IDMC
2024-2025

The Real Deal

PARIS SPORTIFS

ARCHITECTURE ORIENTÉE SERVICES

JULIE BARTHET

MAXIME BRASLEY

Sommaire

Analyse du Domaine

Identification des services

Le projet est découpé en plusieurs services, chacun a des responsabilités et des fonctionnalités précises. Voici les principaux services que l'on a identifiés :

1. **service-user** :
    a. Rôle principal : Gestion des utilisateurs
    b. Fonctionnalité :
        i. Authentification
        ii. Création de comptes
        iii. Récupération des informations de compte

2. **service-match** :
    a. Rôle principal : Gestion des matchs
    b. Fonctionnalités :
        i. Faire le lien avec une API externe "football-data.org"
        ii. Récupérer tous les matchs d'une compétition
        iii. Récupérer les informations d'un match donné

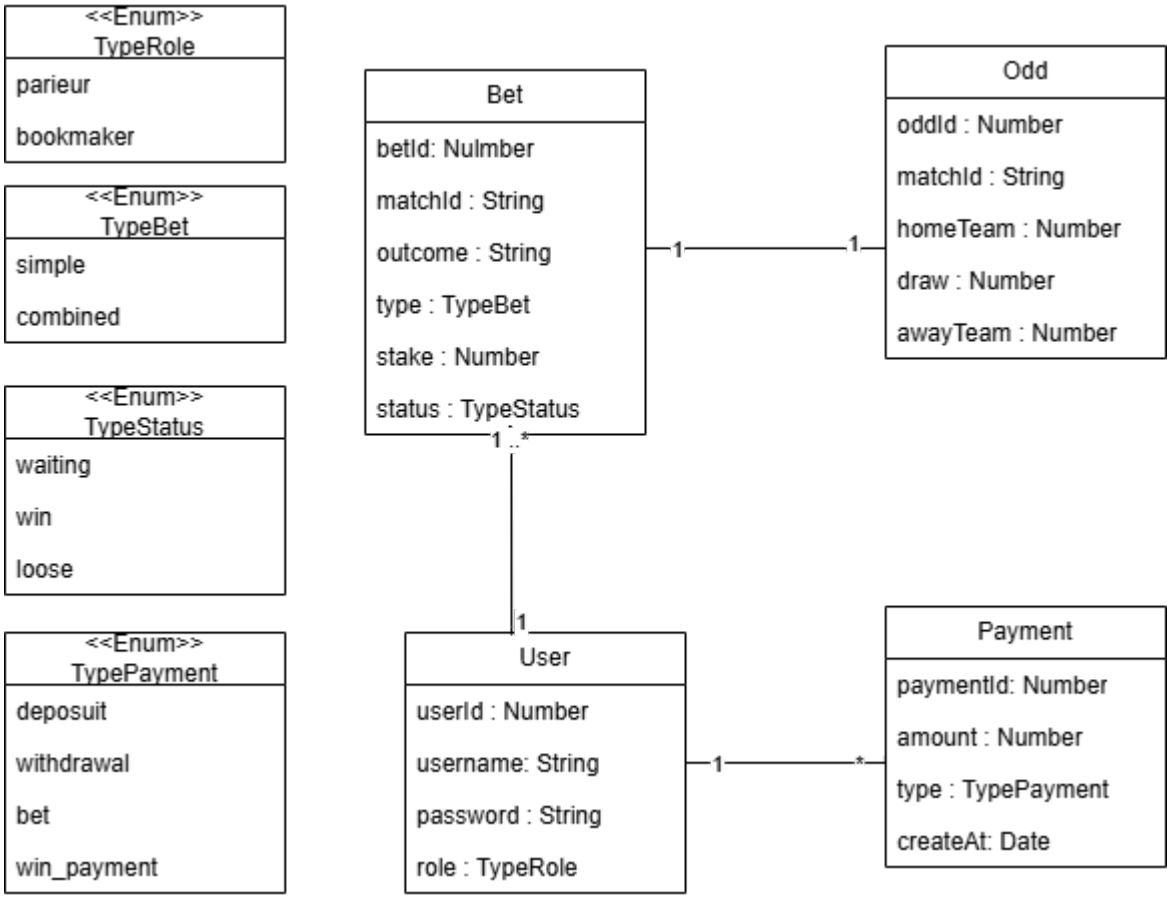
3. **service-bet** :
    a. Rôle principal : Gestion des paris
    b. Fonctionnalités :
        i. Enregistrer les paris
        ii. Vérifier le résultat des paris
        iii. Afficher les paris réalisés

Ce service est interconnecté au service odd, payment et match. Pour placer le pari, on interroge le service odd afin d'obtenir la côte associée, le service payment pour récupérer le montant du portefeuille afin de savoir si l'utilisateur a les fonds nécessaires pour placer son pari ou non, mais aussi pour gérer le débit lorsqu'il place son pari ou le paiement du gain s'il gagne.

4. **service-odd** :
    a. Rôle principal : Gestion des cotes associées aux paris.
    b. Fonctionnalités :
        i. Créer ou modifier une cote en fonction d'un match, le bookmaker n'est pas obligé de saisir les 3 cotes (domicile, extérieur et égalité). Pour une cote mise, on enregistre l'id du match et la cote ; par exemple, si la cote domicile est saisie, on l'enregistre avec le matchId et on met 0 sur les autres (car lors d'une création ou modification, on enregistre systématiquement les 3 cotes). Si le bookmaker ajoute d'autre côté sur ce même match, on met à jour dans la base en modifiant le 0 par la valeur saisie.
        ii. Renvoyer les cotes pour un match

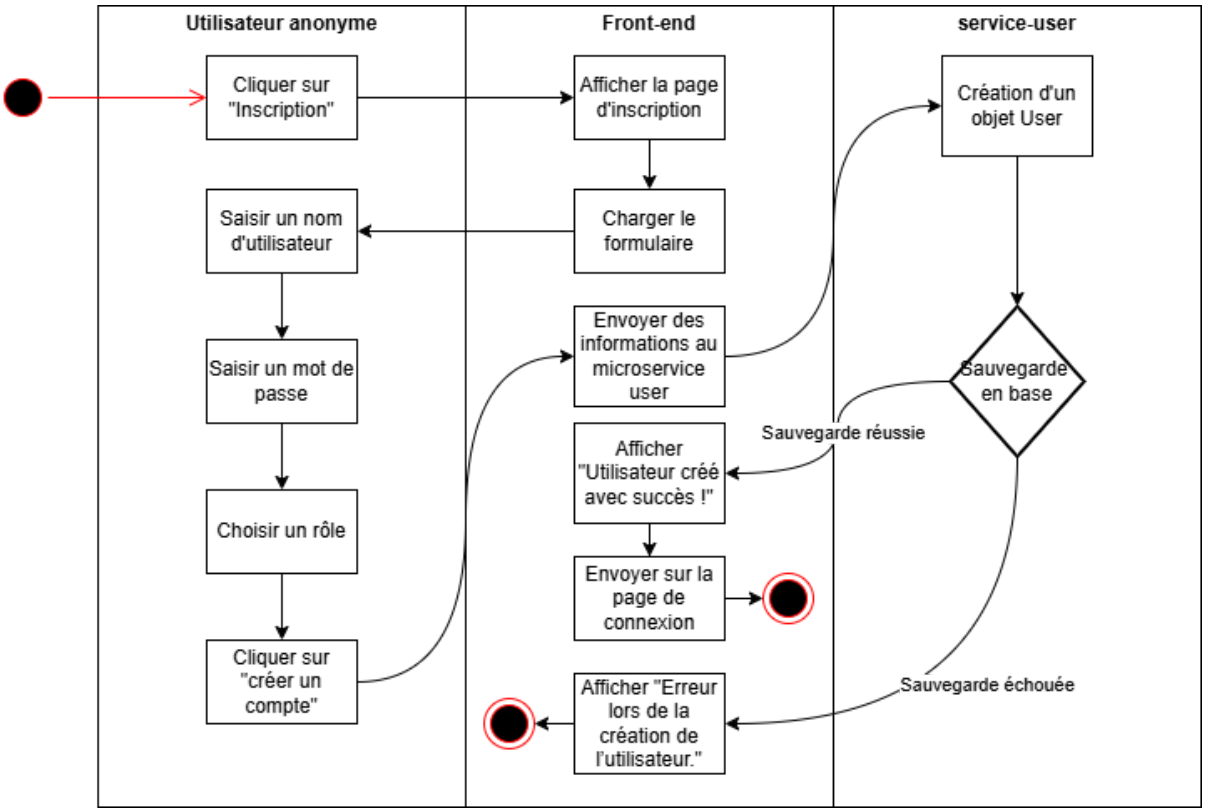
5. **service-payment** :
    a. Rôle principal : Gestion des paiements et transactions financières.
    b. Fonctionnalités :
        i. Stocker un historique de transaction, ce qui permet de retourner le solde du compte.
        ii. Déposer de l'argent
        iii. Retirer de l'argent

Schéma de la base de donnée :



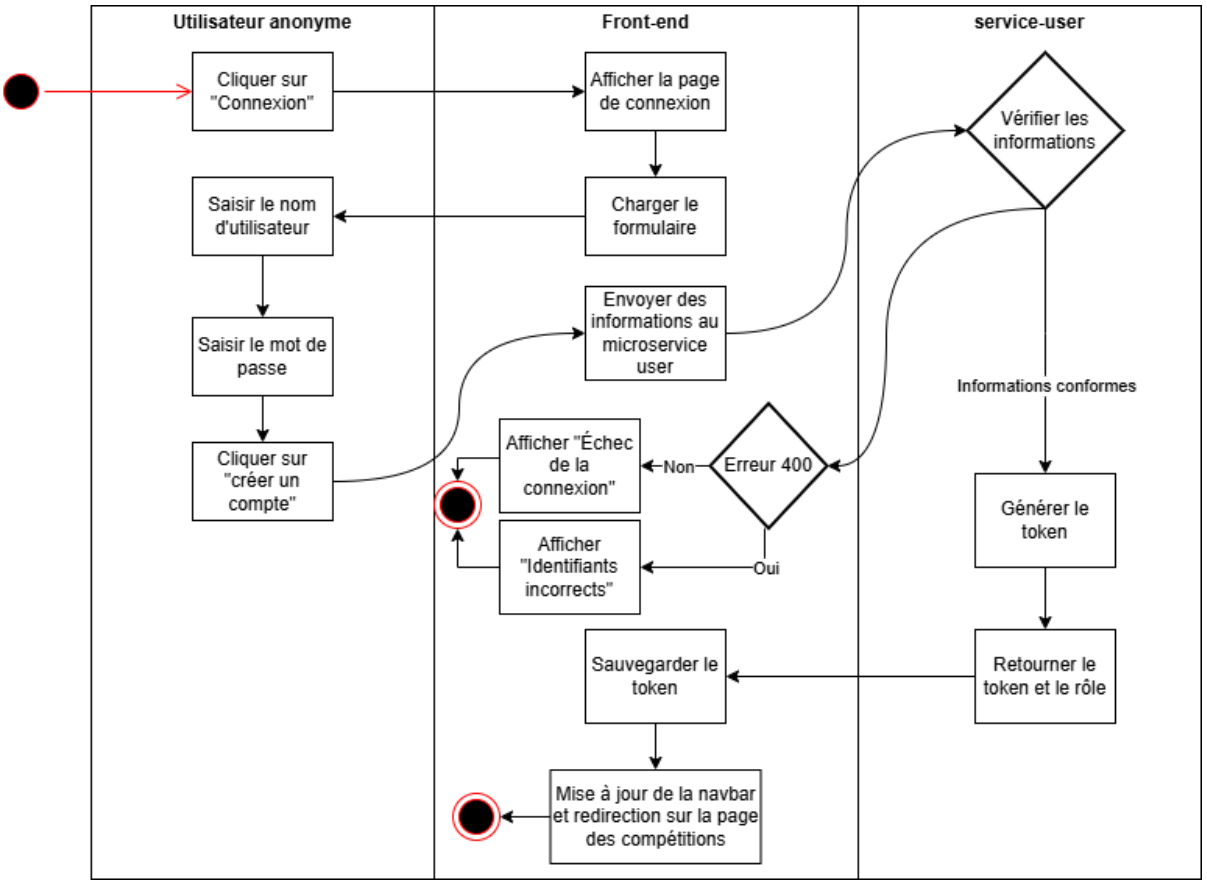
Processus Métier

- **Création de compte** :



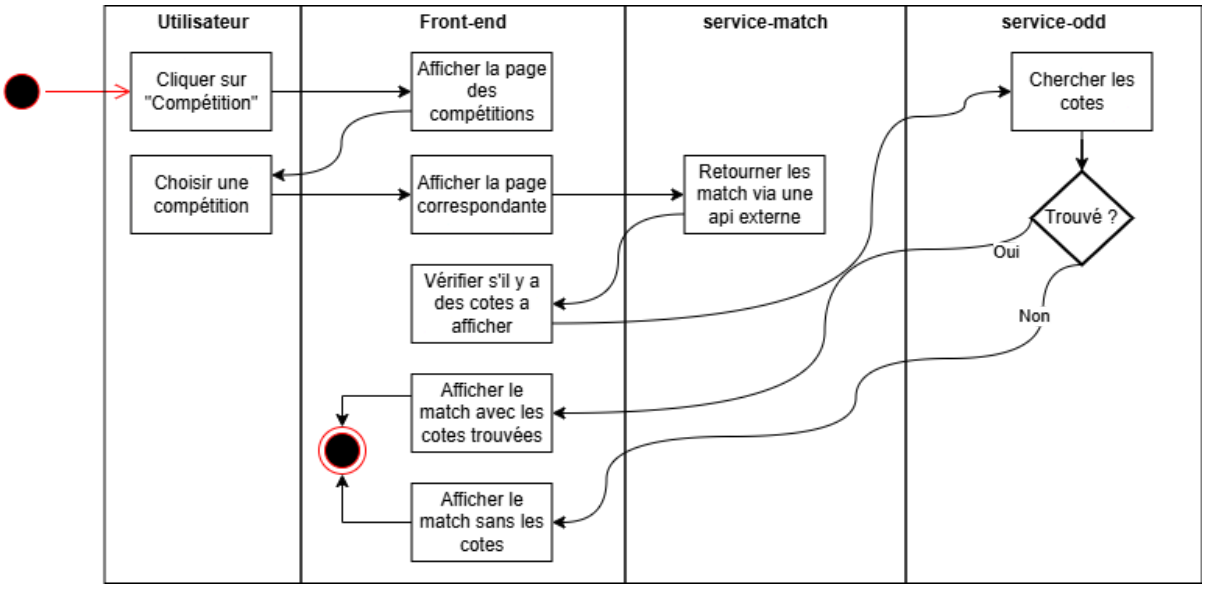
Ici on affiche une liste des rôles et on laisse le choix à l'utilisateur. Dans un cas réel, ce n'est pas la meilleure solution mais pour simplifier le front-end nous avons opté pour ce choix.

- **Connexion :**



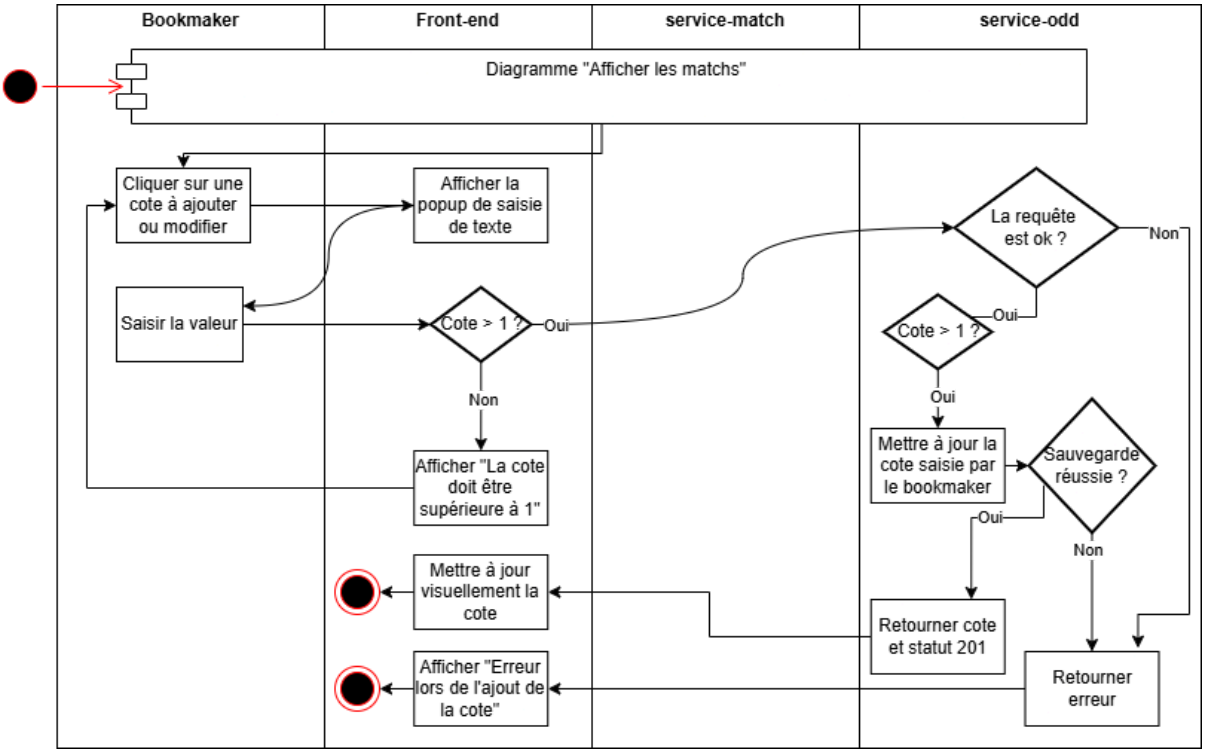
Le token est signé par JWT et il comprend l'identifiant de l'utilisateur ainsi que son rôle et se signe avec la clé secrète de l'application.

- **Afficher des matchs :**



Ce diagramme va être utilisé dans d'autres diagrammes, cela permettrait d'éviter la répétition. La sortie sera l'action des diagrammes qui implémenter "afficher les matchs"

- **Définir des cotes :**

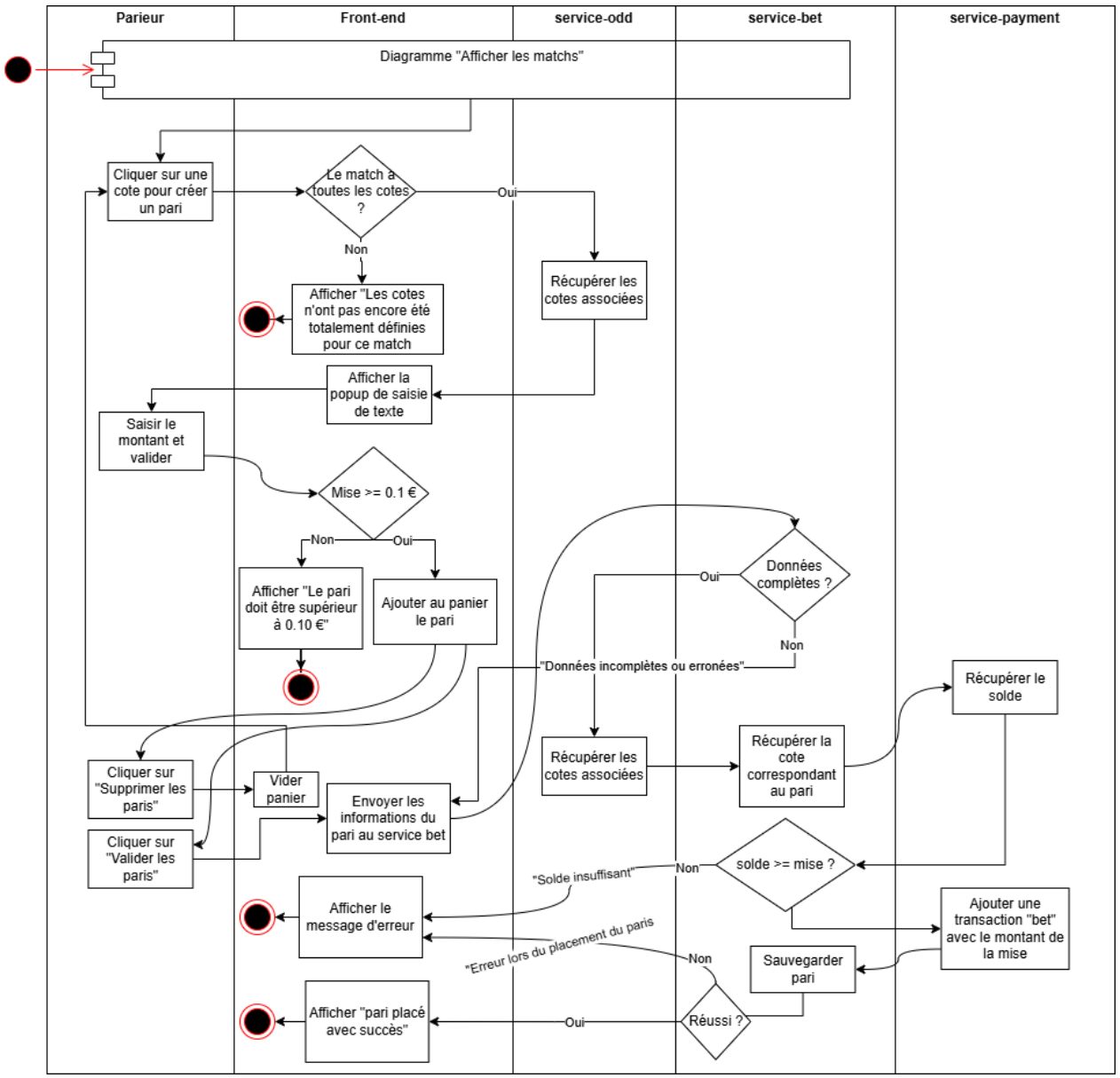


On vérifie dans le front-end que la cote saisie est supérieure à 1 pour éviter d'envoyer une erreur au service odd. Cependant, nous suivons le principe de ne jamais faire confiance à l'utilisateur, de ce fait nous refaisons la vérification dans le service odd.

Une cote ne peut pas être inférieure à 1 car ce serait illogique de faire un pari qui fait perdre de l'argent si l'on gagne.

Lorsque l'on stocke une cote (création ou modification par un bookmaker), on stocke les 3 cotes pour le même id match. Etant donné qu'il n'est possible que de mettre les cotes qu'une par une, on ajoute la cote correspondante (extérieur, domicile, égalité) et on met 0 aux autres si aucune valeur n'existait auparavant (sinon on conserve les valeurs stockées). Pour saisir une cote, le système vérifie si l'utilisateur est bien bookmaker, s'il est parieur ce même bouton aura une autre fonction.

- **Placement d'un pari** :



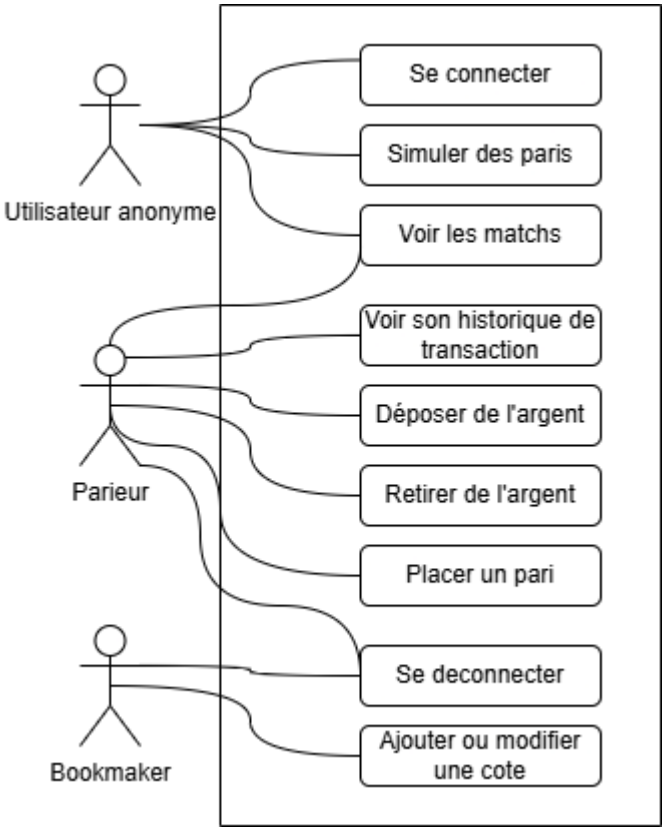
Si l'utilisateur est connecté alors le bouton "Valider les paris" devient cliquable. Si il est bookmaker, le clic sur la cote ne met pas de pari dans le panier mais propose de changer la cote sélectionnée. Et si l'utilisateur est parieur, le clic ajoute le pari dans le panier.

Pour le retrait de la mise, on ne retire pas directement un montant car il n'y a pas de solde stocké, mais il existe une liste de transaction avec un type correspondant. Le solde est récupéré via une fonction qui regarde le type de transaction, ajoute ou soustrait à la somme la valeur de la transaction, ce qui donne le solde.

Ici nous n'avons pas détaillé toutes les fonctionnalités de l'application cependant nous allons en voir certaines.

En tant qu'utilisateur anonyme (non connecté), nous avons la possibilité de consulter les matchs et de simuler des paris. La simulation de paris correspond à la mise dans le panier d'un pari, le calcul de gain y est affiché. Il n'est juste pas possible de cliquer sur le bouton de soumission de paris.
Pour un utilisateur connecté de type "parieur", s'il va dans l'onglet "Mon compte" il pourra retrouver son solde, son historique de transactions mais il pourra aussi déposer ou retirer de l'argent (dans notre cas, on simule le dépôt et le retrait en ajoutant ou soustrayant le montant).

Voici un schéma qui montre les différentes fonctionnalités accessibles en fonction du rôle de l'utilisateur.



Conception de l'Architecture

Vue d'ensemble

L'architecture repose sur une structure en **microservices** orchestrée par une gateway et conteneurisée avec Docker. Chaque service est autonome et focalisé sur une responsabilité spécifique.

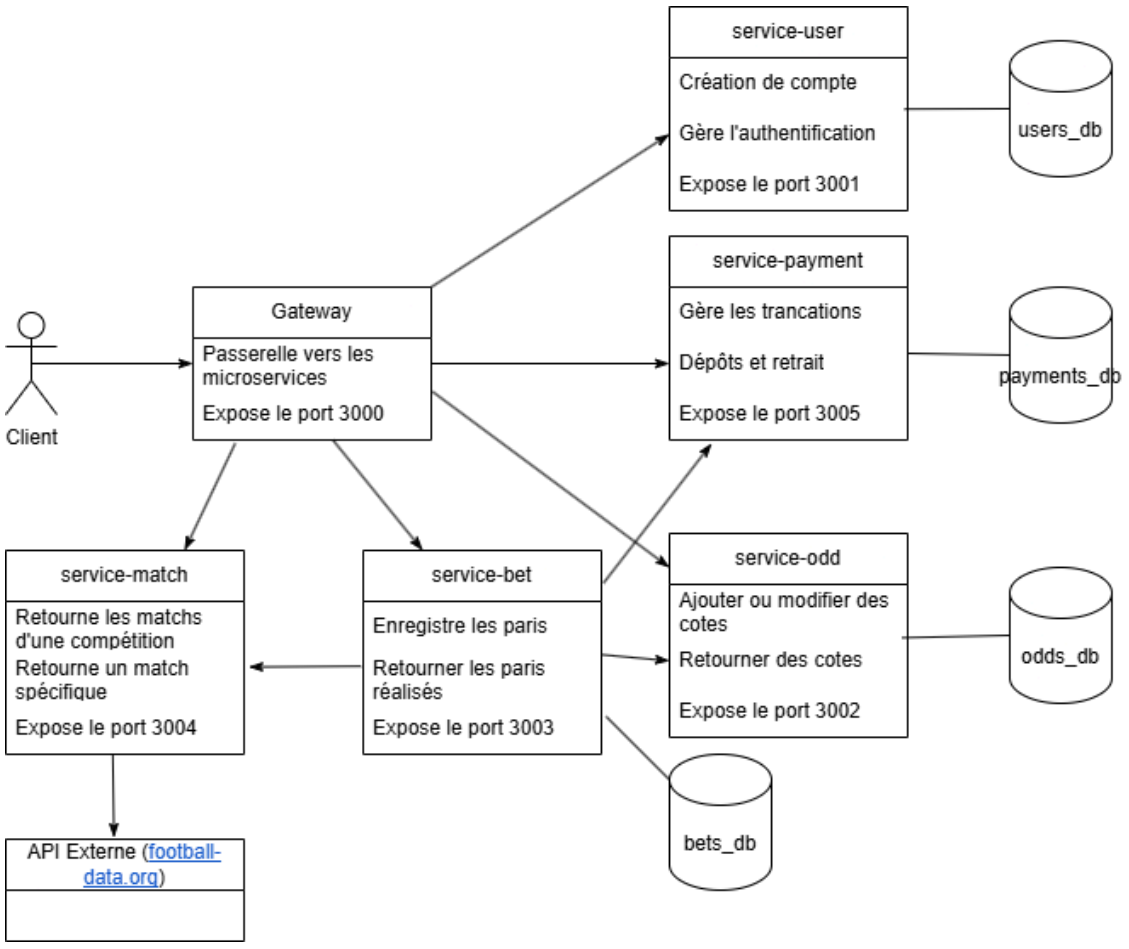
Points clés de l'architecture

1. **Communication entre les services** :

La gateway agit comme un intermédiaire, le client va interroger la gateway qui va rediriger les requêtes faites vers les services backend correspondants. La gateway connaît tous les endpoints et les sécurise en fonction du rôle de l'utilisateur (certains endpoints ne doivent pas être accessibles pour un bookmaker par exemple). Elle constitue l'unique point d'entrée aux services backend ce qui augmente la sécurité de l'application.

2. **Conteneurisation** :
    - Utilisation de Docker pour empaqueter et déployer les services, chaque service a un fichier dockerFile qui permet à docker de suivre certaines instructions pour notamment installer les dépendances nécessaires au fonctionnement du service.
    - A la racine du projet il y a un fichier docker-compose.yml qui permet de décrire les différents services a dockerisé.
3. **Technologies** :
    - Backend : Nous avons choisi d'utiliser Node.js avec Express car c'est assez simple à utiliser, permet de travailler dans un écosystème basé sur du javascript et nous pouvons faire plusieurs requêtes en même temps sans attendre la réponse de la requête précédentes.
    - Base de données : MongoDB car avec son modèle de documents json, il permet de stocker des données de manière plus flexible et plus facile à gérer.
    - Frontend : Angular car il s'agit d'un framework qui est structuré, qui permet d'utiliser des composants, ce qui le rend modulable. Il impose d'avoir une certaine rigueur afin de ne pas faire n'importe quoi mais lorsqu'on maîtrise le framework c'est très puissant.

Schéma de l'architecture :



Ici le client correspond au front-end c'est lui qui envoie les requête à la gateway. Le sens des flèches correspond à qui interroge quel service.

Sécurisation de l'application

La sécurisation d'une application web est très importante. Nous avons mis en place différents procédés afin de sécuriser le mieux possible notre application.

Dans un premier temps nous avons créé dans chaque microservice un fichier .env qui isole les informations sensibles, notre clé secrète, le port du service et l'uri de connexion à mongo. La gateway dispose de toutes les url des services. En théorie, ces fichiers .env évitent d'exposer ses informations dans notre git. Dans notre cas nous ne l'avons pas exclu pour des raisons de facilité dans le développement, cela perd de l'intérêt mais nous a permis de découvrir ce procédé. De plus, malgré l'utilisation de dotenv nous n'arrivions pas à charger la clé, nous l'avons donc malheureusement écrit en dur dans le code, cependant nous avons tout de même découvert ce concept de sécurité.

Nous avons aussi sécurisé la connexion via un token généré par JWT, ce qui permet de créer une vraie connexion et limité l'accès à certaines pages en fonction de nos droits. En

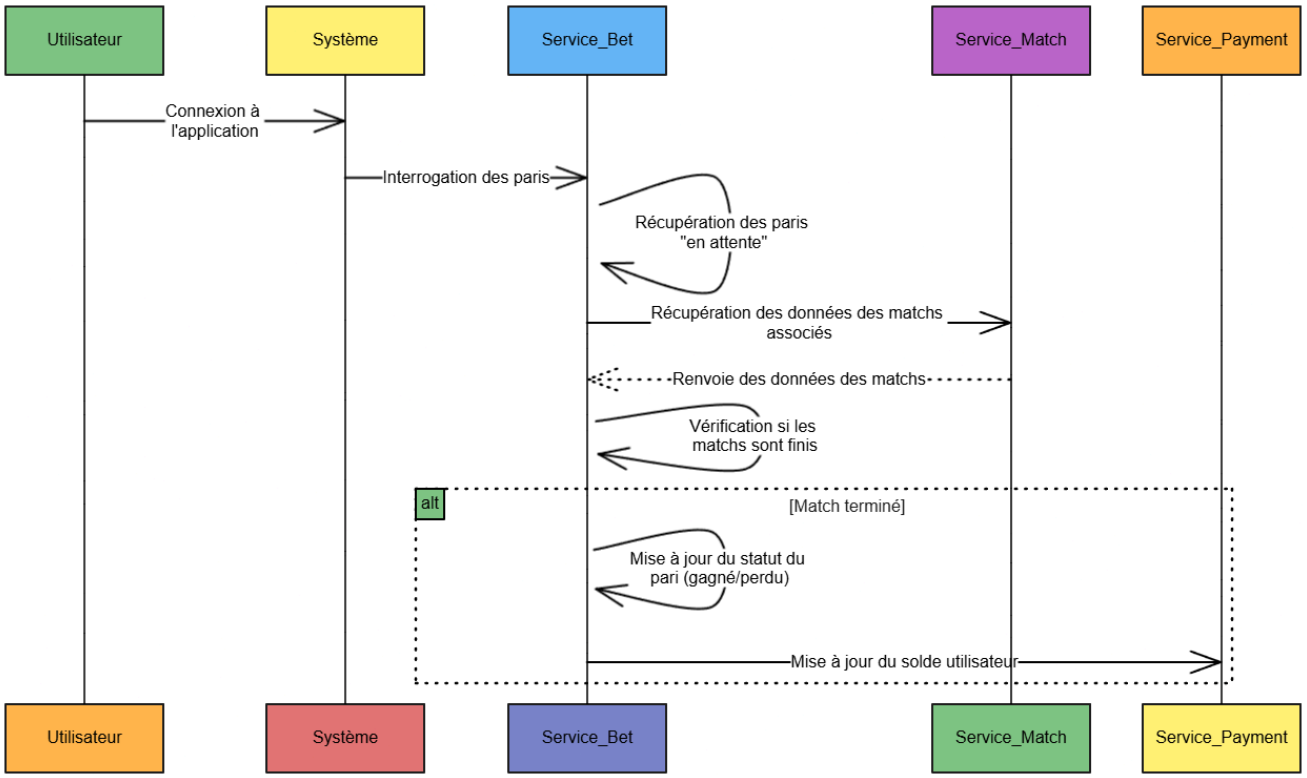
effet, pour les routes nécessitant une connexion la gateway vérifie la présence du token, et pour les routes nécessitant le rôle bookmaker ou parieur la gateway vérifie que l'utilisateur dispose du bon rôle avant d'accéder au service.

Lors de la création d'un utilisateur nous cryptons le mot de passe via bcrypt ce qui permet en cas d'attaque de ne jamais avoir le mot de passe en clair dans notre application.

Fonctionnalités développées et manquantes

Dans le cadre du développement de notre projet, nous avons mis en place plusieurs fonctionnalités essentielles. Les utilisateurs peuvent déposer et retirer de l'argent, avec un minimum fixé à **10€**. Ils ont également accès à l'historique de leurs transactions, ce qui leur permet de suivre leurs dépôts, retraits, paris placés et gains obtenus. Cela leur offre une visibilité sur leur solde disponible et leur activité de jeu.

Pour placer un pari, l'utilisateur doit simplement cliquer sur la cote d'un match. Le pari est alors ajouté au panier avec la cote correspondante et le gain potentiel calculé. La mise minimale a été fixée à **0,10€**, mais nous n'avons pas défini de mise maximale. Une fois le pari validé, il est possible de consulter ses paris en cours, avec toutes les informations nécessaires : le match concerné, la mise, la cote, le gain potentiel et le statut du pari (**gagné, perdu ou en attente**). Pour garantir une mise à jour automatique des matchs, nous avons intégré l'API football-data.org, qui nous permet de récupérer des informations sur les compétitions et les matchs. Pour ne pas alourdir le développement, nous avons choisi de nous limiter à la **Champions League** et à la **Premier League**.



Nous avons également mis en place un système pour gérer les matchs en temps réel. Lorsqu'un match commence, son statut passe à **"en cours"**, ce qui empêche les utilisateurs de parier dessus. A la connexion à l'application, le système interroge le service bet, qui récupère les paris, regarde ceux en statut "en attente" puis récupère les données des matchs associés et si les matchs sont finis alors on change le statut du paris en fonction de si il est gagné ou perdu puis on met à jour le solde dans le service bet (cf. schéma ci-dessus).

Cependant, certaines fonctionnalités n'ont pas pu être développées, principalement par manque de temps. Nous avions envisagé d'ajouter les **paris combinés**, qui permettent de parier sur plusieurs matchs en même temps pour augmenter les gains. L'idée était d'additionner les cotes des matchs sélectionnés et d'appliquer un coefficient **1,1** pour rendre ce type de pari plus intéressant. Le traitement aurait été similaire à celui des paris simples, mais le pari combiné aurait été perdu dès qu'un des matchs sélectionnés n'avait pas le bon résultat.

Un autre ajout possible aurait été la gestion des **freebets**, c'est-à-dire des paris gratuits. Nous aurions pu créer un solde spécifique pour ces paris et permettre aux utilisateurs de choisir entre utiliser leurs freebets ou leur argent réel. À l'inscription, **10 freebets** auraient pu être offerts, et en cas de gain, l'argent obtenu aurait été converti en euros avec un coefficient de **0,9** (exemple : un pari de **10 freebets** sur une cote à **2** aurait rapporté **18€**).

Nous n'avons pas non plus mis en place de solution pour gérer les **matchs annulés ou reportés**. Actuellement, aucune action spécifique n'est prévue dans ces cas-là, mais nous aurions pu adapter notre système de mise à jour des résultats pour détecter ces situations et ajuster les paris en conséquence.

Enfin, nous avions envisagé d'intégrer un **système de notifications** avec un Message Broker comme RabbitMQ. Ce système aurait permis d'envoyer des notifications aux utilisateurs, par exemple pour les informer qu'un pari a été gagné. Pour cela, nous aurions dû mettre en place un service dédié capable de traiter les messages envoyés par les autres services et de générer des notifications automatiques, notamment par e-mail.

Voici comment l'architecture aurait été modifiée avec cet ajout.