Valentin Berger
Axel Doussoux

Projet CAPI : Application de Planning Poker

I- Présentation du projet

1) Contexte : planning poker

Le planning poker est une méthode très permettant d'estimer la difficulté des étapes à réaliser lors d'un projet réalisé avec la méthode Scrum. Chaque membre de l'équipe possède un ensemble de cartes suivant cartes représentant des points d'effort, généralement basés sur la suite de Fibonacci. Certaines cartes spéciales peuvent être également utilisées comme la carte "?" lorsqu'on a aucune idée du niveau de difficulté de la tâche, la carte "café" lorsqu'on ressent le besoin de discuter plus longuement avant de décider, et la carte ∞ lorsqu'une tâche nous semble impossible à estimer car trop incertaine ou complexe. Chaque membre choisit une carte pour chaque estimation à faire en secret, puis tout le monde la révèle en même temps. Pour décider d'une estimation, on peut utiliser plusieurs modes de jeu. Le vote par unanimité demande à ce que tous les membres choisissent la même carte. La majorité absolue retient l'estimation représentant 51% ou plus des suffrages. La majorité relative retient le niveau de difficulté le plus exprimé. On peut également retenir la moyenne ou la médiane de tous les votes.

2) Objectif du projet

L'objectif est de développer une application permettant de joueur au planning poker. L'application peut être utilisée à distance, chaque joueur utilisant son propre dispositif, ou en mode local, où les joueurs choisissent chacun leur tour leurs cartes. Un menu permettrait au joueur de créer une partie ou de rejoindre une partie existante. Au moins deux modes de jeux devront être implémentés donc le mode de jeu strict. Les fonctionnalités à estimer sont entrées sous forme de backlog en JSON.Des fonctionnalités supplémentaires, comme un chronomètre ou un espace de discussion, peuvent être ajoutées pour améliorer l'expérience de jeu.

II - Choix effectués

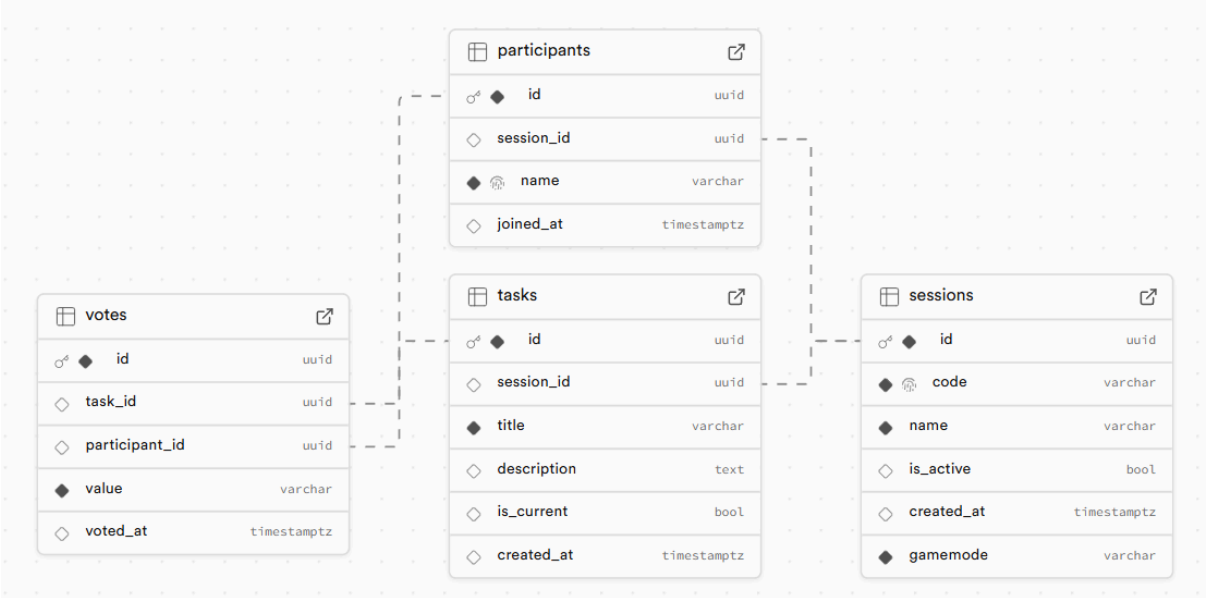
1) Développement en TypeScript

Nous avons choisi de développer notre application en TypeScript, un langage dérivé de JavaScript offrant un typage statique pour les variables, fonctions et objets. Ce typage permet de limiter le nombre de bugs en détectant de nombreuses erreurs dès la compilation, bien avant l'exécution du code. De plus, TypeScript s'intègre parfaitement dans un

environnement de développement comme VSCode, offrant des fonctionnalités telles que l'autocomplétion et la vérification en temps réel des types, ce qui facilite grandement le travail du développeur et améliore la maintenabilité du code.

2) Application multi joueurs distant avec Supabase

Nous voulions initialement essayer de réaliser la partie multi joueur distant de l'application avec Next.js. Cependant, n'ayant aucune expérience dans le développement d'une application multijoueurs, mais disposant en revanche de solides connaissances en bases de données relationnelles, , nous avons choisi de passer par une solution basée sur la base de données pour gérer le multijoueur.
Nous avons donc opté pour Supabase, une solution Backend-as-a-Service qui fournit une base de données relationnelle PostgreSQL avec des fonctionnalités temps réel intégrées. Cette approche nous permet de synchroniser facilement les actions des joueurs entre eux sans avoir à développer un serveur multijoueur complet. De plus, Supabase s'intègre parfaitement avec TypeScript, grâce à la génération automatique des types à partir de la base de données, ce qui réduit le risque d'erreurs et facilite la gestion des données côté front-end. Voici ci-dessous le modèle de données Supabase, où nous avons créé 4 tables : la table sessions stocke toutes les parties. La table participants regroupant toutes les informations sur les participants d'une partie. La table Task regroupe toutes les informations sur les tâches à évaluer. La table vote enregistre tous les votes avec pour chaque tuple, l'identifiant de la tâche et l'identifiant de l'utilisateur.



3) UI avec le Framework React et Tailwind CSS

Pour le développement de l'interface utilisateur, nous avons fait le choix d'utiliser React en combinaison avec Tailwind CSS. React, en tant que bibliothèque JavaScript, nous permet de construire des interfaces modulaires et réactives, où chaque composant peut gérer son propre état et se mettre à jour de manière efficace en fonction des interactions utilisateur.

Cette approche facilite la structuration du code et améliore la maintenabilité de l'application, notamment pour une interface interactive comme celle d'un jeu multijoueur.
Tailwind CSS complète React en proposant un système utilitaire de classes CSS, ce qui nous permet de styliser rapidement et de manière cohérente l'ensemble des composants, sans avoir à écrire de feuilles de style complexes. Ayant déjà l'habitude d'utiliser Bootstrap pour le développement d'interfaces utilisateur, nous avons trouvé Tailwind CSS relativement facile à prendre en main, notamment grâce à son utilisation de Flexbox, un concept que nous connaissions déjà et qui facilite la mise en page des éléments.

III - L'intégration continue

1) Git

Nous avons utilisé Git pour assurer la collaboration ainsi qu'une gestion des versions de notre projet efficacement. En effet, Git nous permet de suivre les modifications apportées au code, de revenir à des versions précédentes si nécessaire, et de gérer plusieurs branches pour le développement de nouvelles fonctionnalités sans perturber la version stable de l'application. Nous avions dans notre cas une branche de développement pour chaque membre du groupe, afin de s'assurer que chacun puisse travailler de manière indépendante sur sa partie du projet : Valentin se chargeait du front-end avec React et Tailwind CSS, tandis qu'Axel développait la partie back-end avec Supabase et la logique serveur. Cette organisation nous a permis de prévenir les conflits de code, de tester les fonctionnalités séparément, et de faciliter l'intégration des modifications dans la branche principale.

2) Tests unitaires

Afin de garantir la fiabilité et la maintenabilité de notre application, nous avons mis en place des tests unitaires en utilisant Jest et React Testing Library. Ces outils nous permettent de simuler des dépendances, de vérifier le comportement des fonctions et d'assurer la robustesse du code. Parmi les tests réalisés, nous avons vérifié la génération de code dans le service des sessions afin de s'assurer que la fonction produit toujours une chaîne de six caractères, garantissant le format attendu pour les sessions. Dans le service des participants, nous avons contrôlé la vérification des pseudos, pour confirmer que la méthode retourne false lorsqu'un pseudo n'existe pas et true lorsqu'il est déjà utilisé. Pour le service des tâches, nous avons testé la création de tâche, en vérifiant que l'ajout à la base de données fonctionne correctement et que l'objet retourné correspond bien à la tâche créée. Enfin, dans le service des votes, nous avons vérifié que la récupération des votes retourne la liste exacte des votes associés à une tâche et que leur nombre correspond aux attentes.

3) Documentation

la compréhension et la maintenance de notre application, nous avons mis en place un processus automatique de génération et de déploiement de la documentation avec TypeDoc et GitHub Actions. À chaque push sur la branche `main`, la documentation est générée à

partir des fichiers TypeScript et publiée sur GitHub Pages, ce qui permet d’avoir une documentation à jour et accessible facilement par tous les membres de l’équipe.