Université Lumière Lyon 2
Projet Académique

Réalisé par :

Amadou Macka Diallo

Cyrine Nighaoui

Dans le cadre du :

Master 1 – Informatique

Planning Poker Application

Intitulé du projet :
Développement d'une application de Planning Poker pour l'estimation agile

Année Universitaire : **2025–2026**

Université Lumière Lyon 2

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Avec l'essor des méthodes agiles, l'estimation des charges de travail est devenue une étape clé dans la gestion des projets informatiques. Parmi les techniques les plus utilisées, le *Planning Poker* s'impose comme une méthode collaborative permettant aux équipes de parvenir à une estimation partagée, fondée sur l'expertise collective et la discussion.

Cependant, dans un contexte de travail de plus en plus distribué et hybride, les séances de Planning Poker traditionnelles présentent certaines limites : manque d'outils interactifs adaptés au temps réel, difficultés de synchronisation entre les participants, et absence de support intelligent pour analyser les divergences d'estimation. Ces contraintes peuvent nuire à la fluidité des échanges et à la qualité des décisions prises par l'équipe.

C'est dans ce contexte que s'inscrit ce projet, dont l'objectif est de concevoir et développer une application web de Planning Poker collaborative, interactive et temps réel. L'application permet à des équipes agiles de créer des salles de vote, gérer un backlog de tâches, voter simultanément, discuter via un chat intégré et révéler les estimations de manière synchronisée. Elle repose sur une architecture moderne combinant un backend robuste basé sur Django et Django REST Framework, un frontend réactif développé avec Next.js et React, ainsi qu'un système de communication temps réel utilisant les WebSockets.

Ce rapport présente l'ensemble du travail réalisé, depuis l'analyse des besoins et la conception du système, jusqu'à l'implémentation technique et la mise en place des fonctionnalités avancées. Il met en évidence les choix architecturaux, technologiques et méthodologiques effectués, ainsi que les défis rencontrés lors du développement de cette application temps réel.

Chapitre 1

Contexte général du projet

1.1 Contexte

L'estimation des tâches est un pilier des méthodes Agiles, notamment Scrum, car elle permet de planifier le travail et d'évaluer l'effort nécessaire aux fonctionnalités. Le Planning Poker est une technique largement utilisée pour son approche collaborative et sa capacité à limiter les biais individuels.

Toutefois, avec la généralisation du télétravail, les outils traditionnels montrent leurs limites. Ce projet s'inscrit donc dans la création d'une plateforme web interactive de Planning Poker en ligne, conçue pour faciliter la collaboration en temps réel .

1.2 Problématique

Malgré l'existence d'outils collaboratifs, plusieurs difficultés subsistent :

— manque de fluidité dans la synchronisation en temps réel ;

— absence de fonctionnalités avancées comme un chat intégré, un système de rôles ou la gestion automatique des votes ;

— limitation des solutions existantes face aux besoins spécifiques des équipes Agile (confidentialité du vote, révélations synchronisées...) ;

La problématique principale peut donc être formulée ainsi :

Comment concevoir et développer une application Web permettant de réaliser des sessions de Planning Poker en temps réel, favorisant la collaboration, l'interaction et une expérience fluide adaptée aux besoins des équipes Agile modernes ?

1.3 Objectifs du projet

Les objectifs de ce projet se déclinent en objectifs généraux et spécifiques :
Objectif général :

— Concevoir et développer une plateforme Web interactive permettant l'estimation Agile via le Planning Poker.

Objectifs spécifiques :

— Permettre la création et la gestion de sessions d'estimation ;

— Intégrer un système de vote confidentiel avec révélation synchronisée ;

— Assurer une communication temps réel à travers WebSockets ;

— Intégrer un chat pour améliorer la collaboration ;

— Fournir une interface intuitive, moderne et responsive.

1.4 Méthodologie adoptée

Pour mener à bien le projet, une démarche Agile itérative et incrémentale a été adoptée. Le développement a été découpé en plusieurs sprints, incluant :

— une phase d'analyse fonctionnelle et technique ;

— la conception de l'architecture (backend, base de données, communication temps réel) ;

— le développement progressif des fonctionnalités clés ;

— la réalisation de tests et améliorations continues basées sur les retours ;

— l'intégration de modules complémentaires (chat,dashboard Admin).

L'approche Agile permettait d'adapter le périmètre du projet en fonction des contraintes techniques, des nouvelles idées, ainsi que de l'évolution des besoins fonctionnels.

1.5 Organisation du travail et Pair Programming

Le projet a été réalisé en **binôme**, en adoptant une approche de *Pair Programming*. Cette méthode de travail collaborative repose sur le développement conjoint du code par deux membres de l'équipe, en alternant régulièrement les rôles de *driver* (chargé de l'écriture du code) et de *navigator* (chargé de l'analyse, de la relecture et de la validation des choix techniques).

L'utilisation du Pair Programming a permis :

— d'améliorer la qualité globale du code grâce à une relecture continue ;

— de réduire les erreurs logiques et techniques dès les phases de développement ;

— de favoriser la prise de décision collective sur les choix d'architecture et de conception ;

— d'assurer une meilleure répartition des tâches et une montée en compétence équilibrée des membres du binôme.

Le suivi du développement et la collaboration ont été assurés via la plateforme GitHub, en respectant les bonnes pratiques de versioning (commits réguliers, revues de code..).

Chapitre 2

État de l'art

2.1 Méthodes d'estimation Agile

Les méthodes Agiles visent à améliorer la flexibilité, la collaboration et la capacité d'adaptation des équipes de développement logiciel. Au sein de Scrum, l'estimation constitue une étape essentielle, permettant de planifier le contenu d'un sprint et d'évaluer la complexité des tâches.

Plusieurs techniques existent, parmi lesquelles :

— **T-Shirt Sizing** : méthode rapide catégorisant les tâches (S, M, L, XL) sans granularité fine ;

— **Three-Point Estimation** : basée sur trois valeurs (optimiste, pessimiste, réaliste) permettant de calculer une moyenne pondérée ;

— **Planning Poker** : méthode collaborative reposant sur la discussion collective et un vote simultané.

2.2 Outils existants et limitations

Plusieurs solutions numériques de Planning Poker existent déjà, telles que *PlanningPoker.com* ou *Scrumpoker Online*. Toutefois, ces outils présentent certaines limites :

— fonctionnalités limitées (absence de chat intégré, manque d'interactivité) ;

— interfaces peu personnalisables ou expérience utilisateur peu immersive.

Ces limitations rendent ces outils peu adaptés à des contextes pédagogiques ou à des équipes distribuées nécessitant une interaction riche, un suivi des participants et une expérience utilisateur plus engageante.

2.3 WebSockets et communication temps réel

La communication temps réel est au cœur des applications collaboratives modernes. Contrairement au protocole HTTP, basé sur un modèle requête/réponse, les *WebSockets* permettent d'établir une connexion persistante entre le client et le serveur, offrant notamment :

— la diffusion instantanée des votes ;

— la synchronisation de la révélation des estimations ;

— la mise à jour dynamique des participants connectés ;

— un chat en direct.

L'usage des WebSockets constitue donc un choix technologique pertinent pour assurer une expérience fluide et collaborative dans une application de Planning Poker en ligne.

Ainsi, malgré l'existence de solutions de Planning Poker en ligne, peu d'entre elles proposent une plateforme réellement interactive, temps réel et orientée vers une collaboration avancée. Ce constat met en évidence l'intérêt de concevoir une solution dédiée, combinant interactivité, communication synchrone et simplicité d'utilisation.

Chapitre 3

Spécifications du projet

Ce chapitre présente les éléments essentiels à la définition du projet, notamment la compréhension des besoins, l'analyse des utilisateurs, les exigences fonctionnelles et non fonctionnelles ainsi que l'organisation du travail à travers un backlog et une planification Agile.

3.1 Capture des besoins

L'objectif principal du projet est de concevoir une application web permettant d'effectuer des séances de *Planning Poker* en ligne, de manière collaborative, interactive et en temps réel. Cette plateforme vise à faciliter l'estimation Agile dans un contexte de travail hybride ou distribué.

Les besoins exprimés incluent :

— la possibilité de créer ou rejoindre une salle de vote ;

— une interface simple d'utilisation pour voter ;

— un système de synchronisation instantané entre les utilisateurs ;

— des outils complémentaires (dashboard administrateur) pour améliorer la communication ;

— la gestion fluide d'une session d'estimation du début à la fin.

3.1.1 Identification des acteurs

Le système comporte deux principaux types d'utilisateurs :

— **Administrateur de salle** : crée la salle, gère l'ordre des tâches, lance la révélation des votes, gère les configurations et peut expulser des utilisateurs si nécessaire.

— **Participant** : rejoint une salle existante, vote, interagit via le chat et participe à l'estimation.

3.1.2 Besoins fonctionnelles

Les besoins fonctionnels décrivent les actions que le système doit permettre. Parmi les fonctionnalités essentielles :

— création et gestion des salles de vote ;

— système de connexion et authentification ;

— sélection d'une carte représentant une estimation ;

— révélation synchronisée des votes ;

— chat en temps réel ;

— affichage de l'état des utilisateurs (présent, connecté, voté, etc.) ;

— gestion du backlog et des participants via dashboard admin.

3.1.3 Besoins non fonctionnelles

Ce projet respecte également des contraintes non fonctionnelles :

— **Performance** : latence minimale grâce à WebSockets.

— **Sécurité** : gestion des tokens JWT pour authentification.

— **Scalabilité** : possibilité d'héberger plusieurs salles simultanément.

— **Expérience utilisateur (UX)** : interface conviviale, fluide et moderne.

3.2 Product Backlog et User Stories

3.2.1 Product Backlog

Le tableau suivant présente le backlog fonctionnel du projet sous forme de User Stories, priorisées et réparties par sprint .

ID	Fonctionnalité	User Story (Description)	Priorité	Sprint
1	Créer un compte	En tant que nouvel utilisateur, je peux créer un compte afin de participer à des sessions de planning poker.	Haute	S1
2	Se connecter	En tant qu'utilisateur existant, je peux me connecter pour accéder à mes sessions précédentes et voter.	Haute	S1
3	Créer une réunion	En tant que product owner, je peux créer une session de planning poker pour estimer le backlog avec l'équipe.	Haute	S2
4	Intégrer une réunion	En tant que joueur, je peux rejoindre une session en cours pour participer aux votes.	Haute	S2
5	Gérer session	En tant qu'admin je peux retirer des membres , modifier , ajouter et supprimer des élément du backlog .	Moyenne	S2
6	Inviter un membre	En tant que product owner, je peux inviter un membre à une session afin qu'il puisse participer.	Haute	S3
7	Voter	En tant que joueur, je peux sélectionner une carte pour voter la complexité d'une user story.	Haute	S2
8	Discuter dans un canal	En tant que joueur, je peux échanger avec les autres membres dans un canal de discussion pour justifier ou discuter les votes.	Basse	S3
9	Importer le backlog	En tant que product owner, je peux importer un backlog en JSON pour préparer la session.	Haute	S2
10	Exporter le backlog	En tant que product owner, je peux exporter un fichier JSON avec les estimations pour sauvegarder la session.	Haute	S2

11	Choisir une règle de vote	En tant que product owner, je peux choisir entre différentes règles de vote (unanimité, moyenne, médiane, etc.) afin d'adapter la session à notre méthode.	Haute	S2
12	Carte "Café"	En tant que joueur, je peux jouer la carte café pour demander une pause .	Moyenne	S4
13	Consulter le dashboard	En tant que product owner, je peux consulter le tableau de bord de la session (votes, avancement, participants).	Basse	S6

TABLE 3.1 – Backlog fonctionnel du projet

3.2.2 Planification des sprints (Agile)

Le développement a été exécuté selon une planification évolutive répartie en sprints, permettant une implémentation incrémentale.

Table 3.2 – Planification des Sprints du projet

Sprint	Période	Objectifs principaux
Sprint 1	20/10/2025 – 02/11/2025	Analyse des besoins, gestion des utilisateurs, création/connexion aux salles, mise en place de l'environnement de développement (Django/Next.js).
Sprint 2	03/11/2025 – 16/11/2025	Architecture WebSockets, affichage des joueurs connectés, sélection de cartes et gestion des votes.
Sprint 3	17/11/2025 – 30/11/2025	Révélation des votes et carte "Café" .
Sprint 4	01/12/2025 – 14/12/2025	Gestion du backlog administrateur (CRUD), sécurisation via JWT.
Sprint 5	15/12/2025 – 28/12/2025	Chat en temps réel, interface utilisateur avancée, indicateurs d'état (connecté/voté).
Sprint 6	29/12/2025 – 15/01/2026	Tests, optimisation UI/UX.

3.3 Environnement de développement matériel

Le tableau 3.1 montre les caractéristiques des ordinateurs utilisés pour réaliser le projet .

Ordinateur : Lenovo LOQ	**Ordinateur : LENOVO**
Processeur : 12th Gen Intel(R) Core(TM) i7-12650HX	**Processeur :** IDEAPAD GAMING 3 15ACH6 / RYZEN 5 5600H
Mémoire vive : 16 Go	**Mémoire vive :** 32 Go
Carte Graphique : NVIDIA GeForce RTX 4050 Laptop GPU (6 GB), Intel(R) UHD Graphics (128 MB)	**Carte Graphique :** NVIDIA GeForce RTX 3050, 4 Go de mémoire dédiée GDDR6
Version Next : 15.3.1	**Version Next :** 15.3.1
Système d'exploitation : Windows 10 famille	**Système d'exploitation :** Windows 11

Table 3.3 – Environnement de développement matériel.

3.4 Choix technologiques et justification

Les choix techniques ont été effectués selon trois critères : performance, évolutivité et disponibilité des ressources techniques.

Technologie	**Rôle**	**Justification**
Next.js / React	Interface utilisateur	Modernité, performance, composants réutilisables, support WebSocket.
Django REST Framework	API Backend	Rapidité de développement, sécurité intégrée, ORM performant.
Django Channels + Redis	Communication temps réel	Gestion native WebSocket, haute scalabilité.
PostgreSQL	Base de données	Fiabilité, transactions, support JSON.
GitHub	Gestion version	Collaboration, versioning

Table 3.4 – Synthèse des choix technologiques.

3.5 Structure des fichiers JSON

L'application de Planning Poker utilise des fichiers JSON pour faciliter l'importation du backlog et la préparation des sessions d'estimation. Le format JSON a été retenu pour sa simplicité, sa lisibilité et sa compatibilité native avec les technologies web utilisées (Django, JavaScript).

3.5.1 Structure du fichier de backlog

Le fichier de backlog est constitué d'une liste d'objets JSON, chaque objet représentant une tâche ou une fonctionnalité à estimer lors de la session de Planning Poker.

Chaque élément du backlog est défini par les champs suivants :

— `id` : identifiant unique de la tâche ;

— `title` : intitulé court de la fonctionnalité à estimer ;

— `description` : description fonctionnelle de la tâche.

Un exemple de structure de fichier de backlog est présenté ci-dessous :

FIGURE 3.1 – fichier de backlog

Ce fichier est importé par l'administrateur de la session via l'interface dédiée. Les tâches sont ensuite stockées en base de données et synchronisées en temps réel avec l'ensemble des participants connectés.

3.5.2 Évolution du backlog après estimation

Une fois la session de Planning Poker terminée, chaque tâche peut être enrichie par une valeur d'estimation issue du vote collectif. Le même format JSON peut alors être utilisé pour l'export des résultats, avec l'ajout d'un champ d'estimation, permettant de conserver une trace des décisions prises par l'équipe.

3.6 Architecture

L'architecture du projet a été conçue de manière modulaire afin de permettre une communication fluide entre le frontend, le backend et les services temps réel. Elle s'appuie sur une approche cliente-serveur enrichie par une couche WebSocket permettant la synchronisation instantanée des actions utilisateurs (votes, chat, présence, révélations).

La figure 3.2 illustre la structure globale du système, ses principaux composants ainsi que les interactions entre eux.

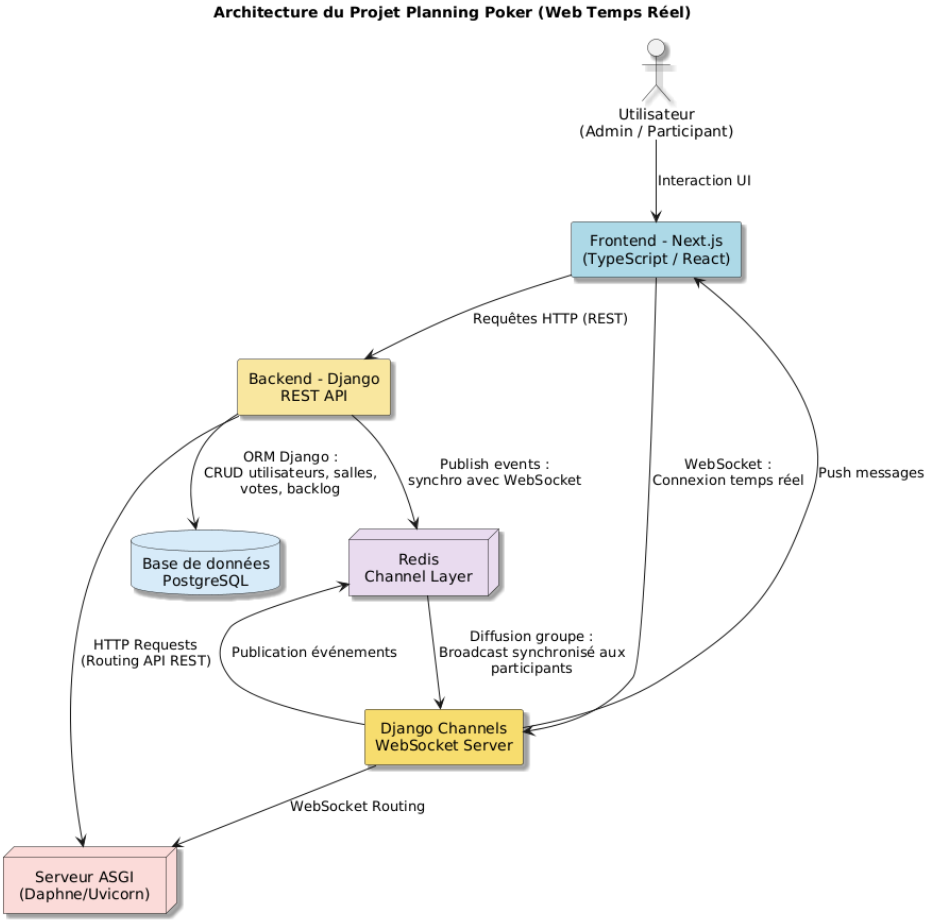

FIGURE 3.2 – Diagramme de cas d'utilisation global.

Chapitre 4

Conception et Implémentation

Ce chapitre décrit la transition entre les spécifications fonctionnelles du projet et leur matérialisation sous forme d'un système logiciel opérationnel. Il présente la modélisation UML réalisée, l'architecture adoptée ainsi que les choix d'implémentation retenus. L'objectif est de démontrer la cohérence entre les besoins exprimés et la solution implémentée, tout en garantissant maintenabilité, robustesse et interactions en temps réel.

4.1 Modélisation Fonctionnelle

La modélisation fonctionnelle permet de représenter de manière abstraite le comportement attendu du système ainsi que les interactions entre acteurs. Elle constitue une étape essentielle avant la phase d'implémentation afin de garantir l'alignement entre les objectifs fonctionnels et la structure du logiciel.

4.1.1 Diagramme des Cas d'Utilisation

Le diagramme des cas d'utilisation présenté ci-dessous permet d'illustrer les interactions entre les deux utilisateurs principaux du système : l'*Administrateur de salle* et le *Participant*.

FIGURE 4.1 – Diagramme des cas d'utilisation du système.

4.1.2 Description textuelle des cas d'utilisation

Les tableaux suivants décrivent les principaux cas d'utilisation conformément aux spécifications définies.

Les tableaux suivants détaillent les cinq cas d'utilisation principaux du système.

UC1 : Créer une salle

Nom	Créer une salle
Acteur principal	Administrateur
Préconditions	L'utilisateur doit être authentifié.

Description	L'utilisateur crée une nouvelle session de Planning Poker. Le système génère automatiquement un code unique et initialise la salle.
Post-conditions	Une salle valide est créée et prête à recevoir des participants.

UC2 : Rejoindre une salle

Nom	Rejoindre une salle
Acteur principal	Participant
Préconditions	La salle doit exister et être accessible.
Description	Le participant saisit le code de la session et rejoint la salle. Le système met à jour en temps réel la liste des participants.
Post-conditions	Le participant apparaît dans la session WebSocket et peut interagir.

UC3 : Voter une estimation

Nom	Voter une estimation
Acteur principal	Participant
Préconditions	Le participant doit être authentifié et connecté à une salle de Planning Poker active, avec une tâche courante définie.
Description	Le participant sélectionne une carte représentant l'effort estimé pour la tâche courante (valeurs numériques ou carte *Café*).

Scénarios alternatifs	— **Carte Café** : si le participant sélectionne la carte *Café*, cela indique une demande de pause . La session est temporairement suspendue. — **Non-unanimité** : si la règle de vote est définie sur *strict* et que les votes ne sont pas identiques, un revote est automatiquement déclenché.
Post-conditions	Le vote est enregistré en base de données, pris en compte dans la session courante et visible par l'ensemble des participants.

UC4 : Révéler les votes

Nom	Révéler les votes
Acteur principal	Administrateur
Préconditions	Tous les participants doivent avoir voté.
Description	L'administrateur déclenche la révélation des votes. Le backend calcule le résultat selon la règle choisie (médiane, majorité, moyenne, unanimité).
Post-conditions	Les estimations deviennent visibles et la session passe à la tâche suivante.

UC5 : Gérer le backlog

Nom	Gérer le backlog
Acteur principal	Administrateur
Préconditions	Une salle doit être active.

Description	L'administrateur peut ajouter, modifier, supprimer ou réorganiser les tâches.
Post-conditions	Le backlog mis à jour est synchronisé en temps réel pour l'ensemble des participants.

4.1.3 Diagramme de Séquence du Processus de Vote

Le diagramme suivant illustre le déroulement d'un vote, depuis l'action de l'utilisateur jusqu'à la diffusion en temps réel de l'état de la session.

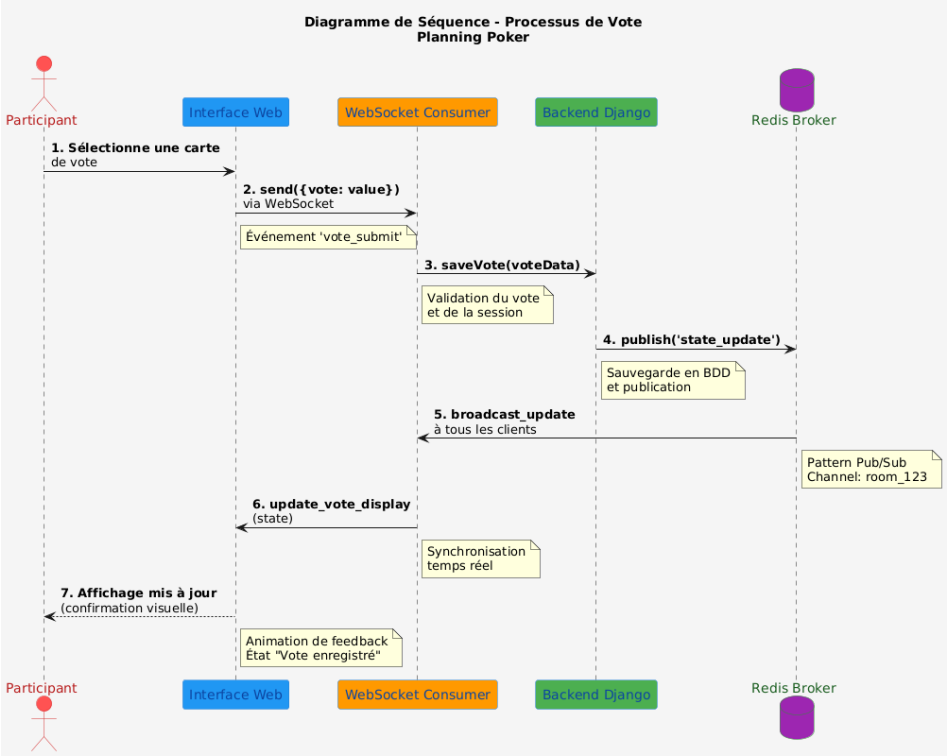

FIGURE 4.2 – Diagramme de séquence : processus de vote.

4.1.4 Diagramme d'Activités

Ce diagramme représente le flux global d'une session de Planning Poker, incluant vote, révélation et passage à la tâche suivante.

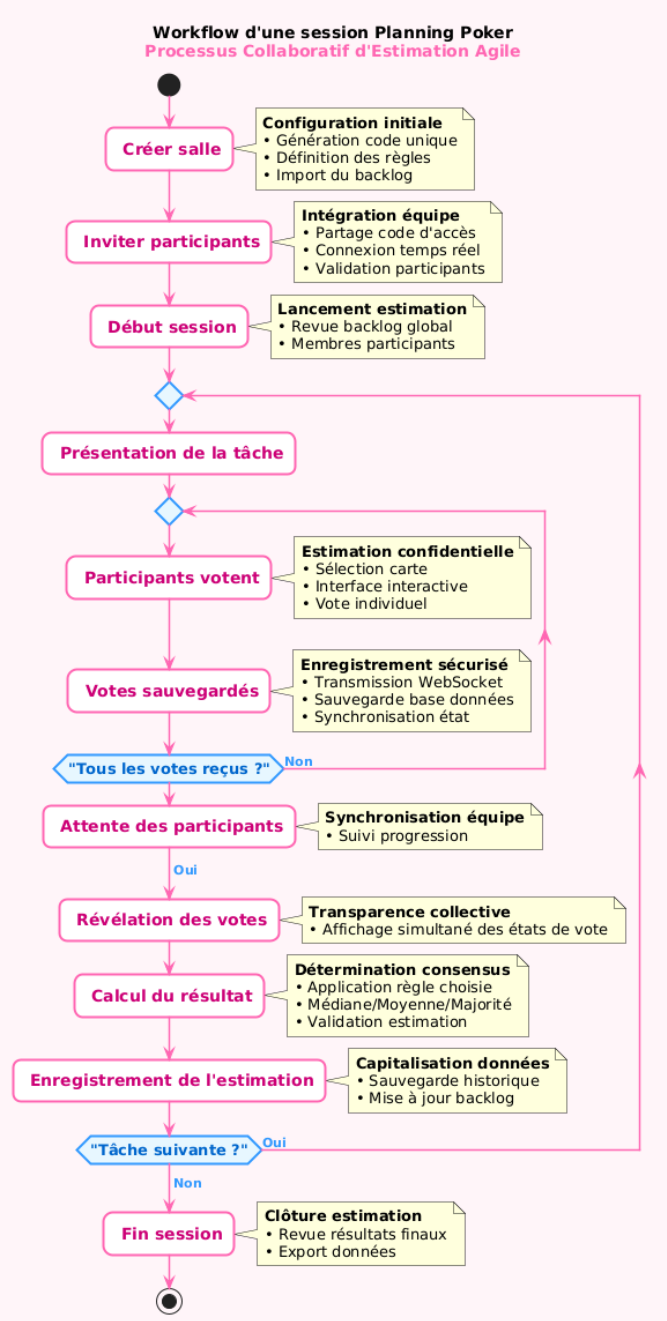

FIGURE 4.3 – Diagramme d'activités : déroulement d'une session.

4.2 Modélisation Conceptuelle

La modélisation conceptuelle repose sur un diagramme de classes représentant les entités métier du système ainsi que leurs associations.

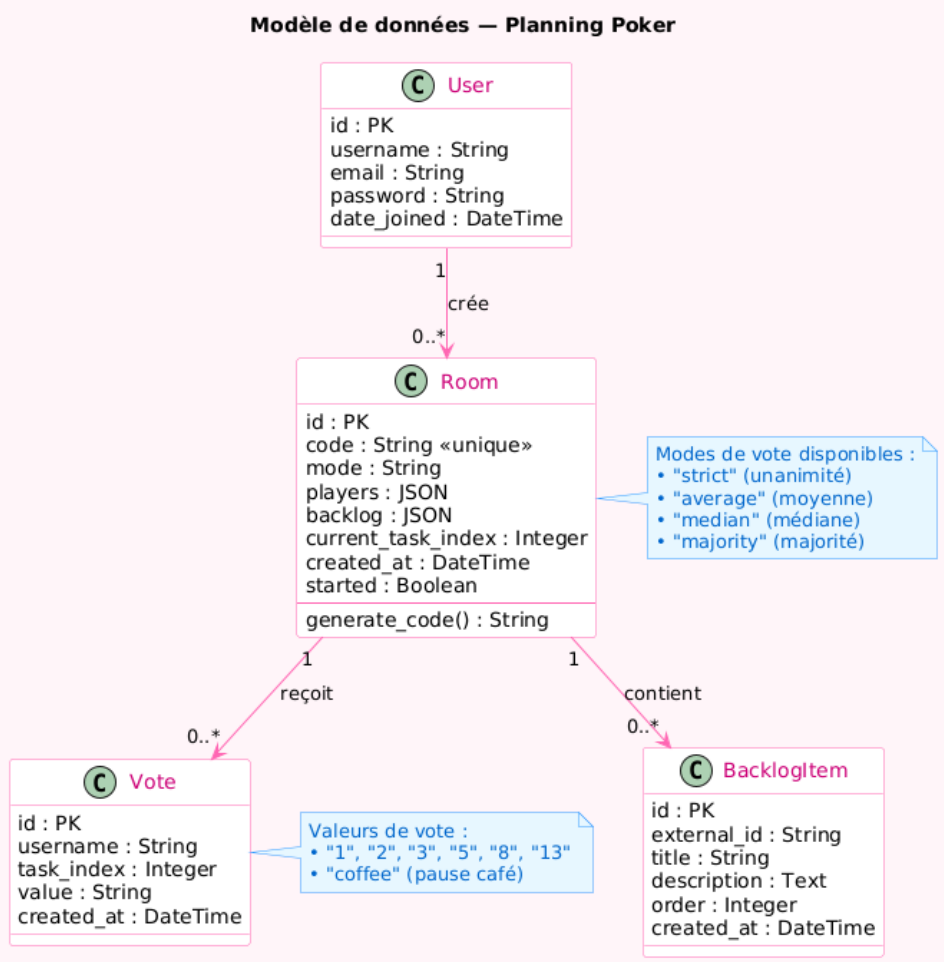

FIGURE 4.4 – Diagramme de classes du modèle de données.

4.3 Architecture et Déploiement

Une architecture distribuée a été adoptée afin de garantir temps réel, scalabilité et modularité. Le système repose sur quatre modules principaux : Frontend (Next.js), Backend (Django REST), WebSockets (Django Channels) et un broker Redis.

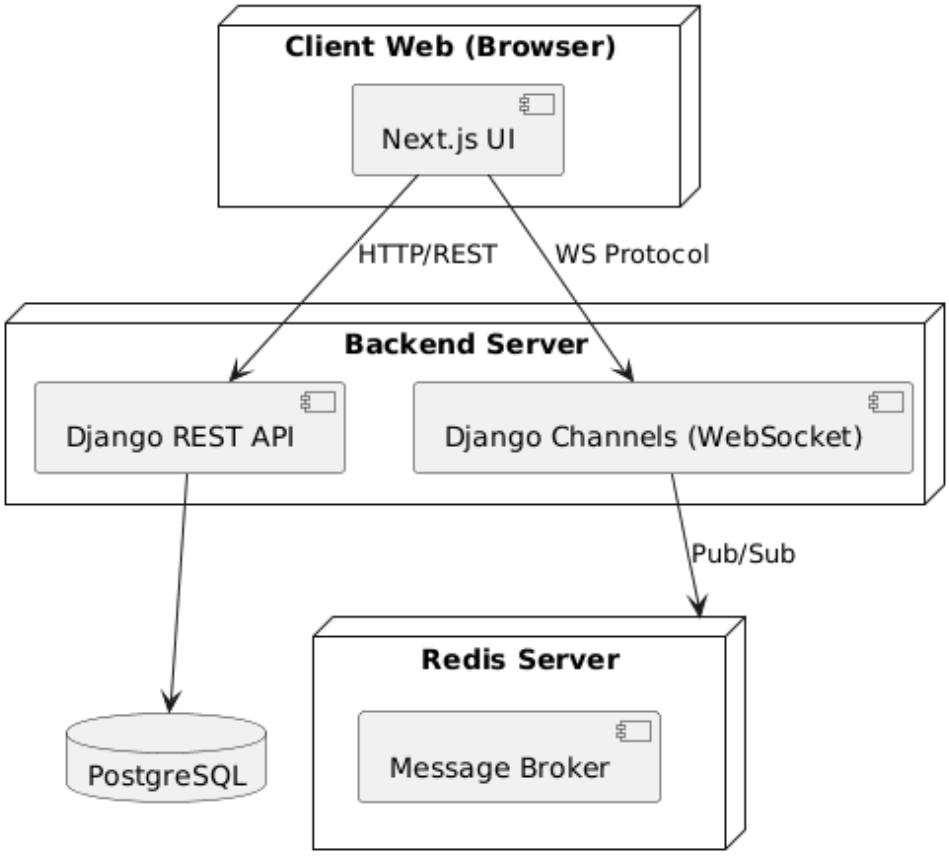

FIGURE 4.5 – Diagramme de déploiement du système.

4.4 Implémentation

L'implémentation du système a été réalisée en suivant une architecture modulaire, facilitant l'évolution et la maintenance du projet.

4.4.1 Authentification et gestion des accès

L'authentification repose sur JWT (JSON Web Token). Les jetons émis permettent d'accéder à l'API REST ainsi qu'au serveur WebSocket. Les permissions sont gérées côté backend afin de sécuriser l'accès aux salles.

4.4.2 Gestion des salles et des sessions

La création et la gestion des sessions Planning Poker sont effectuées via des endpoints REST. Le contrôleur WebSocket assure la synchronisation dynamique des événements (arrivée d'utilisateurs, votes, révélations).

4.4.3 Communication en temps réel

La communication temps réel utilise Django Channels couplé à Redis. Chaque modification (vote, message, mise à jour backlog) est instantanément diffusée à l'ensemble des clients connectés.

4.5 Résumé du Chapitre

Ce chapitre a présenté la conception et l'implémentation du système développé. À travers les modèles UML, l'architecture distribuée et une implémentation modulaire, la solution obtenue répond aux besoins fonctionnels et garantit une interaction fluide en temps réel .

Chapitre 5

Conception et Implémentation du Frontend

Ce chapitre présente la conception et le développement de l'interface utilisateur de la plateforme Planning Poker. L'objectif du frontend est d'offrir une expérience fluide, intuitive et responsive tout en assurant une interaction en temps réel avec le backend.

Le frontend a été réalisé avec **Next.js 15.5.5**, combiné à des WebSockets (Django Channels) pour la communication instantanée, et une gestion d'état permettant l'affichage synchronisé des votes, de la liste des participants et du backlog.

5.1 Architecture Frontend

L'architecture frontend repose sur une organisation par composants réutilisables. Elle suit une logique modulaire organisant l'application en pages, composants et services.

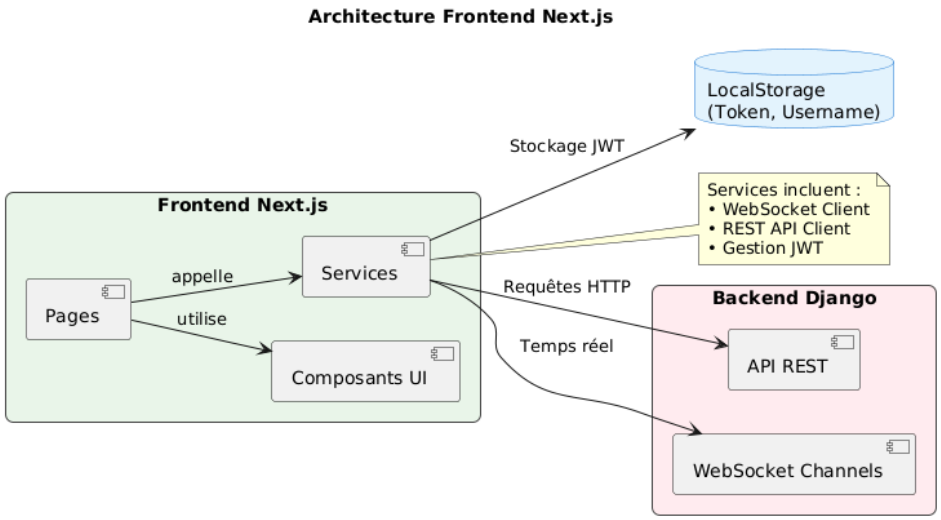

FIGURE 5.1 – Architecture du frontend Next.js.

— **Pages** : accès principal à l'application (connexion, salle, vote, backlog).
— **Composants** : boutons de vote, tableau des joueurs, chat.
— **Services** : gestion WebSocket, API REST, stockage JWT.

5.2 Conception UI/UX

Une attention particulière a été portée sur l'ergonomie afin d'assurer :

— une interface claire et minimaliste,
— une visualisation instantanée des votes et du statut des utilisateurs,
— une disposition intuitive des éléments interactifs,
— un design cohérent basé sur un thème moderne et épuré.

5.2.1 Maquettes et Interface Développée

L'interface graphique a été conçue de manière à guider progressivement l'utilisateur depuis son arrivée sur l'application jusqu'à la fin d'une session d'estimation agile. Les écrans présentés ci-dessous suivent le déroulement naturel d'une utilisation typique : accueil, authentification, choix du mode, participation à une salle, vote et consultation des résultats.

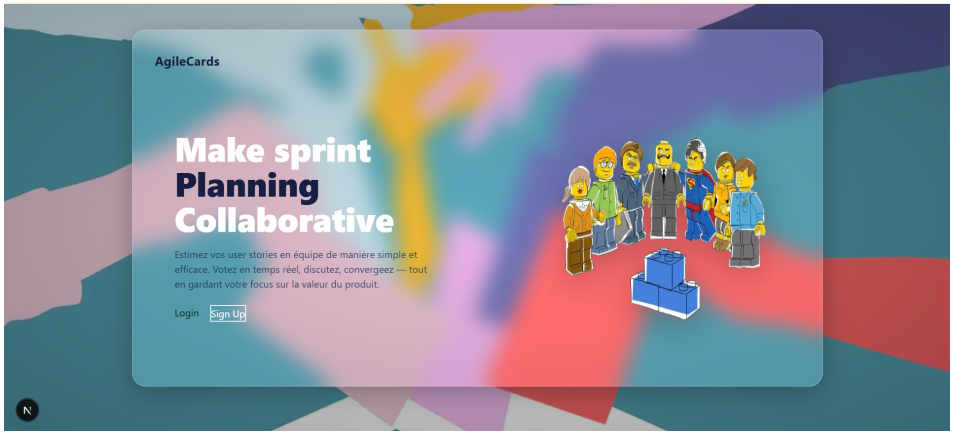

FIGURE 5.2 – Page d'accueil permettant d'accéder à l'application.

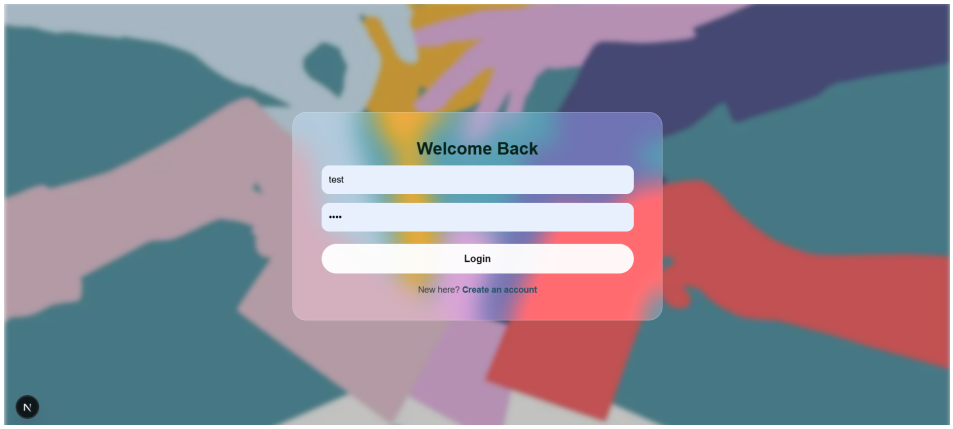

FIGURE 5.3 – Écrans de connexion pour accéder à la plateforme.

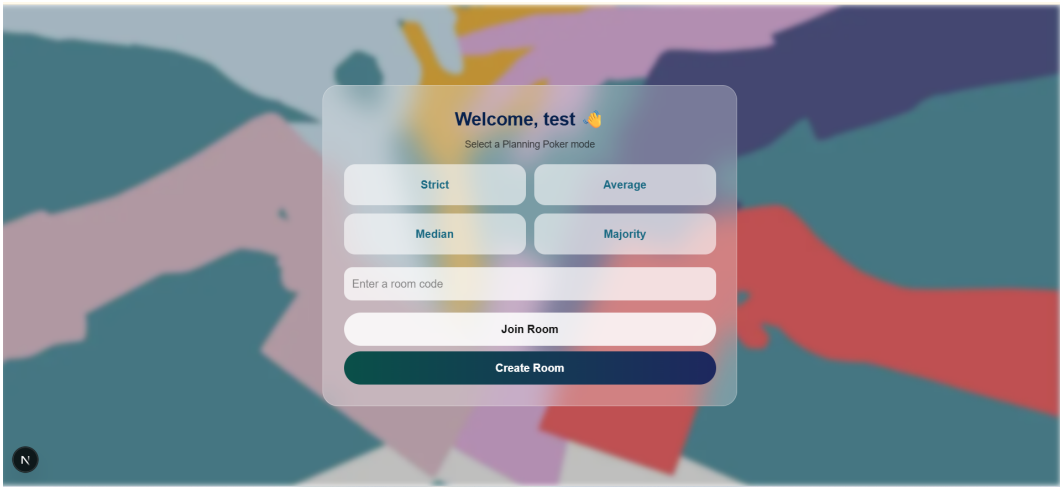

FIGURE 5.4 – Sélection du mode d'estimation et création ou joindre d'une salle.

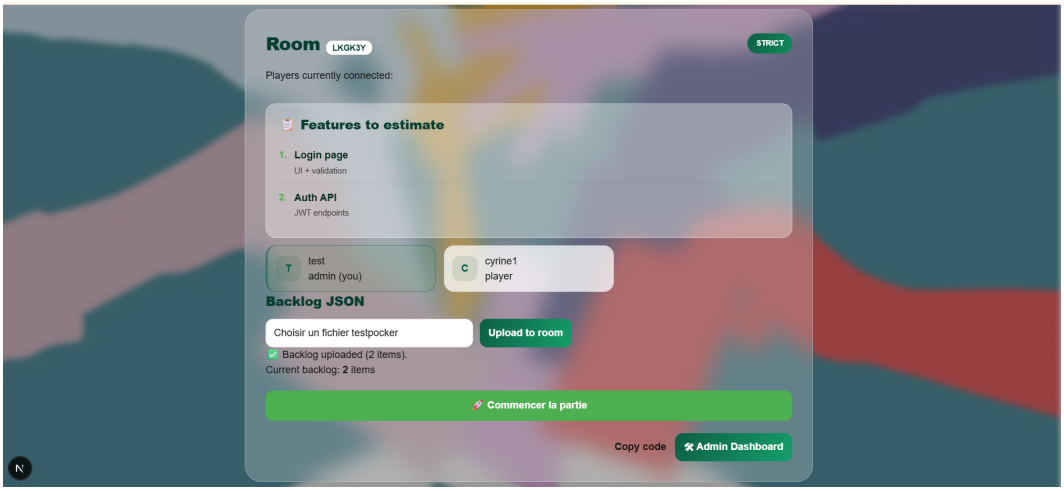

FIGURE 5.5 – Lobby de salle affichant les joueurs connectés et le backlog importé.

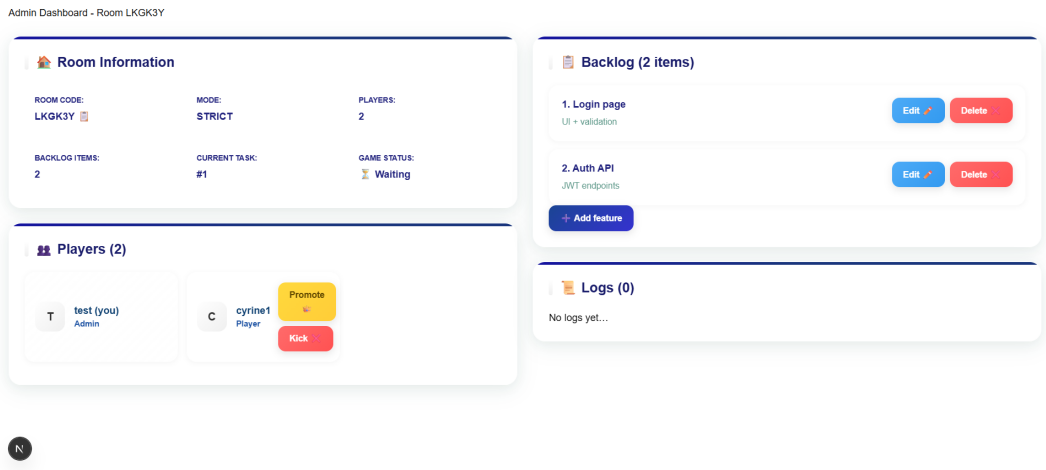

FIGURE 5.6 – Dashboard de l'admin.

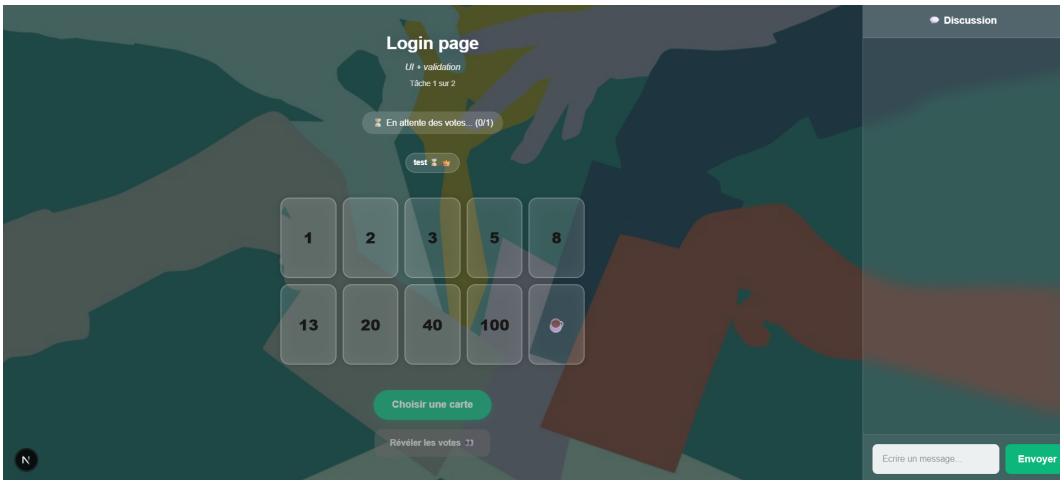

FIGURE 5.7 – Session de vote en temps réel avec affichage de l'état des participants et du chat.

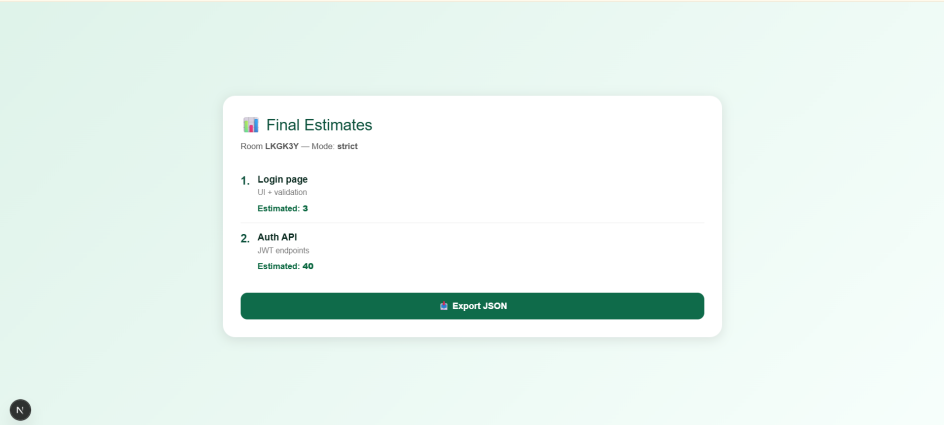

FIGURE 5.8 – Écran final affichant les estimations et export JSON des résultats.

5.3 Gestion du Temps Réel côté Frontend

La communication temps réel est assurée via WebSocket. Lors d'une action utilisateur (vote, message chat, mise à jour du backlog), l'événement est transmis au serveur, puis répliqué auprès de l'ensemble des clients synchronisés.

5.4 Résumé du Chapitre

Ce chapitre a présenté l'implémentation frontend de la plateforme. Grâce à l'utilisation de Next.js, d'une architecture par composants réutilisables, d'un système de communication temps réel et d'une interface intuitive, le frontend offre une expérience fluide adaptée aux contextes collaboratifs modernes.

Chapitre 6

Validation, Tests et Qualité Logicielle

Ce chapitre présente les mécanismes de validation mis en œuvre afin de garantir la fiabilité, la robustesse et la cohérence fonctionnelle de l'application de Planning Poker développée. La démarche adoptée repose sur une combinaison de tests unitaires automatisés, de validation de la communication temps réel, de tests manuels et de l'intégration continue (CI).

L'objectif principal est de détecter précocement les erreurs, de valider la logique métier et de garantir la stabilité du système à chaque évolution du code.

6.1 Stratégie globale de validation

La stratégie de validation du projet repose sur plusieurs niveaux complémentaires :

- **Tests unitaires backend** : validation des règles métier et des fonctionnalités critiques via des tests automatisés Django ;
- **Tests d'intégration** : vérification du bon fonctionnement des endpoints REST et de leur interaction avec la base de données ;
- **Validation de la communication temps réel** : test de l'établissement des connexions WebSocket avec Django Channels ;
- **Intégration continue (CI)** : automatisation des tests et du build à chaque modification du code ;
- **Validation manuelle** : tests fonctionnels portant sur l'ergonomie et le déroulement d'une session de Planning Poker.

Cette approche permet d'assurer une couverture complète du cycle de fonctionnement de l'application tout en restant adaptée au cadre d'un projet académique.

6.2 Tests unitaires et logique métier

Les tests unitaires ont été implémentés à l'aide du framework de test de Django et de l'outil `APIClient` de Django REST Framework. Ils permettent de simuler des requêtes HTTP authentifiées et de valider les réponses de l'API indépendamment de l'interface utilisateur.

Les fonctionnalités testées incluent notamment :

— la création d'une salle de Planning Poker selon différents modes de vote ;

— l'intégration de participants à une session existante ;

— l'importation et la gestion d'un backlog au format JSON ;

— le processus de vote et la révélation des estimations ;

— la gestion des accès non autorisés.

Ces tests assurent la validité de la logique métier, en particulier le respect des règles de vote et la cohérence des données persistées en base.

6.2.1 Synthèse des tests backend

Fonctionnalité testée	**Objectif**	**Statut**
Création de salle	Création d'une session par un utilisateur authentifié	Validé
Rejoindre une salle	Ajout dynamique d'un participant à la session	Validé
Import du backlog	Chargement d'un backlog JSON en base	Validé
Vote et révélation	Enregistrement et calcul des estimations	Validé
Contrôle d'accès	Refus d'accès aux endpoints protégés	Validé

Table 6.1 – Synthèse des tests unitaires backend

6.3 Validation de la communication temps réel

La communication temps réel de l'application repose sur Django Channels et l'utilisation de WebSockets. Un test automatisé a été mis en place afin de vérifier

la capacité d'un client à établir une connexion WebSocket valide avec le serveur ASGI.

Ce test permet de garantir :

— la disponibilité du serveur temps réel ;

— le bon routage des connexions WebSocket ;

— la stabilité de l'infrastructure de communication.

En raison des contraintes spécifiques des environnements d'intégration continue, ces tests WebSocket sont exécutés en environnement local et désactivés dans la pipeline CI afin de garantir la fiabilité de l'exécution automatisée.

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

Afin de garantir la qualité du code et de prévenir toute régression fonctionnelle, une pipeline d'intégration continue a été mise en place à l'aide de **GitHub Actions**. Cette pipeline est déclenchée automatiquement à chaque *push* ou *pull request* sur les branches principales du dépôt.

La CI est structurée autour de deux jobs distincts :

— un job dédié au backend Django ;

— un job dédié au frontend Next.js.

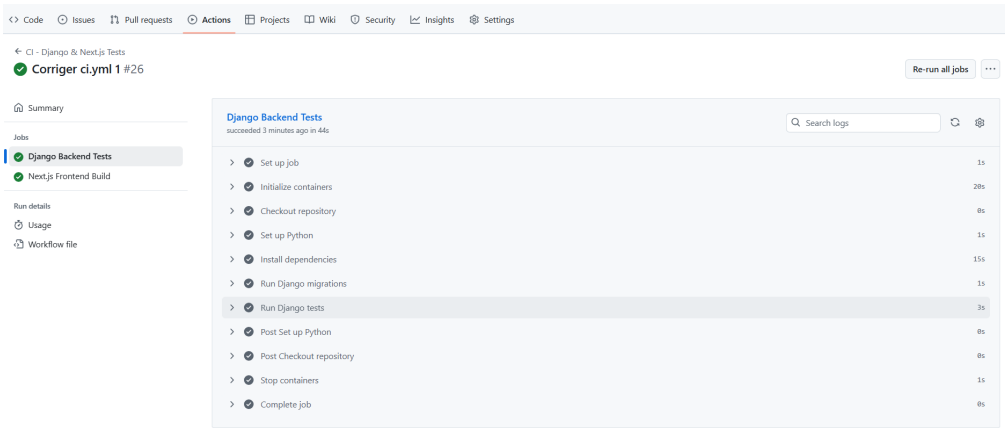

FIGURE 6.1 – Exécution de la pipeline d'intégration continue via GitHub Actions (tests backend et build frontend).

6.4.1 Workflow de la pipeline CI

La pipeline CI suit une succession d'étapes automatisées :

1. récupération du code source depuis le dépôt GitHub ;
2. configuration de l'environnement backend (Python 3.11 et PostgreSQL) ;
3. application des migrations Django ;
4. exécution des tests unitaires backend ;
5. configuration de l'environnement frontend (Node.js 20) ;
6. build de l'application Next.js.

L'échec d'une étape entraîne l'arrêt immédiat de la pipeline, empêchant l'intégration d'un code instable.

6.4.2 Tests automatisés via la CI

Les tests unitaires backend sont exécutés automatiquement lors de chaque exécution de la pipeline CI. Ils permettent de valider la logique métier du Planning Poker, notamment :

— la création et la gestion des salles ;

— la gestion des votes et des règles d'estimation ;

— l'importation des données JSON ;

— la sécurité et le contrôle des accès.

Cette automatisation garantit que toute modification du code respecte les règles fonctionnelles définies.

6.4.3 Génération automatique de la documentation

Dans le cadre de la démarche de qualité logicielle, une génération automatique de la documentation technique du backend a été envisagée. Cette documentation repose sur l'utilisation de commentaires structurés (docstrings) intégrés directement au code Python.

L'outil **Sphinx** permet de générer automatiquement une documentation à partir de ces commentaires, décrivant les modules, classes et méthodes principales du backend.

Cette approche permet :

— de maintenir une documentation cohérente avec le code source ;

— de faciliter la compréhension des modules, classes et méthodes ;

— d'améliorer la maintenabilité du projet.

6.5 Validation manuelle

En complément des tests automatisés, des tests manuels ont été réalisés afin d'évaluer :

— l'ergonomie générale de l'interface utilisateur ;

— la fluidité du déroulement d'une session de Planning Poker ;

— la synchronisation temps réel entre plusieurs utilisateurs.

Ces tests ont permis d'identifier et de corriger des ajustements mineurs liés à l'expérience utilisateur.

6.6 Limites et perspectives

Bien que les mécanismes de validation mis en place couvrent les fonctionnalités essentielles du système, certaines améliorations restent envisageables :

— automatisation complète des scénarios WebSocket ;

— mise en place de tests end-to-end ;

— réalisation de tests de charge pour des sessions à grande échelle ;

— automatisation du déploiement.

6.7 Conclusion

Ce chapitre a présenté l'ensemble des mécanismes de validation, de test et d'intégration continue mis en œuvre dans le projet. La combinaison de tests unitaires, de validation temps réel et de CI garantit une base logicielle robuste, évolutive et conforme aux exigences fonctionnelles du Planning Poker.

Chapitre 7

Manuel Utilisateur

7.1 Installation et Lancement de l'application

Cette section décrit la procédure permettant d'installer et de lancer l'application de Planning Poker développée. Le projet est entièrement **dockerisé**, garantissant un déploiement simple, reproductible et indépendant de l'environnement local de l'évaluateur.

7.1.1 Prérequis

Les seuls prérequis nécessaires à l'exécution du projet sont :

— **Docker**

— **Docker Compose**

Aucune installation supplémentaire (Python, Node.js, PostgreSQL) n'est requise sur la machine hôte.

7.1.2 Architecture Docker

L'application repose sur une architecture Docker Compose composée de deux services distincts :

— **Frontend** : application Next.js exposée sur le port 3000 ;

— **Backend** : application Django avec Django Channels, exécutée via un serveur ASGI (Daphne) et exposée sur le port 8000.

Cette séparation permet une communication claire entre les couches applicatives tout en facilitant le déploiement.

7.1.3 Lancement de l'application

Pour lancer l'application, il suffit d'exécuter les commandes suivantes à la racine du projet :

```
docker compose build
docker compose up
```

Lors du premier lancement, le téléchargement et la construction des images Docker peuvent prendre quelques minutes.

7.1.4 Accès à l'application

Une fois les conteneurs démarrés, l'application est accessible aux adresses suivantes :

— Frontend : `http://localhost:3000`

— Backend : `http://localhost:8000`

7.1.5 Arrêt de l'application

Pour arrêter l'ensemble des services, la commande suivante peut être utilisée :

```
docker compose down
```

7.1.6 Conseils de test multi-utilisateur

Afin de tester correctement le mode multi-utilisateur et les fonctionnalités temps réel, il est recommandé :

— d'utiliser des navigateurs différents (ex. Chrome et Firefox) ou une navigation privée ;

— d'éviter l'utilisation de plusieurs onglets dans un même navigateur, ceux-ci partageant la même session ;

— de tester simultanément un administrateur et plusieurs participants.

Ces précautions permettent d'éviter les conflits de session et de garantir un comportement correct des WebSockets.

7.1.7 Importation du backlog

Avant de démarrer une session de vote, un fichier JSON contenant le product backlog doit être importé dans l'application. Un fichier d'exemple (`product-backlog.json`) est fourni à la racine du projet afin de faciliter les tests et la prise en main.

Conclusion générale

Ce projet a permis de concevoir et de mettre en œuvre une application web de Planning Poker intégrant des fonctionnalités temps réel, fondées sur une architecture moderne combinant Django, Django Channels, Next.js et une base de données PostgreSQL. L'utilisation conjointe d'API REST et de WebSockets a permis de répondre efficacement aux besoins de synchronisation et de collaboration entre les utilisateurs.

Une attention particulière a été portée à la qualité logicielle à travers la mise en place de tests backend, la validation de la communication temps réel et l'intégration continue via GitHub Actions, garantissant la fiabilité et la maintenabilité du système. Les tests utilisateurs ont également permis d'améliorer l'ergonomie et la fluidité de l'expérience proposée.

Enfin, ce travail constitue une base solide et évolutive, pouvant être enrichie par des tests plus avancés, des mécanismes de déploiement automatisés et des optimisations de performance, ouvrant ainsi des perspectives intéressantes pour une utilisation à plus grande échelle.

Bibliographie