**Université De Vincennes – Saint Denis**

**Département Hypermédia**

**Paris 8**

**Master II THYP**

**2018/2019**

**Programmation Mobile**

****

**Liste d’évènements et planification de parties Football collaboratif**

**Réalisé Par :**

**Objectif** :

Notre but est de concevoir une application Android pour organisation d’événements et planification de parties de football collaboratif. Cette dernière permettra à l’utilisateur de voir toutes les parties programmées autour de sa position en affichant la ville, l’adresse du stade, la date et l’heure, ainsi que la distance de chaque partie par rapport à sa position en temps réel. L’utilisateur aura également la possibilité de de rejoindre une partie déjà planifiée par un autre utilisateur si le nombre maximal de joueurs n’est pas encore atteint, ou bien planifier une partie lui-même et ajouter les personnes qui désirent en faire partie.

**Conception et développement :**

Nous avons opté pour  **FootBook** comme nom de l’application, **Foot** par rapport au sport concerné, et **Book** en anglais pour dire « Réserver », et nous avons fait le design d’un logo à ce nom pour l’application.

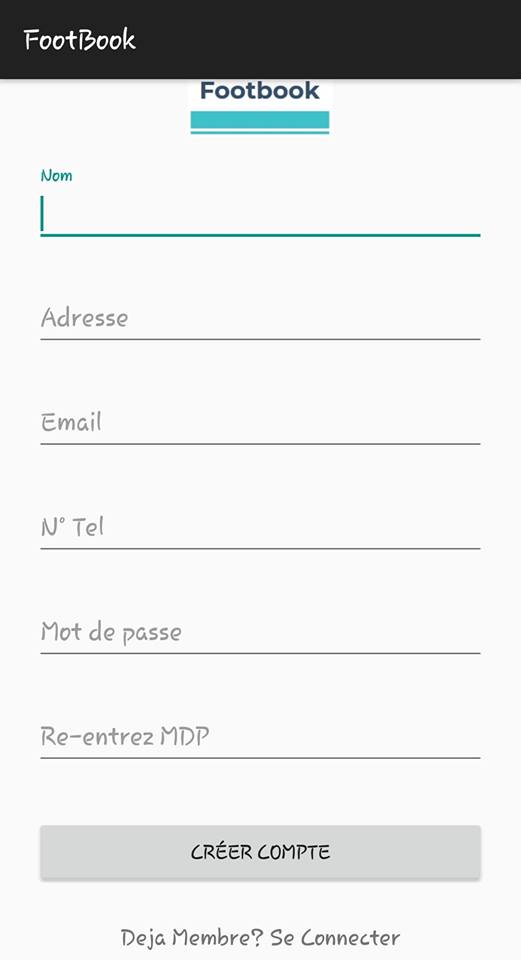
Le développement s’est fait en langage **Kotlin** sous **Android Studio**, et le processus a été réalisé en parallèle avec le cours en guise de TPs, et la correction de ces derniers nous a énormément aidés pour corriger nos erreurs et avancer dans la conception de notre application.

Dans le **Manifest**, on a mis les permissions (Internet, réseau, localisation) dont notre application a besoin pour un bon fonctionnement.

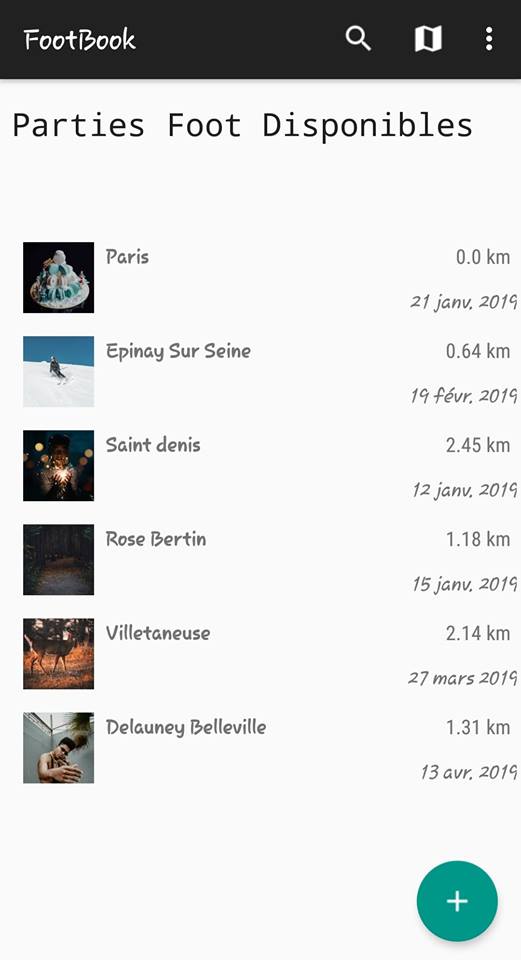
* **Stockage de données :** Concernant le type de stockage, nous avons choisi d’utiliser une base de données (**SQLite Database**) contenant la **TABLE\_PARTIES** qui représente les attributs de chaque partie foot (**id\_Partie,** **Stade/ville, Adresse, Date/heure, Image, longitude, latitude**). La création de la base de données se fait dans la méthode **OnCreate ()** en surchargeant cette dernière dans l’activité **DBParties**.
* **Login Activity :** La première interface en ouvrant l’application est la page de connexion ou l’utilisateur est amené à entrer son Email et mot d passe pour se connecter et accéder à la page d’accueil. Pour cette partie de connexion, on a repris un code existant de git Hub qu’on a adapté à notre perspective et besoins de l’application. Nous avons d’abord commencé par la conception du layout contenant une ImageView pour le logo de notre application, deux EditText de type textEmailAddress pour l’adresse Email et type textPassword pour l’entrée du mot de passe, tous les champs présents dans le layout sont reliés aux variables déclarées dans la fonction **OnCreate ()** grâce à leurs id. Elle comporte également un Bouton pour valider la connexion après avoir remplis les champs Email et password, en cliquant sur **Se connecter**, on fait appel à la fonction **login ()**, cette dernière appelle à son tour la fonction **validate ()** et vérifie le résultat retourné.

La fonction validate récupère les données saisies et vérifie si le champs n’est pas vide ou si l’adresse entrée correspond au format Email, sinon elle affiche un message d’erreur indiquant d’entrer une adresse valide, et vérifie également si le mot de passe entré respecte les normes indiquées (entre 4 et 10 caractères alphanumériques), si l’un de ces deux cas n’est pas satisfait, la fonction validate retourne un booléen faux à login () qui appelle ainsi onLoginFailed () pour afficher un toast « Connexion échouée », sinon cette fonction fait un **select** pour récupérer les utilisateurs existants dans la table **USERS** et vérifie si les données saisies figurent dans cette table et prend la décision finale pour donner ou refuser l’accès à la page d’accueil selon les résultats de la vérification. Si ce n’est pas bon, l’utilisateur sera invité à entrer les données correctes, sinon elle appelle onLoginSuccess () qui lance le Intent pour emmener à MainActivity (Connexion Réussie).

Enfin, un dernier textView sous forme de lien invitant un utilisateur qui n’est pas inscrit à créer un compte. En cliquant dessus, un Intent est crée et lancé pour diriger l’application à signUpActivity.

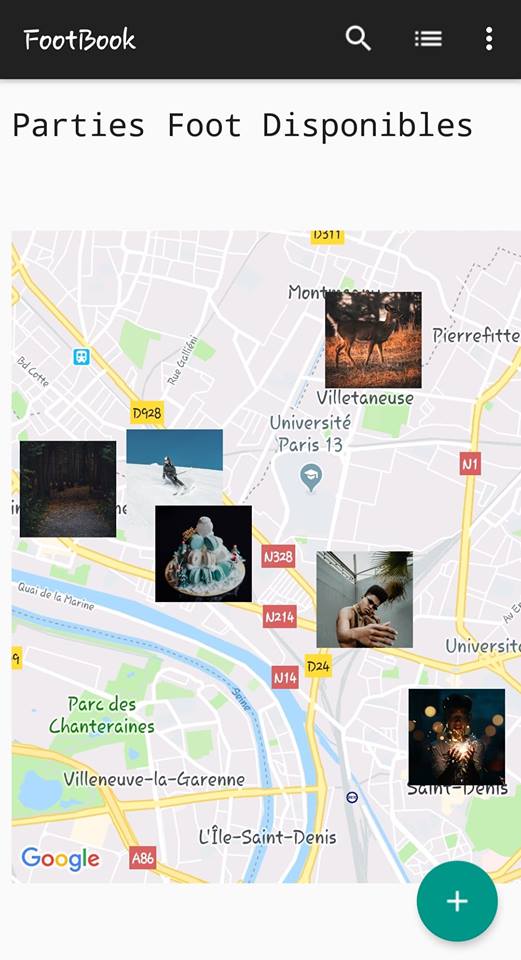
* **Sign Up Activity :** Dans l’écran de cette activité, on a mis des EditText pour recueillir les informations de l’utilisateur de types différents selon l’information concernée (Nom, adresse, Email, N° Tél, mot de passe, confirmation de mot de passe), un textView sous forme de lien qui permet à un utilisateur qui est déjà membre d’aller dans l’écran de connexion afin de se connecter. Et enfin, un bouton pour valider la création du compte, une fois ce bouton est cliqué, la fonction signUp () est appelée.

Cette dernière appelle la fonction validate () qui vérifie que toutes les conditions auxquelles les champs d’informations sont soumis sont bien réspectées et retourne un booléen à signUp(), si le retour est faux un message d’erreur (**toast**) est affiché par onSignupFailed () indiquant de corriger les champs concérnés et si le retour est vrai cette fonction véifie si ces informations ne correspondent pas à un utlisateur dans la table **USERS**, si ce n’est pas le cas ce compte et créé (**insert** dans la table **USERS**), et la fonction signUpSuccess () lance un Intent vers l’écran d’acceuil de l’application.

* **Main Activity :** Le premier écran d’accueil de notre application est composé d’un TextView indiquant le titre de la page, un FrameLayout pour contenir toutes les parties déjà planifiées sous forme d’une liste qui est définie dans le Layout List\_fragment comme illustré dans cette capture, et enfin un FloatingButton pour ajouter un évènement ou bien planifier une partie foot.

Dans la MainActivity on commence par relier le fragment cité précédemment (affichage Liste), et puis intégrer (inflate) le menu qui apparait tout en haut de l’écran (main\_menu) dont l’XML se trouve dans les Ressources/menu. Ce dernier comporte principalement deux items de recherche (un champ et une icône), un autre item pour rajouter d’éventuels options à pourvoir, et enfin un dernier item pour basculer de l’affichage **Liste** à l’affichage **Map**.

L’affichage de chaque partie dans la liste, prend le design qu’on a défini dans le layout **list\_partie**, qui comme illustré dans cette capture contient une photo pour chaque élément récupérée depuis un site générateur de photos aléatoires qu’on a gardé car l’utilisateur ne pourra pas prendre une photo en temps réel d’u stade pour pouvoir planifier une partie. On a également affiché pour chaque élément la ville de son déroulement, la date et heure, et la distance calculée entre l’endroit où se passera l’évènement et la position du l’utilisateur lors de la création.

En cliquant sur le **menuMap**, on a fait en sorte de masquer le menuMap et de rendre visible le **menuList** sous la fonction **onOptionsItemSelected** dans l’activité MainActivity afin de pouvoir rebasculer vers l’ancien affichage, et puis on remplace le fragment Liste par le fragment Map.

Ensuite on fait un **select** sur la table **PARTIES** de la base de données, et on récupère ce dont on a besoin pour un affichage sur la carte ; l’image de la partie, longitude et latitude afin de bien indiquer l’endroit sur la carte ou se déroulera l’évènement, et bien sûr id\_Partie qui servira pour afficher les détails de la partie quand on clique sur son image. Tout ce processus est inversé en cliquant sur le menuList.

On cliquant sur le FloatingButton, on ouvre l’activité AjouterActivity afin de permettre à l’utilisateur de créer un évènement.

* **Ajouter Activity :** Dans l’interface de cette activité, on trouve tous les champs nécessaires pour recueillir les informations indispensables pour planifier une partie. Dans cette activité on commence par définir les services de l’Api client de Google et service de géolocalisation dont on a besoin pour cette tâche. Par la suite, l’utilisateur sera amené à remplir le nom du stade ou de la ville où l’évènement se déroulera, la date et l’heure de la partie selon le format proposé seulement, et pour l’adresse exacte du stade l’application propose trois types d’indication en cochant le RadioButton correspondant :

**Ma position :** en choisissant cette option, l’utilisateur doit de tenir exactement à l’endroit où se trouve le stade car l’application va d’abord demander la permission d’Internet et du GPS déclarées dans e manifest pour utiliser les services de localisation et récupérer la longitude et latitude de la position de l’utilisateur qui sont déclarées comme variables globales, et enfin faire appel à la fonction updatePositionUtilisateur (). Cette dernière fait des copie de ces variables en local et les utilise (géocoder) afin de récupérer l’adresse exacte et l’afficher dans le champ adresse comme on peut le voir dans la capture.

**Cette Adresse :** pour cette option le champ d’adresse est activé et permet à l’utilisateur de saisir manuellement l’adresse.

**Choisir une position sur la carte :** si on coche cette option le bouton **ouvrir la carte** sera activé et en cliquant dessus on appelle l’activité **SelectionCarteActivity**, et l’utilisateur sera basculé vers une interface contenant un fragment Map avec une carte pour sélectionner la position du stade et un bouton pour valider.

Une fois que l’utilisateur trouve l’adresse souhaitée il clique dessus et on crée un marqueur (map.addMarker ()), on récupère la position (long et lat), si ces derniers ne sont pas NULL on les stocke dans les variables locales pour pouvoir les renvoyer via un Intent à l’activité AjouterActivity.

Cette dernière reçoit ces coordonnées et appelle la fonction updatePositionTexte () pour les coder en adresse texte qui sera affichée dans le champ adresse.

Maintenant que tous les champs sont remplis conformément aux conditions de chaque EditText, l’utilisateur clique sur le bouton envoyer pour la création de la partie, et cette action fait appel à la fonction envoyer () dans AjouterActivity.

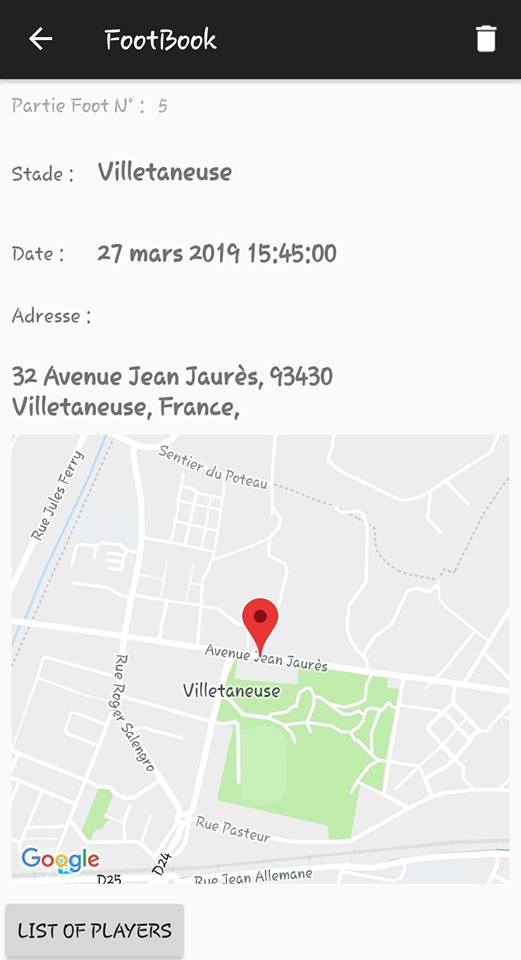
La fonction envoyer () commence par convertir l’adresse saisie manuellement en coordonnées si c’est le cas choisi, sinon on a déjà les coordonnées dans les deux autres cas et donc aucune conversion n’est nécessaire. Si ces coordonnées sont NULL on affiche un message d’erreur indiquant l’impossibilité de trouver la position, sinon on fait un insert pour stocker ces données récupérées dans la Table PARTIES et on affiche un toast de succès de création de la partie, tout en fermant cet écran et en envoyant l’utilisateur à l’écran d’accueil. Le retour à MainActivity fera un appel à la fonction chargerListe () pour charger les données de la table Parties et les afficher y compris la partie qui vient d’être ajoutée.

* Pour une recherche des parties disponibles en cas de grande liste, l’application envoie via un Intent vers Recherche Activity qui fait en sorte de récupérer le mot saisi pour la recherche (query), prépare le fragmentList pour afficher les résultats de la recherche sous forme de liste et appelle la fonction chargerListeMotClef (query) qui se trouve dans l’activité ListeFragment.

Cette fonction fait un select à la table des Partie en faisant usage du **where** accompagné du query afin de faire une recherche d’après les noms du stade/ville dans notre cas. Une fois cette liste est chargée on affiche les résultats dans le fragment du layout activity\_recherche selon le design montré en haut dans le fragmentList.

* **ListeFragment Activity :** dans cette activité on a surchargé la fonction onStart () en rajoutant une connexion à l’Api client et la fonction onClose () en rajoutant une déconnexion du même api.

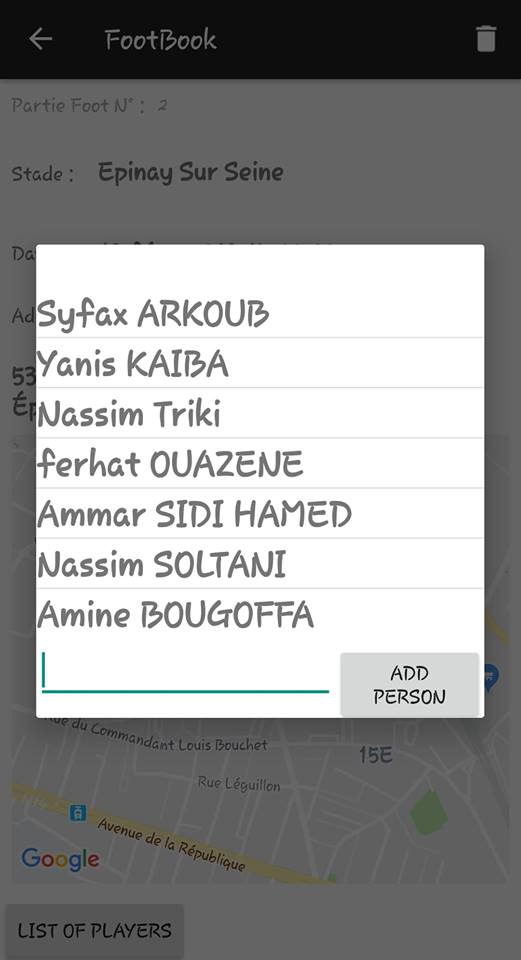
L’activité contient également la fonction chargerListeMotClef (query) précédemment expliquée, ainsi que la fonction calculDistance () qui sert à calculer la distance entre la position de l’utilisateur et l’endroit de l’évènement toujours en utilisant la longitude et la latitude. Aussi la fonction telechargerImage () qui comme son nom l’indique télécharge une image aléatoire pour chaque partie enregistrée.

****Une autre fonction **chargerListe** () qui est appelée à chaque retour à l’écran d’accueil, car elle fait un select de toutes les parties stockées dans la base de données sans where car on a besoin de l’ensemble des parties disponibles. Elle construit une liste et la donne en paramètres à l’activité partieRecycler dont les fonctions s’occupent d’afficher les résultats (image, nom, distance, date) sur les bons TextView selon le design qu’on a défini au début.

* **Détails partie :** Toujours dans Main Activity, lorsque l’utilisateur clique sur une partie pour afficher ses détails, la fonction afficherDetail (id) est appelée dans l’activité DetailFragment en prenant l’id da la partie en question comme paramètre.

Cette fonction fait un select sur la tables des parties en utilisant l’id en paramère pour récupérer les autres données concernant cette partie. Ensuite, elle affiche ces information dans les textView du layout activity\_detail\_fragment et le fragment pour la position sur la carte.

* **Détail Activity :** Le layout activity\_detail comporte un fragment regroupant les détails de la partie séléctionnée.

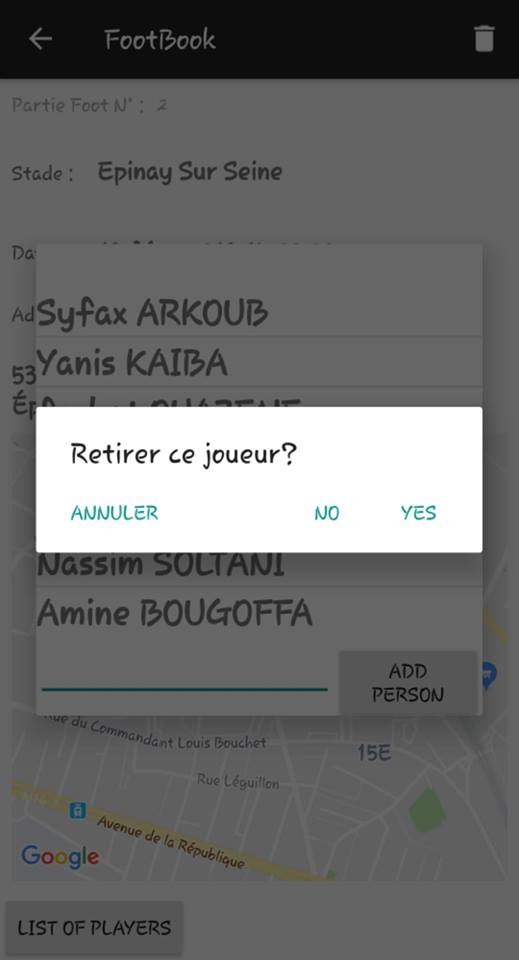
Tout en haut, le detail\_menu est intégré contenant un item pour supprimer la partie sélectionnée. En cliquant dessus la fonction **supprimerItemActif ()** est appelée qui performe simplement un delete sur la table partie en utilisant id récupéré lors de l’affichage de cette partie. Une fois la suppression est éffectuée, le DetailFragment va prévenir Main Activity de cette suppression à l’aide du mListener, et l’utilisateur est envoyé encore une fois dans l’activité DetailActivity tout en affichant un message de succès de suppression via Toast.

Enfin, tout en bas de l’interface, on trouve un bouton **liste des joueurs**. Ce bouton sert essentiellment à consulter la liste des joueurs déjà inscrits pour cette partie, ajouter un nouveau joueur ou bien en retirer un de la liste.

Lorsque ce bouton est cliqué, la fonction addPeople () est appélée ; cette dernière commence par afficher l’interface défini dans people.xml contenant un layout (wrap content) dont la heuteur s’adapte au contenu qui est une ListView comportant la liste des joueurs inscrits pour cette partie dont le style est défini dans adapter, un EditText pour saisir un nouveau nom et un bouton pour valider l’ajout.

L’utilisateur saisit le nom dun joueur et en cliquant sur **Ajouter** **Joueur** la fonction commence par vérifier le contenu de l’EditText, si ce dernier est vide un message sera affiche via Toast invitant l’utilisateur à saisir un nom et puis retourne vers le champ de saisie, sinon elle récupère le nom entré et l’enrigistre avec **SharedPreferences** qui est le type de **stockage** hoisi pour cette partie, et finalisé par un commit() et le nombre de joueur est incrémenté.

Après cela, le champ de saisie est remis à vide et le fonction getPeople () est appelée. Cette dernière récupère tous les noms stockés dans le sharedPreferences et les affiche dans la ListView dans le layout d’ajout.

Concernant l’état du bouton **Ajouter** **Joueur**, le nombre de joueurs est vérifié à chaque fois et si ça dépasse les 10 joueurs le bouton **Ajouter** est désactivé.

Toujours dans la même fonction, pour retirer le nom d’un jour on presse longtmps sur le joueur concerné (**onItemLongClickListener**) un builder apparaitra avec comme titre une question pour confirmer la suppression du item séléctionné avec deux bouttons négatifs **Annuler** et **Non** afin d’nnuler cette décision.

Un autre bouton positif afin de confirmer la suppression, quand ce bouton est cliqué la fonction fait un remove (retrait) d’élément sur le sharedPreferences contenant les noms en utilisant sa position dans la liste comme identifiant, suivi d’un commit() pour valider le remove. Puis le numéro de joueur est décrémenté et un message de confirmation de retrait de joueur est affiché avec un Toast.

Ensuite, on appellela fonction getPeople () pour un affichage des noms toujors stockés dans notre shared preferences, puis la fonction AddPeople () pour revenir à l’état précédent, où une action d’ajout ou de supression de joueur est attendue.

* **Perspectives :**

La principale difficulté était qu’on avait pas eu assez de temps pour pouvoir ajouter une Table **USERS** dans la base de données afin de vraiment enrigistrer les données des utilisateurs lors de la création d’un compte et pouvoir faire l’authentification par rapport aux données de cette table.

Une autre chose qu’on aurait voulu atteindre en plus de l’autentification, est de pouvoir restreindre certaines tâches à certains utilisateurs, telles que :

* Un utilisateur ne peut planifier une partie en même temps qu’une autre partie déjà programmée dans le même stade.
* Un utilisateur qui crée un événement a le droit d’ajouter et de retirer d’autres joueurs que lui-même , mais un autre utilisateur ne peut que rajouter son nom à une partie déjà existante ou le retirer et sans la capacité d’ajouter d’autres personnes.