

# Informaticien/-ne CFC

Travail pratique individuel 2023 (TPI)



**ICT Berufsbildung**  
**Formation professionnelle**  
**Freiburg-Fribourg**

## Modèle de rapport v1.4

Nom du candidat : Killian Pasche

**Candidat N°**

**145341**

## Futnet Single Master

## Sommaire

<b>Résumé du rapport du TPI .....</b>	<b>3</b>
<b>1 Les grandes lignes du projet .....</b>	<b>4</b>
1.1 Analyse de la situation initiale .....	4
1.2 Analyse de l'état désiré .....	4
1.3 Cahier des charges / exigences du système.....	4
1.4 Organisation du projet.....	5
<b>2 Analyse préliminaire .....</b>	<b>6</b>
2.1 Objectifs du système.....	6
2.2 Processus d'entreprises concernées .....	6
2.3 Objectifs .....	6
2.4 Technologies.....	7
2.5 Analyse de risque.....	8
2.6 Sécurité de l'information et protection des données.....	8
2.7 Test technologique.....	8
<b>3 Analyse .....</b>	<b>15</b>
3.1 Use Case .....	15
3.2 Maquettes .....	16
3.3 Diagramme d'activité / séquence .....	19
<b>4 Concept.....</b>	<b>28</b>
4.1 Diagramme de classe .....	28
4.2 Diagramme de séquence interactions.....	30
4.3 Diagramme d'entité-relation .....	34
4.4 Schéma relationnel de la base de données .....	35
4.1 Concept de tests .....	37
<b>5 Réalisation .....</b>	<b>38</b>
5.1 Tests fonctionnels .....	38
5.2 Implémentation de la base de données .....	38
5.3 Réalisation du Serveur.....	40
5.4 Réalisation du client.....	51
5.5 Configuration client .....	56
<b>6 Test.....</b>	<b>59</b>
6.1 Procédure de test.....	59
6.2 Protocol de test .....	59
6.3 Signature du protocole de test .....	60

<b>7</b>	<b>Conclusion.....</b>	<b>61</b>
7.1	Améliorations possibles .....	61
7.2	Auto-évaluation .....	61
<b>8</b>	<b>Bibliographie: liste des sources et références .....</b>	<b>63</b>
<b>9</b>	<b>Glossaire.....</b>	<b>64</b>
<b>10</b>	<b>Signatures.....</b>	<b>67</b>
<b>11</b>	<b>Annexes .....</b>	<b>68</b>

# Résumé du rapport du TPI

## Situation de départ

Le projet ne commence de rien. L'idée est de créer une application web permettant aux pratiquants du futsal (un sport) d'entrer leurs scores de leurs rencontres et de visualiser le classement général du tournoi. L'application dispose également d'un accès administrateur qui permet aux administrateurs d'ajouter des utilisateurs, en modifier et en supprimer.

## Mise en Œuvre

La réalisation du projet implique une analyse poussée sur ce qu'il faudra concevoir et réaliser. Des schémas ainsi que des maquettes devront être réalisés pour pouvoir visualiser le travail à réaliser. L'application web devra remplir les différents critères définis au préalable.

Un diagramme de use case, séquences, activités, classes et interactions devront être réalisés et documentés avant le début de la réalisation. Un modèle relationnel de la BD devra également être réalisé avant de commencer à implémenter la base de données.

## Résultats

L'application web demandée a été réalisée et tous les critères sont remplis. L'application permet donc à des utilisateurs une connexion. Une fois connectés, les joueurs peuvent consulter leurs rencontres, en ajouter, en modifier et en valider. Les joueurs ayant le droit administrateur peuvent ajouter, modifier et supprimer des joueurs.

L'application n'est pas flexible à 100%, il y a toujours un bug sur l'interface téléphone qui fait que les tableaux dépassent de la page.

# 1 Les grandes lignes du projet

## 1.1 Analyse de la situation initiale

La discipline sportive du Futnet a besoin d'une application web pour la gestion de tournoi. L'idée est de créer une application web pour la gestion du tournoi « Single Master » annuel de futnet.

## 1.2 Analyse de l'état désiré

A la fin de mon TPI le projet devra correspondre au cahier des charges, c'est-à-dire qu'un utilisateur pourra faire ces actions :

Si l'utilisateur est un joueur :

- Visualiser le classement et les matchs qui ont eu lieux.
- Ajouter un match contre un adversaire et y inscrire le score.
- Valider les matchs inscrits par les adversaires.

Si l'utilisateur est un admin :

- Inscrire des joueurs dans le tournoi.

L'application aura donc un login qui permettra aux joueurs / admins de se connecter.

## 1.3 Cahier des charges / exigences du système

### 1.3.1 Analyse

- Analyser les tâches complètes du projet et les maquettes préparées auparavant.
- Créer le diagramme des cas d'utilisation global.
- Créer les différents diagrammes de séquences des systèmes pour les applications.
- Créer les différents diagrammes d'activités principaux pour les applications.

### 1.3.2 Conception

- Créer le modèle relationnel de la base de données utile pour l'application Backend.
- Créer le diagramme des classes des applications.
- Créer les différents diagrammes d'interactions des méthodes principales des applications.

### 1.3.3 Réalisation

- Réaliser l'implémentation de la base de données.
- Réaliser les différentes requêtes SQL utiles pour la base de données.

- Réaliser l'implémentation des services REST (application BackEnd).
- Développer l'interface homme-machine (application FrontEnd).
- Réaliser l'implémentation des applications et leur communication.
- Tester le fonctionnement complet des applications et leurs relations.

## 1.4 Organisation du projet

### 1.4.1 Méthode de gestion

La méthode de gestion du projet est la méthode en phase. Elle consiste à réaliser un planning afin d'analyser tout ce que je vais devoir faire sur mon projet. Le planning décrit en bref les différentes étapes du projet. (Analyse, Conception, Réalisation et Test)

### 1.4.2 Contact du projet

Voici les contacts des personnes concernées par le projet.

Candidat : Killian Pasche, [killian.pasche7@gmail.com](mailto:killian.pasche7@gmail.com), +41763103560.

Supérieur professionnel : Silvin Meylan, [silvin.meylan@edufr.ch](mailto:silvin.meylan@edufr.ch), +41797903591.

Expert principale : Didier Perroud, [didier.perroud@gmail.com](mailto:didier.perroud@gmail.com), +41798003353.

Expert secondaire : Samuel Thomet, [samuel.thomet@mobi.ch](mailto:samuel.thomet@mobi.ch), +41793065707.

### 1.4.3 Sauvegarde du projet

Mon TPI sera stocké sur un git ce qui permet d'accéder au projet entier à n'importe quel moment et n'importe où, cela me permet également de garder une trace des diverses versions de mon projet ce qui me permet de revenir en arrière à n'importe quel moment.

## 2 Analyse préliminaire

### 2.1 Objectifs du système

#### 2.1.1 Analyse de l'état actuel

Actuellement l'application est seulement au stade d'idée, aucun prototype, schéma ou analyse n'a été réalisé. Toutes les informations existantes se trouvent dans le cahier de charge.

#### 2.1.2 Analyse de l'état désiré

Lors de la remise du TPI, l'application devra être fonctionnelle et remplir les critères d'évaluation. Le projet sera également documenté, avec une documentation utilisateur rédigée. L'application devra également atteindre les objectifs fixés.

### 2.2 Processus d'entreprises concernées

Les principaux acteurs concernés sont les joueurs de futnet qui auront accès à l'application pour consulter leurs classements.

### 2.3 Objectifs

Pour l'application Backend :

- - Permettre de voir la liste des joueurs dans le tournoi (GET)
- - Permettre d'ajouter (POST) des joueurs dans le tournoi
- - Permettre de modifier (PUT) des joueurs du tournoi
- - Permettre de supprimer (DELETE) des joueurs du tournoi
- - Permettre de voir la liste des matchs (GET) avec leur score
- - Permettre d'ajouter un match avec son score contre un adversaire (POST)
- - Permettre de modifier un match avec son score contre un adversaire (PUT)
- - Permettre de supprimer un match avec son score contre un adversaire (DELETE)
- - Permettre de valider un match et son score (PUT)
- - Permettre de ressortir le classement des joueurs selon le nombre de points (GET)

Pour l'application Frontend : Droit administrateur

- Permettre d'inscrire (de créer) des joueurs dans le tournoi Droit joueur

- - Permettre de visualiser le classement et les matchs qui ont déjà eu lieu
- - Permettre d'ajouter un match contre un adversaire et d'inscrire le score du match
- - Permettre de valider les matchs inscrits par des adversaires

## 2.4 Technologies

Pour réaliser mon TPI, j'ai le choix entre deux technologies pour le client : Angular ou VueJS. Pour le serveur, j'ai le choix de savoir si j'utilise JPA.

### 2.4.1 Client

#### 2.4.1.1 Angular

Angular est un framework robuste et puissant pour le développement d'applications web modernes. Il offre une architecture basée sur les composants, une liaison de données bidirectionnelle, une gestion avancée des routes, la gestion des dépendances, des outils de test et un écosystème solide.

#### 2.4.1.2 VueJS

VueJS est un framework JavaScript progressif, réactif et basé sur les composants. Il offre une approche flexible pour la construction d'interfaces utilisateur interactives, tout en étant léger, performant et bénéficiant d'un écosystème dynamique. C'est un choix populaire pour les développeurs qui cherchent une solution moderne et conviviale pour leurs projets web.

#### 2.4.1.3 Choix de technologie

Pour mon TPI j'ai choisi le framework VueJS, voici les raisons pour lesquelles j'ai choisi VueJS plutôt qu'Angular :

1. VueJS est plus facile à apprendre que Angular, n'ayant aucune connaissance de base sur aucun des deux framework, j'ai préféré choisir le framework sur lequel j'aurais le plus de facilité à apprendre.
2. VueJS est extrêmement flexible ce qui permet de choisir les fonctionnalités spécifiques en fonction du projet. Le fait qu'il adopte une approche progressive est un plus car on peut adapter à n'importe quel moment en fonction des besoins.
3. VueJS est également beaucoup plus léger que Angular, ce qui permet à l'utilisateur une expérience plus rapide et où il y a le moins de chargement sur ordinateur comme sur téléphone.
4. Bien que la communauté d'Angular soit plus grande, VueJS a également une communauté très active et évolutive avec de nombreux supports et de ressources.

### 2.4.2 Serveur

Pour le serveur, je devais choisir si je voulais utiliser JPA pour communiquer avec ma base de données.



### 2.4.2.1 JPA

JPA, ou Java Persistence API, est une spécification Java qui simplifie la persistance des données dans les applications Java en fournissant une abstraction de la couche de persistance, un mapping objet-relationnel, une gestion des transactions, une portabilité et des fonctionnalités avancées.

### 2.4.2.2 Points forts de JPA

Je compte utiliser JPA pour mon TPI car il a des fonctionnalités assez intéressantes qui me permettent de gagner du temps.

1. Fonctionnalités avancées : JPA offre également des fonctionnalités avancées telles que la gestion des relations entre les entités (relations One-to-One, One-to-Many, Many-to-One, Many-to-Many), les requêtes dynamiques, les mécanismes de mise en cache et l'héritage des entités.
2. Abstraction de la couche de persistance : JPA offre une abstraction de la couche de persistance, ce qui signifie que les développeurs n'ont pas à se soucier des détails spécifiques de la base de données sous-jacentes. Ils peuvent travailler avec des objets Java familiers plutôt qu'avec des requêtes SQL complexes.
3. Gestion des transactions : JPA prend en charge la gestion des transactions, ce qui signifie qu'elle permet d'effectuer des opérations sur les objets persistants de manière cohérente et sécurisée. Elle offre des fonctionnalités telles que le contrôle des transactions, la gestion du commit et du rollback, assurant ainsi l'intégrité des données.

## 2.5 Analyse de risque

Le projet ne sera pas hébergé et il ne sera pas forcément utilisé tout de suite. Dès lors, il n'y a aucun vrai problème si l'application n'est pas finie dans les délais.

## 2.6 Sécurité de l'information et protection des données

L'application web sera sécurisée de toute type d'injection possible, html, css, js et sql. Les mots de passe seront encryptés de sorte qu'il ne soit pas lisible dans la base de données. Les différentes requêtes seront protégées par un rôle sur l'api, seulement ceux qui auront le bon rôle pourront les appeler.

## 2.7 Test technologique

### 2.7.1 VueJs

#### 2.7.1.1 Documentation

La documentation de VueJS est disponible ici : <https://vuejs.org/guide/introduction.html>

### 2.7.1.2 Création d'un projet VueJS

Pour créer un nouveau projet en vue il faut au préalable télécharger nodeJS afin de disposer de la commande « npm install » qui permet de télécharger des paquets de librairies.

Une fois que nodeJS est installé on peut utiliser la commande :

```
> npm init vue@latest
```

Cette commande installera et exécutera create-vue, l'outil officiel de création de projets Vue. Il faut ensuite répondre à des questions qui nous permettent de choisir d'installer certaines fonctionnalités de base :

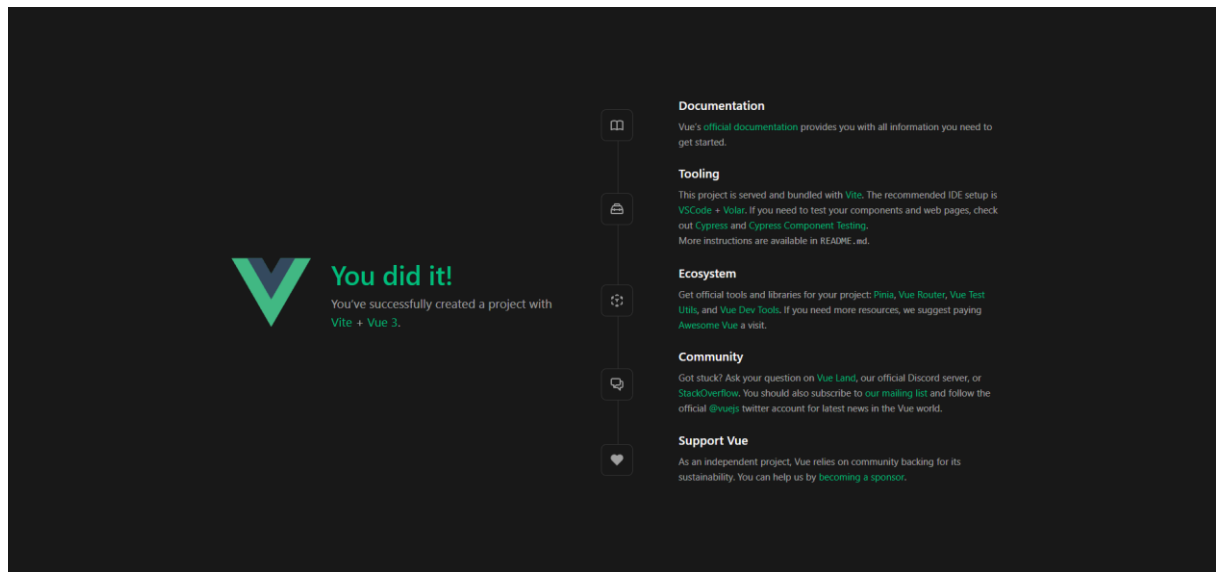
```
✓ Project name: ... <your-project-name>
✓ Add TypeScript? ... No / Yes
✓ Add JSX Support? ... No / Yes
✓ Add Vue Router for Single Page Application development? ... No / Yes
✓ Add Pinia for state management? ... No / Yes
✓ Add Vitest for Unit testing? ... No / Yes
✓ Add Cypress for both Unit and End-to-End testing? ... No / Yes
✓ Add ESLint for code quality? ... No / Yes
✓ Add Prettier for code formatting? ... No / Yes

Scaffolding project in ./< nouveau-projet >...
Done.
```

Il suffit ensuite de se déplacer dans le projet qui vient d'être créé puis lancer le projet avec ces commandes.

```
> cd <nouveau-projet>
> npm install
> npm run dev
```

Et voilà l'application vue a été créée :



Page de création d'une application VueJS

## 2.7.2 SpringBoot

### 2.7.2.1 Introduction

SpringBoot est un framework open source pour le développement d'applications Java. Il simplifie le processus de création d'applications en fournissant une configuration et une infrastructure par défaut, permettant aux développeurs de se concentrer sur la logique métier. Il propose une approche "convention plutôt que configuration" en fournissant des valeurs par défaut intelligentes et en simplifiant la configuration. SpringBoot permet de créer rapidement des applications autonomes, des services web RESTful et des micro-services, en offrant des fonctionnalités telles que la gestion des dépendances, la surveillance de l'application et la gestion de la sécurité. Il s'intègre étroitement avec l'écosystème Spring, ce qui en fait un choix populaire pour le développement d'applications Java robustes et évolutives.

### 2.7.2.2 Documentation

La documentation de SpringBoot est disponible ici : <https://spring.io/quickstart>

### 2.7.2.3 Création d'un projet SpringBoot

Pour créer un nouveau projet SpringBoot leur site propose un outil pour générer un nouveau projet : <https://start.spring.io/>.

Depuis ce site on peut donc configurer et ajouter des dépendances à notre projet. Une fois terminée, il suffit de générer et télécharger le zip. Le projet est généré automatiquement. Il n'y a rien à faire de plus.

#### 2.7.2.4 Installer swagger

Swagger est un outil qui permet de documenter une API de manière efficace, il génère également une page web qui regroupe les diverses requêtes possibles à l'api.

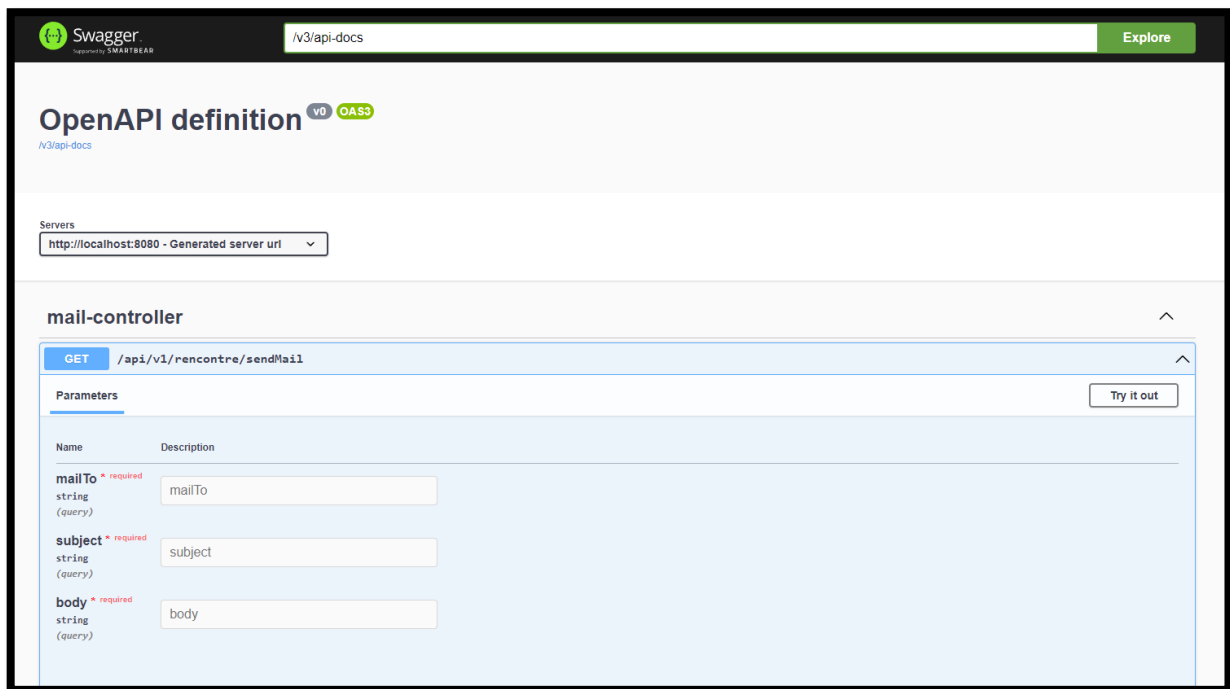
Pour l'installer sur SpringBoot il suffit d'ajouter ces librairies au fichier pom.xml

```
<!-- Swagger UI Dependency -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.0.2</version>
</dependency>
```

Il faut ensuite définir un Controller et créer une méthode qui correspondra à la requête de l'api.

```
@RestController
@CrossOrigin(origins = "*")
@RequestMapping("api/v1/rencontre")
public class MailController {
    @Autowired
    private EmailSenderService senderService;

    @GetMapping(path = "/sendMail")
    public String triggerMail(String mailTo, String subject, String body) throws
MessagingException {
        senderService.sendSimpleEmail(mailTo, subject, body);
        return "Mail bien envoyé";
    }
}
```



### Interface de swagger

Swagger détectera le Controller et on peut tester la requête depuis swagger après ça.

#### 2.7.2.5 Envoyer des mails

Pour envoyer des mails avec SpringBoot, la librairie JavaMail est recommandée. Il faut ajouter la dépendance dans le fichier pom.xml :

```
<!-- Java mail Dependency-->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-mail</artifactId>
</dependency>
```

Une fois que la librairie est installée, il faut configurer les propriétés du serveur d'envoi (SMTP) dans le fichier 'application.properties'.

```
#Mail configuration
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=<mail>
spring.mail.password=<mot-de-passe>
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

Pour le test technologique j'ai utilisé les serveurs de Gmail. J'ai également utilisé le port 587 pour que ce soit le plus sécurisé possible. Il faut ensuite créer un service qui enverra les mails.

```
@Service
public class EmailSenderService {
```



```

@Autowired
private JavaMailSender mailSender;

public String sendSimpleEmail(String toEmail, String subject, String body) {
    String result = "";
    try {
        MimeMessage mimeMessage = mailSender.createMimeMessage();
        MimeMessageHelper mimeMessageHelper = new MimeMessageHelper(mimeMessage,
true);
        mimeMessageHelper.setFrom("killian.pasche7@gmail.com");
        mimeMessageHelper.setTo(toEmail);
        mimeMessageHelper.setSubject(subject);
        mimeMessageHelper.setText(body);
        mailSender.send(mimeMessage);
        result = "Mail envoyé !";
    } catch (MessagingException e) {
        result = "Erreur envoie de mail : " + e.getMessage();
    }
    return result;
}
}

```

Pour tester la méthode, j'ai créé une requête qui permet d'envoyer un mail depuis le swagger :

```

@RestController
@CrossOrigin(origins = "*")
@RequestMapping("api/v1/mail")
public class MailController {
    @Autowired
    private EmailSenderService senderService;

    @GetMapping(path = "/sendMail")
    public String triggerMail(String mailTo, String subject, String body)
    {
        return senderService.sendSimpleEmail(mailTo, subject, body);
    }
}

```



J'ai ensuite lancé l'application et entré une adresse mail ainsi qu'un sujet et un message pour tester :

mail-controller

GET /api/v1/mail/sendMail

Parameters

Cancel

Name	Description
mailTo * required	
string (query)	killian.pasche@studentfr.ch
subject * required	
string (query)	Test envoi de mail
body * required	
string (query)	Ceci est un test

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/v1/mail/sendMail?mailTo=killian.pasche@studentfr.ch&subject=Test%20envoi%20de%20mail&body=Ceci%20est%20un%20test' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/v1/mail/sendMail?mailTo=killian.pasche@studentfr.ch&subject=Test%20envoi%20de%20mail&body=Ceci%20est%20un%20test
```

Server response

Code	Details
200	

Response body

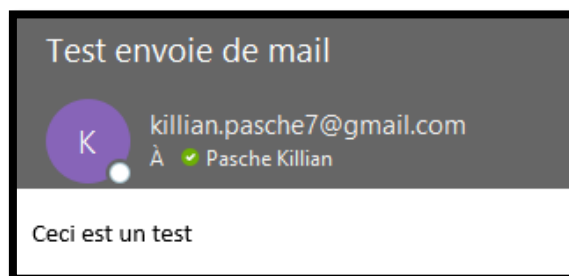
```
Mail envoyé !
```

Response headers

```
connection: keep-alive
content-length: 14
content-type: text/plain; charset=utf-8
date: Mon, 12 May 2023 07:01:21 GMT
keep-alive: timeout=60
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

Affichage du résultat de la requête sur swagger

Comme on peut le voir ci-dessous j'ai bien reçu le mail :



Mail reçu



## 3 Analyse

### 3.1 Use Case

Ce diagramme Use Case correspond à toutes les fonctionnalités principales de l'application FutNet Single Master. Chaque couleur représente une action différente.

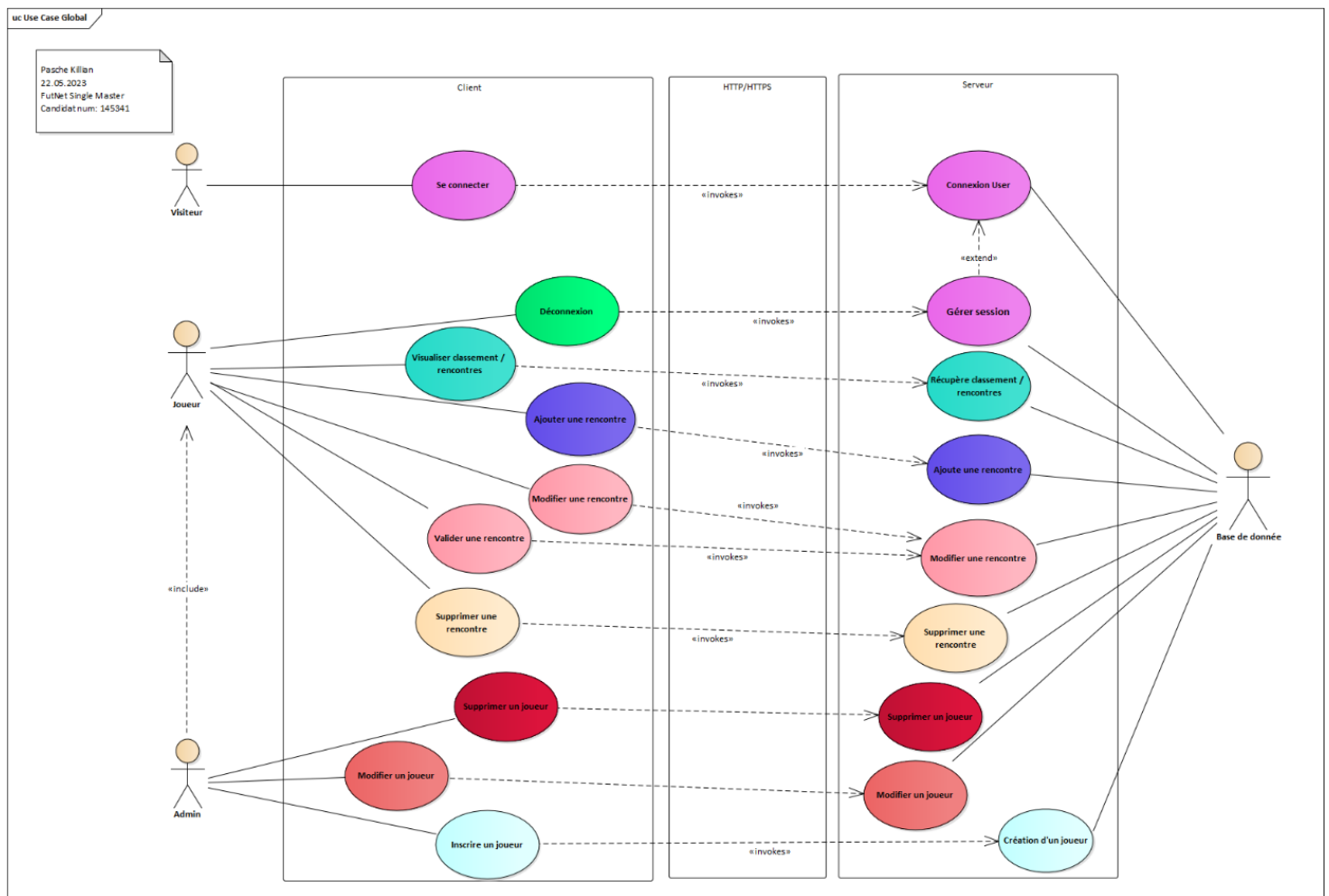


Diagramme Use Case

#### 3.1.1 Architecture

- Client : représente le côté front-end de l'application.
- HTTP/HTTPS : représente la liaison http/s entre le client et le serveur.
- Serveur : représente le côté back-end de l'application.



### 3.1.2 Acteurs

#### 3.1.2.1 Visiteur

Un utilisateur non-connecté qui arrive sur l'application.

- Se connecter : Permet à un visiteur de se connecter.

#### 3.1.2.2 Joueur

Un utilisateur connecté qui a accès à l'application.

- Déconnexion : Permet à un joueur de se déconnecter.
- Visualiser classement / rencontre : Récupère le classement et les matchs pour les affichés.
- Ajouter une rencontre : Permet à un joueur d'ajouter une rencontre.
- Modifier une rencontre : Permet à un joueur de modifier une rencontre qui lui appartient.
- Valider une rencontre : Permet à un joueur de modifier l'état d'une rencontre qui ne lui appartient pas.
- Supprimer une rencontre : Permet à un joueur de supprimer une rencontre qui lui appartient s'il n'est pas validé.

#### 3.1.2.3 Admin

Un utilisateur connecté qui a accès à l'application admin. L'utilisateur admin inclut tous ce qu'un joueur peut faire.

- Supprimer un joueur : Permet à l'admin de supprimer un joueur s'il n'a joué aucun match.
- Modifier un joueur : Permet à l'admin de modifier les informations d'un joueur.
- Inscrire un joueur : Permet à l'admin d'inscrire un joueur.

#### 3.1.2.4 Base de données

La base de données s'occupe de gérer les données qu'on lui envoie.

## 3.2 Maquettes

Voici les maquettes des différentes pages qu'on pourra voir sur l'application web :



### 3.2.1 Login

Écran d'accueil pour les visiteurs, c'est via cette écran qu'ils pourront se connecter.

**Futnet Single Master**

## Connexion

Username  
👤 Votre username

Mot de passe  
🔒 Votre mot de passe

Connexion

### Page de login

### 3.2.2 Classement

Sur cette page les joueurs pourront voir le classement général des utilisateurs.

**Futnet Single Master**

MENU

- Classement
- Rencontre
- Inscription
- Options
- Aide
- Contact
- Log out

## Classement

Admin ▾

### Classement

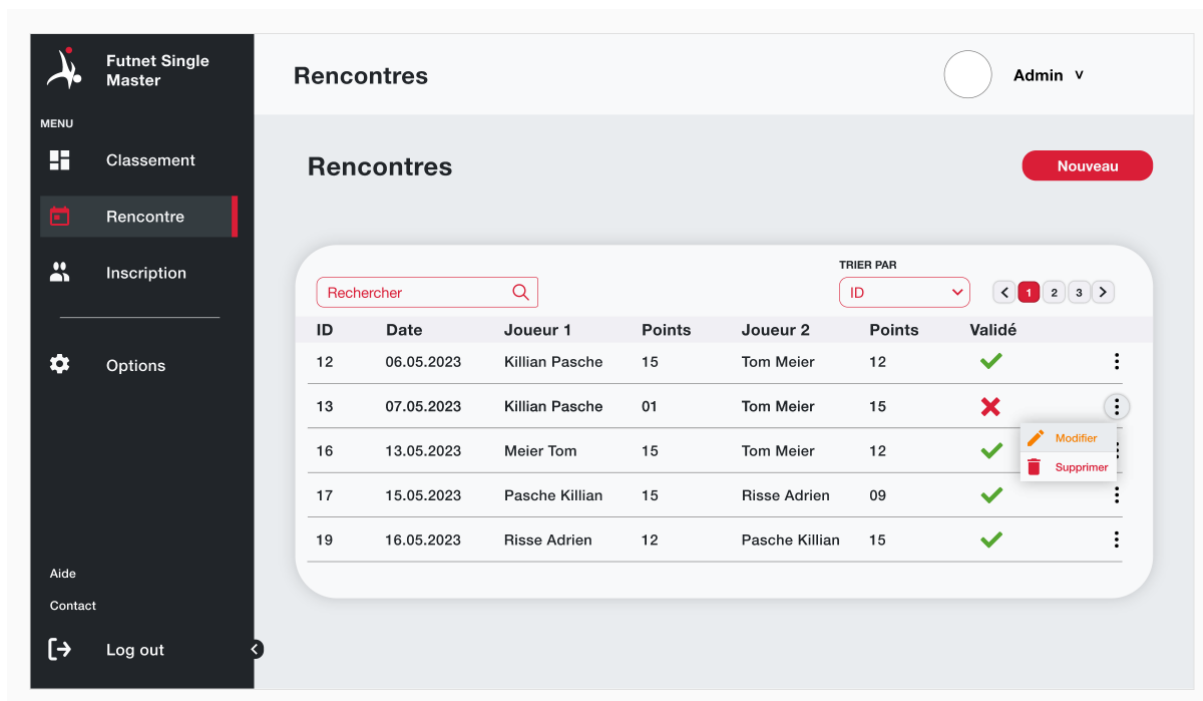
Nom	Équipe	Points	Date
Joueur_1	Barcelona FC	82	17.05.2023
Joueur_2	Barcelona FC	69	17.05.2023
Joueur_3	Real Madrid	65	07.05.2023
Joueur_4	Sevilla FC	63	09.05.2023
Joueur_5	Villareal	56	12.05.2023

< 1 2 3 >

### Page du classement

### 3.2.3 Rencontre

C'est sur cette page que les joueurs pourront voir les matches, en ajouter, en modifier et en valider.



**Rencontres**

Admin ▾

**Rencontres** Nouveau

Rechercher  TRIER PAR ID ▾ < 1 2 3 >

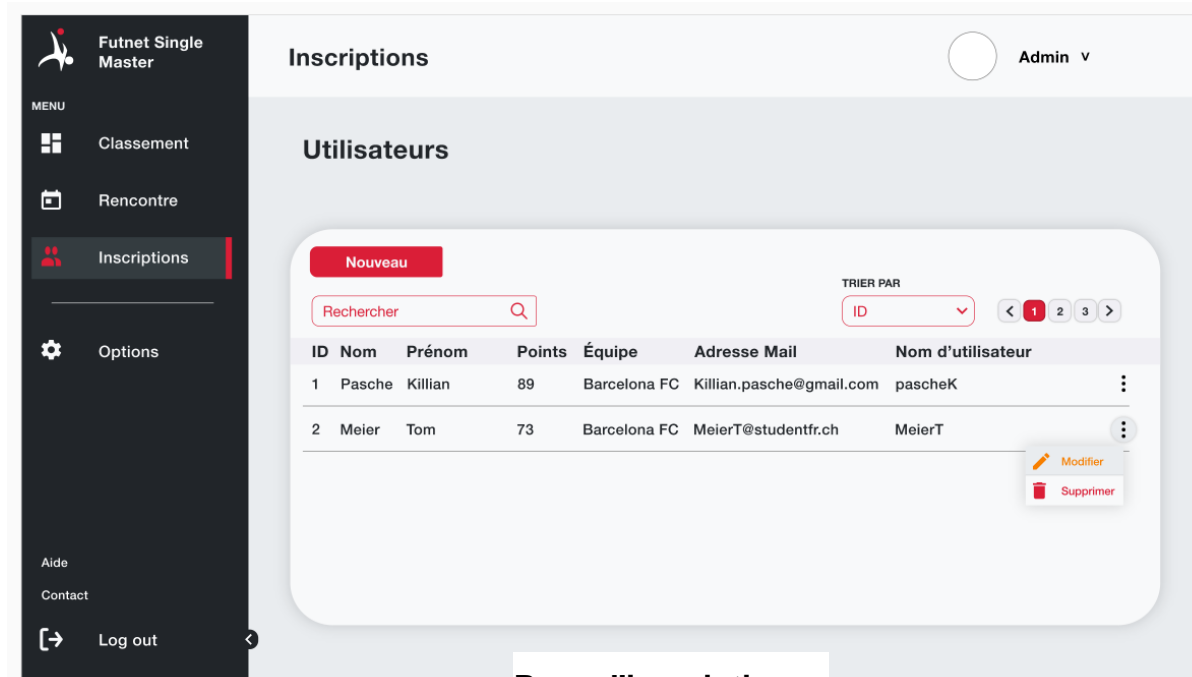
ID	Date	Joueur 1	Points	Joueur 2	Points	Validé
12	06.05.2023	Killian Pasche	15	Tom Meier	12	✓
13	07.05.2023	Killian Pasche	01	Tom Meier	15	✗
16	13.05.2023	Meier Tom	15	Tom Meier	12	✓
17	15.05.2023	Pasche Killian	15	Risse Adrien	09	✓
19	16.05.2023	Risse Adrien	12	Pasche Killian	15	✓

Modifier  
Supprimer

Page des rencontres

### 3.2.4 Inscription

C'est dans cette page que les admins pourront ajouter, modifier, supprimer des utilisateurs.



**Inscriptions**

Admin ▾

**Utilisateurs** Nouveau

Rechercher  TRIER PAR ID ▾ < 1 2 3 >

ID	Nom	Prénom	Points	Équipe	Adresse Mail	Nom d'utilisateur
1	Pasche	Killian	89	Barcelona FC	Killian.pasche@gmail.com	pascheK
2	Meier	Tom	73	Barcelona FC	MeierT@studentfr.ch	MeierT

Modifier  
Supprimer

Page d'inscription



### 3.3 Diagramme d'activité / séquence

#### 3.3.1 Inscription de joueur

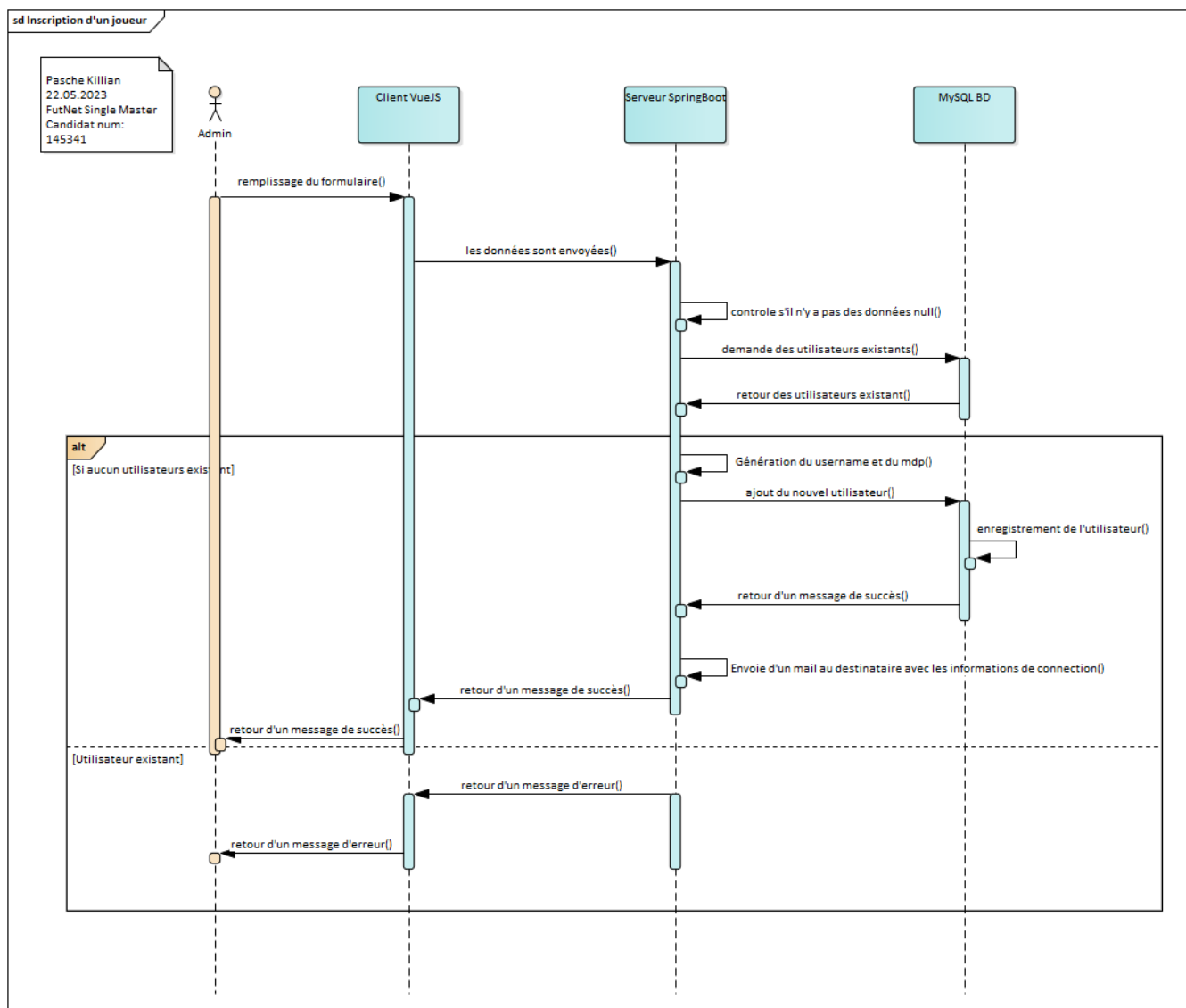
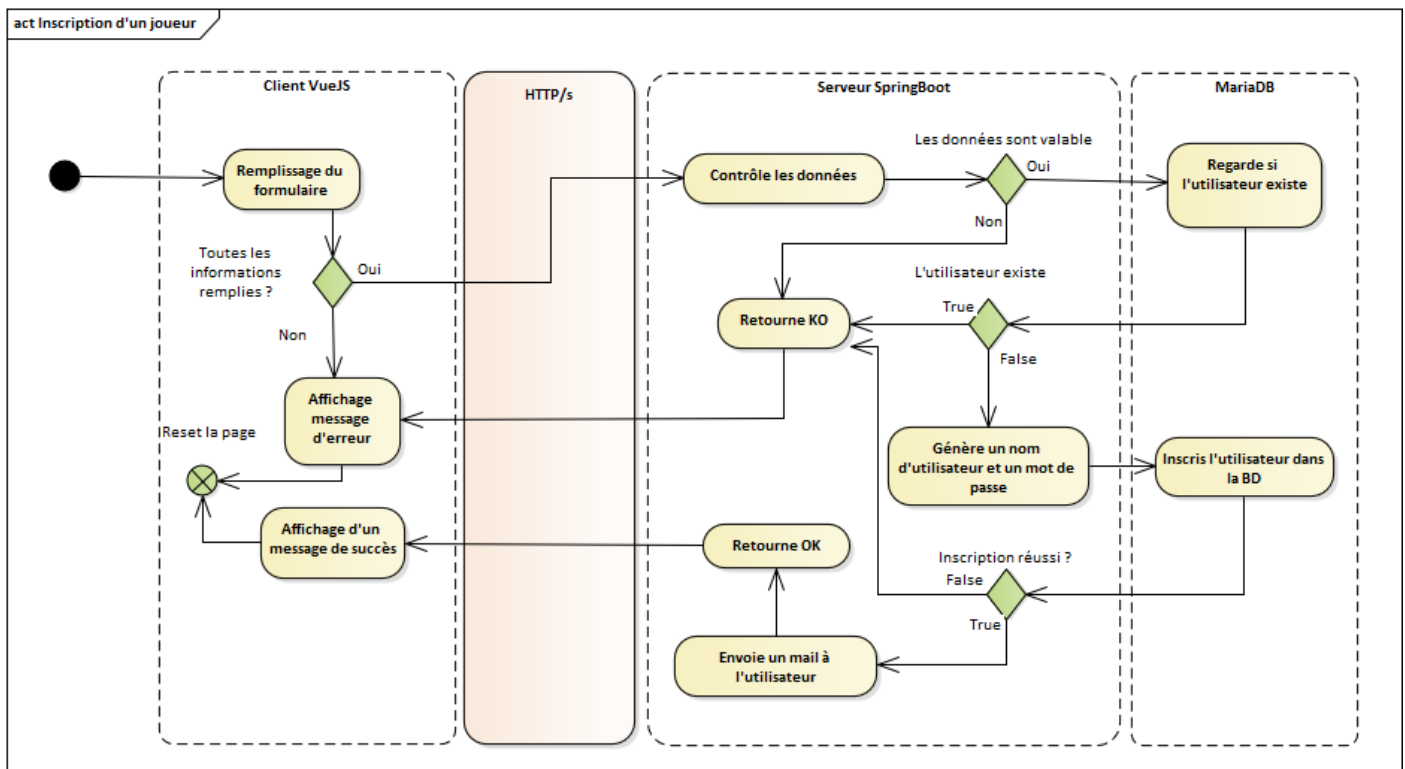


Diagramme de séquence (Inscription d'un joueur)



**Diagramme d'activité (Inscription d'un joueur)**

Pour inscrire un joueur, l'admin devra remplir un formulaire. Une fois remplie, la partie client va s'occuper d'envoyer les données au serveur si toutes les informations sont remplies. Si les informations ne sont pas remplies, un message d'erreur va s'afficher et la page sera rafraîchis. Une fois que le serveur a reçu les données, il va s'occuper de les contrôler. Si elles sont valables il va demander à la base de données si un utilisateur avec ces données existe, si non un mot de passe ainsi qu'un nom d'utilisateur seront générés. Une fois la génération terminée l'utilisateur sera enregistré dans la base de données. Si l'inscription est réussie, un mail sera envoyé à l'utilisateur avec le mot de passe et le nom d'utilisateur. Le serveur ensuite retourne l'état de la requête pour que le client affiche le message en conséquence, c'est-à-dire que si les données ne sont pas valables, l'utilisateur existe ou que l'inscription a échoué l'application retournera KO et affichera un message d'erreur. En cas contraire, l'application retournera OK et affichera un message de succès.



### 3.3.2 Se connecter

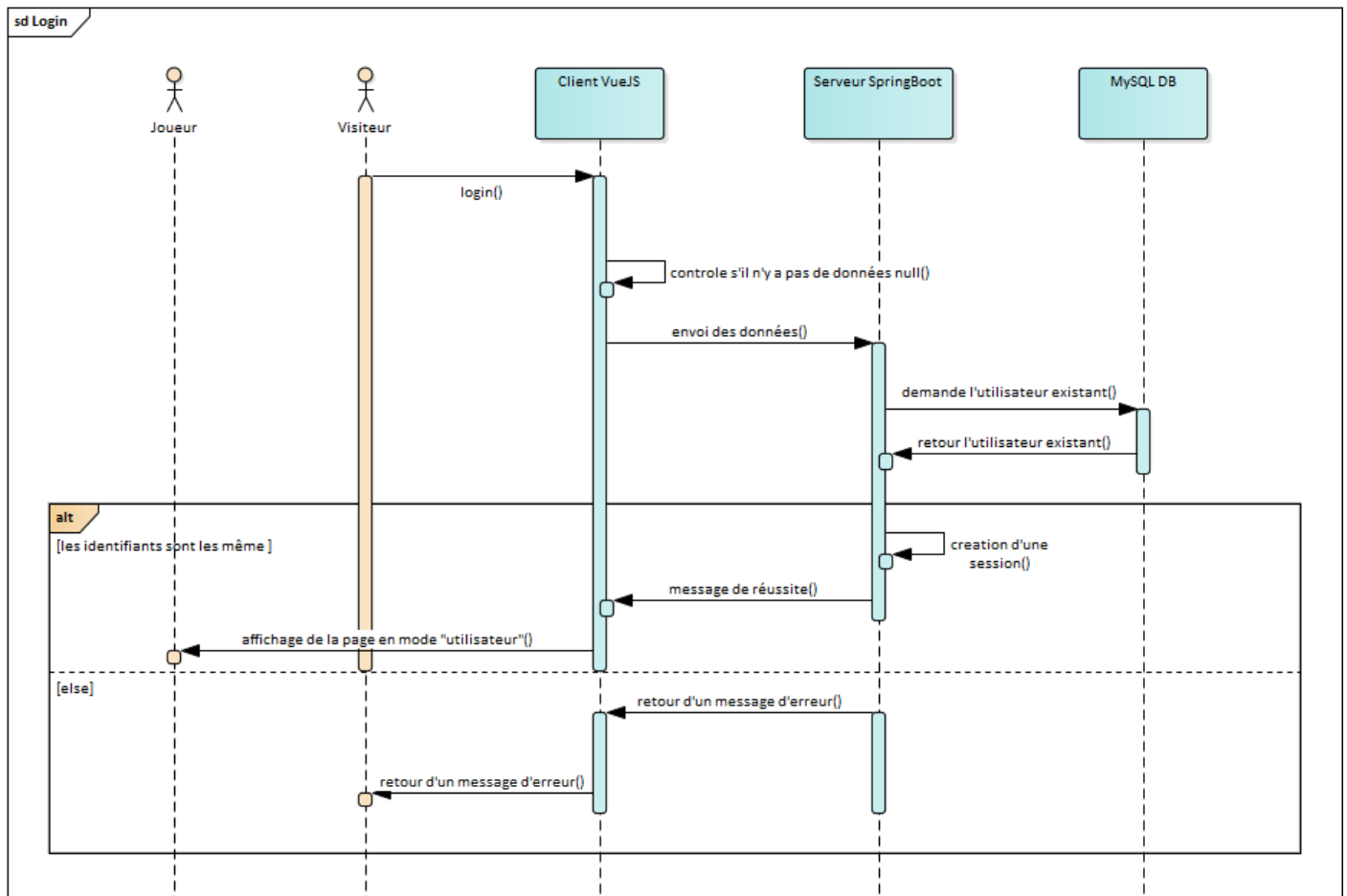


Diagramme de séquence (connexion)

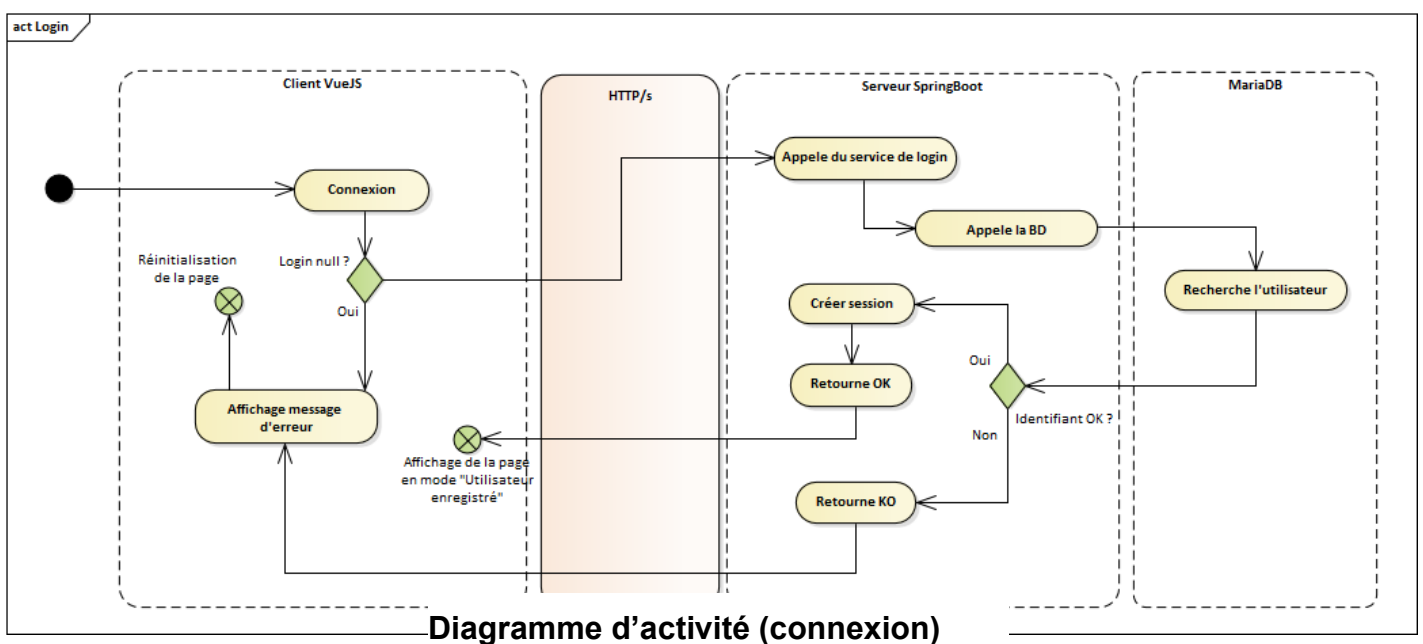


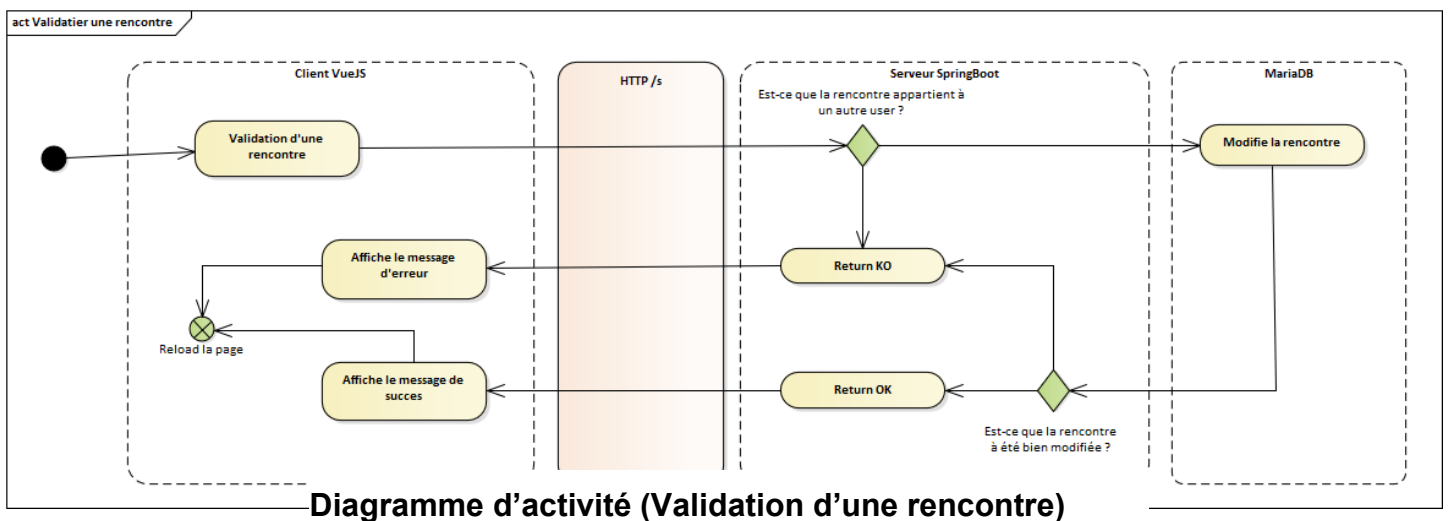
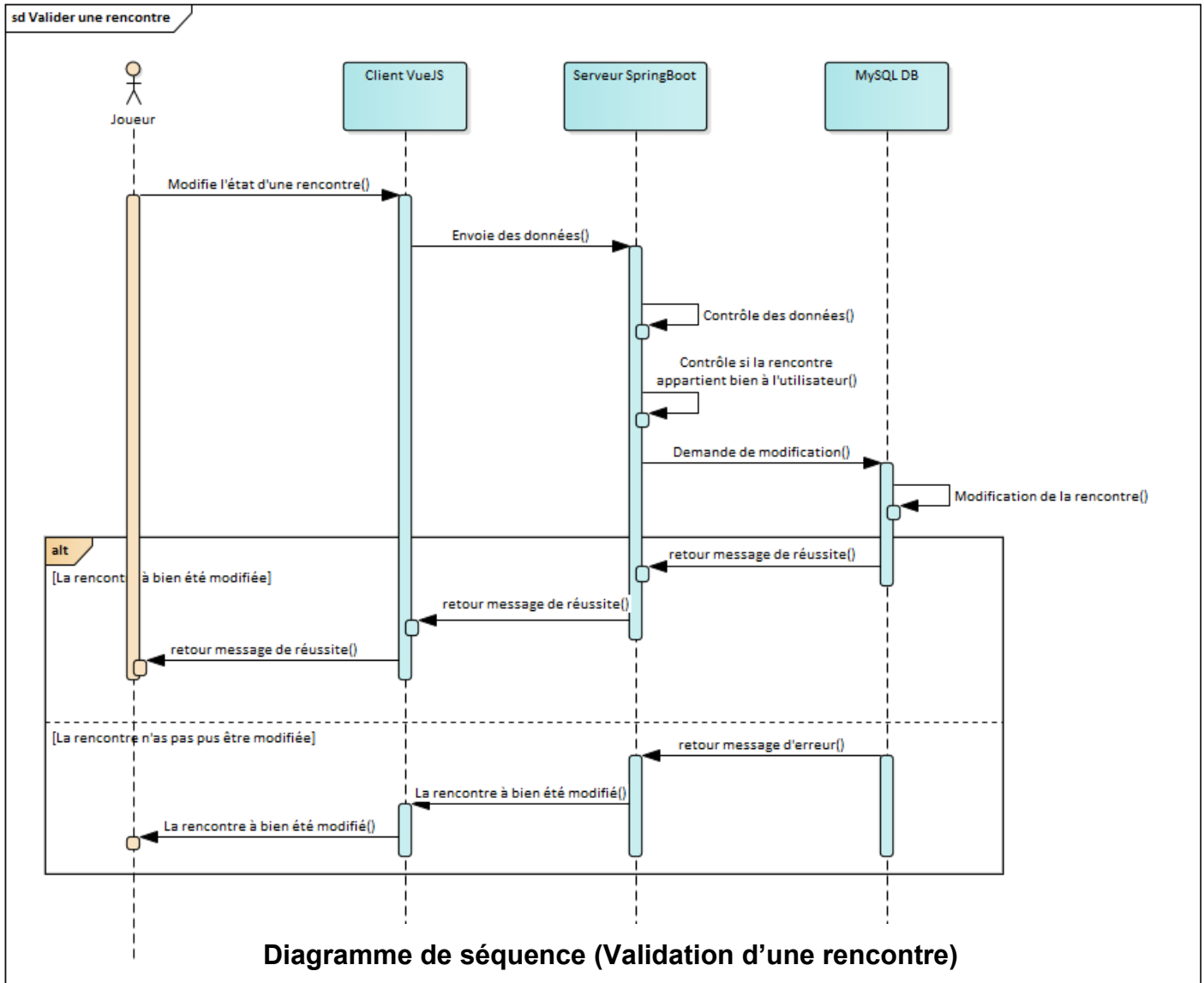
Diagramme d'activité (connexion)



Lorsqu'un visiteur voudra se connecter sur l'application, il devra entrer son login et le client va vérifier les données, si elles sont nulles un message d'erreur sera affiché, sinon les données seront transmises au serveur. Le serveur va s'occuper d'appeler la base de données pour savoir si un utilisateur existe avec ce login, si oui il va créer une session et retourner ok, le client va ensuite afficher la page en mode « utilisateur enregistré ». Si non, il va retourner KO et afficher un message d'erreur sur le client.



### 3.3.3 Valider une rencontre







Lorsqu'un joueur valide un match le client enverras la rencontre au serveur, le serveur s'occupera de contrôler que la rencontre appartient bien au joueur, si elle appartient bien au joueur la base de données modifiera la rencontre. Si la rencontre a bien été modifier le serveur retournera OK et le client affichera un message de succès. Si la rencontre n'a pas été modifié ou que la rencontre appartient à un autre joueur il retournera KO et le client affichera un message d'erreur.

### 3.3.4 Supprimer une rencontre

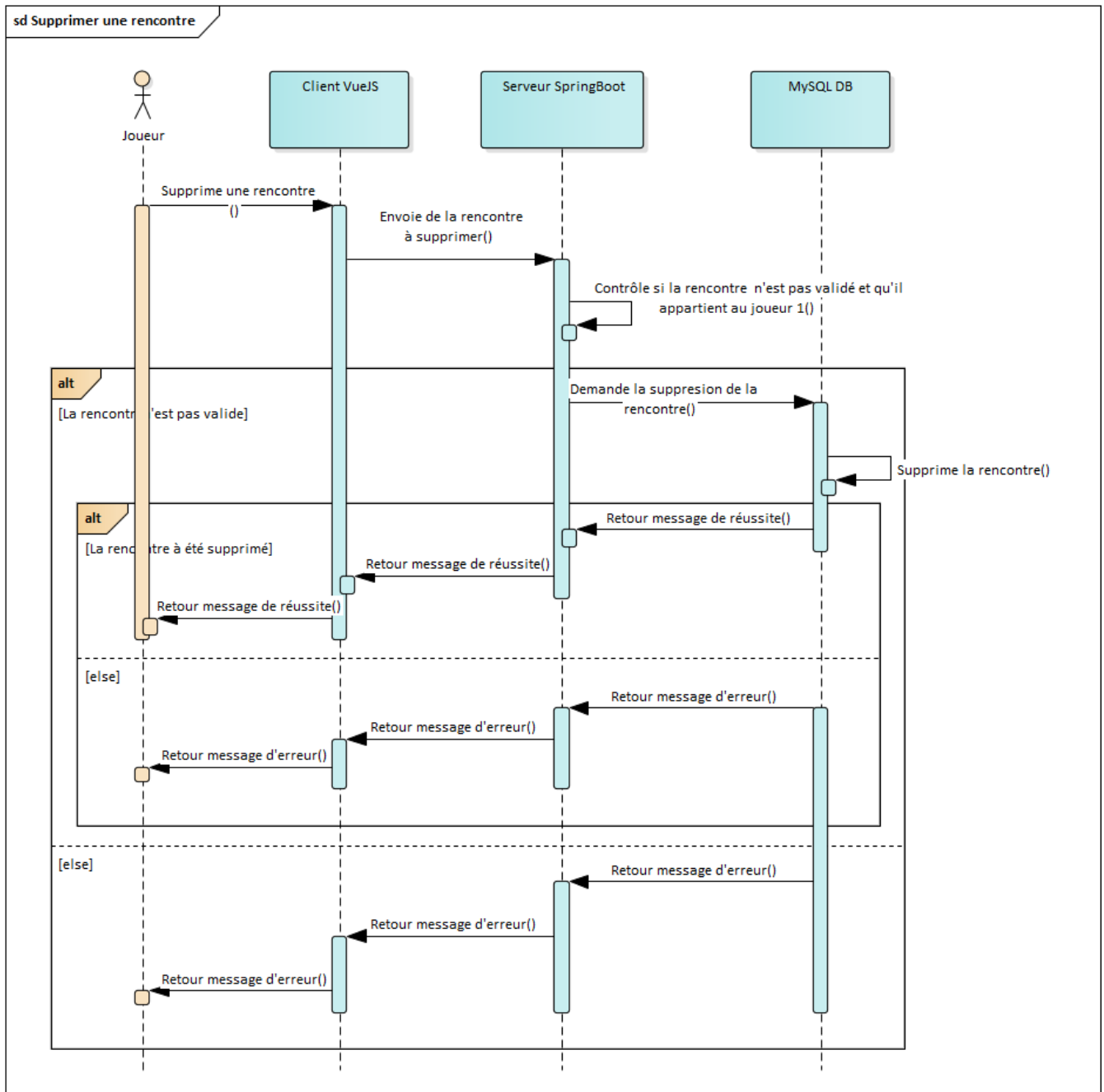
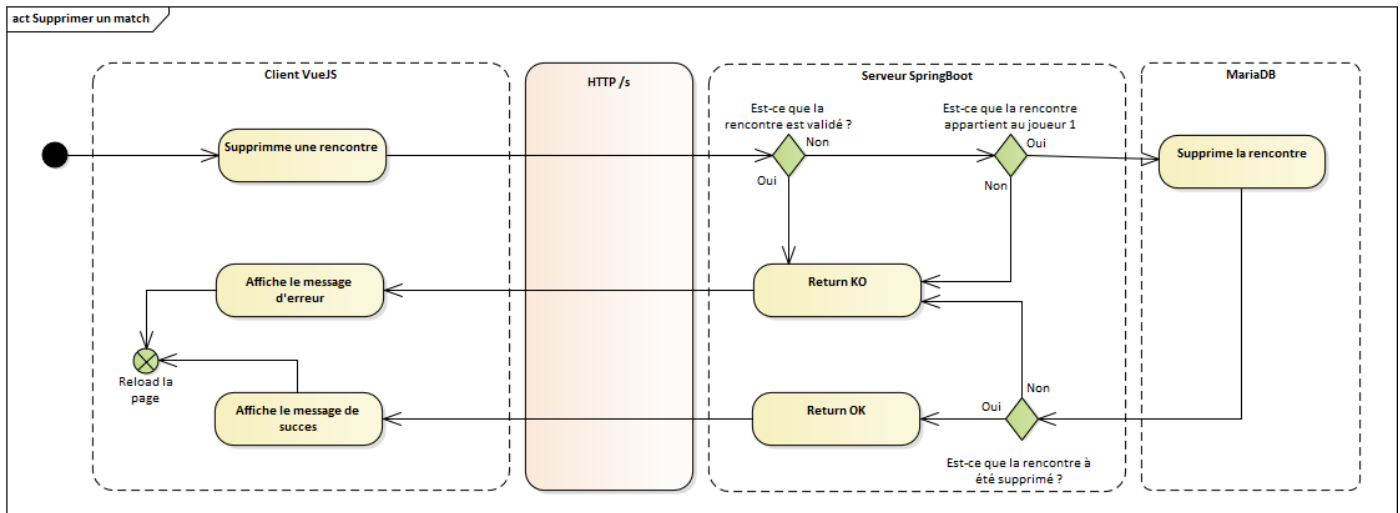


Diagramme de séquence (Supprimer une rencontre)



### Diagramme d'activité (Supprimer une rencontre)

Lorsqu'un joueur supprimera une rencontre, la rencontre sera envoyée au serveur SpringBoot. Le serveur vérifiera si la rencontre a déjà été validée. Il vérifie également si la rencontre envoyée appartient au joueur qui l'a envoyé et qu'il soit le joueur 1 de cette rencontre. Si ces deux tests sont passés, la rencontre sera alors supprimée de la base de données. Si la rencontre a bien été supprimée un message de succès sera retourné. En cas contraire, si la rencontre est valide, qu'elle n'appartient pas au joueur 1 ou qu'elle n'a pas été supprimée, un message d'erreur sera alors retourné.



### 3.3.5 Modifier une rencontre

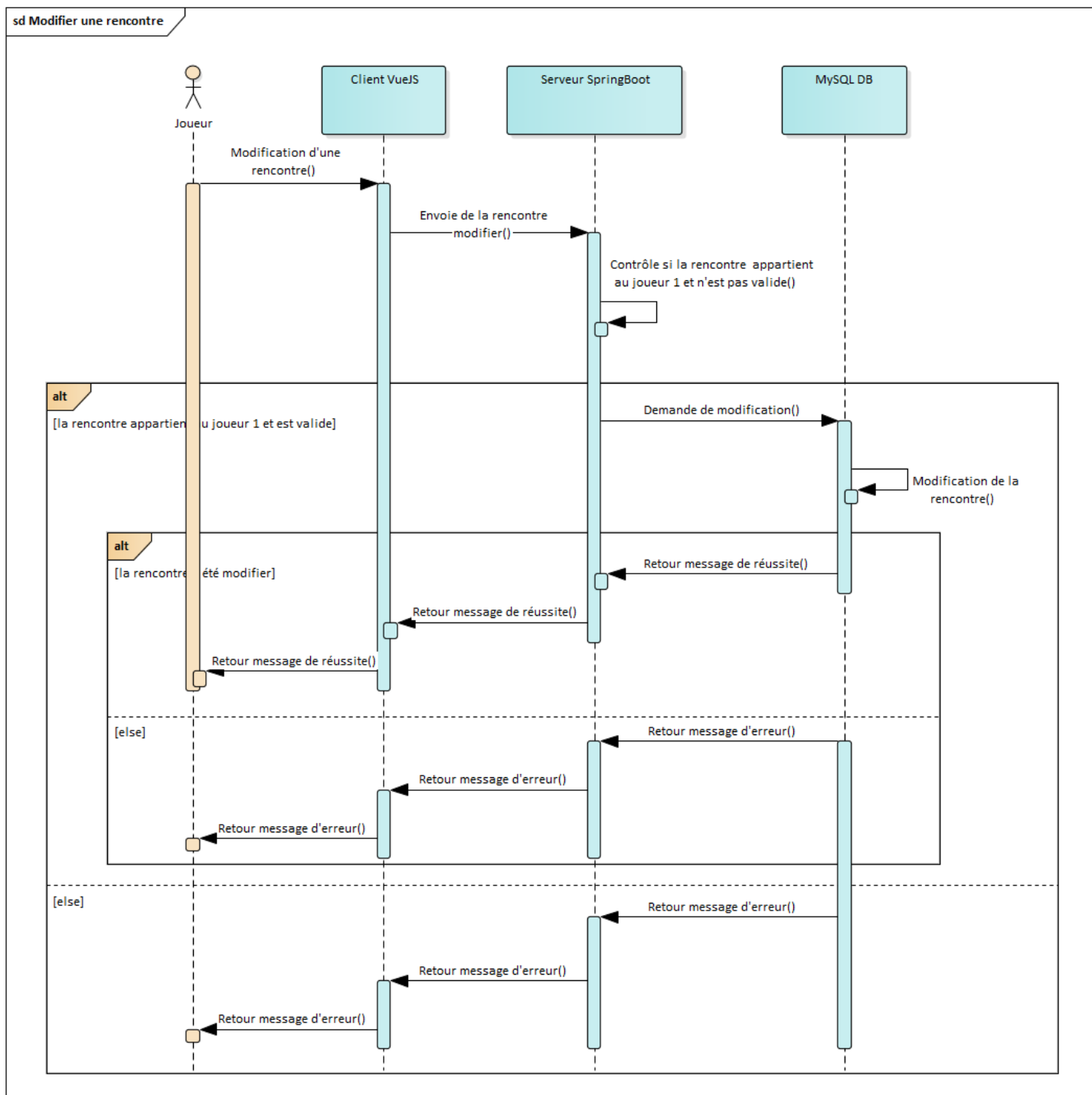
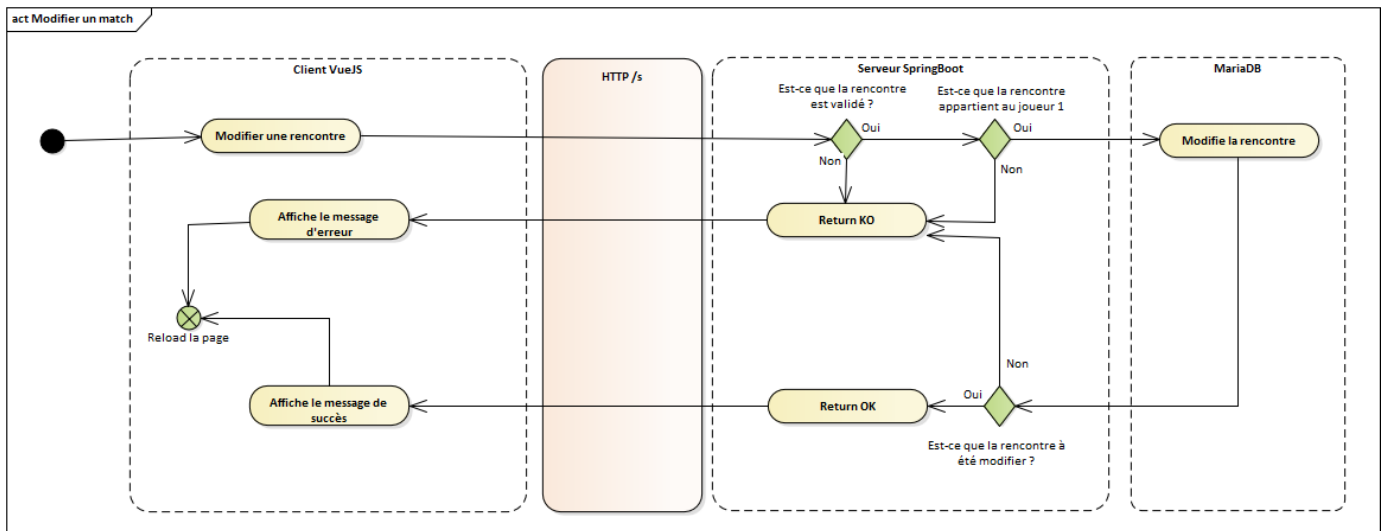


Diagramme de séquence (Modification d'une rencontre)



### Diagramme d'activité (Modification d'une rencontre)

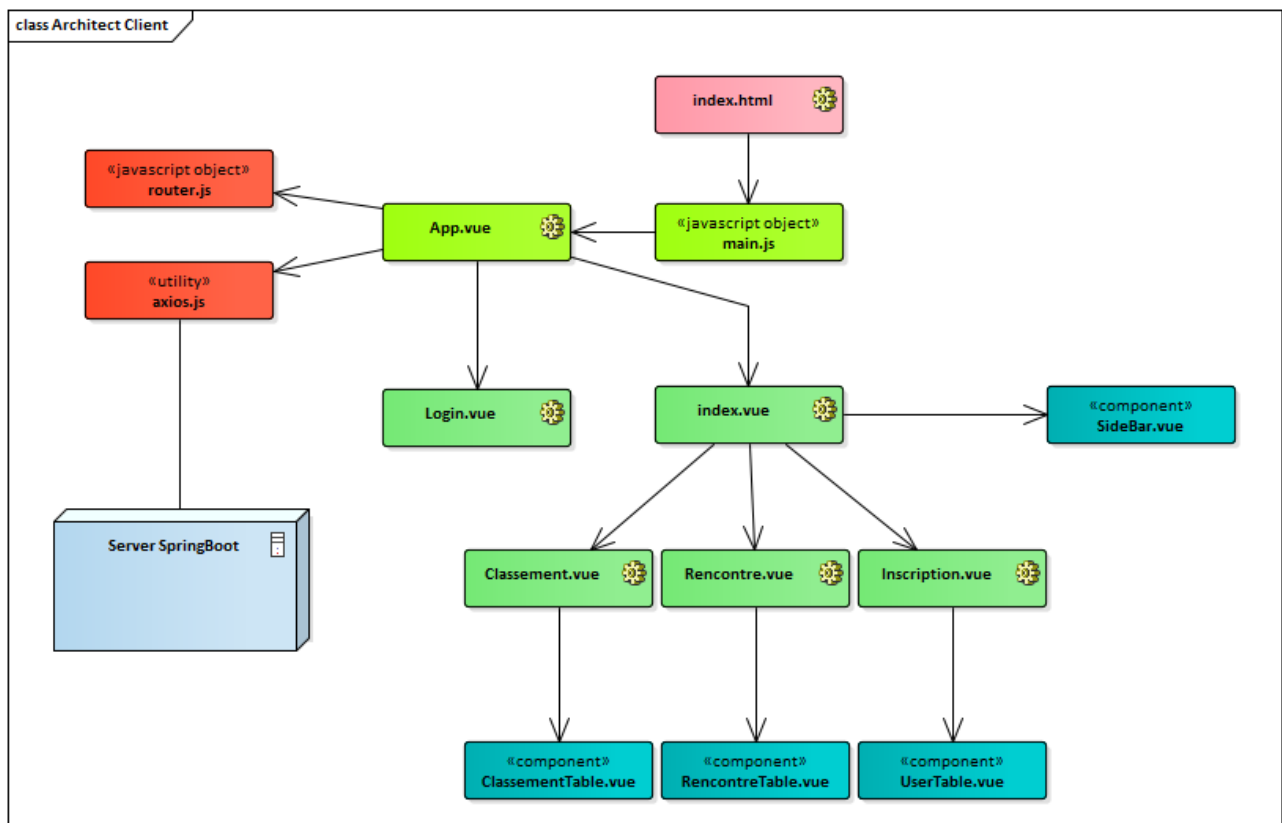
Lorsqu'un joueur aura modifié une rencontre, le client l'enverra au serveur. Le serveur regardera si la rencontre est validée et il vérifie également si la rencontre envoyée appartient au joueur qui l'a envoyée et qu'il soit le joueur 1 de cette rencontre. Si les tests sont réussis, la base de données modifiera la rencontre. Le serveur vérifie si la rencontre a été bien modifiée, si c'est le cas il retourne un message de succès. Si en cas contraire la rencontre n'a pas été modifiée ou bien la rencontre était valide ou elle n'appartenait pas au joueur 1, le serveur retournera un message d'erreur. Le client s'occupe d'afficher le message retourné par le serveur.



## 4 Concept

### 4.1 Diagramme de classe

#### 4.1.1 Partie client



**Diagramme de classe client**

Voici mon diagramme de classe de la partie cliente, qui représente la hiérarchie de mon application en VueJS :

**index.html** : C'est le point de départ de mon application, c'est dans ce fichier que VueJS va écrire son code.

**main.js** : c'est le fichier JavaScript qui initialisera un composant dans la page index.html. Il est également responsable de la configuration des plugins et de composants tiers qui seront utilisés.

**App.vue** : c'est la racine de l'application définie dans Vue.js.

**Login.vue/Index.vue/Classement.vue/Rencontre.vue/Inscription.vue** : Ce sont les différentes vues que l'application possède.



« component » : les éléments possédant cette balise sont les composants divers à mon application.

router.js axios.js : ce sont des bibliothèques de vue qui sont utilisées dans le projet. Router est utilisé pour faire une single app page et axios est utilisé pour faire des requêtes sur le serveur SpringBoot

#### 4.1.2 Partie serveur

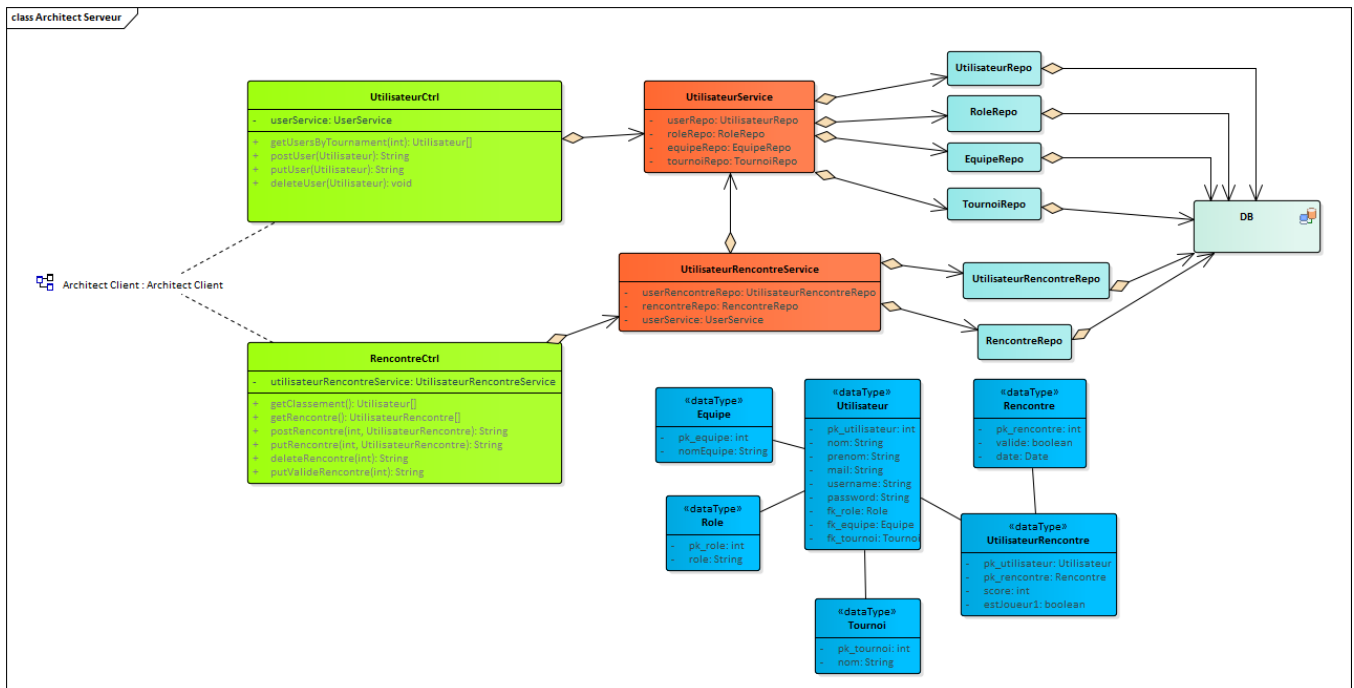


Diagramme de classe serveur

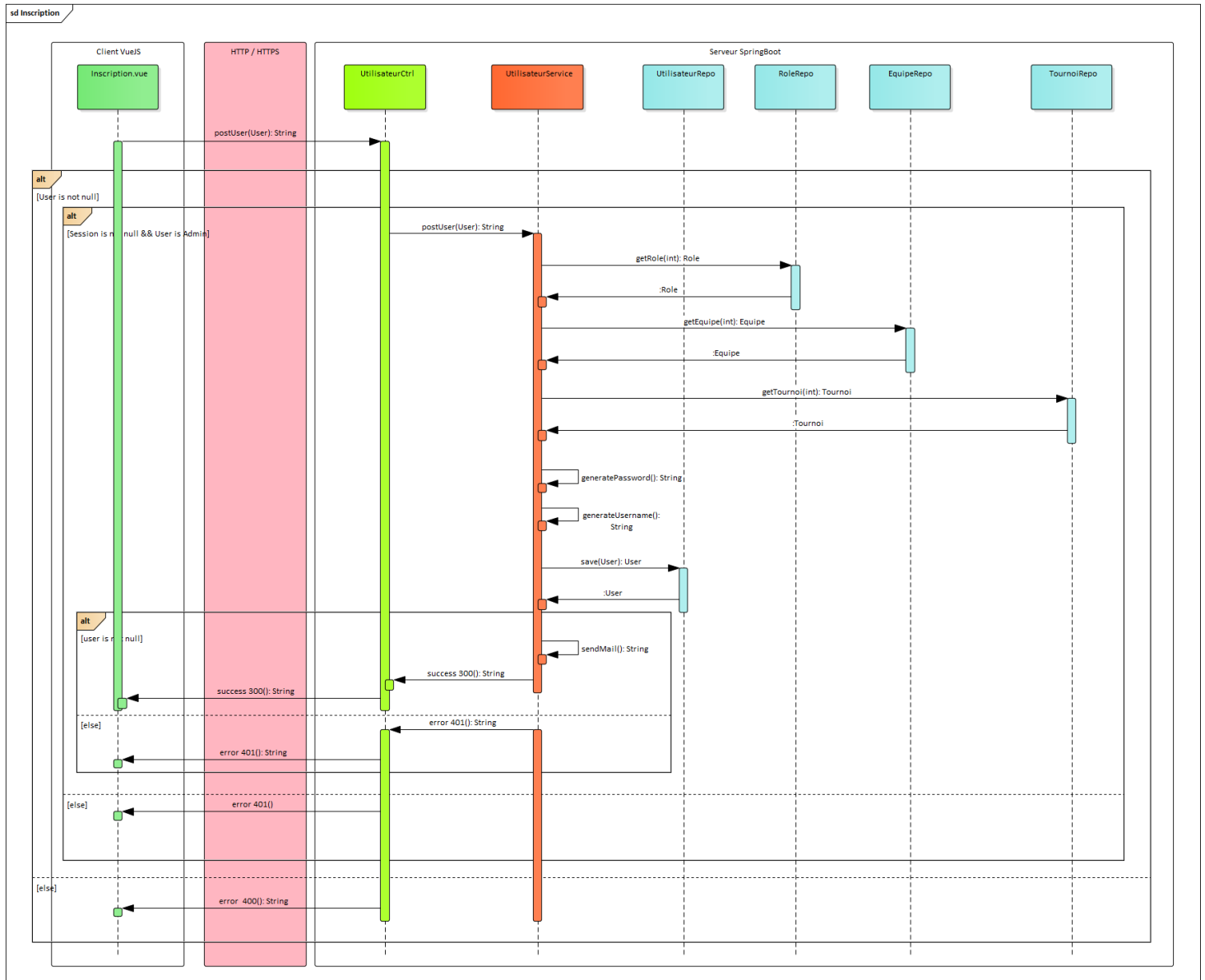
Voici le diagramme de classe de la partie serveur. Il est décomposé en 4 parties :

- Verte : cette partie contient les controllers, c'est le point d'entrée des requêtes.
- Orange : cette partie contient les services. Ce sont les services qui font la relation entre le ctrl et la DB. On peut comparer les services à des WRK.
- Bleu Claire : cette partie contient les dépôts (Repository). Les dépôts s'occupent de parler avec la DB.
- Bleu : Cette partie correspond aux entités, chaque entité représente une table de la DB.



## 4.2 Diagramme de séquence interactions

### 4.2.1 Inscription



### Diagramme de séquence interactions (Inscription)

Ce diagramme représente le déroulement de ce qui passe lors de l'inscription d'un utilisateur.

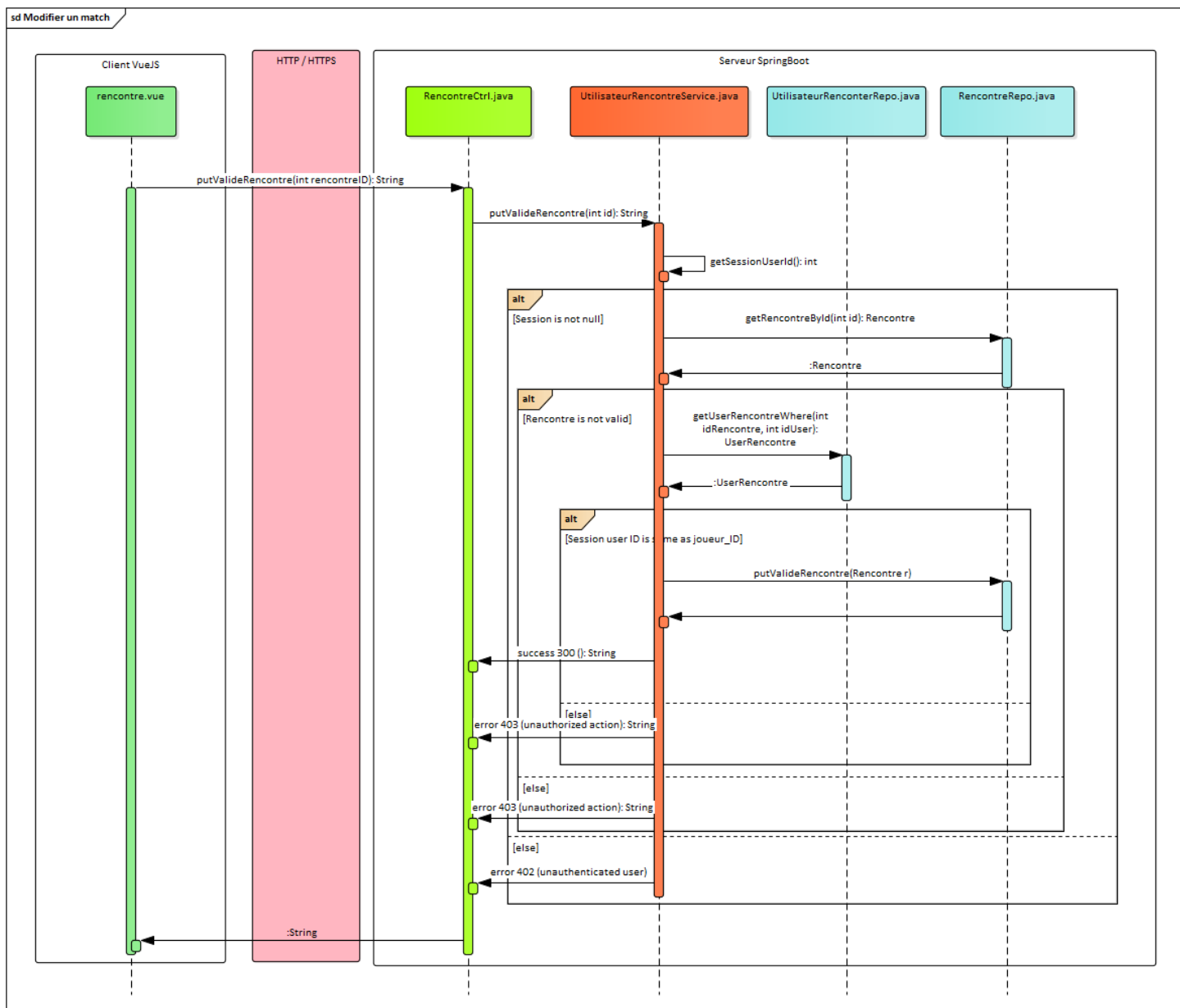
- 1) Le client envoie une requête au serveur SpringBoot avec un utilisateur en paramètre
- 2) Le serveur contrôle que l'utilisateur ne soit pas nul.
  - a. S'il est nul il retournera une erreur 400

- 3) Le serveur va ensuite récupérer toutes les informations de l'utilisateur dans son repository (rôle, équipe, tournoi)
- 4) Il va ensuite générer un mot de passe ainsi qu'un nom d'utilisateur.
- 5) L'utilisateur sera enregistré dans la base de données.
- 6) Un mail sera envoyé à l'utilisateur concerné pour lui communiquer ces informations de login.
- 7) Le serveur retournera un string si tout s'est bien passé.





## 4.2.2 Validation d'une rencontre



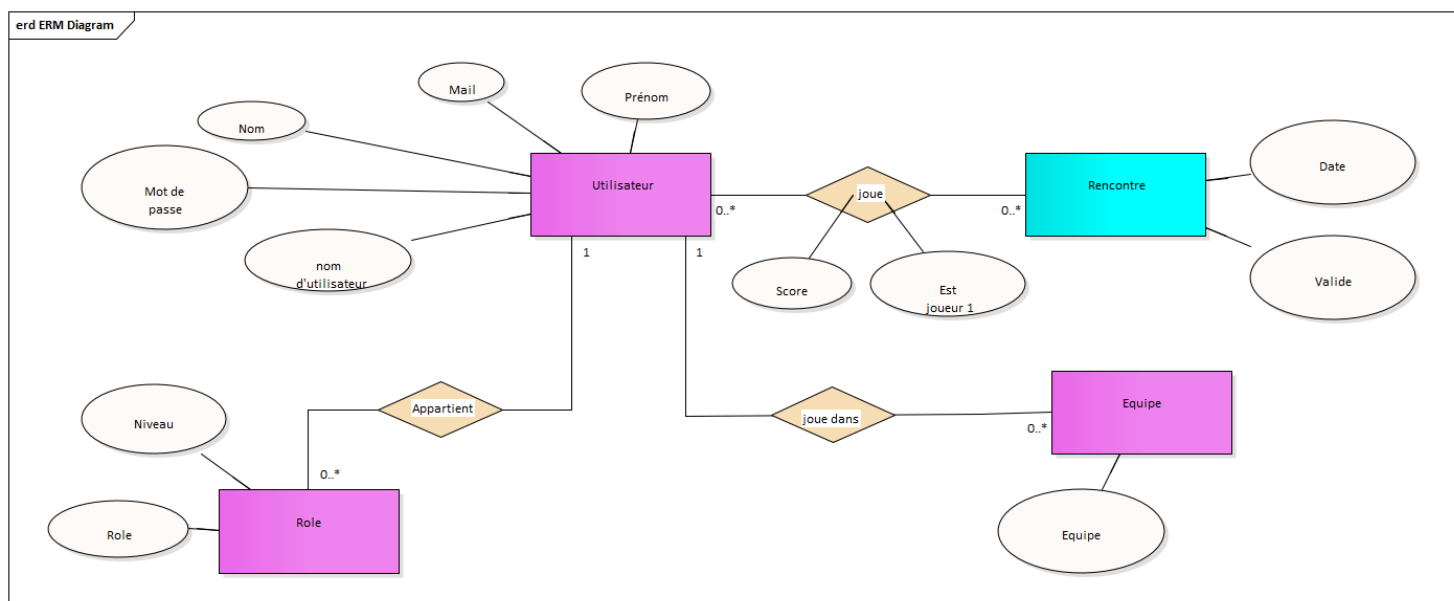
**Diagramme de séquence interactions (Validation d'une rencontre)**

Ce diagramme représente ce qu'il se passe lorsqu'un utilisateur souhaite valider une rencontre.

- 1) Le client envoie une requête au serveur SpringBoot avec l'identifiant de la rencontre en paramètre
- 2) Le Controller appelle le service qui s'occupe des rencontres.
- 3) Le Service contrôle que la session est existante
  - a. Si non : Renvoie un message d'erreur au client
- 4) Le service récupère la rencontre qui a été envoyée
  - a. Si la rencontre n'existe pas : renvoie un message d'erreur au client
- 5) Le service récupère l'entrée qui correspond à l'utilisateur connecté et la rencontre.
  - a. Si l'entrée n'est pas joueur 1 : Renvoie un message d'erreur au client
- 6) Modifie la rencontre pour la faire passer à valide
- 7) Renvoie un message de succès au client.



## 4.3 Diagramme d'entité-relation



**Diagramme d'entité-relation**

Le diagramme d'entité-relation représente les différentes relations que les entités ont envers les autres. Dans le diagramme, il y'a trois relations de type 0 à N (0...\*). Ces trois relations partent toutes de la table Utilisateur, vers les tables (Tournois, Rôle, Equipe) :

Un utilisateur participe à un tournoi.

Un utilisateur appartient à un rôle.

Un utilisateur joue dans une équipe.

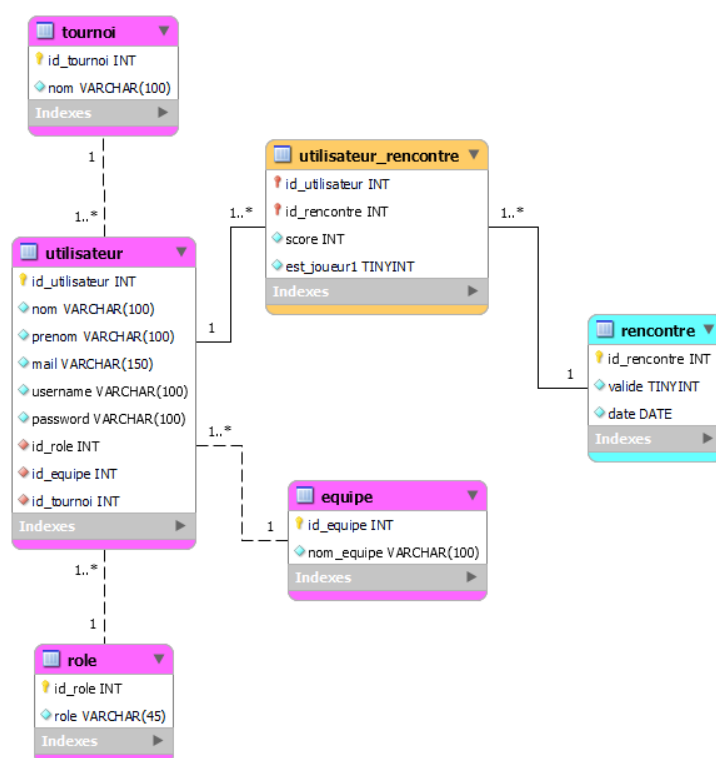
Il y'a une relation de plus qui part de la table utilisateur, cette relation est un peu spéciale, puisque c'est une relation N à N (0...\* a 0...\*). Cela signifie que lorsque la base de donnée sera créée il faudra une table de relation entre utilisateur et rencontre.

Un utilisateur joue 0 ou plusieurs rencontres. Une rencontre peut être jouée par 0 ou plusieurs joueurs.

La bulle blanche correspond aux attributs des entités.



## 4.4 Schéma relationnel de la base de données



**Schéma relationnel**

Pour la nomenclature de ma base de données, j'ai choisi une nomenclature assez standard, qui est souvent utilisé.

### 4.4.1 Tournoi

La table « tournoi » est une table un peu spéciale vue qu'elle n'est pas nécessaire au projet. Cette table est en prévision pour un futur où l'admin peut créer plusieurs tournois et choisir quel tournoi est actif. Pour le moment, cette table n'est pas vraiment nécessaire, elle est surtout utilisée pour que des utilisateurs soit attribués à un tournoi. Voilà ce qu'elle contient :

- id\_tournoi : l'identifiant unique du tournoi
- nom : le nom du tournoi

### 4.4.2 Utilisateur

La table « utilisateur » contient tous les joueurs / admin du tournoi, Voilà ce qu'elle contient :

- id\_utilisateur : l'identifiant unique d'un utilisateur.

- nom : le nom de l'utilisateur
- prenom : le prénom de l'utilisateur
- mail : l'adresse mail de l'utilisateur
- username : le nom d'utilisateur de l'utilisateur
- password : le mot de passe de l'utilisateur
- fk\_role : référence à la table « rôle », contient le rôle de l'utilisateur
- fk\_equipe : référence à la table « équipe », contient l'équipe de l'utilisateur.
- fk\_tournoi : référence à la table « tournoi », contient le tournoi dans lequel l'utilisateur est inscrit.

#### 4.4.3 Role

La table « role » contient les différents rôles qu'un utilisateur peut se voir attribuer. Voilà ce qu'elle contient :

- pk\_role : l'identifiant unique d'un rôle
- role : le rôle.

#### 4.4.4 Equipe

La table « equipe » contient les différentes équipes qu'un utilisateur peut se voir attribuer. Voilà ce qu'elle contient :

- pk\_equipe : L'identifiant unique d'une équipe
- nom\_equipe : le nom de l'équipe

#### 4.4.5 Rencontre

La table « rencontre » définit les matchs que les joueurs peuvent faire entre eux. Cette table ne contient pas tous les attributs d'une rencontre, elle est complétée par la table de relation utilisateur-rencontre. Voilà ce qu'elle contient :

- pk\_rencontre : l'identifiant unique de la rencontre.
- valide : si la rencontre a été validée.
- date : la date de la rencontre

#### 4.4.6 Utilisateur\_rencontre

La table « utilisateur\_rencontre » est une table de relation, c'est-à-dire qu'elle contient les relations entre utilisateur et rencontre. On pourrait supprimer cette table et définir deux utilisateurs ainsi que deux scores dans la table rencontre, mais je pense que c'est mieux de cette manière. Par exemple, lorsque l'on devra récupérer le score de chaque joueur, on aurait dû récupérer le nom de la colonne pour savoir si il est joueur 1 ou joueur 2. Le fait qu'une table de relations soit entre utilisateur et rencontre simplifie la récolte de résultats entre les deux tables.

Voilà ce qu'elle contient :

- id\_utilisateur : Référence à un utilisateur.
- id\_rencontre : Référence à une rencontre.
- score : Score du joueur.
- est\_joueur1 : définit le joueur 1 d'une rencontre.

« est\_joueur1 » permet de définir le joueur qui aura le droit de modifier ou valider une rencontre. Cela permettra aussi de définir le joueur qui aura le score 1 ou le score 2.

### 4.1 Concept de tests

Pour assurer le bon fonctionnement du projet, la conception de tests est un point important. Ils permettront si la solution est optimale et si elle traite tous les problèmes. Afin de tester tous les cas possible les tests seront effectués en « Whitebox ». Tous les tests réalisés seront documentés dans le chapitre [Test](#).

## 5 Réalisation

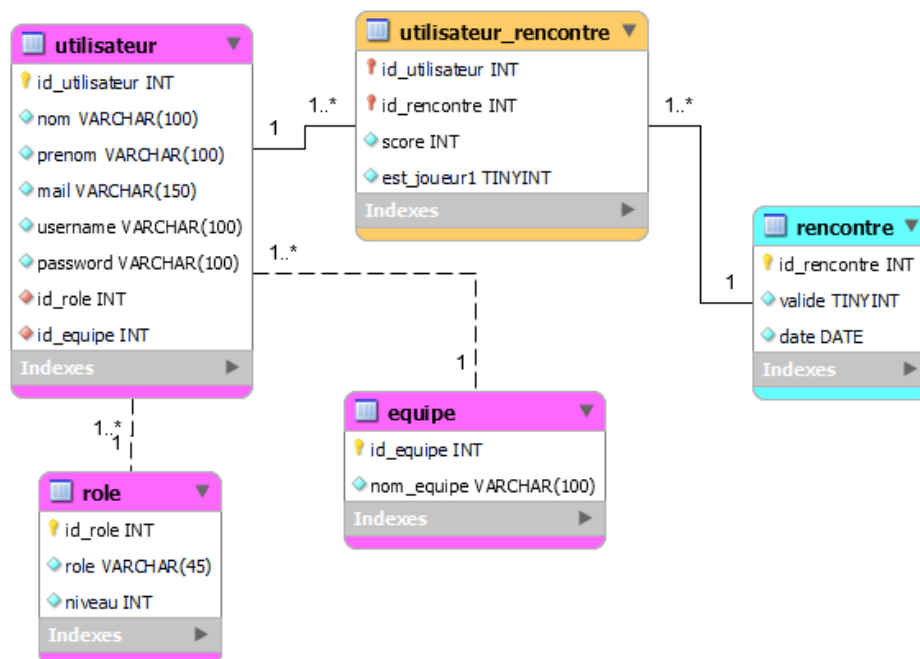
### 5.1 Tests fonctionnels

Tous les tests fonctionnels seront réalisés sur postman

### 5.2 Implémentation de la base de données

Lors de l'implémentation de la base de données, je me suis rendu compte que le schéma actuel va poser un problème dans le futur. C'est-à-dire qu'un utilisateur pouvait seulement participer un tournoi. Et c'est un problème dans le sens que l'on ne veut pas inscrire les mêmes joueurs chaque année. C'est pour cela que j'ai supprimé cette table de mon schéma.

#### 5.2.1 Schéma actuel



**Schéma relationnel actuel**

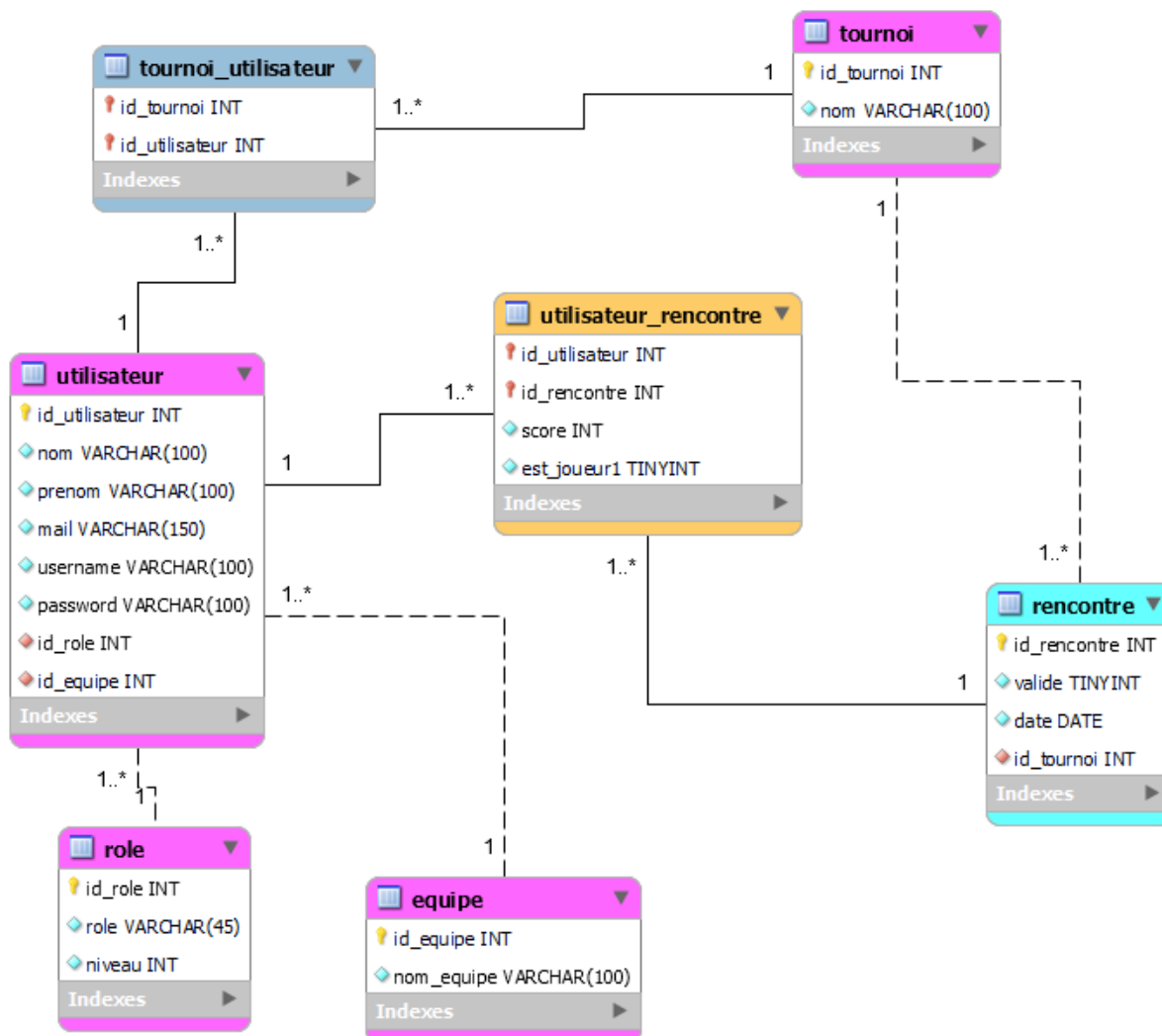
Voilà le schéma utilisé actuellement dans la base de données. On peut voir que la table tournoi a disparu. Il n'y a donc pas de gestion de tournoi dans le projet.

Un nouveau champ dans la table rôle est également apparu. Il permet de gérer le niveau du rôle.



## 5.2.2 Schéma futur

Dans un futur où on aurait besoin de gérer les tournois, j'ai également réalisé ce schéma qui permet de voir à ce que peut ressembler la base de données :



**Schéma relationnel futur**

Il y a dans ce schéma une table de relation entre utilisateur et tournoi. Cela permet de définir plusieurs tournois à un utilisateur. Une relation entre rencontre et tournoi a aussi été ajoutée. Cela permet de définir à quel tournoi appartient les rencontres vu qu'un utilisateur peut avoir plusieurs tournois.



## 5.3 Réalisation du Serveur

### 5.3.1 Requête

Le serveur contient de multiples requêtes qui ont toutes un point commun, elles reçoivent toutes au minimum un paramètre et retournent toute la même chose.

Le paramètre commun est `HttpServletRequest`, elle permet de récupérer et traiter les données envoyées par les clients, ainsi que pour interagir avec les différentes parties d'une requête HTTP, comme les sessions, les attributs et les redirections. Ce paramètre permettra de gérer la session dans les diverses requêtes.

L'objet retourné est `ResultJSON` :

```
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class ResultJSON {

    private int responseCode;
    private String responseTitle;
    private String responseText;
    private Object responseObject;

    public ResultJSON(int responseCode, String responseTitle, String responseText)
    {
        this.responseCode = responseCode;
        this.responseTitle = responseTitle;
        this.responseText = responseText;
    }

}
```

Il a 4 attributs :

- `responseCode`: les codes de retour de la requête (ex : 201)
- `responseTitle` : le titre du retour (ex : Utilisateur créé !)
- `responseText` : le texte du retour (ex : L'utilisateur a bien été créé !)
- `responseObject` : l'objet du retour (ex : Pour la création d'utilisateur on veut récupérer les informations de l'utilisateur en retour.)

`ResultJSON` a également des Getter, Setter et des constructeurs. Le plugin Lombok permet de générer ces codes automatiquement grâce à ces annotations (`@Getter`, `@Setter`,...).

### 5.3.1.1 Liste des requêtes

Voici la liste des requêtes disponible sur l'api

Méthode	Nom	Paramètre	Niveau de droit
GET	/api/user/getAllUsers	-	5
GET	/api/rencontre/getRencontres	-	5
GET	/api/rencontre/getClassement	-	5
GET	/api/utills/getAllRoles	-	5
GET	/api/utills/getAllEquipes	-	5
POST	/api/auth/login	username, password	0
POST	/api/auth/logout	-	0
POST	/api/user/saveUser	nom, prenom, mail, role, nomEquipe	10
POST	/api/rencontre/saveRencontre	joueur2, score1, score2, date	5
PUT	/api/user/putUser	id_user, nom, prenom, mail, role, nomEquipe	10
PUT	/api/rencontre/putValideRencontre	id	5
PUT	/api/rencontre/putRencontre	rencontreId, score1, score2, date	5
DELETE	/api/user/deleteUserById	id	10
DELETE	/api/rencontre/deleteRencontreById	Id	5

### 5.3.1.2 Code de retour

Voici les différents codes de retour :

200 OK : La requête a été traitée avec succès.

201 Created : Une nouvelle ressource a été créée avec succès.

400 Bad Request : La requête est incorrecte ou mal formée.

401 Unauthorized : L'accès à la ressource est refusé en raison d'informations d'identification invalides ou absentes.

403 Forbidden : L'accès à la ressource est refusé pour des raisons de permissions insuffisantes.

### 5.3.1.3 Niveau de droit

Actuellement il y a 2 rôles sur l'application : JOUEUR et ADMIN.

Le rôle JOUEUR a un niveau de 5, tandis que ADMIN a un niveau de 10.

Le niveau est en prévision au jour où on aimerait rajouter un rôle GESTIONNAIRE par exemple. Il aurait un niveau de 7 et pourrait juste gérer les rencontres par exemple.

## 5.3.2 Gestion de session

Lors de la réalisation du serveur, j'ai dû réaliser des sessions afin de permettre à un utilisateur communiquant avec l'utilisateur s'il est connecté. Le manque de connaissance et de préparation sur les sessions ont fait que j'ai eu de la peine à faire fonctionner correctement les sessions.

### 5.3.2.1 Problème

J'étais parti pour utiliser spring security qui est un plugin de spring boot. Ce plugin offre une protection et une gestion des accès aux applications web, en gérant l'authentification, l'autorisation et la gestion des sessions utilisateur. Le problème était que je n'avais jamais utilisé ce plugin et je n'ai fait aucun test technologique sur le sujet. Je n'ai finalement jamais réussi à faire fonctionner spring security et ne voulant pas perdre plus de temps, j'ai mis de côté les sessions.

### 5.3.2.2 Solution

Après quelques recherches j'ai fini par mettre de côté spring security car beaucoup trop complexe. J'ai utilisé l'objet HttpServletRequest qui permet de manipuler la requête qu'on reçoit afin de lui ajouter des attributs dans la session.

```
request.getSession().setAttribute("user", userLogged.getResponseObject());
```

Cette ligne de code permet d'ajouter un utilisateur dans l'attribut « user ». Je peux désormais accéder à cet attribut dans chaque requête jusqu'à la réinitialisation du serveur.

Chaque requête a donc en paramètre « HttpServletRequest request ». Il suffit ensuite de récupérer l'utilisateur dans la requête :



```
UtilisateurDTO sessionUser = (UtilisateurDTO) request.getSession().getAttribute("user");
```

Si « sessionUser » est nul ça veut dire qu'aucun utilisateur n'est connecté. Le serveur retournera une erreur 401. Si un utilisateur est connecté, le serveur testera ensuite s'il a le droit d'exécuter la requête. S'il n'as pas le droit, le serveur retournera une erreur 403.

### 5.3.2.3 UtilisateurDTO

UtilisateurDTO est la classe qui correspond à l'utilisateur qu'on stock dans la session.

```
@NoArgsConstructor
@Data
public class UtilisateurDTO {
    private Integer id;
    private String nom;
    private String prenom;
    private String mail;
    private String username;
    private String role;
    private int niveau;
    private String nomEquipe;
}
```

- Id : Correspond à l'identifiant de l'utilisateur
- Nom : Nom de l'utilisateur.
- Prénom : prénom de l'utilisateur
- Mail : mail de l'utilisateur .
- Username : nom d'utilisatuer de l'utilisateur
- Role : Nom du rôle (ex : ADMIN)
- Niveau : Correspond au niveau de droit du rôle (exemple : le rôle ADMIN est au niveau de 10 tandis que le rôle de JOUEUR est de 5)
- NomEquipe : Nom de l'équipe de l'utilisateur.

### 5.3.3 Création d'utilisateur

La création d'utilisateur commence dans le Controller ou l'on récupère la requête. :

```
@PostMapping(path = "/saveUser", consumes = MediaType.APPLICATION_JSON_VALUE, produces = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<ResultJSON> postUtilisateur(@RequestBody PostUserDTO user,
```



```
HttpServletRequest request) {
    UtilisateurDTO sessionUser = (UtilisateurDTO) request.getSession().getAttribute("user");
    if(sessionUser != null && sessionUser.getNiveau() >= 10){
        ResultJSON res =userService.saveUser(user);
        if(res.getResponseCode() == 200){
            return new ResponseEntity<>(res, new HttpHeaders(), HttpStatus.CREATED);
        }
        return new ResponseEntity<>(res, new HttpHeaders(), HttpStatus.BAD_REQUEST);
    }
    return new ResponseEntity<>(new ResultJSON(401, "Unauthorized", "Vous n'avez pas les droits pour accéder à ces ressources", null), new HttpHeaders(), HttpStatus.UNAUTHORIZED);
}
```

Le Controller reçoit en paramètre PostUserDTO et HttpServletRequest, PostUserDTO correspond à l'information de l'utilisateur que l'on veut recevoir.

Après avoir testé si l'utilisateur de la session est connecté ainsi qu'il ait les droits, on va appeler l'interface qui gère les utilisateurs.

Le service Utilisateur implémente toutes les classes présentes dans l'interface, dont saveUser qui permet de sauvegarder un utilisateur.

```
@Override
public ResultJSON saveUser(PostUserDTO u) {
    Utilisateur saveUser = new Utilisateur();
    saveUser.setNom(u.getNom());
    saveUser.setPrenom(u.getPrenom());
    saveUser.setMail(u.getMail());

    PasswordGenerator passwordGenerator = new PasswordGenerator.PasswordGeneratorBuilder().useDigits(true).useUpper(true).useLower(true).usePunctuation(true).build();
    String password = passwordGenerator.generate(10);

    LocalDateTime myDateObj = LocalDateTime.now();
    DateTimeFormatter myFormatObj = DateTimeFormatter.ofPattern("ddMMyyyyHHmmss");
    String formattedDate = myDateObj.format(myFormatObj);
    String username = u.getNom() + u.getPrenom().toUpperCase().charAt(0)+formattedDate;
    saveUser.setUsername(username);
    saveUser.setPassword(passwordEncoder.encode(password));
    saveUser.setIdRole(roleRepo.getReferenceById(u.getRole()));
    saveUser.setIdEquipe(equipeRepo.getReferenceById(u.getNomEquipe()));
    try {
        Utilisateur user = userRepo.save(saveUser);
        if (user != null){
```



```

        Map<String, Object> model = new HashMap<>();
        model.put("nom", user.getNom());
        model.put("prenom", user.getPrenom());
        model.put("username", user.getUsername());
        model.put("password", password);
        return mailSender.sendSimpleEmail(user.getMail(), "Inscription FutNet
Single Master", model);
    } else {
        return new ResultJSON(400, "User creation error", "L'utilisateur n'as
pas été crée ou une erreur c'est produite");
    }
} catch (Exception e){
    return new ResultJSON(400, "User error", "Les informations entrées ne sont
pas valide !");
}
}

```

On commence par créer un nouvel utilisateur et on lui attribue déjà son nom, prénom, et son mail.

On va ensuite générer un nouveau mot de passe pour pouvoir l'encrypter et l'attribuer à notre utilisateur. On va ensuite générer un nouveau nom d'utilisateur. Afin de le rendre unique, je récupère la date exacte jusqu'à la milliseconde. En théorie, le nom d'utilisateur ne sera jamais unique, car si on inscrit deux personnes ayant le même nom à la même milliseconde, ils auront le même nom d'utilisateur.

Après avoir récupéré la date précise, je récupère le nom de famille et la première lettre du prénom afin de créer son nom d'utilisateur. Par exemple (killian pasche => pascheK). J'y ajoute ensuite la date et j'attribue à notre utilisateur son nom d'utilisateur.

On va ensuite lui attribuer son rôle et son équipe. Pour ça, il faut récupérer leurs références via leur id dans leur repository.

Maintenant que toutes les informations d'un utilisateur ont été récupérées on peut essayer de le sauvegarder.

Si la sauvegarde s'est bien passée on appelle l'interface de mail pour lui donner les informations a envoyé.

```

@Override
public ResultJSON sendSimpleEmail(String mailTo, String subject, Map<String, Object>
model) {
    MimeMessage mimeMessage = mailSender.createMimeMessage();
    try {
        MimeMessageHelper helper = new MimeMessageHelper(mimeMessage, MimeMessage-
Helper.MULTIPART_MODE_MIXED_RELATED,
            "UTF-8");
        Template t = configuration.getTemplate("email-template.ftl");
        String html = FreeMarkerTemplateUtils.processTemplateIntoString(t, model);
    }
}

```



```

    helper.setTo(mailTo);
    helper.setText(html, true);
    helper.setSubject(subject);
    helper.setFrom("killian.pasche7@gmail.com");
    mailSender.send(mimeMessage);
    return new ResultJSON(200, "Envoie du mail", "Le mail mail à bien été envoyé à : "
    + mailTo);
} catch (MessagingException | IOException | TemplateException e) {
    return new ResultJSON(400, "mail error", e.getMessage());
}
}

```

Le service de mail va ensuite s'occuper d'envoyer un mail au destinataire.

### 5.3.3.1 PostUserDTO

PostUserDTO est l'objet que postUtilisateur reçoit.

```

@NoArgsConstructor
@AllArgsConstructor
@Data
public class PostUserDTO {
    private String nom;
    private String prenom;
    private String mail;
    private int role;
    private int nomEquipe;
}

```

PostUserDTO a 5 attributs :

- Nom : nom de l'utilisateur
- Prenom: prénom de l'utilisateur
- Mail: mail de l'utilisateur
- Role: id du rôle
- nomEquipe: id de l'équipe

### 5.3.3.2 Axe d'amélioration

Voilà comment on pourrait améliorer la création d'un utilisateur. On pourrait déjà trouver un moyen pour créer des noms d'utilisateur uniques qui sont plus courts, actuellement un nom d'utilisateur ressemble à ça :

PascheK26052023212304 -> (PascheK : nom+Première lettre du prénom, 26052023212304 : la date de création).



Le nom d'utilisateur est extrêmement long ce qui peut être dérangerant pour la connexion. L'utilisateur devra soit enregistrer ces informations soit aller récupérer son nom d'utilisateur dans le mail qu'il a reçu.

### 5.3.4 Ajout de rencontre

L'ajout de rencontre fonctionne comme l'ajout d'un utilisateur :

```
@PostMapping(path = "/saveRencontre", produces = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<ResultJSON> postRencontre(HttpServletRequest request, @RequestBody PostRencontreDTO r) {
    UtilisateurDTO sessionUser = (UtilisateurDTO) request.getSession().getAttribute("user");
    if(sessionUser != null && sessionUser.getNiveau() >= 5){
        ResultJSON res = rencontreService.postRencontre(sessionUser.getId(), r.getJoueur2(), r.getScore1(), r.getScore2(), r.getDate());
        if(res.getResponseCode() == 200){
            return new ResponseEntity<>(res, new HttpHeaders(), HttpStatus.OK);
        }else {
            return new ResponseEntity<>(res, new HttpHeaders(), HttpStatus.BAD_REQUEST);
        }
    }
    return new ResponseEntity<>(new ResultJSON(401, "Unauthorized", "Vous n'avez pas les droits pour acceder à ces ressources", null), new HttpHeaders(), HttpStatus.UNAUTHORIZED);
}
```

Les paramètres sont PostRencontreDTO qui correspondront aux informations de la rencontre qu'on veut créer et HttpServletRequest qui correspond à la requête.

Après avoir testé si l'utilisateur de la session est connecté et qu'il ait les droits, on va appeler l'interface qui gère les rencontres.

Le service rencontre implémente toutes les classes présentes dans l'interface, dont postRencontre qui permet de sauvegarder une rencontre.

```
@Override
public ResultJSON postRencontre(int joueur1, int joueur2, int score1, int score2, LocalDate date) {
    if(joueur1 == joueur2)
        return new ResultJSON(400, "Erreur rencontre", "Un joueur ne peut pas jouer contre lui même!");
    try {
        Utilisateur j1 = utilisateurRepo.getReferenceById(joueur1);
        Utilisateur j2 = utilisateurRepo.getReferenceById(joueur2);
        if (getCommunRencontreNumber(joueur1, joueur2) <= 9) {
            if(score1 == 15 && score2 == 15)
                return new ResultJSON(400, "Erreur rencontre", "Les deux joueurs ont déjà joué 15 points");
            rencontreService.postRencontre(j1.getId(), j2.getId(), score1, score2, date);
            return new ResultJSON(200, "Succès", "La rencontre a été créée avec succès");
        }
    } catch (Exception e) {
        return new ResultJSON(500, "Erreur serveur", "Une erreur s'est produite lors de la création de la rencontre");
    }
}
```





```

        return new ResultJSON(400, "Score error", "Un des
deux scores entrée n'est pas valide");
        if (score1 >= 0 && score1 <= 15 && score2 >= 0 && score2
<= 15 && score1 == 15 || score2 == 15) {
            if (j1 != null && j2 != null) {
                Rencontre r = addRencontre(date);
                UtilisateurRencontreId urID1 = new Utilisa-
teurRencontreId(r.getId(), j1.getId());

                UtilisateurRencontre url = new UtilisateurRen-
contre(urID1, j1, r, score1, Byte.valueOf("1"));
                utilisateurRencontreRepo.save(url);
                UtilisateurRencontreId urID2 = new Utilisa-
teurRencontreId(r.getId(), j2.getId());

                UtilisateurRencontre ur2 = new UtilisateurRen-
contre(urID2, j2, r, score2, Byte.valueOf("0"));
                utilisateurRencontreRepo.save(ur2);
                return new ResultJSON(200, "Succes", "La ren-
contre à été enregistrée");
            } else {
                return new ResultJSON(400, "User error", "Un des
utilisateurs entrée n'est pas valide");
            }
        } else {
            return new ResultJSON(400, "Score error", "Un des
deux scores entrée n'est pas valide");
        }
    } else {
        return new ResultJSON(400, "Rencontre error", "le chiffre
maximum de rencontre contre un même utilisateur à été atteinds");
    }
} catch (Exception e) {
    return new ResultJSON(400, "Erreur rencontre", "Une erreur à
eu lieux lors de l'ajout de la rencontre !");
}
}

```

La méthode postRencontre a plusieurs paramètres.

- Joueur1 : id du joueur 1.
- Joueur2 : id du joueur 2.
- Score1 : score du joueur 1.
- Score2 : score du joueur 2.
- Date : date de la rencontre.

On commence par tester si joueur1 est égale à joueur2. Si c'est le cas, on retourne une erreur car on ne veut pas qu'un joueur puisse jouer contre lui-même.

On va ensuite récupérer la référence des utilisateurs correspondant au joueur 1 et au joueur 2.

On va ensuite regarder le nombre de rencontres qu'ils ont en commun. S'ils ont 9 rencontres en commun, ils ne peuvent plus en ajouter. On va ensuite tester les scores car il faut qu'un des deux scores soit égale à 15. Si un des deux scores est plus grand que 15 ou plus petit que 0, on retourne une erreur.

Si tous les tests ont été passés, on va créer une rencontre en lui donnant la date de la rencontre.

Une fois que la rencontre a été créée on va créer les relations entre les deux joueurs et la rencontre.

#### 5.3.4.1 getCommunRencontreNumber

Cette méthode est la méthode qui permet de récupérer le nombre de fois que des joueurs ont joué ensemble.

Voilà la requête réalisée pour récupérer :

```
SELECT COUNT(*) FROM utilisateur_rencontre AS c1 INNER JOIN utilisateur_rencontre as c2 ON c1.id_rencontre = c2.id_rencontre AND c1.id_utilisateur != c2.id_utilisateur WHERE c1.id_utilisateur=:u1 AND c2.id_utilisateur=:u2
```

Cette requête SQL permet de compter le nombre de rencontres entre deux utilisateurs spécifiques (:u1 et :u2) dans une table appelée "utilisateur\_rencontre". Voici comment la requête est structurée :

1. La clause "FROM" spécifie la table "utilisateur\_rencontre" et la renomme en "c1" (alias).
2. La clause "INNER JOIN" relie la table "utilisateur\_rencontre" à elle-même en utilisant deux alias différents : "c1" et "c2". Cette jointure est effectuée en comparant les colonnes "id\_rencontre" de "c1" et "c2" pour trouver les rencontres communes entre les deux utilisateurs.
3. La clause "ON" définit les conditions de jointure. Ici, la condition est que les "id\_rencontre" des deux alias doivent être égaux, et en même temps, les "id\_utilisateur" des deux alias doivent être différents.
4. La clause "WHERE" spécifie des critères supplémentaires à appliquer à la requête. Dans ce cas, la condition est que l'"id\_utilisateur" de "c1" doit correspondre à la valeur de ":u1" (premier utilisateur spécifié) et que l'"id\_utilisateur" de "c2" doit correspondre à la valeur de ":u2" (deuxième utilisateur spécifié).
5. Enfin, la clause "SELECT COUNT(\*)" permet de compter le nombre de lignes résultantes de la requête. Dans ce cas, cela correspond au nombre de rencontres entre les deux utilisateurs spécifiés.

En résumé, cette requête SQL permet de déterminer le nombre de rencontres entre deux utilisateurs spécifiques dans la table "utilisateur\_rencontre" en utilisant une jointure interne (INNER JOIN) et des conditions de filtrage pour exclure les rencontres avec le même utilisateur.

#### **5.3.4.2 Axe d'amélioration**

Pour améliorer l'ajout de rencontre on pourrait utiliser les transactions en sql. Vue qu'on manipule beaucoup la base de données ça serait bien qu'on puisse annuler les requêtes si une requête a échoué. N'ayant eu pas le temps d'ajouter cette fonctionnalité, il y a le risque qu'une requête échoue.

#### **5.3.5 Diagramme de classe**

Voilà le diagramme de classe généré par IntelliJ IDEA



Diagramme de classes serveur

Ce schéma représente toutes les classes présentes dans l'api.

## 5.4 Réalisation du client

### 5.4.1 Session

Lors que de l'implémentation de la session sur le client, j'ai rencontré un problème. Le problème était que le client tournait sur le port 8081 et le serveur 8080. C'est-à-dire qu'à chaque



fois que j'appelais le serveur il créait une nouvelle session. Donc impossible de rester connecté vu qu'à chaque appel il oubliait ce qu'il avait stocké dans sa session. Pour remédier à ce problème, j'ai créé un proxy afin que la requête qui soit faite soit redirigée sur le serveur.

```
vite.config.js :
server: {
  port: 8081,
  proxy: {
    '/api': {
      target: 'http://localhost:8080',
      secure: false
    }
  }
},
},
```

- port : 8081: spécifie le port sur lequel le serveur sera exécuté. Dans ce cas, le serveur sera accessible à l'adresse <http://localhost:8081>.
- proxy : déclare un proxy pour rediriger certaines requêtes du serveur vers un autre serveur. Un proxy est utilisé pour contourner les restrictions de la politique de même origine (Same Origin Policy) lorsqu'une application doit accéder à des ressources provenant d'un autre domaine.
- '/api' : spécifie le préfixe des URL qui seront redirigées vers le serveur cible. Par exemple, toutes les requêtes commençant par /api seront redirigées vers le serveur <http://localhost:8080>.
- target : 'http://localhost:8080': spécifie l'URL du serveur cible vers lequel les requêtes seront redirigées. Dans ce cas, toutes les requêtes qui correspondent au préfixe /api seront envoyées au serveur local exécuté à <http://localhost:8080>.
- secure : false: indique si la connexion avec le serveur cible doit être sécurisée. Lorsque cette valeur est définie sur false, la connexion peut être effectuée via HTTP plutôt que via HTTPS.

En résumé, ce code configure un serveur de développement pour écouter sur le port 8081 et redirige les requêtes commençant par /api vers un autre serveur exécuté localement à l'adresse <http://localhost:8080>.

Une fois ce problème réglé, j'ai utilisé piniaJs afin de pouvoir stocker l'utilisateur connecté, localement et pouvoir y accéder dans toute l'application.

```
stores/userSessionStore.ts :
export const useUserSessionStore = defineStore('user', () => {
  const user = ref({} as User)
  const connected = ref(false)
  const isAdmin = computed(() => {
    try {
      return connected.value && user.value.role === 'ADMIN'
    } catch {
      return false
    }
  })
})
```



```

    }
  })
  const idUser = computed(() => {
    try {
      return user.value.id
    } catch {
      return false
    }
  })
  const router = useRouter()
  const route = useRoute()
  function init() {
    try {
      let sessionString = localStorage.getItem('userStore')

      if (sessionString === null) throw new Error()

      let session = JSON.parse(sessionString)
      if (session.connected == undefined || session.user == undefined || session.connected == false)
        throw new Error()

      let sessionUser = session.user as User

      if (
        sessionUser.id == undefined ||
        sessionUser.nom == undefined ||
        sessionUser.prenom == undefined ||
        sessionUser.role == undefined ||
        sessionUser.nomEquipe == undefined ||
        sessionUser.mail == undefined ||
        sessionUser.username == undefined
      )
        throw new Error()

      user.value = new User(
        sessionUser.id,
        sessionUser.nom,
        sessionUser.prenom,
        sessionUser.role,
        sessionUser.nomEquipe,
        sessionUser.mail,
        sessionUser.username
      )
      connected.value = true
      if (route.fullPath === 'login') router.push('/')
    } catch {
      destroy()
    }
  }

```



```

    }
  }
  function save() {
    localStorage.setItem(
      'userStore',
      JSON.stringify({ user: user.value, connected: connected.value })
    )
    router.push('/')
  }
  function destroy() {
    connected.value = false
    user.value = {} as User
    localStorage.removeItem('userStore')
    errorSwal('Veuillez vous connecter !').fire()
    router.push('/login')
  }
  function login(username: string, password: string) {
    axios({
      method: 'POST',
      url: 'auth/login',
      data: {
        username: username,
        password: password
      }
    }).then(function (response: any) {
      if (response.status == 200) {
        if (response.data.responseObject != null) {
          let loggedUser = response.data.responseObject
          connected.value = true
          user.value = new User(
            loggedUser.id,
            loggedUser.nom,
            loggedUser.prenom,
            loggedUser.role,
            loggedUser.nomEquipe,
            loggedUser.mail,
            loggedUser.username
          )
          save()
        }
      }
    })
  }
  function disconnect() {
    axios({
      method: 'POST',
      url: 'auth/logout',
      data: {}
    })
  }

```





```

    }).then(function (response: any) {
      if (response.status === 200) {
        destroy()
      }
    })
  }
  return { user, connected, save, login, init, destroy, disconnect, isAdmin, idUser }
})

```

`const user = ref({} as User)` déclare une variable réactive `user` initialisée avec un objet vide du type `User`.

`const connected = ref(false)` déclare une variable réactive `connected` initialisée à `false`, indiquant si l'utilisateur est connecté ou non.

`const isAdmin = computed(() => {...})` déclare une propriété calculée `isAdmin` qui renvoie `true` si l'utilisateur est connecté et a le rôle "ADMIN", sinon renvoie `false`.

`const idUser = computed(() => {...})` déclare une propriété calculée `idUser` qui renvoie l'identifiant de l'utilisateur, ou `false` s'il n'est pas défini.

`router = useRouter()` et `route = useRoute()` permettent d'accéder à l'instance du routeur `Vue.js` et à l'objet de route actuel.

La fonction `init()` est appelée au démarrage de l'application. Elle récupère les données de l'utilisateur depuis le stockage local (`localStorage`) et les assigne à `user` et `connected`. Si les données sont invalides, la fonction `destroy()` est appelée.

La fonction `save()` enregistre les données de l'utilisateur dans le stockage local et redirige l'utilisateur vers la page principale.

La fonction `destroy()` supprime les données de l'utilisateur du stockage local, réinitialise `user` et `connected`, affiche une alerte d'erreur, et redirige l'utilisateur vers la page de connexion.

La fonction `login(username, password)` envoie une requête `POST` à l'URL `auth/login` avec les identifiants de connexion de l'utilisateur. Si la réponse est réussie (statut 200) et contient des données valides, les informations de l'utilisateur sont extraites et assignées à `user` et `connected`. Ensuite, la fonction `save()` est appelée.

La fonction `disconnect()` envoie une requête `POST` à l'URL `auth/logout` pour déconnecter l'utilisateur. Si la réponse est réussie, la fonction `destroy()` est appelée.

Finalement, la fonction `useUserStore` retourne un objet contenant les variables, les fonctions et les propriétés calculées qui seront utilisées par les composants de l'application.

### 5.4.2 Récupération du classement

Pour afficher le classement dans un tableau, j'utilise la librairie `Vue3EasyDatatable`. La librairie permet de stocker facilement des données dans un tableau.

```

<EasyDataTable
  alternating
  buttons-pagination

```





```
:headers="headers"
:search-value="searchValue"
:items="items"
:rows-per-page="10"
:sort-by="sortBy"
:sort-type="sortType"
/>
```

- alternating : Permet de faire un tableau rayé.
- button-pagination : Ajoute les boutons en fonction du nombre de page du tableau.
- :headers : correspond à l'entête du tableau
- :search-value : la valeur qu'il faut rechercher dans le tableau.
- :items : les valeurs du tableau.
- :rows-per-page : le nombre de ligne par page.
- :sort-by : l'élément qui définit par quoi le tableau est trié
- :sort-type : le sens du tri du tableau.

```
- const headers: Header[] = [
-   { text: 'ID', value: 'userID', sortable: true },
-   { text: 'NOM', value: 'nom', sortable: true },
-   { text: 'PRENOM', value: 'prenom', sortable: true },
-   { text: 'EQUIPE', value: 'nomEquipe', sortable: true },
-   { text: 'SCORE', value: 'score', sortable: true }
- ]
```

Correspond au différent entête et s'ils peuvent être triés

```
onMounted(async () => {
  await classementStore.fetchClassement()
  items.value = classementStore.classement
})
```

Permet de récupérer la liste du classement et l'attribuer au tableau.

## 5.5 Configuration client

### 5.5.1 Axios

Axios.js est librairie utilisée pour effectuer des requêtes HTTP depuis un navigateur. Elle simplifie l'interaction avec les API en fournissant une interface simple et expressive pour envoyer des requêtes et traiter les réponses. Axios.js prend en charge les fonctionnalités telles que la



gestion des erreurs, les interceptions de requêtes et les transformations de données, ce qui en fait un choix courant pour les applications web modernes.

Configuration :

```
import Axios from 'axios'
export const API_BASE_URL = 'http://localhost:8081/api/'
export const axios = Axios.create({
  baseURL: API_BASE_URL
})
```

### 5.5.2 Vue Router

Vue Router est une librairie pour la navigation dans les applications Vue.js. Elle permet de gérer les routes et les vues de manière efficace, facilitant ainsi le développement des applications à page unique (SPA) et des applications web dynamiques.

Configuration :

```
import { createRouter, createWebHistory } from 'vue-router'

import Login from '@views/Login.vue'
import Home from '@views/Home.vue'
import Classement from '@views/Classement.vue'
import Rencontre from '@views/Rencontre.vue'
import Inscription from '@views/Inscription.vue'

const router = createRouter({
  history: createWebHistory(),
  routes: [
    {
      path: '',
      component: Home,
      children: [
        { path: '', component: Classement },
        { path: '/rencontre', component: Rencontre },
        { path: 'Inscription', component: Inscription }
      ]
    },
    { path: '/login', component: Login }
  ]
})

export default router
```

### 5.5.3 TypeScript

TypeScript est un langage de programmation qui est un sur-ensemble de JavaScript. Il ajoute des fonctionnalités de typage statique optionnel, permettant ainsi de détecter des erreurs de programmation avant l'exécution du code. TypeScript améliore la maintenabilité, la lisibilité et la productivité des projets JavaScript, en offrant un système de types plus robuste et en facilitant le développement d'applications complexes.

Aucune configuration nécessaire.

### 5.5.4 PiniaJs

PiniaJS est une Librairie de gestion d'état d'application. Elle fournit une solution légère et performante pour la gestion de l'état centralisé dans les applications Vue.js. PiniaJS est conçu pour être simple et intuitive, offrant des fonctionnalités telles que les actions, les mutations et les getters pour faciliter la gestion de l'état de l'application. Elle se distingue par sa compatibilité avec les fonctionnalités avancées de Vue.js telles que les plugins, les observateurs réactifs et les composants asynchrones.

Aucune configuration nécessaire.

### 5.5.5 Vue3EasyDataTable

Vue3EasyDataTable est une librairie qui simplifie la création de tableaux de données interactifs dans les applications Vue.js. Elle offre des fonctionnalités telles que le tri, la pagination, la recherche et la personnalisation des données tabulaires. Vue3EasyDataTable est conçue pour être facile à utiliser et configurable, permettant aux développeurs de gérer et d'afficher efficacement de grandes quantités de données dans des tableaux conviviaux et esthétiques.

Aucune configuration nécessaire.

### 5.5.6 VueSweetalert2

VueSweetalert2 est une Librairie qui permet d'intégrer facilement les pop-ups et les alertes élégantes et personnalisables de la bibliothèque SweetAlert2 dans les applications Vue.js.

Aucune configuration nécessaire.

## 6 Test

### 6.1 Procédure de test

Le protocole de test permet de trouver des problème dans l'application. Pour tester mon application, j'ai utilisé postman.


### 6.2 Protocol de test

Nr.	Objet Testé	Description du test	Date du test	Attente	Résultat	Visa
1	<b>Session</b>	Test de la session, regarde si l'utilisateur n'est pas connecté.	02.06.2023	Une erreur 401 si l'utilisateur n'est pas connecté	Retourne une error 401 si l'utilisateur n'est pas connecté.	K.P
2	<b>Droit de la requête</b>	Test si l'utilisateur connecté a le droit d'exécuter une requête	02.06.2023	Une erreur 403 s'il n'a pas le droit sinon un code 200	Retourne une erreur 403 s'il n'as pas accès. Sinon le code 200 est retourné	K.P
3	<b>Connexion</b>	Test si un utilisateur peut se connecter	02.06.2023	Un code 200 si ces informations sont justes. Sinon une erreur 401	Retourne un code 200 si la connexion a réussi. Si les credential sont faux retourne une erreur 401	K.P
4	<b>Récupération de la liste des utilisateurs</b>	Test la récupération des utilisateurs	02.06.2023	Un code 200 et la liste des utilisateurs.	Retourne un code 200 et la liste des utilisateurs	K.P
5	<b>Ajout d'utilisateur</b>	Test l'ajout d'un utilisateur	02.06.2023	Un code 201 si l'utilisateur a été ajouté	Retourne un code 201 quand l'utilisateur a été ajouté.	K.P
6	<b>Modification d'un utilisateur</b>	Test la modification d'un utilisateur	02.06.2023	Un code 200 si l'utilisateur a été modifié et l'utilisateur. Une erreur 400 si l'utilisateur n'existe pas	Retourne un code 200 avec l'utilisateur. Retourne une erreur 400 lorsque l'utilisateur n'existe pas	K.P
7	<b>Suppression d'un utilisateur</b>	Test si la suppression d'un utilisateur	02.06.2023	Un code 200 si l'utilisateur a été supprimer. Sinon une erreur 400 l'utilisateur n'existe pas.	Retourne un code 200. Retourne une erreur 400 quand l'utilisateur n'existe pas	K.P



8	<b>Ajout de rencontre</b>	Test si l'ajout de rencontre	02.06.2023	Un code 200 si la rencontre a été ajoutée. Une erreur 400 si les scores sont invalides, si les deux utilisateurs sont les même, si les nombre de rencontre est supérieur à 9.	Retourne un code 200. Retourne une erreur 400 si les données sont invalides.	<b>K.P</b>
9	<b>Modification d'une rencontre</b>	Test si la modification a été effectuée	02.06.2023	Un code 200 si la rencontre est modifiée. Une erreur 400 si les données sont invalides.	Retourne un code 200. Retourne une erreur 400 si les données sont invalides.	<b>K.P</b>
10	<b>Validation d'une rencontre</b>	Test la validation d'une rencontre	02.06.2023	Un code 200 si la rencontre est validée. Une erreur 400 si la rencontre a déjà été validée ou si les données sont invalides.	Retourne un code 200. Retourne une erreur 400 si la rencontre a été validé ou bien si les données sont invalides.	<b>K.P</b>
11	<b>Suppression d'une rencontre</b>	Test la suppression d'une rencontre	02.06.2023	Un code 200 si la rencontre a été supprimée, sinon une erreur 400	Retourne un code 200. Sinon retourne une erreur 400	<b>K.P</b>
12	<b>Récupération d'une rencontre</b>	Test la récupération des rencontres	02.06.2023	Un code 200 et la liste des rencontres.	Retourne la liste des rencontres et un code 200	<b>K.P</b>
13	<b>Récupération des classements</b>	Test la récupération du classement	02.06.2023	Un code 200 et le classement	Retourne le classement le un code 200	<b>K.P</b>

### 6.3 Signature du protocole de test

Date	Nom	Signature
02.06.2023	Pasche Killian	

## 7 Conclusion

En conclusion, mon projet a été couronné de succès, atteignant tous les objectifs fixés. J'ai consacré du temps à l'analyse, à la conceptualisation et à la réalisation, suivis de tests approfondis pour assurer sa fonctionnalité. Je suis extrêmement satisfait des résultats obtenus lors de ce projet TPI. En effet, je n'avais jamais travaillé avec SpringBoot ni VueJS auparavant, et cette expérience m'a permis de découvrir de nouvelles technologies qui seront précieuses dans ma future carrière professionnelle. Non seulement je suis fier de mon accomplissement, mais j'ai également pris un immense plaisir à mener à bien ce projet, étant donné l'intérêt que j'ai porté à toutes les thématiques abordées.

### 7.1 Améliorations possibles

- **Ajout de filtres supplémentaires** : J'aimerais ajouter des filtres supplémentaires aux tableaux, tels qu'un filtre permettant de trier par nombre de défaites ou de victoires dans le tableau de classement, par exemple. Les filtres sont essentiels dans une application où nous avons besoin de visualiser de telles données.
- **Ajout de graphiques** : Ajouter des graphiques pour visualiser les performances d'un joueur ou d'une équipe serait un ajout intéressant. Un support visuel supplémentaire permettrait d'obtenir rapidement des données précises.
- **Gestion d'équipe** : La gestion d'équipe serait un ajout majeur pour le projet. Permettre aux joueurs de consulter des informations telles que le lieu d'entraînement, les coordonnées de l'équipe ou la possibilité de la rejoindre serait appréciable. Les administrateurs pourraient également gérer ces équipes.
- **Profile utilisateur** : L'implémentation des profils utilisateurs serait également un avantage. En cliquant sur le profil d'un joueur, on pourrait découvrir des détails le concernant. Cela faciliterait les recherches sur un joueur et permettrait d'économiser de l'espace dans le tableau.
- **Application PWA** : Transformer l'application web en PWA serait bénéfique, car cela éviterait de devoir ouvrir le navigateur à chaque fois. Avoir l'application sur son téléphone serait un gain de temps considérable, surtout pour consulter les scores.

J'ai encore plein d'idées, une application n'est jamais terminée, on peut toujours l'améliorer...

### 7.2 Auto-évaluation

J'ai le sentiment d'avoir réalisé un travail satisfaisant pendant le TPI. J'ai réussi à atteindre tous les objectifs dans les délais impartis. Toutefois, je reconnais que certaines améliorations auraient pu être apportées, notamment en termes de documentation et d'efficacité dans l'exécution des tâches. Le niveau sonore ambiant dans la salle de classe a parfois entravé ma concentration, et un environnement plus calme aurait été bénéfique. Malgré ces défis, je suis fier



du projet solide que j'ai pu réaliser, d'autant plus que je n'avais aucune connaissance préalable des technologies utilisées avant de me lancer dans ce projet. Je pense que j'ai réalisé un bon travail.

## 8 Bibliographie: liste des sources et références

Axios: <https://axios-http.com/docs/intro>

SweatAlert2: <https://sweetalert2.github.io/>

Vue3 EasyDatatable : <https://hc200ok.github.io/vue3-easy-data-table-doc/>

VueJS : <https://vuejs.org/>

ViteJS : <https://vitejs.dev/>

PiniaJS : <https://pinia.vuejs.org/>

Vue Router: <https://router.vuejs.org/>

TypeScript : <https://www.typescriptlang.org/>



## 9 Glossaire

Terme	Signification
Asynchrones	Le terme "asynchrones" se réfère à des événements ou des opérations qui se produisent de manière indépendante et sans suivre un ordre strict ou synchronisé.
Commit	Un commit est une action dans le système de contrôle de version Git où les modifications apportées à un ensemble de fichiers sont enregistrées de manière permanente.
DELETE	DELETE est une méthode utilisée dans le protocole HTTP pour demander la suppression d'une ressource spécifique sur un serveur.
Dépôts	Un dépôt (JPA) fait référence à une classe ou une interface utilisée dans le cadre de Java Persistence API (JPA) pour effectuer des opérations de persistance sur des objets dans une base de données.
Framework	Un framework est un ensemble de bibliothèques, d'outils et de conventions de développement qui fournissent une structure et des fonctionnalités prédéfinies pour faciliter le processus de développement de logiciels.
GET	GET est une méthode utilisée dans le protocole HTTP pour récupérer des ressources à partir d'un serveur.
Git	Git est un système de contrôle de version décentralisé, permet de suivre les modifications apportées à un ensemble de fichiers au fil du temps
HTTP	HTTP (Hypertext Transfer Protocol) est un protocole utilisé pour la communication entre un navigateur web et un serveur. Il permet d'envoyer des requêtes du navigateur au serveur et de recevoir des réponses contenant des données telles que des pages web, des images ou d'autres ressources.
HTTPS	HTTPS (Hypertext Transfer Protocol Secure) est une version sécurisée du protocole HTTP. Il utilise une couche de chiffrement SSL/TLS pour protéger les données échangées entre le navigateur et le serveur, assurant ainsi la confidentialité et l'intégrité des informations transmises. Cela est particulièrement important lors de la transmission de données sensibles, telles que les informations de connexion, les paiements en ligne ou toute autre donnée confidentielle.



Implémentation	La mise en œuvre pratique ou la réalisation concrète d'un plan, d'un projet ou d'une idée.
JavaScript /JS	JavaScript est un langage de programmation populaire utilisé principalement pour le développement web.
Maintenabilité	La maintenabilité désigne la capacité d'un système, d'un logiciel ou d'un produit à être facilement maintenu, réparé, mis à jour ou modifié.
Many-to-Many	Many-to-Many est une relation dans les bases de données où plusieurs enregistrements d'une table peuvent être associés à plusieurs enregistrements d'une autre table. C'est une relation de type "plusieurs à plusieurs".
Many-to-One	Many-to-One est une relation dans les bases de données où plusieurs enregistrements d'une table peuvent être associés à un seul et unique enregistrement d'une autre table. C'est une relation de type "plusieurs à un".
NodeJS	Node.js est un environnement d'exécution JavaScript côté serveur.
One-to-Many	One-to-Many est une relation dans les bases de données où un enregistrement d'une table peut être associé à plusieurs enregistrements d'une autre table, mais chaque enregistrement de la seconde table ne peut être associé qu'à un seul enregistrement de la première table. C'est une relation de type "un à plusieurs".
One-to-One	One-to-One est une relation dans les bases de données où un enregistrement d'une table est associé à un seul et unique enregistrement d'une autre table. C'est une relation de type "un à un".
Pagination	permet aux utilisateurs de naviguer facilement entre les différentes pages de résultats sans avoir à afficher toutes les données en une seule fois.
POST	POST est une méthode utilisée dans le protocole HTTP pour envoyer des données à un serveur afin de créer une nouvelle ressource.
PUT	PUT est une méthode utilisée dans le protocole HTTP pour envoyer des données à un serveur afin de mettre à jour une ressource existante ou de créer une nouvelle ressource à une URL spécifique.




Rollback	Un rollback est une opération qui consiste à revenir en arrière sur une modification effectuée dans un système, une application ou un projet.
SMTP	SMTP signifie "Simple Mail Transfer Protocol". Il s'agit d'un protocole utilisé pour l'envoi de courriers électroniques sur Internet.
SQL	SQL est l'acronyme de "Structured Query Language", qui signifie langage de requête structuré. C'est un langage informatique utilisé pour communiquer avec et gérer des bases de données relationnelles.



## 10 Signatures

Je soussigné déclare que les informations contenues dans ce rapport de travail pratique individuel rendu ce jour le 06.06.2023 dans le cadre de la procédure de qualification de mon CFC d'informaticien, ne sont pas plagiées. Toutes les informations de sources extérieures ainsi que les informations fournies par des tiers durant le déroulement du travail sont consignées.

Date	Nom	Signature
05.06.2023	Pasche	

# 11 Annexes

Voici les fichiers qui sont contenus dans le dossier « TPI\_2023\_FutNet\_Single\_Master\_Pasche\_Killian\_Annexe »

- 00 – Documents
  - 1\_PASCHE\_Schema
    - Contient les diagrammes / Schémas UML et autres...
  - 2\_PASCHE\_Planning
    - Planning du TPI
  - 3\_PASCHE\_Journal
    - Journal de travail
  - 4\_PASCHE\_rapport\_du\_TPI\_2023
    - Rapport du TPI
  - 5\_PASCHE\_documentation\_utilisateur
    - Documentation utilisateur
  - 6\_PASCHE\_BD\_DATA
    - Schéma de la base de données (Ancien / Actuel / Futur)
    - Export de la base de données SQL
  - 7\_PASCHE\_Maquette
    - Maquette du projet
  - 8\_PASCHE\_PV
    - PV des rencontres avec les experts ou avec le chef de projet
- 01 – Applications
  - futnetsinglemaster
    - Application cliente en VueJS
  - api
    - API REST en SpringBoot
- 03 – Logo
  - Logo de l'application