Documentation de l'API SOKKA :

Création d'une API intégrant un indicateur de performance

Etablissement : Ecole d'ingénieurs Sup Galilée
Spécialité : Informatique
Année scolaire : Troisième année - équivalent M2
Lieu : Université Sorbonne Paris Nord - 93430 Villetaneuse

Equipe :

— Ludovik TEKAM
— Michaël VALBON
— Amina BAHADDI
— Anas HAFSI

Année universitaire : 2019 - 2020

Table des matières

1 Présentation du projet

Dans le cadre de notre projet de fin d'études, nous avons réalisé pour le compte de **SOKKA** une API sous forme de service web qui leur permettrait de gérer leur base de données. Cette API devait aussi intégrer un indicateur de performance.

Ce document a été rédigé dans le but de décrire le processus de développement de l'API et de faciliter sa reprise par d'autres développeurs.

2 Conception, implémentation et borne

Cette partie a été la plus importante de tout le projet. En effet, la bonne compréhension du fonctionnement interne de l'API passe par la compréhension de cette partie.

Donc si vous souhaitez modifier la structure de la base de données, il faudra revoir cette partie et y ajouter, supprimer et/ou modifier ce qui vous semble important.

2.1 La conception de la base de données

2.1.1 Le diagramme de classe

Initialement, nous avions conceptualisé un diagramme de classe classique qui nous permettrait de sauvegarder nos données tout en respectant nos différentes contraintes

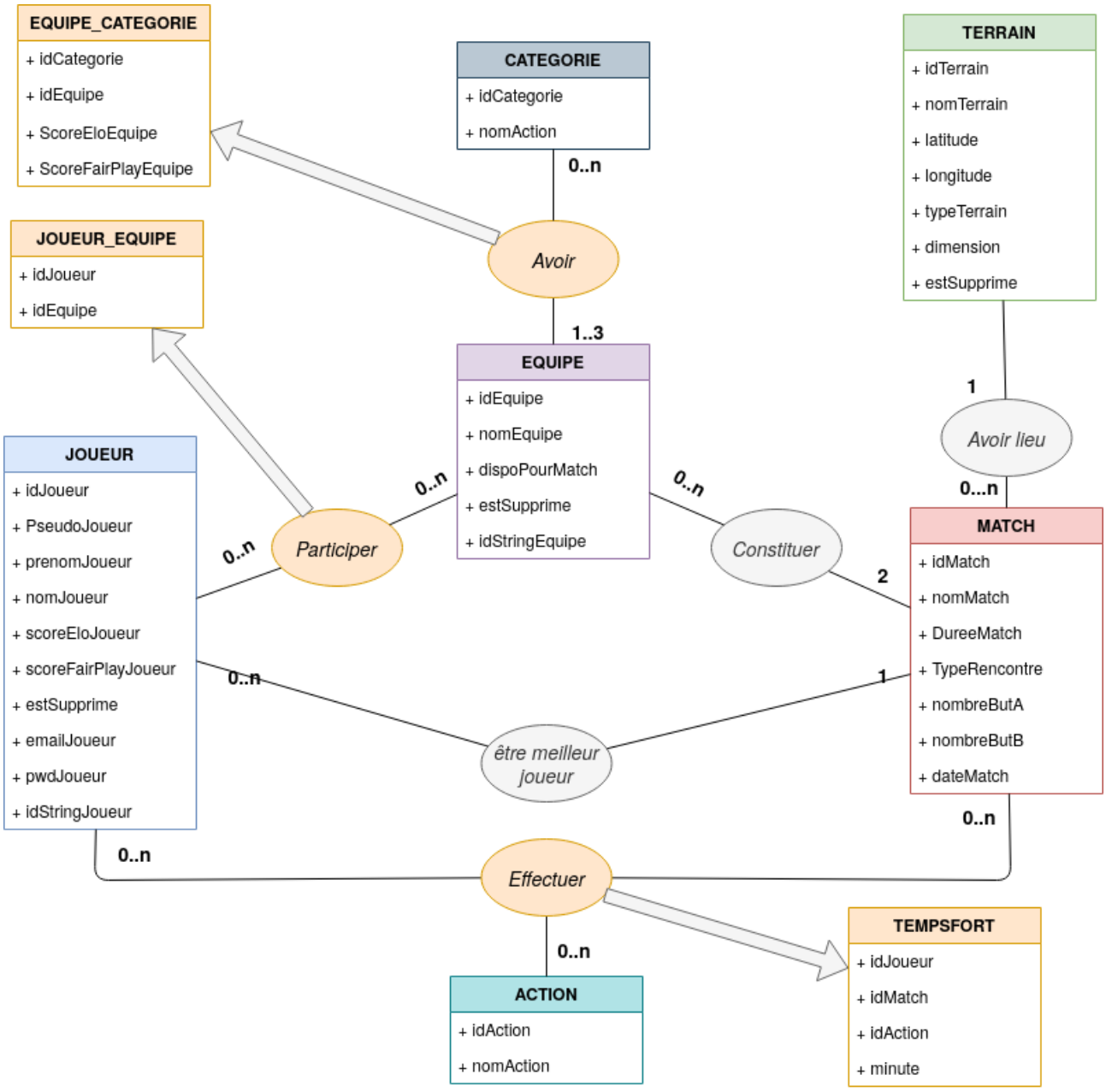

FIGURE 1 – Diagramme de classe

2.1.2 Le diagramme de classe amélioré

Suite de nos nombreux échanges avec le client, nous avons réalisé l'importance de garder un historique de toutes les modifications qui seront effectuées dans la base de données. Ce qui nous a conduits à modifier notre diagramme et ainsi obtenir celui-ci dessous.

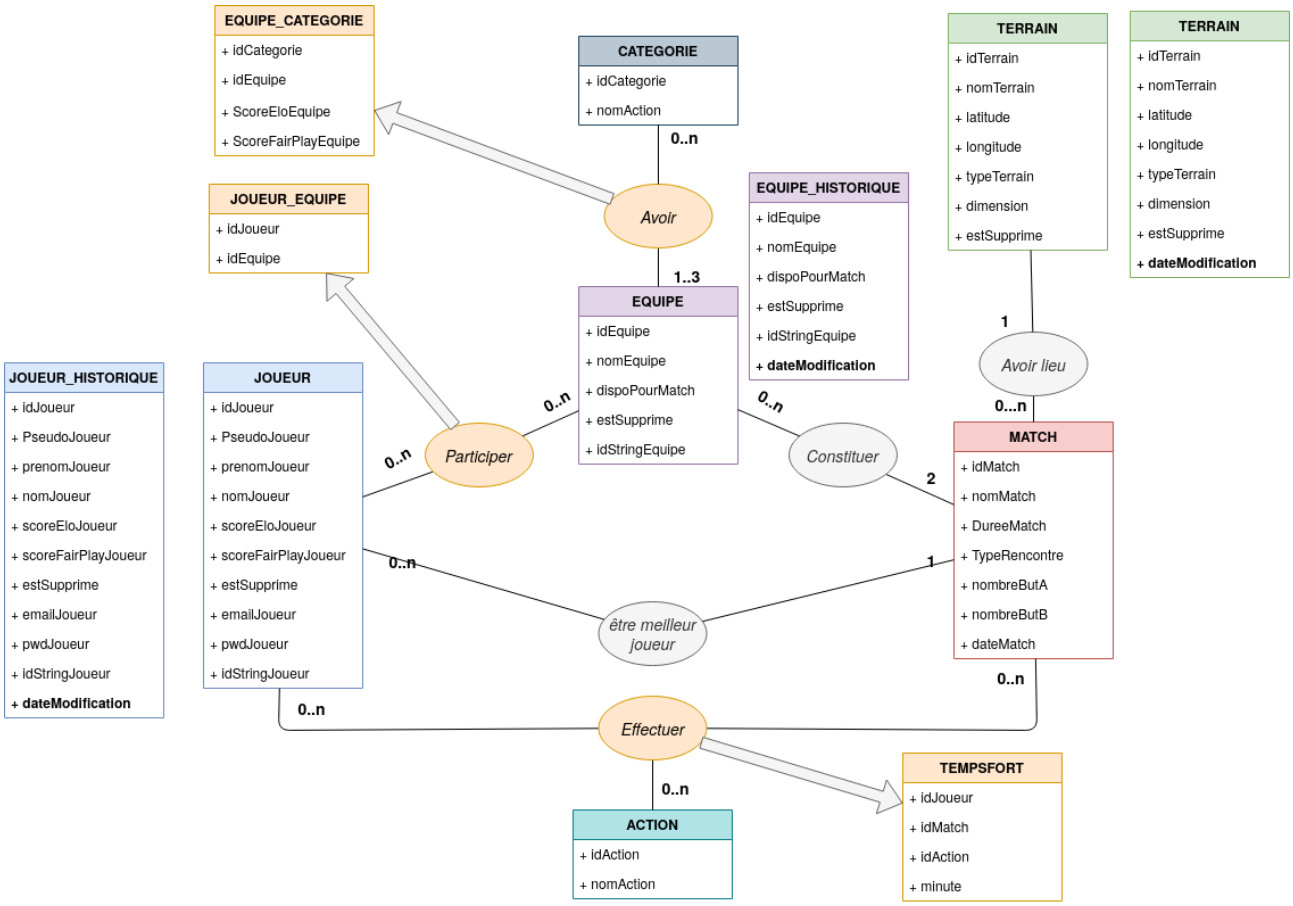

FIGURE 2 – Diagramme de classe amélioré

Comme vous pouvez le remarquer, la différence ici est la duplication des classes susceptibles d'être modifiées plus tard et donc on aimerait conserver historiques des modifications. On duplique donc ces classes en y ajoutant un attribut de temps (dateModification) qui nous permettra de classer les modifications par ordre chronologique.

2.2 Implémentation de l'API

Nous avons choisi d'implémenter l'API sous forme de service web de type *REST*. À parti de là nous aurions pu utiliser n'importe quel langage de programmation. Sachant que ce dernier n'est qu'un outil, vous pouvez très bien si vous le désirez utiliser le langage dans lequel vous êtes le plus à l'aise.

2.2.1 Les technologies

Nous avons choisi **Java** en particulier le framework *Spring* qui d'après nous est le plus adapté et permet une mise en production rapide de l'application, grâce à **Spring Boot** , une fois le développement terminer.

Pour des raisons de commodité, nous avons choisi *MySQL* comme système de gestion de base de données. Si vous souhaitez le modifier plus tard, il vous suffira de modifier le fichier de configuration de l'application Spring et éventuellement rajouté de nouvelles dépendances.

Nous avons décidé d'utiliser *Eclipse* comme environnement de développement. Cependant, rien ne vous empêche d'utiliser *Intelij* ou un autre IDE. Et si vous êtes un aventurier, vous prouvez utiliser un *Makefile* et tout configurer par vous-même.

2.2.2 Les fonctionnalités importantes

Parmi les fonctionnalités que nous avons développées, certaines sont plus complexes à comprendre que d'autres. C'est pourquoi, ici, nous allons vous détailler les plus importantes. Celles qui vous permettront de mieux comprendre le fonctionnement de l'API et ainsi faciliter leurs modifications.

L'historisation des données La gestion de l'historique des données des différentes tables a été en partie implémentée pour les données des joueurs et des équipes.

Il s'agit ici de garder une copie des données d'origine après avoir effectué une modification. Si un utilisateur change d'équipe, de nom ou encore lorsque son score elo change, il faudrait que nous puissions retrouve les données avant et après les différentes modifications.

C'est pourquoi, comme indiquer dans le diagramme de classe amélioré, nous avons dupliqué certaines tables en y ajoutant un nouvel attribut (date de modification). Ainsi quand notre application reçoit une requête des types PUT (modification), il effectue une copie des données dans la table historique associées à la table que l'on souhaite modifier et ensuite il modifie les données.

Nous aurions également pu développé des *déclencheurs* dans la base de données afin d'effectuer des taches similaires. Mais dans ce cas, il aurait fallu modifier ces déclencheurs lors d'une migration éventuelle de base de données. En effet, le développement de déclencheurs dépendant de la base de données dans laquelle elles sont et chaque base de données a un format unique.

Le score *elo* La mise à jour du score Elo est la fonctionnalité principale de notre API.

Cet algorithme prend en paramètre le résultat du match et effectue un ensemble l'opération avant de mettre à jour les scores Elo des équipes et des joueurs.

[insere algo ici apres implémentation .....]

2.3 Les bornes et limites de l'API

Sachant qu'il nous a fallu une période d'auto formation, avant d'entamer ce projet, nous n'avons pas pu implémenté toutes les fonctionnalités que nous nous étions fixées au départ .Toute fois avec cette documentation vous seriez en mesure de rajouter de nouveaux services.

Donc il faudra prête une attention particulière à la section Services disponible afin de savoir ce qu'il est possible ou pas de faire grâce à l'API actuel.

3 Installation et mise en production

3.1 Mise en production local

Pour lancer l'application en utilisant son ordinateur comme serveur. Il vous suffit de :

— S'assurer que vous avez le **.jar** de l'application *sokka-0.0.1-SNAPSHOT.jar*

— Spring Boot Rest

— Avoir JDK (Java Development Kit)

— Avoir MySql configuré comme suit :

```
1       GRANT ALL PRIVILEGES ON *.* TO 'sokka'’localhost’ IDENTIFIED
           BY ’sokka’;
           CREATE DATABASE sokka;
```

— Sur linux ouvrir un terminal et saisir **java -jar sokka-0.0.1-SNAPSHOT.jar**

— Sur Windows en principe un double clique sur **java -jar sokka-0.0.1-SNAPSHOT.jar** est suffisant

3.2 Configuration et personnalisation de la mise en production

Il vous suffit d'installer les outils nécessaires qui support Spring (pour rappelle nous avons utiliser Eclipse - Java - Spring Boot - MySQL). Une foi que vous avez l'environnement adéquat, il vous suffit d'importer le projet et de modifier le fichier de configuration de Spring afin qu'il respecte vos différentes attentes.

4 Prise en main

Vous l'aurez donc compris, la prise en main dans le but d'implémenter de nouvelle fonctionnalité passe donc par la maîtrise de :

— Java

— Spring Boot Rest

— MySQl

5 Services disponible

Voici la liste des services auxquels vous avez accès :

- Consultation
  - Liste des joueurs */sokka/api/v1/services/joueur/all*
  - Joueur qui a l'id i */sokka/api/v1/services/joueur/1*
  - Liste des joueurs de l'équipe d'id x */sokka/api/v1/services/joueur/equipe/1*
  - Liste des équipes */sokka/api/v1/services/equipe/all*
  - équipes qui a l'id i */sokka/api/v1/services/equipe/1*

- Création
  - Ajouter un joueur */sokka/api/v1/services/joueur/add*
  - Ajouter une liste de joueurs */sokka/api/v1/services/joueur/add/list*
  - Ajouter un joueur dans une équipe */sokka/api/v1/services/joueur/equipe/add ?idJoueur=1idEquipe=2*
  - Ajouter une équipe */sokka/api/v1/services/equipe/add*
  - Ajouter une liste d'équipes */sokka/api/v1/services/equipe/add/list*

- Suppression
  - Supprimer un joueur en fonction de son id */sokka/api/v1/services/joueur/delete/1*
  - Supprimer tous les joueurs */sokka/api/v1/services/joueur/delete/all*
  - Supprimer une équipe en fonction de son id */sokka/api/v1/services/equipe/delete/1*
  - Supprimer de toutes les équipes */sokka/api/v1/services/equipe/delete/all*

- Modification
  - Modifier un joueur en fonction de son id */sokka/api/v1/services/joueur/update*
  - Modifier une équipe en fonction de son id */sokka/api/v1/services/equipe/update*

6 Comment utiliser les services disponibles ?

Le moyen le plus pratique serait d'utiliser l'interface administration qui a été développée en parallèle de cette API. Cependant, cette dernière n'implémente pas tous les services développés dans l'API.

C'est pourquoi je vous recommande d'utiliser un outil comme RESTClient ou REST Easy (modules complémentaires pour Firefox) ou PostMan (modules complémentaires pour Chrome)

Dans les exemples suivants vous allons utiliser RESTClient et nous supposerons que l'API est déployé en local (**http ://localhost :8080**/). Après avoir ouvert RESTClient, vous obtiendrait une interface comme celle-ci dessous.

6.1 Service de type GET : consultation

Pour effectuer une requête de type GET il faut :

— Dans la partie **Request**, sélectionnez la méthode GET

— Dans la partie *URL*, saisissez l'URL du service que vous souhaiter consommer

— Cliquez sur *Send*

— Vous pouvez consulter le résultat dans l'onglet *Respose* ou *Preview* de la partie **Response**

Exemple : Consulter la liste des joueurs (http ://localhost :8080/sokka/api/v1/services/joueur/all)

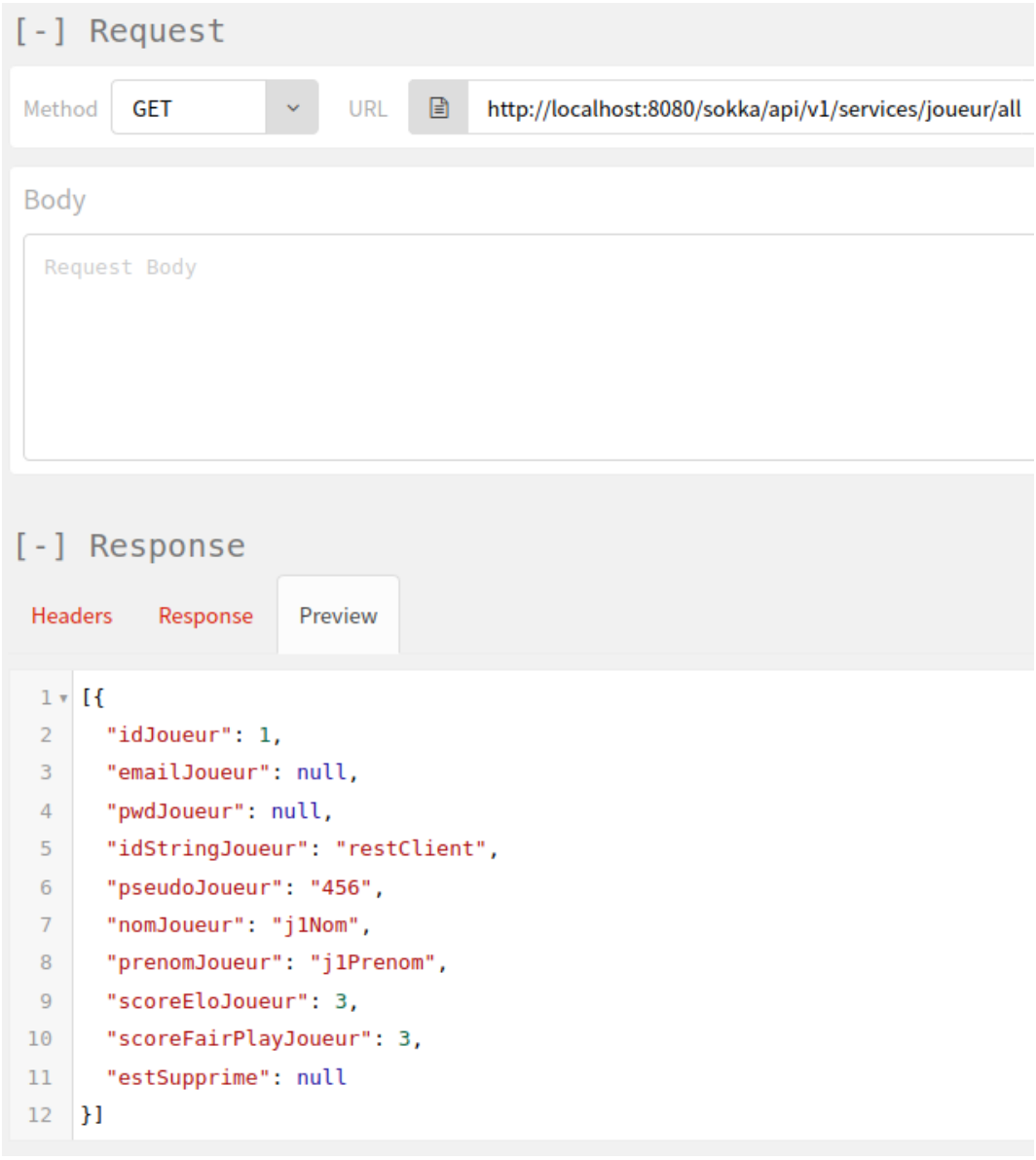

FIGURE 3 – RESTClient Méthode GET

6.2 Service de type POST : création

Pour effectuer une requête de type POST il faut :

— Dans la partie **Request**, sélectionnez la méthode POST

— Modifier le **Headers** : Name = *Content-Type* et Attribute Value = *application/json*

— Dans la partie *URL*, saisissez l'URL du service que vous souhaiter consommer

— Ajouter les données au format json et valider

— Cliquez sur *Send*

— Vous pouvez consulter le résultat dans dans l'onglet *Respose* ou *Preview* de la partie **Response**

Exemple : Ajouter une liste d'équipe (http ://localhost :8080/sokka/api/v1/services/equipe/add/list)

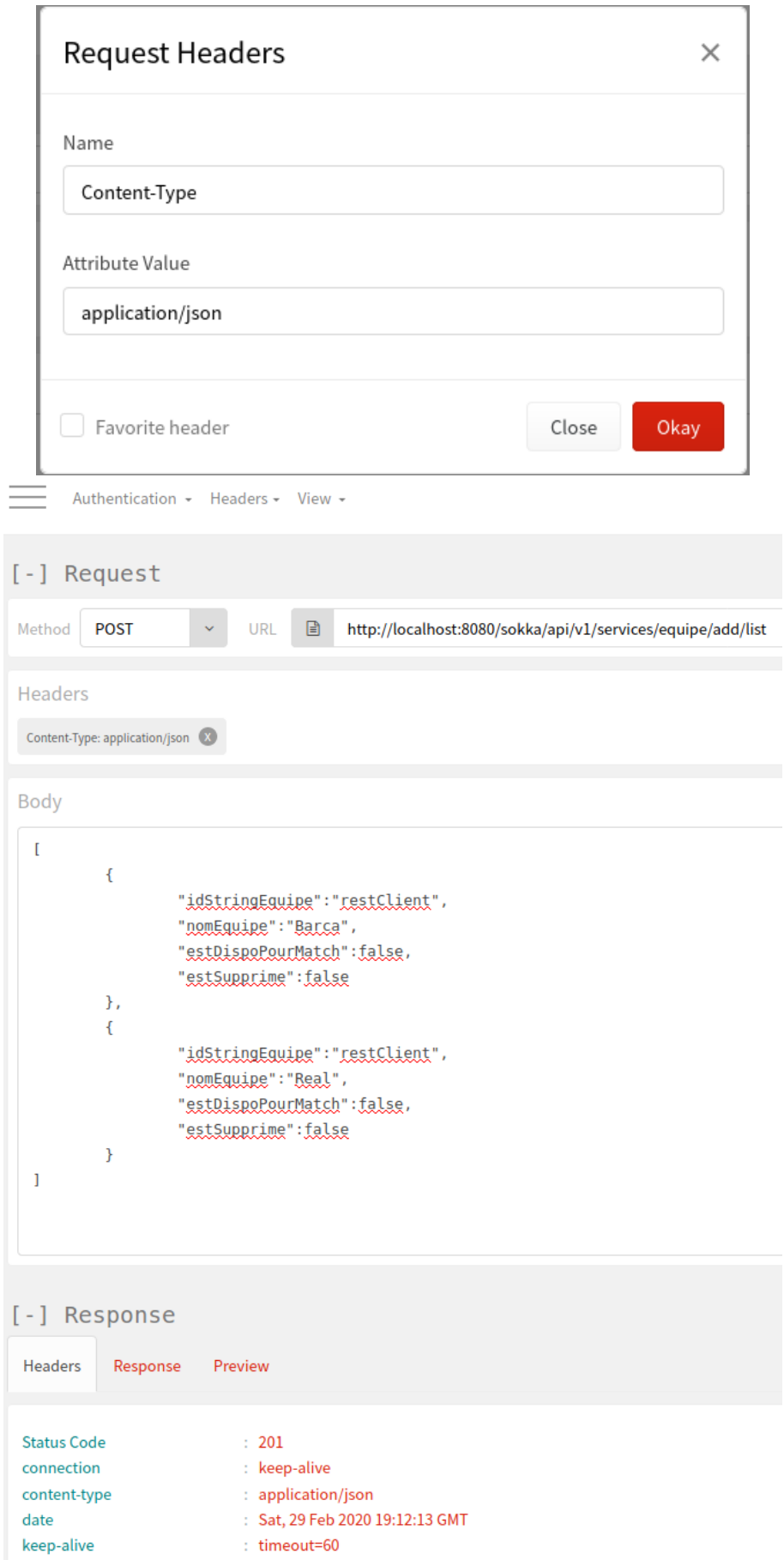

FIGURE 4 – RESTClient Méthode POST

6.3 Service de type DELETE : suppression

Pour effectuer une requête de type DELETE il faut :

— Dans la partie **Request**, sélectionnez la méthode DELETE

— Dans la partie *URL*, saisissez l'URL du service que vous souhaiter consommer

— Cliquez sur *Send*

— Vous pouvez consulter le résultat dans dans l'onglet *Respose* ou *Preview* de la partie **Response**

Exemple : Supprimer le joueur d'id 1 (http ://localhost :8080/sokka/api/v1/services/joueur/delete/1)

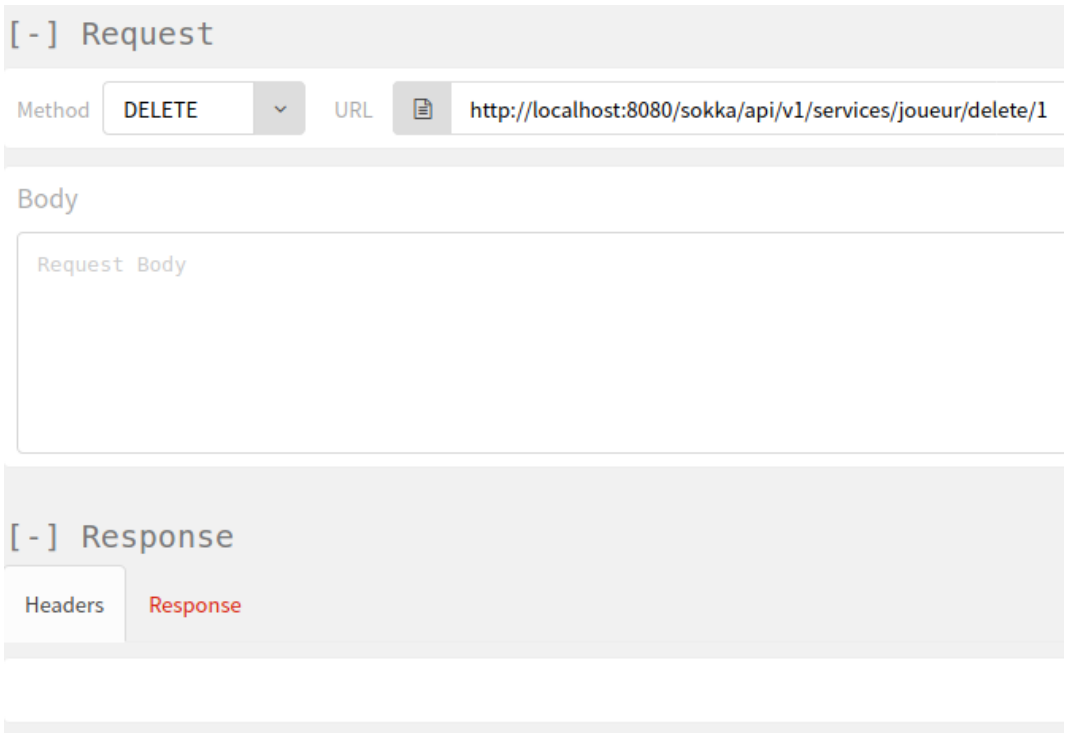

FIGURE 5 – RESTClient Méthode DELETE

6.4 Service de type PUT : modification

Pour effectuer une requête de type PUT il faut :

— Dans la partie **Request**, sélectionnez la méthode PUT

— Modifier le **Headers** : Name = *Content-Type* et Attribute Value = *application/json*

— Dans la partie *URL*, saisissez l'URL du service que vous souhaiter consommer

— Ajouter les données au format json (ne pas oublier de préciser l'id de la donnée à modifier) et valider

— Cliquez sur *Send*

— Vous pouvez consulter le résultat dans dans l'onglet *Respose* ou *Preview* de la partie **Response**

Exemple : Ajouter un joueur (http ://localhost :8080/sokka/api/v1/services/joueur/update)

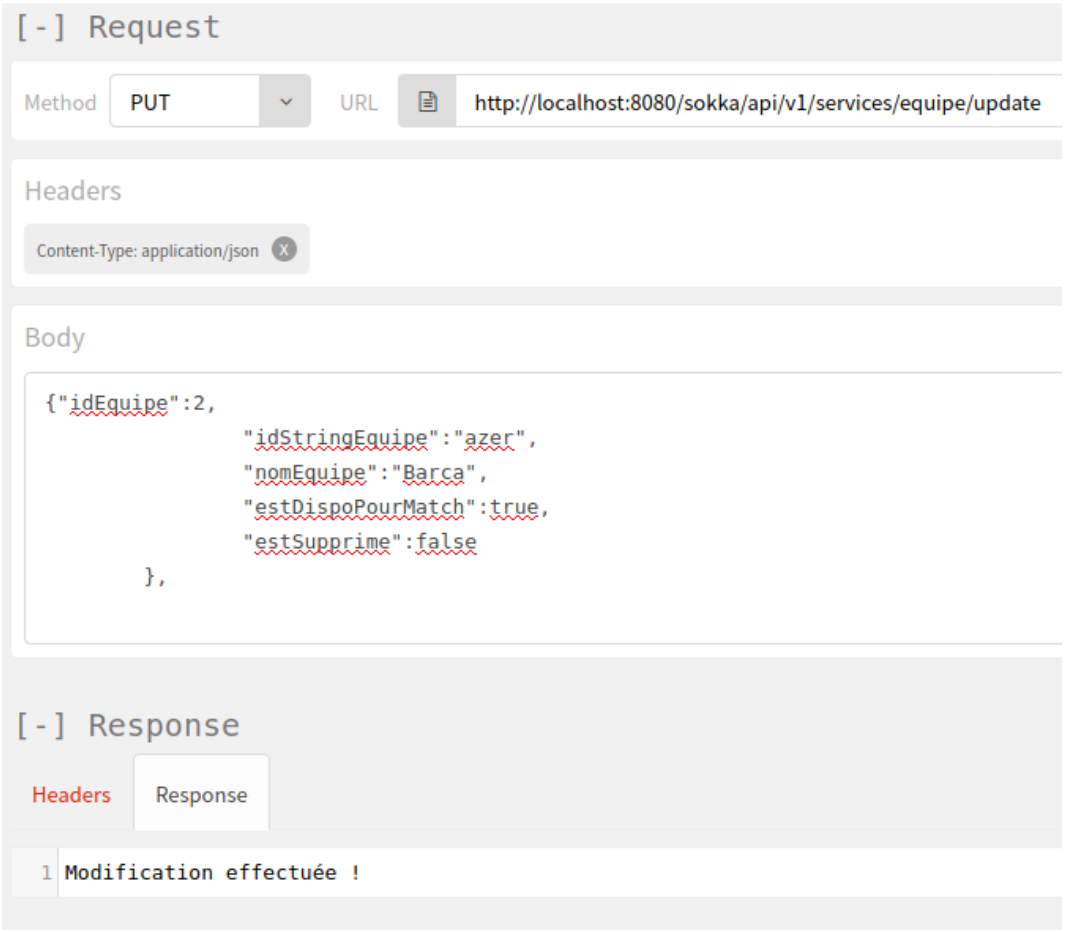

FIGURE 6 – RESTClient Méthode PUT

7 Comment ajouter de nouveaux services ?

Pour ajouter de nouveaux service, il faut :

— Créer une nouvelle interface de dépôt dans le package *com.sokka.dao* si c'est pour un service donc l'interface n'a pas encore été créée

— Créer ou modifier la classe ou le package service qui vous intéresse

— Créer ou modifier la classe controller qui vous intéresse

8 Amélioration

Les améliorations possibles sont nombreuses, tant au niveau de la conception que de l'implémentation. Ici, nous allons vous présenter les améliorations qui nous semblent les plus pertinentes.

8.1 Les nouveaux services

Il y a encore de nombreux services qu'il faudrait rajoutés. En effet dans la version actuelle de l'API, seuls les services sur les joueurs et les équipes ont été développés. Il faudrait donc rajouter des services sur les matchs, les terrains, les actions effectués ...

8.2 La sécurité

Un des points les plus importants à prendre en compte lors de l'amélioration de l'API est la mise en place d'un service d'authentification, afin de réserver l'accès de certains services. Vue que nous utilisons spring, il faudra, pour implémenté la sécurité, utilise **spring sécurity**.

8.3 Les exceptions

La gestion des exceptions n'est pas le plus urgente, car pour notre application elle n'est pas bloquante. Mais elle permettra d'avoir un meilleur retour sur la nature des erreurs que génère l'utilisateur de certains services.

8.4 L'historisation des données

Les fonctionnalités implémentées actuellement font en sorte qu'à chaque modification, il y a deux opérations qui sont effectuées, une opération de modification (on effectue la modification demandée) et une opération de création (on sauvegarde la donnée à modifier dans la table historique correspondante). C'est-à-dire que si on a 10 milles modification notre service web effectuera 20 milles requêtes. Ce qui à notre avis n'ai pas très optimale, il faudrait donc faire une planification des horaires de sauvegardes (en gardant la dernière modification de l'heure ou de la journée par exemple).

Il faudrait donc discuter de cette situation avec le client et trouver la solution qui lui conviendrait le plus.

9 Conclusion

Malgré, tout ce qui a été implémenté, il reste encore beaucoup de travail avant d'arriver à une API complète sécurisée et prêtre a être mise en production.