PROJET TUTORÉ

**Rapport retour d'expérience et nouvelles fonctionnalités**

ligne horizontale

# 

ligne horizontale

**Responsable:** POUZIN Pierre-Emmanuel Groupe n°1

**Responsable Qualité:** GIE Jimmy

**Responsable Données:** STIZ Romain

**Responsable Développeurs:** GUIBARD Théo

**Responsable DevOps:** ROUSSEAU Emile

**Développeurs:** VOUETTE Maxence, POULET Alexandre

[**1 - Introduction, Remise en contexte 3**](#_nh5ctfj7qece)

[1.1 Fonctionnalités et Choix Techniques Associés 3](#_etddaczb45x2)

[1.2 Schéma de Dépendances Global 4](#_kwm2gj8w215j)

[**2 - Retour d’expérience sur l’environnement de développement et les choix techniques 5**](#_wks2dmhqmu6c)

[2.1 Git 5](#_7n7haffbpsly)

[2.2 Docker 5](#_fgzjlmpunge)

[2.3 Technologies Front 5](#_2sla2jexj4v5)

[2.3 Technologies ETL 6](#_qe609uuegqf)

[2.4 Technologies Back 6](#_lkw2veqe4l1b)

[**3 - Retour d’expérience sur l’organisation du groupe 7**](#_xzp7d4a3s3rz)

[3.1 Planification et distribution des tâches 7](#_9hb41yfxa7b1)

[3.2 Communication 8](#_5ch3pqhebv2e)

[**4 - Nouvelles fonctionnalités 10**](#_iozlni407iyv)

[4.1 Nouveau moteur d’échec 10](#_eo0hptxyad0w)

[4.2 Minimax 13](#_ce0evry7iho)

[4.3 Détection de triche 15](#_ajuq0c8pmmrs)

[4.4 Jeu en réseau et socket sur nouveau moteur 16](#_t4dsylw73cor)

[4.5 Authentification 17](#_r6frjcf7cen2)

[4.6 Application mobile 19](#_a54zp4slwfxm)

[4.7 Restructuration front/back 21](#_h98usnexxu6e)

[4.8 Modifier schéma BD 22](#_6efuqv561c7x)

[4.9 Interactions entre utilisateurs 24](#_heyw3q50drsm)

[4.10 Visualisation des données 24](#_n20xoz80i4hy)

[4.11 Répartition des tâches 25](#_d3b9uqfqtug6)

[**5 - Conclusion 27**](#_l7lub0702nar)

## **1 - Introduction, Remise en contexte**

Le semestre dernier, nous avons entrepris le développement d'une application dédiée aux échecs, ayant pour but de fournir une aide à la décision en analysant les parties en temps réel. Notre projet s'appuie sur l'intégration de données provenant de Lichess pour offrir des recommandations sur les coups joués et des suggestions pour les prochains coups. L'application permet également aux utilisateurs de jouer des parties contre d'autres joueurs, avec ou sans l'assistance de moteurs comme Stockfish. De plus, nous avons intégré un système de détection de tricheurs basé sur l'analyse des coups.

Ce qui a été réalisé:

**Analyse en temps réel** : Utiliser les données de parties de Lichess pour fournir des indications et recommandations de coups en cours de partie.

**Jeu contre des bots et des joueurs humains** : Permettre aux utilisateurs de jouer contre d'autres joueurs ou contre des moteurs d'IA comme Stockfish.

**Programme ETL** : Développer un programme ETL pour filtrer et intégrer les données pertinentes de Lichess dans notre base de données.

### 1.1 Fonctionnalités et Choix Techniques Associés

#### 1.1.1 Frontend

L'application utilise des technologies modernes telles que React, TypeScript et Vite pour une interface utilisateur fluide et interactive. Les principaux composants sont :

**React et TypeScript** : Pour une interface réactive et bien structurée.

**Chess.js et React-Chessboard** : Pour la logique du jeu et l'affichage interactif de l'échiquier.

**IA Stockfish via Web Workers** : Pour exécuter les calculs sans perturber l'interface utilisateur.

**OAuth 2.0 avec PKCE** : Pour l'authentification sécurisée avec Lichess.

**Exportation en format PGN** : Pour permettre aux utilisateurs de sauvegarder l'historique des parties.

#### **1.1.2 Backend**

Le backend assure la gestion et l'exploitation des données des parties d'échecs, offrant des points d'accès RESTful pour la communication avec le frontend. Les principales responsabilités incluent :

**Recommandations de coups** : Fournir des suggestions de coups basées sur les analyses en temps réel.

**Statistiques détaillées** : Générer des statistiques pour une meilleure analyse des parties.

**Détection de tricherie** : Identifier les joueurs non humains.

**Persistance des données** : Utilisation de JPA/Hibernate pour la manipulation et la persistance des données.

#### **1.1.3 ETL**

Le programme ETL permet d’enrichir notre base de données avec l’ensemble des parties fournies par Lichess. L’unique technologie mentionnable utilisée dans ce programme est JDBC. Cette dernière nous permet d’insérer les parties, chargées en mémoire vive depuis les fichiers pgn, dans notre base de données.

#### **1.1.4 API de Lichess**

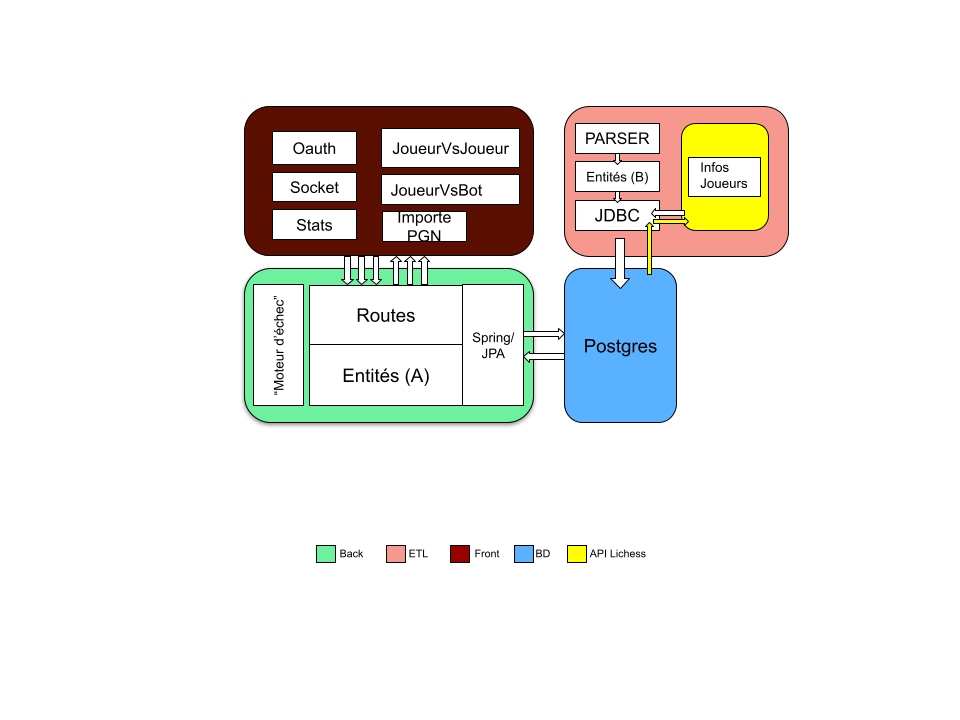
L'API Lichess permet de récupérer des informations structurées sur les joueurs, leurs statistiques, résultats, et parties passées, enrichissant ainsi notre base de données.

#### **1.1.5 Base de Données**

Nous avons choisi PostgreSQL pour sa performance, scalabilité, et fiabilité. Il gère efficacement les données structurées et semi-structurées, assurant la sécurité et l'intégrité des données tout au long du cycle de vie.

### **1.2 Schéma de Dépendances Global**

Le schéma de dépendances montre la liaison entre les différentes parties du projet, garantissant une intégration harmonieuse entre le frontend, le backend, le programme ETL, l'API de Lichess, et la base de données.



## 

## **2 - Retour d’expérience sur l’environnement de développement et les choix techniques**

### **2.1 Git**

L'implémentation de Git au sein du groupe s'est faite de manière assez naturelle. Un avantageux transfert de connaissance a pu se réaliser, où ceux qui maîtrisaient l'outil ont initié ceux qui ne le maîtrisaient pas aux bonnes pratiques et manières de faire, afin que chacun puisse mettre en commun ses travaux efficacement. De plus, certaines branches ont déjà dû subir un « merge », et peu, voire aucun conflit majeur n'a eu lieu lors des fusions.

Cependant, certains aspects peuvent encore être améliorés, comme une utilisation plus poussée des fonctionnalités proposées par Git. Le système d'*issue*, par exemple, permet d'obtenir un suivi de l'avancement d'une tâche en particulier en y associant une branche spécifique avec la possibilité d'écrire des notes d'activité, des commentaires ou de téléverser des fichiers, qui seront vus par les développeurs affectés à l'*issue* en question. Il en va de même pour le système des *Merge Request*, qui permet non seulement d'affecter des « rewiewer » qui auront pour tâche de valider le code du développeur, mais qui en plus permet de suivre en direct l'évolution (les commits) de la branche de feature par rapport à la branche originelle en affichant les conflits potentiels qui pourraient survenir en cas de fusion. Bien que cela semble un peu accessoire, ce genre de fonctionnalités pourraient, une fois prises en main, être une réelle valeur ajoutée.

### **2.2 Docker**

L’utilisation de Docker s’est avérée judicieuse pour de nombreuses raisons. Cette technologie nous a permis de distribuer des environnements d’exécution aux membres de l’équipe sans que ces derniers n’aient besoin d’installer d’outils supplémentaires sur leurs machines. De plus, ces environnements sont utilisables à l’aide de deux commandes: *docker compose up* et *docker compose down* rendant leur utilisation très simple. L’isolation des différents services dans des containers à éliminer les éventuels conflits provoqués par une implémentation sur des machines dont les environnements diffèrent.

La mise en place fut également simple. Docker est une technologie facile d’accès et documentée. Cet outil nous offre une modularité sur les différents services composant notre application. De ce fait, il nous sera aisé de regrouper ces derniers (répartis sur ETL, back, front) pour composer notre application complète. La spécification des versions de chaque service nous permet de contrôler ces dernières limitant alors les problèmes de mises à jour indésirées.

### **2.3 Technologies Front**

Ce qui a fonctionné :

Le composant React ChessGameWithBotApp permet de jouer contre le bot Stockfish. L'initialisation de Stockfish, la mise à jour de l'échiquier après chaque coup, la gestion du chronomètre pour les deux joueurs, ainsi que l'affichage des historiques de coups et de l'ouverture, fonctionnent globalement bien.

Ce qui n’a pas fonctionné :

La mise à jour de l’ouverture reste incomplète et ne reflète pas l'ouverture jouée par les blancs et les noirs de manière précise.

Quelques fonctionnalités (comme l'ajout dynamique de l'ECO dans le fichier PGN) ne sont pas encore implémentées.

Solutions à mettre en place pour améliorer :

Gestion des ouvertures : Intégrer une bibliothèque ou un fichier de données permettant de détecter et d'afficher les ouvertures en fonction des coups joués (par exemple, une base de données PGN pour correspondances), soit notre BD ChessOpening.

Ajout des métadonnées ECO : Automatiser l'identification du code ECO et son intégration dans les métadonnées PGN.

### **2.3 Technologies ETL**

Ce qui à fonctionné :

Nous avons opté pour des technologies que nous connaissons bien ce qui a rendu le développement de l’ETL très compréhensible et intuitif. Après plusieurs mises à jour, nous avons réussi à obtenir un ETL plutôt performant.

Ce qui n’as pas fonctionné :

Cependant, les expressions régulières sont une potentielle source d’erreur car les fichiers de la base de données Lichess sont étalés sur 10 ans, ce qui rend leur contenu différent d’une année à l’autre. Et d’un autre côté, le temps d’exécution est assez long, ce qui peut être pénible pour certains tests.

Solution à mettre en place pour améliorer :

Pour résoudre le problème du temps d’exécution serait d’optimiser le développement de la pipeline séquentielle puis de la “clusterisée” afin de répartir la charge de travail (que ce soit pour la lecture ou la transformation ou l’insertion).

### **2.4 Technologies Back**

L’utilisation de Spring Boot s’est avérée une excellente idée car il simplifie considérablement le développement d’applications en offrant des outils intégrés pour configurer rapidement un serveur web, gérer les dépendances avec Maven, et établir des APIs REST. Sa compatibilité avec Spring Data JPA et Hibernate a facilité le mapping des entités Java avec les tables de la base de données, tout en automatisant les opérations courantes. De plus, Spring Boot suit une architecture structurée (Controller, Service, Repository, Entity), ce qui rend le code clair, maintenable et extensible. Enfin, sa vaste documentation et communauté ont été d'une grande aide.

Mis à part la gestion du temps, notamment sous-estimé pour l’apprentissage des technologies comme Spring Boot et l’ API REST, tout s’est déroulé comme prévu.

Pour améliorer la gestion du projet, il serait important de réaliser une planification plus réaliste en tenant compte des temps nécessaires pour l’apprentissage des technologies, comme Spring Boot et les API REST. Il serait également utile de renforcer la formation initiale sur ces outils afin d’en accélérer la prise en main. Une meilleure répartition des tâches et une communication efficace entre les membres de l’équipe pourraient éviter les blocages. Enfin, utiliser l'outil de gestion de projet Openproject de façon plus efficace permettrait de suivre l’avancement en temps réel et d’ajuster les priorités si nécessaire.

## **3 - Retour d’expérience sur l’organisation du groupe**

### **3.1 Planification et distribution des tâches**

#### **Ce qui a fonctionné**

**Établissement d’un planning global** : Nous avons défini dès le départ un planning global des grands jalons du projet, ce qui a permis d’avoir une vision d’ensemble des étapes principales à réaliser. Bien que cette planification n’ait pas été détaillée, elle a offert une structure minimale pour guider les efforts collectifs.

**Avancement global malgré une planification sommaire** : Les tâches globales identifiées ont été prises en charge par les membres du groupe en fonction de ce qui avait été défini dans notre premier rapport. Cette approche, bien que non optimale, a permis d’avancer sur le projet.

**Utilisation d’OpenProject** : Nous avons utilisé OpenProject, un outil complet pour la planification et le suivi des tâches. Cependant, notre utilisation a été limitée, et nous n’avons pas exploité toutes ses fonctionnalités.

**Priorisation des tâches critiques** : Nous avons su identifier et prioriser les tâches dont d’autres dépendent, notamment la conception et le développement de l’ETL. Cela a permis de structurer les efforts autour des livrables les plus importants.

**Adaptabilité** : Malgré le manque de structuration, nous avons su nous adapter et orienter les développements lorsque cela était nécessaire, montrant une capacité de réaction efficace face aux imprévus.

#### **Ce qui n’a pas fonctionné**

**Absence de planification détaillée** : Si un planning global a été établi, aucun détail précis n’a été ajouté concernant les tâches individuelles, leurs délais ou les ressources nécessaires. Cela a conduit à une répartition improvisée des tâches, avec des membres prenant des responsabilités "à la volée" selon leurs disponibilités.

**Manque de visibilité globale** : L’absence de suivi régulier et structuré a rendu difficile la compréhension de l’état d’avancement global du projet. Les membres avaient parfois une perception partielle ou incomplète de la progression collective.

**Utilisation limitée et inefficace d’OpenProject** : Bien que cet outil ait été choisi pour la gestion des tâches, il s’est révélé lent et quelque peu complexe à utiliser. En conséquence, il a été sous-exploité, ce qui a réduit son utilité pour le suivi et la coordination.

**Manque de synchronisation des efforts** : Bien que nous ayons avancé collectivement, l'absence de structure claire a occasionné une progression moins fluide, avec des moments où certains membres étaient en attente d'autres.

#### **Solutions à mettre en place pour améliorer la planification des tâches**

1. **Adopter une méthode simplifiée et adaptée**

Utiliser le **tableau Kanban** seul d’OpenProject, plus intuitif; pour suivre les tâches à faire, en cours, et terminées.

Décomposer les grandes tâches en sous-tâches détaillées, avec des descriptions claires et un suivi précis.

1. **Introduire des délais réalistes et des responsabilités claires**

Associer chaque tâche à une **deadline réaliste** et désigner un **responsable identifié**, afin d’améliorer la gestion des priorités et d’assurer un meilleur suivi.

1. **Favoriser une meilleure visibilité**

Mettre en place un tableau de bord collaboratif qui montre l’état global d’avancement du projet et les tâches en cours, afin que chaque membre puisse facilement suivre la progression collective.

En appliquant ces solutions, la planification des tâches pourra devenir plus claire et efficace, permettant une gestion plus fluide et mieux coordonnée pour les projets à venir.

### **3.2 Communication**

#### **Ce qui a fonctionné**

La communication au sein du groupe s’est appuyée principalement sur deux canaux : **Discord** pour les échanges en ligne et les **réunions physiques** pour des interactions directes. Malgré une utilisation irrégulière, Discord s’est révélé plus accessible et flexible que Mattermost, car il ne nécessite pas de connexion VPN, ce qui a simplifié et fluidifié la tenue des échanges.

Les réunions physiques ont constitué le socle de notre communication, permettant à chaque membre de participer activement. Ces moments d’échanges directs ont permis une coordination efficace et une implication collective.   
Par ailleurs, l’utilisation de Discord pour la communication en temps réel a aidé à résoudre certains problèmes concrets liés au développement.

#### 

#### **Ce qui n’a pas fonctionné**

Cependant, plusieurs limitations ont impacté la fluidité de nos échanges :

**Manque de communication hors réunions physiques** : Durant les périodes de vacances ou lorsque nous travaillions à distance, les interactions étaient limitées. Cela a entraîné des mises à jour sporadiques sur l’avancement du projet, rendant difficile un suivi continu.

**Organisation sur Discord** : Bien que pratique, l’utilisation d’un seul canal pour tous les échanges a rapidement conduit à une surcharge d’informations, rendant les discussions moins claires.

**Absence de règles claires de communication** : Le groupe n’a pas établi de directives spécifiques, comme la fréquence des mises à jour ou les délais de réponse attendus, ce qui a parfois nui à l’efficacité globale du projet.

#### **Axe d’amélioration**

**Structurer les canaux de communication sur Discord**

Créer des **canaux thématiques** pour organiser les échanges selon les différents aspects du projet, tels que :

Développement (questions techniques, bugs, solutions proposées)

Organisation (planning, échéances)

Documentation (partage de fichiers et notes)

Discussions générales.  
Cela permettra d'éviter les discussions brouillonnes sur un seul canal et de retrouver plus facilement des informations spécifiques.

**Encourager la communication asynchrone**

Introduire des pratiques favorisant les échanges en différé, comme l’utilisation de courriels ou de messages résumant les avancées et décisions importantes.

Prévoir un moment fixe chaque semaine pour que chaque membre partage une **mise à jour individuelle** sur l’état d’avancement de ses tâches (via Discord ou un autre outil collaboratif). Cela garantira une visibilité continue, même lorsque les membres travaillent à distance ou pendant les périodes moins actives.

**Instaurer des règles de communication claires**

Préciser les outils et situations pour les interactions (par exemple, Discord pour les discussions courantes, réunions physiques pour les prises de décision importantes).

Encourager une culture de **signalement** lorsqu’un membre rencontre une difficulté ou un blocage, afin que le groupe puisse intervenir rapidement.

## **4 - Nouvelles fonctionnalités**

Au cours de ce semestre, nous allons implémenter diverses fonctionnalités afin d'affiner et d'améliorer notre projet. Une restructuration sera donc nécessaire.

Dans un premier temps, nous nous concentrerons sur le développement de notre propre moteur d'échecs, dont l'implémentation est déjà bien avancée. Parallèlement, nous travaillerons sur la mise en place de notre propre système d'authentification ainsi que sur la refonte des pages de l'application.

Une fois le moteur achevé, nous devrons nous atteler à la ré-implémentation de la partie réseau en utilisant des sockets, d'abord en local, puis, pourquoi pas, en ligne.

Enfin, nous conclurons par le portage de notre projet en application mobile.

### 4.1 Nouveau moteur d’échec

*Nous avons notre propre moteur d’échec, alors pourquoi en faire un nouveau?*

Cette question nécessite de comprendre l’un de nos objectifs de ce semestre.

En effet, nous avons vu au cours du semestre précédent que notre algorithme *Minimax* est plutôt “mauvais”. Il ne regarde qu’à une profondeur de 4 en quelques secondes. Le problème ne réside pas de l’algorithme en lui-même mais bien du moteur.

Pour comprendre, il faut savoir que Minimax fonctionne principalement vite grâce à la génération des coups légaux de notre moteur. Or, le moteur précédent était très moyen du fait de son implémentation en java avec les tableaux en 2D, ce qui rend le moteur très lourd, sans compter le fait que chaque vérification de coups, utilisait des copies de l’échiquier.

De ce fait, durant ce semestre, nous allons réécrire tout notre moteur d’échec afin de pouvoir générer le plus de coups possibles à la seconde, afin d’ultérieurement, pouvoir améliorer notre algorithme Minimax.

#### 4.1.1 Bitboard

Premièrement, nous allons complètement changer notre architecture. En effet, nous allons utiliser les *Bitboards*.

Les Bitboards sont une manière très efficace de représenter l’échiquier sous forme de bits.

Un échiquier contenant 64 cases, pour le représenter il suffit d’avoir un **long** en binaire de la forme suivante: **0b000000000000[...]00L** (le bit de poid faible étant A1 et celui de poid fort H8, les cases de notre échiquier).

Pour ajouter une pièce dans notre représentation de l’échiquier, il suffira simplement de mettre le bit voulu à 1.

Mais cela ne suffit pas à savoir quelle est la pièce à telle position. Pour cela, nous utiliserons plusieurs **long** représentant chacun des types de pièces. Une fois cela fait, nous pourrons utiliser les opérateurs **Bitwise** afin d’obtenir sur notre représentation, le type de pièce correspondant.

*Exemple:*

**long plateau = 0b00001111L**

**long fou\_noir = 0b00001001L**

En faisant l’opération **&** sur les deux long, on obtiendrait la position des fous noirs sur notre plateau.

L’avantage premier des Bitboards, est justement l’utilisation des **opérations logiques binaires** comme le ET, OU, XOR,...

En effet, l’utilisation de ces opérateurs, qui effectuent directement cela sur des bits, rend cela très rapide pour des processeurs modernes.

#### 4.1.2 Matrice d'attaque

Après avoir vu les Bitboards, nous allons continuer dans l’optimisation du moteur. En effet, nous allons générer les matrices d’attaques de chaque joueur afin de regarder si telle pièce est attaquée ou non. En bitboard, nous pouvons non seulement avoir une variable qui stocke chacune des pièces mais on peut tout à fait stocker leurs attaques.

Dans une représentation bitboard, une **matrice d'attaque** pour un type de pièce est simplement un bitboard où chaque bit activé représente une case attaquée par cette pièce ou ce camp.

Cela va permettre de mettre à jour les positions attaquées beaucoup plus rapidement que de recalculer les attaques à chaque fois.

Pour savoir comment une pièce attaque, il est nécessaire de comprendre le décalage de bit.

Prenons l’exemple du cavalier, celui-ci peut bouger en ‘L’. Dans une représentation de l’échiquier sous forme de Long, cela reste aussi simple de comprendre comment effectuer une attaque. Voici, ci-dessous, un Long que l’on représente sous forme plus naturelle. (*‘0’ bit de poid faible et ‘63’ bit de poid fort*)

**56 57 58 59 60 61 62 63**

**48 49 50 51 52 53 54 55**

**40 41 42 43 44 45 46 47**

**32 33 34 35 36 37 38 39**

**24 25 26 27 28 29 30 31**

**16 17 18 19 20 21 22 23**

**8 9 10 11 12 13 14 15**

**0 1 2 3 4 5 6 7**

**Représentation d’un échiquier**

**selon l’ ordre des bits**

Imaginons un cavalier à la case **28** en rouge, et en bleu les positions à laquelle il peut aller.

Pour lui donner ses attaques, il suffit de connaître la position du cavalier et d’effectuer les décalages de bits suivants: 43 - 28 = **15**; 45 - 28 = **17**; 34 - 28 = **6**; 38 - 28 = **10**

et l’inverse pour les positions nécessitant de reculer.  
Soit pour aller vers le haut du plateau, on utilisera l’opérateur **<<** et pour descendre **>>**.

Le *Long* qui en résulterait pour la matrice d’attaque du cavalier à la position **e4** serait de la forme suivante: **0b0000000000010100001000100000000000100010000101000000000000000000**

Cela va générer les attaques d’une pièce très rapidement afin de pouvoir ensuite regarder s’il peut attaquer une pièce ennemie en faisant un **&** entre ses attaques et les pièces ennemies.

#### 4.1.3 Magic bitsboard

Pour faciliter la génération des attaques de pièces, nous pouvons encore optimiser le moteur en utilisant les **Magics Bitboard**.

Les **magic bitboards** sont une technique avancée utilisée dans les moteurs d’échecs pour calculer efficacement les mouvements des pièces à longue portée, comme les **fous**, les **tours**, et les **dames**. Ils permettent de générer des attaques en diagonales et ligne droite de manière très rapide et compacte.

Bien que cela semble encore abstrait, il faut comprendre l’utilité. Sans magics bitboard, pour générer les attaques distantes, il faut regarder dans chaque direction que si on avance d’une case, on ne rencontre pas un obstacle, et recommencer pour une autre direction… Et cela à chaque coup effectué.

L’idée des Magics Bitboards est l’utilisation d’un **magic number** afin de calculer un index unique dans une table de résultat précalculé.

Ces nombres doivent être soigneusement choisis pour que la multiplication binaire et un décalage produisent un index sans collisions dans une table de résultats.  
Les tables pré-calculées sont pour chaque position et chaque direction possible ,toutes les générations d’obstacles possibles.

Ces résultats sont ensuite indexés en utilisant le magic number.

Pendant l’exécution, au lieu de recalculer les attaques à chaque fois, on génère un index unique pour la situation actuelle en combinant le bitboard des obstacles et le magic number.

Ce processus accède directement au bon résultat dans la table.

Cela va permettre au moteur de ne calculer qu’une seule fois les attaques possibles et derrière de ne regarder que la table pré-calculé afin de connaître les cases d’attaques possibles.

#### 4.1.4 Temps estimé

Afin de produire un nouveau moteur d’échec, il est nécessaire de comprendre comment fonctionnent toutes les fonctionnalités envisagées dans ce nouveau moteur avant de pouvoir commencer à le produire.

Ensuite, il faudra implémenter ses fonctionnalités une à une en faisant des classes de Tests à chaque fois afin de vérifier que celle-ci fonctionne bien.

Nous avons décidé de mettre environ 100 heures sur cette partie afin de pouvoir programmer toutes ces fonctionnalités.

#### 4.1.5 Dépendance entre les fonctionnalités

Ultérieurement, nous aurons des routes permettant de lier le Front au Back. En effet, on pourrait, en tant qu’utilisateur, pouvoir demander le meilleur coup possible. Et, dans ce cas, on enverrait de part la route, une demande de recherche du meilleur coup. Ce coup sera ensuite transmis à l’aide des routes afin d'être récupéré par le Front.  
On pourrait imaginer différentes routes comme pour l’initialisation du plateau, afin de faire la synchronisation avec le moteur d'échec dans le Back.  
De même, une route qui renvoie les coups légaux afin de savoir si le coup joué est autorisé ou non, l'exécution des coups et autres routes utiles au bon fonctionnement de l'application.

### 4.2 Minimax

Comme vu au-dessus, une fois notre nouveau moteur fait, nous verrons afin de réimplanter notre *Minimax* avec de nouvelles fonctionnalités permettant d’avoir des recherches toujours plus poussées et une profondeur plus grande.

Actuellement, notre algorithme ne recherche que pour une profondeur de 4, ce qui est très peu pour un moteur. Bien que celui-ci puisse affronter des bots à **1500** elo avec une moyenne de **75%** de précision, nous essaierons d’améliorer ce résultat afin d’avoir un objectif de 7 en profondeur. Cela représenterait une amélioration gigantesque.

En effet, à **7** de profondeur, le nombre de coups possibles est de **3,195,901,860** pour l’échiquier en début de partie contre seulement **197,281** coups pour une profondeur de **4**. Normalement, à l’aide de notre futur moteur et des autres fonctionnalités que nous rajouterons à notre Minimax, cela semblerait possible.

#### 4.2.1 Alpha-Bêta

Nous allons implémenter une meilleure version de notre algorithme Minimax avec l’aide de différentes méthodes comme l’**élagage alpha-bêta**.

L'élagage alpha-bêta réduit le nombre de nœuds évalués dans l'arbre de décision par Minimax, sans affecter le résultat final. Voici comment ça fonctionne:

**Alpha** représente la valeur minimale garantie que le joueur *maximisant* peut obtenir.

**Bêta** représente la valeur maximale garantie que le joueur *minimisant* peut obtenir.

Lors de l'exploration de l'arbre de décision, si l'algorithme trouve un mouvement qui est clairement pire que les mouvements évalués précédemment, par rapport à alpha et bêta, il coupe (élague) cette branche de l'arbre, car elle n'influencera pas le choix final.

Si, bien implémenté, cela devrait couper un grand nombre de branches inutiles et permettre une évaluation plus rapide.

#### 4.2.2 Quiescence Search

La recherche de quiescence prolonge l'évaluation au-delà de la profondeur 0 en continuant à explorer uniquement les **coups de capture** et les **échecs**, jusqu'à ce que la position atteigne un état **stable** ou "quiescent". Une position est considérée comme *quiescente* lorsqu'elle ne contient pas de coups tactiques imminents qui pourraient changer drastiquement l'évaluation de la position.

En évitant les évaluations basées sur des positions non quiescentes, l'algorithme peut fournir des évaluations plus stables et réalistes.

On limite l**'effet d'horizon** qui se produit lorsque l'algorithme ne voit pas au-delà d'une certaine profondeur, ce qui peut conduire à des évaluations erronées. La recherche de quiescence aide à atténuer ce problème en prolongeant l'exploration pour inclure des coups tactiques critiques.

#### 4.2.3 Moves Ordering

L'ordonnancement des coups consiste à explorer les **mouvements** les plus **prometteurs en premier**, ce qui permet à l'algorithme d'*élagage alpha-bêta* de couper plus efficacement les branches non pertinentes de l'arbre de recherche. En trouvant rapidement de bons mouvements, l'algorithme peut établir des bornes de coupure (alpha et bêta) plus strictes, réduisant ainsi le nombre de nœuds à explorer.

Par exemple, un ordre de tri possible serait de mettre en premier les *échecs et mat*, ensuite les *captures*, les *promotions* et les *autres coups*.

Cela permettra, comme dit plus haut, de couper les branches beaucoup plus rapidement et de pouvoir accélérer la recherche de notre Minimax.

#### 4.2.4 Table de Transposition

Une table de transposition est essentiellement une grande table de hachage qui enregistre les positions déjà explorées pendant la recherche, en soit c’est de la **mémoïsation,** commevue avec Mr. Bailleux.

Chaque entrée dans la table contient une position de jeu et des informations associées telles que la valeur évaluée de cette position, la profondeur de recherche à laquelle elle a été évaluée et les coups optimaux trouvés.

Lorsque l'algorithme rencontre une position déjà évaluée, il peut directement utiliser les résultats stockés dans la table de transposition au lieu de recalculer l'évaluation. Cela économise beaucoup de temps de calcul.

En réutilisant les résultats des positions précédemment évaluées, l'algorithme peut établir des bornes plus strictes pour les coupures alpha et bêta, ce qui permet d'élaguer plus de branches inutiles.

#### 4.2.5 Hachage de Zobrist

Le hachage de Zobrist est une technique spécifique utilisée pour générer un identifiant unique pour chaque position possible dans un jeu comme les échecs. Cette méthode est particulièrement efficace car elle permet de calculer rapidement des identifiants uniques, même après de petits changements dans la position.  
On l’utilisera à l’aide des tables de transposition, en effet, au lieu de stocker les positions actuelles de l’échiquier, on hachera le plateau afin de le stocker en le compressant.

#### 4.2.6 Livre d’opening

Pour finir, nous essaierons de développer un **livre d’opening**. Cela consiste à donner un ‘opening’ à notre recommandation de coups afin de ne pas avoir toujours les mêmes répétitions en termes de coups recommandés au début de partie.

En effet, on sait par exemple que le coup le plus répandu pour les blancs est “e4”, on peut proposer en retour “Nf6” ou bien “d5”. Cela se baserait sur un fichier texte qui récupérait les coups les plus joués par les joueurs ‘pro’ d'échecs.

#### 4.2.7 Temps estimé

La constitution d’un *Minimax* solide et précis est une tâche plutôt laborieuse en termes de temps nécessitant de jouer plusieurs parties afin de s’assurer qu' il génère des bons mouvements dans pleins de situations différentes.

En termes de codage cela devrait tourner dans les alentours de 50 heures, et pour les tests, une *50-aines* de tests seraient suffisant.

Nous pourrons, ultérieurement tester notre ancien *Minimax* avec notre futur afin de suivre les progrès de celui-ci.

#### 4.2.8 Dépendances entre les fonctionnalités

De la même manière que le moteur d'échec, nous utiliserons des routes grâce à **Spring** afin de pouvoir faire communiquer le **Front** avec le **Back**, cela permettra de pouvoir calculer le meilleur coup pour un utilisateur dans le Back et le retourner sur le Front.

### 4.3 Détection de triche

L’ajout d’un système de détection de triche doit se faire par l’intermédiaire d’un protocole qui permet de déterminer si un joueur triche ou non. Pour cela on peut se baser sur ce qui se fait déjà dans le monde des échecs en ligne et même dans les compétitions officielles.

Une approche répandue est de créer une note de suspicion qui, si elle atteint un trop haut niveau, indique que nous sommes en présence d’un tricheur. Cette note se base sur beaucoup de critères, pour en citer une liste non exhaustive : la précision des coups, l’elo du joueur au cours du temps, sa vitesse de jeu, schéma de répétition dans ses coups avec ses précédentes parties, son pourcentage de victoire, etc.

On peut alors donner un poids à chacun de ces critères en fonction de leur importance et en faire une somme pondérée qui reflète le degré de suspicion d’un joueur.

Cependant, cette méthode n’est pas infaillible. Certains joueurs très compétents ou utilisant des algorithmes de triche subtils peuvent passer sous le radar de ce système. En effet, si l'algorithme de triche est conçu pour imiter parfaitement les décisions humaines tout en restant cohérent avec des niveaux de jeu élevés, la détection peut devenir plus difficile.

En somme, la détection de triche est une fonctionnalité complémentaire visant à améliorer à la fois la sécurité et la convivialité de l’application. Cet outil doit être équilibré de manière à garantir une expérience de jeu équitable et enrichissante pour tous les utilisateurs.

### 4.4 Jeu en réseau et socket sur nouveau moteur

Une fois la mise en place terminée de notre moteur, nous allons ensuite utiliser celui-ci pour pouvoir disputer des parties et en réseau et en socket.

#### 4.4.1 Socket

Nous allons implémenter un système de jeu en réseau en utilisant les WebSockets. Ces WebSockets permettront une communication bidirectionnelle et persistante entre le client et le serveur, offrant ainsi une expérience de jeu fluide et réactive. Grâce aux routes *Spring* préalablement créées, nous pourrons recevoir les coups légaux, les mises à jour de l'échiquier et toutes les informations nécessaires pour le bon déroulement des parties en temps réel.

#### 4.4.2 Réseau

Pour le jeu en réseau, nous hébergerons notre backend sur un serveur. Cela permettra aux joueurs de se connecter au serveur et de disputer des parties d'échecs en temps réel contre d'autres joueurs. Chaque joueur pourra ainsi se connecter, créer ou rejoindre des parties, et voir l'état de l'échiquier se mettre à jour instantanément à chaque coup joué.

En utilisant les WebSockets et un serveur dédié, nous pouvons assurer que le jeu en réseau sera rapide, efficace et agréable pour les utilisateurs.

#### 4.4.3 Dépendances entre les fonctionnalités

Les dépendances de cette partie incluent principalement les routes de communication entre le Frontend et le Backend. Une fois le moteur d'échec finalisé, ces routes permettront de relier les deux, en gérant les demandes des utilisateurs comme l'initialisation du plateau, la vérification des coups légaux et l'exécution des coups. Pour le jeu en réseau, le backend hébergé sur un serveur utilisera les WebSockets pour maintenir une communication en temps réel et sans latence entre les joueurs. Chaque fonctionnalité dépend de la précédente pour garantir une expérience utilisateur cohérente et performante.

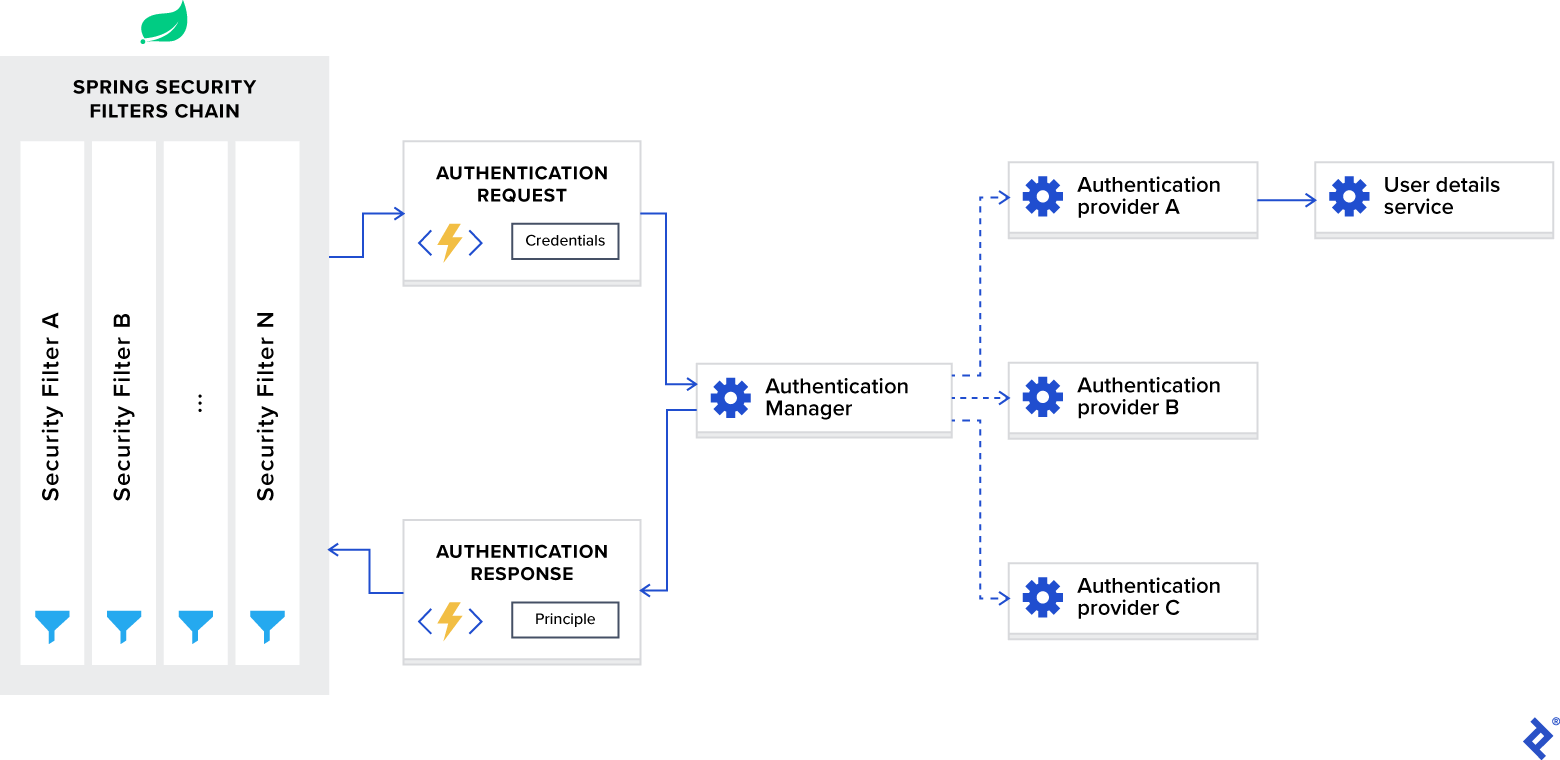
### 4.5 Authentification

#### 4.5.1 Description

Afin de garantir la sécurité et une gestion optimale des utilisateurs, nous avons décidé de mettre en place notre propre système d’authentification. Ce choix nous permet d’assurer un contrôle total sur les mécanismes de connexion, de protection des données et de gestion des permissions. Chaque utilisateur disposera d’un compte sécurisé, protégé par des identifiants uniques et un mot de passe crypté.

#### 4.5.2 Technologie

**Spring Security :** Spring Security est une bibliothèque puissante et flexible dédiée à la sécurité des applications Spring Boot. Elle nous permettra de gérer l'ensemble des mécanismes d'authentification et d'autorisation. Grâce à ses fonctionnalités intégrées, nous pourrons configurer facilement des filtres de sécurité, des règles de gestion des sessions, et des protections contre les attaques courantes.



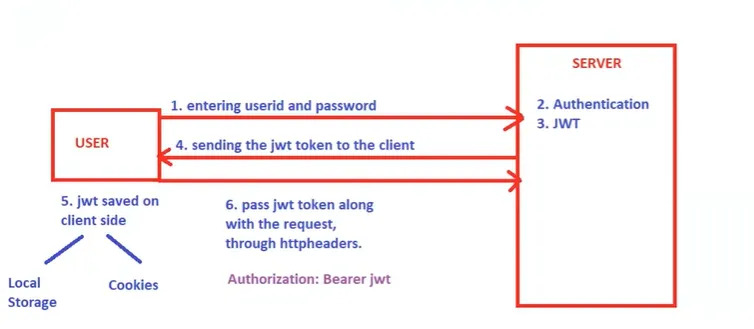
Lorsqu’on intègre **Spring Security** dans une application, une chaîne de filtres est automatiquement configurée. Elle intercepte toutes les requêtes entrantes pour appliquer des mécanismes de sécurité (authentification, autorisation, etc.), chaque filtre gérant un aspect spécifique.

L'**AuthenticationManager** est un orchestrateur qui délègue les requêtes d'authentification à un ou plusieurs **AuthenticationProvider**, en fonction du type d'authentification demandé. Un **AuthenticationProvider** gère un type spécifique d'authentification. Deux méthodes principales sont fournies :

* **authenticate** : Traite la logique d'authentification
* **supports** : Vérifie si le type d’authentification est pris en charge.

Un exemple courant est le **DaoAuthenticationProvider**, qui utilise un **UserDetailsService** pour récupérer les données utilisateur.Le **UserDetailsService** est une interface centrale utilisée pour charger les données utilisateur. Elle expose une seule méthode **loadUserByUsername** prenant un nom d'utilisateur en paramètre et renvoyant un objet contenant les détails de l'utilisateur. Elle est souvent utilisée pour récupérer les informations depuis une base de données avant de valider l'authentification.

**Token d'authentification (JWT) :**  
 Nous mettrons en place un système basé sur des JSON Web Tokens (JWT) pour gérer les sessions des utilisateurs. Une fois l'utilisateur authentifié via la route de connexion (/login), un token unique sera généré et envoyé en réponse. Ce token, signé avec une clé secrète, permettra de garantir l'intégrité et l'authenticité des requêtes provenant de l'utilisateur. Ce système réduit la dépendance aux sessions serveur adaptées aux API REST.



Le flux de travail avec un jeton JWT peut être illustré par l'exemple de la connexion à Facebook depuis un navigateur. L'utilisateur commence par saisir ses informations d'identification (email/nom d'utilisateur et mot de passe) et clique sur "Connexion". Le serveur vérifie alors ces informations, et si elles sont correctes, il génère un jeton JWT contenant des informations sur l'utilisateur (comme son identifiant ou son rôle), signé avec une clé secrète pour garantir son intégrité. Ce jeton est ensuite envoyé au client, qui le stocke dans un emplacement sécurisé, tel que le Local Storage ou les Cookies. Pour toutes les requêtes ultérieures, le client transmet ce jeton via l'en-tête HTTP **Authorization** sous le format Bearer <votre\_token\_JWT>. Le serveur, à chaque requête, décode et valide le jeton pour s'assurer de son authenticité et de sa validité. Si le jeton est valide, le serveur autorise l'accès aux ressources demandées, offrant ainsi un système stateless et sécurisé.

**Chiffrement des mots de passe :** Les mots de passe des utilisateurs seront systématiquement chiffrés avant leur stockage en base de données afin de protéger les données sensibles. Nous utiliserons l'algorithme *bcrypt*, reconnu pour sa robustesse et ses mécanismes de hachage adaptatif, rendant les attaques par force brute particulièrement difficiles. Avec bcrypt, chaque mot de passe sera chiffré, garantissant une sécurité accrue contre les attaques par dictionnaire.

**Deux routes POST :**

/register : Cette route permettra aux nouveaux utilisateurs de s'enregistrer. Lorsqu'un utilisateur soumettra ses informations (par exemple, email, nom d'utilisateur, mot de passe), le backend vérifiera leur validité, appliquera les règles de sécurité nécessaires (cryptage du mot de passe) et stockera les données en base.

/login : Cette route sera utilisée pour l'authentification. Après avoir soumis leurs identifiants (email/nom d'utilisateur et mot de passe), les utilisateurs recevront un token JWT en cas de succès. Ce token pourra être utilisé pour accéder aux ressources sécurisées de l'application.

#### 4.5.3 Dépendance

Les dépendances nécessaires pour mettre en œuvre ce système d'authentification incluent **Spring Security**, qui gère l'authentification, l'autorisation, et la sécurité globale de l'application, en particulier l'intégration avec des jetons JWT. Nous avons également besoin de **Spring Boot Starter Web**, qui permet d'exposer des routes REST telles que /login et /register, nécessaires pour l'inscription et l'authentification des utilisateurs. Enfin, le **frontend** joue un rôle essentiel en gérant le jeton JWT reçu après une authentification réussie. Ce jeton est stocké de manière sécurisée (dans le Local Storage ou les Cookies) et inclus dans les en-têtes des requêtes pour accéder aux ressources protégées, garantissant une communication fluide et sécurisée entre le client et le serveur.

### 4.6 Application mobile

Notre objectif est de créer une application mobile afin d’avoir une portabilité à portée de main.

*Accessibilité et Commodité*

**Jouer n'importe où** : Les utilisateurs peuvent jouer aux échecs à tout moment et n'importe où.

**Accessibilité** : Les applications mobiles permettent d'atteindre un large public, y compris ceux qui n'ont pas accès à un ordinateur.

*Apprentissage et Amélioration*

**Analyse en temps réel** : Les applications peuvent offrir des analyses de parties en temps réel, aidant les joueurs à comprendre leurs erreurs et à s'améliorer.

**Recommandations de coups** : Les joueurs peuvent recevoir des suggestions de coups, ce qui est particulièrement utile pour les débutants.

*Interaction Sociale*

**Jouer contre des amis ou des inconnus** : Les utilisateurs peuvent jouer contre leurs amis ou des joueurs du monde entier, favorisant ainsi les interactions sociales et la compétitivité.

*Flexibilité*

**Modes de jeu variés** : Les utilisateurs peuvent choisir entre différents modes de jeu, tels que blitz, bullet, ou parties longues.

**Personnalisation** : Les joueurs peuvent personnaliser l'interface, les thèmes de l'échiquier, et les niveaux de difficulté.

Nous étudions la possibilité d’effectuer le portage de notre application sur mobile. La conversion de notre front end rédigée avec React et TypeScript en APK semble pouvoir s’effectuer relativement aisément.. Cette approche nous éviterait de devoir recréer toute la logique de programmation développée au premier semestre. Il suffirait simplement de restructurer l’interface pour l’adapter au format mobile.

#### Questions à se poser

1. **Comment connecter le backend au front end de notre application mobile ?**
2. **Comment permettre à l'utilisateur sur son appareil mobile de se connecter au serveur back end ?**

**React Native:**

Pourquoi ce choix:

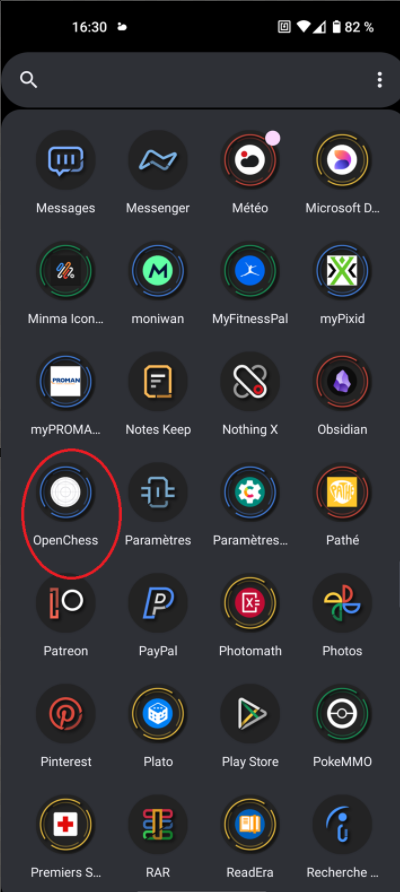
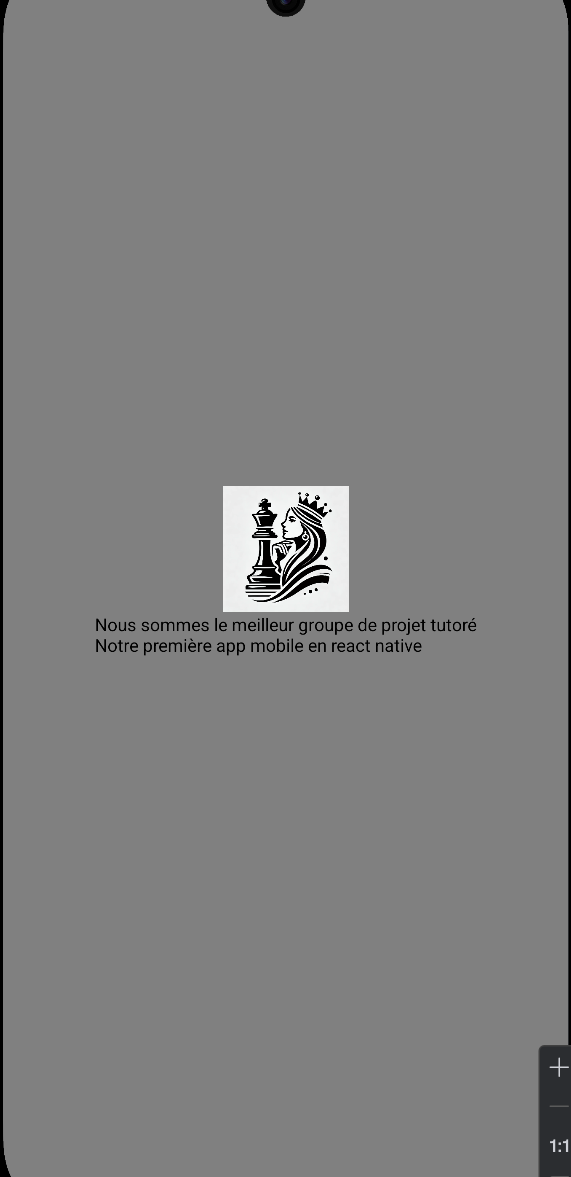
React avec TypeScript et React Native sont deux technologies liées mais destinées à des usages différents. **React avec TypeScript** est utilisé pour construire des applications web avec une typage statique fort pour améliorer la qualité et la maintenabilité du code. Il permet de créer des interfaces utilisateur dynamiques avec des composants réutilisables, tout en offrant une meilleure autocomplétion et vérification des erreurs grâce à TypeScript. **React Native**, en revanche, est un framework basé sur React qui permet de créer des applications mobiles natives pour iOS et Android. Il utilise des composants spécifiques aux plateformes mobiles, tout en conservant une syntaxe et une approche similaires à React.

Nous avons choisi **React Native** pour notre projet d'application mobile car notre code de base pour le frontend est déjà en React TypeScript. Cette transition nous permet de réutiliser une grande partie de notre logique d'application, tout en adaptant les composants UI pour les plateformes mobiles. Les avantages incluent une courbe d'apprentissage réduite, car nous continuons à utiliser des concepts et des outils familiers, et la possibilité de maintenir une base de code partagée entre le web et le mobile.

**Une apk/aab c'est quoi ?**

Une APK (Android Package Kit) est le format de fichier utilisé pour distribuer et installer des applications sur les appareils Android. Elle contient tous les éléments nécessaires pour exécuter l'application, comme le code, les ressources et les métadonnées. Les utilisateurs peuvent télécharger une APK directement depuis le Google Play Store ou d'autres sources pour installer une application sur leur appareil.

Un AAB (Android App Bundle) est un format de publication pour les applications Android introduit par Google. Contrairement aux APK, les AAB ne sont pas directement installables. Ils contiennent l'ensemble des ressources et du code de l'application, mais Google Play génère dynamiquement des APK spécifiques à l'appareil de l'utilisateur à partir de l'AAB lors du téléchargement. Cela permet d'optimiser la taille de l'application et de garantir une meilleure expérience utilisateur.

**Test réalisé, création d' une première application en react native:**  
(1) (2) 

**Application apk installer sur le téléphone de Maxence (téléphone physique)(1)  
Rendu de l’application test sur ce même téléphone (2)**

Actuellement nous avons contacté Mr.Meunier Charles chargé de l’UE programmation mobile pour lui demander de nous partager son cours, ainsi que des recommandations (en attente de son retour)..

### 4.7 Restructuration front/back

Après une analyse approfondie et de nombreuses réflexions, nous avons décidé d’entreprendre une restructuration de la partie front-end et back-end de notre application. Cette initiative s’inscrit dans une démarche d’amélioration continue, visant à optimiser l’expérience utilisateur tout en renforçant la robustesse et l’efficacité globale de notre système.

Du côté front-end, une refonte graphique majeure est prévue. L’objectif principal est de proposer une interface utilisateur nettement plus ergonomique, intuitive et visuellement attrayante. Nous souhaitons offrir une expérience plus fluide et engageante pour les utilisateurs finaux, en prenant en compte les derniers standards en matière de design et d’accessibilité.

Par ailleurs, une attention particulière est portée à l’amélioration de la communication entre le front-end et le back-end. Dans cette optique, de nouvelles routes sont progressivement implémentées afin d’enrichir les échanges et de mieux répondre aux besoins fonctionnels de l’application. L’un des chantiers prioritaires dans ce cadre est l’intégration d’un système d’authentification fiable et sécurisé, qui constitue une pierre angulaire de la sécurisation des données et de l’accès aux fonctionnalités sensibles de l’application.

Le but global de la restructuration est d’ôter le maximum d’intelligence à notre front end. Jusqu’à lors, ce dernier était responsable de notre authentification ou encore des parties JcJ. Ce type de tâche doit être effectué par notre application Spring. De ce fait, notre front-end sera bien plus léger et n’aura pour rôle que de permettre à l’utilisateur d'interagir avec notre application.

### 4.8 Modifier schéma BD

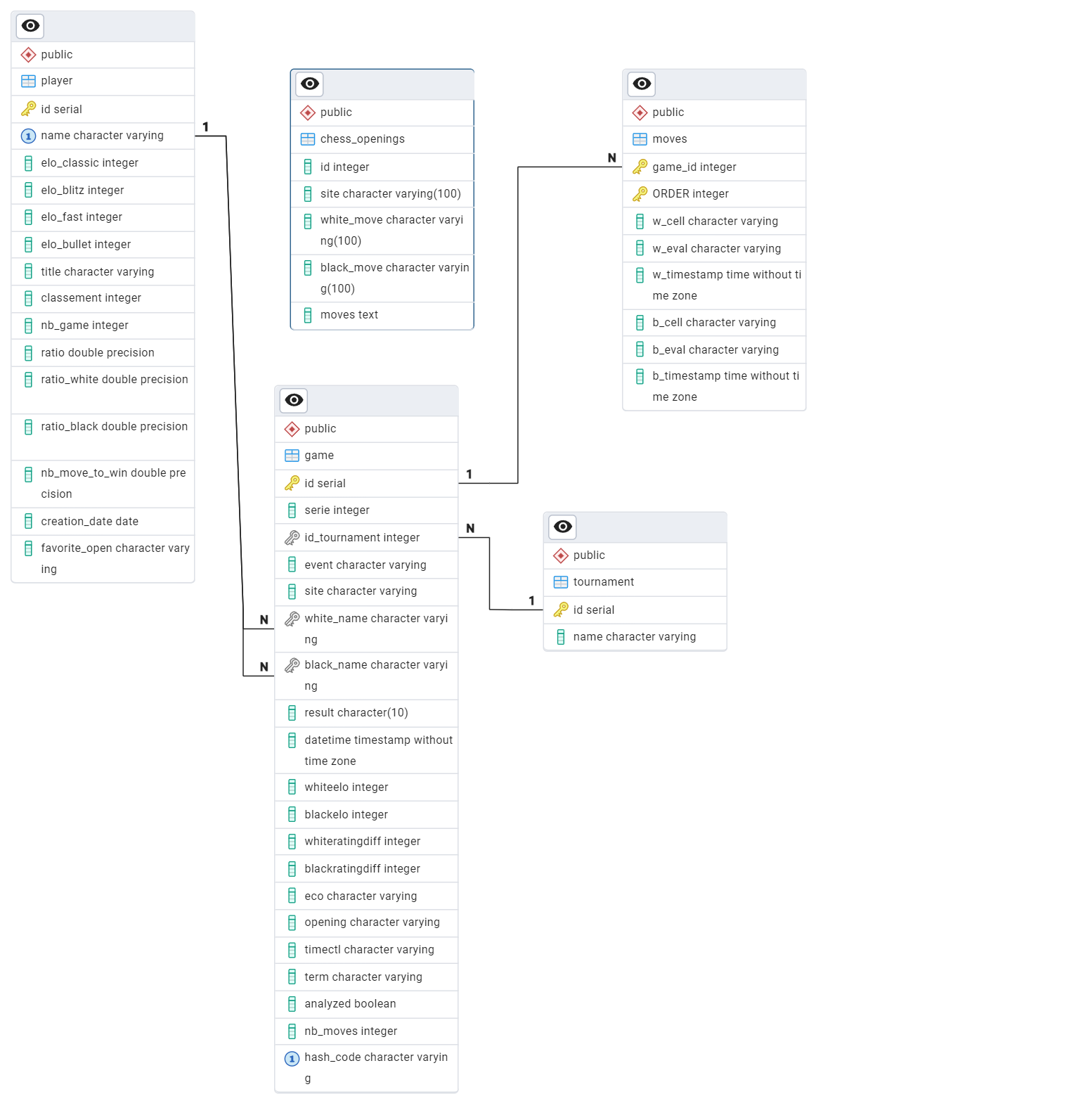
Dans le cadre de l'implémentation des nouvelles fonctionnalités de notre application, nous avons procédé à l'ajout d'un nouveau fichier de migrations Flyway afin de structurer et enrichir la base de données. Cette étape s’inscrit dans une démarche de mise en place d’une architecture relationnelle robuste et adaptée aux besoins croissants de notre projet.

Concrètement, cette migration a pour objectif d’introduire trois nouvelles tables, chacune jouant un rôle clé dans la gestion des utilisateurs et de leurs droits. La première table, dédiée aux utilisateurs, est conçue pour recenser les informations essentielles les concernant. Chaque utilisateur est ainsi défini par un identifiant unique (id), un nom d’utilisateur (username), une adresse email (email) et un mot de passe sécurisé (password). Ces données permettront de gérer efficacement les sessions et l’authentification au sein de notre application.

La deuxième table concerne les rôles. Elle a pour but de stocker les différents rôles attribuables aux utilisateurs, chacun d’eux étant caractérisé par un identifiant unique (id) et un nom (name). Cette table nous offre la flexibilité nécessaire pour gérer les autorisations et les niveaux d'accès de manière centralisée et évolutive.

Enfin, la troisième table joue un rôle de pivot entre les deux premières. Il s'agit d'une table d'association qui établit la relation entre les utilisateurs et leurs rôles. Elle est structurée autour de deux clés étrangères : la première pointe vers l’identifiant de la table des utilisateurs, tandis que la seconde fait référence à celui de la table des rôles. Grâce à cette conception, il devient possible de gérer efficacement des relations de type « plusieurs à plusieurs », permettant ainsi à un utilisateur de se voir attribuer plusieurs rôles et à un rôle d’être partagé par plusieurs utilisateurs.

Cette étape marque une avancée significative dans la structuration de notre application, renforçant à la fois la clarté et la cohérence de notre base de données, tout en posant les bases nécessaires pour intégrer de manière fluide les futures fonctionnalités de gestion des droits et permissions.



Par ailleurs, des tables supplémentaires sont envisagées car nécessaires pour l'implémentation de nouvelles fonctionnalités, comme le réseau social abordé dans la partie ci-dessous. Ces tables varieront en fonction de la solution que nous retiendrons — un système de « follow » ou un système d'ami classique. Il n'est pas à exclure également l'ajout d'attributs supplémentaires dans des tables déjà existantes pour proposer de nouvelles statistiques sur les parties et les joueurs. Les délibérations à ce sujet ne sont pas encore terminées au sein du groupe.

### 

### 4.9 Interactions entre utilisateurs

L’implémentation d’un pseudo réseau social dans notre application est envisagée. Ce dernier permettrait aux utilisateurs d'interagir entre eux. Les principales fonctionnalités seraient l’échange de messages privés entre utilisateurs, la consultation des profils des joueurs, l’invitation à des parties et un système de contact.

La consultation des profils d’utilisateurs nécessiterait un annuaire permettant de rechercher des utilisateurs à partir de leur pseudonyme. Pour ce faire, une nouvelle route se devra d’être implémentée dans notre API afin de rechercher ces derniers par expressions régulières (opérateur LIKE). De plus, une distinction entre profil utilisateur “public” et “personnel” se devra d’être implémentée. L’utilisateur pourra potentiellement modifier certains aspects de sa page personnelle tandis que la consultation de profils d’autres joueurs sera en lecture seule.

L’intégration de messages privés implique majoritairement une révision du schéma de données. Une table stockant le contenu, l’émetteur, le destinataire et un timestamp sera nécessaire afin de restituer les messages aux bonnes personnes dans l’ordre chronologique. Un index posé sur les émetteurs et récepteurs pourrait suffir à restituer les messages de manière réactive aux utilisateurs dès lors qu’ils consultent ces derniers. Un système de cache pourrait également être mis en place pour limiter les interrogations au SGBD (ou encore a Spring si ce dernier est implémenté dans le front). Les messages seraient alors stockés en mémoire vive et mis à jour dès lors que de nouvelles entrées sont produites.

De la même manière que pour les messages, l’implémentation des contacts nécessite une nouvelle table effectuant la correspondance. Deux possibilités sont envisagées, les utilisateurs peuvent se suivre unilatéralement (follow) ou s’ajouter en amis (accord mutuel). La première option semble la plus simple puisqu’elle ne doit pas gérer le cas intermédiaire ou une demande d’ajout est en attente. Néanmoins, chaque option devrait être relativement simple à implémenter.

La majorité du travail requis par ces différentes fonctionnalités sera soumise au front end afin d’intégrer ces dernières à l’interface utilisateur de manière à ce qu’elles soient accessibles depuis la majorité du site. Un système de notifications sera également nécessaire pour prévenir l’utilisateur et regrouper l’ensemble des évènements l’atteignant. Ces derniers comprendraient à ce jour les messages, invitations à des parties ou encore demande de contact.

### 4.10 Visualisation des données

Les statistiques de l’utilisateur n’étaient présentées uniquement sous forme de texte. Durant ce semestre, nous allons améliorer l'expérience utilisateur en proposant des statistiques au format graphique. Entre autres, l’elo de l’utilisateur sera représenté sous forme de courbe, permettant à l’utilisateur de mieux se rendre compte de l’évolution de son niveau de jeu, ou encore, le nombre de parties remportées, nulles ou perdues en tant que blanc ou noir sous forme d’histogramme cumulatif.

Au-delà des différentes données affichées sur l’ensemble du site comme les données sur la page personnelle d’un joueur, nous envisageons de construire un outil de visualisation de données. Cet outil permettrait aux utilisateurs d'interroger notre base de données selon des critères de sélection mis à leur disposition et de restituer les résultats sous la forme de leur choix. Une telle fonctionnalité soulève de nombreuses problématiques techniques.

Tout d’abord, l’utilisation des ressources de cet outil pourrait s’avérer intense. En effet, si un grand nombre d’utilisateurs souhaitent visualiser des données concernant l’ensemble des parties stockées dans notre base, de nombreux composants s’en trouveront surchargés détériorant grandement les performances de l’ensemble des services de notre application. La prise en charge de tels flux nécessiterait une duplication en parallèle de nos différents composants associées à une répartition de la charge. De plus, il serait judicieux de limiter la quantité de ressources qu’un utilisateur donné puisse solliciter ou encore de mutualiser les ressources communes demandées par des utilisateurs distincts. Néanmoins, ces problématiques engendrées nécessitent des solutions de taille dépassant certainement nos capacités ainsi que celles notre matériel. La question de la transmission et du traitement des données se pose également, il serait préférable que Spring se charge de préparer les données avant de les restituer au front end pour que ce dernier ne subissent pas de ralentissement. Or, l’interrogation de notre API suivie de la mise en forme des données semble être plus aisée avec React. L’implémentation de nombreuses routes spécifiques à l’outil de visualisation pourrait alors s’imposer. Des tests prématurés devront certainement être effectués afin d’affirmer notre approche.

De ce fait, il est envisageable d’ignorer ces problématiques ou encore de trouver des alternatives pouvant rendre l’utilisation de l’outil de visualisation viable. Ainsi, ce dernier pourrait être implémenté en dépit de performances acceptables. Comme dit précédemment, des solutions préventives pourraient être mises en place afin de limiter la potentielle charge générée par un utilisateur donnée. La mise en place de cet outil ne produira certainement pas un résultat satisfaisant mais les différentes difficultés rencontrées au cours de son développement seront assurément enrichissantes.

Le temps requis pour mettre en place cet outil sera probablement conséquent (plusieurs dizaines d’heures cumulées sur plusieurs membres) néanmoins ce dernier dépendra des fonctionnalités internes de l'outil telles que les différents formats sous lesquels seront restituées les résultats ou encore la diversité d’interactions offertes avec notre base de données.

### 4.11 Répartition des tâches

**Répartition des rôles et priorités**

#### Romain – *Responsable des Données*:

Romain est chargé de l'implémentation du moteur d'échecs et de l'amélioration de l'algorithme MiniMax.  
Cependant, nous aurons besoin de son expertise pour réfléchir à la meilleure façon de programmer le livre d'ouvertures, en envisageant l'utilisation d'une base de données dédiée. Nous travaillerons donc en étroite collaboration pour atteindre cet objectif.

Temps estimé : 100 heures

**Priorité actuelle : Refonte du moteur d'échecs et de Minimax**

#### Jimmy – *Responsable Qualité*:

Jimmy est chargé de la ré-implémentation de la partie réseau avec des sockets, ainsi que des routes associées, cette fois pour notre propre moteur. Il avait déjà réalisé cette tâche au premier semestre avec le moteur d’échecs basé sur la bibliothèque *chess.js*.  
Il prendra également en charge le portage de l’application en format mobile et la restructuration des pages.

Temps estimé : 100 heures

**Priorité actuelle : Mise en place du réseau pour notre nouveau moteur**

#### Maxence & Pierre-Emmanuel – *Responsable du Projet*

Pierre-Emmanuel et Maxence sont responsables de l’implémentation de notre propre système d’authentification, ainsi que de la restructuration des pages, des routes et de la base de données, ainsi que la gestion des rôles/permissions et des tokens.

Temps estimé : 100 heures

**Priorité actuelle : Création du système d’authentification**

#### Alexandre:

Alexandre s’occupera de la restructuration de la visualisation des statistiques sous forme de graphiques et de la mise en place de nouvelles routes associées.

Temps estimé : 100 heures

**Priorité actuelle : Définition des statistiques à afficher (quelles données, quel format, pertinence)**

#### Émile – *DevOps*:

Émile sera chargé des interactions entre utilisateurs, notamment la gestion de la liste de contacts et des amis. Il intégrera également un système de chat entre amis ainsi qu’un outil de visualisation de données.

Temps estimé : 100 heures

**Priorité actuelle : Mise en place du chat de discussion**

Théo – *Responsable Développeurs* :

Théo va mettre en place le système de détection de triche basé sur l’algorithme de jeu et les coups des joueurs. Il aidera aussi sur la visualisation des données globales de l’application.

Temps estimé : 100 heures

**Priorité actuelle : Implémentation du système de détection de tricheurs**

**À noter** : Tous les membres de l’équipe ont le rôle de développeur. Les tâches attribuées ne sont pas figées ou fixes à chacun et peuvent évoluer en fonction des besoins du projet. De même, les estimations de temps sont indicatives et sujettes à ajustement.

## **5 - Conclusion**

Ce projet marque une avancée significative dans la conception et le développement d'une application d'échecs robuste et innovante, enrichie par des fonctionnalités techniques avancées et des améliorations continues. L'expérience acquise et les choix effectués, qu'ils soient technologiques ou organisationnels, témoignent de l'engagement d’une certaine collaboration au sein de l'équipe.

Parmi les fonctionnalités majeures, la refonte du moteur d'échecs avec les Bitboards, des Magic Bitboards et d'autres optimisations techniques permettront d’avoir une base solide pour améliorer les performances et la profondeur de recherche de l'algorithme Minimax. Ces avancées garantiront une expérience utilisateur plus fluide et une analyse des parties plus précise.

L'intégration d'une authentification sécurisée avec Spring Security et JWT, couplée à une gestion des utilisateurs améliorée grâce à des schémas de base de données bien structurés, renforcera la sécurité et l'évolutivité de l'application. Par ailleurs, le portage vers une application mobile via React Native illustre une volonté d'étendre l'accessibilité et d'offrir une expérience multiplateforme cohérente.

En parallèle, l'accent sera mis sur l'expérience utilisateur qui se traduira par des fonctionnalités telles que la visualisation des données statistiques, l'intégration d'interactions sociales et une refonte graphique du front-end. Ces éléments visent à rendre l'application plus engageante et intuitive.

D'un point de vue global sur l'avancée du projet, notre groupe fonctionne bien. Chaque partie de ce projet, qui se déroule en parallèle, avance à son rythme et communique de façon efficace son avancement pour garder une bonne coordination et un bon suivi. Cette nouvelle façon de procéder permet de pallier les problèmes que nous rencontrions au *premier semestre* de façon efficace en prenant du recule sur le projet et en voyant l’avancement des tâches sur chaque partie grâce à *OpenProject*.

En conclusion, ce projet va permettre de développer une application d'échecs ambitieuse, tout en renforçant les compétences techniques et organisationnelles des membres de l'équipe. Les bases solides posées par ce travail ouvrent la voie à de futures évolutions, avec pour objectif de proposer une solution toujours plus performante et adaptée aux besoins des utilisateurs.