HE2B ESI

BACHELIER EN INFORMATIQUE DE GESTION

Développement de compétences informatiques dans un environnement unique

Rapport de stage effectué dans la firme Union Royale Belge des Sociétés de Football-Association

Winska Marika

Février 2022

PÔLE UNIVERSITAIRE EUROPÉEN DE BRUXELLES WALLONIE

Abrégé

Dans ce rapport de stage est présenté le suivi de ma première expérience professionnelle dans le domaine de l'informatique au sein de la Royal Belgian Football Association, la plus grande fédération sportive de Belgique qui compte plus d'un demi-million de membres. Durant mon stage, j'étais chargée du poste d'ingénieur de données. Je devais donc manipuler le grand volume de données que possède la fédération. Ces données sont utilisées par leur site web et leur application qui sont eux-mêmes utilisés par des milliers d'utilisateurs quotidiennement dans le but d'avoir un suivi de leur passion et de rassembler l'essentiel du football belge en un seul et même canal pour tous les fans et membres.

Ces services récupèrent des données telles que les matchs prévus durant la saison, les joueurs, les arbitres, le staff, etc. Ceux-ci étaient déjà implémentés mais certains avaient besoin d'être corrigés en fonction des rapports de bugs faits par les utilisateurs. Certains services renvoyaient des données erronées ou parfois aucune donnée ceci créait des problèmes d'intégrité de données. Ces services sont utilisés en interne par l'application mobile et l'application web de la RBFA. Mais ils sont également utilisés par des applications externes, ce qui accroit le besoin de cohérence.

Ce stage s'est déroulé du 07 février au 27 mai 2022.

Abstract

In this internship report I follow up on my first professional experience in IT at the Royal Belgian Football Association, the largest sports federation in Belgium with more than half a million members. During my internship, I was data engineer and I had to handle the large flow of data that the federation has. This data is used by the website and application serving thousands of users per day to follow their passion and to bring together the essentials of Belgian football in one channel for all fans and members.

The main goal of this internship was the implementation of services. These services retrieved data such as the matches scheduled during the season, the players, the referees, the staff etc. These services were already implemented but some needed to be fixed according to bug reported by users. Some services presented some data inconsistencies and integrity problems. These services are used internally by the mobile application and the web application of the RBFA but will in a near future also be used by external applications. The fact that these services will be used by external applications, made the need for consistency grow.

This internship took place from 07 February to 27 May 2022.

Table des matières

1. Introduction

Pour l'obtention du grade de bachelor en informatique de gestion à la Haute Ecole Bruxelles-Brabant, j'ai effectué, du 07 février 2022 au 27 mai 2022, un stage d'intégration dans le monde professionnel dans l'entreprise Union Royale Belge des Sociétés de Football-Association (URBSFA), la plus grande fédération sportive de Belgique.

Etant joueuse de football depuis mon plus jeune âge, je fais partie des 542.387 membres de la fédération. Mon intérêt pour l'URBSFA n'est donc pas anodin. Je connais l'importance et les responsabilités de celle-ci dans son secteur d'activité, ce qui a d'autant plus motivé mon choix d'y être stagiaire. De ce fait, j'ai pu connaître l'autre face de la fédération, ne plus avoir le simple rôle de membre et y avoir réellement une part active.

Au cours de ce stage au département informatique, j'ai pu m'intégrer intégralement dans une entreprise et développer mes compétences dans un cadre propice grâce aux différents outils mis à ma disposition et l'expérience et expertise des personnes autour de moi. J'ai appris en profondeur le métier d'ingénieur de données, plus connu sous le nom de Data Engineer. Durant mon stage j'ai analysé en détails l'architecture des bases de données contenants un large volume de données que j'ai pu manipuler dans le cadre des tâches qui m'ont été confiées. Grâce à ce stage j'ai pu mettre en pratique en situation réelle principalement les cours de persistances de données donnés lors de mon cursus de 3 ans à l'Ecole Supérieure d'Informatique.

Tout au long de ce rapport, je détaillerai mes 16 semaines de stage informatique. Dans un premier temps, je présenterai l'environnement, la méthode et les outils de travail. L'objectif plus détaillé de mon stage. Ensuite, je décrirai les différents projets sur lesquels j'ai travaillé et j'énumérerai les différentes missions qui m'ont été données avec une présentation, les problèmes rencontrés et les solutions trouvées pour chacune de celles-ci. Je terminerai par expliquer comment cette expérience a contribuée à mon développement et ce qu'elle a apportée pour ma carrière future.

2. Présentation

2.1. Entreprise

2.1.1. Histoire

L'Union royale belge des Sociétés de Football Association (URBSFA), est la fédération belge de football officiellement reconnue par la Fédération Internationale de Football Association (FIFA). Fondée le 1er septembre 1895, par dix clubs de football dont cinq existent toujours actuellement, elle est la plus grande fédération sportive belge. L'URBSFA se compose de deux branches. La première est le football professionnel, où les clubs professionnels se sont associés au sein d'une organisation représentative, la Pro League. La deuxième partie est le football amateur, constitué de deux ailes régionales : l'ACFF (Association des Clubs Francophones de Football) et Voetbal Vlaanderen. Au fil du temps, la fédération et le nombre de membres grandissent. En 2010, l'URBSFA compte 412.122 affiliés qui représentent plus de 2.000 clubs et 17.000 équipes. Au XXI siècle la fédération a lancé son propre site web et son application BBF (Best of Belgian Football) qui permettaient à 450.000 utilisateurs de voir les matchs prévus durant la saison en cours, les clubs et les équipes. Le strict minimum afin que les membres puissent suivre leur activité saison par saison.

2.1.2. Présent

En 2020, à l'occasion de son 125e anniversaire, l'URBSFA se réinvente et met en place plusieurs nouveautés afin de se moderniser et de se développer toujours plus. Notamment en adoptant une nouvelle identité visuelle, avec un nouveau logo qui met en avant l'appellation internationalisée, RBFA (Royal Belgian Football Association), à des fins marketings. En 2021, la RBFA s'est développée au point que l'application existante ne suffisait plus. En mars de la même année, elle décide donc de lancer une toute nouvelle application RBFA qui apporte nombreuses nouveautés tout en gardant ses fonctionnalités principales. Nous pouvons désormais retrouver des publicités, des quizz qui favorisent l'interaction avec ses nombreux fans de football belge professionnel et amateur, un tableau de bord avec les dernières actualités, des

interviews, des diffusions en temps réel, la possibilité d'achat des tickets de matchs etc. Douze ans plus tard, la fédération a dépassé le demi-million de membres et a presque doublé le nombre de clubs. En collaboration avec l'ACFF et Voetbal Vlaanderen, l'URBSFA organise 12.000 matches par week-end et 300.000 matchs par an, y compris les matches des équipes nationales. Le siège social de l'entreprise se situe à Tubize (480 Rue de Bruxelles, 1480 Tubize), où plus de 200 collaborateurs travaillent dans un tout nouveau bâtiment au Proximus Basecamp, appelé également le cœur du football en Belgique.

2.1.3. Département informatique

Le département informatique est composé d'une dizaine de personnes, qui collaborent avec une équipe externe de consultants pour le développement des services comme le site web et l'application, utilisés quotidiennement par des milliers d'utilisateurs. Le responsable IT est Monsieur Koen Landsheere qui fait partie de la fédération depuis plus de 25 ans. Toute l'équipe travaille dans un open space qui avantage la communication. Plusieurs salles de réunion sont également à notre disposition.

2.1.4. Méthode de travail

La méthode de travail utilisé au sein de l'équipe IT en collaboration avec les équipes externes de consultants est la méthode SCRUM. Il y a un Scrum Master qui favorise la communication au sein de l'équipe. Un Product Owner, le chef de projet, qui prend les décisions concernant les priorités et les fonctionnalités à développer ou corriger. Il y a également l'équipe de développement qui met en pratique ses compétences afin de transformer les besoins définis par le Product Owner en fonctionnalités utilisables. Chaque matin une réunion est organisée pour suivre l'avancement du travail qui est structuré en « sprint ». Un sprint dure en moyenne deux semaines après lesquelles une version améliorée et utilisable du produit doit être effectuée. Un Scrum Board (voir ci-dessous) est également utilisé afin de suivre l'évolution du projet avec les tâches répertoriées : à faire, en cours, à vérifier, à tester et achevées. Cette méthode de travail favorise la productivité, la créativité, l'organisation et installe un rythme de travail qui avantage l'efficacité sur le long terme.

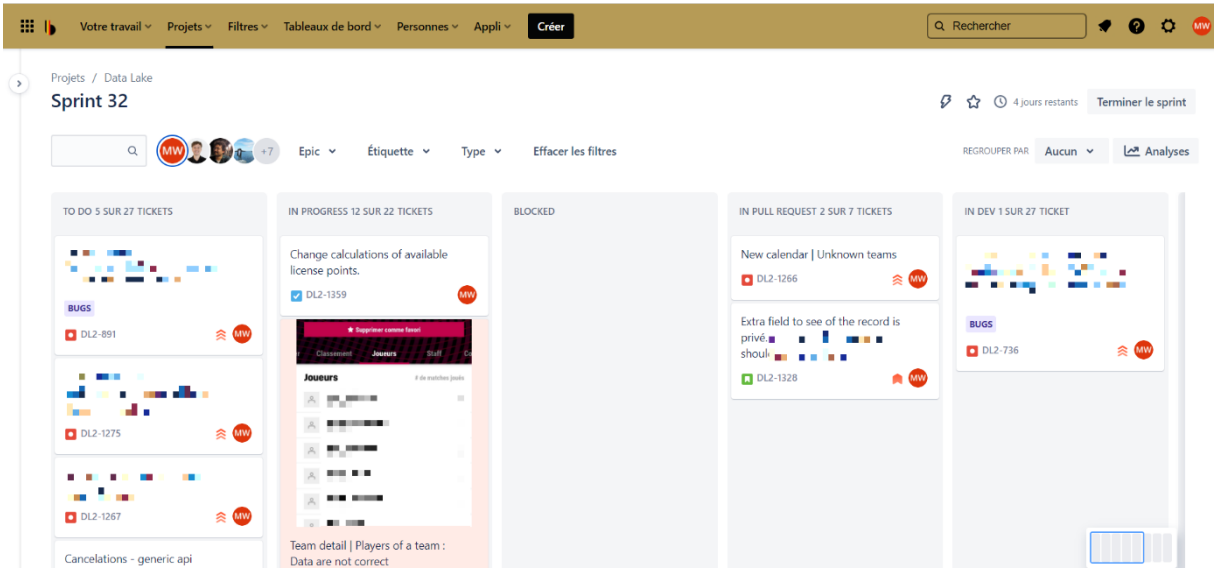


Tableau de bord

De plus, l'équipe ICT organise mensuellement une réunion « rétrospective » qui a pour but de faire le point sur les choses qui se sont bien passés durant le mois passé et à identifier des points à améliorer au sein de l'équipe, comme la communication, la ponctualité, la gestion des salles de réunions etc. Elle organise également tous les mois une réunion « démonstration » qui permet à quelques membres de l'équipe de montrer leurs accomplissements (développements) du dernier mois. Celle-ci permet donc d'en apprendre davantage sur les projets sur lesquels l'équipe travaille actuellement. Enfin, un team-building est organisé tous les mois avec toute l'équipe ICT afin de renforcer les liens, la communication et l'esprit d'équipe.

2.1.5. Mission

Dans le cadre de mon stage, en tant qu'ingénieur de données, j'ai travaillé sur un projet de nouvelle plateforme digitale, qui est le nouveau canal de communication entre l'URBSFA, les clubs, les arbitres, etc. La fédération a un grand nombre de données relatives aux clients à gérer dû à son importance dans son domaine sportif. Mon objectif est de saisir, traiter, structurer, documenter et stocker ces données volumineuses qui proviennent de nombreux endroits comme les réseaux sociaux, service de gestion des membres, application, site web. Une des tâches est l'automatisation de l'ingestion de données dans le data lake, endroit où affluent toutes les données, par le développement de pipelines ETL (Extract, Transform, Load). Pour exemple, lorsqu'un nouveau membre est ajouté à la fédération ses diverses données

doivent être récupérées, chargées dans la base de données au bon endroit pour pouvoir ensuite être réutilisées par le Generic API notamment. Ceci est donc fait grâce au processus ETL. L'objectif final du projet/service était de mieux comprendre ses joueurs, ses membres, ses fans et toutes les personnes qui interagissent d'une manière ou d'une autre avec son organisation, afin de mieux les servir et d'offrir une expérience personnalisée à chaque utilisateur des plateformes mais avant ça il y a de nombreuses étapes à exécuter qui seront détaillées ci-dessous.

2.2. Outils et systèmes

2.2.1. Système d’exploitation

La fédération a mis à ma disposition un ordinateur portable avec le système d’exploitation Windows 10 qui m’a permis, une fois connecté au réseau de l’entreprise, d’accéder aisément à toutes les ressources utiles dans le cadre de mon stage.

2.2.2. Langages de programmation

Le langage de programmation principalement utilisé pour développer les différents services de la fédération est Python en raison de sa facilité d’utilisation, sa polyvalence, sa performance et de ses bibliothèques intégrées permettant l’accès aux bases de données qui deviennent de plus en plus volumineuses avec l’accumulation quotidienne des données. De plus, pour transformer les données issues du data lake vers le data warehouse nous utilisons le langage SQL, inévitable lorsque nous manipulons des bases de données.

2.2.3. Services Cloud

La plateforme utilisée pour la gestion de l’infrastructure est la plateforme Google Cloud Platform (GCP) qui regroupe les différents services cloud de Google tels que le stockage (Datastore) d’un large volume de données non structurées, le Big Data (BigQuery qui permet d’accéder rapidement et facilement aux données grâce aux requêtes SQL), la sécurité, le développement d’applications etc. GCP présente l’avantage d’épargner à l’entreprise la gestion de l’infrastructure, l’approvisionnement

des serveurs et la configuration des réseaux. Constamment mise à jour et optimisée, la plateforme est performante, économique et sécurisée. De plus, en utilisant GCP et, de ce fait, l'utilisation des ressources de Cloud partagées favorise l'optimisation de coûts.

2.2.4. Pycharm

Pycharm est un environnement de développement intégré (IDE) développé par JetBrains pour programmer en Python. Cet IDE est utilisé sur conseil de l'équipe de développeurs de la RBFA. Cet IDE présente des avantages tels que la coloration de code, la navigation dans le code et surtout la possibilité de l'utiliser pour lancer les script pour lancer le code en local.

2.2.5. GitHub

Github est un site d'hébergement de code qui utilise le logiciel de version Git. Pour chaque tâche je devais créer une nouvelle branche depuis la branche develop. Ceci m'a permis de travailler sans effectuer de changements sur la branche principale de développement tant que ceux-ci ne soient pas approuvés par les responsables via les pull requests.

2.2.6. Firestore

Firestore Cloud est une base de données NoSQL. Celle-ci est utilisée pour persister les données de la RBFA. Via la console Google, je peux faire des requêtes sur les tables des bases de données de la RBFA. J'ai uniquement reçu l'accès pour faire des requêtes sur une base de données en développement. Ceci compliquait parfois les choses, car les bugs étaient reportés en production et donc pas nécessairement reproduisible en développement.

2.2.7. Oracle

Certaines données sont persistés sur Oracle, mais uniquement en production. Pour le développement une copie de cette base de données a été faite sur Firestore.

2.2.8. Swagger UI

Swagger UI est une application qui permet de donner une interface visuelle à un projet de services. Cette application crée une page qui regroupe tous les endpoints pour les APIs d'un projet et permet de plus facilement faire appel à ces APIs en précisant les paramètres nécessaire à un appel. Cet outil facilite le testing des endpoints du projet grâce à l'interface facile à utiliser pour aussi bien les développeurs que les clients qui ne connaissent pas la logique d'implémentation. Le projet Generic API a été paramétré avec un script pour se lancer automatiquement dans SwaggerUI.

2.2.9. Slack

Slack est l'outil utilisé pour communiquer au sein de l'équipe. Via les multiples canaux, nous pouvions contacter les membres travaillant sur les différents projets. Celui-ci m'a également été utile dans la communication avec les personnes externes à la RBFA avec qui nous étions en collaboration.

2.2.10. Instabug

Instabug est un outil de signalement de bugs et de crashs. Ajouté à une application mobile, cet outil permet aux utilisateurs d'envoyer un rapport complet en cas de problème. Ces rapports sont ensuite utilisés lors de la création des tickets sur lesquels j'ai travaillé durant mon stage.

3. Projets

3.1. RBFA Workshop Sandboxes

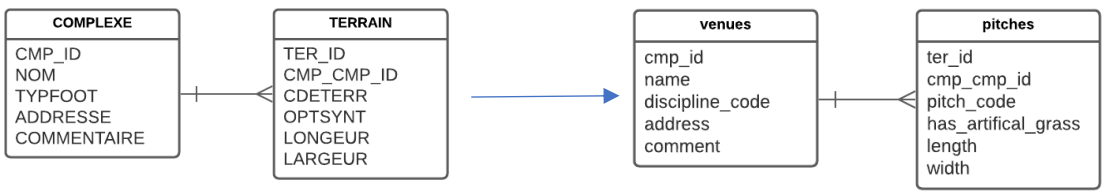
RBFA Workshop Sandboxes est le premier projet sur lequel j'ai travaillé qui m'a servi d'introduction et d'apprentissage de Google Cloud Platform, outil permettant la gestion des services cloud décrits ci-dessus. Ce projet consiste à créer un service web comportant différents endpoints/tâches en utilisant le langage de programmation Python. J'ai eu la liberté de choix du framework. Etant donné que le projet principal, le

RBFA Generic API utilise FastAPI, j'ai décidé de l'utiliser et de, par la même occasion, l'apprendre. La première tâche consistait à simplement afficher « Hello World ! » pour le endpoint principal et déployer cela sur App Engine. Pour la seconde tâche, lorsque l'endpoint « /insert » était appelé, la date et l'heure de l'appel devait être insérées dans une table BigQuery sur GCP (Insert annexe 5.2.1). La troisième tâche consistait à envoyer un fichier sur le Cloud Storage de GCP qui contenait la date et l'heure de l'appel du endpoint « /upload » (Upload annexe 5.2.2). La quatrième tâche était de mettre à jour une entité dans le Datastore. Ici le timestamp était changé en current timestamp (voir annexe 5.2.3). La cinquième et dernière tâche consistait à automatiser une des tâches. Le endpoint « /insert » devait être appelé toutes les 5 minutes grâce au Cloud Scheduler (voir annexe 5.2.4).

Ce projet est réalisé par chaque nouvelle personne dans l'équipe IT afin de s'intégrer et apprendre l'un des outils principaux utilisés. Ceci m'a pris une semaine à le faire car chaque tâche effectuée devait être contrôlée via des pull requests sur GitHub par mon responsable et un collègue qui a lui-même réalisé ce projet un mois plus tôt.

3.2. RBFA Data Pipeline

Le projet RBFA Data Pipeline est le projet sur lequel j'ai majoritairement travaillé durant mon stage. L'un des principaux objectifs de celui-ci est de créer des transformations de la base de données Oracle SQL vers la base de données NoSQL Firestore (GCP). Par exemple, nous avons deux tables dans Oracle, *complexe* et *terrain* de la manière suivante.



Oracle *Firestore*

Ici, *complexe* est une table avec une clé primaire *cmp_id* et *terrain* est une table avec une clé primaire *ter_id* et une clé étrangère *cmp_cmp_id* qui pointe vers la table *complexe*.

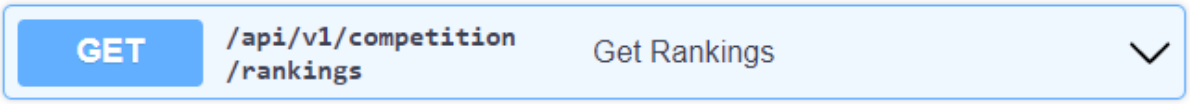
Dans Firestore, *venues* est une collection et *pitches* est une sous-collection de *venues*.

Nous faisons cela parce que Cloud Firestore est une base de données orientée documents ce qui permet le scaling automatique pour le stockage, la synchronisation et l'accès rapides aux données grâce à la génération automatique des ids par Google Cloud Platform pour les applications mobiles et web. Elle offre une intégration transparente avec d'autres produits Firebase et Google Cloud Platform.

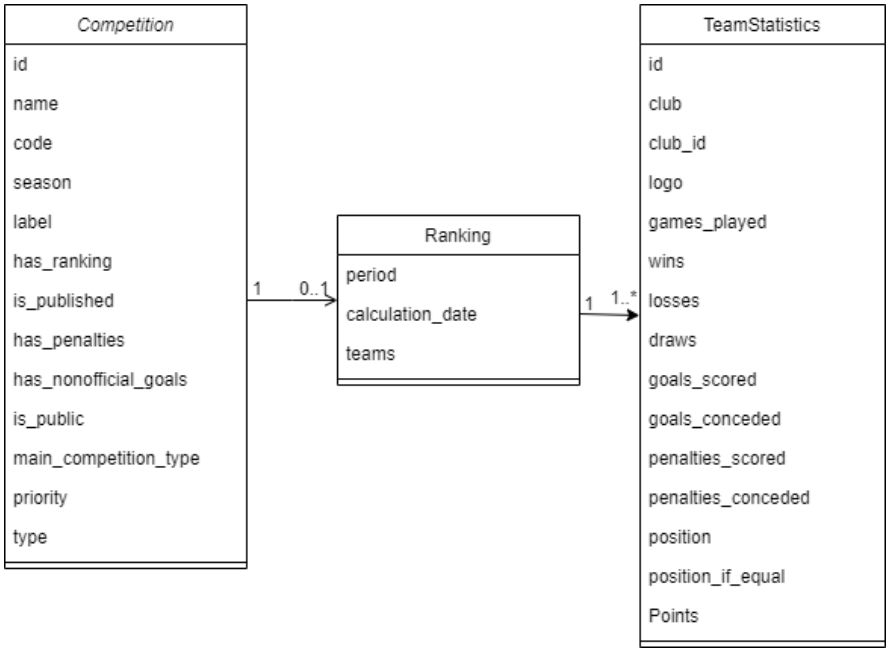
Ceci est simple d'utilisation, il n'y a pas de schéma et l'utilisation de RESTful HTTP API est possible (qu'on verra un peu plus loin). De plus, nous passons de 439 tables dans la base de données Oracle à 64 tables dans Firestore ce qui facilite grandement la manipulation de toutes ces données.

L'ingestion de données est faite en temps réel et comme nous l'avons pu remarquer énormément de données doivent être gérées ce qui peut provoquer des incohérences entre les deux et notamment des données manquantes dans Firestore. Ceci provoque donc des confusions chez les utilisateurs de l'application qui seront détaillées ci-dessous. De ce fait, l'intégrité de ces données doit être régulièrement vérifiée afin que la qualité des données soit la meilleure possible. Pour cela, les responsables ont décidé de mettre en place des scripts de validation de données qui sont lancés afin de récupérer les incohérences et pouvoir recharger les données manquantes. De nombreux bugs ont été rapportés par les utilisateurs grâce à l'outil Instabug intégré à l'application. Pour tous ces bugs j'ai donc créé des scripts, constitués de requêtes SQL afin de récupérer ces données dans les nombreuses et complexes tables existantes. La manipulation de ces bases de données était faite sur BigQuery, service de Google Cloud Platform. Pour l'utilisation de ce service, la tarification se fait à la demande et le prix est de 6$ par TB. De ce fait, mes requêtes doivent être le plus optimisé possible et utiliser le moins de ressources possibles pour effectuer une tâche étant donné que ces scripts vont être relancés régulièrement.

3.2.1. Classement d'une compétition

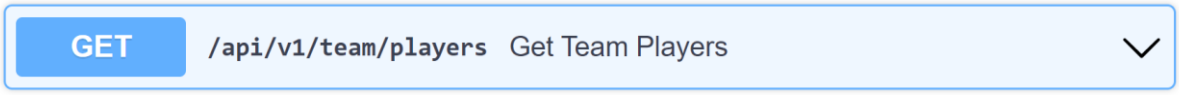


Pour cet endpoint, l'API rend le classement pour une compétition donnée. Ce classement est calculé à partir des statistiques de chaque équipe qui fait partie de cette compétition. Le classement est calculé lorsque le champ *has_ranking* est égal à « Y »

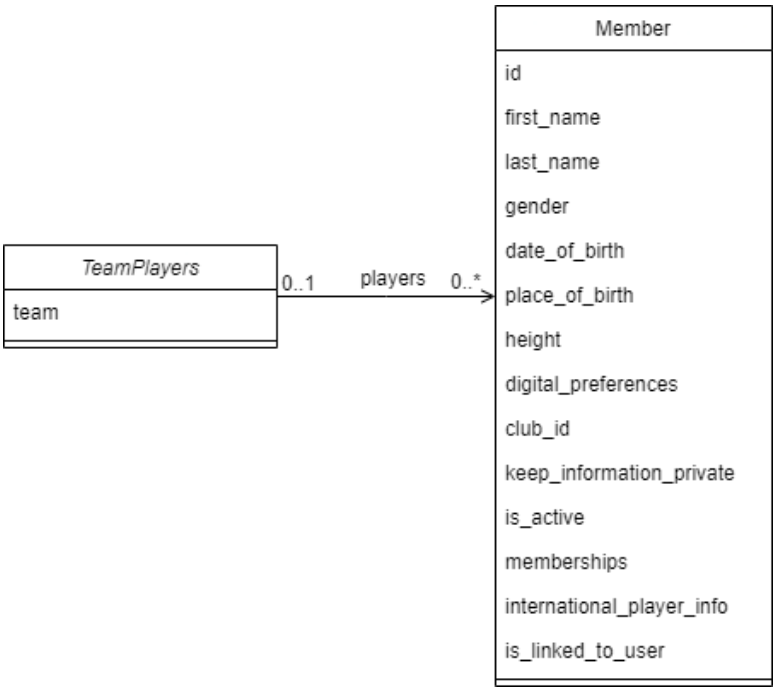


Un des bugs signalés était le classement qui ne s'affichait pas pour certaines compétitions. Ceci était provoqué par le fait que ces compétitions n'avaient pas de sous collection « rankings » dans Firestore. En utilisant les tables *competitions* et *rankings* de BigQuery j'ai créé une requête SQL (voir annexe 5.3.1) qui récupère tous les ids et types des compétitions qui ont leur champ « has_ranking » = Y et qui n'apparaissent pas dans la table *rankings.* L'id et le type servent à récupérer les bons documents d'Oracle grâce au chemin construit de la sorte « type/id ». Cette requête rendait 8.059 compétitions sans classement alors que le champ has_ranking était égal à « Y » pour un total de 30.470 compétitions dans la base de données.

3.2.2. Membres d'une équipe



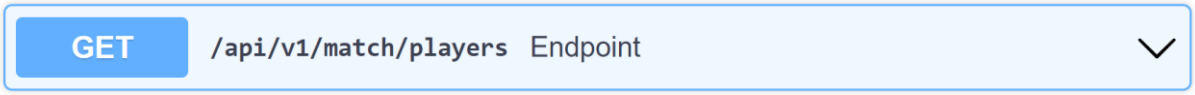
L'application permet d'avoir un aperçu général des joueurs pour une équipe spécifique. Dès qu'un joueur joue un match pour cette équipe il doit apparaitre dans l'onglet correspondant.

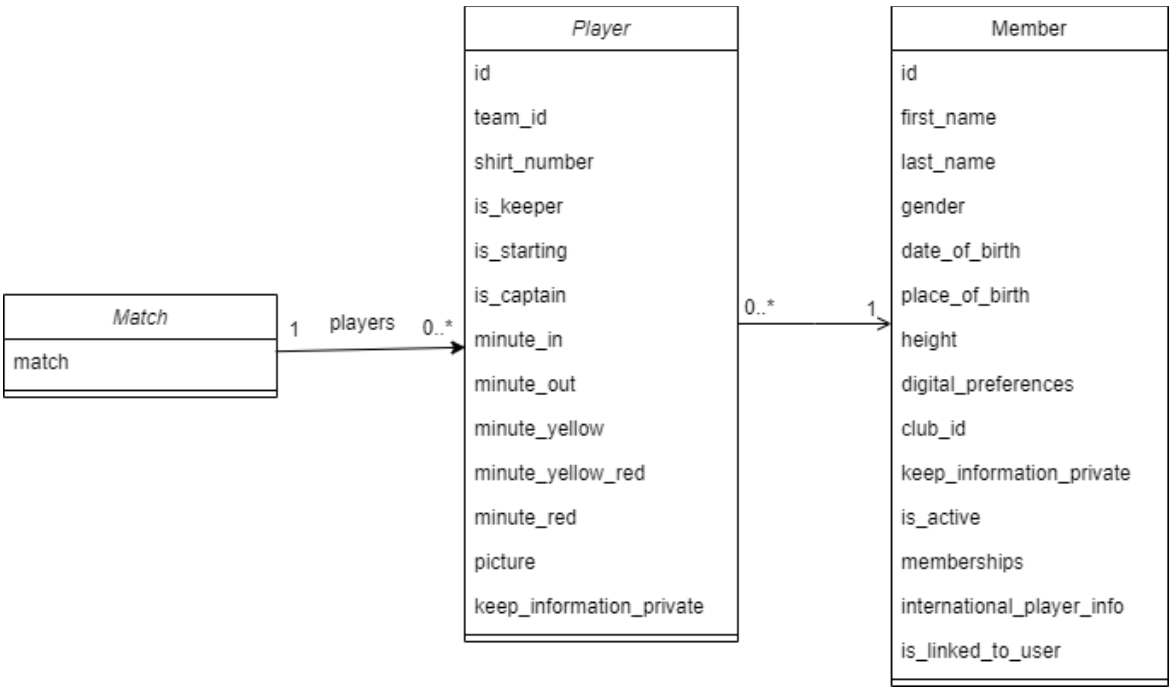


Cependant, nous avons pu constater que certains membres/joueurs ne s'affichaient pas. La cause de ceci était que la référence vers ces membres ne se trouvait pas dans le champ *member_refs* dans la table *teams* pour l'équipe concernée. Pour écrire la requête SQL (voir annexe 5.3.2), je n'avais pas de lien direct entre un membre et son équipe si ce membre ne s'y trouvait pas. C'est pourquoi il fallait faire une jointure avec la table *player_stats.* Dans cette table nous avons les deux informations nécessaires, c'est-à-dire l'id de l'équipe avec le champ eqsec_id et la référence vers le membre que je récupère depuis le champ __key__path. Ce champ contient dans un premier lieu le nom de table *members,* ensuite l'id Firestore du membre correspondant, le nom de la table *player_stats* et enfin l'id Firestore du joueur. Je devais donc seulement récupérer l'id FIrestore du membre afin de pouvoir faire le join sur la table *members.* Par la suite,

je devais formatter le résultat du champ __key__.path de la table *members* qui était sous la forme *([members], [id])* afin d'avoir « members/id ». Une fois que j'avais ce résultat je pouvais récupérer les équipes où la référence vers ce membre manquait et ce membre avait l'id de cette équipe dans la table *player_stats*. Au total, 220 membres ne se trouvaient pas dans l'aperçu général de leur équipe.

3.2.3. Joueur membre

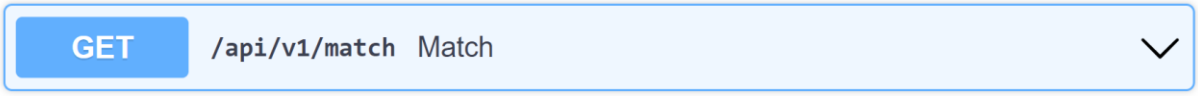


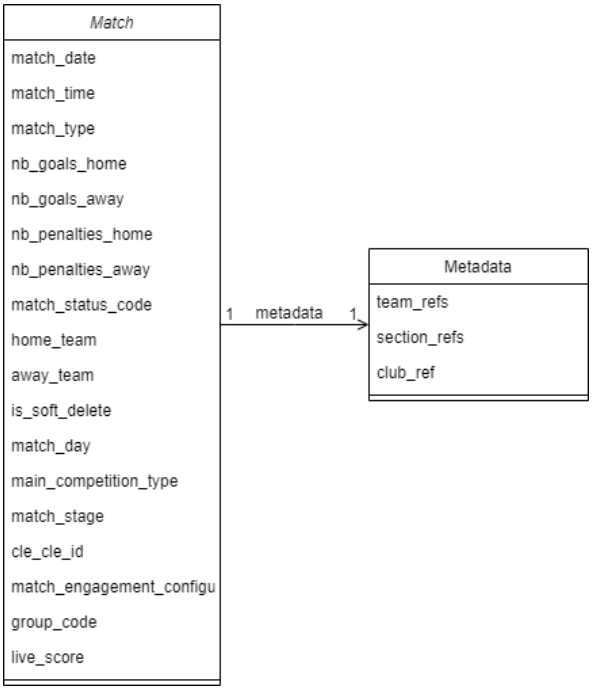


Le problème suivant concernait les joueurs d'un match qui n'étaient pas cliquables dans l'application parce que les détails de ceux-ci ne pouvaient pas être chargés. Ceci était dû au fait que dans la base de données ces joueurs n'avaient pas de référence vers un membre ou le membre n'existait plus dans la table correspondante. Ceci peut être provoqué par le fait qu'un membre n'existe plus car il a été supprimé par erreur au cours du temps. Pour la requête SQL (voir annexe 5.3.3), j'ai utilisé deux ensembles de données (dataset) différents, FIRESTORE_STAGING et FIRESTORE_CURRENT. Grâce au premier j'ai pu récupérer l'id et le type correspondants à la base de données Oracle. Le second m'a servi à récupérer le chemin Firestore afin de faire la jointure

entre deux tables nécessaires expliquées ci-dessous. Pour résoudre le problème, je devais récupérer les joueurs de la table *players* où la référence vers un membre manquait ou où la référence vers le membre spécifié n'existait pas dans la table *members.* La jointure des deux tables a pu être fait grâce au champ *member_ref* du côté de la table *players* et le chemin Firestore dans la table *members.* Le problème concernait 29.163 joueurs pour un total de 15.118.702 joueurs pour tous les matchs organisés jusqu'à présent.

3.2.4. Equipes d'un match

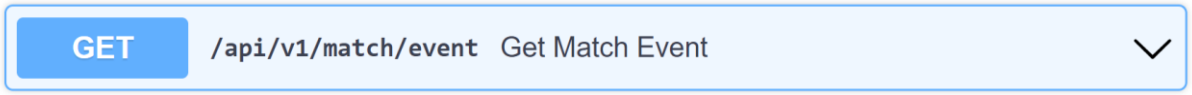


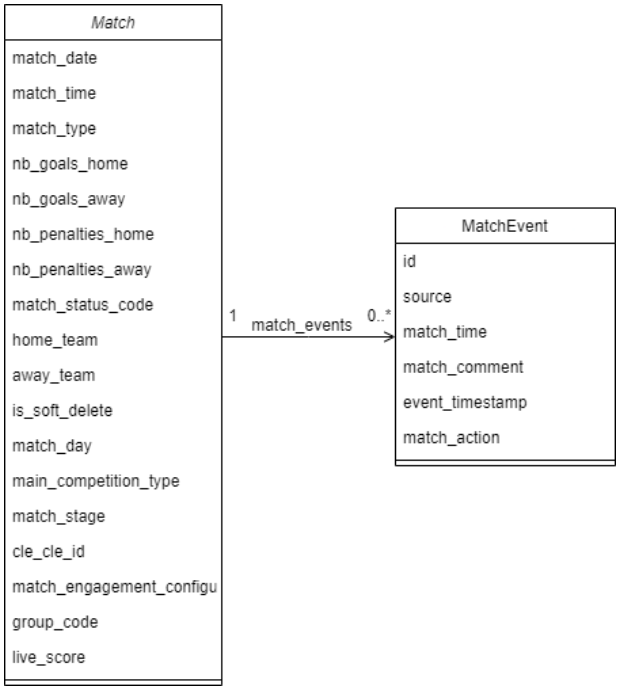


Un des bugs majeurs qui a joué sur la fiabilité de la fédération était que certains matchs étaient affichés dans le calendrier d'une équipe mais pas de l'autre. De ce fait, une équipe voyait le match mais l'autre pas ce qui créait une situation d'ambigüité. C'est pourquoi ce problème devait être réglé urgemment. J'ai constaté que lorsqu'un match ne s'affichait pas dans un calendrier c'était dû au fait que les métadonnées d'un match manquaient. C'est pourquoi j'ai écrit une requête qui contrôlait le champ

metadata.team_refs. Celui-ci doit contenir les références des deux équipes jouant, alors que lorsque le bug survenait une ou les références manquaient. La requête SQL (voir annexe 5.3.4) permettait donc de récupérer les matchs où les métadonnées des équipes ou d'une des deux équipes ne s'y trouvaient pas.

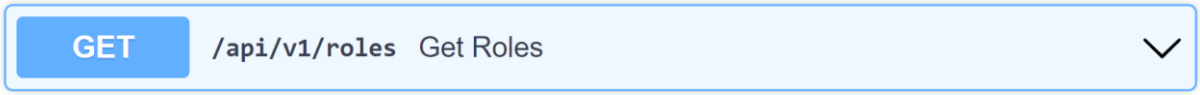
3.2.5. Evènements d'un match

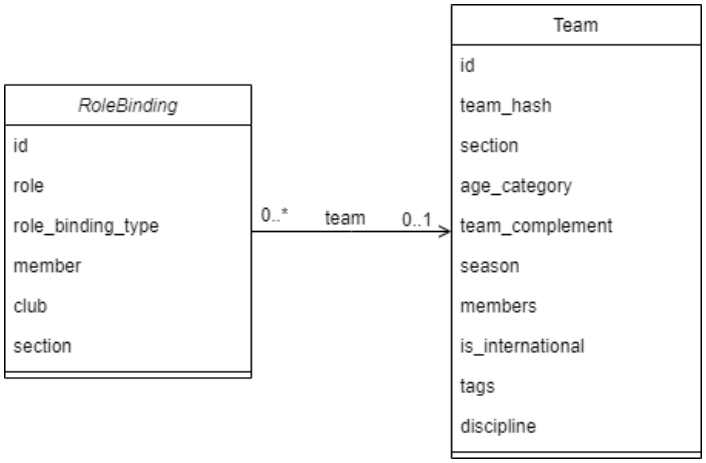




Le bug suivant rapporté concernait l'affichage manquant d'un joueur qui avait marqué. En observant la base de données, j'ai pu apercevoir que certains documents de la collection *match_events* n'avaient pas de référence vers un joueur, le champ *player.ref* n'existait donc pas. Pour cela, j'ai écrit une simple requête SQL (voir annexe 5.3.5) qui récupère tous les ids et types (pour pouvoir faire la liaison avec Oracle) des matchs concernés avec comme *event_type = « goal » et player.ref = null.* 7.591 match_events étaient concernés par ce bug pour un total de 2.577.022 match_events.

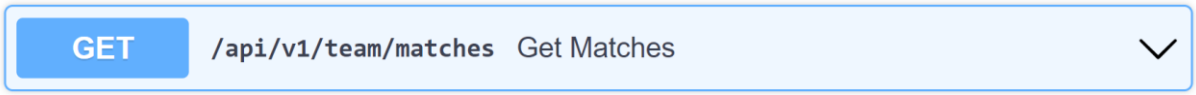
3.2.6. Rôles

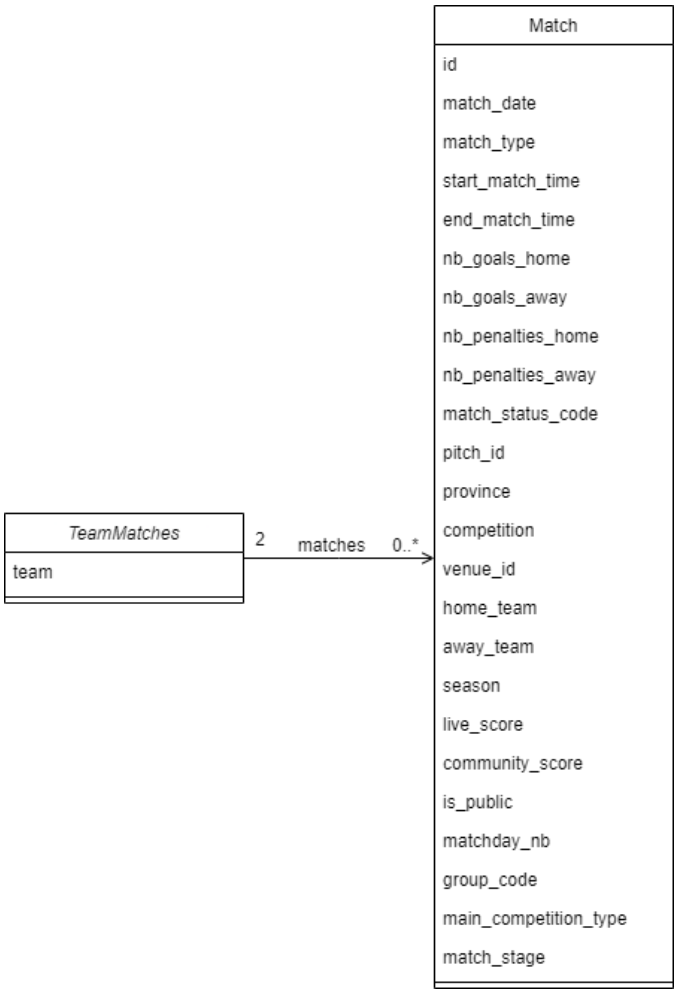




Certains rôles, comme le responsable d'équipe, ont logiquement besoin dans la base de données d'avoir une référence vers une équipe d'un club et d'autres rôles, comme secrétaire, n'ont au contraire pas besoin de cette référence. Cependant, il arrive que certains rôles qui ont besoin de cette référence n'en ont pas. Pour remédier à cela j'ai dû écrire une requête SQL (voir annexe 5.3.6) qui récupère les ids et types de FIRESTORE_STAGING.ROLE_BINDINGS nécessaires pour faire le lien avec Oracle. Ce type représente la table correspondante dans la base de données Oracle et l'id représente l'id correspondant dans cette table. Mais dans FIRESTORE_STAGING.ROLE_BINDINGS nous n'avons pas de champ pour le teamID. De ce fait, pour savoir si un rôle a besoin d'une référence je devais faire une jointure sur FIRESTORE_CURRENT.ROLE_BINDINGS qui possède un champ teamID grâce auquel je peux récupérer les rôles qui ont un teamID mais pas de référence vers une équipe dans FIRESTORE_STAGING.ROLE_BINDINGS.

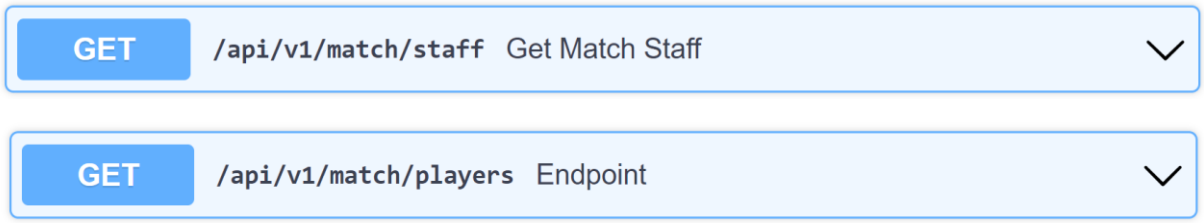
3.2.7. Matchs dupliqués

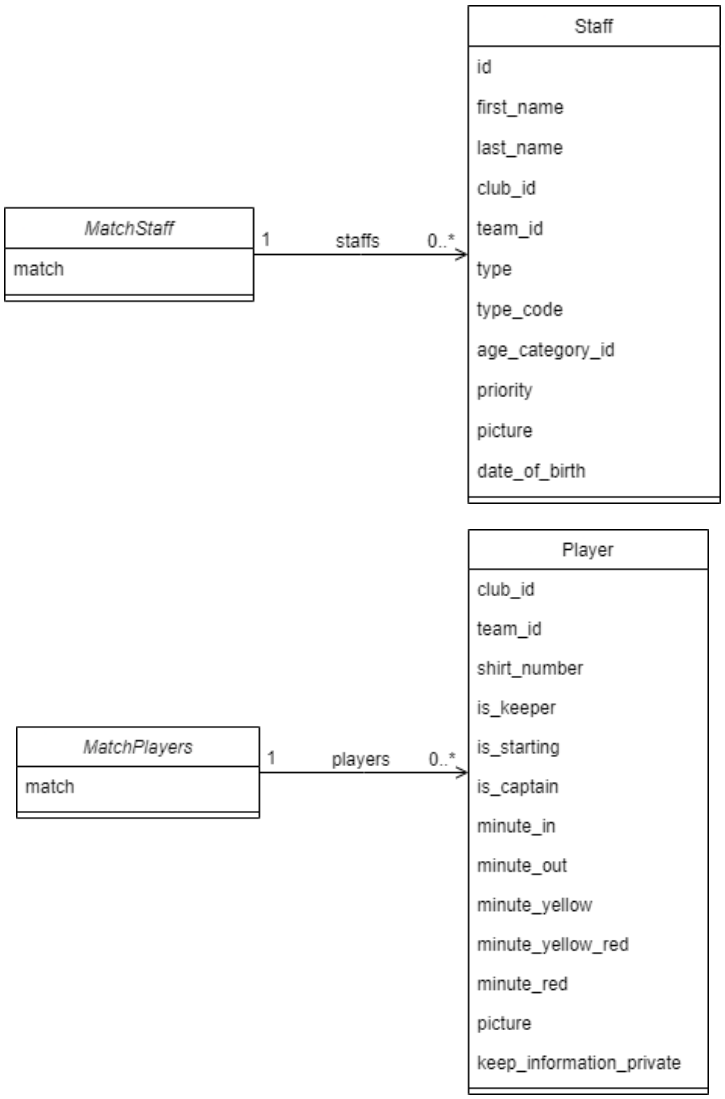




Un bug qui apparait souvent est les matchs qui apparaissent deux fois dans le calendrier. Ceci s'explique par le fait que ces matchs existent deux fois dans la base de données avec des ids différents car ces matchs ont été déclarés à deux moments différents. Tous ces matchs doivent être récupérés dans la table *matches* et celle-ci doit être nettoyée régulièrement. Pour la requête SQL (voir annexe 5.3.7), j'ai fait une jointure sur la même table en récupérant les matchs qui ont la même équipe qui joue à domicile (home_team.team_ref), la même équipe qui joue en déplacement (away_team.team_ref) et à la même date et heure (u_metadata.match_datetime). Au total, il y avait 6.022 matchs dupliqués au moment de la création de ce script.

3.2.8. Staff et joueurs dupliqués

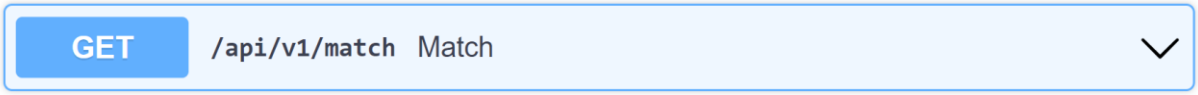


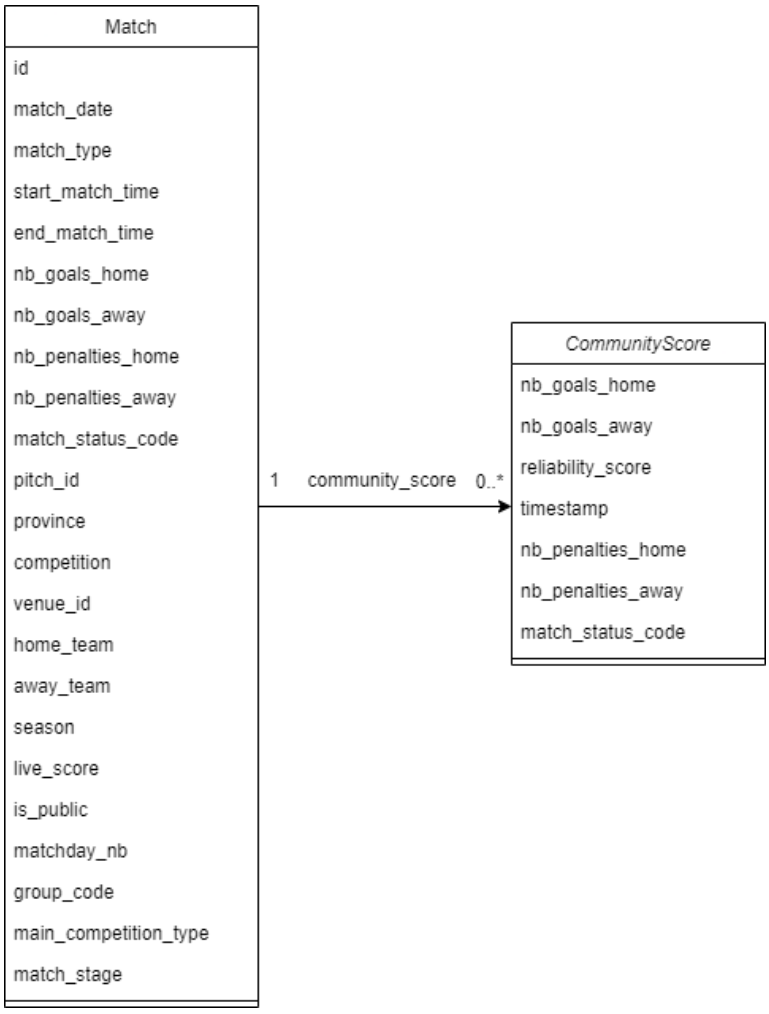


L'API permet de récupérer tous les joueurs et membres du staff pour un match donné. Ceux-ci sont affichés sur le site web et l'application. Cependant, il arrive que des joueurs ou membres du staff sont affichés plusieurs fois dans l'aperçu d'un match car dans les tables correspondantes des duplicatas existent pour un même match.

Pour la requête SQL (voir annexe 5.3.8), je joins le champ *member_ref* sur lui-même dans la même table *staff* en plus de récupérer l'id du match correspondant dans le chemin firestore grâce au champ *__key__.path*. Ceci me donne tous les duplicatas de membres de staff pour un seul match. Au total 47.956 membres du staff ont été dupliqués. J'ai ensuite fait la même requête sur la table *players* et celle-ci m'a donné 234.688 joueurs dupliqués.

3.2.9. Score ajouté par les supporteurs

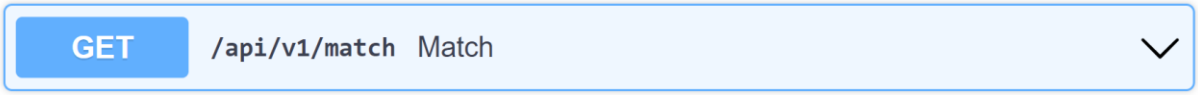




L'application permet d'avoir de l'interaction avec ses utilisateurs et ceux-ci ont la possibilité d'encoder le score d'un match auquel ils ont assisté. Cependant, le score

officiel communiqué par l'arbitre reste prioritaire sur le score des utilisateurs. Le problème qui arrive est que parfois le community score est affiché car le score officiel n'existe pas dans Firestore. Pour pouvoir mettre à jour les champs nb_goals_home et nb_goals_away, qui représentent le score officiel, j'ai écrit une requête SQL (voir annexe 5.3.9). Celle-ci permet de récupérer l'id et le nom de la table Oracle grâce aux champs DWH_OBJET_ID et DWH_OBJECT_TYPE qui permettent de recharger dans Firestore depuis Oracle les matchs problématiques. Pour cela, j'ai fait une jointure des tables *matches* et *community_score* sur le champ __key__.path où se trouve l'id du match afin de récupérer les matchs qui ont un *community_score* mais les champs *nb_goals_home* et *nb_goals_away* sont à null. De ce fait, 119 matchs ont pu être corrigés.

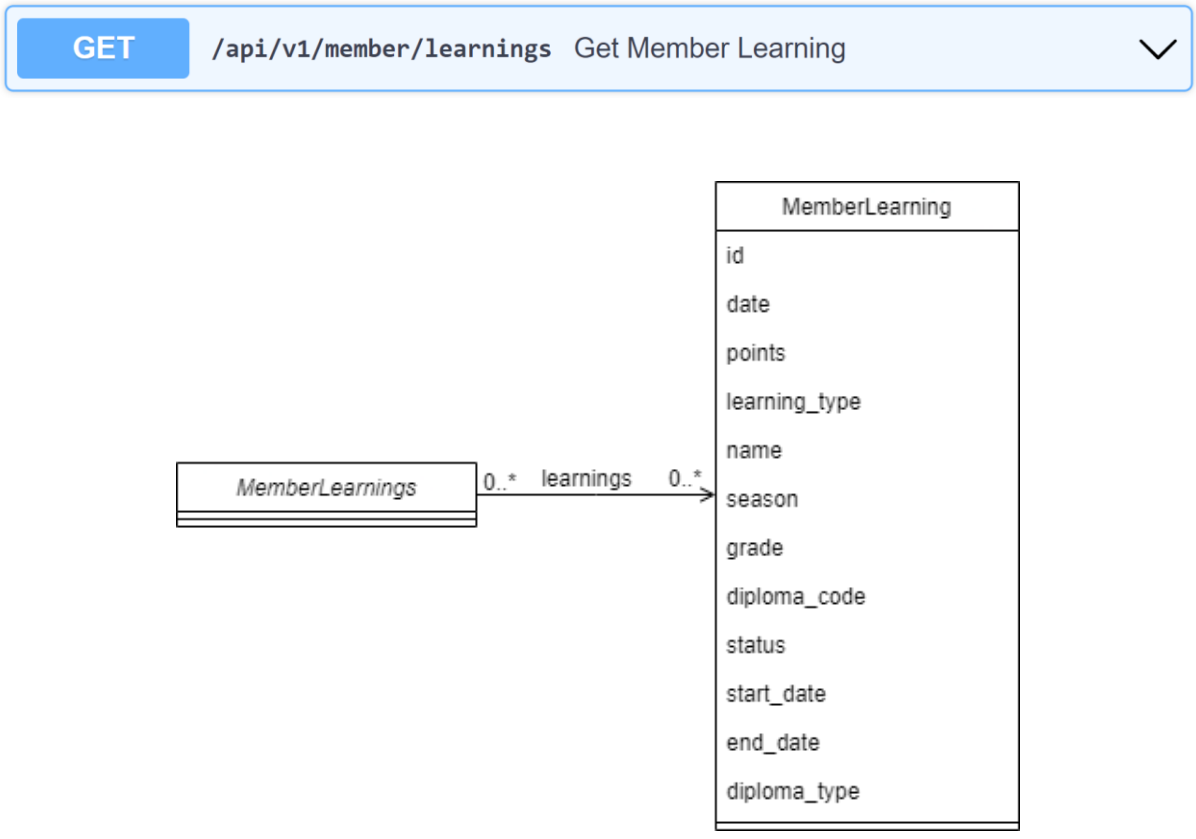
3.2.10. Matchs



Match
id
match_date
match_type
start_match_time
end_match_time
nb_goals_home
nb_goals_away
nb_penalties_home
nb_penalties_away
match_status_code
pitch_id
province
competition
venue_id
home_team
away_team
season
live_score
is_public
matchday_nb
group_code
main_competition_type
match_stage

Avant que je commence à travailler sur la tâche de validation de données, un employé a déjà créé un script pour les matchs où il manque la référence vers l'équipe à domicile (home_team) ou l'équipe à l'extérieure (away_team). Cependant, la requête SQL (voir annexe 5.3.10) ne renvoie plus de résultats car celle-ci fait une jointure sur la table *competition_teams*, qui elle-même ne contient pas toutes les équipes. De ce fait, il existait toujours des matchs où il manquait des références. Pour ne plus faire de jointure sur cette table, j'ai écrit une requête qui fait une jointure directement sur la copie de la table correspondante de Oracle sur Firestore qui elle contient toutes les équipes. Néanmoins, dans cette table, il existe toujours des équipes qui ne le devraient pas. Elles contiennent un champ *optbye* qui est égal à « Y ». Dans ma requête j'ai ajouté une condition qui récupère seulement les équipes où le champ *optbye* est égal à « N ». Grâce à cela, nous pouvons récupérer tous les matchs problématiques.

3.2.11. Doublons des évènements d'apprentissage



Quelques utilisateurs de l'application se sont plaints du fait que certains de leurs cours d'entraineur sont affichés plusieurs fois. Comme dans le cas des Matchs dupliqués,

ces cours apparaissent plusieurs fois dans la base de données avec des ids différents. Pour les récupérer et pouvoir nettoyer la table *learning_events* j'ai écrit une requête SQL (voir annexe 5.3.11) qui fait une jointure sur la même table et qui compare le nom du cours (*translations)*, le nombre de points gagnés (*nb_points)* à l'issu de celui-ci, la date à laquelle le cours a été donné (date) et où l'id de l'objet est différent.

3.2.12. Conclusion

Ce projet m'a demandé beaucoup de patience pour analyser les différents bugs et incohérences qui concernaient de nombreuses et grandes tables. Celui-ci m'a donc permis de manipuler un grand volume de données et totalement mettre en pratique mes cours de persistance de données suivies durant mes trois années d'études à l'Ecole Supérieure d'Informatique. Lors de l'écriture des requêtes SQL, j'ai dû effectuer un travail d'analyse conséquent afin d'arriver à un résultat optimal car parfois même si une requête réalisait ce qui était demandé elle n'était pas celle qui utilisait le moins de ressources. J'ai dû également prendre en compte les nombreux manques de données dans les différentes tables. Les scripts écrits servent et serviront dans le futur pour la partie de data validation sur laquelle les employés travaillent régulièrement afin d'avoir le moins de différences possibles entre Oracle et Firestore et que le contenu affiché sur l'application soit le plus correct possible.

3.3. RBFA Generic API

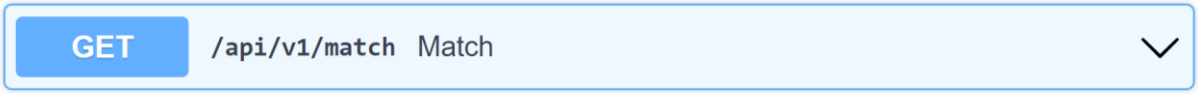
Le RBFA Generic API est, comme son nom l'indique, une API, une interface de programmation, qui est développée en FastAPI, framework web de haute performance en Python. Ce projet permet d'accéder aux données de manière structurée et optimisée grâce aux différents endpoints. Actuellement, le Generic API suit la Monolithic Architecture qui consiste à avoir un seul service centré et beaucoup d'endpoints dans un seul endroit. Cette méthode est efficace pour tester et déployer mais est compliquée à maintenir dû au fait que tout est mélangé, et est plus lente vu la taille de l'application. Le but de l'équipe IT est de passer à une architecture microservices, c'est-à-dire que toutes les fonctionnalités sont divisées en plus petits services, plus simple à gérer.

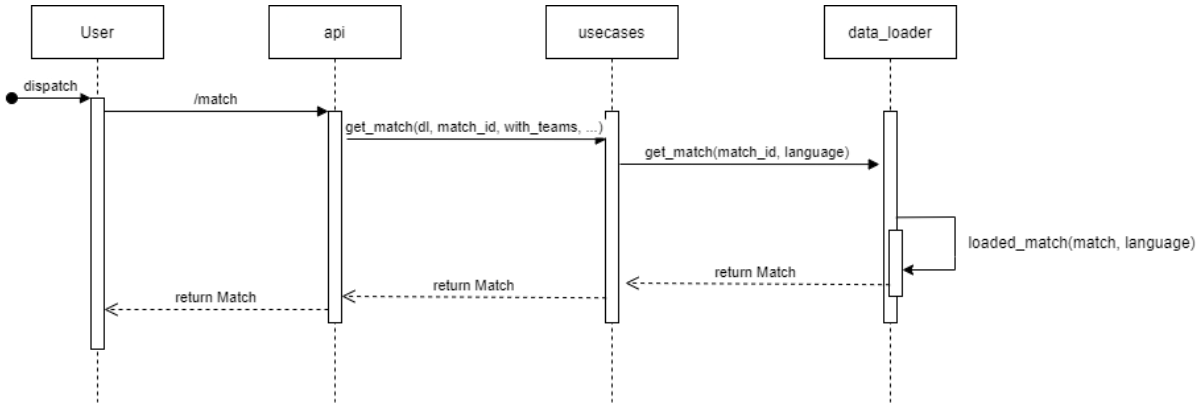
Pour le moment, le site web de la RBFA n'utilise pas le Generic API mais seul l'application mobile le fait. Le but est de faire progresser le Generic API afin de l'utiliser sur ces deux plateformes. Ceci améliorera également la cohérence des données affichées dans les deux applications où des divergences sont présentes.

De plus, des entreprises externes utilisent le Generic API pour développer leur propre application avec les données mises à disposition. Celui-ci doit donc répondre aux attentes de beaucoup. Un exemple d'endpoints est */clubs* qui renvoient tous les clubs et ses nombreux détails enregistrés au sein de la fédération.

J'ai eu l'occasion de travailler sur quelques optimisations que je vais présenter ci-dessous.

3.3.1. Equipes internationales

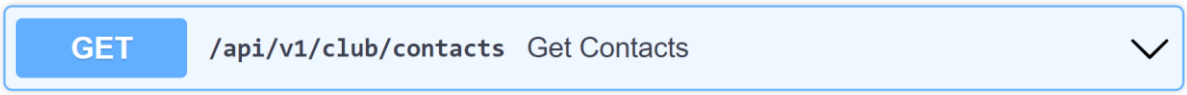


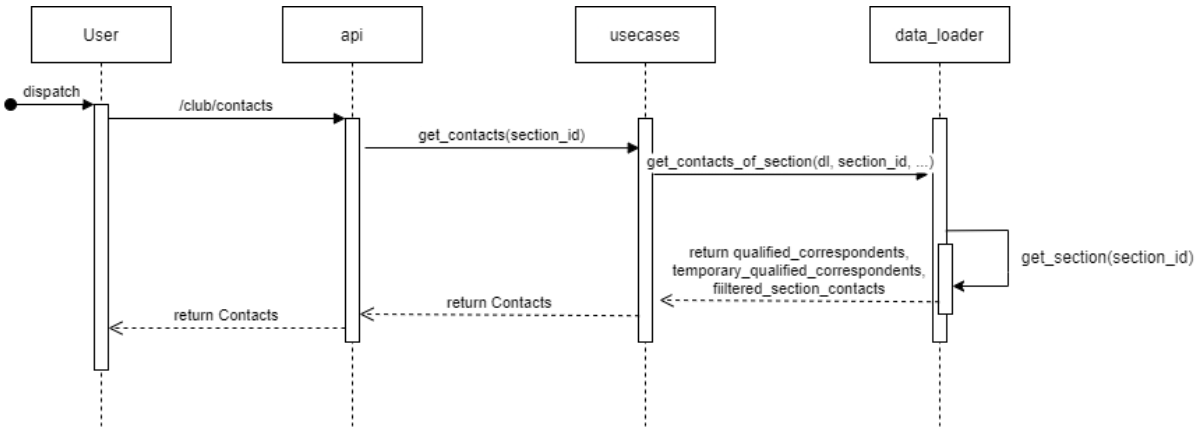


Jusqu'à présent, lorsqu'une équipe belge jouait contre une équipe étrangère l'API rendait une équipe vide pour l'équipe étrangère car celle-ci n'était pas connue dans la base de données. Une amélioration de l'API était donc nécessaire. J'ai analysé le code existant et ajouté 4 nouveaux champs pour l'entité Match afin d'y récupérer le nom de l'équipe et le complément (domicile et extérieure) à partir d'Oracle. De plus, avant de rendre l'équipe, elle doit être parsée en un objet *Team.* Maintenant, lors du parse, si une équipe n'est pas connue, la méthode rend seulement le nom de l'équipe et le complément. Ces champs sont automatiquement récupérés grâce au projet RBFA

Data Pipeline où l'ingestion de ces labels d'Oracle était déjà effectuée. A partir de maintenant, le frontend peut afficher les équipes correctement et l'application ne montre plus « équipe inconnue » pour une équipe étrangère.

3.3.2. Confidentialité d'un contact

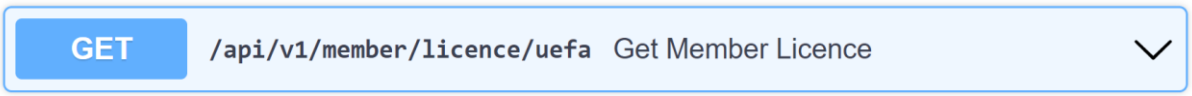


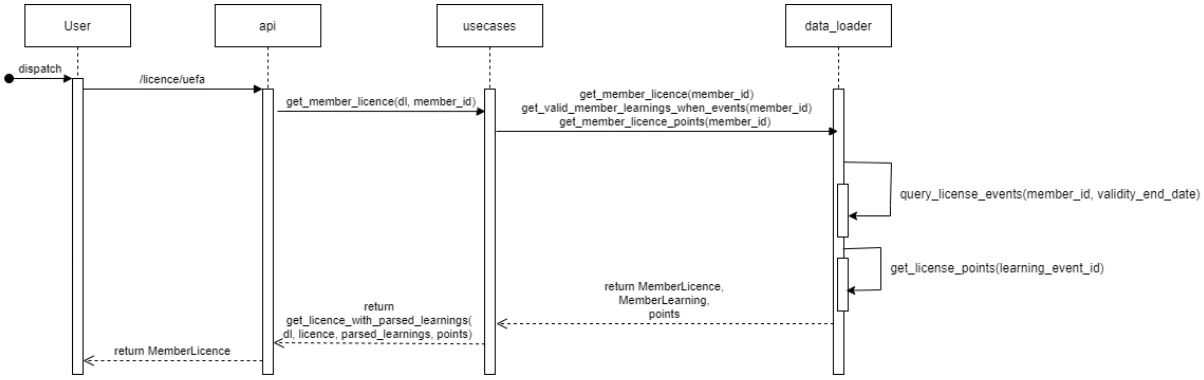


Jusqu'à présent, pour le endpoint club/contacts l'API rendait tous les contacts (numéros de téléphone, adresses mail) des responsables d'un club même si les personnes concernées ne le souhaitaient pas, c'est-à-dire que le champ *optpublic* dans Oracle vaut « N ». Dans l'application, des contacts privés étaient donc accessibles à tous les utilisateurs. Une amélioration de l'API était urgemment nécessaire. Dans un premier temps, mes responsables souhaitaient qu'un champ « is_public » soit ajouté dans l'API et le frontend afficherait ou non les contacts en fonction de celui-ci. C'est ce que j'ai fait mais, après réflexion, une meilleure solution a été trouvée. Etant donné que l'API est également utilisée par des clients externes, seuls les contacts publics sont renvoyés. De ce fait, ils n'ont pas accès aux contacts privés.

Pour cela, j'ai ajouté un champ is_public dans l'entité *Contact* qui correspond au champ *optpublic* d'Oracle, migré dans le fichier loader du projet. Avant de retourner un contact, une vérification de ce champ est maintenant effectuée.

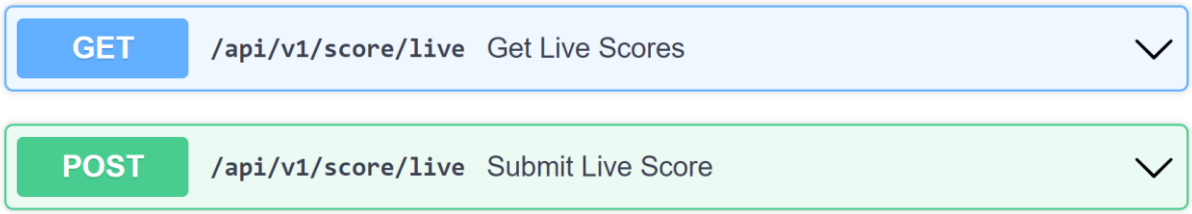
3.3.3. Calcul des points de licence

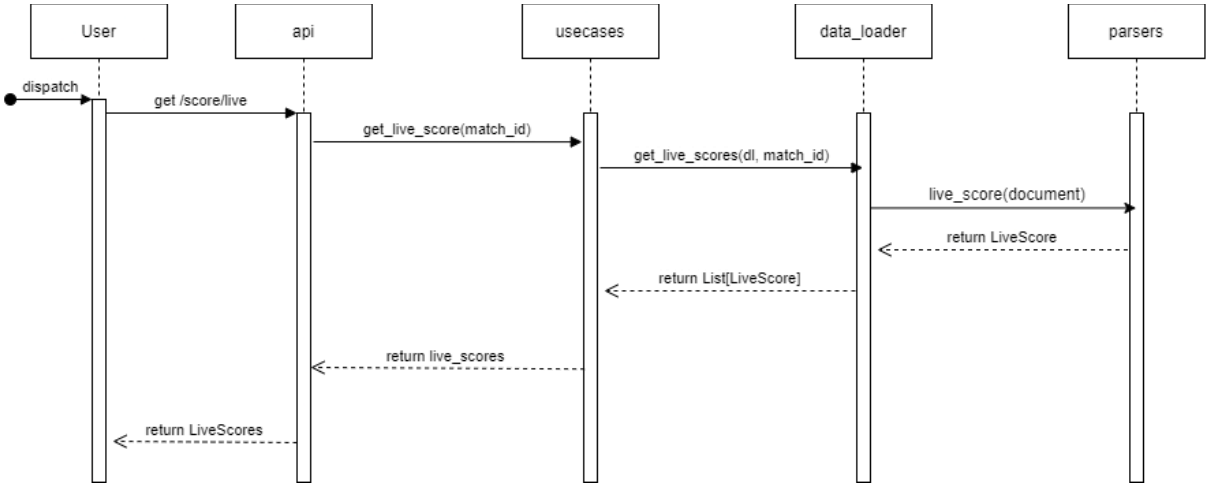


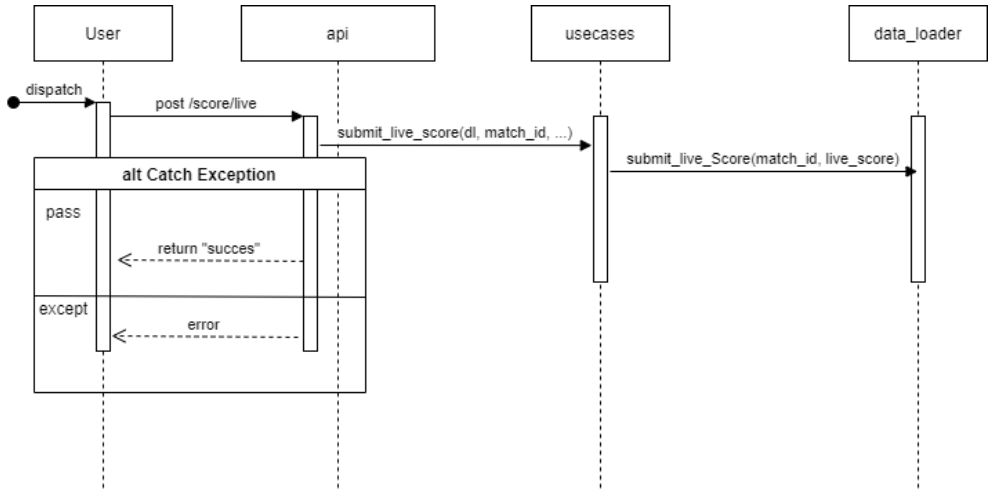


Le département business a décidé de changer la méthode de calcul des points de licence obtenue par un entraineur diplômé. J'ai donc dû modifier l'endpoint */licence/uefa* qui permet de récupérer la licence d'un entraineur et les cours suivis pour cette licence. Dans un premier temps, j'ai listé tous les cours réussis d'un entraineur durant toutes ses licences avec une date de validité plus grande qu'aujourd'hui (validity_end_date > datetime.today()). Ensuite, j'ai uniquement gardé les cours d'une licence réussie. Après j'ai contrôlé pour chaque évènement d'un cours s'il a bien eu lieu après le début de validité du cours (learnings[0].validity_start_date < start_date). Si c'est bien le cas, ce cours est ajouté à la liste des cours réussis pour cette licence et le nombre de points est ajouté aux points acquis pour celle-ci. De plus, pour l'entité MemberLicence j'ai ajouté un champ avec le nombre de points obtenus pour une licence afin que le frontend ne doivent plus faire le calcul de son côté en récupérant les points des cours retournés.

3.3.4. Score en temps réel



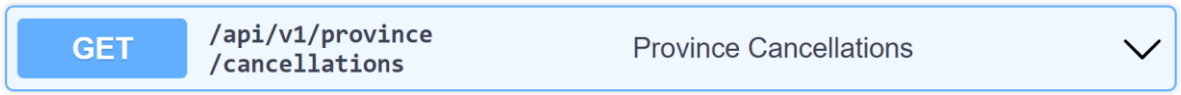


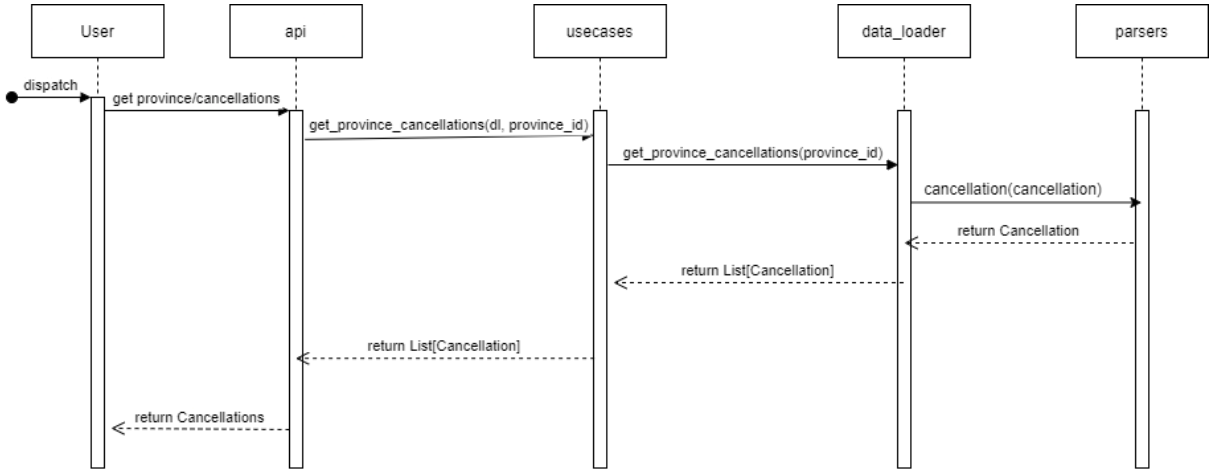


Dans l'application mobile, pour certains matchs nous avons la possibilité de suivre le score en direct. Cependant, un bug a été rapporté pour le match international Irlande – Belgique où le score était de 0-1 mais l'application affichait toujours 0-0. Après vérification des méthodes écrites pour la soumission d'un score en direct je n'ai pas

trouvé d'incohérence pour expliquer le bug survenu à ce moment précis. Actuellement, le champ live_score est correct et tout est rentré dans l'ordre depuis.

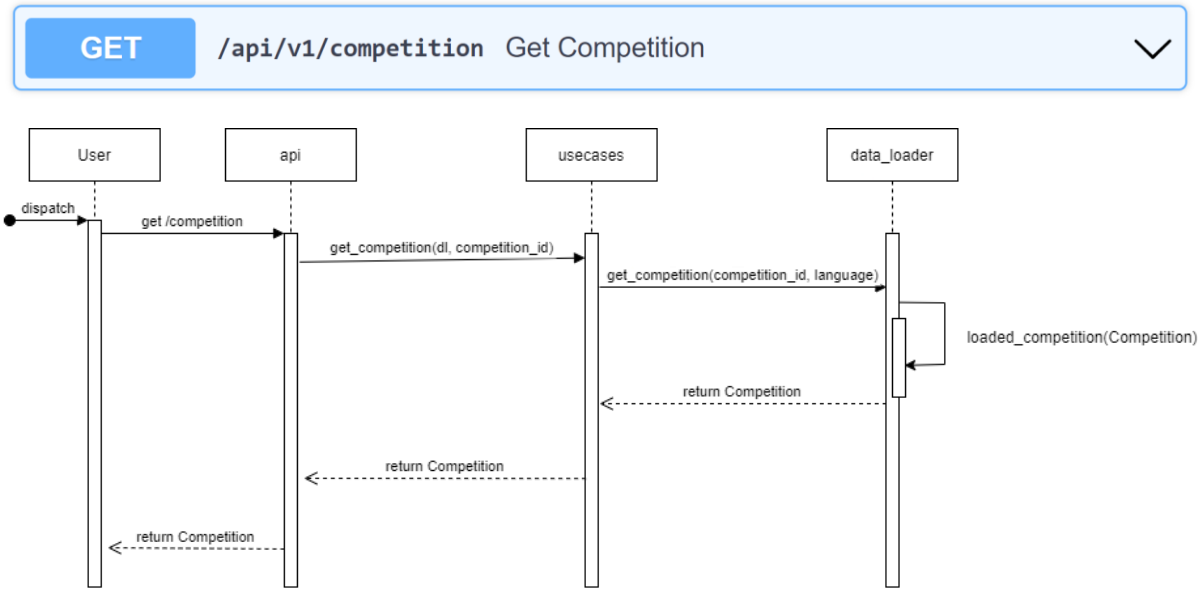
3.3.5. Remises





Cet endpoint affiche toutes les remises publiées pour une province donnée. Certaines provinces avaient une description aussi bien en français qu'en néerlandais avec deux champs distincts retournés, label_fr et label_nl. Au lieu de retourner la description en deux langues, j'ai ajouté un paramètre à cet endpoint qui permet de choisir une des 4 langues à disposition (français, néerlandais, anglais et allemand). Dorénavant, l'endpoint doit être appelé avec une langue pour avoir une description de la remise. Pour faire cela, j'ai remplacé les deux champs label_fr et label_nl par un seul champ description. Cependant, dans la base de données, nous n'avons pas de remises avec un label en anglais ou allemand. De ce fait, lorsque l'endpoint est appelé avec une de ces langues la remise est rendue avec tous les champs retournés sauf la description.

3.3.6. Compétitions



L'endpoint ci-dessus retourne simplement une compétition avec son nom, saison etc. Mais elle ne retourne pas quelle fédération organise cette compétition. Pour cela, j'ai dû faire l'ingestion du champ *federation_code* depuis la base de données Firestore afin que celui-ci puisse être utilisé dans le futur par le site web et afficher la fédération organisatrice d'une compétition. Pour cela, j'ai ajouté un nouveau champ federation_code au model et à l'entité Competition. Dorénavant, lorsqu'une compétition est parse, celle-ci est retourné avec la fédération si celle-ci existe dans la base de données.

3.4. RBFA Terraform

Le projet RBFA Terraform a été créé dans le but d'automatiser la gestion de l'infrastructure cloud et des ressources. L'infrastructure-as-code (IaC) est un outil de codage déclaratif et est une réelle plus-value pour les développeurs car elle permet de créer et mettre à jour un service facilement et de visualiser son serveur en toute simplicité grâce au code. Tout ceci est également géré via git, c'est-à-dire que si un développeur a besoin d'une ressource spécifique sur Google Cloud Platform, par exemple la permission de lecture sur Firestore, il écrit dans le fichier de configuration

.tf en HashiCorp Configuration Language sa demande, crée un pull request sur GitHub et une fois que le responsable a validé sa demande, le merge est fait et il ne reste plus qu'à relancer l'application sur le site de Terraform pour avoir accès à la ressource. Le responsable n'a donc plus besoin de naviguer manuellement sur Google Cloud Platform pour donner des ressources spécifiques à ses développeurs. De même pour la destruction de ressources, il suffit de supprimer le code relatif à la ressource donnée et la ressource ne sera plus disponible. C'est donc un réel gain de temps. Dans mon cas, j'ai utilisé Terraform afin de demander des permissions sur trois projets différents. Pour le projet 5.5.1RBFA-DATALAKE où le service BigQuery est disponible afin de manipuler la base de données Firestore, j'ai demandé les permissions de dataViewer et jobUser. Ensuite, pour le projet RBFA-DP-OLTP j'avais droit aux permissions firebase.viewer et datastore.viewer. Je pouvais donc seulement consulter et ne pas modifier ou supprimer des données dans la base de données. Enfin, pour le projet RBFA-DATA-PIPELINE j'avais les même droits que pour RBFA-DATALAKE sauf que cette fois-ci le projet contenait la copie de la base de données Oracle et non les données utilisés par le Generic API et permettait donc d'effectuer les requêtes SQL sur cette base de données afin de comparer Oracle et Firestore grâce à BigQuery.

4. Conclusion

Le stage en tant que Data Engineer était ma première expérience professionnelle dans le monde l'informatique et celui-ci m'a apporté de nombreuses nouvelles connaissances et savoir-faire. Grâce à celui-ci j'ai pu recueillir, nettoyer, traiter et rendre disponible un large volume de données à travers les différents projets détaillés précédemment.

Dans un premier temps, l'initiation à la plateforme Google Cloud Platform via le projet RBFA Workshop Sandboxes m'a permis de maitriser l'outil de Google afin de me lancer dans les projets principaux qui utilisent grandement cette plateforme. Ce projet était un bon moyen de m'intégrer au sein de l'équipe et de la fédération sans pression.

Ensuite, dans le projet RBFA Data Pipeline, j'ai pu assainir les données de la RBFA et me rendre compte de l'importance de la justesse des données. La qualité des données a un grand impact sur l'image que le monde extérieur a sur une entreprise. L'application avec de nombreuses incohérences ne donne pas envie de l'utiliser et cela se ressent dans l'avis des utilisateurs. Ce projet m'a aussi permis d'appliquer les concepts vu en cours de persistance de données sur de larges et volumineuses tables. Nous avons pu remarquer que lorsqu'il y a un flux de données important, ces données sont plus difficilement maitrisables et la gestion de ceci nécessite une bonne connaissance des outils utilisés et une analyse approfondie de l'architecture est requise. Tout cela dans le but de produire des données facilement utilisables et manipulables par la suite. Une structure pointilleuse est nécessaire afin de mener à bien les différentes tâches de traitement de données.

Mais encore, le Generic API m'a donné la possibilité de gérer un grand flux de données à travers un code en Python et de passer de documents NoSQL à des classes. J'ai appris à manier un projet d'une envergure conséquente qui demande une structure et organisation précise afin de maintenir la qualité au fur et à mesure de la quantité de code qui augmente. Ce projet m'a permis d'optimiser les données d'une telle manière que ceux-ci soient exploitables par aussi bien des clients internes qu'externes. Seules les données nécessaires et demandées pouvaient être mise à disposition. De plus, elles devaient être structurées de sorte que les clients puissent utiliser l'API aisément.

Le projet RBFA Terraform m'a permis de découvrir et d'apprendre un nouveau concept de gestion de projet, l'Infrastructure as Code. Celui-ci permet d'automatiser l'infrastructure et facilité sa gestion grâce au code déclaratif. Grâce à ce projet j'ai pu demander facilement des ressources nécessaires aux bon développement de mon stage et, par la même occasion, apprendre une nouvelle manière de gérer ses projets. La compréhension de solutions modulables était un vrai plus durant ce stage.

Enfin, ce stage m'a permis de découvrir pleinement le monde professionnel dans une grande entreprise qui a beaucoup de responsabilités, de travailler en équipe, d'apprendre l'organisation et la gestion de projets. De plus, j'ai pu développer mes compétences de programmation dans un environnement unique et mettre en pratique la méthode de travail Scrum qui est, comme j'ai pu l'expérimenter, nécessaire au bon fonctionnement d'un projet d'une telle taille.

5. Annexes

5.1.

5.2. RBFA Workshop Sandboxes

5.2.1. Insert

```python
@app.post("/insert")
async def insert_row_to_bigquery():
    client = bigquery.Client()
    table_id = "rbfa-workshop-sandboxes.marika.currenttime"
    ct = datetime.datetime.now().strftime("%m/%d/%Y, %H:%M:%S")

    rows_to_insert = [
        {u"name": u"Marika", u"currenttime": ct},
    ]

    errors = client.insert_rows_json(table_id, rows_to_insert)

    if errors == []:
        print("New row has been added.")
    else:
        print("Encountered errors while inserting row: {}".format(errors))
```

5.2.2. Upload

```python
def create_bucket_class_location(bucket_name):
    storage_client = storage.Client()
    bucket = storage_client.bucket(bucket_name)
    bucket.storage_class = "STANDARD"
    new_bucket = storage_client.create_bucket(bucket, location="eu")

    print(
        "Created bucket {} in {} with storage class {}".format(
            new_bucket.name, new_bucket.location, new_bucket.storage_class
        )
    )

    return new_bucket


def create_file_with_current_time():
    current_time = datetime.datetime.now().strftime("%m/%d/%Y, %H:%M:%S").encode()
    f = open("current_time.txt", "wb")
    f.write(current_time)
    f.close()

    return f


@app.post("/upload/{bucket_name}")
async def upload_file_cs_bucket(bucket_name: str):
    file = create_file_with_current_time()
    filename = file.name
    source_file_name = os.path.abspath(filename)

    storage_client = storage.Client()
    check_bucket = storage_client.bucket(bucket_name)
    if not check_bucket.exists():
        bucket_name = create_bucket_class_location(bucket_name)
    bucket = storage_client.get_bucket(bucket_name)
    destination_blob_name = filename.split(".")[0]
    blob = bucket.blob(destination_blob_name)

    blob.upload_from_filename(source_file_name)

    print(
        "File {} uploaded to {}.".format(
            source_file_name, destination_blob_name
        )
    )
```

5.2.3. Update

```python
@app.put("/update/{key_id}")
async def update_entity(key_id: int):
    current_time = datetime.datetime.now()

    client = datastore.Client()

    with client.transaction():
        key = client.key("TestState", key_id)
        entity = client.get(key)
        entity["timestamp"] = current_time
        client.put(entity)
```

5.2.4. Cron Job

```python
@app.post("/cron-job")
async def create_scheduler_job():
    client = scheduler.CloudSchedulerClient()
    parent = f"projects/rbfa-workshop-sandboxes/locations/europe-west1"

    job = {
        "name": "projects/rbfa-workshop-sandboxes/locations/europe-west1/jobs/marika-cron-job",
        "app_engine_http_target": {
            "app_engine_routing": {"service": "svc-marika-test2"},
            "relative_uri": "/insert",
            "http_method": scheduler.HttpMethod.POST,
        },
        "schedule": "*/5 * * * *",
        "time_zone": "Europe/Brussels",
    }

    response = client.create_job(request={"parent": parent, "job": job})
    print("Created job: {}".format(response.name))
```

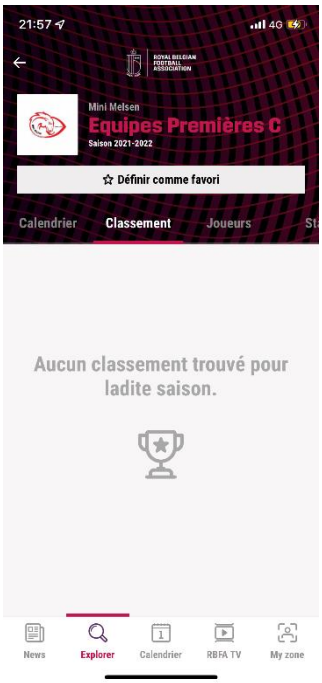
5.3. RBFA Data Pipeline

5.3.1. Classement d'une compétition

```python
ENVIRONMENT = os.getenv("ENVIRONMENT", "dev")
RUN_ANALYSIS_ONLY = os.getenv("RUN_ANALYSIS_ONLY", True)

def fix_competitions(env='dev', test=True):
    query = f"""SELECT [Confidentiel] .COMPETITIONS`
                WHERE
                (SELEC
    create_concept_children_query(query, env=env, test=test)


def run_data_validations_competitions():
    fix_competitions(ENVIRONMENT, test=RUN_ANALYSIS_ONLY)
```



Classement manquant

5.3.2. Membres d'une équipe

```
SELECT DISTINCT t.DWH_OBJECT_ID, t.DWH_OBJECT_TYPE FROM `Confidentiel
dev.FIRESTORE_STAGING.TEAMS` t JOIN Confidentiel.FIRESTORE_STAGING.PLAYER_STATS` p ON
t.eqsec_id = p.eqsec_eqsec_id JOIN `FIRESTORE_STAGING.MEMBERS` m ON
(split(p.__key__.path, ','))[safe_ordinal(2)] = (split(m.__key__.path, ','))[safe_ordinal(2)]
WHERE CONCAT(SUBSTR((split(m.__key__.path, ','))[safe_ordinal(1)], 2,
LENGTH((split(m.__key__.path, ','))[safe_ordinal(1)])-2), '/', SUBSTR((split(m.__key__.path, ','))
[safe_ordinal(2)], 3, LENGTH((split(m.__key__.path, ','))[safe_ordinal(2)])-3)) NOT IN
UNNEST(t.member_refs)
```

5.3.3. Joueur membre

```
SELECT DWH_OBJECT_ID, DWH_OBJECT_TYPE FROM Confidentiel FIRESTORE_STAGING.PLAYERS` WHERE
DWH_OBJECT_TYPE = 'EKO_SELECTIONJOUEUR' AND (MEMBERSHIP.MEMBER_REF IS NULL OR MEMBERSHIP.MEMBER_REF NOT IN
(SELECT C.DWH_OBJECT_PATH FROM Confidentiel.FIRESTORE_CURRENT.MEMBERS` C JOIN `Confidentiel
Confidentiel FIRESTORE_STAGING.MEMBERS` S ON C.ORACLE_DWH_OBJECT_ID = S.DWH_OBJECT_ID))
```

5.3.4. Equipes d'un match

```
SELECT DWH_OBJECT_ID, DWH_OBJECT_TYPE FROM `Confidentiel FIRESTORE_STAGING.MATCHES` WHERE
(((u_metadata.team_refs)[SAFE_OFFSET(0)] IS NULL) OR ((u_metadata.team_refs)[SAFE_OFFSET(1)]) IS NULL) AND
home_team.team_ref IS NOT NULL AND away_team.team_ref IS NOT NULL
```

Match existant

Match inexistant

5.3.5. Evènements d'un match

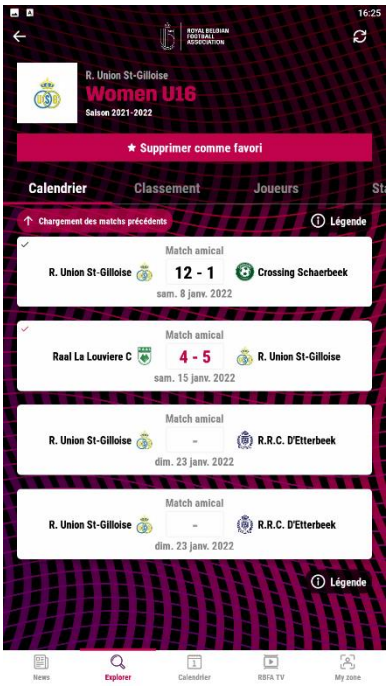
```
SELECT DWH_OBJECT_ID, DWH_OBJECT_TYPE FROM `Confidentiel.FIRESTORE_STAGING.MATCH_EVENTS` WHERE
event_type = "goal" AND player.ref IS NULL
```

5.3.6. Rôles

```
SELECT S.DWH_OBJECT_ID, S.DWH_OBJECT_TYPE FROM `Confidentiel.FIRESTORE_STAGING.ROLE_BINDINGS` S JOIN
Confidentiel.FIRESTORE_CURRENT.ROLE_BINDINGS` C ON S.DWH_OBJECT_ID = C.ORACLE_DWH_OBJECT_ID WHERE
C.TEAM.REF IS NULL AND C.TEAM_ID IS NOT NULL
```

5.3.7. Matchs dupliqués

```
SELECT t1.DWH_OBJECT_ID, t1.DWH_OBJECT_TYPE, t2.DWH_OBJECT_ID, t2.DWH_OBJECT_TYPE FROM `rbfa-
Confidentiel.FIRESTORE_STAGING.MATCHES` t1, `Confidentiel.FIRESTORE_STAGING.MATCHES` t2
WHERE t1.DWH_OBJECT_ID != t2.DWH_OBJECT_ID AND t1.home_team.team_ref = t2.home_team.team_ref AND
t1.away_team.team_ref = t2.away_team.team_ref AND t1.u_metadata.match_datetime =
t2.u_metadata.match_datetime
```

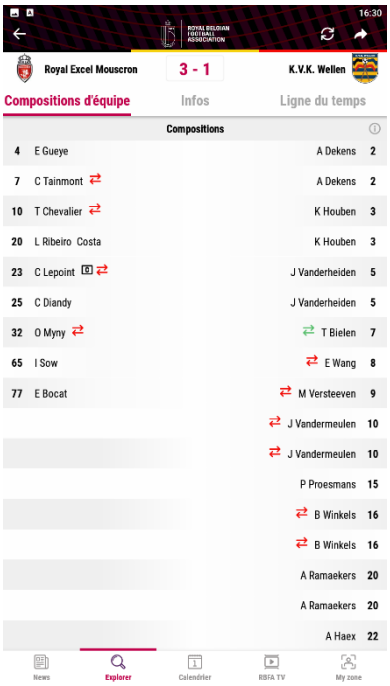


Même match en double

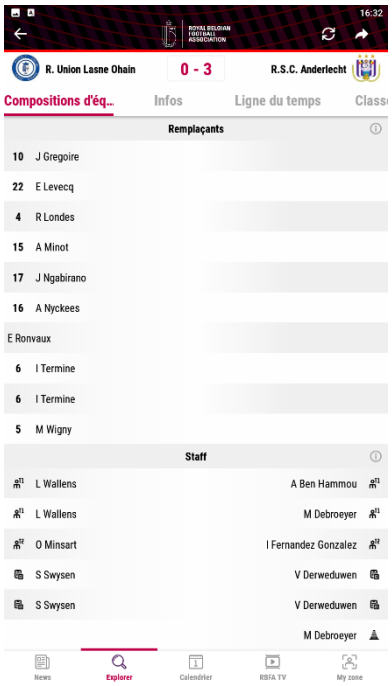
5.3.8. Staff et joueurs dupliqués

```
SELECT t1.DWH_OBJECT_ID, t1.DWH_OBJECT_TYPE, t2.DWH_OBJECT_ID, t2.DWH_OBJECT_TYPE FROM `rbfa-
Confidentiel
FIRESTORE_STAGING.STAFF` t1, `
Confidentiel
.FIRESTORE_STAGING.STAFF` t2 WHERE
t1.DWH_OBJECT_ID != t2.DWH_OBJECT_ID AND t1.membership.member_ref = t2.membership.member_ref AND
(split(t1.__key__.path, ','))[safe_ordinal(2)] = (split(t2.__key__.path, ','))[safe_ordinal(2)]
```

```
SELECT t1.DWH_OBJECT_ID, t1.DWH_OBJECT_TYPE, t2.DWH_OBJECT_ID, t2.DWH_OBJECT_TYPE FROM `rbfa-
Confidentiel
FIRESTORE_STAGING.PLAYERS` t1, `r
Confidentiel
FIRESTORE_STAGING.PLAYERS` t2
WHERE t1.DWH_OBJECT_ID != t2.DWH_OBJECT_ID AND t1.membership.member_ref = t2.membership.member_ref
AND (split(t1.__key__.path, ','))[safe_ordinal(2)] = (split(t2.__key__.path, ','))
[safe_ordinal(2)]
```



Plusieurs joueurs en double



Plusieurs membres du staff en double

5.3.9. Score ajouté par les supporteurs

```
SELECT M.DWH_OBJECT_ID, M.DWH_OBJECT_TYPE FROM `Confidentiel.FIRESTORE_STAGING.MATCHES` M
JOIN `Confidentiel.FIRESTORE_STAGING.COMMUNITY_SCORE` C ON (split(C.__key__.path, ','))
[safe_ordinal(2)] = (split(M.__key__.path, ','))[safe_ordinal(2)] WHERE M.nb_goals_home IS NULL OR
M.nb_goals_away IS NULL
```

5.3.10. Matchs

```
SELECT M.DWH_OBJECT_ID, M.DWH_OBJECT_TYPE FROM `Confidentiel.FIRESTORE_STAGING.MATCHES` M
JOIN `Confidentiel.ORACLE_CURRENT.EQUIPE` E ON E.EQU_ID = M.EQU_EQU_ID WHERE
M.HOME_TEAM.TEAM_REF IS NULL AND M.EQU_EQU_ID IS NOT NULL AND E.OPTBYE = "N" AND E.EQSEC_EQSEC_ID
IS NOT NULL UNION ALL SELECT M.DWH_OBJECT_ID, M.DWH_OBJECT_TYPE FROM Confidentiel
Confidentiel FIRESTORE_STAGING.MATCHES` M JOIN `Confidentiel.ORACLE_CURRENT.EQUIPE` E ON
E.EQU_ID = M.EQU_EQU_ID_AWAY WHERE M.AWAY_TEAM.TEAM_REF IS NULL AND M.EQU_EQU_ID_AWAY IS NOT NULL
AND E.OPTBYE = "N" AND E.EQSEC_EQSEC_ID IS NOT NULL
```

5.3.11. Doublons des évènements d'apprentissage

```
SELECT t1.DWH_OBJECT_ID, t2.DWH_OBJECT_ID FROM Confidentiel
Confidentiel.LEARNING_EVENTS` t1 INNER JOIN Confidentiel
.LEARNING_EVENTS` t2 on t1.translations.F = t2.translations.F AND t1.date =
t2.date AND t1.nb_points = t2.nb_points WHERE t1.DWH_OBJECT_ID < t2.DWH_OBJECT_ID
```

5.4. RBFA Generic API

5.4.1. Equipes internationales


```
{
  "id": "********",
  "match_engagement_configuration": {
    "score_prediction_enabled": false,
    "man_of_the_match_enabled": false
  },
  "match_date": "2019-07-27",
  "match_type": "friendly",
  "start_match_time": "10:30:00",
  "end_match_time": "12:15:00",
  "pitch_id": "********",
  "venue_id": "********",
  "province": {
    "id": "********",
    "name": "Hainaut",
    "language_regime_code": "FR",
    "federation_code": "ACFF",
    "sportive_province_code": 5
  },
  "season": 2020,
  "home_team": {
    "id": "********",
    "team_hash": "********",
    "club_id": "********",
    "club_name": "Sporting Confidentiel ",
    "p Confidentiel
    "age_category": {
      "name": "U21",
      "short_name": "-21",
      "sorting_position": 19,
      "code": "U21",
      "gender": "M"
    },
    "season": 2020
  },
  "is_public": false
}
```


Avant – away_team inexistant

```
{
  "id": "********",
  "match_engagement_configuration": {
    "score_prediction_enabled": false,
    "man_of_the_match_enabled": false
  },
  "match_date": "2019-07-27",
  "match_type": "friendly",
  "start_match_time": "10:30:00",
  "end_match_time": "12:15:00",
  "pitch_id": "********",
  "venue_id": "********",
  "province": {
    "id": "********",
    "name": "Hainaut",
    "language_regime_code": "FR",
    "federation_code": "ACFF",
    "sportive_province_code": 5
  },
  "season": 2020,
  "home_team": {
    "id": "********",
    "team_hash": "********",
    "club_id": "********",
    "club_name": "Sporting Confidentiel ',
    "p Confidentiel
    "age_category": {
      "name": "U21",
      "short_name": "-21",
      "sorting_position": 19,
      "code": "U21",
      "gender": "M"
    },
    "season": 2020
  },
  "away_team": {
    "club_name": "Valenciennes",
    "team_complement": "B"
  },
  "is_public": false
}
```


Après – away_team existant

5.4.2. Confidentialité d'un contact


```
{
  "club": "********",
  "contacts": [
    {
      "first_name": "M
      "last_name": "Hu
      "gender": "M",
      "role_code": "Q",
      "role": "Authorized Correspondent",
      "contact_methods": [
        {
          "value": "04**/******",
          "type_code": "TEL",
          "type": "Phone"
        },
        {
          "value": "********@hotmail.com",
          "type_code": "EMAIL",
          "type": "Email"
        },
        {
          "value": "04**/******",
          "type_code": "GSM",
          "type": "Gsm"
        }
      ]
    },
    {
      "first_name": "P
      "last_name": "Le
      "gender": "M",
      "role_code": "COMITE",
      "role": "Committee member",
      "contact_methods": [
        {
          "value": "04**/******",
          "type_code": "GSM",
          "type": "Gsm"
        },
        {
          "value": "********@hotmail.com",
          "type_code": "EMAIL",
          "type": "Email"
        }
      ]
    },
```


Confidentiel

Confidentiel

Avant 1 – Tous les contacts retournés

```
    {
      "first_name": "R
      "last_name": "Kr
      "gender": "M",
      "role_code": "COMITE",
      "role": "Committee member",
      "contact_methods": [
        {
          "value": "04**/******",
          "type_code": "TEL",
          "type": "Phone"
        },
        {
          "value": "info@******.be",
          "type_code": "EMAIL",
          "type": "Email"
        }
      ]
    },
    {
      "first_name": "M
      "last_name": "Pa
      "gender": "M",
      "role_code": "[",
      "role": "RTFJ Amateur Part 1",
      "contact_methods": [
        {
          "value": "00324********",
          "type_code": "GSM",
          "type": "Gsm"
        },
        {
          "value": "********@hotmail.fr",
          "type_code": "EMAIL",
          "type": "Email"
        },
        {
          "value": "04**/******",
          "type_code": "GSM",
          "type": "Gsm"
        },
        {
          "value": "********@hotmail.com",
          "type_code": "EMAIL",
          "type": "Email"
        }
      ]
    },
    {
      "first_name": "C
      "last_name": "Le
      "gender": "M",
      "role_code": "COMITE",
      "role": "Committee member",
      "contact_methods": [
        {
          "value": "********@hotmail.com",
          "type_code": "EMAIL",
          "type": "Email"
        },
        {
          "value": "00324********",
          "type_code": "GSM",
          "type": "Gsm"
        }
      ]
    }
  ]
}
```


Confidentiel

Confidentiel

Confidentiel

Avant 2 – Tous les contacts retournés

```
{
  "club": "jRS66lJJQcFe94lVshoh",
  "contacts": [
    {
      "first_name": "M
      "last_name": "Hu
      "gender": "M",
      "role_code": "Q",
      "role": "Authorized Correspondent",
      "contact_methods": [
        {
          "value": "04**/******",
          "type_code": "TEL",
          "type": "Phone"
        },
        {
          "value": "********@hotmail.com",
          "type_code": "EMAIL",
          "type": "Email"
        }
      ]
    },
    {
      "first_name": "P
      "last_name": "Le
      "gender": "M",
      "role_code": "COMITE",
      "role": "Committee member"
    },
    {
      "first_name": "R
      "last_name": "Kr
      "gender": "M",
      "role_code": "COMITE",
      "role": "Committee member",
      "contact_methods": [
        {
          "value": "info@******.be",
          "type_code": "EMAIL",
          "type": "Email"
        }
      ]
    },
    {
      "first_name": "M
      "last_name": "Pa
      "gender": "M",
      "role_code": "[",
      "role": "RTFJ Amateur Part 1"
    },
    {
      "first_name": "C
      "last_name": "Le
      "gender": "M",
      "role_code": "COMITE",
      "role": "Committee member"
    }
  ]
}
```


Confidentiel

Confidentiel

Confidentiel

Confidentiel

Après – Contacts privés non-retournés

5.4.3. Calcul des points de licence

src/svc_generic_api/modules/data_loaders/firestore/loader.py

```python
async def get_member_licence_points(self, member_id) -> int:
        """Return the points for the member licence"""
        validity_end_date = validity_end_date = datetime.today()
        licenses = await self._query_license_events(
            member_id, datetime_str(validity_end_date)
        )  # TODO This condition may break if field change to datetime
        if not licenses:
            licenses = await self._query_license_events(member_id, validity_end_date)
        memory = {}
        jobs = [
            self.get_license_points(licence.learning_event.id(), memory)
            for licence in licenses
            if licence.learning_event and licence.status == "SUCCEEDED"
        ]
        sum_points = 0
        for result in await asyncio.gather(*jobs):
            id, nb_points, start_date = result
            valid_date = await self.check_date_event(member_id, id, start_date)
            if valid_date:
                sum_points += nb_points
        return sum_points

    async def check_date_event(self, member_id, event_id, start_date):
        from svc_generic_api.modules.firestore.models.member_learning import (
            MemberLearning,
        )

        learnings = (
            await MemberLearning.query(db=self.engine.db, collection_args=member_id)
                .where("learning_event.ref", "==", f"learning_events/{event_id}")
                .get()
        )

        return learnings[0].validity_start_date < (
            start_date if start_date else learnings[0].validity_start_date - timedelta(1)) and
learnings[0].license_number is not None
```

src/svc_generic_api/modules/data_loaders/firestore/loader.py

```python
async def get_license_points(self, learning_event_id, memory):
        from svc_generic_api.modules.firestore.models.learning_event import (
            LearningEvent,
        )
        if learning_event_id not in memory:
            memory[learning_event_id] = asyncio.create_task(
                LearningEvent.get(learning_event_id, db=self.engine.db)
            )
        learning = await memory[learning_event_id]
        if learning and learning.nb_points and (
                learning.validity_end_date if learning.validity_end_date else datetime.today() +
timedelta(days=1)) > datetime.today():
            return learning_event_id, learning.nb_points, learning.validity_start_date
        return 0
```

src/svc_generic_api/modules/data_loaders/firestore/loader.py

```python
async def get_valid_member_learnings_when_events(
        self,
        member_id: str,
        language: Language,
    ) -> List[entities.MemberLearning]:
        from svc_generic_api.modules.firestore.models.member_learning import (
            MemberLearning,
        )

        from svc_generic_api.modules.firestore.models.learning_event import (
            LearningEvent,
        )

        query = (
            MemberLearning.query(db=self.engine.db, collection_args=member_id)
            .where("learning_type", "==", "event")
            .where("status", "==", "SUCCEEDED")
            .order_by("validity_end_date", direction=OrderFirestore.descending)
            .end_at({"validity_end_date": datetime_str(datetime.today())})
        )

        results = await query.get_entity(language)
        valid_events = []
        for result in results:
            learning_event = await
self.engine.collection(f"learning_events").document(result.learning_event.id)
            learning_event = LearningEvent.load(learning_event).to_entity(language)
            valid_date = await self.check_date_event(member_id, result.learning_event.id,
                                                     learning_event.validity_start_date)
            if valid_date:
                valid_events.append(result)
        return valid_events
```

```
{
  "licence": {
    "date": "2021-05-01",
    "learning_type": "course",
    "season": 2021,
    "grade": "EQ",
    "diploma_code": "Q",
    "end_date": "2022-05-01",
    "diploma_type": {
      "name": "UEFA Pro License",
      "discipline_code": "P",
      "code_type": "TYPCOURS",
      "priority": "5"
    },
    "licence_id": "123",
    "is_illegible_for_renewal": false
  },
  "licence_points": [
    {
      "id": "********",
      "date": "2021-06-01",
      "points": 2,
      "learning_type": "event",
      "name": "VFV Clinic : Jeugdopleiding - kaatsen met indoorvoetbal",
      "season": 2021,
      "status": "SUCCEEDED",
      "start_date": "2020-06-01",
      "end_date": "2022-12-31"
    },
    {
      "id": "********",
      "date": "2015-12-21",
      "points": 2,
      "learning_type": "event",
      "name": "Analyse van het EK en WK voor U17",
      "season": 2021,
      "status": "SUCCEEDED",
      "start_date": "2015-12-21",
      "end_date": "2022-06-01"
    },
```


Avant 1 – Cours qui ne devraient plus être pris en compte retournés

```
    {
      "id": "********",
      "date": "2017-06-14",
      "points": 2,
      "learning_type": "event",
      "name": "DENDERLEEUW - OPWARMING, MEER DAN LOPEN ALLEEN",
      "season": 2021,
      "status": "SUCCEEDED",
      "start_date": "2017-06-14",
      "end_date": "2022-06-01"
    },
    {
      "id": "********",
      "date": "2016-08-22",
      "points": 1,
      "learning_type": "event",
      "name": "Oost-Vlaanderen Infoavond Jeugdvoetbal Interprovinciaal en Provinciaal",
      "season": 2021,
      "status": "SUCCEEDED",
      "start_date": "2016-08-22",
      "end_date": "2022-06-01"
    },
    {
      "id": "********",
      "date": "2016-11-15",
      "points": 2,
      "learning_type": "event",
      "name": "Opleiding veilig voetballen - Reanimatie - EHBO (Melle)",
      "season": 2021,
      "status": "SUCCEEDED",
      "start_date": "2016-11-15",
      "end_date": "2022-06-01"
    },
    {
      "id": "********",
      "date": "2016-05-30",
      "points": 4,
      "learning_type": "event",
      "name": "Meevoetballende doelman (Oost Vlaanderen)",
      "season": 2021,
      "status": "SUCCEEDED",
      "start_date": "2016-05-30",
      "end_date": "2022-06-01"
    }
  ]
}
```


Avant 2 - Cours qui ne devraient plus être pris en compte retournés

```
{
  "licence": {
    "date": "2021-05-01",
    "learning_type": "course",
    "season": 2021,
    "grade": "EQ",
    "diploma_code": "Q",
    "end_date": "2022-05-01",
    "diploma_type": {
      "name": "UEFA Pro License",
      "discipline_code": "P",
      "code_type": "TYPCOURS",
      "priority": "5"
    },
    "licence_id": "123",
    "is_illegible_for_renewal": false,
    "nb_points": 2
  },
  "licence_points": [
    {
      "id": "********",
      "date": "2021-06-01",
      "points": 2,
      "learning_type": "event",
      "name": "VFV Clinic : Jeugdopleiding - kaatsen met indoorvoetbal",
      "season": 2021,
      "status": "SUCCEEDED",
      "start_date": "2020-06-01",
      "end_date": "2022-12-31"
    }
  ]
}
```


Après – Seuls les cours qui doivent être pris en compte retournés

5.4.4. Remises


```
{
  "id": "********",
  "cancellations": [
    {
      "id": "********",
      "discipline_code": "P",
      "start_date": "2017-01-13",
      "end_date": "2017-01-15",
      "modification_date": "2017-01-12T20:36:35+00:00",
      "label_nl": "Voor het weekend van 13, 14 en 15 januari 2017 heeft het Provinciaal Comité van Brabant beslist over te gaan tot een algemene afgelasting van alle officiële & vriendschappelijke wedstrijden.\n\nDe wedstrijden van de groeperingen ABSSA en Interbanken zijn eveneens afgelast.",
      "label_fr": "Pour le week-end des 13, 14 et 15 janvier 2017, la province de Brabant a décrété une remise générale de tous les matches officiels & amicaux.\n\nLes rencontres des groupements corporatifs ABSSA et Interbanques sont également remises.",
      "federation_code": "VFV"
    }
  ]
}
```


Avant – 2 labels retournés (FR et NL)


```
{
  "id": "********",
  "cancellations": [
    {
      "id": "********",
      "discipline_code": "P",
      "start_date": "2017-01-13",
      "end_date": "2017-01-15",
      "modification_date": "2017-01-12T20:36:35+00:00",
      "description": "Pour le week-end des 13, 14 et 15 janvier 2017, la province de Brabant a décrété une remise générale de tous les matches officiels & amicaux.\n\nLes rencontres des groupements corporatifs ABSSA et Interbanques sont également remises.",
      "federation_code": "VFV"
    }
  ]
}
```


Après 1 – Description en français

```
{
  "id": "********",
  "cancellations": [
    {
      "id": "********",
      "discipline_code": "P",
      "start_date": "2017-01-13",
      "end_date": "2017-01-15",
      "modification_date": "2017-01-12T20:36:35+00:00",
      "description": "Voor het weekend van 13, 14 en 15 januari 2017 heeft het Provinciaal Comité van Brabant beslist over te gaan tot een algemene afgelasting van alle officiële & vriendschappelijke wedstrijden.\n\nDe wedstrijden van de groeperingen ABSSA en Interbanken zijn eveneens afgelast.",
      "federation_code": "VFV"
    }
  ]
}
```


Après 2 – Description en néerlandais

5.4.5. Compétitions


```
{
  "id": "********",
  "name": "U15 Iris Elite (15Ila)",
  "code": "Confidentiel",
  "season": 2019,
  "label": "Confidentiel",
  "has_ranking": true,
  "is_published": true,
  "has_penalties": false,
  "has_nonofficial_goals": false,
  "is_public": true,
  "priority": 200,
  "type": "championship"
}
```


Avant – Pas de fédération retournée

```
{
  "id": "********",
  "name": "U15 Iris Elite (15Ila)",
  "code": Confidentiel
  "season": 2019,
  "label": Confidentiel
  "has_ranking": true,
  "is_published": true,
  "has_penalties": false,
  "has_nonofficial_goals": false,
  "is_public": true,
  "priority": 200,
  "type": "championship",
  "federation_code": "ACFF"
}
```

Après – Fédération retournée

5.5. Terraform

5.5.1. RBFA-DATALAKE

```
resource "google_project_iam_member" "marika_rbfa" {
  for_each = toset([
    "roles/bigquery.dataViewer",
    "roles/bigquery.jobUser",
  ])
  project    = var.project_id
  role       = each.value
  member     = "user:mawin.ext@rbfa.be"
  depends_on = [google_project_service.cloudresourcemanager]
}
```

5.5.2. RBFA-DP-OLTP

```
resource "google_project_iam_member" "marika_rbfa" {
  for_each = toset([
    "roles/firebase.viewer",
    "roles/datastore.viewer"
  ])
  project    = var.project_id
  role       = each.value
  member     = "user:mawin.ext@rbfa.be"
  depends_on = [google_project_service.cloudresourcemanager]
}
```

5.5.3. RBFA-DATA-PIPELINE

```
resource "google_project_iam_member" "marika_rbfa" {
  for_each = toset([
    "roles/bigquery.dataViewer",
    "roles/bigquery.jobUser",
  ])
  project    = var.project_id
  role       = each.value
  member     = "user:mawin.ext@rbfa.be"
  depends_on = [google_project_service.cloudresourcemanager]
}
```

6. Bibliographie

[1] Services internes de le RBFA. (2022).

[2] Royal Belgian Football Association. RBFA. https://www.rbfa.be

[3] Google Cloud documentation | *Documentation*. (2022). Google Cloud. https://cloud.google.com/docs

[4] Introduction to BigQuery administration |. (2022). Google Cloud. https://cloud.google.com/bigquery/docs/admin-intro

[5] REST API Documentation Tool | *Swagger UI*. (2022). Swagger UI. https://swagger.io/tools/swagger-ui/

[6] D. (2022, 21 mai). *Do Data Engineers Build APIs*. Data Science Nerd. https://datasciencenerd.com/do-data-engineers-build-apis/#:%7E:text=Data%20engineers%20build%20APIs%20in,performing%20other%20data%20engineering%20tasks.

[7] Geekflare Editorial. (2022, 16 février). *How to Successfully Build an API with Firebase*. Geekflare. https://geekflare.com/build-api-with-firebase/

[8] L, B. (2022, 17 janvier). *Data Engineer : tout savoir sur le métier d'ingénieur des données*. LeBigData.fr. https://www.lebigdata.fr/data-engineer-tout-savoir#:%7E:text=Le%20Data%20Engineer%2C%20ou%20ing%C3%A9nieur,Data%20Engineer%20con%C3%A7oit%20et%20fabrique.