



16/12/2024

Rapport de projet

Conception agile de projets
informatique



Afdal BOURAIMA, Yousra BOUHANNA, Elias AIT HASSOU
MASTER 1 – INFORMATIQUE – UNIVERSITE LUMIERE LYON 2

Table des matières

Introduction	2
Contexte et problématique	2
Objectifs	2
Présentation de l'application	3
Fonctionnalités principales	3
Modes de jeu	3
Interface utilisateur	4
Justification des choix techniques	6
Langage de programmation et technologies	6
Choix des langages et Framework	6
Outils	7
Modélisation des données	8
Choix de base de données	8
Structure du fichier JSON	9
Architecture de l'application	10
Modèle architectural	10
Composants principaux	10
Connexions entre les Composants	11
Diagramme de l'architecture	11
Intégration continue	12
Présentation du processus CI/CD	12
Étapes automatisées	12
Outils utilisés	12
Conclusion et perspectives	13
Résumé des accomplissements	13
Perspectives	13
Références	13

Introduction

Contexte et problématique

Les méthodologies agiles ont révolutionné la gestion des projets informatiques, mettant en avant la collaboration et la flexibilité. L'une des étapes cruciales de ces méthodologies consiste à estimer les efforts nécessaires à la réalisation des tâches. Ces estimations permettent de planifier les sprints, de définir les priorités et de coordonner efficacement le travail au sein des équipes.

Cependant, l'estimation des difficultés et des priorités des tâches peut être source de désaccords au sein des équipes. Ces divergences, souvent causées par des interprétations différentes des exigences ou des compétences variées, entraînent parfois des retards et des inefficacités dans la gestion des projets.

Pour répondre à ces défis, des outils comme le **Planning Poker** ont été introduits.

Objectifs

Dans le cadre de ce projet, nous avons entrepris la conception et la réalisation d'une application de Planning Poker, ayant pour objectif :

- De fournir un outil numérique moderne pour faciliter les sessions d'estimation en équipe.
- De proposer une interface conviviale.
- De permettre une gestion efficace des backlogs via l'import et l'export de fichiers JSON.
- D'intégrer une fonctionnalité de chat pour permettre aux membres d'échanger en temps réel pendant les sessions de planification.

En apportant une solution concrète à la problématique posée, cette application vise à renforcer la collaboration et la productivité des équipes agiles.

Présentation de l'application

Fonctionnalités principales

Notre application de Planning Poker permet aux membres d'une équipe de participer à des sessions d'estimation collaborative, respectant les principes des méthodes agiles. Elle propose les fonctionnalités suivantes :

- **Création de parties et partage de code d'accès** : Les utilisateurs peuvent créer une partie, puis partager un code d'invitation, un lien, ou même un QR code pour permettre aux autres membres de rejoindre la session.
- **Saisie de pseudo et choix d'avatar** : Chaque joueur entre son pseudo et choisit un avatar pour personnaliser son expérience dans l'application.
- **Choix du mode de jeu** : L'application propose plusieurs modes de jeu, dont le mode strict (unanimité), moyenne et médiane, pour s'adapter aux préférences de l'équipe et aux besoins du projet.
- **Importation du backlog en format JSON** : Les utilisateurs peuvent importer une liste de tâches (backlog) sous forme de fichier JSON, facilitant la gestion des tâches à estimer.
- **Affichage des cartes de vote et chat intégré** : Lors des sessions d'estimation, les joueurs peuvent choisir leur carte de vote. Un chat intégré permet aux participants de discuter des estimations et d'échanger des opinions en temps réel.
- **Sauvegarde des résultats** : À la fin de chaque session, les résultats sont enregistrés sous forme de fichier JSON. De plus, un mécanisme de carte café permet de sauvegarder l'état de la partie afin de la reprendre ultérieurement.
- **Minuteur** : Un minuteur intégré permet de mieux gérer le temps alloué à chaque estimation, assurant ainsi un déroulement fluide de la session.

Modes de jeu

L'application propose trois modes de jeu pour répondre aux besoins des différentes équipes agiles. Ces modes sont définis comme suit :

- **Mode strict (unanimité)** : Dans ce mode, les joueurs doivent parvenir à une unanimité sur l'estimation de chaque fonctionnalité. Aucun vote n'est validé tant que tous les joueurs n'ont pas sélectionné la même carte. Ce mode favorise la discussion et le consensus au sein de l'équipe.
- **Mode moyenne** : Ce mode commence par un premier vote en mode strict. Si l'unanimité n'est pas atteinte, un second vote est effectué, et la moyenne des estimations des joueurs est calculée pour déterminer l'estimation finale. Ce mode

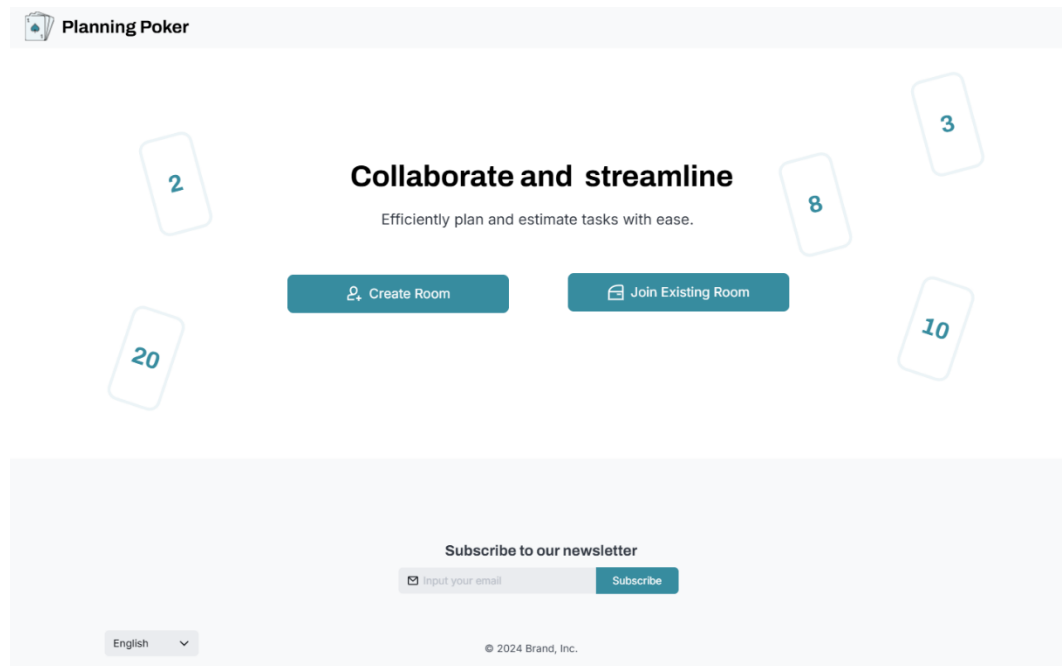
permet d'obtenir une estimation collective en prenant en compte les opinions de tous les membres de l'équipe.

- **Mode médiane** : À l'instar du mode moyenne, ce mode commence par un premier vote en mode strict. Si l'unanimité n'est pas atteinte, l'estimation médiane des votes est retenue comme estimation finale pour chaque tâche. Ce mode privilégie une estimation plus représentative de l'équipe en prenant la valeur centrale parmi les votes.

Interface utilisateur


Pour illustrer l'interface de l'application, voici quelques captures d'écran des pages principales :

- **Page d'accueil** : L'écran d'accueil présentant les options pour créer ou rejoindre une partie.



Conception agile de projets informatique

- **Page de création de room** : Permet à un utilisateur de créer une nouvelle session.

 Planning Poker

2

20

8

3

10

Create Room

Enter the room's name

Choose the game's rule

Dropdown

Import your backlog in JSON format


click on the Import icon

Create

English

© 2024 Brand, Inc.

- **Page pour rejoindre une room** : Permet aux joueurs de rejoindre une session existante en entrant un code d'invitation.

 Planning Poker

2

20

8

3

10

Join Existing Room

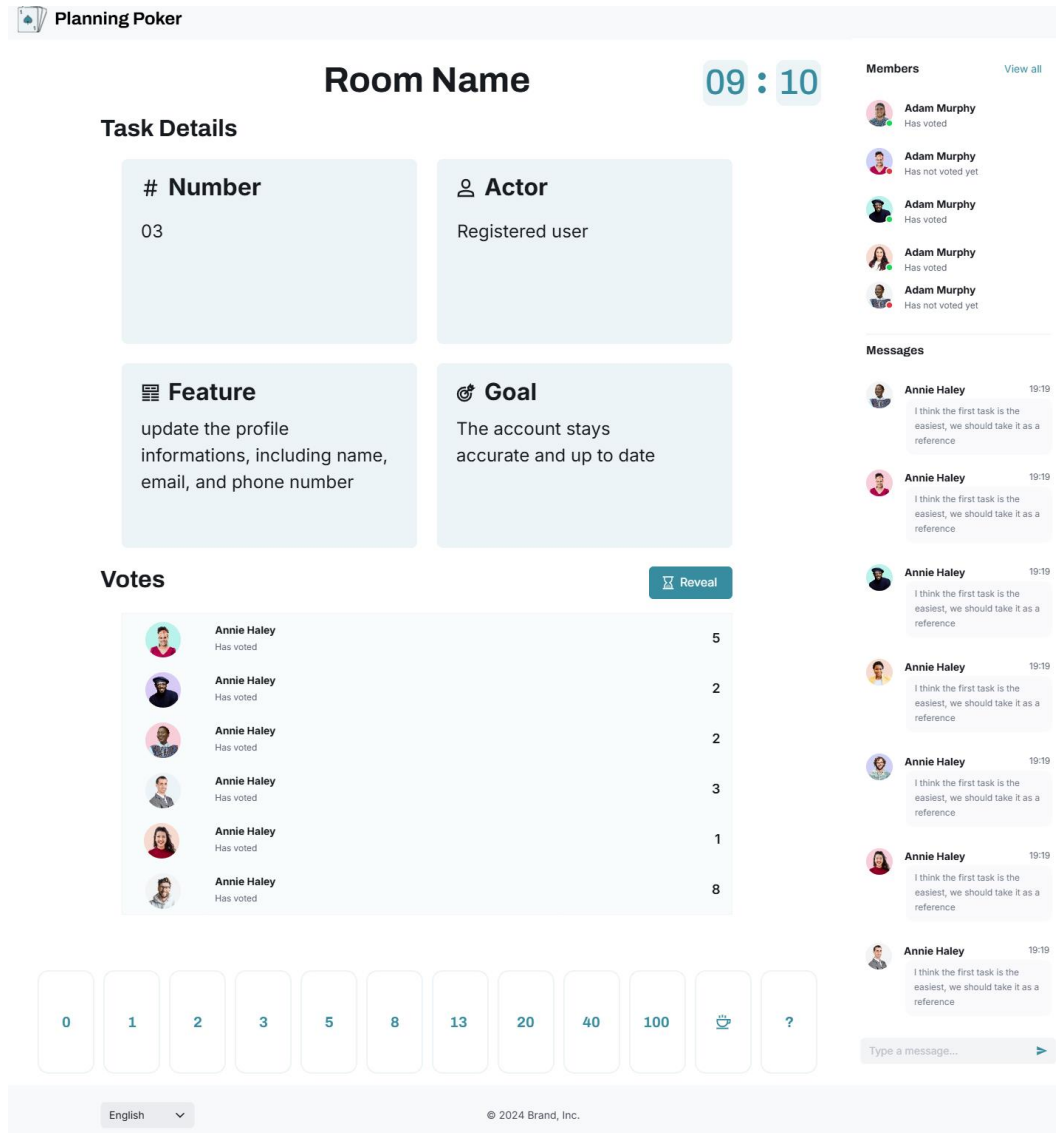
Please enter the code provided to you

Join

English

© 2024 Brand, Inc.

- **Page principale (interface du jeu)** : L'écran où les joueurs peuvent voir le backlog, voter, discuter et suivre les résultats en temps réel.



Justification des choix techniques

Langage de programmation et technologies

Choix des langages et Framework

- **React.js** : Choisi pour la création de l'interface (frontend) en raison de sa popularité, de sa grande communauté et de sa modularité. Permet la création de composants réutilisables et l'utilisation des Hooks. Bibliothèques comme React Router facilitent la gestion de la navigation, rendant le projet maintenable à long terme.
- **Tailwind CSS** : Utilisé pour styliser l'interface de manière flexible et rapide. Cette bibliothèque utilitaire permet de concevoir un design moderne tout en optimisant

le temps de développement grâce à sa facilité d'utilisation et à sa personnalisation.

- **Flask** : Framework Python léger utilisé pour construire les API du backend. Permet une gestion souple des routes et une intégration facile avec des bases de données comme MongoDB dans notre cas, offrant ainsi une solution robuste et évolutive pour l'application.
- **Node.js & NPM** : Node.js utilisé pour l'exécution du JavaScript côté serveur, et NPM pour la gestion des dépendances (React, Tailwind CSS, Lucide, etc.). Facilite l'installation et la mise à jour des bibliothèques nécessaires au projet et assure une gestion efficace des environnements de développement.

Outils

Plusieurs outils ont été utilisés pour le développement, la collaboration et la gestion du projet

- **Visily** : Conception des maquettes d'interface.
- **Microsoft Teams** : Coordination de l'équipe et organisation des réunions régulières dans une approche agile.
- **Git & GitHub** : Collaboration sur le code et gestion des versions.
- **Kanban** : Suivi des tâches et gestion agile du projet.
- **Microsoft Word** : Rédaction du rapport.
- **Visual Studio** : Environnement de développement intégré utilisé pour coder et tester l'application.
- **Postman** : Tests des routes et validation des API pour assurer une communication fluide entre les différents composants de l'application.
- **ESLint** : Pour garantir un code JavaScript conforme aux bonnes pratiques.
- **Prettier** : Pour maintenir un style de code uniforme à travers le projet.
- **Draw.io** : Conception des diagrammes de modélisation.
- **Vercel** : Déploiement continu du frontend sur le web.
- **Render** : Hébergement et mise à jour de l'API backend.
- **GitHub Actions** : Automatisation des processus CI/CD, incluant les tests et la génération de documentation.
- **Sphinx** : Génération de la documentation technique pour le backend en HTML.
- **MongoDB Atlas** : Service cloud pour héberger la base de données MongoDB utilisée dans l'application.
- **Socket.IO** : Gestion en temps réel des interactions entre les utilisateurs.

Modélisation des données

Choix de base de données

Pour notre application **Planning Poker**, nous avons choisi **MongoDB**, une base de données NoSQL orientée document, en raison de sa flexibilité et de sa capacité à gérer des données structurées tout en s'adaptant facilement à des schémas évolutifs. Cette approche est particulièrement adaptée à notre projet, où les entités telles que les tâches, les rounds ou les messages ont des relations hiérarchiques et des champs dynamiques. MongoDB permet également de gérer efficacement les sessions en temps réel, comme le chat et les votes, grâce à sa compatibilité avec des technologies comme **Socket.IO**. De plus, cette base de données assure une rapidité et une efficacité indispensables pour le traitement des données dans un environnement collaboratif.

Structure des Collections

La base de données se compose des collections principales suivantes :

- **Rooms** : Stocke les informations relatives aux sessions, y compris les liens vers les tâches, les rounds et les discussions associées.
- **Tasks** : Contient les éléments du backlog avec leurs estimations et les liens vers les rounds associés.
- **Rounds** : Enregistre les votes, le statut et les résultats des rounds d'estimation.
- **Users** : Gère les informations des utilisateurs, tels que leurs identifiants, pseudos et avatars.
- **Messages** : Archive les messages échangés entre les utilisateurs dans les sessions.
- **Avatars** : Stocke les informations relatives aux avatars des utilisateurs pour personnaliser leur expérience.

Pour mieux visualiser la structure de notre base de données et la manière dont les différentes collections interagissent, nous avons créé le diagramme suivant :

Conception agile de projets informatique



Structure du fichier JSON

Dans notre application, nous utilisons un fichier **backlog** structuré en format JSON pour définir les tâches et fonctionnalités du projet. Afin de répondre à différents besoins, deux structures JSON sont prises en charge :

1. **Première structure JSON** : Elle contient les informations de base pour chaque tâche, à savoir le rôle associé ("en_tant_que"), la fonctionnalité ("fonctionnalite") et l'objectif de la fonctionnalité ("objectif"). Ce format est utilisé lors de la première initiation du jeu pour un backlog donné.

```
{
  "tasks": [
    {
      "en_tant_que": "Utilisateur",
      "fonctionnalite": "Créer un compte",
      "objectif": "Permettre aux utilisateurs de s'inscrire pour accéder à la plateforme"
    },
    {
      "en_tant_que": "Administrateur",
      "fonctionnalite": "Gérer les utilisateurs",
      "objectif": "Ajouter, modifier ou supprimer des utilisateurs afin de maintenir une base de données propre"
    },
    {
      "en_tant_que": "Utilisateur",
      "fonctionnalite": "Créer une session de planning poker",
      "objectif": "Collaborer avec les membres de l'équipe pour estimer les tâches"
    }
  ]
}
```

2. **Deuxième structure JSON** : Cette structure est presque identique à la première, mais elle inclut des informations supplémentaires, telles que les estimations pour chaque tâche ainsi que la règle de jeu choisie lors de la première initiation. Ce

format permet la reprise après une pause, notamment après l'utilisation de la "carte café" pour reprendre les sessions de manière fluide.

```
{
  "game_rule": "mean",
  "tasks": [
    {
      "en_tant_que": "Utilisateur",
      "fonctionnalite": "Créer un compte",
      "objectif": "Permettre aux utilisateurs de s'inscrire pour accéder à la plateforme",
      "estimation": null
    },
    {
      "en_tant_que": "Administrateur",
      "fonctionnalite": "Gérer les utilisateurs",
      "objectif": "Ajouter, modifier ou supprimer des utilisateurs afin de maintenir une base de données propre",
      "estimation": null
    },
    {
      "en_tant_que": "Utilisateur",
      "fonctionnalite": "Créer une session de planning poker",
      "objectif": "Collaborer avec les membres de l'équipe pour estimer les tâches",
      "estimation": null
    }
  ]
}
```

Les deux formats sont acceptés afin de garantir la flexibilité du système en fonction de la situation.

Pour tester l'application, nous fournirons un exemple de fichier backlog, nommé **backlog.json** dans la racine du projet.

Architecture de l'application

Modèle architectural

Pour l'architecture de notre application Planning Poker, nous avons adopté un modèle **client-serveur** en utilisant une architecture **modulaire** et **scalable** pour garantir la performance et l'évolutivité de l'application. Ce modèle permet de séparer la logique métier (côté serveur) et l'interface utilisateur (côté client), ce qui simplifie la gestion des différentes fonctionnalités et la mise à jour de l'application sans perturber l'expérience utilisateur.

Composants principaux

- **Client (Frontend) :**
 - Développé avec **React** pour une interface dynamique et **Tailwind CSS** pour la mise en page.
 - Il permet à l'utilisateur d'interagir avec l'application, de voter, de discuter via le chat intégré et de se connecter aux sessions de jeu.
- **Serveur (Backend) :**

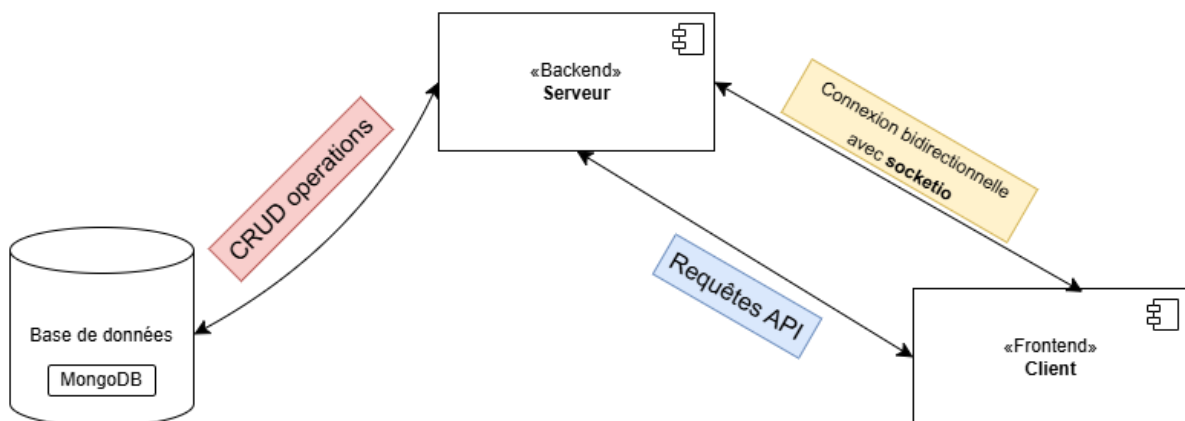
- Utilise **Flask**, un framework Python léger, pour gérer les routes, les API et la logique métier.
- Il interagit avec la base de données **MongoDB** pour stocker et récupérer les données relatives aux sessions, utilisateurs, messages et tâches.
- **Base de données (MongoDB) :**
 - **MongoDB** est utilisé pour stocker les données sous forme de documents JSON. Elle gère plusieurs collections comme Rooms, Tasks, Rounds, Users, Messages, et Avatars, offrant ainsi une grande flexibilité et évolutivité.

Connexions entre les Composants

- **API REST (Backend - Frontend) :** Le frontend communique avec le backend via des API REST, permettant l'échange de données telles que les votes, les messages, et les informations de session. Ces API utilisent des requêtes HTTP classiques pour effectuer des opérations CRUD sur les données de la base MongoDB.
- **Socket.IO (Communication en Temps Réel) :** Pour les fonctionnalités en temps réel, comme la mise à jour des votes ou l'échange de messages instantanés, Socket.IO est utilisé pour établir une connexion bidirectionnelle entre le frontend et le backend. Cette technologie permet aux utilisateurs de voir les changements en temps réel, garantissant une expérience fluide et interactive.

Diagramme de l'architecture

Pour mieux visualiser les connexions entre les différents composants de notre application, nous avons créé un diagramme d'architecture :



Intégration continue

Présentation du processus CI/CD

Le processus CI/CD de l'application Planning Poker garantit la qualité, la fiabilité et la rapidité des déploiements en automatisant les étapes d'intégration et de déploiement.

- **Intégration continue (CI)** : Gérée via **GitHub Actions**, elle permet de valider les changements de code grâce à des tests automatisés et à la génération de documentation.
- **Déploiement continu (CD)** : Géré via **Vercel** et **Render**, connectés directement au dépôt GitHub, pour déployer automatiquement les dernières versions de l'interface utilisateur et de l'API backend.

Chaque modification sur la branche principale déclenche une série de vérifications et met en production une version stable de l'application.

Étapes automatisées

1. **Clonage du dépôt** : Récupération du code source depuis GitHub après chaque modification ou pull request.
2. **Installation des dépendances** :
 - a. **Frontend** : Installation via npm.
 - b. **Backend** : Installation via pip dans un environnement virtuel Python.
3. **Exécution des tests** :
 - a. **Frontend** : Tests réalisés avec **React Testing Library**.
 - b. **Backend** : Utilisation de **Pytest** pour valider les fonctionnalités et assurer la stabilité du code.
4. **Génération de la documentation** :
 - a. **Frontend** : Scripts npm pour créer des fichiers explicatifs.
 - b. **Backend** : Utilisation de **Sphinx** pour produire une documentation technique en HTML.
5. **Déploiement continu** :
 - a. **Frontend** : Déploiement automatique sur **Vercel**.
 - b. **Backend** : Hébergement et mise à jour de l'API sur **Render**.

Outils utilisés

- **GitHub Actions** : Pour orchestrer les pipelines CI/CD.
- **Vercel** : Déploiement automatique du frontend.
- **Render** : Hébergement et mise à jour continue de l'API backend.
- **React Testing Library** : Tests unitaires pour le frontend.
- **Pytest** : Validation des fonctionnalités backend.

- **Sphinx** : Génération de la documentation technique backend

Conclusion et perspectives

Résumé des accomplissements

Le projet **Planning Poker** a permis de créer une application collaborative complète, avec une interface conviviale en **React** et un backend fiable en **Flask**. La base de données **MongoDB** et l'intégration de **Socket.IO** assurent une synchronisation fluide des fonctionnalités. Grâce à un pipeline **CI/CD** automatisé, les tests, la documentation et le déploiement continu sur **Vercel** et **Render** sont entièrement gérés, offrant une solution stable et efficace pour les utilisateurs.

Ce projet nous a aussi permis d'acquérir de nombreuses nouvelles compétences, en particulier en ce qui concerne les processus **CI/CD**, le déploiement continu, et la gestion des environnements de développement. Travailler sur cette application a également renforcé notre expérience de collaboration, de gestion de version avec **GitHub**, et de travail en équipe. Ces défis ont non seulement approfondi notre compréhension des bonnes pratiques en développement logiciel, mais aussi mis en lumière l'importance de l'automatisation dans le cycle de vie d'une application. En somme, cette expérience a été une véritable opportunité d'apprentissage et de développement de nos compétences techniques et collaboratives.

Perspectives

Le projet Planning Poker a atteint ses objectifs, mais des améliorations et ajouts peuvent être envisagés pour l'avenir. Parmi les perspectives, nous pouvons citer :

- L'ajout d'une fonctionnalité d'inscription à une newsletter mensuelle pour permettre aux utilisateurs de recevoir des conseils sur les méthodes agiles et la gestion de projets.
- L'implémentation d'un bouton permettant de traduire l'ensemble du site en français, rendant l'application plus accessible à un public plus large.
- L'intégration de l'application avec des outils de collaboration populaires tels que Microsoft Teams, afin d'enrichir la collaboration en temps réel et d'améliorer la gestion des projets à travers des plateformes déjà utilisées par les équipes

Références

- Documentation officielle de React : <https://reactjs.org>
- Documentation officielle de Flask : <https://flask.palletsprojects.com>

Conception agile de projets informatique

- Documentation officielle de Socket.IO : <https://socket.io/docs/>
- Documentation officielle de MongoDB : <https://www.mongodb.com/docs/>
- Documentation officielle de GitHub Actions : <https://docs.github.com/en/actions>
- Documentation officielle de Vercel : <https://vercel.com/docs>
- Documentation officielle de Render : <https://render.com/docs>
- Documentation officielle de Sphinx : <https://www.sphinx-doc.org>
- ChatGPT