Projet CAPI

Conception agile de projets informatiques

SOURAYA Ahmed Abderemane 5201790
TEL-TELLAS Alyson 2239163

Master 1 : Informatique
Université Lyon Lumière
2023-2023

Table des matières

Spécification des besoins 2
Fonctionnels 2
Non-fonctionnels 3
Analyse & conception 3
L'implémentation 4
Langages de développement 4
Structure de CAPI 4
Design pattern 6
Patrons architecturaux 6
Intégration continue 7
Tests unitaires 7
Documentation 8
Annexes 9
Annexe 1 : Maquettes FIGMA 9
Annexe 2 : Diagramme de classe base de données 11
Annexe 3 : Architecture du site 11

Spécification des besoins

Fonctionnels

Le projet **CAPI** est la réalisation d'un Planning Poker à utiliser sur machine (ordinateur, téléphone, etc). Ainsi, il doit permettre à une équipe de réaliser l'estimation du coût concernant la difficulté d'une tâche.

Le Planning Poker utilise un jeu de cartes, ici virtuel, suivant une suite de Fibonacci modifiée et deux cartes spéciales : "interrogation" et "café".

Chaque joueur choisit individuellement une carte pour l'estimation de la difficulté d'une tâche. Puis toutes les cartes sont présentées à l'ensemble des joueurs. Seuls les extremums peuvent débattre. Suite à ce débat, tous les joueurs votent une seconde fois l'estimation de cette tâche. Le processus est répété pour chaque tâche jusqu'à que l'estimation d'une tâche soit obtenue par l'unanimité. Ensuite, les joueurs pourront passer à l'estimation de la seconde tâche. Ainsi de suite, jusqu'à ce que toutes les tâches soient traitées. À noter que si tous les joueurs choisissent la carte café, une pause sera effectuée. Si tous les joueurs choisissent la carte interrogation, la tâche sera votée à nouveau.

Le mode de jeu décrit est celui du mode strict. Il existe aussi différentes variantes, qui se caractérisent par une attribution de l'estimation différente. Ici, nous allons utiliser le mode strict et la majorité relative. Pour la majorité relative, seul le premier tour de vote pour chaque tâche se joue sur les extrêmes pour avoir au moins un temps d'échange. À partir du second tour de vote, cela se base sur la majorité relative, c'est-à-dire, la carte avec l'occurrence la plus élevée. S'il n'y a pas de majorité relative, l'estimation de la tâche est jouée de nouveau. La carte "interrogation" et la carte "café" ont le même effet que pour le mode strict.

Ainsi, nous avons pu définir différentes fonctionnalités pour **CAPI** :

- L'authentification grâce à un pseudo-unique.
    - La connexion.
        - La création d'une nouvelle partie.
            - Le choix du nom du projet.
            - Le choix du nombre de membres dans l'équipe.
            - La création des pseudos unique des joueurs.
            - Le choix du mode jeux "strict" ou "moyenne relative".
            - La création d'un backlog avec des taches toutes différentes les unes par rapport aux autres.
            - Le vote de l'estimation à partir d'un jeu de cartes virtuel.
            - Un listing des tâches déjà évaluées.
            - Un listing des tâches restantes à évaluer.
            - Un chronomètre pour chaque vote et temps de discussions.
            - Enregistrement de toutes les données d'une partie en cours.
            - La possibilité de faire une pause si tous les membres choisissent la carte café.
            - Téléchargements du backlog final en JSON.
        - La reprise d'une partie en cours ultérieurement.
            - Récupération des données enregistrées d'une partie en cours.
    - La déconnexion.

Non-fonctionnels

Le projet **CAPI** présenté ici à une portabilité local. Nous avons fait ce choix, car même si le télétravail est en grand essor, les réunions en présentiel restent très présentes. Ainsi, l'utilisateur pourrait par exemple être un chef de projet ou une personne devant gérer de nombreux projets à la fois.De ce fait, avoir un contenu local peut-être intéressant.
Nous n'avons pas travaillé sur les questions de sûreté pour ce projet.

Analyse & conception

Pour commencer, nous avons déjà eu une première expérience en testant le Planning Poker avec un jeu de cartes réel. Cela nous a permis d'avoir une première idée de comment fonctionne le Planning Poker. Pour commencer le projet, nous avons tout d'abord été voir ce qui se faisait déjà pour avoir une première idée de ce à quoi pourrait ressembler un Planning Poker sur machine.

Ensuite, nous avons réalisé des maquettes et modèles UML ainsi qu'un premier fichier JavaScript rassemblant les différentes fonctionnalités que nous souhaitions développer sous forme de fonction vide pour débuter. (cf. Annexe 1).

Dès le départ, nous avons souhaité ajouter un chronomètre pour limiter le temps lors des tours de vote. Vers la fin du projet, nous avons aussi décidé d'en rajouter un pour contraindre les temps de discussion. Avoir un chronomètre s'avère être un plus, car les réunions ne sont pas illimitées, il faut donc savoir gérer son temps.

Nous avons ensuite souhaité réaliser en plus un listing des tâches validées et des tâches restantes. Permettant de visualiser l'avancement et l'estimation des tâches.

Étant donné que nous utilisons une base de données, la question de l'authentification, c'est posée afin que chaque utilisateur puisse voir uniquement les parties. Nous l'avons donc implémenté.

Nous avons aussi voulu ajouter un tutoriel dans ***Aide***, pour ce faire, nous avons réalisé une vidéo démonstrative.

Nous aurions aimé implémenter d'autres fonctionnalités, si nous avions plus de temps :

- L'utilisation d'un tableau Kanban pour les parties finies.
- La mise en place d'éléments sonores pour les chronomètres.
- L'enregistrement de la partie même si un des joueurs clique sur profil ou planning poker (accueil).
- La possibilité de télécharger le backlog final sous la forme d'un PDF.
- La possibilité de mettre en couleur les joueurs qui ont joué le max et le min pour les distinguer des autres joueurs.
- Ajouter une page si aucun profil n'est connecté.
- Une meilleure gestion des erreurs pour le formulaire. Actuellement, nous pouvons quand même passer au champ suivant même si celui-ci est vide. Le message d'erreur ne sera affiché que si arrivé au backlog l'un des champs est vide. Nous aurions voulu bloquer l'accès aux champs tant qu'un champ n'est pas rempli.
- Si l'estimation retenue pour une tâche est 100, proposer à l'équipe de la découper en sous-tâches et de les insérer dans le backlog. Il suffit de supprimer la tâche à 100 et la modifier en ajoutant d'autres tâches, ainsi, on aurait pu utiliser un modèle **CRUD** pour le backlog. Ou créer un nouveau backlog fils connecté aux backlog père.
- Rendre le site responsive.

Pour faciliter la gestion de notre projet, nous avons intégré divers outils tout au long du processus de développement. Nous avons opté pour **GitHub** et **GitHub Desktop** comme gestionnaires de version, ce qui a grandement simplifié le contrôle des versions et la collaboration au sein de l'équipe. La documentation a été gérée efficacement avec l'utilisation de **JsDoc**, assurant une documentation claire et complète de notre code source.

Pour le maquettage de l'interface utilisateur, nous avons choisi **FIGMA** (cf. Annexe 1), un outil puissant qui nous a permis de créer des maquettes interactives et de collaborer efficacement sur le design. Pour la modélisation **UML**, **Draw.io** a été notre choix, fournissant une interface intuitive pour représenter visuellement l'architecture du système (cf. Annexe 2).

En ce qui concerne les tests unitaires, nous avons utilisé **JEST**, un framework de test JavaScript populaire, assurant la robustesse et la fiabilité de notre code. Enfin, pour explorer et concevoir l'architecture d'utilisation du site, nous avons utilisé Figma Jam, ajoutant une couche collaborative et créative à notre processus de planification.

L’implémentation

Langages de développement

Le choix des technologies web pour le développement de notre application a été motivé par la nécessité de concilier nos compétences, provenant de filières différentes, en matière de langages de programmation. Opter pour le développement web nous a offert un compromis idéal. De plus, le contexte du projet, axé sur la conception d'une application de poker, nous a conduits vers le web en raison des vastes possibilités qu'il offre en termes de design d'interface.

Initialement, nous avons débuté avec du développement web natif en utilisant les langages JavaScript, HTML et CSS. Cependant, pour répondre aux exigences spécifiques du projet, nous avons fait le choix d'intégrer des frameworks, en particulier Laravel, pour travailler avec PHP. Cette décision a été motivée par la gestion simplifiée de notre base de données avec **phpMyAdmin** et l'utilisation de bibliothèques existantes pour le traitement des formulaires. Néanmoins, la mise en place de **Laravel** a présenté des défis persistants tout au long du projet, entravant les tests et l'implémentation, car une machine pouvait exécuter le code Laravel correctement, tandis que l'autre devait se contenter de l'ancienne version. Nous avons réussi à pallier cela en utilisant le pair programming.

Structure de CAPI

Le projet, étant basé sur le framework Laravel, il présente une structure complexe, mais bien organisée. Parmi ses éléments clés, on retrouve une base de données gérée par **phpMyAdmin**. Pour chaque partie du Planning Poker, la base de données comprend plusieurs informations cruciales : un identifiant unique, le nom de l'hôte, l'état actuel de la partie, son format JSON, ainsi que toutes les autres données nécessaires pour créer et reprendre une partie. Ces informations sont soigneusement structurées pour garantir une gestion efficace des sessions de Planning Poker. (cf. Annexe 3).

copiepoker.js**,** qui récupère les variables du formulaire ***affichagemneu.js*** et les rend globales afin de pouvoir les utiliser dans le jeu. Ce code initialise le jeu en convertissant les données du backlog et des joueurs en objets. Il établit l'ordre des joueurs, et initialise diverses variables nécessaires au déroulement du jeu. Ce code gère tout ce qui concerne le jeu. Par conséquent, il implémente plusieurs fonctionnalités, parmi elles, on retrouve par exemple la gestion du tour des joueurs, du vote, de l’estimation d’une tâche, etc. Il utilise également des fonctionnalités de chronomètre pour définir des

limites de temps lors du vote des joueurs et des phases de discussion. Enfin, des fonctions d'enregistrement et de téléchargement des données de la partie sont intégrées, offrant une expérience utilisateur complète. De plus, il intègre des éléments d'interface utilisateur, tels que des boutons pour passer à la tâche suivante ou relancer le vote, le listing des tâches en cours et validées.

affichagemenu.js, qui implémente un système de récupération des données à partir d'un formulaire en trois parties. Ce formulaire permet à l'utilisateur de spécifier les informations nécessaires pour une partie de Planning Poker. Soit le pseudo de l'utilisateur, le nom du projet, le nombre de joueurs, la liste des noms de joueurs, le backlog, et la règle de jeu sélectionnée. Il utilise des sélecteurs pour récupérer les divers éléments du formulaire cité plus haut. De plus, il génère dynamiquement en fonction du nombre de joueurs des champs de saisie pour la création de leurs pseudos. Le code inclut également des fonctionnalités de gestion d'affichage pour différentes parties du formulaire. Donc, elles seront affichées l'une après l'autre. Ainsi, que la visualisation de pop-up pour la description des règles en fonction du mode de jeu choisi.

De plus, un nombre considérable de feuilles de style CSS ont été nécessaires pour notre projet. Pour chaque page, nous avons créé un fichier CSS dédié, intégrant déjà les variables de la palette de couleurs que nous avions choisie. Toutes ces variables ont été centralisées dans le fichier ***root.css***. Par la suite, nous avons inclus ce fichier dans l'ensemble de nos différentes feuilles de style CSS pour assurer une cohérence visuelle. Notamment, le fichier layout.css contient les styles définis pour la mise en page générale, qui s'applique à nos différentes pages, notamment à travers le fichier ***app.blade.php***. Ce fichier fixe la structure fondamentale de l'interface utilisateur que nous avons élaborée.

Dans le dossier ***asset*** ont retrouve toutes les animations et images.

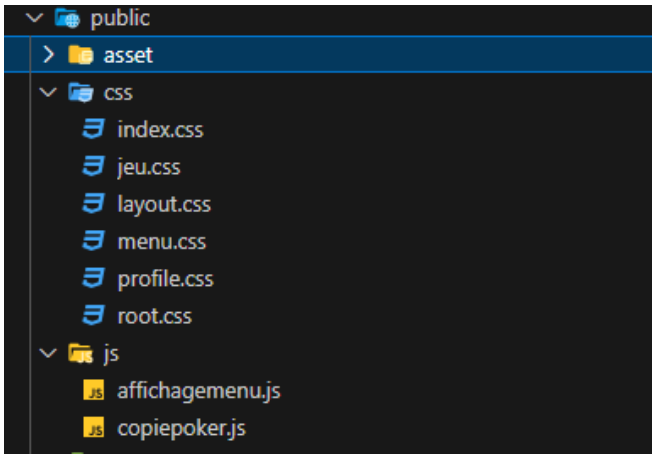


Image 1 : Structure des fichiers 1/3

Dans le fichier ***web.php***, toutes les routes essentielles ont été définies pour assurer une navigation fluide de page en page au sein de notre application Planning Poker.

Le Planning Poker se compose de quatre vues principales: ***menu.blade.php***, ***profile.blade.php***, ***jeu.blade.php*** *et* ***index.blade.php***. Ces vues sont soutenues par deux classes indispensables, ***Partie.php*** et ***User.php***, qui agissent en tant que modèles représentant les données. Chacune de ces classes est associée à un contrôleur dédié. Ces trois répertoires, à savoir ***views***, ***models***, et

controllers, s'organisent conformément à l'architecture Modèle-Vue-Contrôleur (MVC), comme illustré en page 7 de notre documentation. Cette structure permet une séparation claire des préoccupations, facilitant ainsi la maintenance et l'évolutivité de notre application.

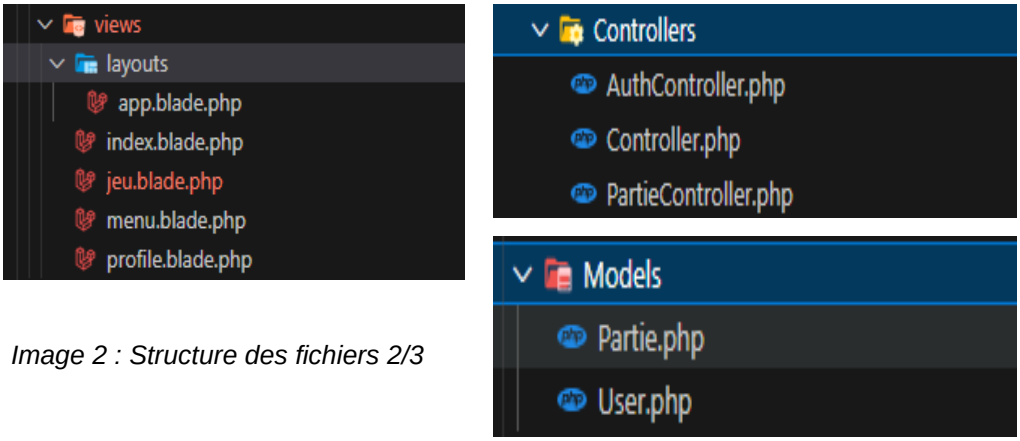


Image 2 : Structure des fichiers 2/3

Design pattern

Le design pattern **Strategy** est intégré de manière robuste dans le code à travers la classe ***EstimationStrategy*** et ses sous-classes, notamment ***StrictEstimationStrategy*** et ***MajorityEstimationStrategy*****.** Ce pattern offre la possibilité de définir des sous-classes interchangeables, permettant ainsi une grande flexibilité dans le choix de la stratégie d'estimation en fonction du mode de jeu.

La classe mère, ***EstimationStrategy*****,** déclare une méthode commune, ***estimate***, prenant en paramètre le mode de jeu sélectionné. Cette méthode sert d'interface unifiée pour tous les modes de jeu possibles. Les classes filles héritant de ***EstimationStrategy*** fournissent leur propre implémentation de la méthode ***estimate***, adaptée à la logique spécifique de chaque stratégie.

L'avantage majeur de ce design pattern réside dans sa capacité à offrir une extensibilité sans perturber le code existant. Ainsi, de nouvelles règles de jeu ou stratégies d'estimation peuvent être ajoutées sans nécessiter de modifications significatives du code existant. Cela facilite l'adaptation du Planning Poker à d'éventuels changements dans les règles du jeu et permet l'intégration aisée de nouvelles approches d'estimation. En résumé, le design pattern **Strategy** favorise une conception souple et évolutive du système.

Patrons architecturaux

Durant ce projet, deux “Modèles Vue Contrôleur” (MVC) ont été implémentés. Un pour le jeu, l’autre pour l'hôte de la partie.

De manière générale, dans ce projet, les “Vues” de la partie sont affichées dynamiquement à l'aide de balises Blade utilisées pour générer les vues HTML. Ces fichiers incluent des balises PHP qui intègrent les données provenant du contrôleur, permettant ainsi la personnalisation des pages en fonction des besoins de l'utilisateur, créant ainsi une interface utilisateur interactive.
Et le "Contrôleur" gère la logique métier. Il reçoit les requêtes HTTP, valide les données, interagit avec le modèle, et retourne les réponses appropriées.

Pour commencer, nous allons regarder de plus près le MVC concernant le jeu.

La “Vue” est représentée par le fichier ***jeu.blade.php***. Elle est la représentation visuelle de l'interface utilisateur du jeu. Lorsqu'un utilisateur accède à cette page, elle affiche différentes sections en fonction de l'état actuel du jeu. La première section présente des informations sur le projet, telles que le nom du projet et la tâche en cours de débat. En dessous, il y a une série de cartes utilisées pour voter sur la complexité des tâches. En fonction des actions des joueurs et des règles du jeu, différentes sections sont activées ou désactivées dynamiquement. La vue affiche également la liste des joueurs, le chronomètre du jeu, et le backlog des tâches restantes et validées. Les boutons interactifs permettent aux joueurs de revoter, de passer à la tâche suivante, ou de quitter la partie.

Le "Contrôleur" est représenté par le fichier ***PartieController.php***. Ses fonctions clés sont :

- ***creerPartie(Request $request)*** qui est appelée lorsqu'un utilisateur souhaite créer une nouvelle partie. Elle valide les données du formulaire, crée une nouvelle instance de la classe ***Partie***, initialise ses propriétés en fonction des données fournies, applique la logique métier spécifique à la règle choisie, puis redirige vers la vue du jeu.
- ***afficherPageJeu()*** qui récupère les données de la dernière partie enregistrée dans la base de données et les passe à la vue du jeu. Elle est utilisée pour afficher la page de jeu.
- ***enregistrerPartie(Request $request)*** qui est appelée pour mettre à jour les données d'une partie. Elle récupère la partie à enregistrer, puis met à jour ses propriétés en fonction des données fournies.
- ***telechargerBacklog($partieId)*** qui génère un fichier JSON contenant les données du backlog et le nom du projet pour le téléchargement.

Enfin, le "Modèle" est représenté par la classe ***Partie*** dans le fichier ***Partie.php***. Les méthodes définies dans le modèle facilitent l'enregistrement, la mise à jour et la suppression des données liées à une partie.

Maintenant, nous allons nous intéresser au MVC de l'utilisateur. La particularité de ce MCV est qu’en fonction de la situation de l'utilisateur, l’interface ne sera pas la même. Si un utilisateur est authentifié, la “Vue” représente par ***profile.blade.php*** affiche : les parties en cours, qu’il peut reprendre et les parties terminées où il peut télécharger le backlog. Sinon, la “Vue” représentée par ***menu.blade.php*** lui permet de créer une nouvelle partie.

Le “Contrôleur” est représenté par le fichier ***AuthController.php***. Ce dernier gère l'inscription, la connexion et la déconnexion des utilisateurs. Il vérifie si un utilisateur avec le même pseudo-utilisateur existe déjà dans la base de données. S'il existe, l'utilisateur est connecté, sinon un nouvel utilisateur est créé.

Pour finir, le "Modèle" est représenté par la classe ***User*** dans le fichier ***User.php***. Il gère les données de l'utilisateur, en particulier son pseudo unique utilisé pour son authentification.

Intégration continue

Tests unitaires

Nous avons rencontré des problèmes lors de la mise en place de l’intégration continue, au niveau des tests unitaires.La création des tests unitaire en passant par GitHub Action génère des erreurs. Finalement, nous avons réussi à les mettre en place. Pour cela, nous avons utilisé la bibliothèque JEST. Puis, nous l’avons testé avec quelques fonctions. Notamment, pour la fonction permettant d’obtenir le backlog. La majorité de nos autres fonctions n’avaient pas besoin d'être testées avec ce type de test car elles n’ont pas de paramètre et ne retournent rien. Mais tout le long du projet nous avons testés les

différentes situations possibles afin de gérer un maximum d'erreur et contrôler le bon fonctionnement de notre code.

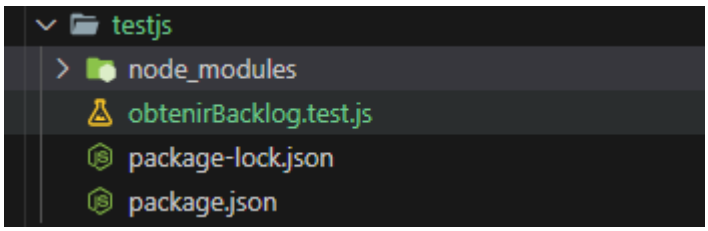


Image 3 : Structure des fichiers 3/3

Documentation

Pour la documentation, nous avons choisi d'utiliser JsDoc pour le code JavaScript et GitHub Actions avec le workflow ***documentationjs.yml*** pour automatiser le processus. Nous avons soigneusement documenté notre code en utilisant les annotations appropriées de JsDoc pour détailler les différents paramètres, fonctions,et méthodes, facilitant ainsi la génération automatique de la documentation.

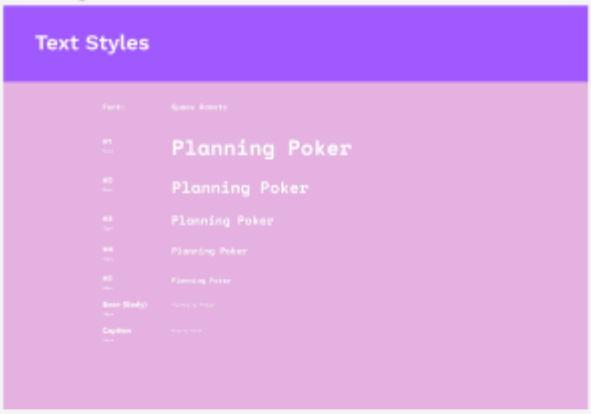
Concernant la partie PHP, nous avons opté pour l'utilisation de la bibliothèque **phpDocumentor** de Laravel pour documenter notre code PHP. Cependant, nous avons rencontré des difficultés lors de l'installation de cette bibliothèque, ce qui a malheureusement entravé la génération automatique de la documentation PHP.

.

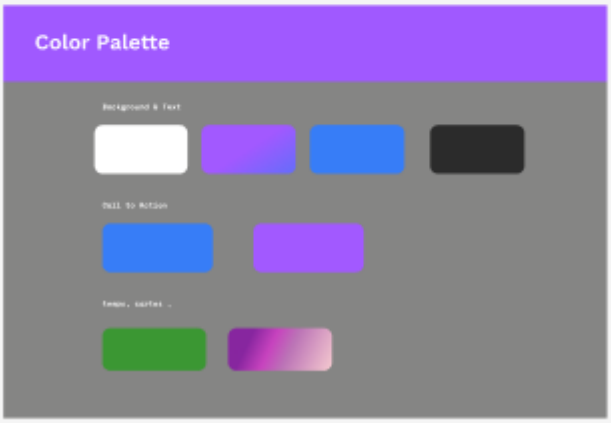
Annexes

Annexe 1 : Maquettes FIGMA

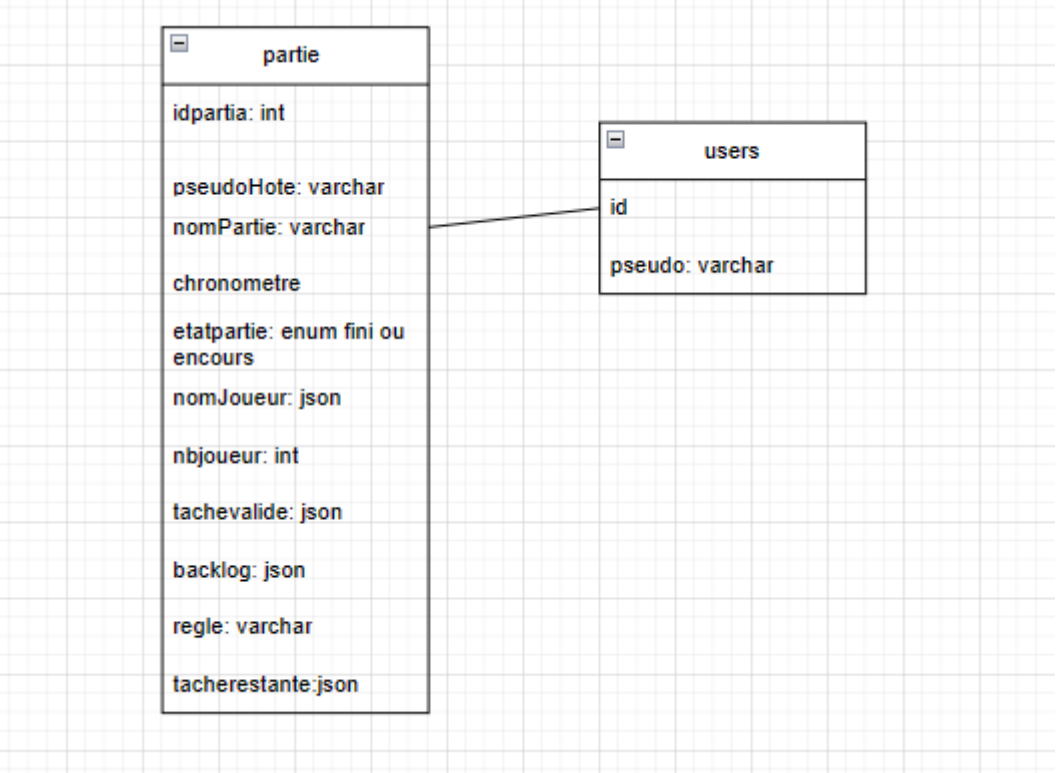
Text Styles



Color Palette



Annexe 2 : Diagramme de classe base de données



Annexe 3 : Architecture du site