Rapport de Projet – Application Planning Poker

Conception Agile de Projets Informatiques

Binôme (Trio) : JOULIE Benjamin, YAKOUBENE Gibril, RICHAND Marius

1. Introduction et Contexte

L'objectif de ce projet était de concevoir et de développer une application de Planning Poker, un outil utilisé dans les méthodes agiles (Scrum) pour estimer l'effort des tâches d'un backlog de manière collective. Notre application permet à plusieurs joueurs connectés simultanément de voter sur l'estimation d'une tâche, en s'appuyant sur différentes règles de consensus telles que l'unanimité, la moyenne, la médiane ou la majorité.

L'objectif était de se rapprocher au maximum du fonctionnement d'une session réelle de Planning Poker, tout en proposant une solution simple à installer et à utiliser. L'application a donc été pensée pour être légère et facilement déployable.

Le projet a été réalisé en trio (JOULIE Benjamin, YAKOUBENE Gibril, RICHAND Marius). L'organisation du travail reposait sur une approche de pair programming adaptée à trois personnes, avec une collaboration continue entre les membres de l'équipe. Les rôles (backend, frontend, tests, documentation et DevOps) ont été alternés afin que chacun puisse travailler sur l'ensemble de la stack. Le suivi du projet a été assuré à l'aide d'un dépôt Git et de la gestion d'issues et de tâches.

2. Choix Techniques et Architecture

2.1 Langage et technologies utilisées

Nous avons opté pour une stack Full JavaScript (Node.js) pour plusieurs raisons stratégiques :

-**Node.js (Backend)** : Choix technique majeur pour sa gestion native des entrées/sorties asynchrones et surtout pour son écosystème WebSocket (via `socket.io`). Le Planning Poker nécessitant une synchronisation temps réel (affichage des votes des autres joueurs instantanément), Node.js est bien plus adapté qu'un serveur PHP ou Python classique (souvent basés sur du requête/réponse).

-**JavaScript (Frontend)** : Langage maîtrisé par l'équipe, permettant une cohérence totale Client/Serveur (partage de logique possible).

-**HTML/CSS (Vanilla)** : Nous n'avons pas utilisé de framework lourd (React/Angular) pour garder l'application légère, performante et sans étape de "build" complexe, facilitant ainsi l'installation par l'évaluateur.

2.2 Architecture logicielle

Notre application repose sur une architecture événementielle, on s'est inspiré du modèle MVC :

-Vue (Client) : `index.html` et `style.css` gèrent l'affichage. `client.js` écoute les événements utilisateur (clics) et les envoie au serveur.

-**Contrôleur (Serveur)** : `server.js` reçoit les événements WebSocket (`submit_vote`, `next_question`, `join_game`), met à jour l'état de la partie, et renvoie le nouvel état à tous les clients.

-**Modèle (État)** : L'état du jeu est stocké en mémoire côté serveur sous forme d'objets JavaScript structurés (Parties, Joueurs, Tâches).

2.3 Modélisation (diagrammes)

Voici la structure logique de nos données (Diagramme de classe simplifié) :

classDiagram

```
class Partie {
    String code
    String modeVote
    Int round
    Boolean enPause
}
class Joueur {
    String id
    String nom
    String vote
    Boolean estMaitre
}
class Tache {
    String description
    String priorite
}
class ChatMessage {
```

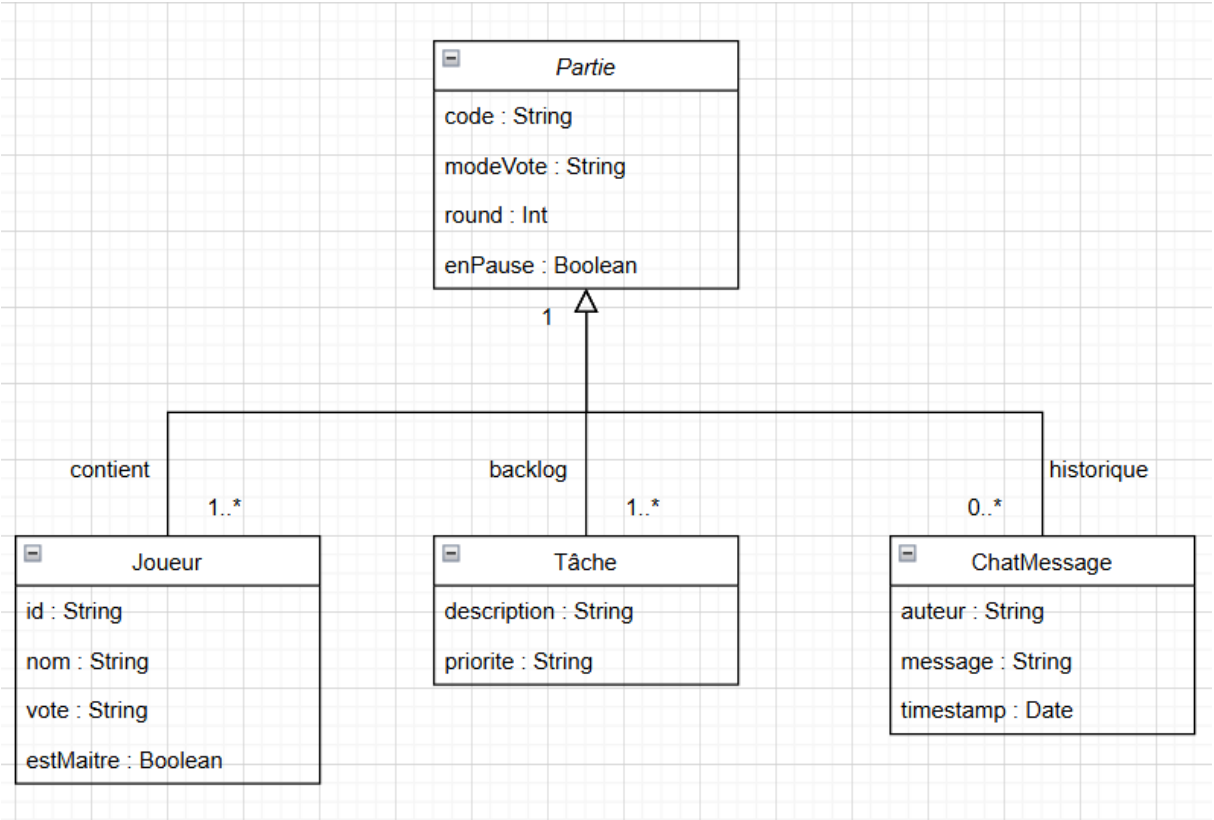
String auteur

String message

Date timestamp

}

Le système est organisé autour d'une partie.
Une partie regroupe plusieurs joueurs, possède un backlog constitué de plusieurs tâches et conserve un historique des messages échangés par les joueurs.



2.4 Gestion des données

Les données sont stockées de la manière suivante :

- Backlog : Importé via JSON. Structure simple : [{ "nom": "Tâche 1", "description": "..." }, ...].

-Sauvegarde : En cas de pause ("Café"), l'état complet est exporté en JSON pour permettre une reprise ultérieure sans perte de données.

3. Mise en place de l'Intégration Continue

3.1 Workflow CI

Nous avons configuré GitHub Actions pour automatiser la qualité du projet. Le fichier .github/workflows/documentation.yml définit le pipeline suivant :

1. **Checkout** : Récupération du code.
2. **Setup Node** : Installation de l'environnement Node.js.
3. **Install** : 'npm install' pour récupérer les dépendances (Jest, JSDoc).
4. **Tests** : Lancement automatique des tests unitaires ('npm test'). Si un test échoue, le push est marqué comme "Failed".
5. **Documentation** : Génération de la JSDoc et déploiement automatique sur GitHub Pages.

Le pipeline est automatiquement déclenché à chaque push et à chaque pull request sur la branche principale du dépôt. Cela garantit que toute modification du code est systématiquement testée et validée avant d'être intégrée.

En cas d'échec d'une étape du pipeline (par exemple un test unitaire non valide), le workflow est interrompu et le commit échoue. Cela empêche l'intégration de code instable et facilite l'identification rapide des régressions.

3.2 Tests unitaires

Les tests (avec 'Jest') couvrent la logique critique du métier, située dans `calc.js` et les modules de vote :

-Calcul des moyennes (arrondies), médianes et majorités.

-Logique "Strict" : Vérification que l'unanimité est bien détectée.

-Règles spéciales : Gestion du vote "Café" (exclus des calculs numériques).

3.3 Documentation

La documentation technique est générée via JSDoc. Chaque fonction clé (`calculerResultatVote`, `obtenirEtat`) est annotée. Le site de documentation est accessible en ligne via GitHub Pages.

4. Manuel Utilisateur et Fonctionnalités

4.1 Installation et lancement (Local)

Pour tester l'application sur votre machine :

1. Cloner le dépôt :

```
git clone https://github.com/votre-repo/projet-agile.git
cd projet-agile
```

2. Installer les dépendances :

   npm install

3. Lancer le serveur :

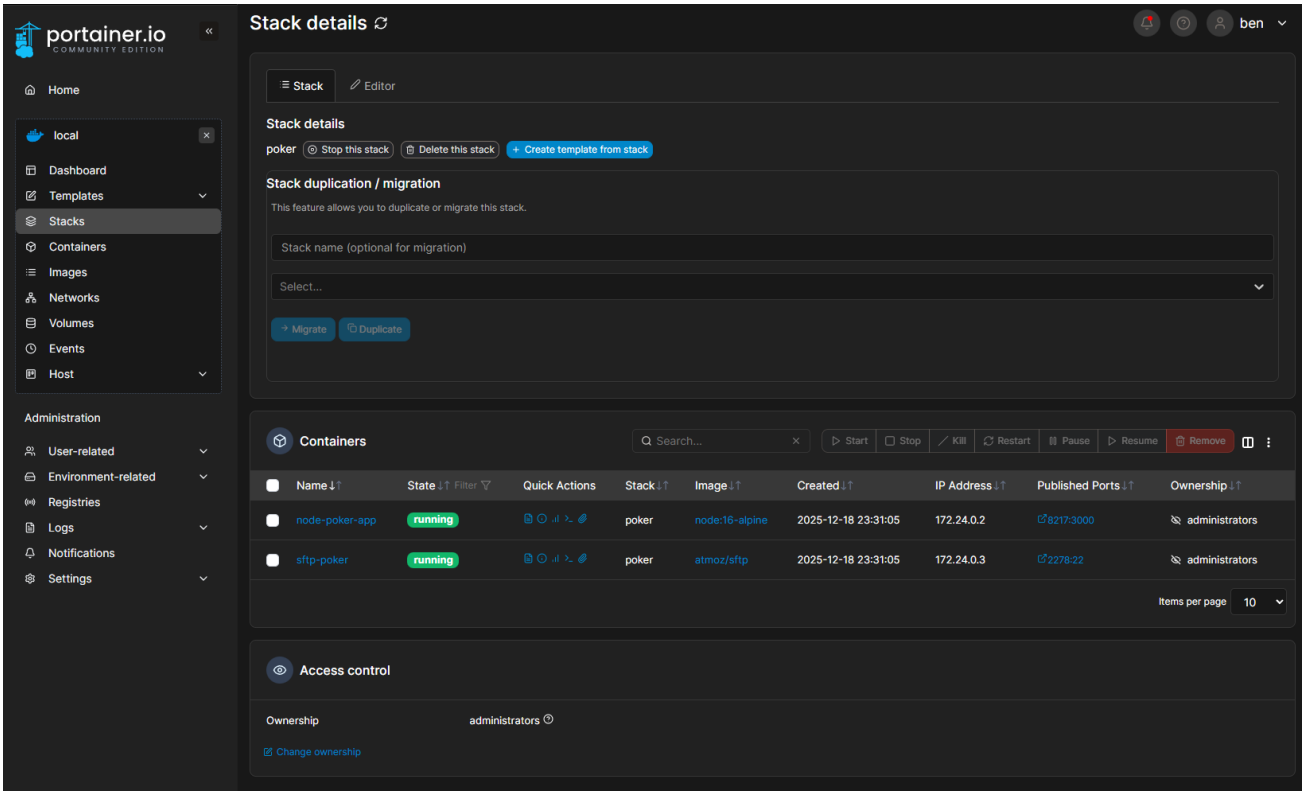
   npm start

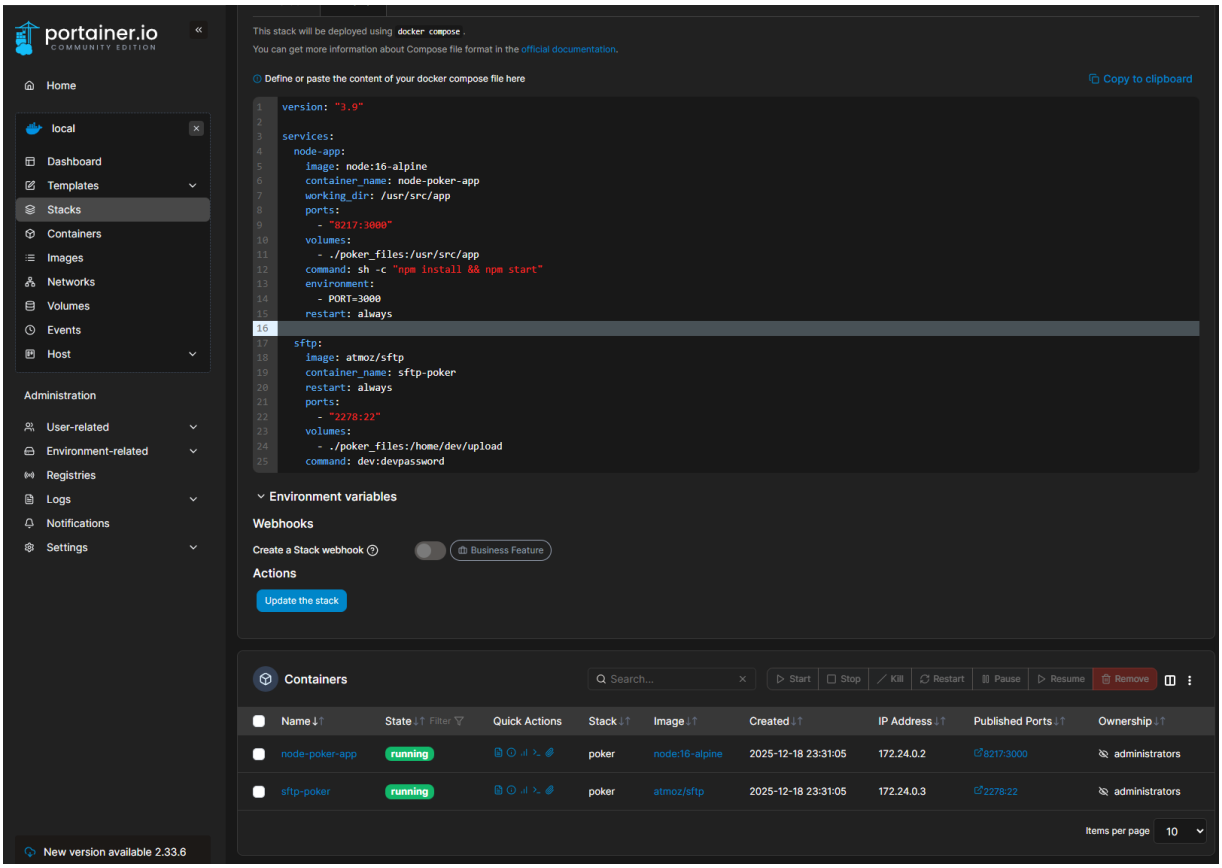
4. Accéder à l'application : Ouvrez votre navigateur sur `http://localhost:3000`.

4.2 Déploiement Docker (Bonus)

Nous avons également "dockerisé" l'application pour faciliter son déploiement sur un serveur (ex: LWS).

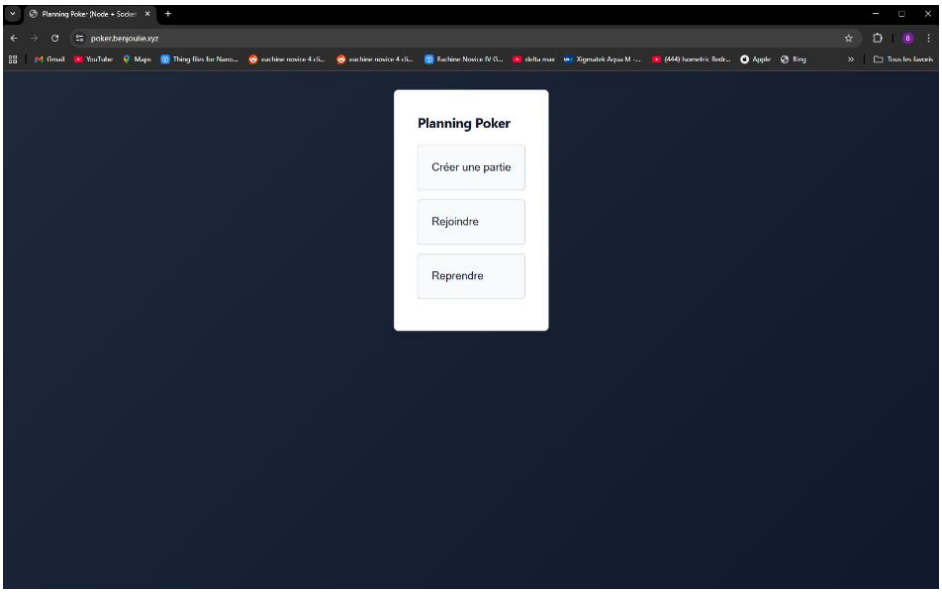
Un fichier `docker-compose.yml` est fourni. Il permet de lancer l'application Node.js ainsi qu'un serveur SFTP pour la gestion des fichiers en une seule commande.





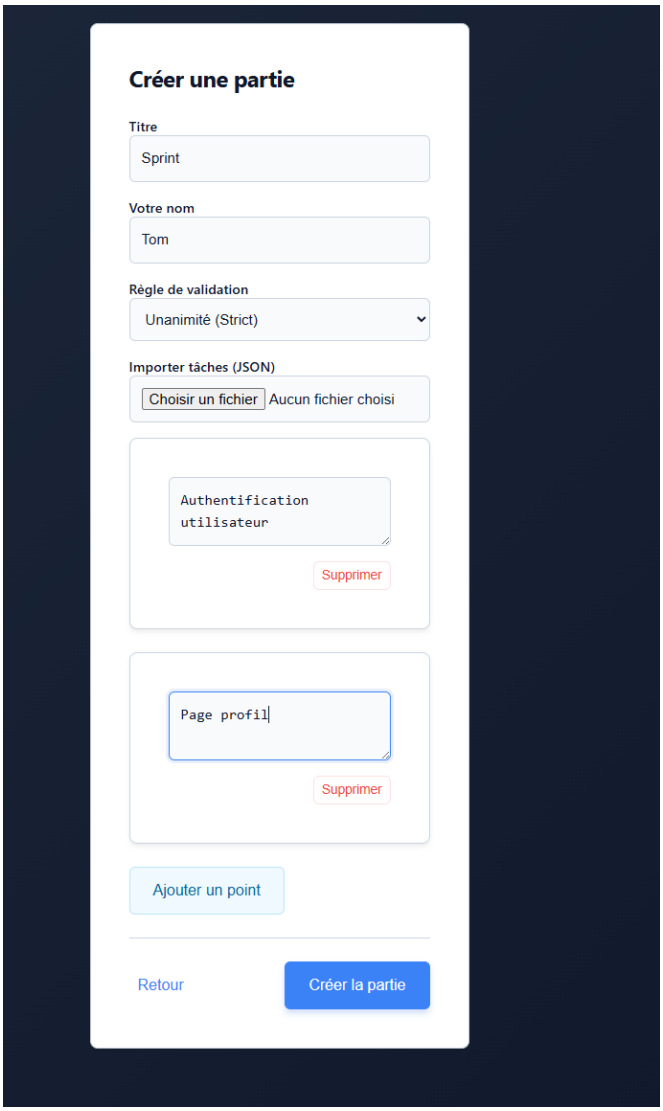
4.3 Modes de jeu et déroulement

1. **Menu** : Entrez votre pseudo et choisissez "Créer une partie".



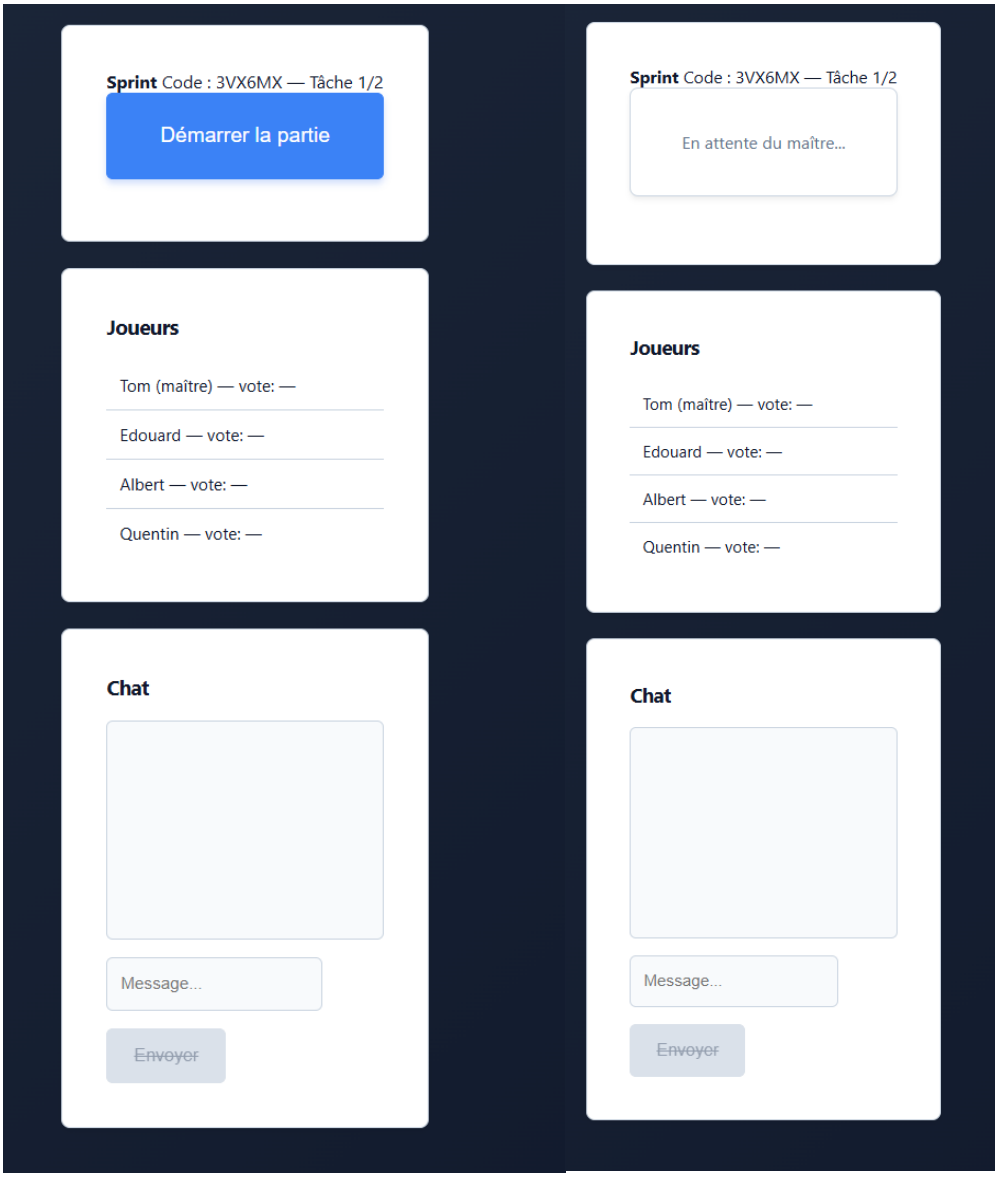
← Le menu principal

2. **Configuration** : Chargez votre fichier JSON de backlog. Choisissez le mode (ex: "Strict" pour l'unanimité au tour 1).



3. **Jeu** :

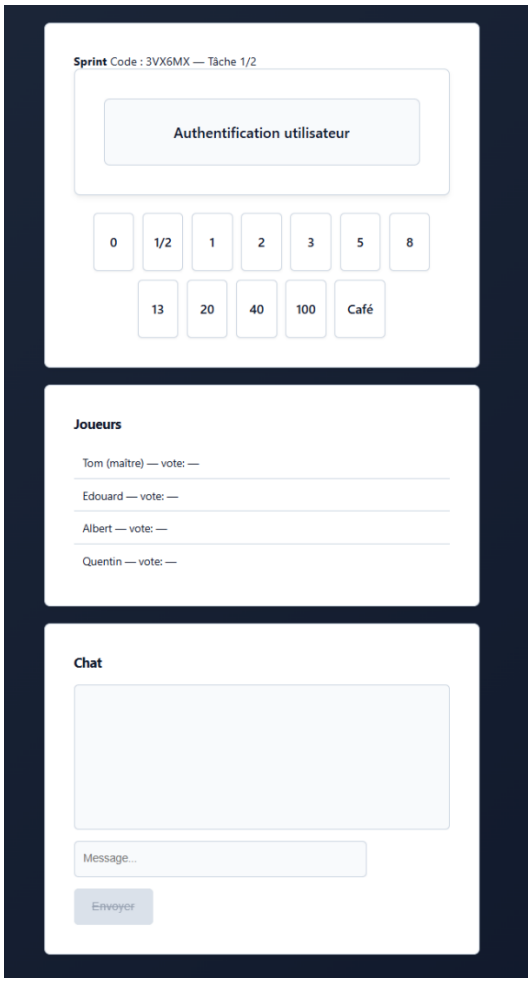
1- Les joueurs rejoignent avec le code partie.



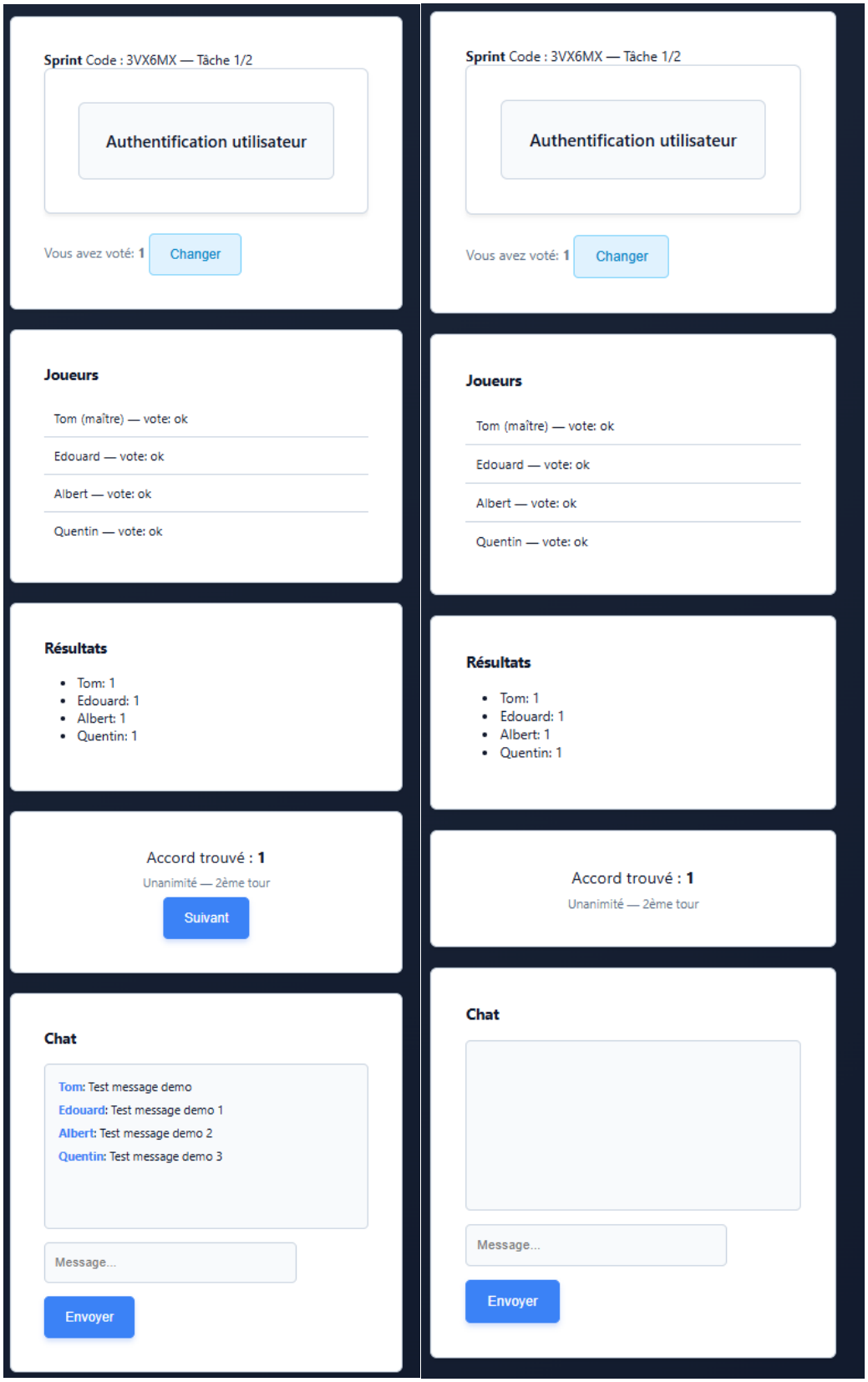
A gauche : on est du côté « maître ».

A droite : on est du côté « joueur ».

2- Tout le monde vote.



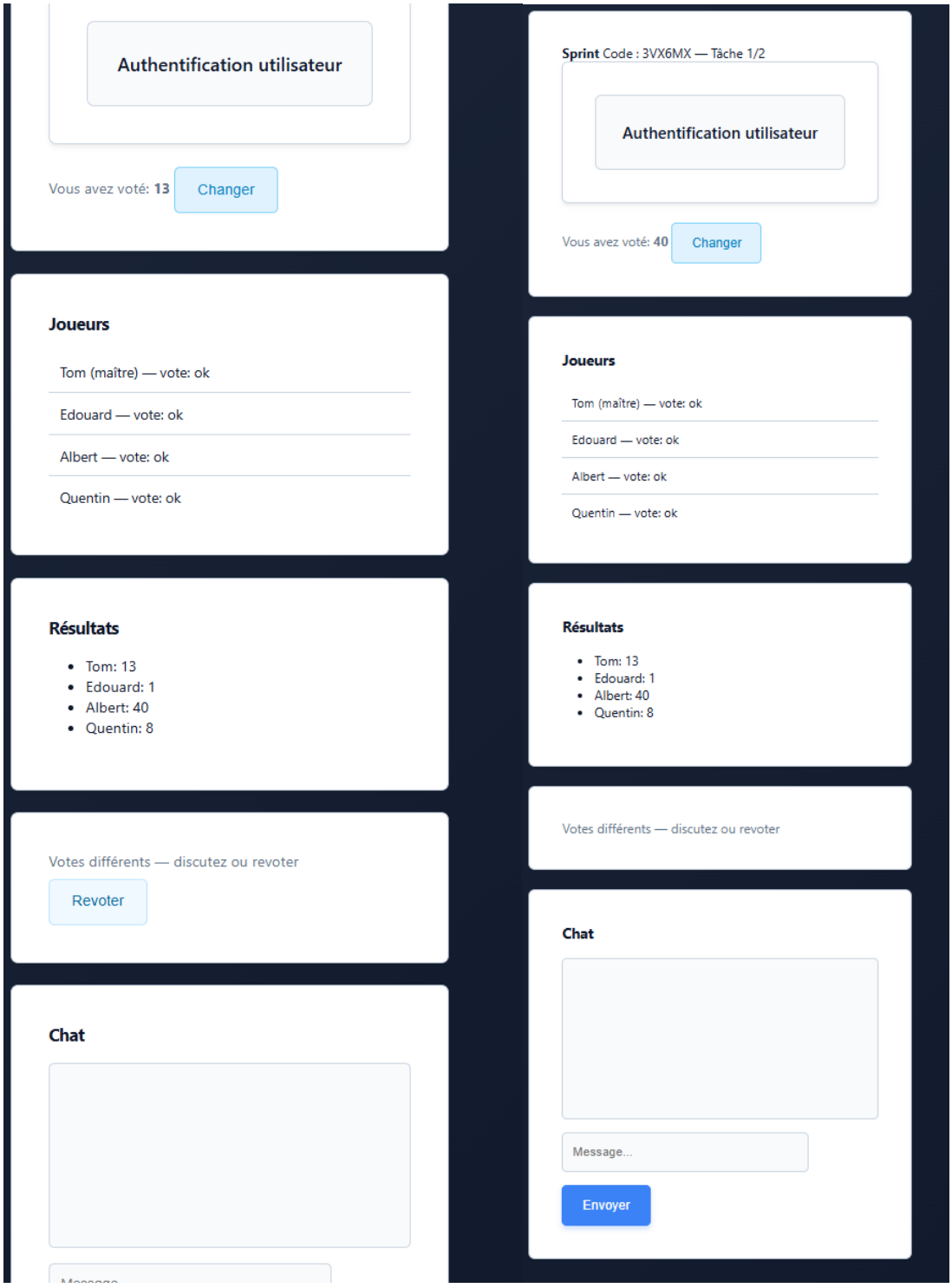
3- Si Unanimité/Accord : La tâche est validée, on passe à la suivante.



A gauche : on est du côté « maître ».

A droite : on est du côté « joueur ».

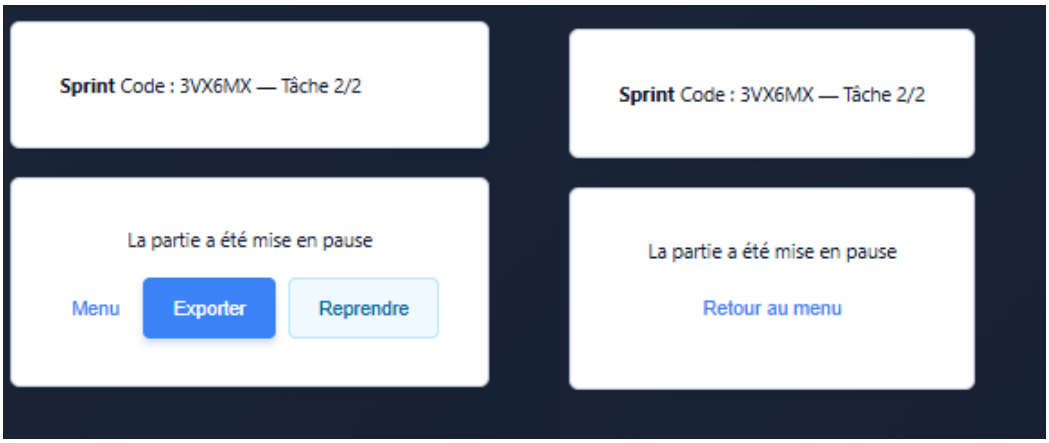
4- Si Désaccord : Le système demande un revote (et passe en mode Moyenne/Majorité au tour 2 si configuré).



A gauche : on est du côté « maître ».

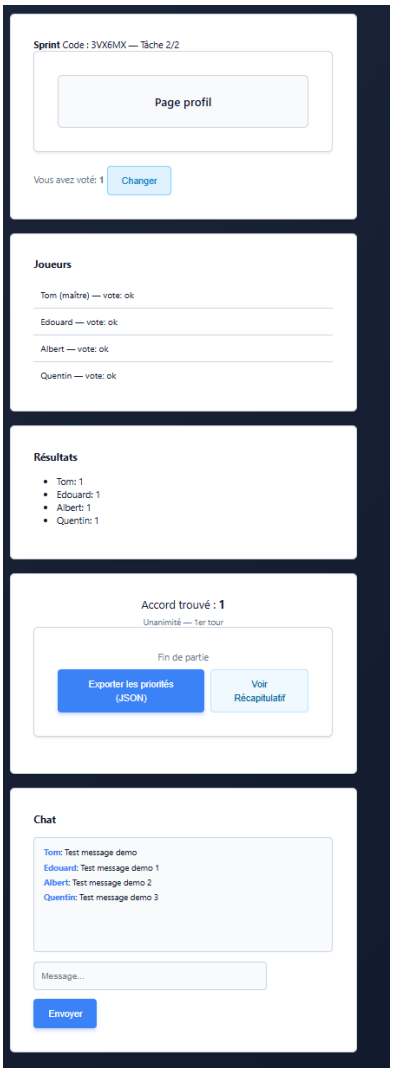
A droite : on est du côté « joueur ».

4. **Pause Café** : Si tous les joueurs votent la carte café, le jeu se met en pause et propose de sauvegarder.



A gauche : on est du côté « maître ». A droite : on est du côté « joueur ».

5. **Fin du jeu** : Lorsque toutes les tâches ont été votées, le maître du jeu peut exporter les résultats.



Conclusion

Ce projet de Planning Poker nous a permis de mettre en pratique les notions vues en cours, aussi bien sur le plan technique que méthodologique. La mise en place d'une application temps réel a nécessité une réflexion sur l'architecture, la synchronisation des données et la gestion de l'état du jeu.

Le choix de Node.js et de Socket.io a été adapté aux besoins du projet, notamment pour permettre aux joueurs de voter et de voir les résultats instantanément. La modélisation des données et l'architecture mise en place ont permis de garder une application claire et facile à faire évoluer.

Notre travail en trio, organisé autour du pair programming et de la répartition tournante des tâches, a favorisé la collaboration et une meilleure compréhension globale du projet par chaque membre de l'équipe. L'utilisation de tests unitaires et d'une intégration continue a également contribué à améliorer la fiabilité du code.

Enfin, même si certaines améliorations restent possibles, l'application répond aux objectifs fixés et constitue une base fonctionnelle pour une session de Planning Poker en ligne.