ChessJS

Maxime Borgeaud

CPNV, pré-TPI 2024

Table des matières

1 Introduction

Le projet consiste en un jeu d'échec en réseau basé sur un environnement d'exécution côté serveur basé sur JavaScript, sera utilisé pour gérer les interactions en temps réel entre les joueurs connectés à notre plateforme. Ce projet représente une opportunité passionnante de combiner les technologies web modernes avec le jeu classique d'échecs, offrant ainsi une nouvelle dimension à l'une des plus anciennes formes de divertissement intellectuel.

1.1 Objectifs

Le premier objectif est d'avoir un jeu d'échec fonctionnel. Le déplacement des pièces, la détection des « échecs » & « échecs et mat » doivent se faire selon les règles. Il y a certaines règles que je n'implémenterais pas, notamment le pat, le petit et grand roque ainsi que le changement de grade du pion lorsqu'il atteint le bord de l'adversaire. Plutôt que d'implémenter ces petites règles, j'ai préféré me concentrer sur le deuxième objectif décrit ci-dessous.

Le second objectif est la gestion en temps réel du déplacement des pièces. Je profite de ce projet pour apprendre les bases de la manipulation des « WebSocket » avec NodeJS. Le serveur doit gérer l'affichage de la partie en cours en temps réel. Les 2 joueurs doivent avoir leurs pièces de leurs coté et donc les calculs de déplacements doivent être adaptés également. Lorsqu'un joueur déplace une pièce, le déplacement se fait en temps réel sur l'écran de tous les joueurs connectés à la partie.

Un troisième objectif est de push en production, d'abord en local puis sur le serveur de swisscenter.

2 Analyse

2.1 Fonctionnalités prévues

Voici une description des fonctionnalités prévues pour ce projet :

- **Gestion des utilisateurs :**
  Les utilisateurs pourront créer un compte lié à un « username » et un mot de passe, puis se déconnecter et se reconnecter plus tard.
- **Création de parties d'Echec :**
  Les utilisateurs pourront créer autant de parties qu'ils le souhaitent. Le créateur d'une partie doit sélectionner un adversaire valide (un utilisateur qui a créé un compte), puis choisir sa propre couleur.
- **Jouer aux Echec en réseaux :**
  Lorsqu'un utilisateur est dans la partie, il peut voir si l'adversaire est connecté dans la partie ou non grâce à un cercle qui est vert si connecté et rouge dans le cas contraire. Lorsque c'est son tour de jouer, le joueur peut sélectionner une de ses pièces. À ce moment, si les conditions sont réunies, ses déplacements possibles s'affichent puis le joueur peut déplacer la pièce s'il clique sur une case possible.
  A chaque déplacements l'état de la partie et sauvegarder afin que chaque joueur puisse quitter la partie et revenir à tout moment sans que cela génère des bugs.

⇨ **Fin de partie :**

Le jeu détecte automatiquement si un « Echec & mat » est produit, cela met fin immédiatement à la partie et la détermination du perdant et du gagnant est enregistrée. Les joueurs peuvent aussi déclencher manuellement la fin de la partie (les 2 joueurs doivent valider), cela met fin à la partie et une égalité est définie et enregistrée.

2.2 Méthode de projet

J'utilise la méthode Agile. J'ai défini 5 sprints de deux semaines dans lesquels j'ai mis les tâches à faire, qui sont détaillées dans la section ci-dessous

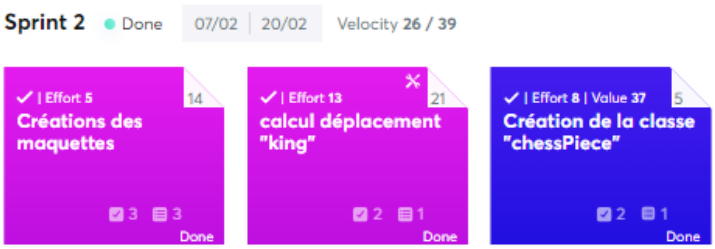
2.3 Planification

La planification se trouve dans le IceScrum du projet suivant :

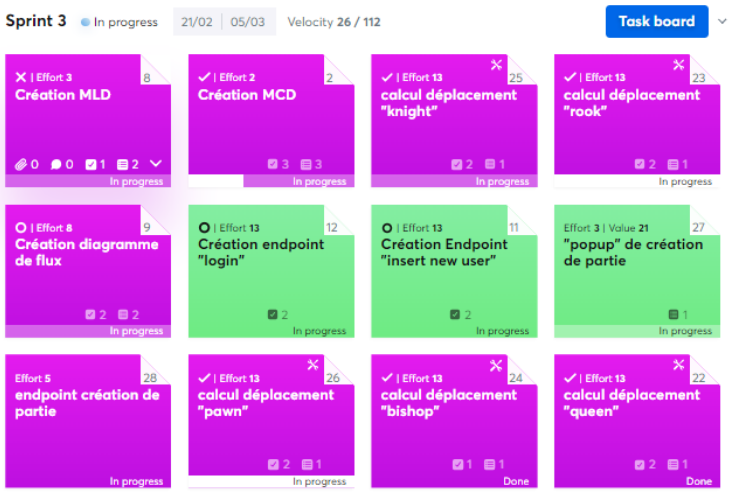
https://icescrum.cpnv.ch/p/CHESSJS/#/planning

Sprint goal : La structure de base du projet est posée

Sprint goal : Déplacements possibles des pièces visibles



Sprint goals : Affichage de tous les déplacements possibles + Les endpoints nodeJS de connexions/inscriptions sont faits



Sprint goal : Déplacement des pièces opérationnel

Sprint goal : mise en production du site

2.4 Maquettes

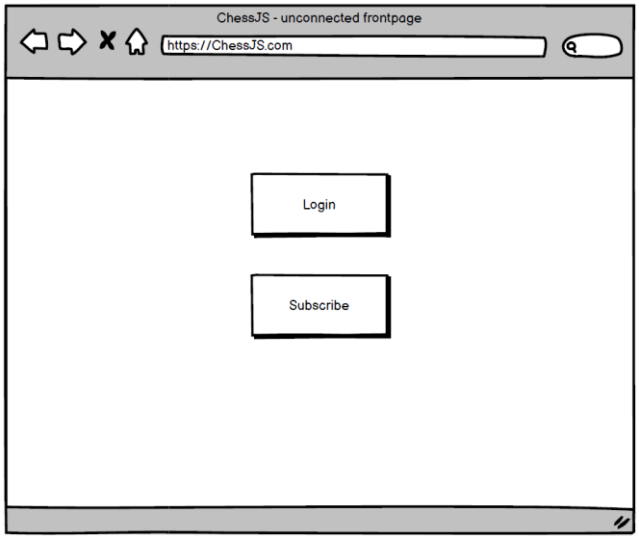


Figure 1 : page d'accueil en mode « utlisateur non-connecté »

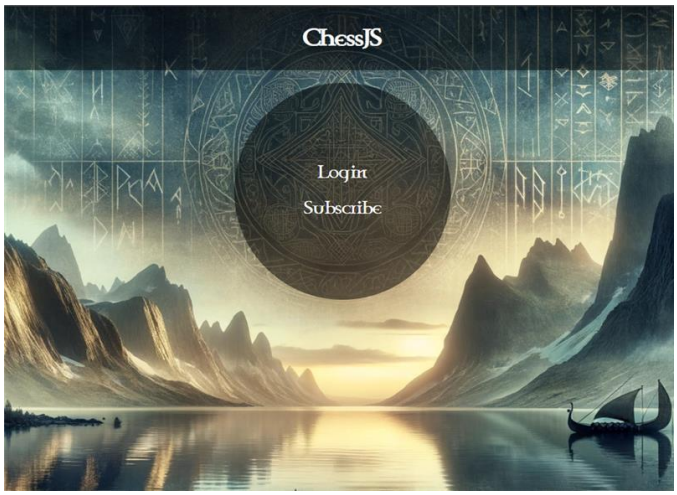


Figure 2 : page d'accueil en mode « utlisateur non-connecté »

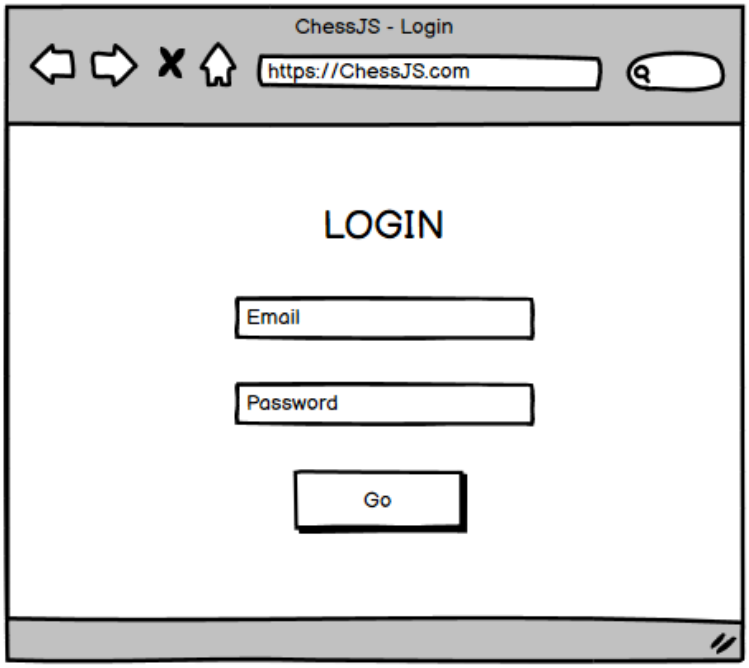


Figure 3 : page de login

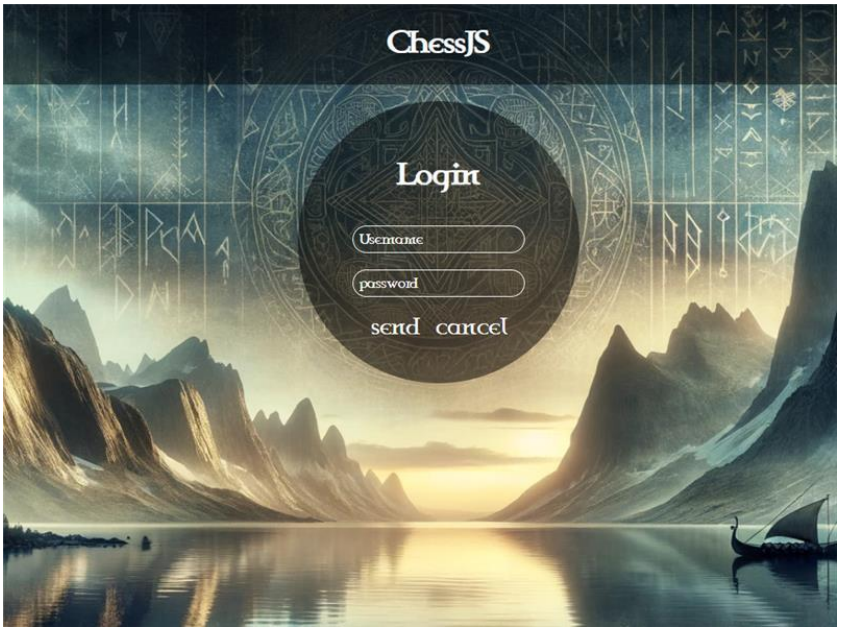


Figure 4 : page de login

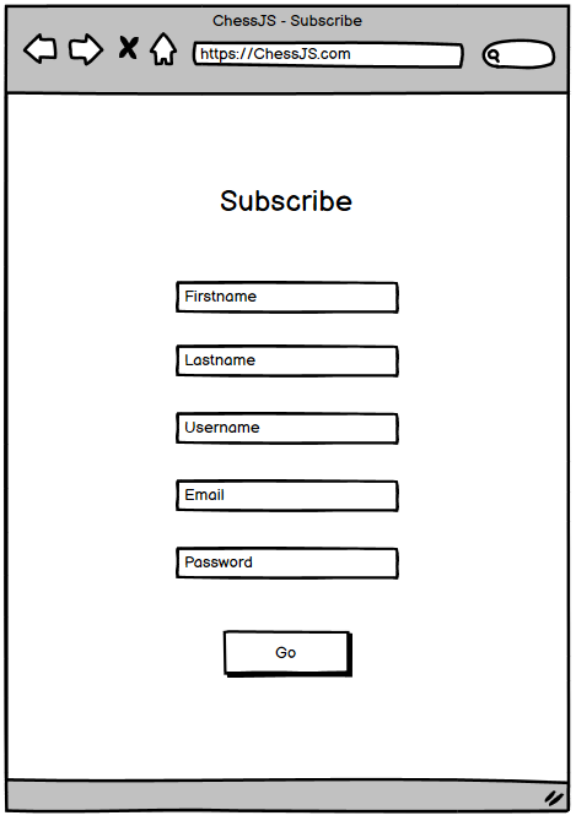


Figure 5 : page d'inscription

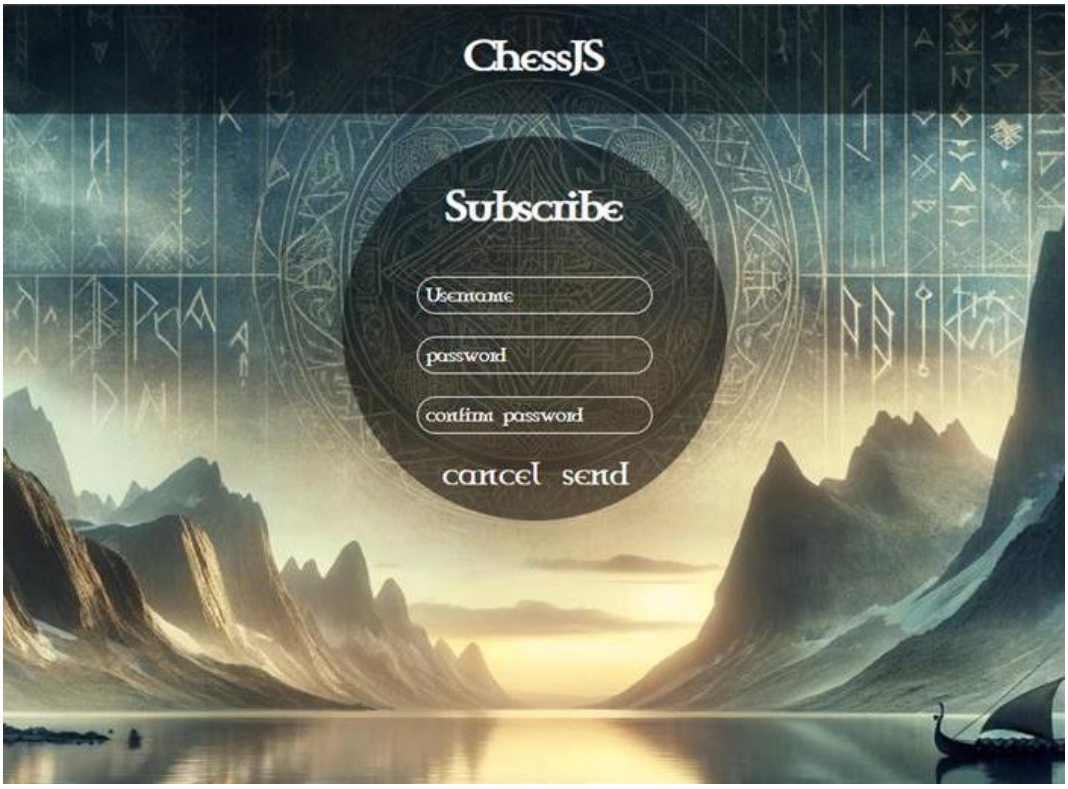


Figure 6 : page d'inscription

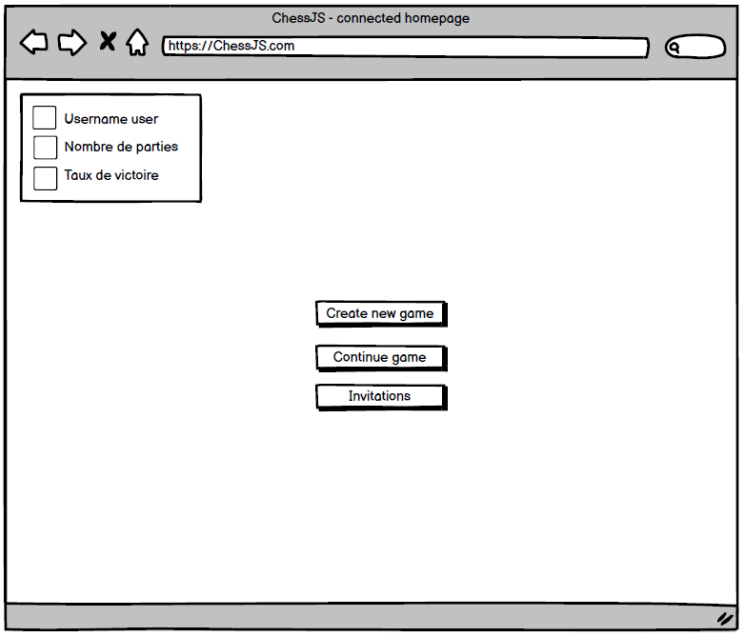


Figure 7 : Page d'accueil en mode « utilisateur connecté »

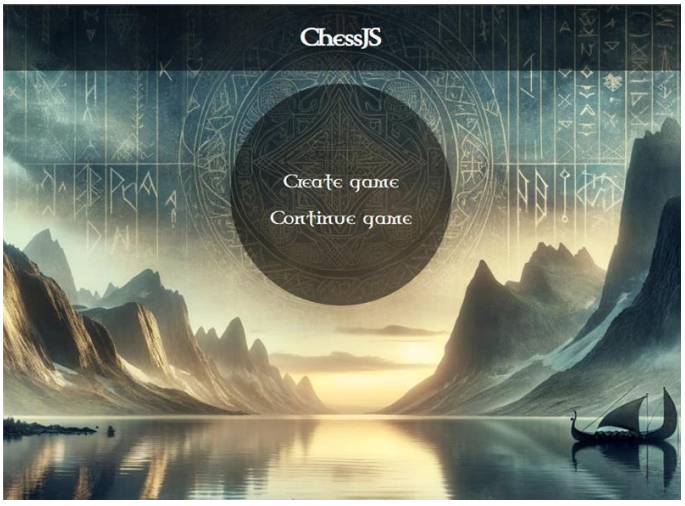


Figure 8 : Page d'accueil en mode « utilisateur connecté »

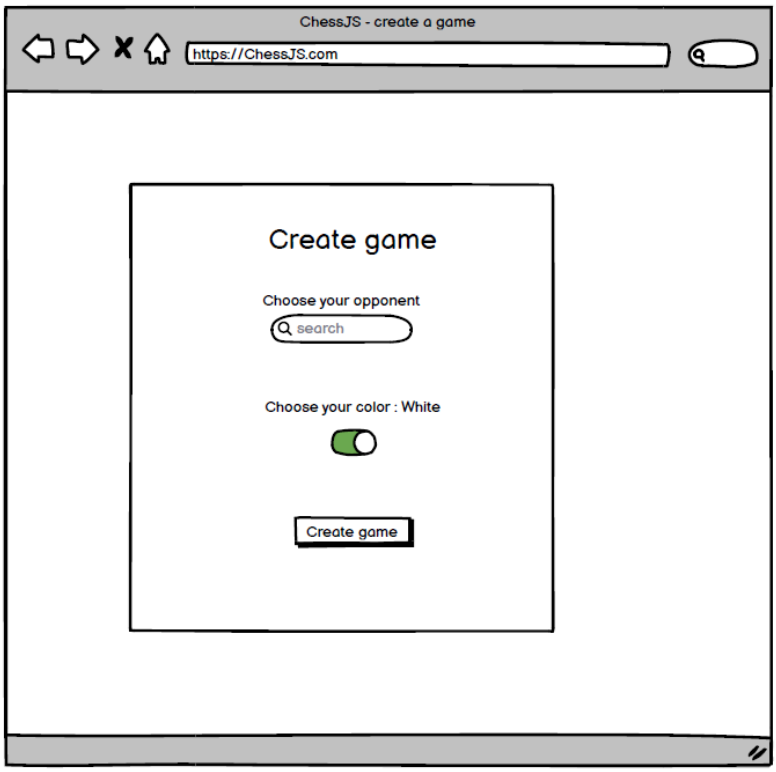


Figure 9 : page « create game »

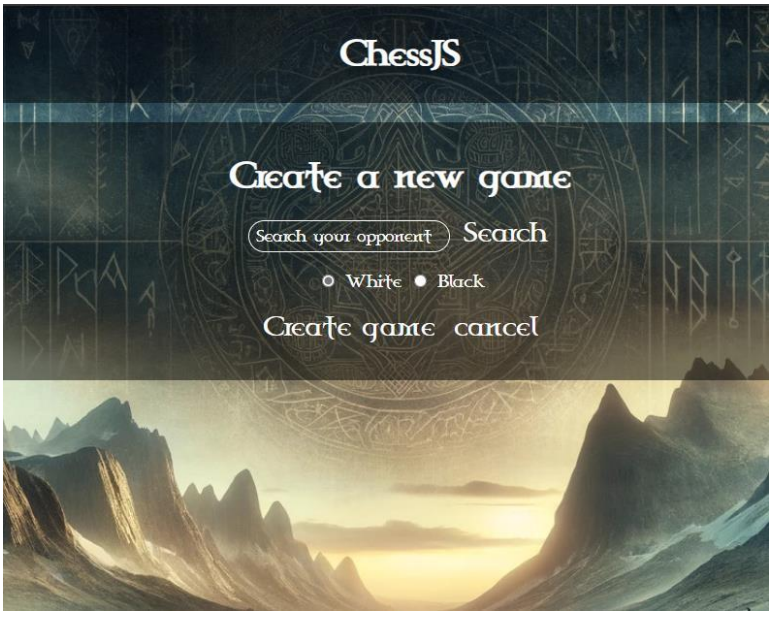


Figure 10 : page « create game »

:

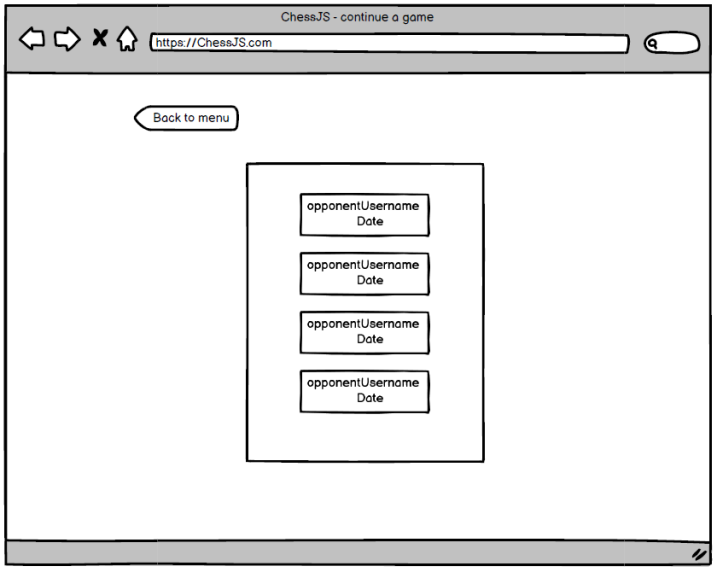


Figure 11 : Page « continue game »

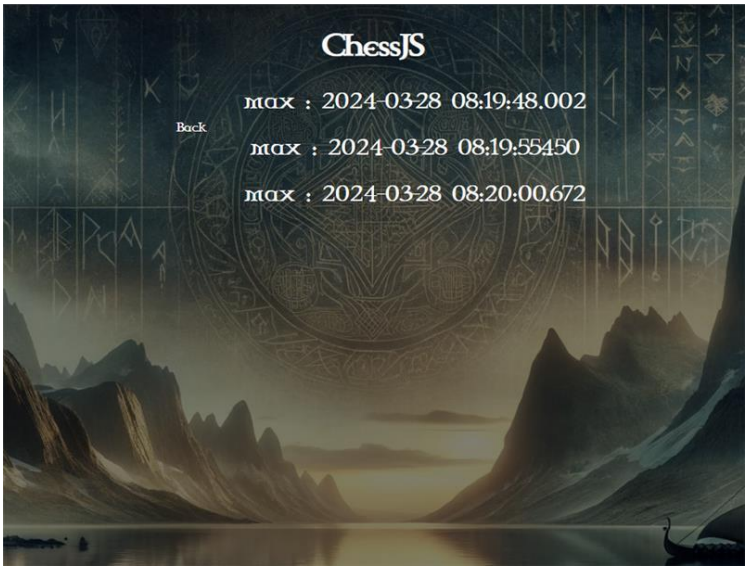


Figure 12 : Page « continue game »

:

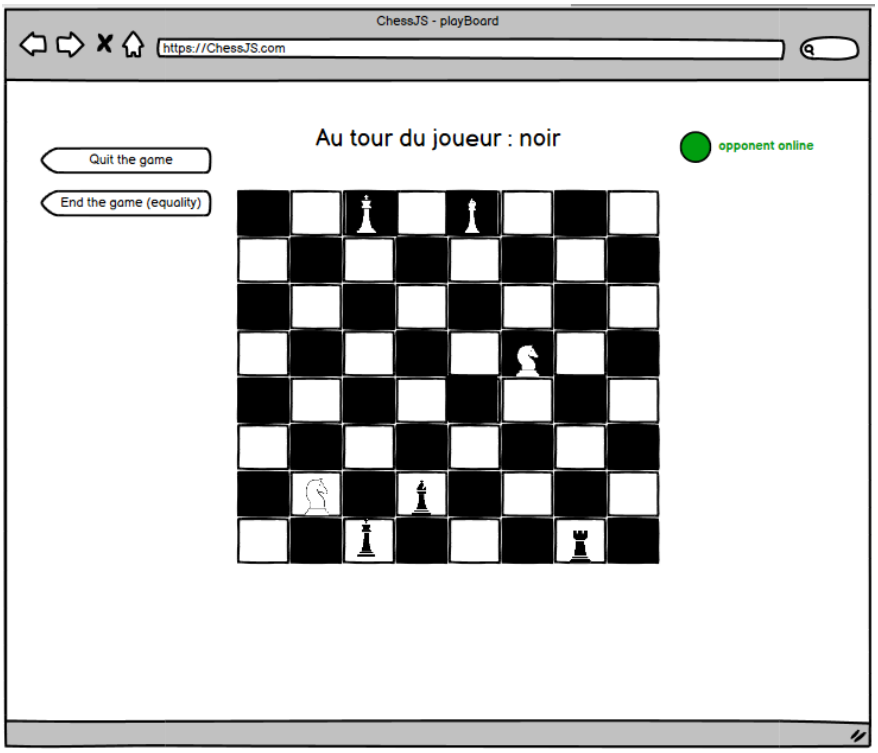


Figure 13 : Page de jeu

Figure 14 : Page de jeu

:

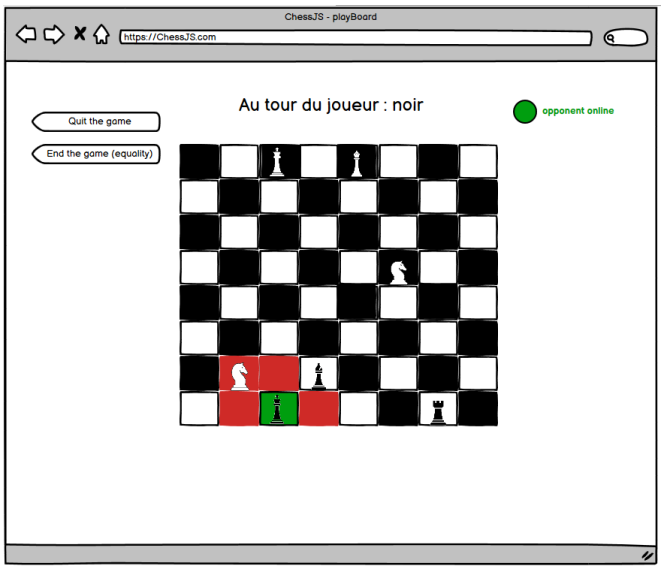


Figure 15 Page de jeu avec une pièce sélectionnée

Le joueur noir clique sur son roi pour afficher ses déplacements possibles

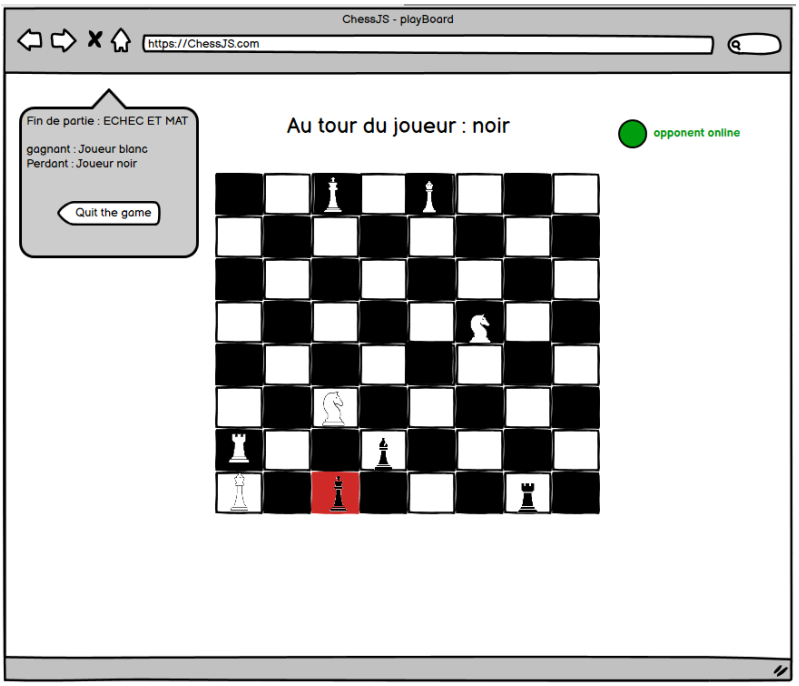


Figure 16 : Page de jeu « echec & mat »

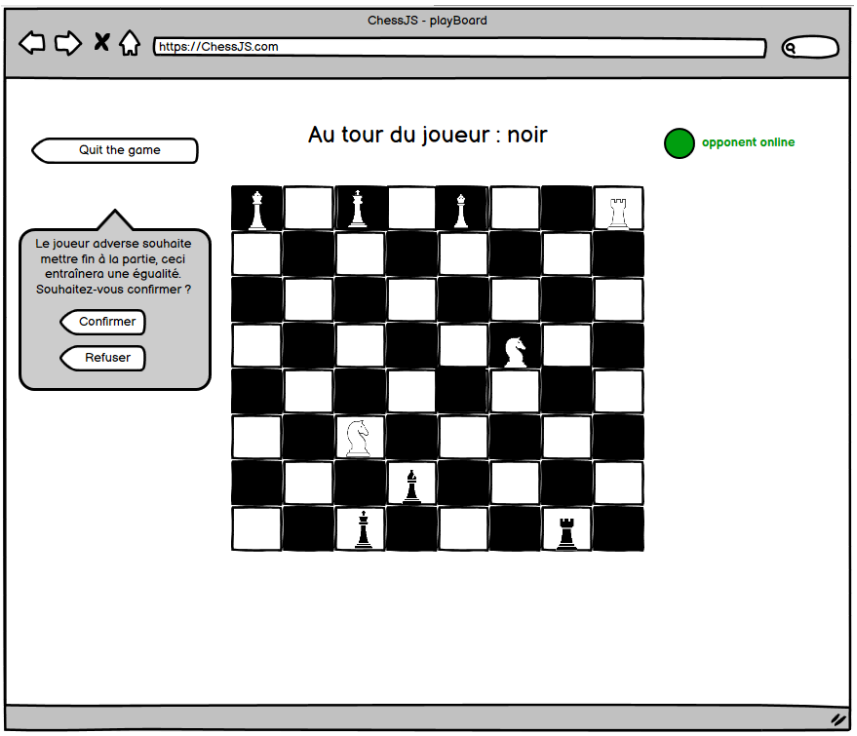


Figure 17 : Page de jeu « demande de fin de partie (égalité)

2.5 Modèle Conceptuel de données

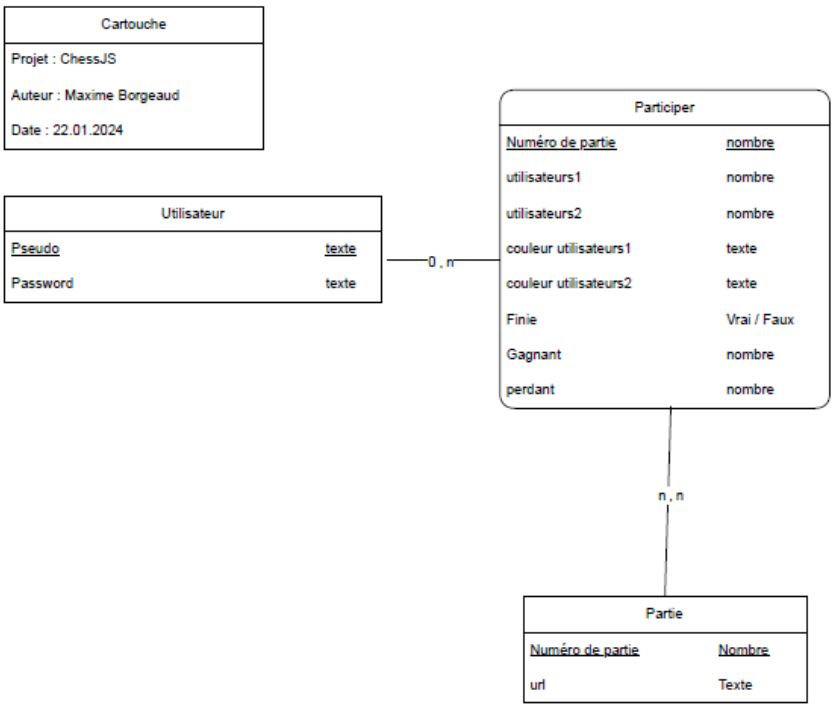


Figure 18 : MCD

2.6 Risques techniques

Le point critique du projet concerne la mise en production sur un serveur web public. Ayant très peu fait ce genre d'opérations, je ne suis pas certain que mes compétences techniques soient suffisantes pour que cela fonctionne dans le délai imparti. J'ai remarqué qu'il y avait un léger bug lors de l'accès à ma base de données qui se trouve sur SwissCenter. Lorsque je fais une requête http, de façon aléatoire, elle ne fonctionne pas du premier coup mais en réessayant cela fonctionne. J'ignore si le problème vient de mon code ou de Swisscenter (je suppose plutôt la deuxième option).

L'une des difficulté de ce point est de créer une gestion efficace et sécurisée des utilisateurs, car les règles ne sont pas les mêmes qu'en local.Lors de mon dernier projet, j'ai eu des problèmes à ce niveau car le navigateur bloquait les requêtes POST. A priori l'implémentation des tokens « JWT » devrait résoudre le problème

2.7 Use cases

Page d'accueil			
Action	**Condition**	**Scénarion echec condition**	**Scénario réussite condition**
Afficher page d'accueil en mode connecté	avoir déjà été authentifié par le server	La page d'accueil d'authentification s'affiche (login / subscribe)	La page d'accueil du jeu s'affiche en mode non-connecté
Page de login			
Action	**Condition**	**Scénarion echec condition**	**Scénario réussite condition**
Se connecter	Avoir déjà un compte avec ce username	La bordure des textbox de username & password s'affichent en rouge	La page d'accueil en mode connecté s'affiche
	Avoir écrit le mot de passe associé à ce compte		
Page d'inscription			
Action	**Condition**	**Scénarion echec condition**	**Scénario réussite condition**
S'inscrire	Ecrire un username qui n'a pas déjà été pris	Un message "username already taken" s'affiche	La page d'accueil en mode connecté s'affiche
	Les deux champs "username", "Password" & "Confirm password" doivent être remplis	la border de la textbox s'affiche en rouge	
	Les champs"password" et "confirm password" doivent être identiques		

Page de création d'une partie			
Action	**Condition**	**Scénarion echec condition**	**Scénario réussite condition**
Rechercher un advérsaire	Le username doit être présent dans la db	La border de la textbox s'affiche en rouge	Le username de l'adversaire entré s'affiche en vert en dessous de la textbox
	Le username doit être différent de celui de l'utilisateurs qui l'entre		
Créer la partie	Un advérsaire à été trouvé dans la section de recherche d'advérsaire	La border de la textbox de recherche d'advérsaire s'affiche en rouge	La partie est crée dans la db et la partie est affichée sur la page

Page de jeu			
Action	**Condition**	**Scénarion echec condition**	**Scénario réussite condition**
Séléctionner une pièce à déplacer	variable "clickActif" doit être à "false"	Les déplacements possible de la pièce séléctionnée précedemment disparaisent. La variable "clickActif" revient à l'état "false"	La pièce est séléctionnée et affiche en rouge ses déplacements possible
	La pièce cliquée doit avoir des déplacements possibles	Rien ne se passe visuellement et la variable "clickActif" reste à false	
	La pièce cliquée doit être de la même couleur que celle du joueur		
	Il faut que la couleur du joueur corresponde à la couleur du joueur dont c'est le tour		
Déplacer une pièce	Il faut avoir deja séléctionné une pièce		La pièce séléctionnée se déplace sur la case clickée, la case source de la pièce devient vide et les déplacements possibles en rouge disparaissent. La variable "clickActif" passe à "false" et le tour change
	Le click du joueur doit se situer sur une des cases de déplacements possibles de la pièce séléctionnée à l'étape précédente	Les déplacements possibles de la pièce (en rouge) disparaissent et la variable "clickActif" passe à "false"	
Check de la présence de l'adversaire dans la partie	Etre connecté dans une partie	Le popup de check ne s'affiche pas sur la page	Le popup de check s'affiche sur la page: vert si l'utilisateurs est connecté, rouge si non-connecté
Check echec	dans au moins un des déplacements possibles de l'adversaire il se trouve la case de mon roi	Changement de tour	la case de mon roi s'affiche en rouge et tout les déplacements qui ne mettent pas le roi hors de dangers sont bloqués
	Une pièce doit avoir été déplacée		
Check "echec & mat"	Dans les déplacements possibles de l'adversaire se trouve mon roi		mon roi est en echec et mat et la partie est officiellement finie, le perdant/gagnant sont définis et enregistrés
	Mon roi ne peut pas sortir des déplacements possibles de l'adversaire	le roi est en echec	
	Aucunes de mes pièces ne peuvent protèger le roi		
Quitter page de jeu	Etre sur la page de jeu		la page de jeu se ferme
	Cliquer sur bouton "back to menu"		
			le menu en mode connecté s'affiche

Page de continuation de partie			
Action	**Condition**	**Scénarion echec condition**	**Scénario réussite condition**
Afficher toutes les parties commencées et non-finies	Avoir déjà une partie à son nom	Rien ne s'affiche sur la page "continue game"	La ou les parties commencées s'affichent sur la page avec "nom du créateur de la partie", "date de création"
Séléctionner et charger une partie précédemment commencée et jamais finie	Avoir une ou des parties affichées dans la page		le tableau de jeu s'affiche au point ou il était au dernier déplacement
Charger une partie commencée et non finie	Cliquer sur une proposition de partie de la page "continue game"		

2.8 Requêtes http

Le serveur NodeJS comprend 2 types de requêtes : http & webSocket. Voici des schémas la façon dont le serveur gère ces différentes requêtes :

2.8.1 Requêtes http ''classiques''

Ce sont les requêtes classiques du protocole http. Lorsqu'un client effectue une requête, le serveur répond à ce client et personne d'autre. Dans ce cas, le serveur a besoin de recevoir une requête d'un client pour pouvoir répondre, sinon il ne peut rien faire.

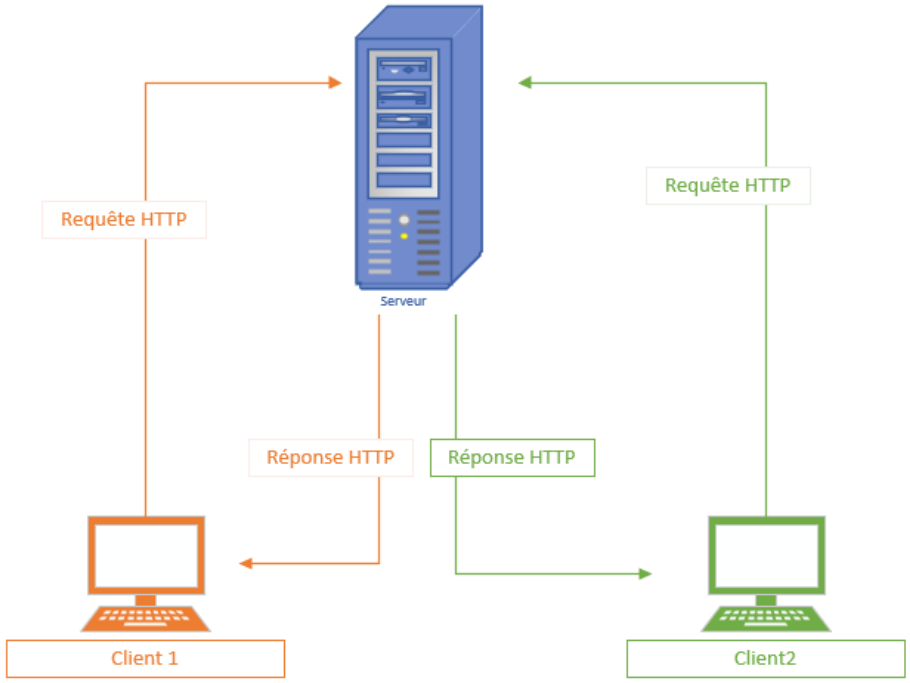


Figure 19 : Schéma Http

2.8.2 Requêtes HTTP WebSocket

Une WebSocket est une connexion http persistante, c'est à dire que le serveur et les clients restent connectés en permanence durant une période déterminée. Concrètement cela signifie que le serveur peut envoyer une requête vers le client sans que celui-ci ne demande quoi que ce soit. J'ai donc utilisé cette technologie pour mettre à jour le tableau de jeu en temps réel lorsqu'un joueur déplace une pièce.

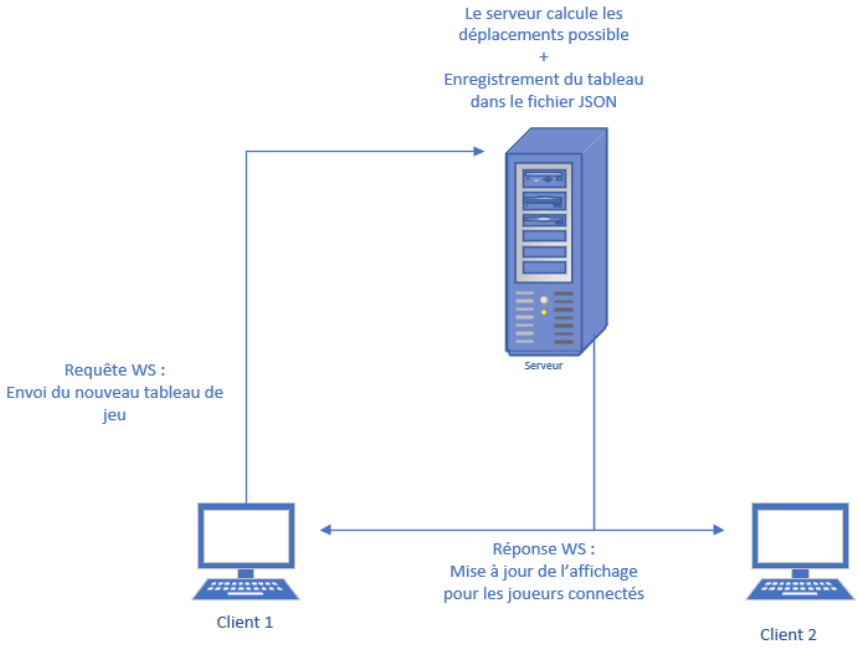


Figure 20 : Schéma WS

3 Conception

3.1 Architecture

Cette application est conçue comme « client lourd » dans le sens qu'il y a une architecture type client-serveur. J'ai opté pour une architecture application web plutôt que « site web », c'est-à-dire que le serveur, au lieu de fournir un fichier html par page, il ne fournit qu'un seul fichier html qui évolue selon les actions de l'utilisateur. Le code que je considère comme ma version finale se trouve dans le dossier « version_developpement », l'autre dossier contient le dossier à mettre en ligne (intestable en local et code moins abouti).

3.2 Logiciels

Voici les logiciels utilisés :

- ⇨ Editeur de texte : Visual Studio Code
- ⇨ Framework : Node.JS
- ⇨ SGBD : Xampp (Apache, MySql)

3.3 Principaux concepts

Voici les principaux éléments principaux qui composent le projet :

- ⇨ Html, Css, Javascript (natif & JQuery)
- ⇨ Modèle – Vue - Contrôleur
- ⇨ Programmation Orienté Objet
- ⇨ Un serveur HTTP (module 'express')
- ⇨ Un serveur Web Socket (module 'WS')
- ⇨ Un système d'authentification (module 'JsonWebToken')
- ⇨ Stockage des mots de passe encryptés avec algorythme de hashage (module 'bcrypt')
- ⇨ Base de données MariaDb

3.4 Diagrammes de classes

3.4.1 Classes frontend

Ces classes sont utilisées pour simplifier la gestion des données côté client. La fonction « CreateGame » envoi une requête au serveur, qui va utiliser les classes backend pour générer un nouveau plateau de jeu.

Figure 22 : diagramme de classe

3.4.2 Classes Backend

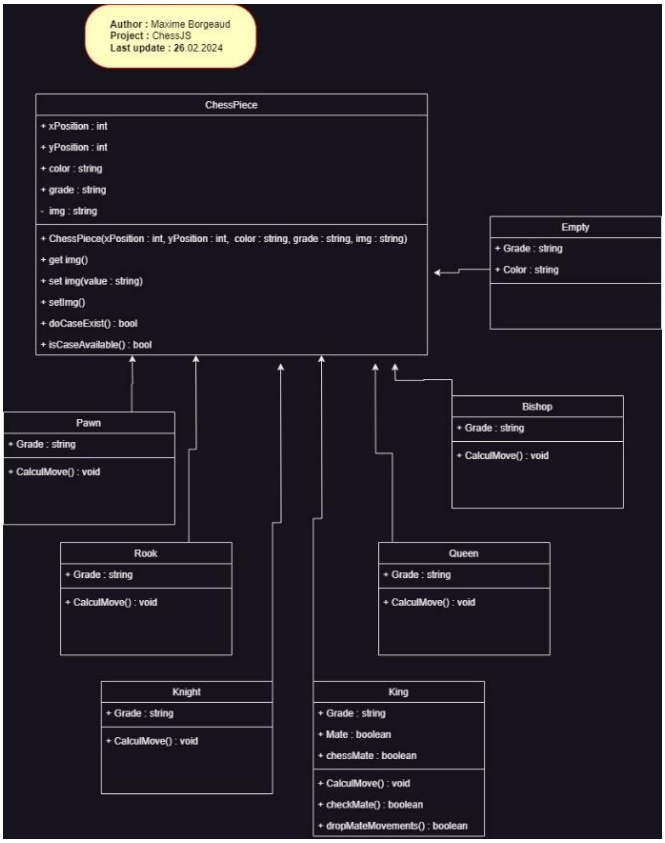


Figure 21 : diagramme de classe

Ce diagramme représente les classes utilisées pour gérer un plateau d'échec dans l'application. Il y a la classe parente qui représente chaque case du tableau de jeu (même les cases vides). Chacune instance a une position x & y, une couleur, un grade et une image. Les getter/setter servent à attribuer automatiquement l'image qui correspond à la pièce, selon le grade etla couleur. Les classes enfants servent principalement à calculer les mouvements des pièces. La seule spécificité est que le roi a deux fonctions supplémentaires : « checkMate() » qui sert à détecter la situation d'échec et « dropMateMovements » qui sert à enlever les possibilités de mouvements du roi qui le mettrait en situation d'échec.

3.5 Diagramme de flux

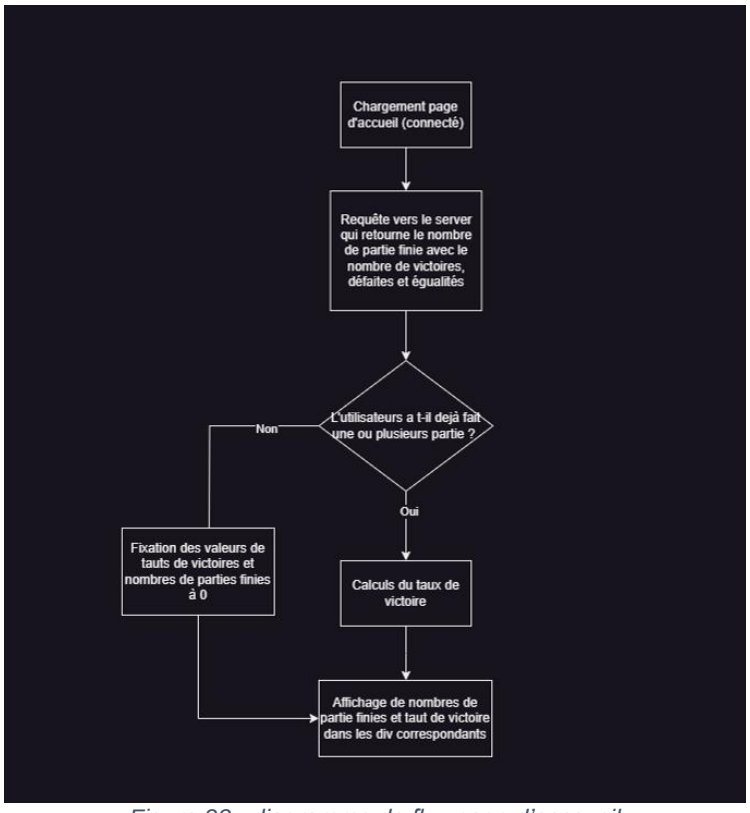


Figure 23 : diagramme de flux page d'acccueil

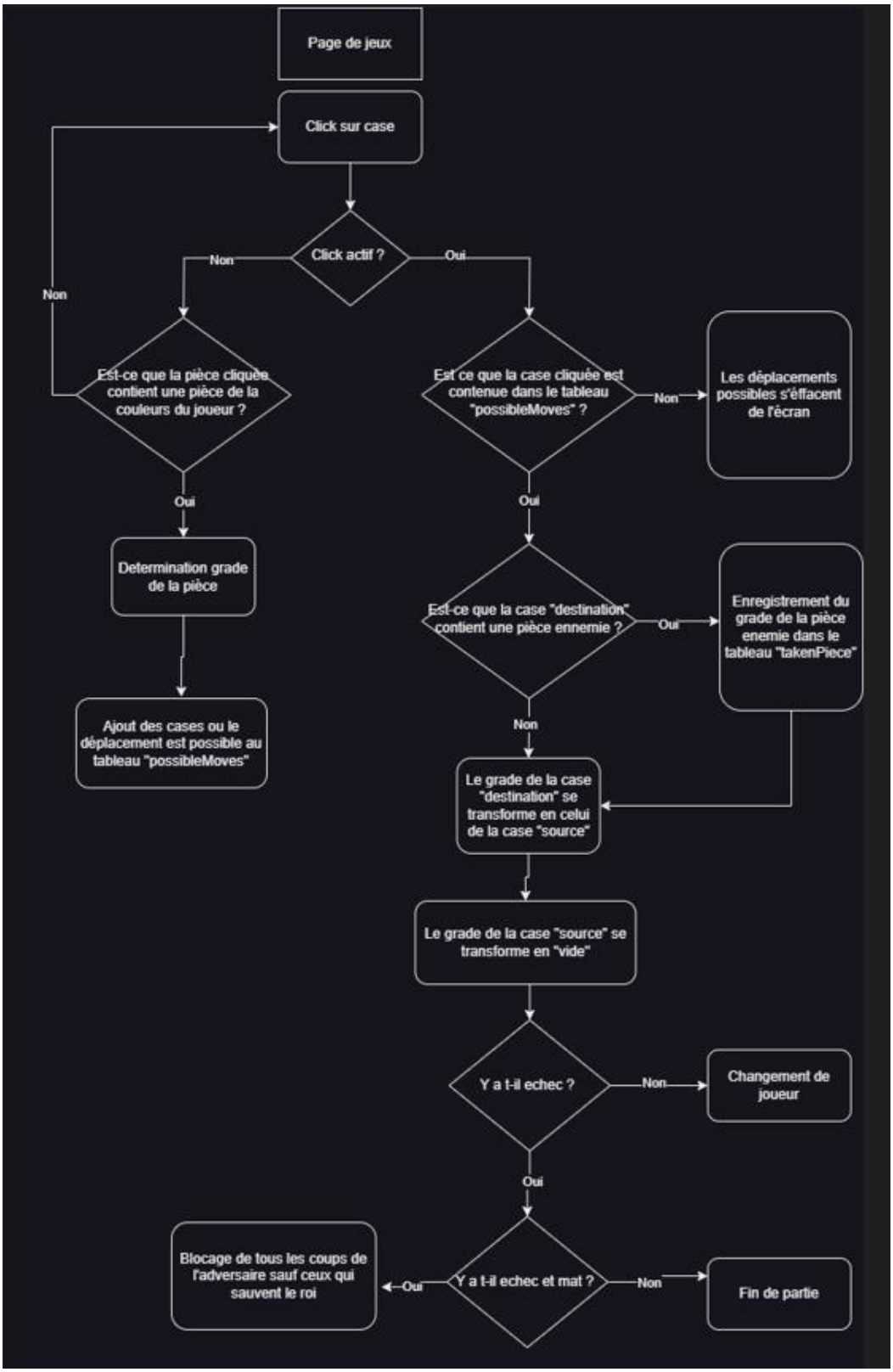


Figure 24 : diagramme de flux page de jeu

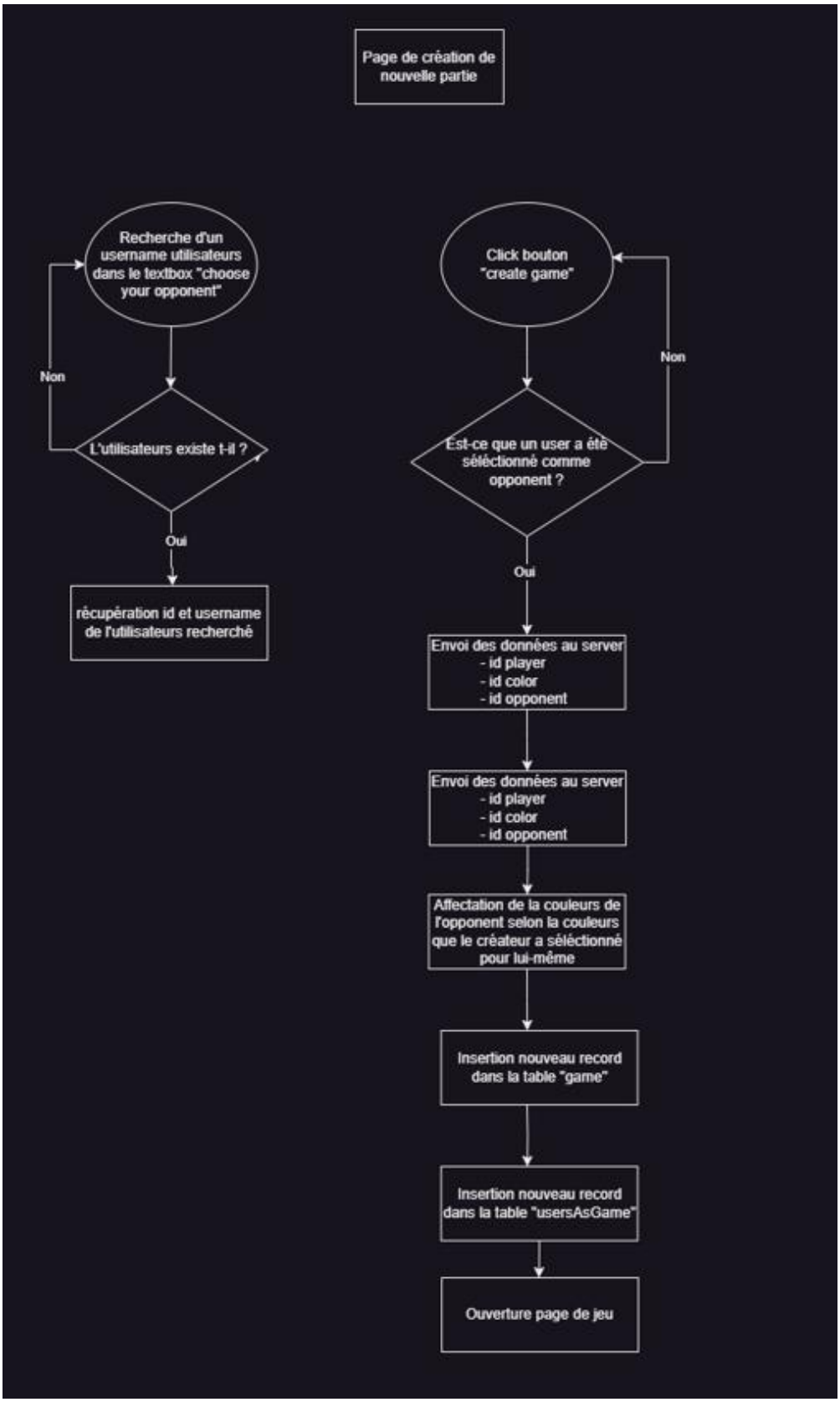
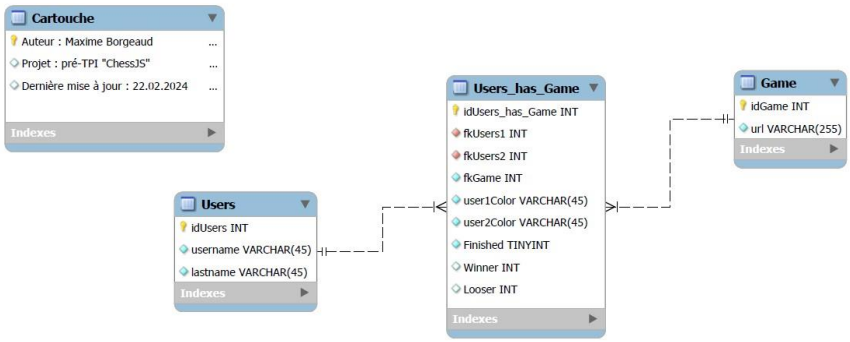


Figure 25 : diagramme de flux page de création de jeu

Figure 25 : diagramme de flux page de jeu

3.6 Modèle logique de données



Quelques explications concernant le MLD s'imposent. Je précise que le champs « fkUser1 » représente dans tous les cas le créateur de la partie. C'est lui qui choisit sa propre couleur et donc la couleur adverse est attribuée automatiquement au « fkUser2 ». Le champs « colorTurn » représente la couleur du joueur à qui c'est le tour de jouer, pendant ce temps l'autre joueur est bloqué. Le champs booléen « finished » précise si la partie est finie ou non, sa valeur par défaut est « NULL ». Il sert principalement à la détermination de l'affichage de la partie dans l'onglet « continue game », si la partie est effectivement finie, on n'affiche pas la partie. Il permet également l'établissement des statistiques du joueur.

3.7 Stratégie de tests

Page d'accueil					
Action	**Condition**	**Comportement attendu echec condition**	**comportement obtenu echec condition**	**comportement attendu reussite condition**	**comportement obtenu réussite condition**
Afficher page d'accueil en mode connecté	avoir déjà été authentifié par le server	La page d'accueil d'authentification s'affiche (login / subscribe)	La page d'accueil d'authentification s'affiche (login / subscribe)	La page d'accueil du jeu s'affiche en mode non-connecté	La page d'accueil du jeu s'affiche en mode non-connecté
Page de login					
Action	**Condition**	**Comportement attendu echec condition**	**comportement obtenu echec condition**	**comportement attendu reussite condition**	**comportement obtenu réussite condition**
Se connecter	Avoir déjà un compte avec ce username	La bordure des textbox de username & password s'affichent en rouge avec un message "le mot de passe ou username est incorrect"	La bordure des textbox de username & password s'affichent en rouge	La page d'accueil en mode connecté s'affiche	La page d'accueil en mode connecté s'affiche
	Avoir écrit le mot de passe associé à ce compte				
Page d'inscription					
Action	**Condition**	**Comportement attendu echec condition**	**comportement obtenu echec condition**	**comportement attendu reussite condition**	**comportement obtenu réussite condition**
S'inscrire	Ecrire un username qui n'a pas déjà été pris	La bordure des textbox et un message "username already taken" s'affiche	Il ne se passe rien	La page d'accueil en mode connecté s'affiche	La page d'accueil en mode connecté s'affiche
	Les deux champs "username", "Password" & "Confirm password" doivent être remplis	la border de la textbox s'affiche en rouge			
	Les champs"password" et "confirm password" doivent être identiques				

Page de création d'une partie					
Action	**Condition**	**Comportement attendu echec condition**	**comportement obtenu echec condition**	**comportement attendu reussite condition**	**comportement obtenu réussite condition**
Rechercher un advérsaire	Le username doit être présent dans la db	La border de la textbox de recherche d'advérsaire s'affiche en rouge et un message d'erreur s'affiche "this opponent is not available"	La border de la textbox de recherche d'advérsaire s'affiche en rouge	Le username de l'adversaire entré s'affiche en vert en dessous de la textbox	Le username de l'adversaire entré s'affiche en vert en dessous de la textbox
	Le username doit être différent de celui de l'utilisateurs qui l'entre				
Créer la partie	Un advérsaire à été trouvé dans la section de recherche d'advérsaire			La partie est crée dans la db et la partie est affichée sur la page	La partie est crée dans la db et la partie est affichée sur la page

Page de jeu					
Action	**Condition**	**Comportement attendu echec condition**	**comportement obtenu echec condition**	**comportement attendu reussite condition**	**comportement obtenu réussite condition**
Séléctionner une pièce à déplacer	variable "clickActif" doit être à "false"	Les déplacements possible de la pièce séléctionnée précedemment disparaisent. La variable "clickActif" revient à l'état "false"	Les déplacements possible de la pièce séléctionnée précedemment disparaisent. La variable "clickActif" revient à l'état "false"	La pièce est séléctionnée et affiche en rouge ses déplacements possible	La pièce est séléctionnée et affiche en rouge ses déplacements possible
	La pièce cliquée doit avoir des déplacements possibles	Rien ne se passe visuellement et la variable "clickActif" reste à false	Rien ne se passe visuellement et la variable "clickActif" reste à false		
	La pièce cliquée doit être de la même couleur que celle du joueur				
	Il faut que la couleur du joueur corresponde à la couleur du joueur dont c'est				
Déplacer une pièce	Il faut avoir déjà séléctionné une pièce			La pièce séléctionnée se déplace sur la case clickée, la case source de la pièce devient vide et les déplacements possibles en rouge disparaissent. La variable "clickActif" passe à "false" et le tour change de joueur.	La pièce séléctionnée se déplace sur la case clickée, la case source de la pièce devient vide et les déplacements possibles en rouge disparaissent. La variable "clickActif" passe à "false" et le tour change de joueur.
	Le click du joueur doit se situer sur une des cases de déplacements possibles de la pièce séléctionnée à l'étape précédente	Les déplacements possibles de la pièce (en rouge) disparaissent et la variable "clickActif" passe à "false"	Les déplacements possibles de la pièce (en rouge) disparaissent et la variable "clickActif" passe à "false"		
Check de la présence de l'adversaire dans la partie	Etre connecté dans une partie	Le popup de check ne s'affiche pas sur la page	Le popup de check ne s'affiche pas sur la page	Le popup de check s'affiche sur la page: vert si l'utilisateurs est connecté, rouge si non-connecté	Le popup de check s'affiche sur la page: vert si l'utilisateurs est connecté, rouge si non-connecté
Check echec	dans au moins un des déplacements possibles de l'adversaire il se trouve la case de mon roi	Changement de tour	Changement de tour	la case de mon roi s'affiche en rouge et tout les déplacements qui ne mettent pas le roi hors de dangers sont bloqués	
	Une pièce doit avoir été déplacée				
Check "echec & mat"	Dans les déplacements possibles de l'adversaire se trouve mon roi			mon roi est en echec et mat et la partie est officiellement finie, le perdant/gagnant sont définis et enregistrés	
	Mon roi ne peut pas sortir des déplacements possibles de l'adversaire	le roi est en echec	le roi est en echec		
	Aucunes de mes pièces ne peuvent protéger le roi				
Quitter page de jeu	Etre sur la page de jeu			la page de jeu se ferme	la page de jeu se ferme
	Cliquer sur bouton "back to menu"			le menu en mode connecté s'affiche	le menu en mode connecté s'affiche

Page de continuation de partie					
Action	**Condition**	**Comportement attendu echec condition**	**comportement obtenu echec condition**	**comportement attendu reussite condition**	**comportement obtenu réussite condition**
Afficher toutes les parties commencées et non-finies	Avoir déjà une partie à son nom	Rien ne s'affiche sur la page "continue game"	Rien ne s'affiche sur la page "continue game"	La ou les parties commencées s'affichent sur la page avec le username de l'adversaire 1 la date de création	La ou les parties commencées s'affichent sur la page avec le username de l'adversaire du créateur de la partie (static) + la date de créstion
Séléctionner et charger une partie précédemment commencée et jamais finie	Avoir une ou des parties affichées dans la page			le tableau de jeu s'affiche au point ou il était au dernier déplacement	le tableau de jeu s'affiche au point ou il était au dernier déplacement
Charger une partie commencée et non finie	Cliquer sur une proposition de partie de la page "continue game"				

4 Réalisations

4.1 Démarrer l'application en local

Prérequis :

1) Il faut avoir ''NodeJS'' installé sur l'ordinateur.
2) Avoir une connexion internet (pour se connecter à la base de données sur SwissCenter)
3) Les fichiers à importer se trouve dans ce repository : https://github.com/maxDevelopement/ChessJS

Une fois les fichiers importés depuis le github (, il faut ouvrir un bash et se déplacer dans le dossier « version_developpement » puis taper la commande ''npm start''. Vous devriez avoir un retour de ce genre dans la console :

```
url repository\ ChessJS\version_developpement> npm start

> start
> node app.js

server run on port 3000
(node:13152) Warning: Accessing non-existent property 'King' of module exports inside circular dependency
(Use `node --trace-warnings ...` to show where the warning was created)
(node:13152) Warning: Accessing non-existent property 'Bishop' of module exports inside circular dependency
(node:13152) [SEQUELIZE0006] DeprecationWarning: This database engine version is not supported, please update your database server. More information https://github.com/sequelize/sequelize/blob/main/ENGINE.md
connexion to db successfull
```

A ce moment le serveur tourne sur le port 3000 de l'ordinateur hôte, une fois démarré, l'application est disponible à l'url : http://< adresseIP || localhost>:3000. Je conseille l'utilisation de « Microsoft Edge » car parfois Google Chrome bloque les requêtes HTTP

Si vous obtenez une erreur de connexion à la base de données, faites un « ctrl + C » pour stopper le serveur puis refaire un « npm start »

4.2 Base de données

Dans les captures d'écrans suivantes, les clefs jaunes représentent les « primary keys » et les clefs grises représentes les champs qui ont une contrainte d'unicité

4.2.1.1 Table « Games »

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
1	**idGame**	int(255)			Non	*Aucun(e)*		AUTO_INCREMENT
2	**url**	varchar(255)	utf8mb4_general_ci		Non	*Aucun(e)*		

4.2.1.2 Table UserAsGame »

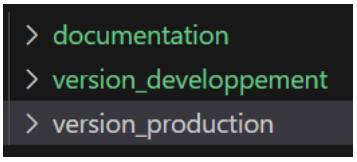
#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
1	idUserAsGame	int(255)			Non	*Aucun(e)*		AUTO_INCREMENT
2	fkUser1	int(255)			Non	*Aucun(e)*		
3	fkUser2	int(255)			Non	*Aucun(e)*		
4	fkGame	int(255)			Non	*Aucun(e)*		
5	fkActualPlayer	int(255)			Oui	*NULL*		
6	user1Color	varchar(255)	utf8mb4_general_ci		Non	*Aucun(e)*		
7	user2Color	varchar(255)	utf8mb4_general_ci		Non	*Aucun(e)*		
8	finished	tinyint(1)			Non	0		
9	winner	int(255)			Oui	*NULL*		
10	looser	int(255)			Oui	*NULL*		
11	createdAt	date			Non	*Aucun(e)*		

4.2.1.3 Table « User »

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
1	idUser	int(255)			Non	*Aucun(e)*		AUTO_INCREMENT
2	username	varchar(255)	utf8mb4_general_ci		Non	*Aucun(e)*		
3	password	varchar(255)	utf8mb4_general_ci		Non	*Aucun(e)*		

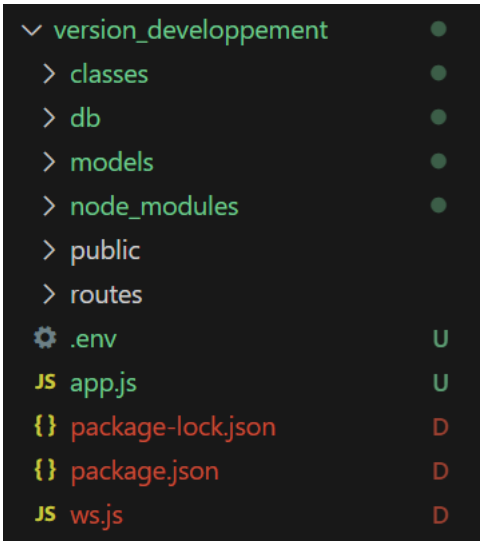
4.3 Structure du projet

4.3.1 Dossier racine



Ceci est le dossier principal qui contient les deux versions finales du projet : la version locale et celle prévue pour la mise en production sur le serveur de SwissCenter

4.3.2 Dossier de projet (« version_developpement »)



« app.js » : Il s'agit du fichier racine du server Express. Il gère toutes les requêtes http du type « subscribe » « login » etc. Si aucune route n'est trouvée pour la requête, le serveur renvoie une erreur 404.

« ws.js » : Ce fichier contient le serveur websocket. Il créé dynamiquement des tableaux qui contiennent les joueurs qui se connectent à une partie. Il gère la détection de la présence de l'adversaire et le déplacement des pièces

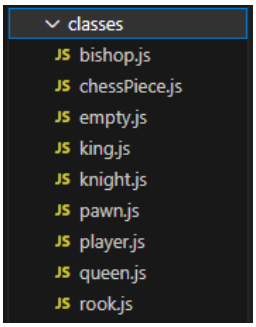
« node_modules » : contient toutes les librairies nécessaires au bon fonctionnement de l'application.

« .env » ci-dessous

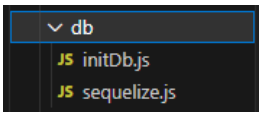
« .env » : fichier qui contient les variables d'environnement ''ACCESS_TOKEN_SECRET'' & ''REFRESH_TOKEN_SECRET'', qui servent de clefs d'identifications des utilisateurs.

« package-lock.json » & « package.jso » : Contiennent les informations sur les dépendances & les fichiers de ces dépendances.

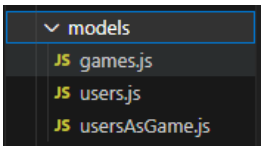
4.3.3 Sous-dossiers



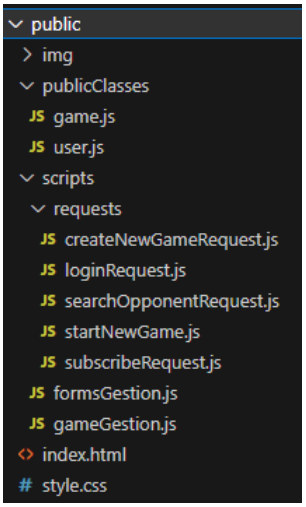
« classes » : Il contient les fichiers de classes non-accessibles depuis le client. Ces classes sont utilisées pour la gestion des pièces lors d'une partie. Effectivement les calculs de déplacements, les conditions d'echec etc. se feront sur le serveur et non sur les clients.



« db » : Il contient le fichier « sequelize » qui fait la connexion avec la base de données. « initDb » fait la synchronisation entre les modèles et la db elle-même. Ce deuxième fichier est exécuté lors du démarrage du server



« models » : Contient les modèles qui sert à faire des requêtes SQL à la db via la librairie « sequelize ». Ces modèles correspondent à des objets identiques aux tables de la db.



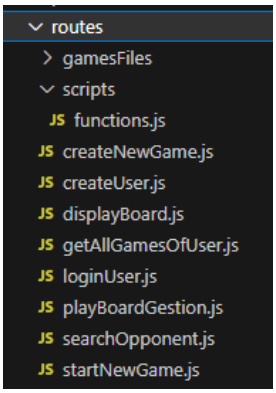
« public » : Contient tous les fichiers qui sont automatiquement envoyés au client lorsqu'il fait une requête qui pointe vers la racine du serveur

« img » contient les images de pièces

« publicClasses » : Contient les classes utiles à la manipulation des données côté client

Le dossier « scripts » contient 2 fichiers javascripts : « FormGestion »qui sert pour la gestion de l'interface frontend (ouverture/ fermeture des formulaire, envoi des requêtes etc) et « gameGestion » qui contient uniquement les éléments utiles à la gestion du jeu d'échec en lui-même.

Le dossier « requests » contient les fichiers de requêtes « fetch() » qui servent à communiquer avec le serveur



« routes » contient les fichiers javascript qui réceptionnent les requêtes http et effectuent les instructions demandées.

« gameFiles » contient les fichiers json dans lesquels sont enregistrés les parties commencées.

« scripts » contient un fichier qui lui-même contient une fonction très utile qui sert à identifier l'utilisateurs via son id.

4.4 Système de token

J'ai utilisé le module « jsonwebtoken ». J'ai créé un fichier caché « .env » qui contient les variables d'environnement suivantes :

```
ACCESS_TOKEN_SECRET=y3pgzrvoj8z34brolrf93ek6yn8dmd7mgsijkahzzq3zrjx5ne
REFRESH_TOKEN_SECRET=nw08mcu7bortp59w2uxzm45b6zbtahp1pmo426azg2b4vep8vp
```

Il s'agit de clefs de cryptage qui servent à identifier de façon sécurisée la source d'une requête. Lorsqu'un utilisateur se logue ou s'inscrit, un AccessToken est créé avec ses informations cryptées en combinaison du token secret. Toutes les requêtes (excepté celles de login & subscribe) envoyées doivent avoir dans le « header » une section « autorization » qui contient le AccessToken du user. A la racine du serveur il y a un middleware qui vérifie que toutes les requêtes possèdent ce token, si elle ne contient pas, cela signifie que l'utilisateurs n'est pas identifié et le serveur renvoie une erreur.

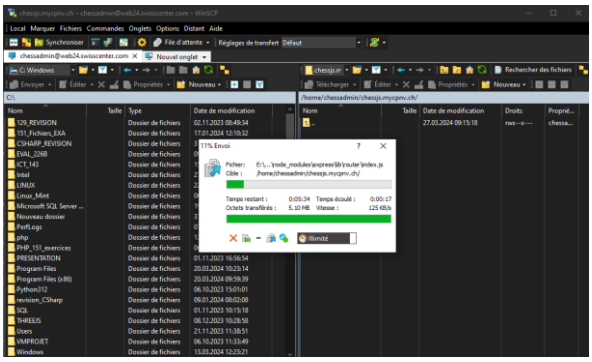
4.5 Mise en production sur Swisscenter

4.5.1 Modification de code

Premièrement j'ai commencé par désinstaller les modules « cors » et « nodemon » qui ne sont utiles que pendant la phase de développement. Probablement que le dernier poserait même des problèmes.

4.5.2 Procédure

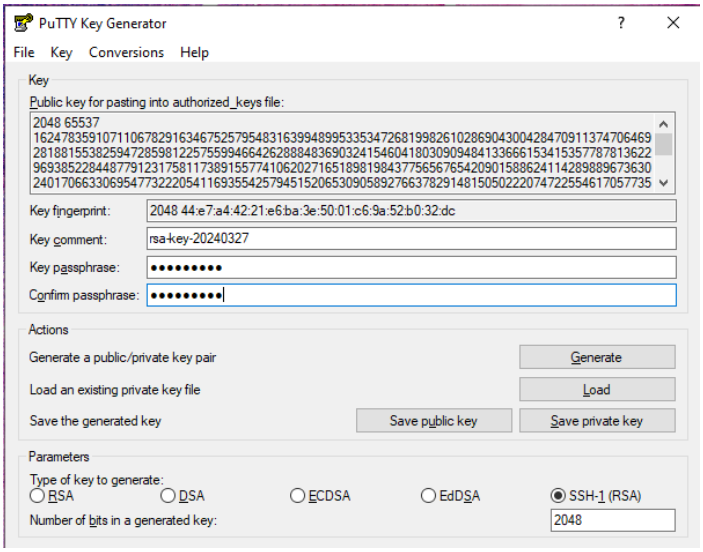
Je me suis connecté sur le server de Swisscenter via FTP afin d'uploader tous mes fichiers de projet.



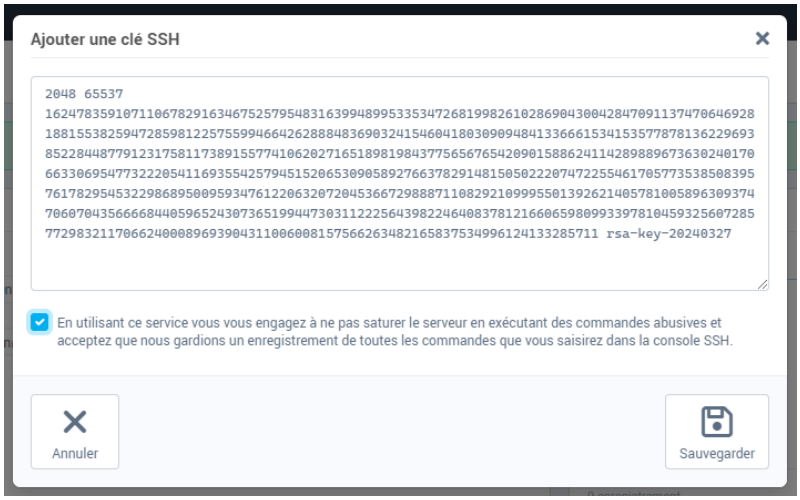
Mon dossier ne servant pas directement un fichier « index.html » qui se trouve à la racine du répertoire, il faut que je me connecte au terminal du server afin de pouvoir démarrer mon application pour qu'elle puisse servir les fichiers aux clients.

Pour faire cette étape, le server exige une connexion SSH, voici comment j'ai procédé :

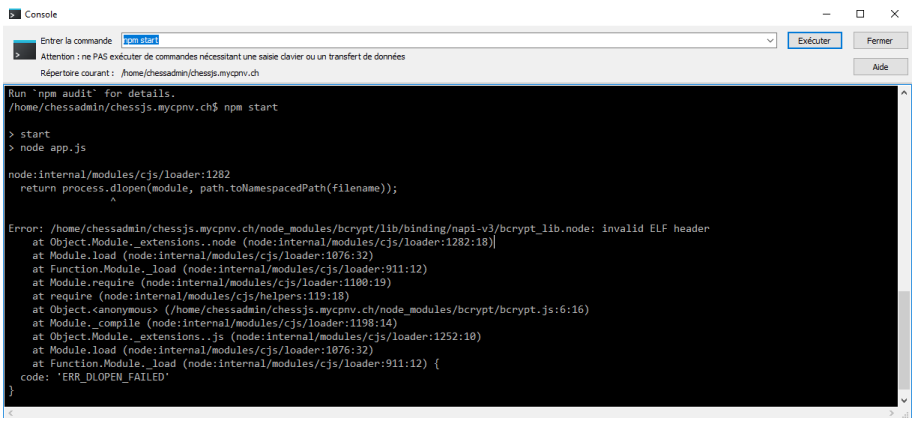
1) Création de clefs publique/privée SSH via « PuttyGen » :



2) Enregistrement des clefs dans un dossier « .ssh »
3) Ajout de la clef publique sur le server de Swisscenter :

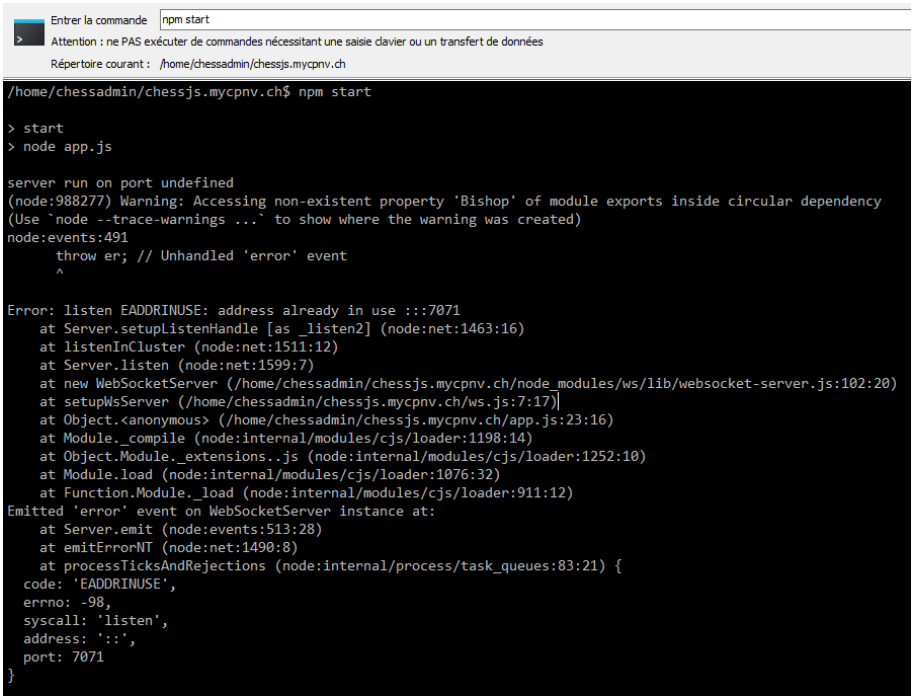


4) Connexion au terminal du server avec connexion SSH via le logiciel « WinSCP »
5) Démarrage du server avec la commande « npm start » :



A ce moment j'ai eu un problème de compatibilité avec le module « bcrypt », il semblerait que la version installée sur le server ne soit pas exactement la même que celle installée dans mon projet.

J'ai simplement désinstallé puis réinstallé le module directement dans le serveur et l'erreur a disparu. Ensuite j'ai eu cette deuxième erreur qui semble être un problème de numéro de port d'écoute qui sont paramétré de façon incorrecte :



C'est l'étape que je n'ai pas pu dépasser. Etant donné que nous n'avons aucun support de SwissCenter pour l'hébergement et que toutes mes recherches se sont révélées infructueuse, ce problème de port d'écoute n'a pas pu être réglé. Peut-être qu'encapsuler l'application dans un container Docker résoudrait le souci ?

5 Conclusion

5.1 Erreurs restantes

5.1.1 Erreurs au démarrage de l'application

Premièrement au démarrage j'ai ces deux erreurs qui apparaissent, bien qu'elles n'entravent pas le fonctionnement de l'application :

1)

```
(node:16756) [SEQUELIZE0006] DeprecationWarning: This database engine version is not supported, please update your databa
se server. More information https://github.com/sequelize/sequelize/blob/main/ENGINE.md
(Use `node --trace-deprecation ...` to show where the warning was created)
```

Résoudre ce problème mérite une petite investigation car le lien github qu'ils proposent mène vers une page 404. Également après avoir rapidement testé la commande « --trace-deprecation », cela n'a rien donné.

2)

```
(node:13152) Warning: Accessing non-existent property 'King' of module exports inside circular dependency
(Use `node --trace-warnings ...` to show where the warning was created)
(node:13152) Warning: Accessing non-existent property 'Bishop' of module exports inside circular dependency
```

Il semblerait que ce soit un problème d'import des classes de pièces. Le problème est qu'elles apparaissent de façon aléatoire et non systématiquement, parfois seulement le ''bishop'', parfois rien, parfois d'autres. J'ai tenté de trouver la source de ce problème mais le temps m'a manqué pour trouver la solution.

5.1.2 Affichage des messages d'erreurs des formulaires

Dans la version actuelle, lorsqu'un utilisateur essaie de s'inscrire avec un username déjà pris, aucuns messages d'erreur ne s'affiche pour préciser le problème, ce qui est embêtant pour l'utilisateurs car il ne sait pas pourquoi cela ne fonctionne pas. Également lors de la création de partie, dans le textBox de recherche d'adversaire, si l'utilisateurs rentre son propre username ou un qui n'existe pas dans la base de données, l'application signale que ça ne fonctionne pas en mettant la bordure en rouge. Un petit message précisant la raison du problème faciliterait la compréhension de l'utilisation du programme.

5.1.3 Fonctionnalités non-terminées ou non-implémentées

La plupart des éléments qui se trouvent dans cette section sont dus à un manque de temps ou à une mauvaise stratégie d'organisation. Je détaille cela dans la conclusion personnelle.

- ⇨ **« Echec » & « echec et mat »**

Cette fonctionnalité a été partiellement implémentée mais des problèmes, ont fait que je n'ai pas pu la finir à temps. Cette partie du code se trouve dans la classe « King » et a été mise en commentaire car cela interférait avec le bon fonctionnement de l'application. En effet sinon la fonction entraine un ralentissement énorme du temps de traitement d'un déplacement d'une pièce.

- ⇨ **Page d'accueil en mode connecté**

Premièrement l'affichage des statistiques du joueurs n'a pas été implémenté pour des questions de temps. Deuxièmement la section « invitations » (voir figure 7) a disparu car cela compliquait passablement l'architecture du projet (encore un problème de temps/organisation). Maintenant lorsqu'un joueur crée une partie avec un adversaire, au lieu de passer par un système d'invitations que le receveur doit accepter/refuser, celui-ci la voit instantanément dans la section « continue game », sans possibilités de refuser.

- ⇨ **Affichage des statistiques du joueurs (page d'accueil en mode connecté – voir figure 7)**
- ⇨ **Mettre fin à la partie manuellement page de jeu – voir figure 17)**

5.1.4 Détails interface

1) Page « Continue game »

Lorsque l'on souhaite afficher toutes les parties commencées de notre profil, le jeu affiche l'information suivante d'une partie :

Deux détails (importants) sont à changer. Premièrement le username (ici « max ») est sensé représenter l'adversaire de l'utilisateur connecté mais en réalité il représente l'adversaire du créateur de la partie. L'adversaire voit donc la même chose. Deuxièmement il faudrait modifier le format de date de création de la partie (enlever les secondes et millisecondes).

2) Page de jeu

L'affichage des déplacements possibles d'une pièce lorsque l'on clique dessus n'est pas optimale. De plus, il serait mieux d'ajouter un petit message pour ces 3 cas :

- ⇨ Lorsque l'utilisateur clique alors que ce n'est pas son tour
- ⇨ Lorsque l'utilisateurs clique sur une pièce de la mauvaise couleur
- ⇨ Lorsque l'utilisateur clique sur une case vide

5.2 Améliorations possibles

1) Ajouter tous les éléments qui manquent au projet de base (points explicités dans les sections précédentes « Fonctionnalités non-terminées ou non-implémentées » et « Détails interface »)
2) Visualisation des pièces ennemies que l'utilisateur a « mangé » durant la partie
3) Pouvoir créer un partie « solo » avec une IA basique qui remplacerait l'adversaire

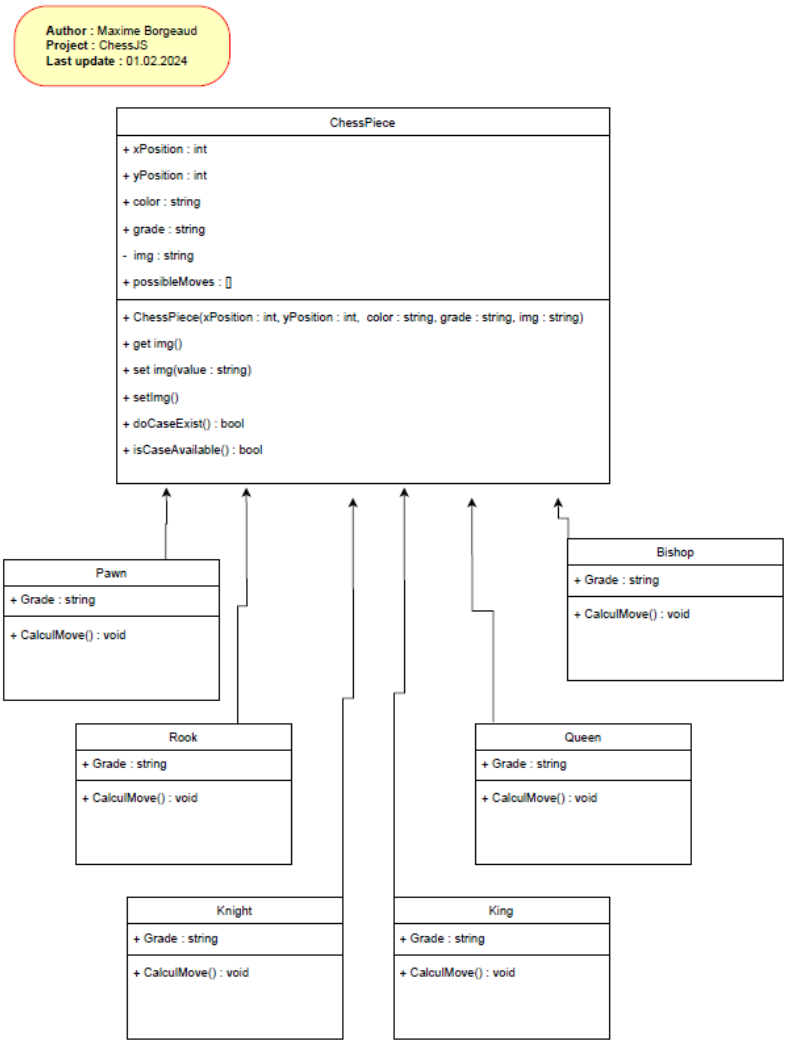
5.3 Conclusion personnelle

Je dois commencer ma conclusion par dire que le projet était, selon moi, un petit peu ambitieux pour le temps à disposition. Je dois également admettre que j'ai commis une terrible erreur de planification au début du projet. En effet j'ai commencé par l'affichage du tableau de jeu et des calculs de déplacement, puis je me suis s'occupé de la base de données et de l'interface pour finir avec le calcul de l'échec et mat. Le résultat est que pour faire ce dernier point je pense que je devrais refactoriser tout mon code car il n'est absolument pas optimisé pour faire cette détection de façon propre et efficace. Cette erreur fut particulièrement riche d'apprentissage et j'en ai tiré cette leçon : Il faut commencer par le frontend et finir avec le backend (ou vice-versa), mais ne pas trop mélanger les deux.

6 Archives

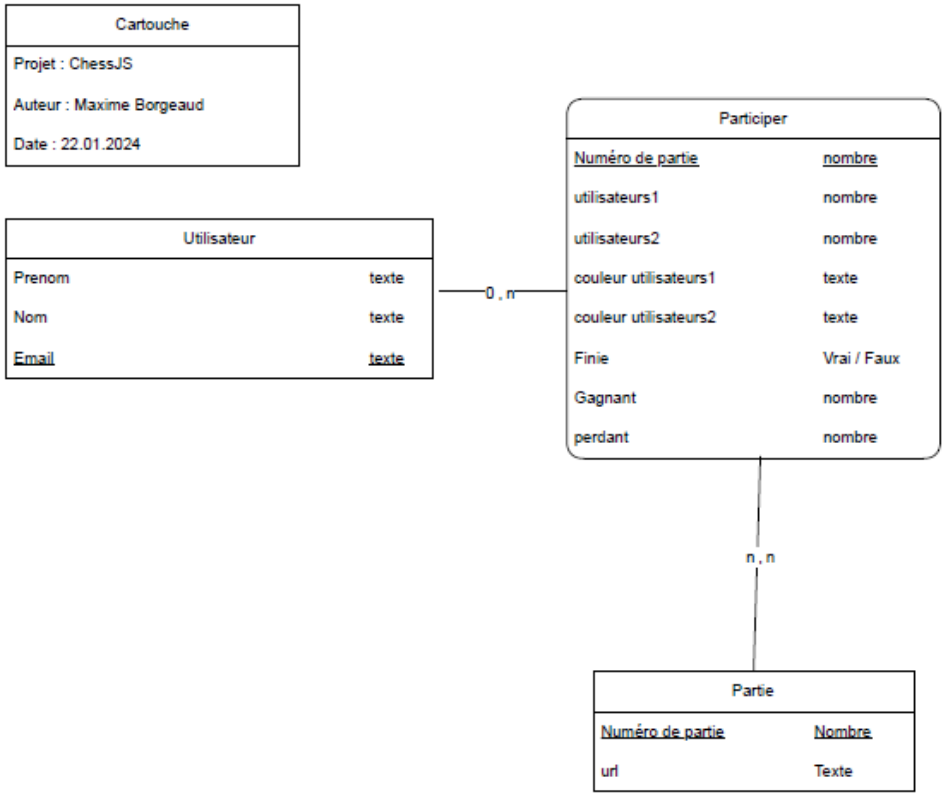
Vous trouverez ci-dessous les premières versions de certains éléments du projet. Ces éléments ont dû être modifiés pour des questions d'optimisations ou il y a eu suppressions d'éléments inutiles ou redondants :

1.1.1 Diagramme de classe

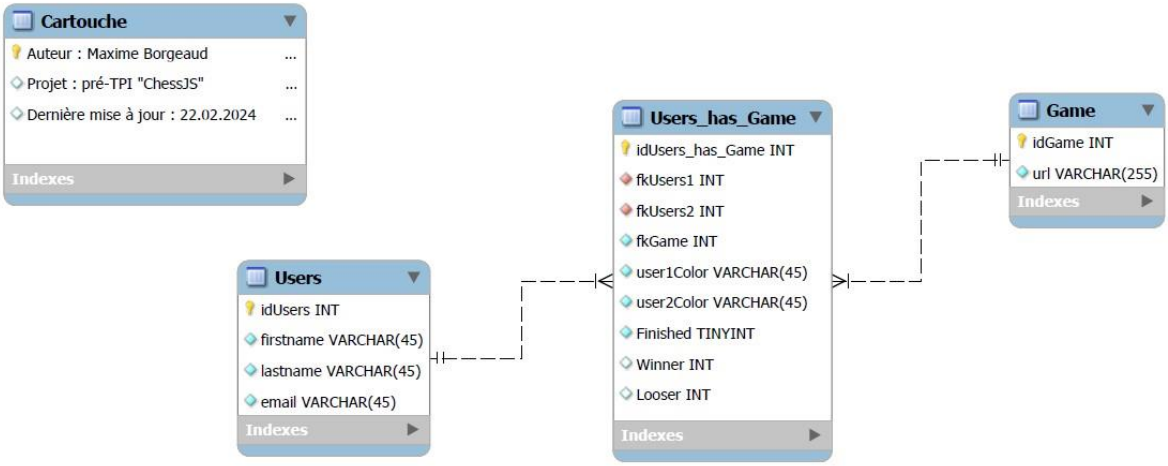


1.1.2 MCD

Dans ce diagramme et également dans le MLD ci-dessous, j'ai remplacé les 3 champs « nom, prenom et email » par « username » pour des questions de simplicités



1.1.3 MLD



7 Annexes

⇨ Journal de travail.pdf