Conception Agile De Projets Informatiques

Projet CAPI

Table des matières

I. Introduction

A. Contexte et Objectifs du projets

Ce projet, réalisé en binôme selon les principes de pair programming, a pour but de développer une application de planning poker. Cet outil, utilisé dans les méthodologies agiles, permet aux membres d'une équipe d'estimer la complexité des tâches d'un Backlog, de voter sur leur difficulté et de sauvegarder les résultats pour une analyse ou une reprise ultérieure. Le développement intègre des pratiques modernes telles que l'intégration continue et la production d'une documentation technique complète.

B. Présentation du projet (backend et frontend)

Le projet est structuré en deux parties principales :

1. Backend

a) Conception et développement des API permettant la gestion des fonctionnalités du planning poker, incluant la validation des votes, la gestion des règles, et l'enregistrement des résultats.

b) Documentation détaillée des endpoints et des modèles de données utilisés.

2. Frontend

c) Interface utilisateur permettant aux joueurs d'interagir avec l'application, de configurer les parties et de visualiser les résultats.

d) Intégration des API backend pour gérer les interactions entre les utilisateurs et le système.

QUELQUES PHOTOS DU PROJET

II. Justification des choix techniques

A. Architecture globale du projet

L'application adopte une architecture client-serveur classique, organisée en trois couches principales qui collaborent pour assurer son fonctionnement :

- Le **frontend**, qui gère l'interface utilisateur et les interactions avec les utilisateurs
- Le **backend**, qui traite la logique métier et les requêtes venant du frontend
- La **base de données**, qui stocke et gère les informations nécessaires au bon déroulement des fonctionnalités de l'application.

B. Choix Technologiques

Pour développer notre application, nous avons choisi des technologies adaptées à nos besoins en termes de robustesse, performance, et maintenabilité :

1. Base de données : PostgreSQL

- Un système de gestion de base de données relationnel robuste et fiable, idéal pour gérer les relations complexes entre les entités de l'application.

2. Backend : Java avec Spring Boot

- Un framework puissant et mature pour construire des APIs RESTful performantes, tout en offrant des outils pour gérer les dépendances, la sécurité, et l'interaction avec la base de données via Spring Data JPA.

3. Frontend : Angular

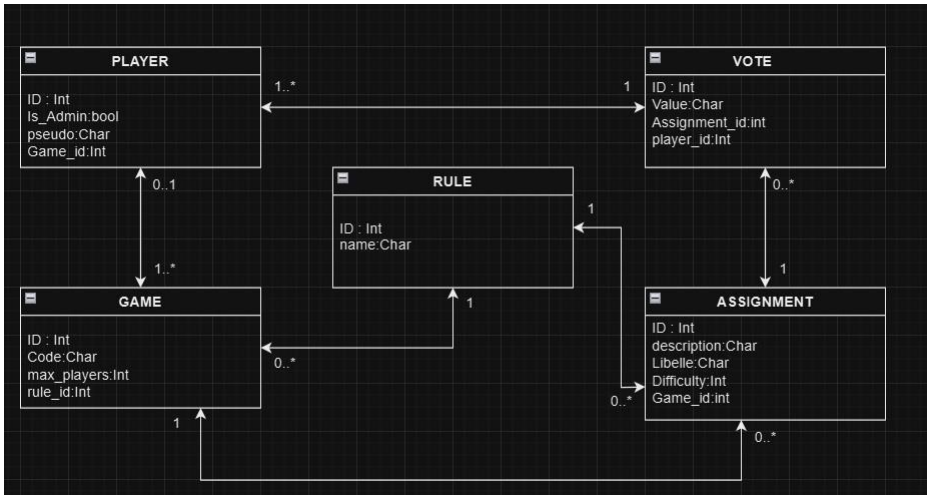
- Un framework frontend moderne basé sur TypeScript, permettant de créer une interface utilisateur interactive et modulaire, tout en simplifiant la consommation des API REST du backend.

III. Conception des classes et des composants

1. Diagramme de classes

Le diagramme de classes présenté ci-dessous illustre la structure de l'application en mettant en évidence les principales entités, leurs attributs, leurs méthodes, et les relations entre elles. Chaque classe représente une abstraction d'un élément clé du système de jeu "Planning Poker".

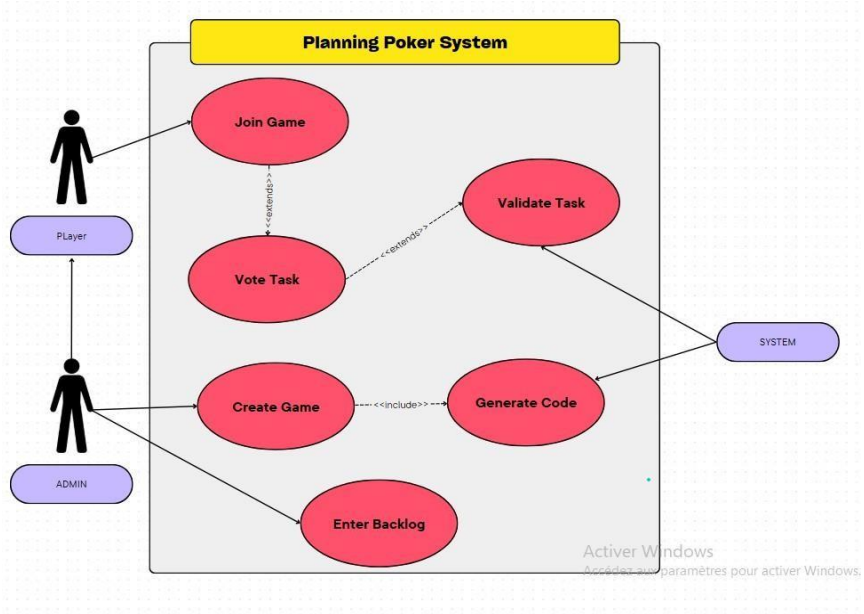
- **Player** : Représente les joueurs, associés à un jeu et effectuant des votes.
- **Game** : Définit un jeu avec ses paramètres (code, règles, nombre de joueurs).
- **Rule** : Contient les règles appliquées aux votes.
- **Assignment** : Regroupe les tâches à voter dans un jeu.
- **Vote** : Associe un joueur à un vote pour une tâche donnée.
-



2. Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation montre les interactions entre les acteurs (`Player`, `Admin`) et le système "Planning Poker". Les fonctionnalités principales incluent la création de jeu, la saisie d'un backlog, le vote pour une tâche, et la validation d'une tâche.

- **Admin** :
  - Crée un jeu et génère un code.
  - Ajoute le backlog (ensemble des tâches) pour un jeu.
- **Player** :
  - Rejoint un jeu en entrant un code valide. ○ Participe au vote des tâches dans un jeu en respectant les règles définies.



3. Composants front-end

L'interface utilisateur est développée en Angular et se compose de deux composants principaux :

1. **CreateGameComponent** :
   - Fonctionnalité : Permet de créer un jeu ou de rejoindre un jeu existant via une interface déroulante.
   - Actions possibles :
     - Création d'un jeu avec définition des paramètres (nombre de joueurs, règle de jeu).
     - Entrée d'un code pour rejoindre un jeu existant.
2. **PlanningGameComponent** : ○ Fonctionnalité : Affiche l'interface du jeu où les joueurs peuvent :
   - Visualiser les tâches (backlog).
   - Voter pour les tâches.
   - Suivre l'avancement en temps réel.

Ces composants consomment les API du backend pour interagir avec la base de données et gérer les différentes entités (jeux, tâches, votes).

IV. Documentation technique

1. Documentation du backend

A. Outils Utilisés (Doxygen)

Doxygen est un outil puissant pour générer automatiquement la documentation de projets Java. Voici les points essentiels :

Fonctionnalités principales :

- **Documentation automatique :** Scanne le code source et génère des documents HTML ou PDF.
- **Navigation facile :** Liens entre classes, méthodes, et packages.
- **Diagrammes UML :** Visualisation des relations entre classes (nécessite Graphviz).
- **Code source en couleur :** Inclut le code avec mise en évidence syntaxique.
- **Personnalisation :** Configurable avec un fichier `Doxyfile`.

2. DTOs

Requêtes (Request DTOs)

1. **`AddAssignmentRequest` :** Représente une requête pour ajouter ou modifier une tâche (assignment) dans le jeu.
2. **`AddRuleRequest` :** Représente une requête pour ajouter ou modifier une règle dans le système de Planning Poker.
3. **`CreateGameRequest`** : Représente une requête pour créer ou modifier un jeu dans le système de Planning Poker.
4. **`CreatePlayerRequest` :** Représente une requête pour créer ou modifier un joueur dans le système de Planning Poker.
5. **`JoinGameRequest` :** Représente une requête pour qu'un joueur rejoigne une partie spécifique dans le système de Planning Poker.
6. **`VoteRequest` :** Représente une requête pour soumettre un vote dans le système de Planning Poker.

Réponses (Response DTOs)

1. **`AssignmentResponse`** : Représente la réponse liée à une tâche (assignment) dans le système. 2. **`GameResponse`** : Représente la réponse détaillée d'un jeu.
3. **`PlayerResponse`** : Représente la réponse détaillée d'un joueur dans le jeu.
4. **`RuleResponse`** : Représente la réponse détaillée d'une règle dans le jeu.
5. **`VoteResponse`** : Représente la réponse détaillée d'un vote effectué par un joueur sur une tâche.

3. Modèles

1. **`Assignment`** : Représente une tâche ou un élément de backlog dans un jeu de Planning Poker.
2. **`Game`** : Représente une partie de Planning Poker, un jeu où les joueurs estiment la difficulté des tâches.

3. **`Player`** : Représente un joueur dans une partie de Planning Poker.
4. **`Rule`** : Représente une règle associée à une partie de Planning Poker.
5. **`Vote`** : Représente un vote effectué par un joueur pour une tâche (assignment) dans une partie de Planning Poker.

4. Contrôleurs

1. **`AssignmentController`** : Gestion des requêtes HTTP liées aux tâches (assignments).
2. **`GameController`** : Gestion des requêtes HTTP liées aux jeux (games).
3. **`PlayerController`** : Gestion des requêtes HTTP liées aux joueurs (players).
4. **`RuleController`** : Gestion des requêtes HTTP liées aux règles (rules).
5. **`VoteController`** : Gestion des requêtes HTTP liées aux votes.

5. Services

Interfaces

1. **`IAssignmentService`** : Interface pour la gestion des tâches dans le système.
2. **`IGameService`** : Interface pour la gestion des jeux dans le système.
3. **`IPlayerService`** : Interface pour la gestion des joueurs.
4. **`IRuleService`** : Interface pour la gestion des règles.
5. **`IVoteService`** : Interface pour la gestion des votes.

Implémentations

1. **`AssignmentService`** : Implémentation du service responsable de la gestion des tâches.
2. **`GameService`** : Implémentation du service responsable de la gestion des jeux.
3. **`PlayerService`** : Implémentation du service responsable de la gestion des joueurs.
4. **`RuleService`** : Implémentation du service responsable de la gestion des règles.
5. **`VoteService`** : Implémentation du service responsable de la gestion des votes.

6. Mappers

1. **`AssignmentMapper`** : Mapper pour la conversion entre les entités `Assignment`, les requêtes DTO (`AddAssignmentRequest`), et les réponses DTO (`AssignmentResponse`).
2. **`GameMapper`** : Mapper pour la conversion entre les entités `Game`, les requêtes DTO (`CreateGameRequest`), et les réponses DTO (`GameResponse`).
3. **`PlayerMapper`** : Mapper pour la conversion entre les entités `Player`, les requêtes DTO (`CreatePlayerRequest`), et les réponses DTO (`PlayerResponse`).
4. **`RuleMapper`** : Mapper pour la conversion entre les entités `Rule`, les requêtes DTO (`AddRuleRequest`), et les réponses DTO (`RuleResponse`).
5. **`VoteMapper`** : Mapper pour la conversion entre les entités `Vote`, les requêtes DTO (`VoteRequest`), et les réponses DTO (`VoteResponse`).

7. Repositories

1. **`AssignmentRepository`** : Interface pour accéder aux données relatives aux tâches (`Assignment`).
2. **`GameRepository`** : Interface pour accéder aux données relatives aux jeux (`Game`).
3. **`PlayerRepository`** : Interface pour accéder aux données relatives aux joueurs (`Player`).
4. **`RuleRepository`** : Interface pour accéder aux données relatives aux règles (`Rule`).
5. **`VoteRepository`** : Interface pour accéder aux données relatives aux votes (`Vote`).

8. Point d'Entrée

1. **`PlanningPokerApplication`** : Classe principale permettant de démarrer l'application Spring Boot.

V. Documentation du frontend:

A. Structure du projet

Le projet frontend est structuré suivant une architecture modulaire Angular, favorisant une séparation claire des responsabilités. Cette organisation optimise la maintenance, la réutilisation des composants et l'intégration fluide avec le backend grâce à des services dédiés.

B. Composants principaux et leur rôle

Create-Game : Ce composant permet aux utilisateurs de créer ou rejoindre une nouvelle partie de Planning Poker. Il inclut des formulaires pour définir les paramètres du jeu, tels que le nom, les participants, et les règles associées.

Planning-Game : Ce composant gère le processus interactif de Planning Poker, permettant aux joueurs de voter sur des tâches et de collaborer en temps réel. Il consomme les API backend pour récupérer et mettre à jour les données du jeu.

C. Consommation des API backend

La communication du Frontend et du Backend s'effectue grâce à un service **game.service.ts** qui utilise les modules HttpClient en encapsulant les requêtes rendant les interactions réutilisables et maintenables. Ce service est utilisé par les composants principaux comme create-game et planning-game, assurant une interaction fluide avec le backend.

VI. Mise en place de l'intégration continue

A. Définition des objectifs de l'intégration continue

L'objectif principal de l'intégration continue (CI) est d'automatiser le processus de test, de build et de déploiement pour garantir que le code développé par différents contributeurs soit régulièrement intégré dans le dépôt central. Cela permet d'assurer une détection rapide des erreurs et des problèmes de qualité de code, d'améliorer la productivité des équipes de développement, et de livrer des fonctionnalités de manière plus fiable et rapide.

Les objectifs spécifiques de la mise en place de l'intégration continue pour ce projet backend sont :

- **Automatisation des tests unitaires :** Exécuter automatiquement les tests à chaque modification du code pour détecter immédiatement les erreurs de logique.
- **Garantir la qualité du code :** Assurer que les tests réussissent à chaque push sur une branche spécifique (par exemple backend-*) et que les nouveaux changements ne brisent pas les fonctionnalités existantes.
- **Améliorer la collaboration et la visibilité :** Chaque membre de l'équipe peut voir immédiatement si ses modifications sont compatibles avec le code existant, évitant ainsi les conflits d'intégration.
- **Faciliter le déploiement :** Automatiser le processus de déploiement des builds sur un environnement de pré-production ou de production une fois que les tests sont validés.

B. Mise en place pour le backend

1. Stratégie de test adoptée

La stratégie de test adoptée pour ce projet repose sur les tests unitaires afin de garantir que les fonctionnalités du backend fonctionnent comme prévu dans des scénarios isolés.

Les tests sont exécutés à chaque push sur une branche commençant par backend- et à chaque pull request vers la branche main. Cette approche permet de valider chaque modification du code et de s'assurer qu'aucune nouvelle fonctionnalité n'introduit de régressions dans les fonctionnalités existantes.

Les tests sont écrits en utilisant JUnit 5, un framework populaire pour écrire et exécuter des tests unitaires en Java. Cette stratégie permet de couvrir plusieurs aspects du backend, notamment :

Les services métier : Vérification de la logique des services, par exemple, la création de joueurs, l'édition de règles de jeu, la gestion des votes, etc.

Les contrôleurs et les APIs : Tests des points d'entrée du backend (comme les requêtes HTTP) pour s'assurer qu'ils répondent correctement aux demandes et aux erreurs.

Les interactions avec la base de données : Validation des opérations CRUD (Create, Read, Update, Delete) dans les repositories, en simulant l'accès aux données sans se connecter à une base de données réelle (via l'utilisation de H2 ou Mockito).

B. Frameworks utilisés

Pour la mise en place de l'intégration continue et l'exécution des tests unitaires, plusieurs frameworks et outils sont utilisés :

JUnit 5 : Framework de tests unitaires pour écrire, organiser et exécuter les tests. Il est utilisé pour tester les services, les contrôleurs et les composants du backend.

Exemple : Tests de services comme createGameRequest, addRuleRequest, ou encore submitVote.

Mockito : Framework de mock pour simuler les comportements des dépendances externes (par exemple, les repositories ou autres services) lors des tests unitaires. Cela permet de tester des composants de manière isolée.

Spring Boot Test : Le framework Spring Boot Test permet d'intégrer et de tester les applications Spring avec des configurations minimales, ce qui est utile pour les tests d'intégration des API REST.

Maven : Outil de gestion de projet pour exécuter les tests, réaliser des builds et gérer les dépendances du projet.

GitHub Actions : Service d'intégration continue pour automatiser le processus de test et de build chaque fois qu'une modification est poussée vers une branche backend-* ou une pull request vers main.

C. Exemples de cas de test et résultats

Les cas de test sont créés pour vérifier que les différentes fonctionnalités du backend fonctionnent correctement dans des conditions spécifiques. Voici quelques exemples de tests que nous avons écrits pour les services de notre application de Planning Poker :

Test pour la création d'un joueur (CreatePlayerRequest)

But du test : Vérifier si un joueur peut être créé avec succès en utilisant une requête de création de joueur.

Méthode testée : createEditPlayer dans IPlayerService.

Résultat attendu : Le joueur est créé avec un pseudonyme, un rôle (admin ou non), et un jeu associé.

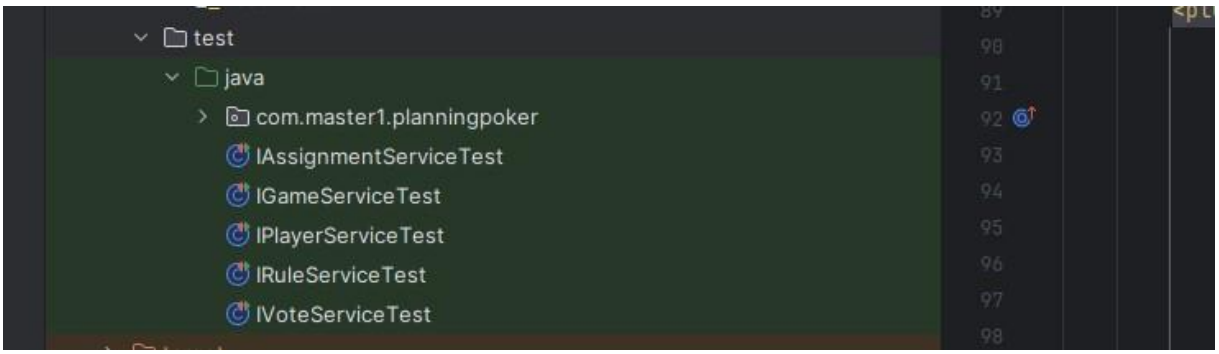


Figure 1: Capture du dossier contenant les test

```java
@Test  AGBLA kokou Jacques
void testAddEditAssignment() {
    // Arrange
    AddAssignmentRequest request = new AddAssignmentRequest();
    request.setLibelle("New Task");
    request.setDescription("Task Description");
    request.setDifficulty(3);

    AssignmentResponse expectedResponse = new AssignmentResponse();
    expectedResponse.setLibelle("New Task");
    expectedResponse.setDescription("Task Description");
    expectedResponse.setDifficulty(3);

    when(assignmentService.addEditAssignment(request)).thenReturn(expectedResponse);

    // Act
    AssignmentResponse response = assignmentService.addEditAssignment(request);

    // Assert
    assertNotNull(response);
    assertEquals( expected: "New Task", response.getLibelle());
    assertEquals( expected: "Task Description", response.getDescription());
    verify(assignmentService, times( wantedNumberOfInvocations: 1)).addEditAssignment(request);
}
```

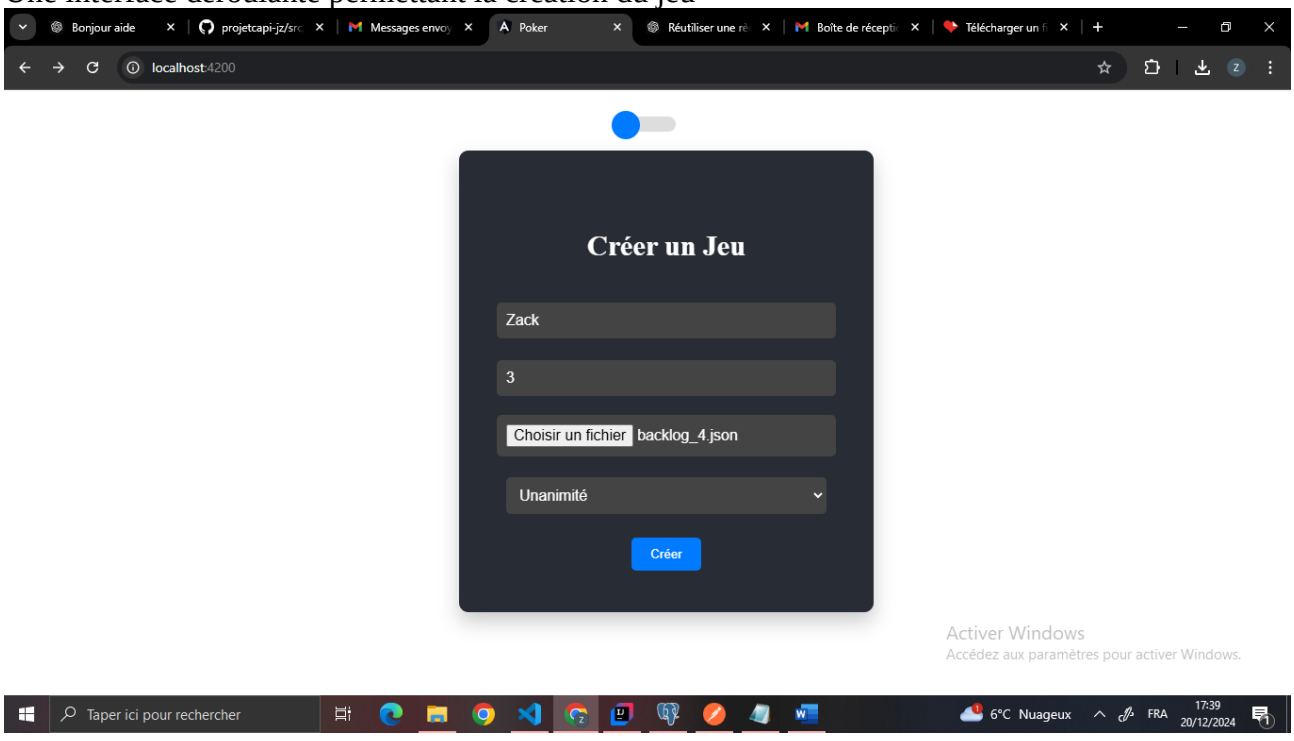
Figure 2: Exemple de test pour ajouter une tâche

2. Mise en place pour le Frontend

A. Tests d'interface utilisateur et validation de la consommation des APIs backend

1-Creation de jeu

Une interface déroulante permettant la création du jeu

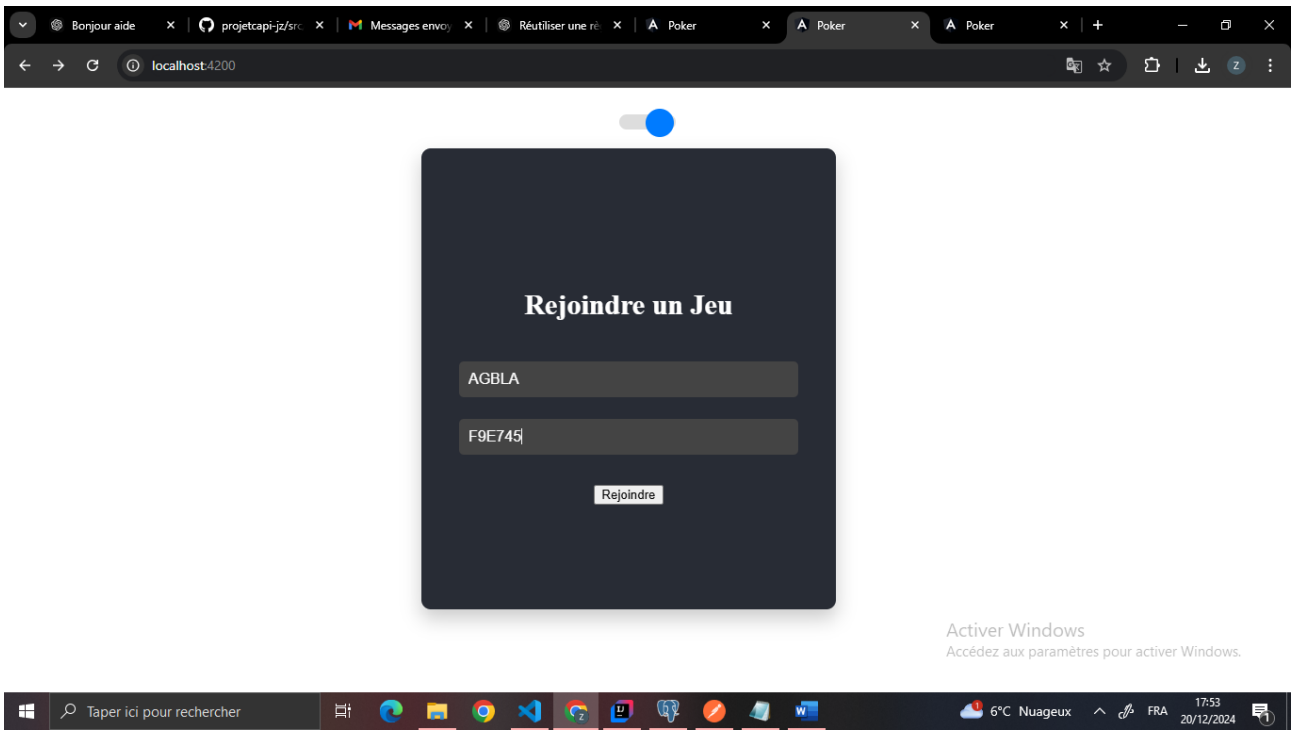


2-Le planning poker

L’interface qui permet aux joueurs de voter

3-Rejoindre le jeu

L’interface permettant aux joueurs de rejoindre le jeux



4-Le vote

Une fois le vote effectuée on demande au joueur d'attendre le votes autres joueurs

5-Regle Respectée

Une fois les joueurs ont voté et la règle respecte l'application affiche le message suivant

6-Tache suivante

Une fois la règle respectée on passe a la tache suivantes

VII. Conclusion et perspectives

A. Évaluation des résultats obtenus

Le projet a permis d'atteindre les principaux objectifs définis au départ, à savoir la mise en place d'une application robuste pour le jeu de Planning Poker avec un backend fonctionnel et une interface frontend intuitive. La gestion des joueurs, des règles et des parties a été mise en œuvre avec succès, tout en assurant la validation des données et l'interaction fluide entre les différentes couches du système.

Les résultats obtenus montrent une application stable avec une bonne couverture fonctionnelle. Les tests unitaires ont permis d'assurer une qualité de code satisfaisante et la plupart des fonctionnalités demandées ont été implémentées selon les spécifications. Le travail sur les API, l'intégration des données a été bien maîtrisé. L'interface utilisateur est en cours de développement et présente un design clair et moderne pour une expérience utilisateur optimale.

b) Limites rencontrées

Malgré les progrès réalisés, plusieurs limites ont été rencontrées au cours du projet, notamment en ce qui concerne la documentation frontend.Les API de communication entre le backend et le frontend nécessitaient des ajustements, notamment pour la gestion des erreurs et des retours

d'information. Ces ajustements ont retardé le développement de certaines fonctionnalités, mais ont permis d’améliorer la résilience du système.

Une autre limitation notable est la gestion de certaines interactions complexes dans le frontend, comme la gestion en temps réel des joueurs et des cartes. Elles auraient pu être developpé davantage si le temps de développement avait été plus important et si une meilleure planification avait été effectuée sur les besoins de documentation et d'interfaces côté utilisateur.

c) Améliorations et extensions futures

Pour l'avenir, plusieurs pistes d'amélioration et d'extensions sont envisagées pour rendre l'application plus performante et mieux adaptée aux besoins des utilisateurs.

Une extension serait de déployer l'application en ligne. Cela permettrait de tester l'application dans un environnement réel, avec des utilisateurs réels, et de récolter des retours sur son ergonomie et ses fonctionnalités. Le déploiement pourrait se faire sur des plateformes cloud comme AWS, Heroku ou Netlify, en fonction des besoins en scalabilité et des choix techniques.

Concernant la gestion de la CI/CD via GitHub Actions, bien qu'elle ait facilité le processus d'intégration continue, des difficultés ont été rencontrées dans la configuration de certaines étapes d'automatisation, notamment pour l'intégration des tests, la mise à jour automatique des dépendances et le processus de déploiement en ligne via GitHub Actions n'ont pas toujours été aussi fluides qu'anticipés. Plusieurs erreurs de configuration liées à la gestion des environnements de production.