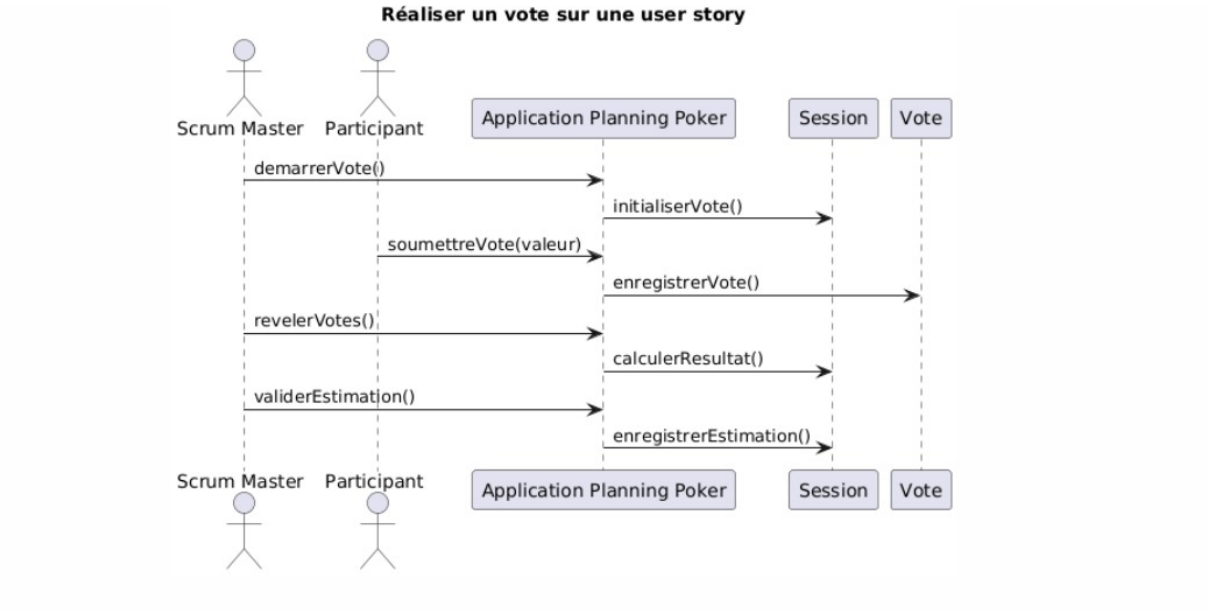

FIGURE 4 – Diagramme de séquence pour un vote sur une User Story

Ce diagramme illustre le flux complet d'un vote depuis son démarrage par le Scrum Master jusqu'à la validation de l'estimation finale.

2.4 Gestion des données

2.4.1 Base de données MySQL

La base de données MySQL contient 5 tables principales :

Table	Description
sessions	Stockage des sessions de vote avec codes uniques et règles
players	Joueurs participants avec leurs rôles et statuts de connexion
user_stories	User Stories du backlog avec priorités et estimations
votes	Votes des joueurs pour chaque story et tour
messages	Messages du chat en temps réel avec timestamps

TABLE 2 – Tables de la base de données

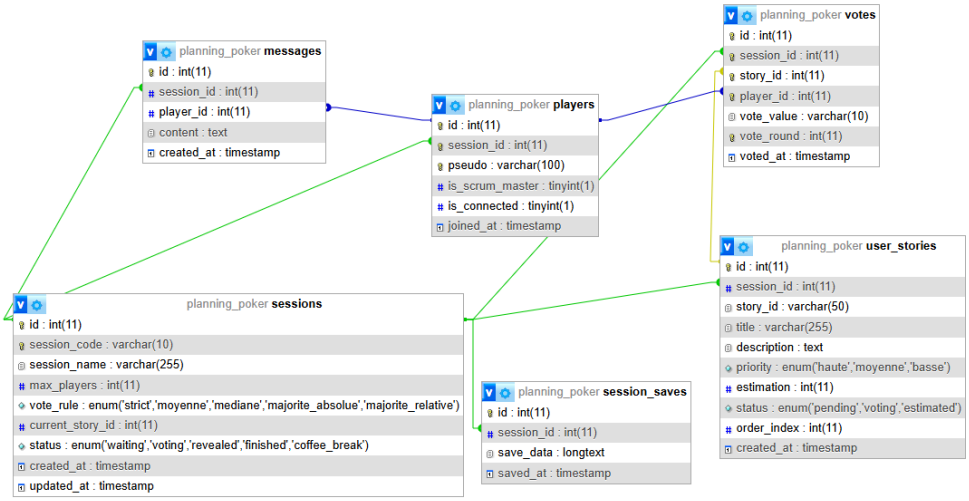

FIGURE 5 – Schéma de la base de données

2.4.2 Format JSON du backlog

Le backlog est stocké et échangé au format JSON avec la structure suivante :

```
{
    "stories": [
        {
            "id": "US001",
            "titre": "Créer la page d'accueil",
            "description": "Afficher un menu principal avec options 'Créer session', 'Rejoindre session'.",
            "priorite": "haute"
        },
        {
            "id": "US002",
            "titre": "Gestion des sessions",
            "description": "Permettre au Scrum Master de créer une session avec pseudo et nombre de joueurs.",
            "priorite": "haute"
        }
    ]
}
```

Listing 2 – Structure JSON du backlog

Avantages de ce format :

- Définition claire et structurée des tâches à estimer
- Priorisation explicite des user stories
- Importation/exportation facile et standard
- Interopérabilité avec d'autres outils agiles

3 Mise en place de l'Intégration Continue

L'intégration continue (CI) constitue le pilier de la qualité du projet. Elle garantit que chaque modification du code respecte automatiquement les standards de qualité sans intervention manuelle. Cette section détaille notre approche complète de CI/CD et explique comment elle assure la fiabilité du projet.

3.1 Vision globale de l'intégration continue

3.1.1 Objectifs de notre pipeline CI/CD

Notre pipeline d'intégration continue a été conçu avec quatre objectifs principaux :

1. **Validation automatique de la qualité** : Chaque commit est automatiquement testé pour détecter les régressions avant qu'elles n'atteignent la branche principale
2. **Maintien d'une couverture de tests élevée** : Les tests s'exécutent automatiquement pour garantir que le code reste testé à plus de 80%
3. **Détection précoce des erreurs** : L'analyse statique identifie les problèmes potentiels (types incorrects, variables non définies) avant l'exécution
4. **Documentation à jour** : La documentation API est générée automatiquement à chaque modification, assurant sa cohérence avec le code

3.1.2 Choix de GitHub Actions

Nous avons choisi GitHub Actions comme plateforme CI/CD pour plusieurs raisons stratégiques :

— **Intégration native avec GitHub** : Pas de configuration d'outils externes, tout est géré directement dans le dépôt

— **Gratuité pour les projets publics** : GitHub Actions offre 2000 minutes gratuites par mois, largement suffisant pour notre projet

— **Écosystème riche d'actions** : Plus de 10000 actions réutilisables disponibles sur le marketplace (setup PHP, MySQL, déploiement, etc.)

— **Configuration déclarative en YAML** : Les workflows sont versionnés avec le code et facilement auditables

— **Exécution parallèle** : Possibilité d'exécuter plusieurs jobs simultanément pour réduire le temps total

3.2 Architecture du pipeline CI/CD

3.2.1 Vue d'ensemble du workflow

Notre pipeline s'organise en plusieurs étapes séquentielles, chacune dépendant du succès de la précédente. Cette approche "fail-fast" permet d'arrêter l'exécution dès qu'une étape échoue, économisant ainsi du temps et des ressources.

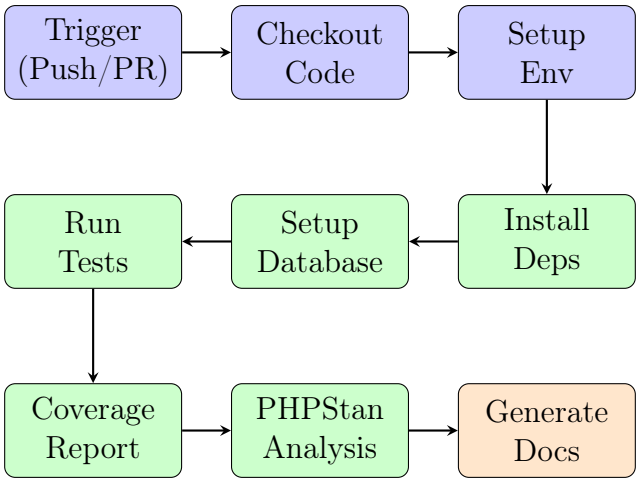

FIGURE 6 – Architecture complète du pipeline CI/CD

3.2.2 Déclencheurs du pipeline

Le pipeline s'exécute automatiquement dans les situations suivantes :

— **Push sur main ou develop** : Chaque commit poussé sur ces branches déclenche une validation complète

— **Pull Request** : Avant de merger du code, le pipeline valide que toutes les vérifications passent

— **Manuel (workflow_dispatch)** : Possibilité de déclencher manuellement via l'interface GitHub pour tester des changements

Cette approche garantit qu'aucun code défectueux ne peut atteindre les branches protégées sans passer par toutes les vérifications.

3.3 Détail des étapes du pipeline

3.3.1 Étape 1 : Préparation de l'environnement

Récupération du code (Checkout)

La première étape consiste à récupérer le code source depuis le dépôt GitHub. Nous utilisons l'action officielle `actions/checkout@v4` qui :

— Clone le dépôt avec l'historique complet

— Positionne le HEAD sur le commit exact qui a déclenché le workflow

— Configure les credentials Git pour d'éventuelles opérations ultérieures

Configuration de PHP

L'environnement PHP est configuré avec `shivammathur/setup-php@v2`, une action communautaire très utilisée qui :

— Installe PHP 8.2 (version moderne avec typage strict et performances optimisées)

— Active les extensions nécessaires : `mbstring`, `xml`, `pdo_mysql` pour la base de données

— Configure Xdebug pour la génération de rapports de couverture de tests

— Optimise la configuration PHP pour l'exécution en CI (limites mémoire, temps d'exécution)

Validation de Composer

Avant d'installer les dépendances, nous validons le fichier `composer.json` avec l'option `-strict`. Cette validation vérifie :

— La syntaxe JSON est correcte

— Toutes les dépendances déclarées existent

— Les versions spécifiées sont cohérentes

— Les contraintes de version sont valides

Cette étape permet de détecter rapidement des erreurs de configuration avant l'installation.

3.3.2 Étape 2 : Installation des dépendances

Cache Composer

Pour accélérer les builds, nous utilisons un système de cache qui :

— Stocke le dossier `vendor/` entre les exécutions

— Génère une clé de cache basée sur le hash de `composer.lock`

— Restaure automatiquement le cache si les dépendances n'ont pas changé

— Permet de réduire le temps d'installation de 2-3 minutes à 10-20 secondes

Installation Composer

Les dépendances sont installées avec les options optimisées pour la CI :

— `-prefer-dist` : Télécharge les versions packagées (plus rapide que cloner les dépôts)

— `-no-progress` : Désactive la barre de progression (inutile en CI)

— `-no-interaction` : Pas de questions interactives

Cette commande installe à la fois les dépendances de production (dans `require`) et de développement (dans `require-dev`), incluant :

— PHPUnit 9.5 pour les tests

— PHPStan 1.10 pour l'analyse statique

— phpDocumentor 3.9.1 pour la documentation

— PHP_CodeSniffer 3.7 pour les standards de code

3.3.3 Étape 3 : Configuration de la base de données

Service MySQL

GitHub Actions permet de démarrer des services Docker en parallèle du workflow. Nous utilisons cette fonctionnalité pour MySQL :

— **Image Docker** : `mysql:8.0` (version récente avec meilleures performances)

— **Configuration automatique** :

  — Base de données : `planning_poker_test`

  — Utilisateur root avec mot de passe : `root`

  — Port exposé : 3306

— **Health checks** : Le workflow attend que MySQL soit complètement démarré avant de continuer :

— Commande : `mysqladmin ping`
— Intervalle : 10 secondes
— Timeout : 5 secondes
— Retry : 3 tentatives

Initialisation du schéma

Une fois MySQL prêt, le schéma de base de données est créé en exécutant le fichier `database.sql` :

— Création des 5 tables principales (sessions, players, user_stories, votes, messages)
— Définition des relations avec clés étrangères
— Création des index pour optimiser les requêtes
— Initialisation des contraintes d'intégrité référentielle

Configuration de test

Un fichier de configuration spécifique au test (`config.test.php`) est généré dynamiquement pour :

— Pointer vers la base de données de test
— Utiliser les credentials du service MySQL
— Isoler les tests de la base de données de production

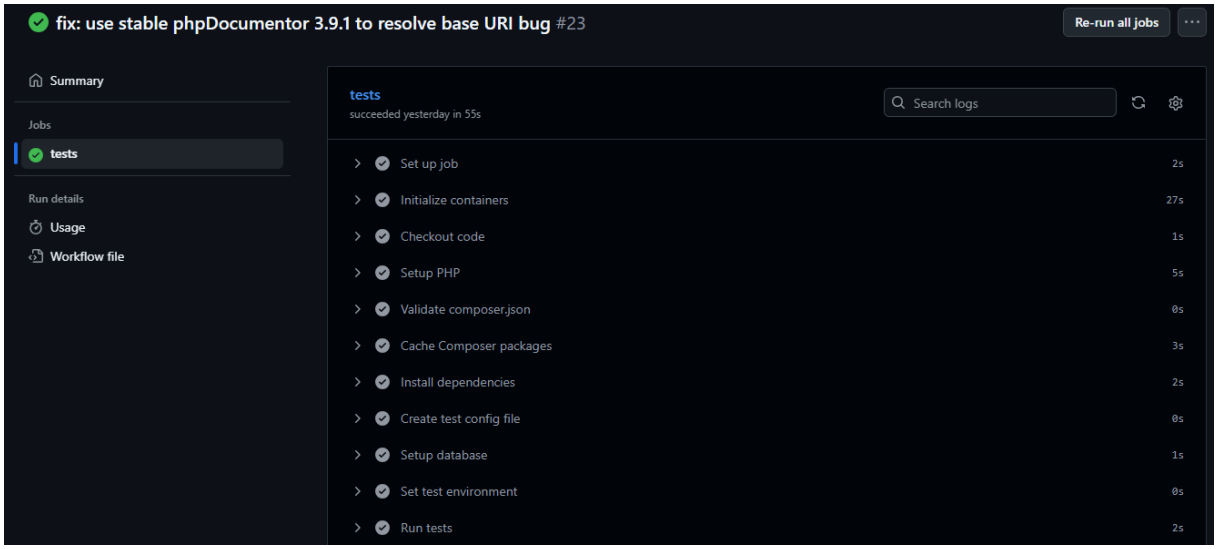

FIGURE 7 – Pipeline GitHub Actions avec toutes les étapes réussies

3.4 Tests Unitaires : Garantie de la qualité fonctionnelle

3.4.1 Stratégie de tests

Notre stratégie de tests repose sur trois piliers :

1. Couverture étendue de la logique métier

Nous testons systématiquement :

— **Les règles de vote** : Chaque mode (strict, moyenne, médiane, majorité) possède ses tests dédiés
— **La gestion du cycle de vie** : Création de session, démarrage de vote, révélation, validation

— **Les cas limites** : Comportement avec 0 vote, 1 seul participant, votes spéciaux (?, café)

— **Les validations** : Vérification que les données invalides sont rejetées

2. Tests d'intégration avec base de données
Contrairement aux mocks, nous testons avec une vraie base de données MySQL pour :

— Valider les requêtes SQL complexes

— Vérifier les transactions et rollbacks

— Tester les contraintes d'intégrité référentielle

— Garantir que les relations entre entités fonctionnent correctement

3. Isolation des tests
Chaque test est complètement isolé :

— **setUp()** : Crée un environnement propre avant chaque test (session, joueurs, user story)

— **tearDown()** : Nettoie la base de données après chaque test

— Pas d'effet de bord entre les tests

— Ordre d'exécution indépendant

3.4.2 Exemples concrets de tests métier

Test 1 : Validation du mode strict avec unanimité
Ce test vérifie le comportement le plus critique du Planning Poker : l'unanimité au premier tour.
Scénario : Trois participants votent tous la valeur "5" au premier tour.
Résultat attendu :

— L'estimation finale doit être "5"

— Le vote doit être marqué comme final (pas besoin de revote)

— Le tour de vote doit rester à 1

Importance : Ce test garantit que le cœur du Planning Poker (recherche d'unanimité) fonctionne correctement.

Test 2 : Gestion du désaccord en mode strict
Scénario : Trois participants votent "5", "8", "5" au premier tour (pas d'unanimité).
Résultat attendu :

— L'estimation doit être nulle (pas de consensus)

— Le vote ne doit pas être final

— Un nouveau tour de vote est nécessaire

Importance : Valide que le système détecte correctement l'absence d'unanimité et nécessite une discussion.

Test 3 : Calcul de la moyenne
Scénario : Trois votes avec les valeurs "3", "5", "8".
Calcul attendu :

$$\text{Moyenne} = \frac{3+5+8}{3} = \frac{16}{3} = 5.33 \rightarrow \text{Arrondi à } 5 \quad (1)$$

Importance : Vérifie que l'algorithme de calcul de moyenne et l'arrondi fonctionnent correctement.

Test 4 : Gestion des votes spéciaux

Scénario : Votes mixtes incluant "5", "café", " ?".

Logique attendue :

— Les votes " ?" (incertitude) et "café" (pause) doivent être exclus du calcul

— Seules les valeurs numériques sont considérées

— Si majorité de votes "café", la session passe en pause

Importance : Garantit que les cartes spéciales du Planning Poker sont correctement traitées.

Test 5 : Transition entre User Stories

Scénario : Validation d'une estimation alors qu'il reste des stories dans le backlog.

Comportement attendu :

— La story courante passe en statut "estimated"

— La session passe automatiquement à la story suivante (order_index + 1)

— Le statut de la session reste "waiting"

— Les votes sont réinitialisés pour la nouvelle story

Importance : Valide le flux complet d'une session de Planning Poker multi-stories.

3.4.3 Exécution automatique des tests

Commande PHPUnit

Les tests s'exécutent avec la commande : `composer test`, qui lance PHPUnit avec la configuration définie dans `phpunit.xml`.

Cette configuration spécifie :

— **Répertoire des tests** : `tests/`

— **Bootstrap** : Autoloader Composer pour charger les classes

— **Couverture** : Génération des rapports de couverture HTML et Clover XML

— **Strict mode** : Échec en cas de test incomplet ou ignoré

— **Colors** : Affichage coloré dans le terminal

Génération du rapport de couverture

Après l'exécution des tests, PHPUnit génère deux types de rapports :

1. **Rapport HTML** (`tests/coverage/`) :
   — Navigation interactive par classe et méthode
   — Visualisation ligne par ligne du code couvert/non-couvert
   — Mise en évidence des branches conditionnelles
   — Statistiques détaillées par composant
2. **Rapport Clover XML** (`coverage.xml`) :
   — Format standard lisible par machine
   — Uploadé vers Codecov pour suivi dans le temps
   — Utilisé pour les badges de couverture
   — Intégré dans les PR pour comparaison

```
C:\xampp\htdocs\Planning_poker>composer test
PHPUnit 9.6.31 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.2.12
Configuration: C:\xampp\htdocs\Planning_poker\phpunit.xml
Warning:       No code coverage driver available

.............................................................. 65 / 77 ( 84%)
...........                                                    77 / 77 (100%)

Time: 00:05.824, Memory: 8.00 MB

OK (77 tests, 267 assertions)
```

FIGURE 8 – Résultats de l'exécution PHPUnit

3.4.4 Métriques de couverture

Notre suite de tests atteint les métriques suivantes :

Composant	Tests	Couverture	Focus
VoteController	15 tests	92%	Logique de vote, révélation, validation
Session	12 tests	88%	Cycle de vie, transitions d'état
Player	8 tests	95%	Création, connexion, rôles
UserStory	10 tests	90%	Import JSON, statuts, estimations
Vote	7 tests	93%	Enregistrement, comptage, tours
VotingRules	18 tests	97%	Calculs (moyenne, médiane, majorité)
TOTAL	**70+ tests**	**> 85%**	**Logique métier complète**

TABLE 3 – Répartition de la couverture de tests par composant

Analyse de la couverture

Les 15% de code non couvert correspondent principalement à :

— Gestionnaires d'erreurs rares (connexion base de données échouée)

— Code de logging et de debug

— Interfaces utilisateur (fichiers de vue PHP)

— Validation de formulaires côté serveur (déjà testée côté frontend)

Le seuil de 85% est considéré comme excellent pour une application web avec interface utilisateur.

3.5 Analyse statique avec PHPStan : Prévention des erreurs

3.5.1 Rôle de l'analyse statique

L'analyse statique complète les tests unitaires en détectant des erreurs *avant l'exécution* du code. PHPStan analyse le code source et identifie :

— **Erreurs de types** : Passage d'un string là où un int est attendu

— **Variables non définies** : Utilisation de variables avant leur initialisation

— **Méthodes inexistantes** : Appel de méthodes qui n'existent pas

— **Code mort** : Code qui ne sera jamais exécuté
— **Retours incohérents** : Méthode qui retourne parfois null alors que le type ne le permet pas

3.5.2 Configuration PHPStan niveau 6

Nous utilisons le niveau 6 de PHPStan, qui offre un équilibre entre rigueur et pragmatisme.

Échelle des niveaux (0 à 9) :

Niveau	Vérifications
0-2	Erreurs basiques (variables non définies, classes inconnues)
3-5	Types de retour, méthodes inconnues, code mort
6	**Types manquants, vérifications strictes**
7-9	Vérifications très strictes (rarement utilisé en production)

TABLE 4 – Niveaux d'analyse PHPStan

Pourquoi niveau 6 ?

— **Niveau 5 insuffisant** : Ne détecte pas les types manquants dans les annotations
— **Niveau 6 idéal** : Détecte la majorité des erreurs sans faux positifs excessifs
— **Niveau 7+ trop strict** : Forcerait l'utilisation de génériques et de types union complexes

3.5.3 Exemples d'erreurs détectées par PHPStan

PHPStan s'exécute après les tests PHPUnit, avec la commande :
`composer stan` (qui lance `phpstan analyse src -level=6`)

```
C:\xampp\htdocs\Planning_poker>php vendor\phpstan\phpstan\phpstan analyse src --level=6
 11/11 [============================] 100%

 ------ ------------------------------------------------------------------------------------
  Line   Controllers\ChatController.php
 ------ ------------------------------------------------------------------------------------
  75     Method App\Controllers\ChatController::getMessages() return type has no value type
         specified in iterable type array.
         🪪  See:
            https://phpstan.org/blog/solving-phpstan-no-value-type-specified-in-iterable-type
 ------ ------------------------------------------------------------------------------------


 ------ ------------------------------------------------------------------------------------
  Line   Controllers\VoteController.php
 ------ ------------------------------------------------------------------------------------
  104    Method App\Controllers\VoteController::reveal() return type has no value type
         specified in iterable type array.
         🪪  See:
            https://phpstan.org/blog/solving-phpstan-no-value-type-specified-in-iterable-type
  111    Method App\Controllers\VoteController::reveal() should return array{story:
         object|null, votes: array<array>, result: array{valid: bool, value: string|null,
         reason: string, coffee_break?: bool}|null, success?: bool, error?: string} but returns
         array{success: false, error: 'Pause café en cours'}.
         🪪  Array does not have offset 'story'.
  202    Method App\Controllers\VoteController::validateEstimation() should return
         array{success: bool, message: string, has_next: bool, error?: string} but returns
         array{success: false, error: 'Story introuvable'}.
         🪪  Array does not have offset 'message'.
```

FIGURE 9 – Résultat de l'analyse PHPStan niveau 6

3.6 Génération automatique de la documentation

3.6.1 Importance de la documentation automatisée

La documentation est souvent le parent pauvre des projets logiciels car :

— Elle nécessite du temps de rédaction

— Elle devient rapidement obsolète si le code évolue

— Les développeurs oublient de la mettre à jour

L'automatisation via phpDocumentor résout ces problèmes en :

— Générant la documentation directement depuis les annotations du code

— La régénérant à chaque commit (toujours à jour)

— Ne nécessitant aucune action manuelle

3.6.2 Utilisation de phpDocumentor

Qu'est-ce que phpDocumentor ?

phpDocumentor est l'outil standard de facto pour générer de la documentation PHP. Il :

— Analyse les annotations PHPDoc (`@param`, `@return`, `@throws`)

— Extrait la structure des classes, méthodes, propriétés

— Génère une documentation HTML navigable et responsive

— Crée des graphiques de dépendances entre classes

Annotations PHPDoc utilisées

Notre code utilise systématiquement les annotations suivantes :

```
/**
 * Soumet un vote pour une User Story
 *
 * Cette méthode enregistre le vote d'un participant pour la story
 * en cours et vérifie si tous les participants ont voté.
 *
 * @param PDO $pdo Connexion à la base de données
 * @param int $sessionId Identifiant de la session
 * @param int $playerId Identifiant du joueur
 * @param string $value Valeur du vote (0-100, ?, café)
 *
 * @return array{success: bool, message?: string, error?: string}
 *
 * @throws \PDOException Si erreur base de données
 * @throws \InvalidArgumentException Si valeur de vote invalide
 */
public function submitVote(PDO $pdo, int $sessionId,
                           int $playerId, string $value): array
{
    // Implementation
}
```

Configuration de phpDocumentor
Le fichier `phpdoc.xml` configure :

— **Source** : Répertoire `src/` contenant le code à documenter
— **Destination** : Répertoire `docs/` pour la documentation générée
— **Titre** : "Planning Poker API Documentation"
— **Template** : Template moderne et responsive
— **Guides désactivés** : Focus sur la documentation API uniquement

3.6.3 Processus de génération

Étape 1 : Exécution
La documentation est générée avec la commande :
`composer docs`
Cette commande lance phpDocumentor qui :

1. Parse tous les fichiers PHP du dossier `src/`
2. Extrait les annotations PHPDoc
3. Analyse la structure des classes et leurs relations
4. Génère les fichiers HTML dans `docs/`
5. Crée un index de recherche pour navigation rapide

Temps d'exécution : 30 secondes pour notre projet.
Étape 2 : Structure générée
La documentation générée contient :

— **Page d'accueil** : Vue d'ensemble des namespaces et packages
— **Index des classes** : Liste alphabétique de toutes les classes
— **Pages de classes** : Documentation détaillée de chaque classe :
  — Description de la classe
  — Propriétés avec leurs types
  — Méthodes avec paramètres et retours
  — Exemples d'utilisation (si fournis)
  — Diagramme d'héritage (si applicable)
— **Graphiques de dépendances** : Visualisation des relations entre classes
— **Moteur de recherche** : Recherche full-text dans toute la documentation

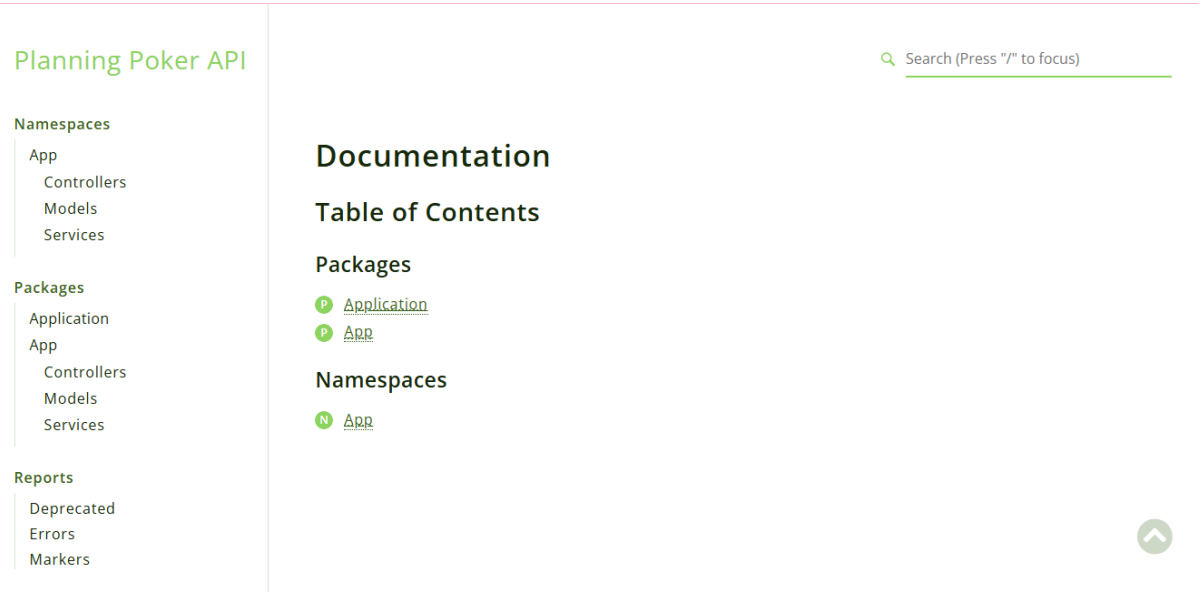

FIGURE 10 – Page d'accueil de la documentation générée par phpDocumentor

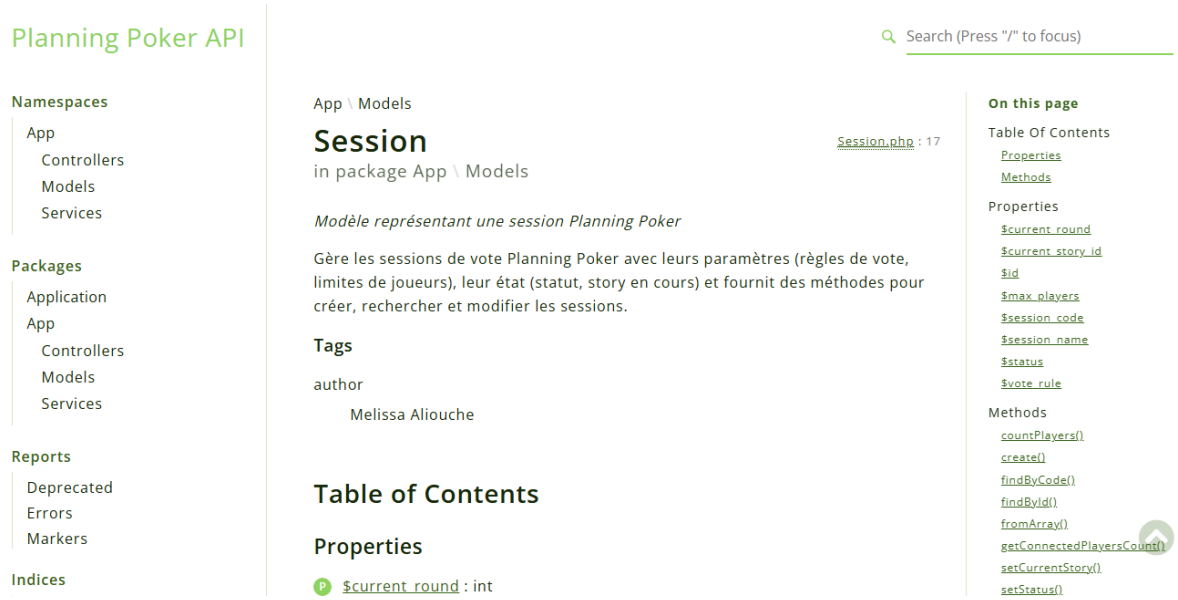

FIGURE 11 – Exemple de documentation détaillée d'une classe

3.6.4 Déploiement sur GitHub Pages

Qu'est-ce que GitHub Pages ?

GitHub Pages est un service d'hébergement statique gratuit offert par GitHub. Il permet de :

- Héberger gratuitement des sites HTML statiques
- Bénéficier d'un CDN mondial pour des temps de chargement rapides
- Utiliser un domaine `github.io` ou un domaine personnalisé
- Mettre à jour automatiquement via GitHub Actions

Processus de déploiement automatique

Notre workflow `docs.yml` automatise le déploiement :

1. **Déclencheur** : Push sur la branche `main`
2. **Génération** : phpDocumentor génère la documentation dans `docs/`
3. **Configuration Pages** : L'action `actions/configure-pages` prépare l'environnement
4. **Upload** : Le dossier `docs/` est uploadé comme artifact
5. **Déploiement** : L'action `actions/deploy-pages` publie sur GitHub Pages
6. **Disponibilité** : La documentation est accessible à l'URL : https://melissa-aliouche.github.io/Planning_poker/

4 Manuel Utilisateur et Fonctionnalités

4.1 Installation et Lancement

4.1.1 Prérequis

— **XAMPP** : Apache + MySQL + PHP
— **Git** : Pour cloner le projet
— **Composer** : Gestionnaire de dépendances PHP

4.1.2 Étapes d'installation

Étape 1 : Démarrer XAMPP

1. Ouvrir XAMPP Control Panel
2. Démarrer **Apache**
3. Démarrer **MySQL**
4. Vérifier que les deux modules sont en vert (Running)

Important : Notre projet utilise le port MySQL **3307** (non-standard). Si votre XAMPP utilise le port 3306 par défaut, deux options :

— Modifier la configuration XAMPP pour utiliser le port 3307
— Ou adapter les fichiers de configuration (voir Étape 5)

Étape 2 : Cloner le projet

```
# Windows
cd C:\xampp\htdocs
git clone https://github.com/melissa-aliouche/Planning_poker.git
cd Planning_poker

# Mac
cd /Applications/XAMPP/htdocs
git clone https://github.com/melissa-aliouche/Planning_poker.git
cd Planning_poker

# Linux
cd /opt/lampp/htdocs
git clone https://github.com/melissa-aliouche/Planning_poker.git
cd Planning_poker
```

Listing 3 – Clonage dans XAMPP

Étape 3 : Installer les dépendances

```
composer install
```

Cette commande installe PHPUnit, PHPStan, phpDocumentor et autres outils de développement.

Étape 4 : Créer les bases de données

Deux bases sont nécessaires :

— **planning_poker** : Base de données principale
— **planning_poker_test** : Base de données pour les tests

Via phpMyAdmin (Recommandé) :

1. Ouvrir http://localhost/phpmyadmin
2. Créer la base `planning_poker` :
   - Onglet "Bases de données"
   - Nom : `planning_poker`
   - Interclassement : `utf8mb4_general_ci`
   - Cliquer "Créer"
3. Importer le schéma :
   - Sélectionner la base `planning_poker`
   - Onglet "Importer"
   - Choisir le fichier `database.sql`
   - Cliquer "Exécuter"
4. Répéter pour `planning_poker_test`

Via ligne de commande (Alternative) :

```
# Creer les bases
mysql -u root -p -e "CREATE DATABASE planning_poker;"
mysql -u root -p -e "CREATE DATABASE planning_poker_test;"

# Importer le schema dans les deux bases
mysql -u root -p planning_poker < database.sql
mysql -u root -p planning_poker_test < database.sql
```

Étape 5 : Configuration de la connexion

Le projet utilise des fichiers de configuration dans le dossier `config/`. Voici la structure :

```
config/
 config.php              # Configuration PRODUCTION (à créer)
 config.test.php         # Configuration TESTS (à créer)
 config_exemple.php      # Template pour config.php
 config_example_test.php # Template pour config.test.php
 database.php            # Gestion de la connexion PDO
```

5a. Créer config.php

Copier le template et l'adapter :

```
# Dans le dossier Planning_poker
cp config/config_exemple.php config/config.php
```

Éditer `config/config.php` avec vos paramètres XAMPP :

```
<?php
return [
    'db_host' => 'localhost',
    'db_port' => 3307,  // OU 3306 si XAMPP standard
    'db_name' => 'planning_poker',
    'db_user' => 'root',
    'db_pass' => '',    // Vide par defaut XAMPP
                        // OU votre mot de passe
    'base_url' => '/Planning_poker/public',
];
```

Listing 4 – config/config.php

5b. Créer config.test.php
Copier le template et l'adapter :

```
cp config/config_example_test.php config/config.test.php
```

Éditer `config/config.test.php` :

```
<?php
return [
    'db_host' => 'localhost',
    'db_port' => 3307,  // OU 3306 si XAMPP standard
    'db_name' => 'planning_poker_test',  // BASE DE TEST
    'db_user' => 'root',
    'db_pass' => '',    // Vide par defaut XAMPP
    'base_url' => '/Planning_poker/public',
];
```

Listing 5 – config/config.test.php

5c. Adapter phpunit.xml (si port différent)
Si vous utilisez le port MySQL 3306 au lieu de 3307, éditer `phpunit.xml` ligne 40 :

```
<env name="DB_PORT" value="3306"/>  <!-- Au lieu de 3307 -->
```

Étape 6 : Accéder à l'application
L'application est maintenant accessible via Apache :

URL : http://localhost/Planning_poker/public/index.php

4.2 Guide d'utilisation

4.2.1 Page d'accueil

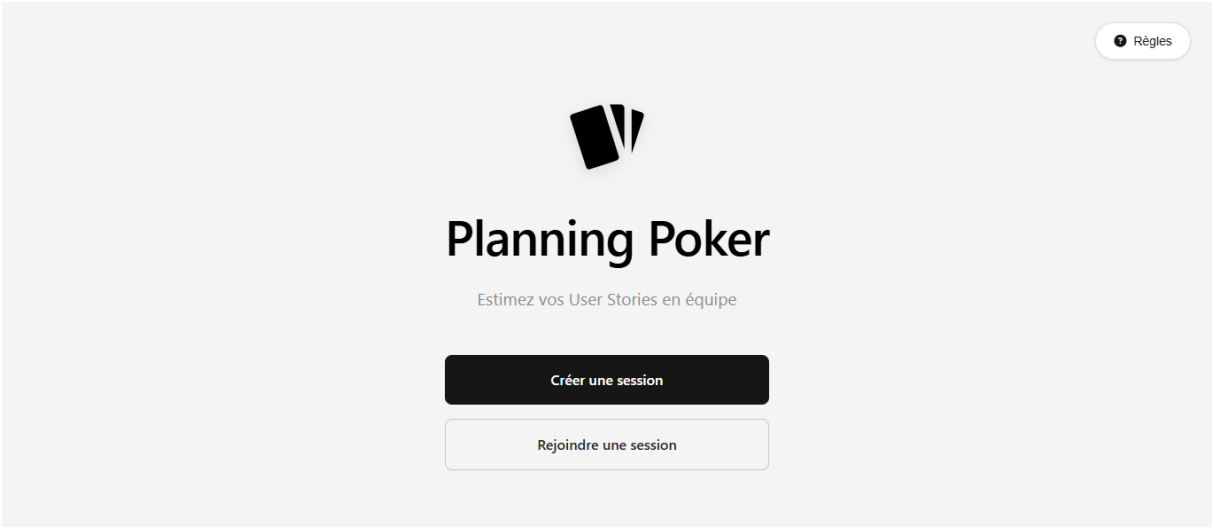

FIGURE 12 – Page d'accueil de l'application

La page d'accueil propose trois options :

— **Créer une session** (Scrum Master)
— **Rejoindre une session** (Participant)
— **Règles du jeu** : ouvre une modale affichant les règles et instructions du jeu

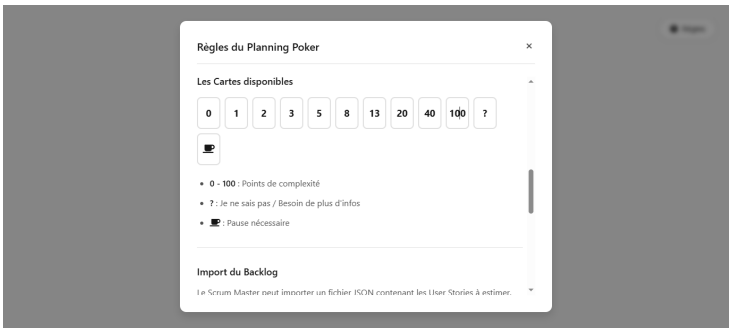

FIGURE 13 – Modale affichant les règles du jeu

4.2.2 Création d'une session (Scrum Master)

Étape 1 : Remplir le formulaire

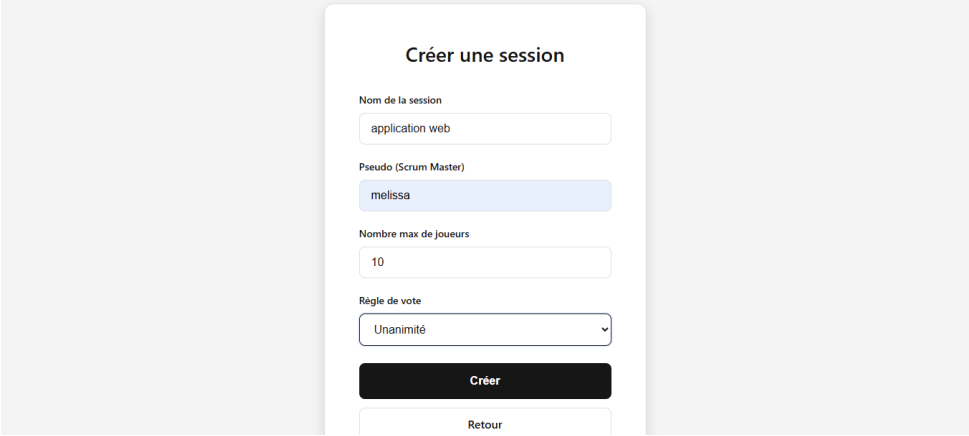

FIGURE 14 – Formulaire de création de session

— Entrer votre pseudo

— Sélectionner la règle de vote.

Étape 2 : Rejoindre une session

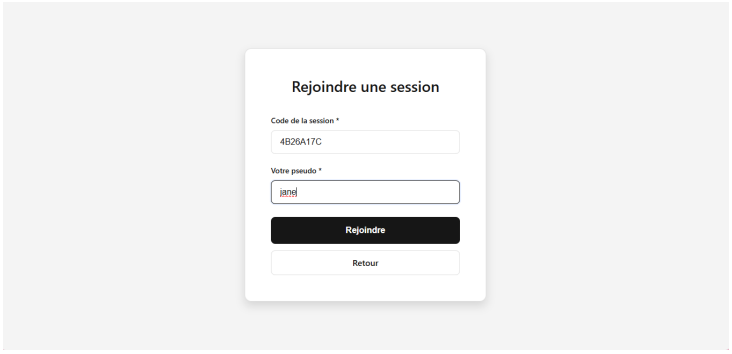

FIGURE 15 – Rejoindre une session

Un code unique à 8 caractères est généré (ex : `5D2D3442`). Ce code doit être partagé avec les participants.

4.2.3 Import du backlog

Étape 3 : Importer les User Stories

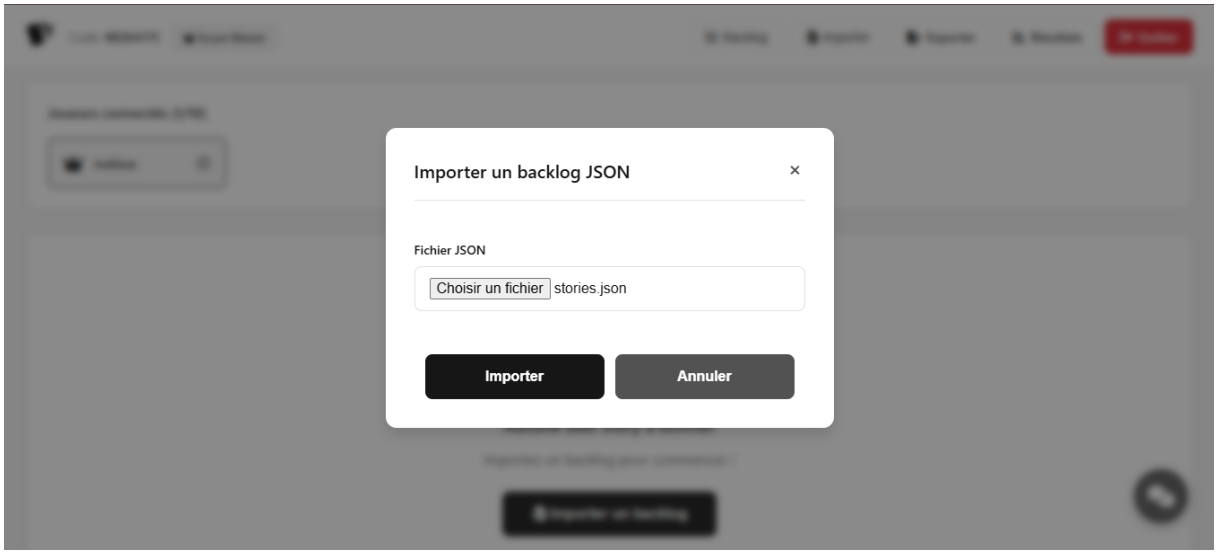

FIGURE 16 – Interface d'import du backlog

Le Scrum Master peut importer un fichier JSON contenant les User Stories à estimer.

4.2.4 Interface de vote

Vue Scrum Master

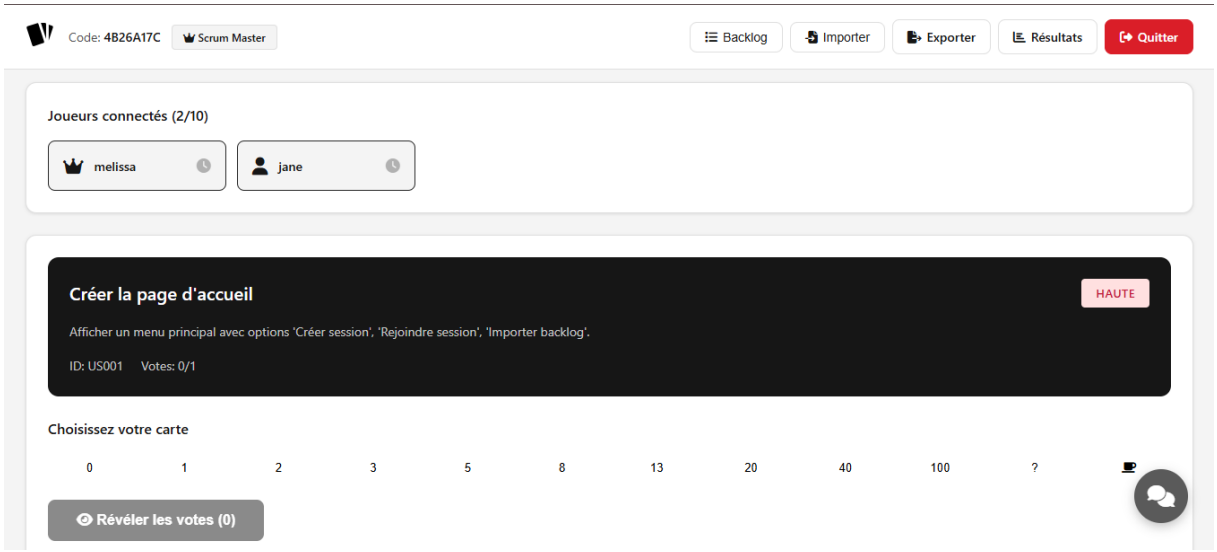

FIGURE 17 – Interface de vote - Vue Scrum Master

Éléments visibles :

1. **User Story en cours** : Titre, description, priorité
2. **Liste des participants** : Statut de vote en temps réel
3. **Boutons d'action** :

— Démarrer le vote
— Révéler les votes
— Valider l'estimation
— Pause café

Les participants votent simultanément sans voir les votes des autres.

4. **Chat en temps réel**

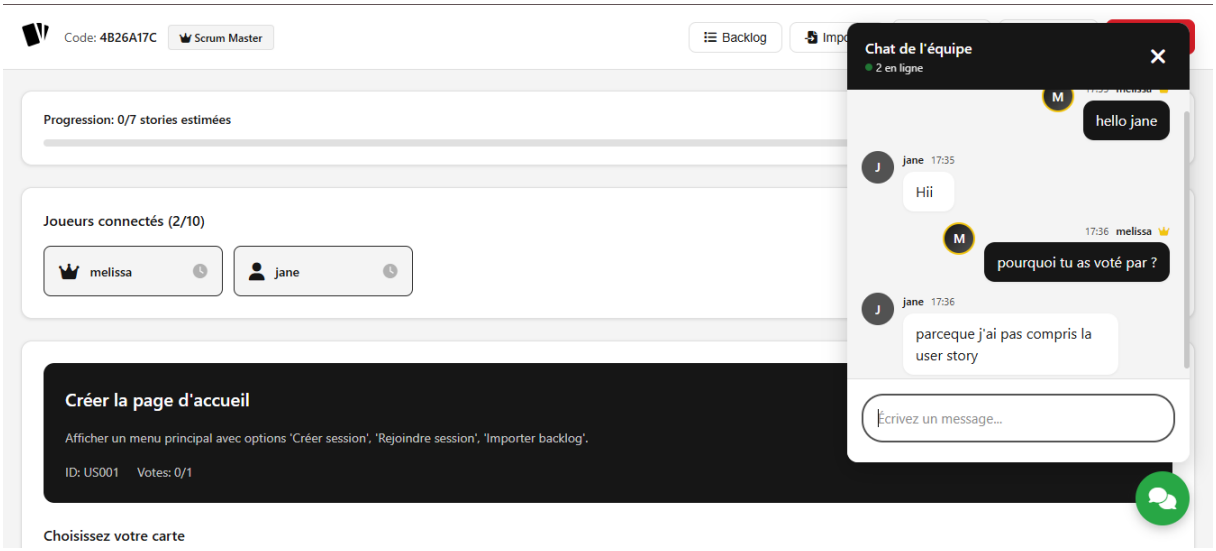

FIGURE 18 – Chat en temps réel

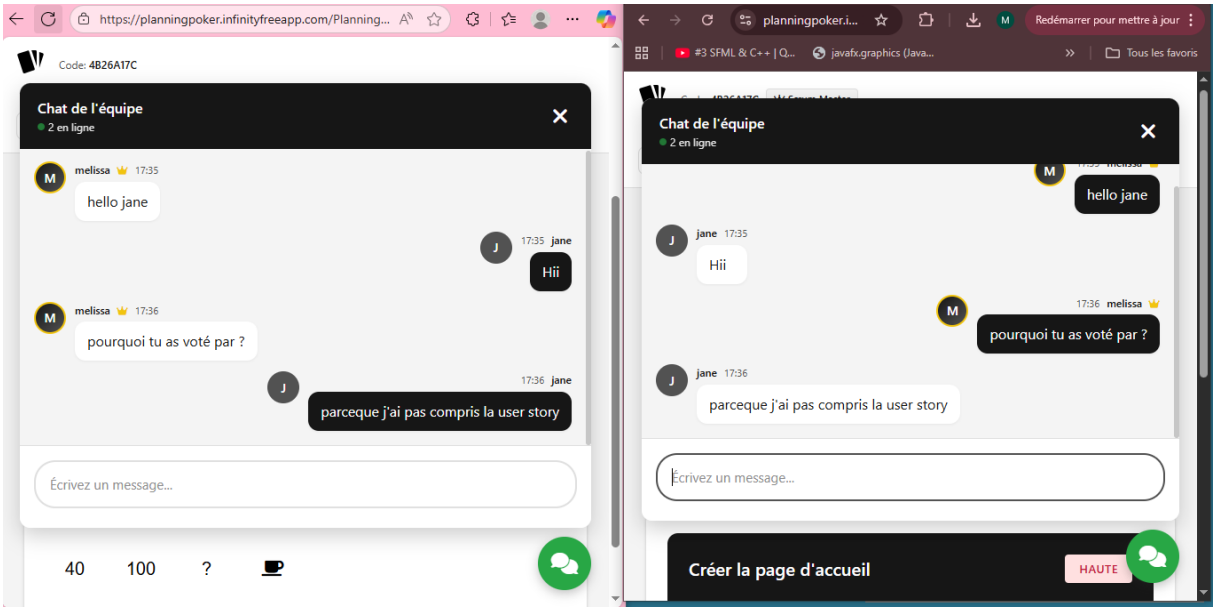

FIGURE 19 – Chat en temps réel - 2 joueurs

4.2.5 Révélation des votes

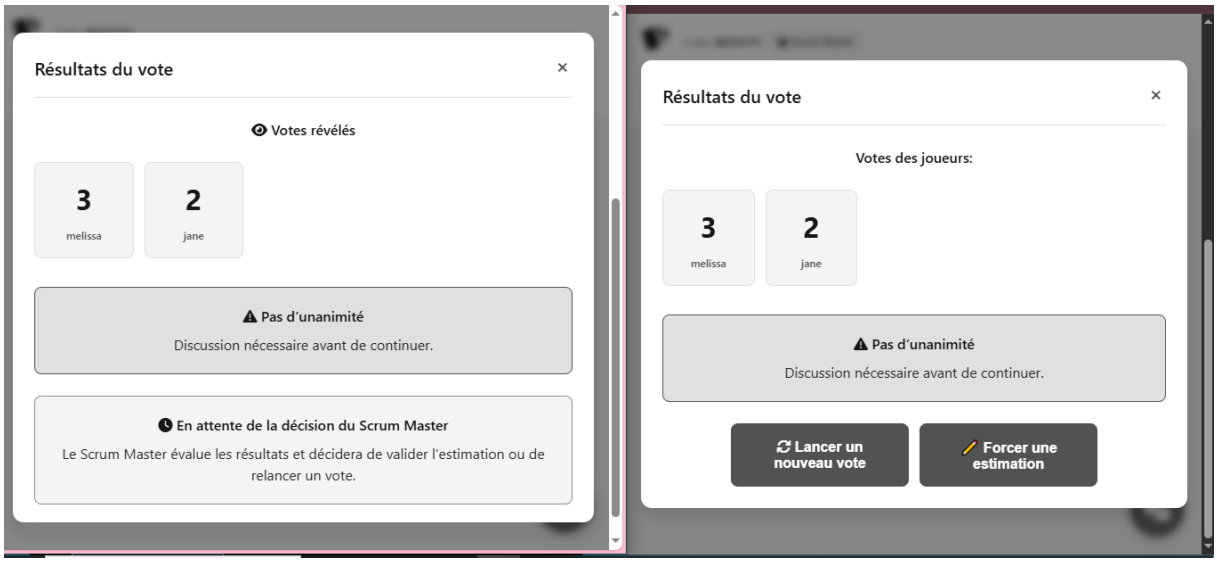

FIGURE 20 – Révélation des votes

Le Scrum Master clique sur "Révéler les votes" pour afficher tous les résultats.

4.2.6 Validation

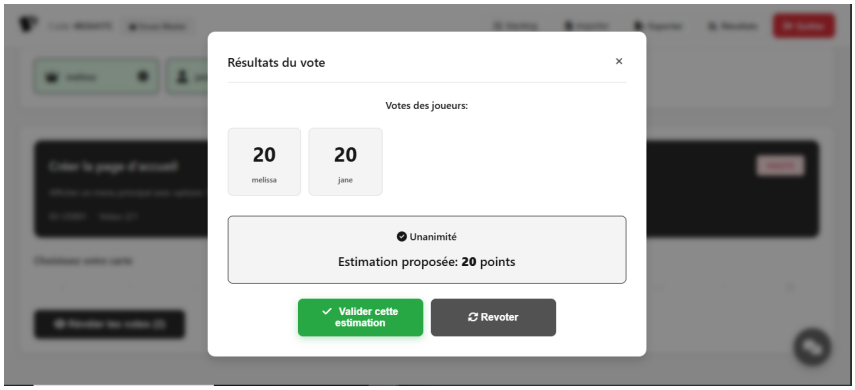

FIGURE 21 – Cas unanimité

L'application calcule automatiquement l'estimation selon la règle définie.

4.2.7 Fonctionnalité pause café

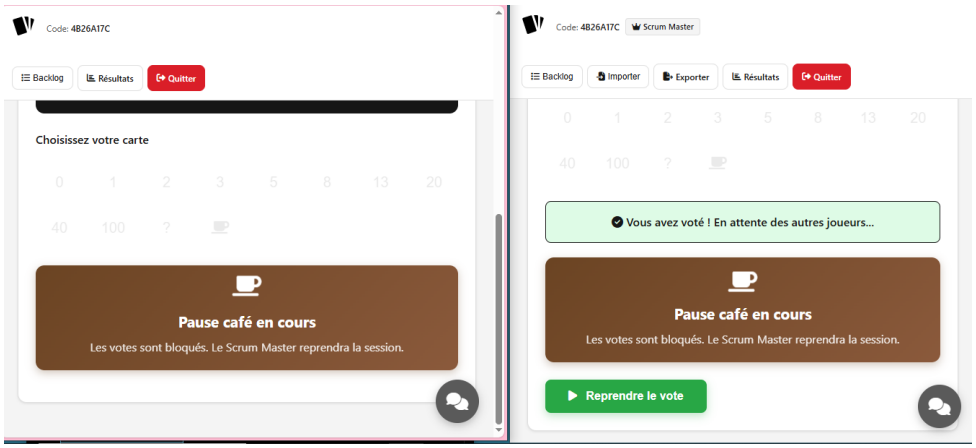

FIGURE 22 – Mode pause café

Le Scrum Master peut mettre la session en pause pour une pause café. Tous les participants voient l'écran de pause.

4.2.8 Résultats et Export du backlog

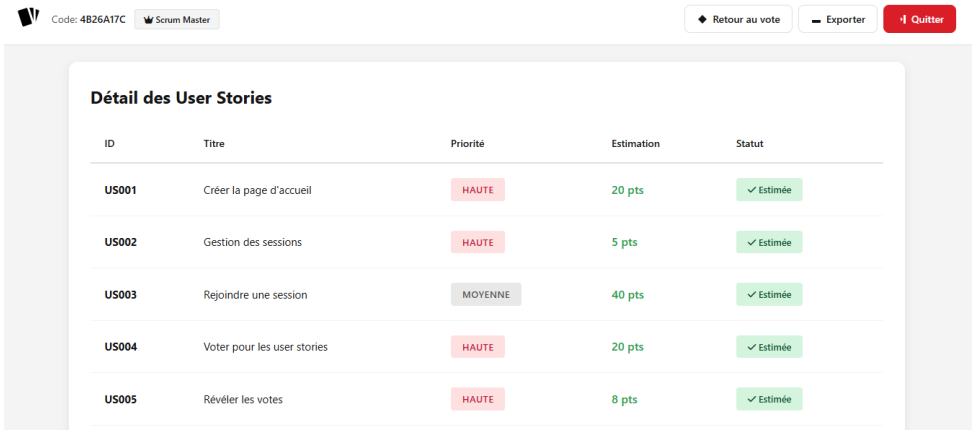

FIGURE 23 – Résultats

Le Scrum Master peut exporter le backlog avec toutes les estimations au format JSON.

4.3 Modes de Vote Détaillés

4.3.1 Mode Strict (Unanimité)

Règle :

- **Premier tour** : L'estimation n'est validée que si TOUS les participants ont voté la même valeur
- **Tours suivants** : Si pas d'unanimité au premier tour, un nouveau vote est lancé. Après discussion, si toujours pas d'unanimité, la moyenne est calculée.

Exemple :

Joueur	Vote T1	Vote T2	Vote T3	Résultat
Alice	5	5	5	Unanimité : 5
Bob	5	5	5	
Charlie	5	5	5	

TABLE 5 – Exemple mode strict avec unanimité

Joueur	Vote T1	Vote T2	Vote T3	Résultat
Alice	5	8	5	Moyenne : 6
Bob	8	5	8	
Charlie	3	8	5	

TABLE 6 – Exemple mode strict sans unanimité

4.3.2 Mode Moyenne

Formule :

$$\text{Estimation} = \text{round}\left(\frac{\sum_{i=1}^{n} v_i}{n}\right)$$

où v_i sont les votes et n le nombre de participants.
Exemple :
Votes : 3, 5, 8, 8

$$\text{Estimation} = \text{round}\left(\frac{3+5+8+8}{4}\right) = \text{round}(6) = 6$$

4.3.3 Mode Médiane

Règle : La médiane est la valeur centrale après tri.
Exemple :
Votes : 3, 5, 8, 13
Votes triés : [3, 5, 8, 13]

$$\text{Médiane} = \frac{5+8}{2} = 6.5 \approx 7$$

4.3.4 Mode Majorité Absolue

Règle : Une valeur doit avoir $> 50\%$ des votes.
Exemple :
5 joueurs votent : 8, 8, 8, 5, 3
8 apparaît 3 fois sur $5 = 60\% \rightarrow$ Estimation $= 8$

4.3.5 Mode Majorité Relative

Règle : La valeur la plus votée l'emporte.
Exemple :
Votes : 5, 5, 8, 8, 8, 13
8 apparaît 3 fois (le plus) → Estimation = 8

5 Conclusion

5.1 Objectifs atteints

Ce projet a permis de développer une application web complète et fonctionnelle de Planning Poker avec :

- ✓ Implémentation complète des règles de vote (strict, moyenne, médiane, majorité)
- ✓ Interface utilisateur moderne et intuitive
- ✓ Synchronisation en temps réel entre participants
- ✓ Chat intégré pour faciliter les discussions
- ✓ Notifications : Alertes en temps réel
- ✓ Import/Export JSON du backlog
- ✓ Pipeline CI/CD complet avec tests et documentation automatisés
- ✓ Couverture de tests $> 80\%$
- ✓ Documentation API complète et accessible
- ✓ Déploiement en production

5.2 Compétences acquises

Le développement de ce projet a permis de renforcer nos compétences dans :

— **Architecture logicielle** : Application rigoureuse du pattern MVC
— **Qualité du code** : Tests unitaires, analyse statique, documentation
— **DevOps** : Configuration et utilisation de GitHub Actions pour la CI/CD
— **Travail collaboratif** : Utilisation efficace de Git et méthodologie agile
— **Design UX/UI** : Création d'une interface utilisateur moderne et responsive

5.3 Améliorations futures

Plusieurs pistes d'amélioration ont été identifiées pour étendre les fonctionnalités :

1. **Authentification** : Système de comptes utilisateurs avec historique
2. **Statistiques** : Tableaux de bord avec métriques d'équipe
3. **API REST complète** : Pour intégration avec d'autres outils agiles
4. **Multi-sessions** : Gestion de plusieurs sessions simultanées par utilisateur
5. **Export PDF** : Génération de rapports au format PDF

A Annexes

— **Application en ligne** : https://planningpoker.infinityfreeapp.com/Planning_poker/public/index.php
— **Code source GitHub** : https://github.com/melissa-aliouche/Planning_poker
— **Documentation API** : https://melissa-aliouche.github.io/Planning_poker/
— **PHPUnit Documentation** : https://phpunit.de/documentation.html
— **PHPStan** : https://phpstan.org/
— **GitHub Actions** : https://docs.github.com/en/actions