Établissement : *Université Lumière Lyon 2*
Formation : *Master 1 Informatique*

RAPPORT DE PROJET

Application de Planning Poker (CAPI)

Awa KARAMOKO
Matthieu GUILLEMIN

Année Universitaire : 2023-2024

Table des matières

Table des figures

“Dans les détours compliqués du développement web, la clé du succès réside dans une planification judicieuse et une exécution précise.”

Introduction

Ce rapport présente notre vision audacieuse de repenser le Planning Poker en le transposant dans le monde numérique. À travers la création de ce site web dédié, nous cherchons à élever la planification au-delà des frontières physiques, permettant à notre équipe de collaborer de manière transparente, de voir des statistiques et la représentation graphique de chaque partie. De l'interface conviviale aux fonctionnalités innovantes, chaque ligne de code de ce site web est imprégnée de notre désir de simplifier le processus de planification et de favoriser une collaboration fluide. Nous plongerons dans les aspects techniques, les défis rencontrés et les avantages attendus de cette initiative, dévoilant ainsi comment notre site web de Planning Poker deviendra un atout essentiel dans notre boîte à outils de gestion de projet.

I. Compréhension du sujet

L'application de Planning Poker a pour objectif de faciliter la planification de projets logiciels en utilisant une méthode de vote collaborative.

Une "partie" de planning poker est découpé comme suit :

1. Configuration initiale de la partie

L'utilisateur (administrateur-joueur) démarre l'application, peut ***spécifier le nombre de joueurs***, ***leur attribuer un pseudo à chacun*** et ***saisir la liste des fonctionnalités (backlog) à planifier*** (celles-ci seront converties au format JSON une fois saisies depuis l'interface). De plus, il peut ***choisir parmi différents modes de jeu***, à savoir : *strict* (par défaut), *moyenne*, *médiane*, *majorité absolue, majorité relative*.

2. Vote des joueurs

Les fonctionnalités saisies précédemment apparaissent les unes après les autres à l'écran. ***Chaque joueur est invité à voter via une pop-up*** ("[pseudo], à ton tour de voter"). Pour voter, ***il choisit une carte parmi celles qui sont affichées à l'écran*** en cliquant dessus - ces dernières représentent différentes estimations (par exemple, 1, 2, 3, 5, 8). Une fois le vote du joueur en cours effectué, ***un bouton de confirmation se révèle*** en bas de l'écran. En cliquant dessus, une pop-up appelle le joueur suivant à voter. La boucle se répète jusqu'à ce que tous les joueurs aient voté.

3. Analyse des votes

Une fois que tous les joueurs ont voté, ***l'application valide ou rejette la fonctionnalité*** en fonction du mode de jeu sélectionné durant la configuration initiale de la partie.

- Mode "*strict*" : La fonctionnalité est validée si et seulement si tous les joueurs ont émis la même estimation.
- Mode "*moyenne*" : La fonctionnalité est validée dans tous les cas. Son estimation est la moyenne des estimations de tous les joueurs.
- Mode "*médiane*" : La fonctionnalité est validée dans tous les cas. Son estimation est la médiane des estimations de tous les joueurs.
- Mode "*majorité absolue*" : La fonctionnalité est validée si et seulement si une même estimation a été votée par 50% des joueurs, ou plus.
- Mode "*majorité relative*" : La fonctionnalité est validée si et seulement si une estimation a été votée par une majorité de joueurs (comparée aux autres estimations).

Lorsqu'une fonctionnalité est validée, ***l'application enregistre l'estimation de complexité attribuée par l'équipe*** (suivant le mode de jeu). Sinon, ***la fonctionnalité reste dans la liste pour un deuxième tour de votes***.

4. Gestion de la carte café

Si tous les joueurs utilisent la carte "café" (pour signaler un besoin de pause), ***l'application enregistre l'état d'avancement du backlog***, ce qui permet de reprendre la partie ultérieurement.

A la fin d'une partie, une fois que toutes les fonctionnalités ont été planifiées, un fichier JSON est généré. Il contient les estimations de complexité pour chaque fonctionnalité. Ce fichier peut être utilisé pour suivre la planification du projet.

Fonctionnalité bonus : Affichage de la répartition des votes

Nous avons pris la décision d'afficher la répartition des votes sous la forme d'un diagramme en bâtons après le vote de tous les joueurs pour un backlog donné. Nos motivations pour la réalisation de cette fonctionnalité bonus seront présentées un peu plus tard dans ce document.

II. Maquettes

Pour la bonne réalisation de notre projet, nous avons trouvé judicieux de mettre en place 6 interfaces. Afin de faciliter la réalisation de ces différentes pages et d'avoir une compréhension globale du projet, nous vous présentons ci-dessous la capture de nos maquettes :

Figure 1 : Maquette du MENU

Le ***MENU*** permet de choisir un mode de jeu, de définir le nombre de joueurs ainsi que leurs pseudos, puis de saisir une liste de backlogs dans le but de générer un fichier JSON.

Cliquer sur le bouton JOUER amène vers le ***DASHBOARD***.

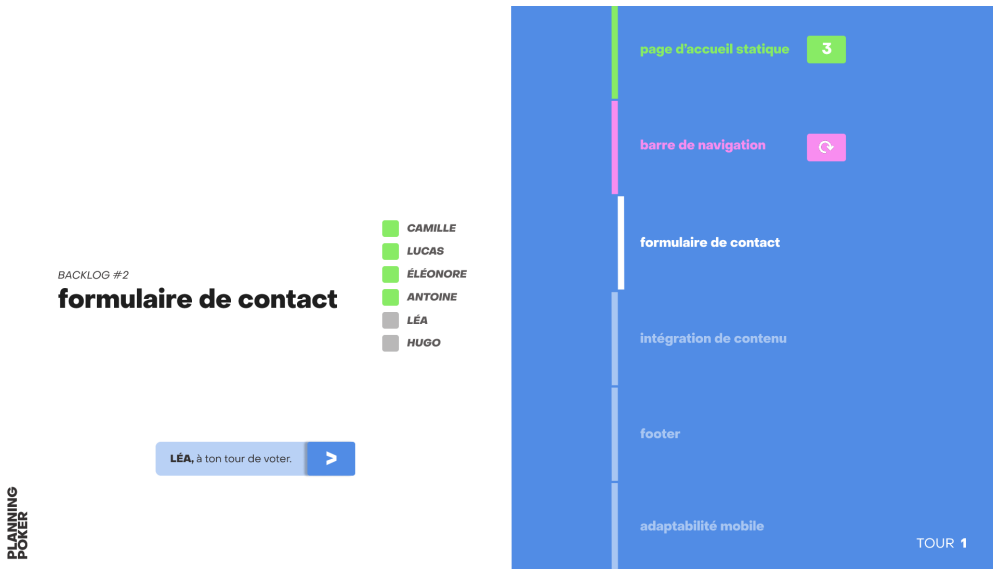


Figure 2 : Maquette du DASHBOARD

Le ***DASHBOARD*** est l'interface de suivi de l'avancée d'une partie. A droite, sont indiqués l'état des votes des différents backlogs : soit "à voter", soit "validé", soit (suivant le mode de jeu) "nécessitant un deuxième tour". A gauche, apparaît l'avancement des votes pour le backlog en cours : quels joueurs ont déjà voté et qui est le prochain à devoir voter.

Cliquer sur la flèche bleue dans la partie gauche de l'écran amène vers le ***CARDSBOARD***.

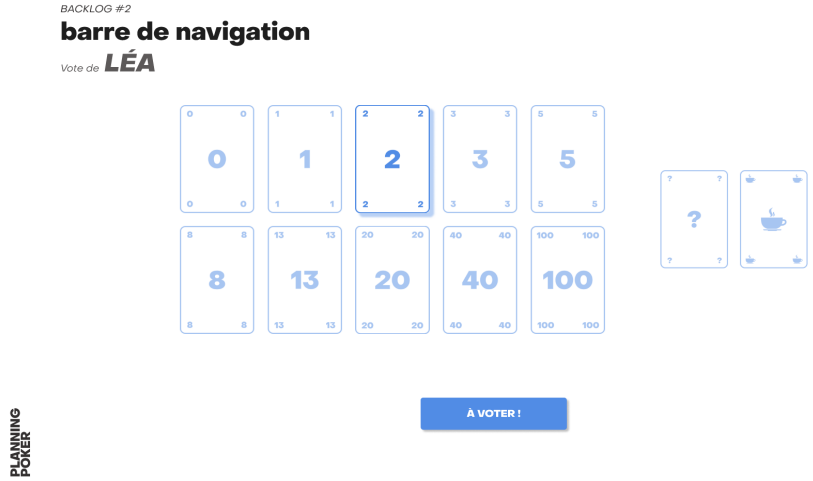


Figure 3 : Maquette du CARDSBOARD

Le ***CARDSBOARD*** est l'interface de vote. Elle permet au joueur courant de voter pour le backlog courant. Il sélectionne une carte et clique sur le bouton A VOTER !

Cette action ramène vers le ***DASHBOARD*** ou vers l'écran ***PAUSE CAFÉ*** si le joueur courant est le dernier à voter pour le backlog en cours et que tous les joueurs ont sélectionné la carte CAFÉ.

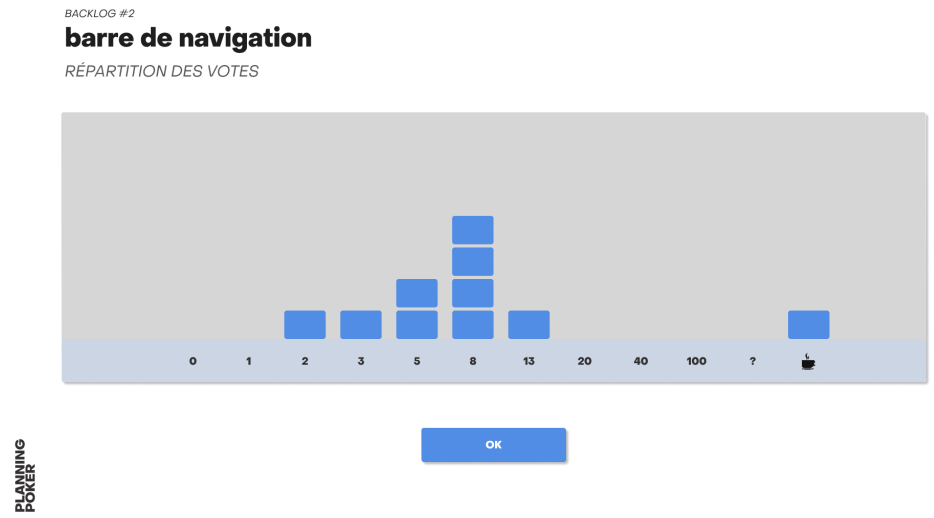


Figure 4 : Maquette de la page RÉPARTITION DES VOTES

La ***RÉPARTITION DES VOTES*** apparaît à la fin de chaque backlogs. Le graphique généré permet de connaître les estimations tendances de l'équipe, anonymement, pour aider à voter au tour d'après.

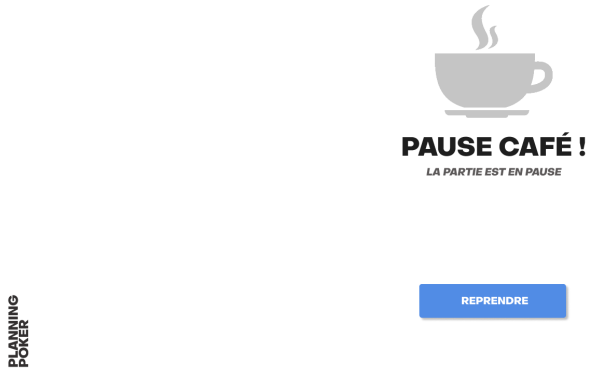


Figure 5 : Maquette de l'écran PAUSE CAFÉ

La ***PAUSE CAFÉ*** est un écran de pause. L'avancée des votes est sauvegardée et les joueurs peuvent penser à autre chose. Dès qu'ils sont prêts à reprendre la partie, ils ont juste à cliquer sur le bouton REPRENDRE.

Cela les ramène vers le ***DASHBOARD***.

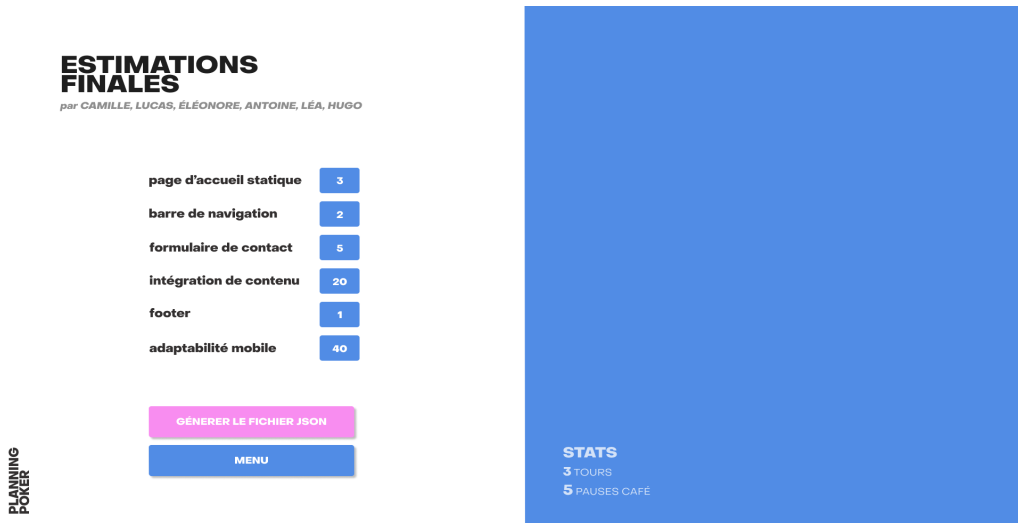


Figure 6 : Maquette de la page RESULTS

La page ***RESULTS*** présente les résultats des différents votes une fois que la partie est terminée, autrement dit quand tous les joueurs sont parvenus à un consensus (suivant le mode de jeu) pour chacun des backlogs. C'est depuis cette page qu'on peut retourner au ***MENU***, et générer le fichier JSON final, contenant la liste des backlogs associés à leurs estimations ainsi que le mode de jeu utilisé. Quelques statistiques "optionnelles" (nombre de pauses café) sont également visibles dans la partie droite de l'écran.

III. Langages de programmation et frameworks

Dans le cadre de notre projet, nous avons décidé d'utiliser JavaScript comme langage de programmation. Les frameworks Vue.JS et Chart.JS nous ont également été utiles.

1. JavaScript

JavaScript est un langage de programmation de haut niveau, interprété et orienté objet. Il est principalement utilisé pour créer des pages web interactives et dynamiques. Il a été créé à l'origine pour être exécuté dans les navigateurs web des utilisateurs afin d'améliorer l'expérience utilisateur en permettant des interactions plus riches et en temps réel sur les pages web.

2. Vus.JS

Vue.JS est un framework JavaScript progressif pour la construction d'interfaces utilisateur (UI). Il est souvent utilisé pour créer des applications web à page unique (Single Page Applications - SPA) et offre une approche réactive à la construction d'interfaces utilisateur.

3. Chart.JS

Chart.JS est une bibliothèque JavaScript simple et flexible permettant de créer des graphiques et des visualisations de données interactives sur des pages web. Nous l'utilisons dans notre

projet pour afficher des données de manière graphique dans le navigateur, facilitant la compréhension et l'analyse des informations présentées.

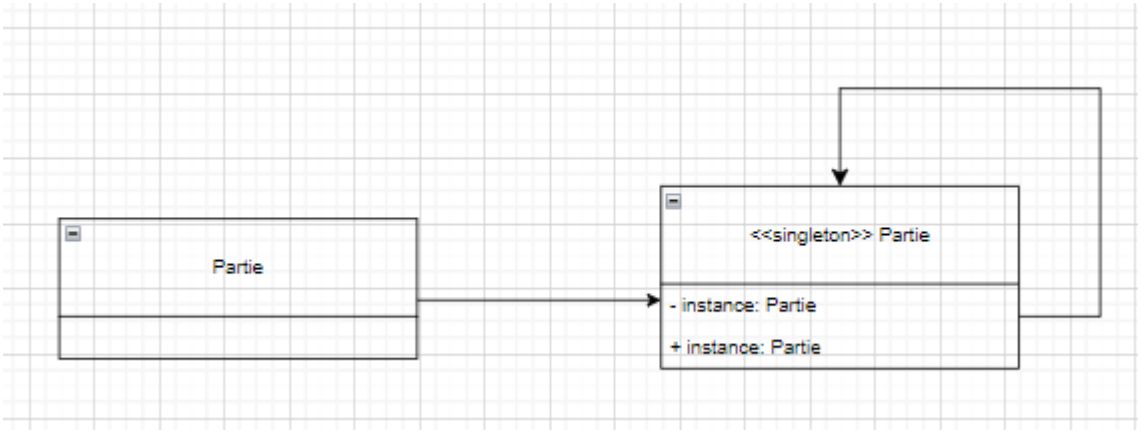
IV. Design patterns utilisés et diagrammes de classe associés

1. Singleton pattern

Le modèle Singleton est un modèle de conception qui garantit qu'une classe a une seule instance et fournit un point d'accès global à cette instance. Cette approche est utile dans divers contextes, y compris pour garantir qu'il n'y a qu'une seule instance d'une classe spécifique dans une application.

Voici comment nous l'avons utilisé dans le cadre de notre projet :

- ***Diagramme de classe Singleton :***



- ***Code :***

```
class Partie {
      constructor() {
            if (!Partie.instance) {

                  this.mode = "strict";
                  this.voteStrategy =
                  VoteStrategyFactory.createStrategy(this.mode);


                  this.backlogs = [];
                  this.currentBacklog = 0;


                  this.players = [];
                  this.currentPlayer = 0;
```

```
                    this.showChart = true;
                    this.infosForChart = {};

                    this.nbCoffeeBreak = 0;

                    Partie.instance = this;
                }
        …
        }
}
```

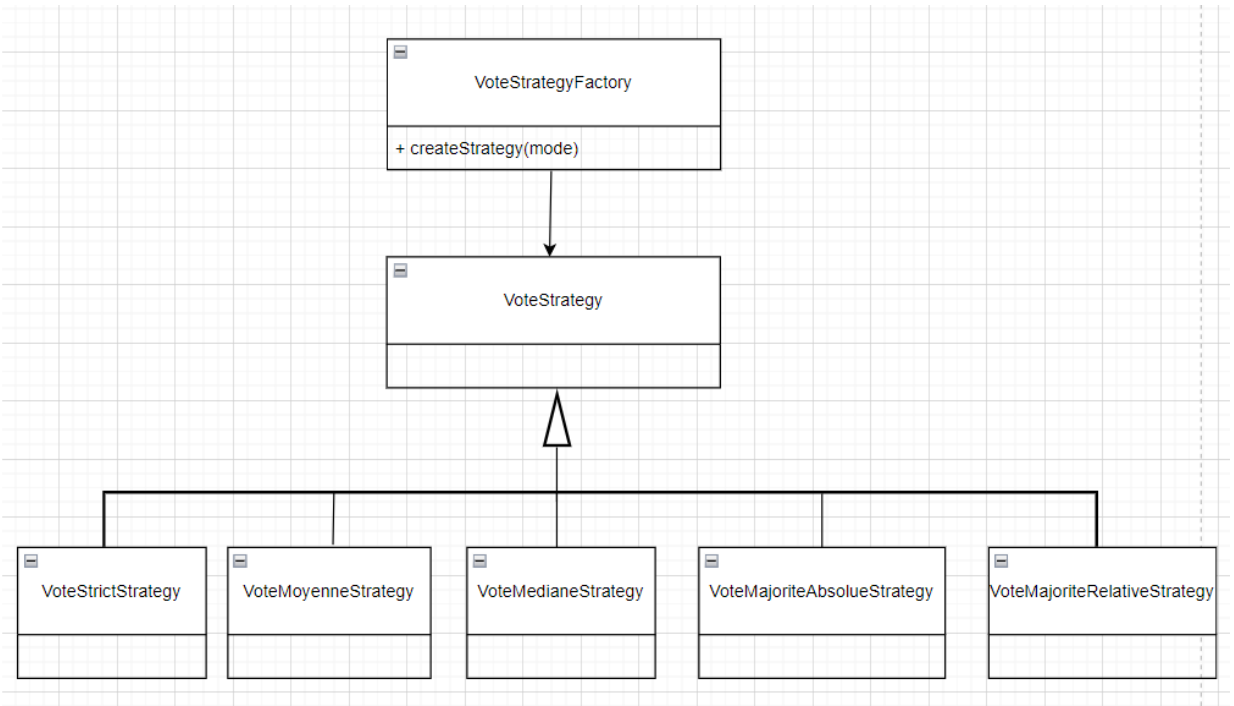
La classe `Partie` utilise un motif de conception singleton pour s'assurer qu'une seule instance de la classe existe. Cela signifie qu'à chaque tentative de création d'une nouvelle instance de `Partie`, la même instance existante est renvoyée. Cela est réalisé en utilisant une variable statique `instance` et en retournant cette instance si elle existe déjà dans le constructeur.

2. Factory pattern

La méthode Factory (ou Factory Method) est un motif de conception permettant de créer des objets sans spécifier explicitement la classe de l'objet que nous voulons créer. Cela peut être appliqué à une application de Planning Poker pour créer différentes instances d'objets liés à la planification, comme des cartes de vote, des sessions de planification, etc.

Voici comment nous l'avons utilisé dans le cadre de notre projet :

- ***Diagramme de classe Factory:***



- ***Code :***

```
class VoteStrategyFactory {
	static createStrategy(mode) {
	switch (mode) {
		case "strict":
		return new VoteStrictStrategy();
		case "moyenne":
		return new VoteMoyenneStrategy();
		case "médiane":
		return new VoteMedianeStrategy();
		case "majorité absolue":
		return new VoteMajoriteAbsolueStrategy();
		case "majorité relative":
		return new VoteMajoriteRelativeStrategy();
		default:
		return null;
	}
	}
}
```

La classe `VoteStrategyFactory` est une classe statique qui offre une méthode statique `createStrategy(mode)` permettant de créer des instances de stratégies de vote en fonction du mode de jeu spécifié. Cette classe utilise une instruction switch pour déterminer le type de stratégie à instancier en fonction du mode passé en paramètre.

3. Strategy pattern

Le modèle de conception Strategy vise à définir une famille d'algorithmes, les encapsuler, et les rendre interchangeables. Il permet à un client de choisir l'algorithme approprié à utiliser à partir d'une famille d'algorithmes, et il permet également de modifier l'algorithme indépendamment des clients qui l'utilisent.

Voici comment nous l'avons utilisé dans le cadre de notre projet :

- ***Diagramme de classe Strategy:***

(voir le diagramme du pattern Factory)

- ***Code :***

```
class VoteStrategy {
	computer Vote() {
		return;
	}
}
```

```
class VoteStrictStrategy extends VoteStrategy {
      computeVote(playersWithNumericVotes) {
            let state, value;
            let firstVote, currentVote;

            firstVote = parseFloat(playersWithNumericVotes[0].hasVoted);
            value = firstVote;
            state = 1;
            for (let i = 1; i < playersWithNumericVotes.length; i++) {
                  currentVote =
                        parseFloat(playersWithNumericVotes[i].hasVoted);
                  if (isNaN(currentVote) || currentVote !== firstVote) {
                  value = undefined;
                  state = 2;
                  }
            }
            return {'value': value, 'state': state};
      }
}
```

La classe `VoteStrategy` est la classe de base abstraite qui définit une méthode `computeVote()`. Cette méthode est vide dans cette classe de base, laissant aux sous-classes la responsabilité de fournir leur propre logique de calcul de vote.
Par exemple, la classe `VoteStrictStrategy` étend `VoteStrategy` et implémente une stratégie de vote stricte. Elle vérifie si tous les votes numériques des joueurs sont identiques. Si c'est le cas, elle retourne la valeur du vote et l'état 1 (réussite), sinon elle retourne undefined et l'état 2 (échec).

V. Produit final : Déroulement d'une partie

Dans cette partie, nous allons effectuer une simulation avec le mode "strict" et vous mettre des captures d'écran afin de vous permettre de comprendre le fonctionnement de notre application.

1. Présentation de la page du MENU

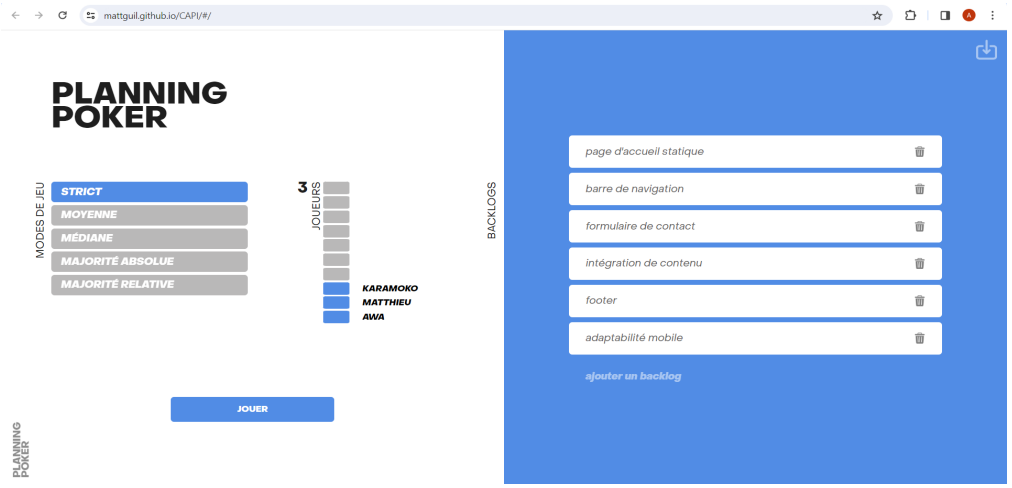


Figure 7 : ¨Page MENU

Cette capture nous montre la page d'accueil, c'est-à-dire le menu. L'utilisateur devra choisir le mode de jeu, le nombre de joueurs, insérer leurs pseudos et ajouter ou télécharger (en cliquant sur le bouton en haut à droite) un ou des backlogs. Le mode par défaut est strict et si l'une de ces fonctionnalités n'est pas renseignée, l'utilisateur ne pourra pas lancer un jeu.

2. Présentation de la page DASHBOARD

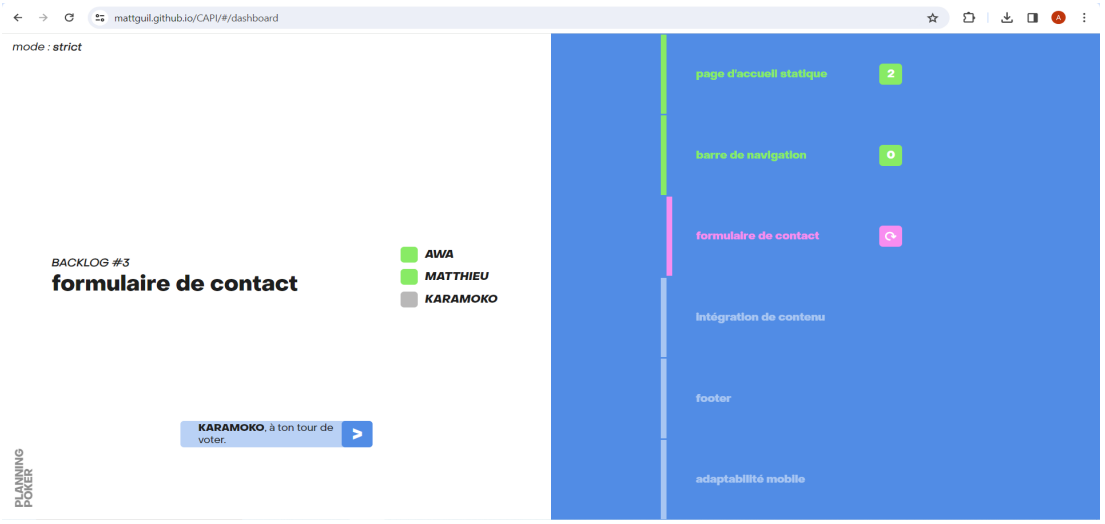


Figure 8 : Page DASHBOARD

Cette page est le dashboard du backlog. Elle s'affiche lorsque l'utilisateur clique sur le bouton "jouer" depuis la page du menu. Nous avons à gauche le nom du backlog en cours, à côté la liste des joueurs dans l'ordre d'appel et un bouton qui permet au joueur suivant de voter.

Rappel : Pour le mode strict, tous les joueurs doivent jouer la même carte.

Ici, pour le backlog “page d'accueil statique” tous les joueurs ont voté 2 et cela est indiqué à côté du backlog en question, donc il devient vert (il est validé). A contrario, pour le backlog “formulaire de contact”, les joueurs ont voté des cartes différentes alors le backlog se met en rose pour indiquer qu'un autre tour de vote est nécessaire.

3. Présentation de la page CARDSBOARD

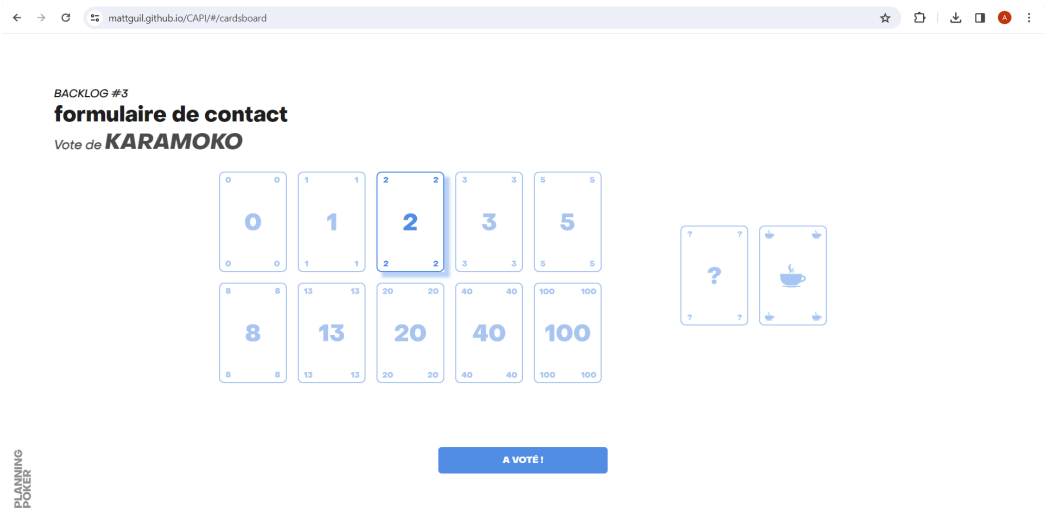


Figure 9 : Page CARDSBOARD

Cette page est l’interface de vote. Elle s’affiche lorsque l'utilisateur clique sur le bouton “KARAMOKO, à ton tour de voter” . Elle permet au joueur courant de voter pour le backlog courant. Il sélectionne une carte et clique sur le bouton A VOTER ! Nous avons deux cartes spéciales qui sont : la carte *café* et la carte *joker*, elles sont les seules à ne pas avoir une valeur numérique et elles sont traitées différemment dans le calcul des votes.

4. Présentation de la page RÉPARTITION DES VOTES

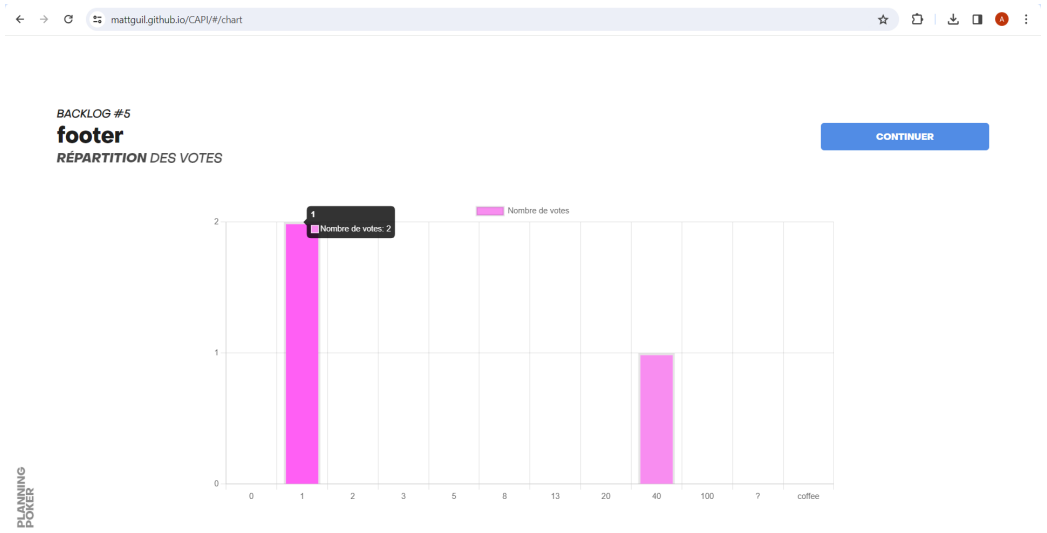


Figure 10 : Page RÉPARTITION DES VOTES

Cette page est une représentation graphique des votes de manière anonyme et donne la possibilité de voir le nombre de joueurs ayant voté chaque carte. Elle s'affiche automatiquement lorsque tous les joueurs finissent de voter pour le backlog en cours. Elle permet de connaître les estimations tendances de l'équipe, anonymement, pour aider à voter au tour d'après. Prenons l'exemple du mode strict, si tous les joueurs ont voté la même carte les barres du graphe s'affichent en vert, sinon en rose.

5. Présentation de la page PAUSE CAFÉ

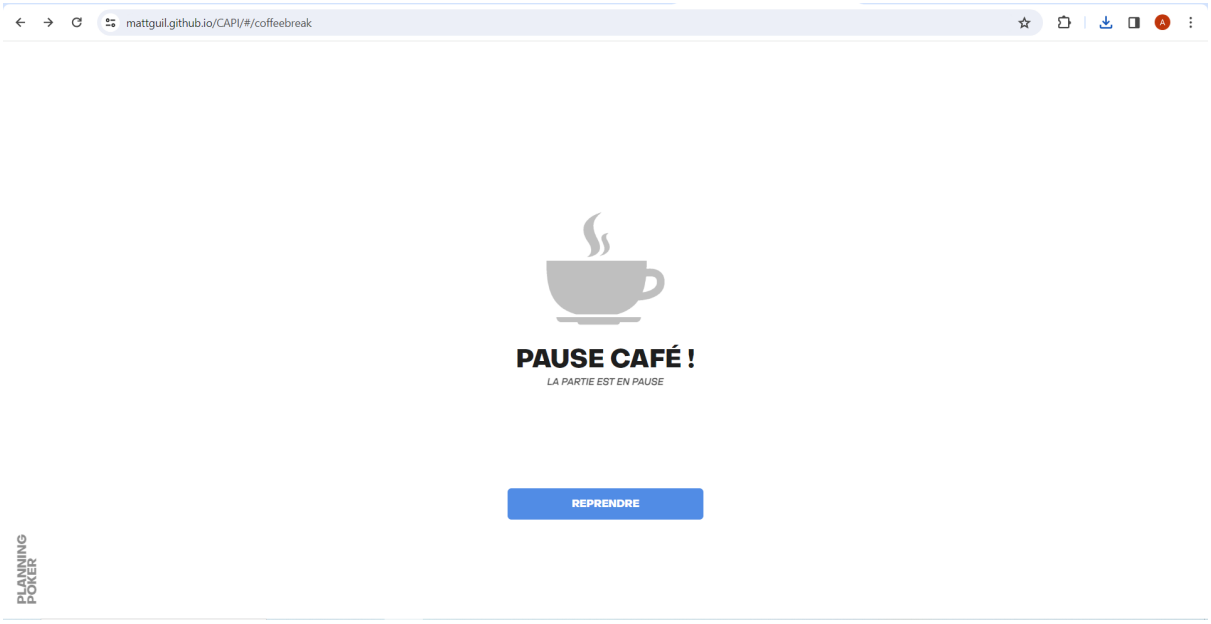


Figure 11 : Page PAUSE CAFÉ

Cette page indique que tous les joueurs ont voté la carte café. C'est la pause ! L'avancée des votes est sauvegardée automatiquement dans un fichier JSON (*backlogs.json*) et les joueurs peuvent penser à autre chose. Dès qu'ils sont prêts à reprendre la partie, ils ont juste à cliquer sur le bouton REPRENDRE.

6. Présentation de la page RESULTS

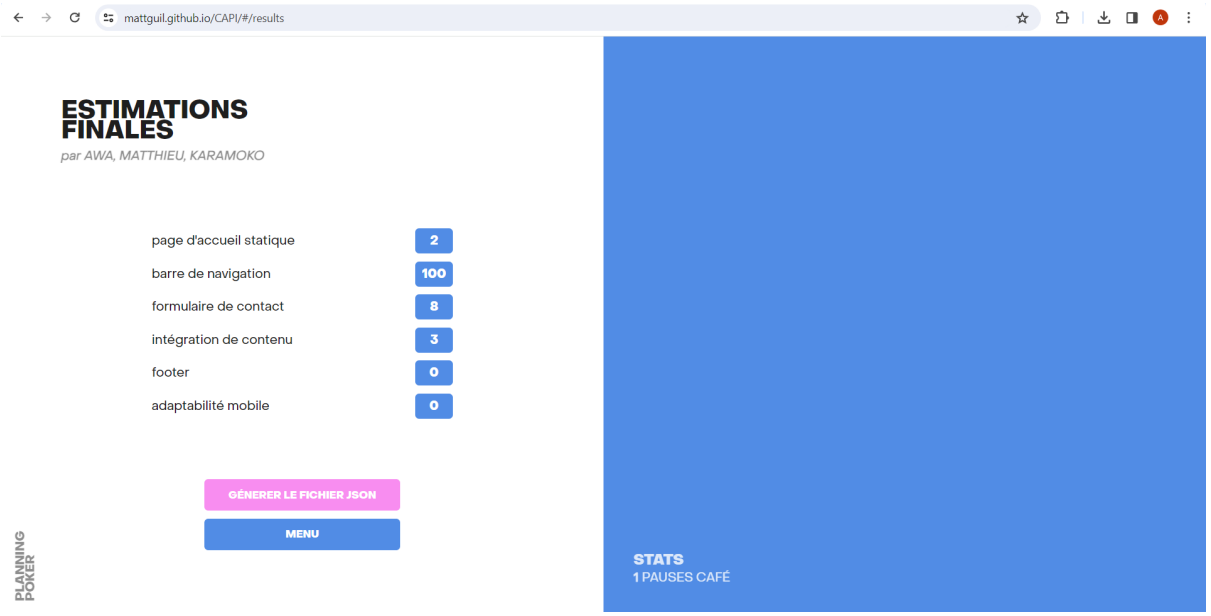


Figure 12 : Page RESULTS

Cette page présente les résultats des différents votes une fois que la partie est terminée : les joueurs sont parvenus à un consensus (suivant le mode de jeu) pour chacun des backlogs. C'est depuis cette page qu'on peut retourner au menu, et générer le fichier JSON final, contenant la liste des backlogs associés à leurs estimations. Quelques statistiques "optionnelles" sont également visibles dans la partie droite de l'écran.

VI. Intégration continue

L'intégration continue de notre projet est garantie grâce à un fichier main.yml présent dans le dossier .github/worflows à la racine du projet/dépôt GitHub.

Les instructions de ce fichier sont exécutées à chaque push sur la branche main.

```
on:
  push:
      branches:
            - main
```

Une série de `jobs` permet alors la réalisation automatique des tâches suivantes.

1. Génération de la documentation (`document`)

La documentation de toutes les classes JavaScript est générée grâce au langage de balisage JSDoc, et stockée dans un dossier docs/.

```
jsdoc -r ./src/classes/ -d ./docs
```

2. Tests unitaires (`test`)

Les tests unitaires sont une pratique de développement logiciel qui consiste à tester individuellement chaque composant (unité) d'un programme de manière isolée. L'objectif principal des tests unitaires est de vérifier que chaque unité du logiciel fonctionne comme prévu. Une unité peut être une fonction, une méthode, une classe ou même un module, selon le niveau de granularité que vous choisissez.

Pour effectuer des tests unitaires dans le contexte de notre projet, nous allons utiliser un outil de test approprié pour le framework que nous utilisons : *Vitest*. Il va nous permettre, avec la commande `vitest --dom`, de lancer les tests écrits avec la librairie *Vue Test Utils* dans tous les fichiers *.spec.js de notre projet. En l'occurrence, notre fichier index.spec.js est rangé dans ./tests/unit

Ci-dessous un bout de code pour vous expliquer comment nous avons effectué les tests dans le cadre de notre projet :

```
describe('Menu.vue', () => {
	let wrapper;

	beforeEach(async () => {
		const app = createApp(Menu);
		app.use(store);

		wrapper = await mount(app);
	});

	it('montée avec succès.', () => {
	expect(wrapper).toBeTruthy();
	});
});
```

La fonction `describe` prend deux arguments : une description textuelle de la suite de tests (dans ce cas, "Menu.vue") et une fonction de rappel qui contient les tests spécifiques. Elle utilise *Vue Test Utils* pour créer une instance du composant, monté le composant, puis utilise une assertion *Vitest* pour vérifier si le `wrapper` (l'instance montée du composant) existe. Si le `wrapper` existe, le test réussit, ce qui signifie que le composant peut être monté avec succès.

3. Déploiement sur GitHub Pages (`deploy`)

Enfin, si tous les `jobs` précédents se sont déroulés avec succès, l'application est déployée sur GitHub Pages, permettant ainsi à n'importe qui d'y accéder facilement en ligne à l'adresse : https://mattguil.github.io/CAPI/

VII. Problèmes rencontrés

Pendant la phase de développement du projet, plusieurs défis ont émergé, impactant la progression du travail. Parmi les principaux problèmes rencontrés, on peut citer les problèmes de gestion de versions et le déploiement de l'application finale sur GitHub Pages. En effet, lors de cette étape phare pour rendre le projet accessible en ligne, le contenu dynamique de la page n'était pas rendu. Cela était en fait dû à un problème de configuration du Vue-Router. Le mode à utiliser était "hash", et non "history".

Malgré les obstacles rencontrés, nous sommes fiers du résultat obtenu et des compétences que nous avons acquises tout au long de ce projet.

Conclusion

À la clôture de cette passionnante aventure qu'a été la création de notre site web dédié au Planning Poker, il est gratifiant de contempler le chemin parcouru. Ce projet, fusion audacieuse entre technologie et méthodologie agile, a ouvert de nouvelles perspectives dans notre approche de la planification collaborative.

Nous sommes parvenus à créer une plateforme web qui permet, au-delà des avantages du planning poker pour un projet, une visualisation graphique de l'avancement de son avancement. Notre site web de planning poker, CAPI, est bien plus qu'un simple outil. Nous avons surmonté des défis techniques, peaufiné des détails minutieux, et intégré des fonctionnalités qui visent à enrichir notre expérience de planification, à la rendre plus interactive et dynamique. On espère qu'il vous plaira autant qu'à nous.