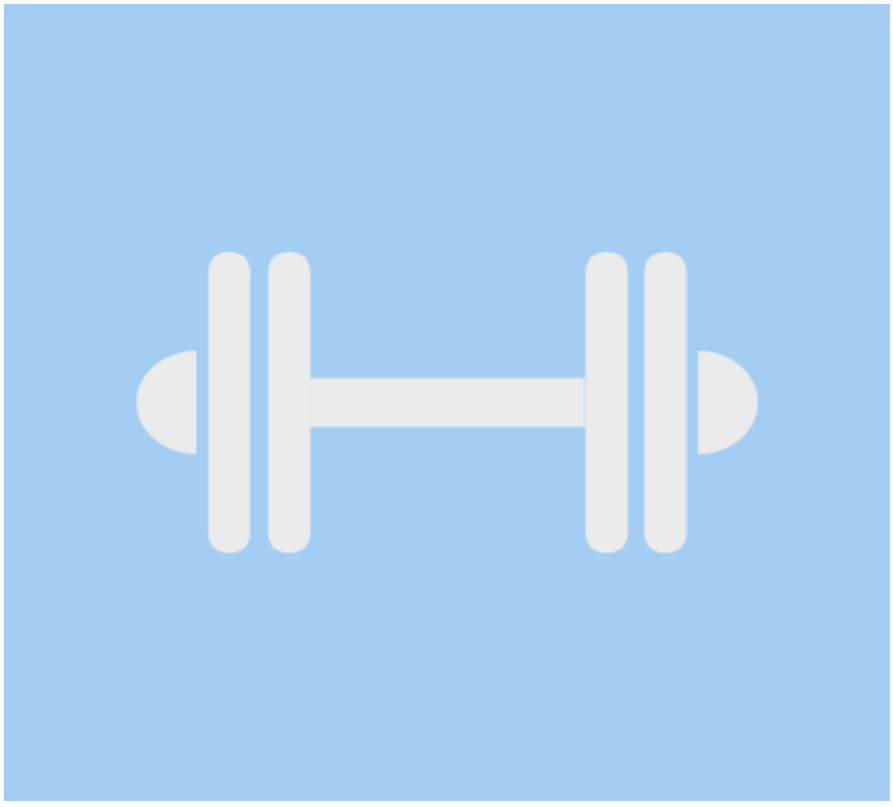
RAPPORT PROJET TUTORE - SPORT'ETU

Application de suivi sportif étudiant



AMINE ABOUSALHAM-DANOVAN NGUYEN
BENOIT FAUCHERY-ANTOINE PIRET

RAPPORT PTUT

IUT Lyon 1 -Département informatique

Sommaire

Introduction

C'est avec un grand plaisir que nous vous présentons notre projet tutoré, réalisé par quatre étudiants : Amine ABOUSALHAM, Antoine PIRET, Danovan NGUYEN et Benoît FAUCHERY.

Présentation

Notre projet tutoré consiste en la réalisation d'une application mobile Android à l'aide du langage Kotlin.

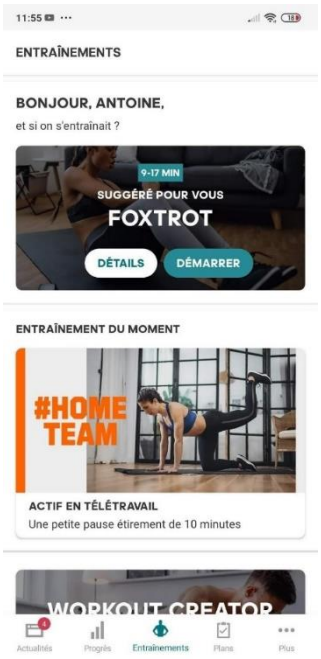
Après discussion, nous avons rapidement souhaité nous tourner vers une application qui serait orientée sur le sport, un sujet qui nous intéressait tous les quatre. Nous souhaitions également quelque chose qui soit utilisable par nos camarades de l'IUT. En cherchant un problème auquel apporter une solution, nous nous sommes dit que face à des emplois du temps chargés, certains étudiants ont de moins en moins de temps pour pratiquer une activité physique. C'est de ce constat qu'est née l'application Sport'Etu.

Il s'agit d'une application de suivi sportif classique mais qui permet néanmoins de prévoir très rapidement une activité et de se faire recommander différents sports en fonction de nos préférences. L'application se destine donc plutôt à des étudiants bien qu'elle soit tout à fait utilisable par n'importe quelle personne souhaitant insérer une séance de sport dans son emploi du temps. Les informations à traiter seront enregistrées dans une base de données.

Idées initiales et finales

Les idées pré-développement

Lors de la phase de préparation du projet au semestre 2, nous avons décidé de comparer 3 applications de suivi sportif différentes déjà existantes pour en dégager des fonctionnalités plus poussées qui seraient importantes ou intéressantes à mettre en place. Nous avions également prévu d'ajouter d'autres fonctionnalités qui collaient à notre problématique.



Tout d'abord, le premier aspect que nous avons souhaité garder par rapport à ces applications était leur ergonomie. Nous avons donc conservé la barre inférieure de navigation pour se déplacer entre les différents panneaux. De plus, nous nous sommes énormément inspirés des pages d'accueil personnalisées pour réaliser la nôtre.

Le côté que l'on souhaitait également réutiliser était la partie sociale de l'application. Nous souhaitions insérer un système d'amis pour que les utilisateurs puissent se confronter aux autres dans un classement en fonction du nombre d'activités qu'ils terminaient.

De notre côté, nous avons imaginé d'autres fonctionnalités qui seraient intéressantes. Tout d'abord, la possibilité pour l'utilisateur d'entrer son emploi du temps pour que l'application lui suggère elle-même différents créneaux. Nous voulions également qu'à la fin de chaque séance, l'utilisateur puisse entrer quelques phrases de ressenti par rapport à l'activité qu'il vient de terminer.

Pour citer quelques autres fonctionnalités plus minimes, on peut parler de l'implémentation d'un mode Nuit ou la création d'un système de trophées en fonction des performances de l'utilisateur.

Constat post-développement

Forcément, nous avions prévu beaucoup de fonctionnalités et nous avons dû les filtrer. Certaines étaient peu intéressantes et d'autres trop compliquées à implémenter.

La première que nous avons dû abandonner était l'idée de l'insertion de l'emploi du temps. Bien qu'intéressante, l'idée était très complexe à ajouter et elle n'apportait pas d'avantage qui soient incontournables.

Le côté social a été également mis de côté pour des raisons purement techniques. En effet, il aurait été très intéressant (notamment pour booster la motivation) que les utilisateurs puissent se confronter entre amis. Malheureusement, cela demandait des compétences qui allaient bien au-delà du domaine de l'application.

L'ajout des notes de fin d'activité ne s'est pas fait non plus. Cette fonctionnalité n'était plus pertinente par rapport à ce qu'est devenue l'application.

Le mode Nuit n'a pas été conservé puisque l'utilisation de cette application la nuit n'a que peu d'intérêts.

Ce ne sont que les principaux exemples qui montrent que l'application a beaucoup évolué entre sa conception et son développement. Il y a également beaucoup de fonctionnalités qui se sont ajoutées en cours de développement que nous vous laisserons découvrir dans la suite de cette présentation.

Technologies utilisées

Android studio

Il s'agit de l'environnement de développement intégré que nous avons utilisé pour le développement de l'application. Comme son nom l'indique, c'est un logiciel développé par Google qui est dédié à la création d'applications Android.

Kotlin

Kotlin est un langage de programmation orienté objet qui permet de compiler sur la machine virtuelle de Java. Depuis 2019, il s'agit du langage recommandé par Google pour la création d'applications sur la plateforme Android.

Git

Git est un logiciel de gestion de versions. Il nous a permis de collaborer sur le projet pendant toute sa durée.

Firebase

Firebase est un ensemble de service d'hébergement en NoSQL proposé par Google. Il est optimisé pour le fonctionnement avec les applications Android. Il nous a permis notamment d'enregistrer les données des utilisateurs et de gérer les inscriptions et authentifications.

Google drive

Google Drive est un service de stockage et de partage de fichiers dans le cloud proposé par Google. Nous nous en sommes notamment servis pour mettre en commun les ressources pré-développement comme notre cahier des charges ou nos comparatifs.

Conception

Architecture de l'application

Tout d'abord, il faut définir ce que sont une activité et un fragment.

Selon le site officiel de développement Android (https://developer.android.com/), une activité est une composante de l'application qui a pour rôle d'interagir avec l'utilisateur. En d'autres termes une activité correspond à une « page » de l'application affichée à l'utilisateur.

Un fragment quant à lui permet de scinder les activités en composants réutilisables. Cela permet de mettre en place des I.H.M. (Interface Homme Machine) qui s'adaptent aux différents écrans et à leur orientation.

Maintenant que nous avons défini ces deux notions, nous pouvons commencer la hiérarchisation de l'application.

Tout d'abord, l'application se lance avec l'activité authentification, qui va obtenir les données entrées par l'utilisateur, et vérifier que ces données correspondent bien à un compte utilisateur. Si l'utilisateur n'a pas de compte, l'activité « authentification » va nous emmener sur l'activité « inscription », où cette dernière prendra les informations entrées par l'utilisateur et les enregistrera comme un compte utilisateur dans la base de données.

Cliquer sur le bouton « CONNEXION » ou « INSCRIPTION » nous amène ensuite sur la page d'accueil : la page « Entrainement ».

Cette page d'accueil est en fait une activité composée de 3 fragments :

- Progression
- Succès
- Entrainement

On accède à ces différents fragments en cliquant sur les icones de la barre de navigation affichée en bas de l'écran.

Chaque fragment possède des références à d'autres activités, permettant de naviguer rapidement dans l'application.

De plus, on a ajouté une fenêtre de navigation que l'on peut faire glisser. On peut voir son nom, prénom, son adresse mail, le nombre de connexions ainsi que le nombre d'activités terminées. On peut aussi se déconnecter et revenir à la page d'authentification.

En annexe, ***le schéma de l'architecture de l'application***.

Design global

Le design a été réalisé sous Paint.net, par manque de bon logiciel d'imagerie et de design graphique.

Le design est le premier contact avec l'utilisateur. Il doit donc être réalisé avec soin.

Pour offrir à l'utilisateur un premier contact le plus agréable possible, il nous fallait des couleurs qui n'agresse pas les yeux de l'utilisateur. Il fallait donc exclure des couleurs trop agressives comme les extrêmes : blanc, le noir, le jaune, le rouge…

Nous avons donc choisi de faire l'application en nuances de bleu ciel, car la couleur est agréable à regarder, de nuit comme de jour.

Il fallait ensuite permettre à l'utilisateur de pouvoir naviguer entre les pages principales de l'application. Un moyen simple et instinctif de naviguer a été de créer une barre de navigation en bas l'écran, composé de 3 icones représentant les 3 pages. Lorsque l'on clique sur un des 3 boutons, celui-ci devient surélevé et un effet de bulle est rapidement affiché sur la barre de navigation, pour donner l'impression d'appuyer sur un vrai bouton.

Chaque page devait aussi être instinctive. Il fallait donc ajouter des boutons représentant les différentes fonctionnalités (planifier, consulter). Nous avons donc créé ces boutons, et nous avons affiché une image clipart sur chaque bouton pour renforcer le côté instinctif de l'application. Nous avons aussi ajouté du texte au-dessus de chaque bouton pour décrire la fonctionnalité.

Pour le menu déroulant permettant d'accéder aux paramètres ou de se déconnecter, nous avons choisis de l'afficher en haut à droite de l'écran car étant droitier, lorsque nous utilisons nos téléphones d'une seule main (la droite), nous pouvons facilement atteindre la partie droite de l'écran, et difficilement la partie gauche. Un mode gaucher pourrait alors être une bonne idée d'implémentation.

Dans ce menu déroulant nous avons choisis d'afficher en haut les informations importantes comme le nom et l'e-mail de l'utilisateur. Nous avons aussi décidé d'ajouter le nombre de connexions et d'activités terminées pour remplir un peu plus le menu, sans pour autant le saturer, ni mettre des informations non-pertinentes. Sous ces données sont aussi affichés les boutons paramètres et déconnexion. La différence de couleurs entre la partie des informations affichées et le reste du menu rend plus agréable la lisibilité, et rend plus instinctif le repérage de la partie « interaction » du menu.

Base de données

Pour authentifier nos futurs utilisateurs et stocker leurs informations personnelles, nous nous sommes, au départ, tournés vers une base de données Oracle SQL avec SQL Developer. En septembre 2020, il s'agissait du seul SGBD que nous avions vu à l'IUT. Seulement, nous nous sommes vite rendu compte que ce choix n'était pas du tout adapté à notre projet d'application mobile.

En effet, il n'existait sur internet aucun tutoriel, aucune documentation sur l'intégration d'Oracle SQL sur Android Studio. Ainsi, il nous aurait fallu construire toute l'architecture d'authentification nous-mêmes et la rendre sécurisée ce qui nous aurait pris bien trop de temps.

Ainsi, nous avons préféré opter pour une plateforme qui nous faciliterait grandement l'interaction avec une base de données : Firebase

Qu'est-ce que Firebase ?

Il s'agit d'un ensemble de services d'hébergement appartenant à Google, principalement utilisé pour la plateforme mobile (Android, IOS), mais aussi pour le Web (JavaScript, NodeJS), le développement logiciel ou les jeux-vidéos.

Nous avons utilisé le service d'authentification qui propose une connexion avec une adresse mail et un mot de passe et la possibilité de modifier ces 2 informations par l'intermédiaire d'un mail ou d'un sms de récupération. De plus, chaque compte contient un identifiant unique que l'on pourra utiliser avec la fonction getUserID() et qui nous sera essentiel par la suite pour identifier chaque utilisateur.

Nous avons également utilisé Firestore : c'est un service de gestion de base de données NoSQL en temps réel. À L'inverse d'une base de données SQL, les données ne sont pas structurées autour de tables. La base de données est non relationnelle. On n'a donc plus à réfléchir à l'architecture de notre base de données et cela nous fait gagner du temps.

Firestore s'organise de la manière suivante : chaque collection est composée d'un ou plusieurs documents et chaque document est composé d'un ou plusieurs champs qui peuvent être de plusieurs types (Integer, String, Boolean, Float…).

A l'inscription de chaque utilisateur, on enregistre son nom, prénom, adresse mail, ses 3 sports préférés et on initialise 2 Integer : le nombre de connexions ainsi que le nombre d'activités terminées. Ces compteurs nous seront utiles pour générer des succès. De plus, on peut déjà dire que le nombre de connexions est incrémenté de 1 à chaque connexion de l'utilisateur.

Voici une photo illustrant la structure de la BDD :

Chaque collection est identifiée par l'identifiant de l'utilisateur. A l'intérieur de chaque collection, le document Sport_utilisateur contient les sports préférés et profil_utilisateur contient les informations personnelles décrites ci-dessus. On verra par la suite que l'on a également stocké les activités planifiées ou terminées sur chaque collection d'un utilisateur.

Réalisation

Graphique

La partie design est très importante pour notre application, elle sert à attirer les utilisateurs et leur offrir un confort dans l'utilisation. Nous avons donc opté pour un thème plutôt bleu.

Les gradients permettent d'avoir un background avec un effet dynamique et le rendu suggère la modernité.

Le logo est un haltère, caractéristique d'une application sportive. Les couleurs du logo sont assez simples (blanc et bleu) et correspondent à notre charte graphique.

Le gradient se fait avec un choix de couleur au début et à la fin, et il faut aussi préciser un angle pour le mouvement et l'inclinaison des couleurs.

```
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <gradient
        android:angle="225"
        android:endColor="#0E478E"
        android:startColor="#23A6BF" />
</shape>
```

Les gradients sont organisés autour d'une gradient_list. Cela consiste à enchainer des gradients à la suite, avec l'ordre et la durée que l'on définit en fonction des parties.

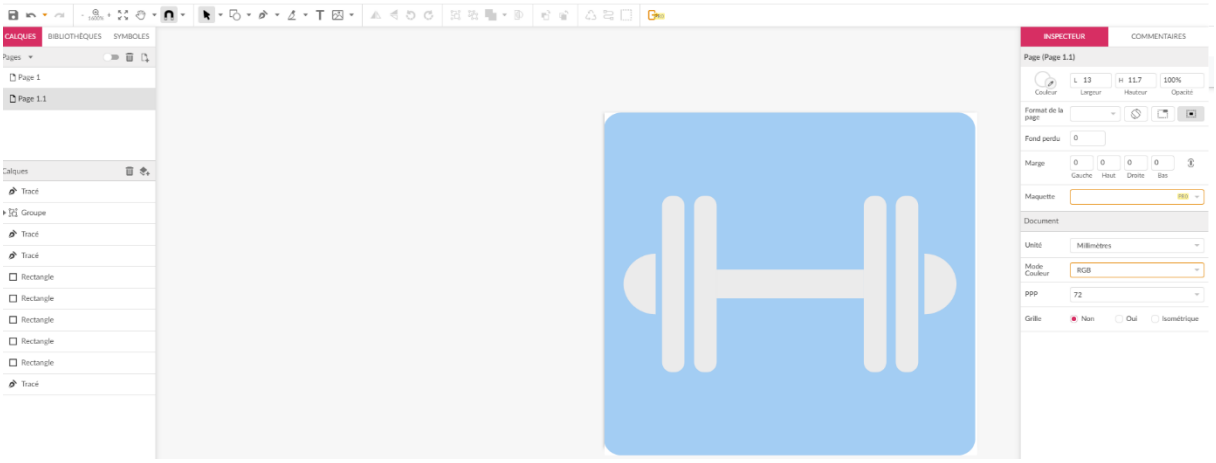
```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/gradient_1"
        android:duration="10000"/>
    <item android:drawable="@drawable/gradient_2"
        android:duration="10000"/>
    <item android:drawable="@drawable/gradient_3"
        android:duration="10000"/>
    <item android:drawable="@drawable/gradient_4"
        android:duration="10000"/>
    <item android:drawable="@drawable/gradient_5"
        android:duration="10000"/>
</animation-list>
```

Ensuite, il ne faut pas oublier de rajouter sur le code de l'activité le code de l'animation pour passer d'un gradient à un autre.

```
ConstraintLayout constraintLayout = findViewById(R.id.layouthisto);
AnimationDrawable animationDrawable = (AnimationDrawable)
        constraintLayout.getBackground();
animationDrawable.setEnterFadeDuration(2000);
```

```
        animationDrawable.setExitFadeDuration(4000);
        animationDrawable.start();
```

Pour le logo, nous avons utilisé le logiciel Gravit Designer (gratuit), qui permet une bonne précision et un large choix pour nos logos.



Services Authentification & Inscription

A l'authentification

Pour ne pas avoir à entrer ses identifiants à chaque fois que l'on lance l'application, on a décidé d'implémenter une fonctionnalité « se souvenir de moi » qui va enregistrer les identifiants à la fermeture de l'application et les écrire directement sur le formulaire de connexion à la réouverture.

Le principe est le suivant : si l'utilisateur a coché la checkbox « se souvenir de moi », lorsqu'on clique sur le bouton connexion et que les identifiants sont corrects, on vient enregistrer les identifiants saisis dans ce que l'on appelle les *Shared Preferences.*

Il s'agit d'un ensemble de fichiers stockés localement dans lesquels va enregistrer des variables. Ici, on va créer un fichier de préférences en mode privée, c'est-à-dire uniquement accessible lorsque l'on appelle le fichier. On fait cela car les identifiants sont à priori confidentiels. On vient aussi ajouter les deux variables correspondant à l'email et au mot de passe. De plus, on vient également ajouter une autre variable pour enregistrer l'état de la checkbox à true.

Lorsque l'utilisateur se reconnecte, il verra alors ses identifiants précédemment enregistrés directement écrits sur le formulaire. S'il décide de ne plus vouloir enregistrer ses identifiants et les supprimer, il n'a qu'à décocher la checkbox. En se connectant tout de suite après, on va vérifier si la checkbox est cochée ou que le mot de passe enregistré sur les préférences est égal à celui actuellement écrit sur les préférences. Si une de ces conditions est vérifiée, on vient supprimer les préférences. Sinon on enregistre encore les identifiants dans les préférences. On enregistre à chaque fois sur les préférences pour anticiper un changement dans les identifiants (si l'utilisateur veut enregistrer d'autres identifiants).

De plus, au lancement de l'activité, on vient d'abord récupérer la valeur de la checkbox stockée dans les préférences (sous la forme d'un booléen avec une valeur à true). De cette manière, si la valeur est fausse (false), c'est que le fichier n'existe pas, a été supprimé ou est incomplet. On vient donc supprimer les préférences pour être sûr. Sinon, on récupère le reste des informations.

A l'inscription

Lorsque l'utilisateur a rempli le formulaire d'inscription, il est redirigé vers ***une fenêtre de dialogue*** qui lui informe qu'il doit encore choisir ses 3 sports préférés. La fenêtre de dialogue est en fait toute une activité. On vient juste rajouter un background flou autour de la vue du dialogue pour donner l'impression qu'il s'agit d'un dialogue.

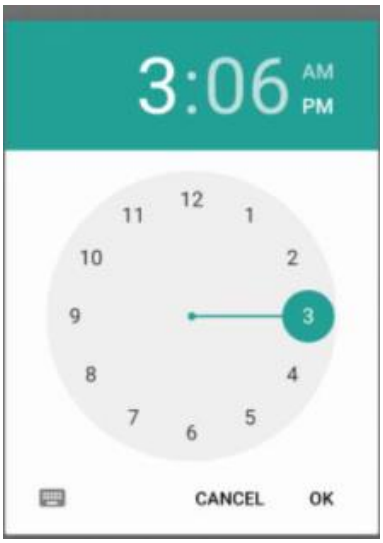
Ainsi, lorsque l'on appuie sur suivant, on est donc redirigé vers une autre activité où l'on vient choisir ses 3 sports préférés avec 3 simples sélecteurs. Pour finir, on vient enregistrer ses 3 sports sur le document Sport_utilisateur et lancer l'activité de la page d'accueil.

Planification d'une activité

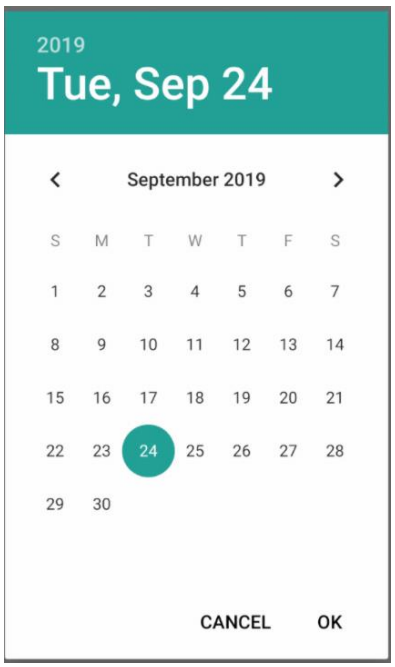
Cette fonctionnalité est en quelque sorte le cœur de notre application puisqu'elle correspond à l'objectif premier. On y accède par un bouton sur la page d'accueil.

L'action se fait via cette interface de saisie. Le premier panneau affiche d'abord un sport conseillé, basé sur l'algorithme de recommandation que nous aborderons plus tard. L'utilisateur dispose d'un champ de saisie en dessous pour définir une autre activité dans le cas où il ne souhaite pas choisir l'activité conseillée. Un deuxième panneau en dessous lui demande de renseigner une durée d'activité. Il alors simplement besoin de renseigner un nombre. Enfin, en dernière partie, deux boutons sont disposés pour lui permettre de sélectionner la date ainsi que l'heure auxquelles il souhaite effectuer son activité. Ces deux boutons ouvrent respectivement des fenêtre Android spécialement conçues pour saisir ces types de données.

Une fois tous les champs remplis, l'utilisateur n'a plus qu'à valider sa saisie à l'aide du bouton positionné en bas de l'écran.



Date Picker



Time Picker

Recommandation d'un sport

La partie recommandation de l'application est une des parties « phares » de notre application.

Appelé dans la partie « programmer une activité », juste après avoir cliqué sur le bouton Planifier, l'algorithme des recommandations n'est en effet pas le plus complexe de l'application, ni le plus long en lignes de code.

Etant donné que notre application vise à faire pratiquer du sport aux étudiants, il est facilement imaginable que la majorité des utilisateurs de cette application ne soient pas des sportifs de haut niveau (voir pas du tout pour certains). Notre application ayant pour but d'insérer ces personnes dans la pratique sportive, il est donc primordial de posséder un algorithme de recommandation solide pour pouvoir bien guider ces utilisateurs.

Nous avons tout d'abord réfléchi vers quel *type* de recommandation se tourner :

- Sur une liste non exhaustive de sports permettant à l'utilisateur
  plus de liberté quant à sa prise de décision

- Sur des sports en relations entre eux pour permettre à l'utilisateur
  de bien prendre ses marques dans une catégorie de sports

Le choix logique aurait été de faire des relations entre ces sports, mais étant donné que notre application vise à faire découvrir la pratique du sport, il est nécessaire que l'utilisateur soit sorti de sa zone de confort pour pouvoir s'ouvrir à toutes les possibilités et trouver le sport qui lui convient le mieux.

À partir de cette réflexion, nous avons pu commencer à modéliser l'algorithme.

Nous avons tout d'abord créé 2 listes non exhaustives de sports :

- 1 liste de sports collectifs
- 1 liste de sports individuels

Nous avons ensuite créé 4 variables aléatoires, nous permettant de choisir un sport entre :

- Les sports favoris
- Les sports généraux

Si les sports favoris sont choisis par l'algorithme, une variable va permettre de sélectionner lequel des 3 sports choisir.

Sinon, une variable va choisir le type de sport, et une autre va piocher un sport dans la liste de sports sélectionnés.

La fonction aléatoire nous a permis de donner plus de chances à certains évènements de se produire. Par exemple, il y a 2 chances sur 3 de tomber sur un sport favori, contre 1/3

chance de tomber sur un des sports généraux. ***En annexe l'arbre de probabilité de l'algorithme***

Cela permet donc à l'utilisateur de découvrir différents sports, de ne pas rester sur ces acquis, tout en suivant ses goûts sportifs.

Cet algorithme aurait cependant pu être écrit différemment, nous pensons qu'il n'y a pas qu'une seule façon de recommander les utilisateurs, mais c'est cette façon de pousser l'utilisateur à découvrir différents sports qui nous paraissait la plus séduisante.

Enregistrement d'une activité

Cette étape commence dès lors que l'utilisateur a validé la saisie de son activité. L'application vérifie déjà si le champ du nom de l'activité est saisi ou non pour sélectionner le sport conseillé ou le sport défini par l'utilisateur. Elle vérifie également que les champs soient bien remplis et que la date de départ de l'activité ne coïncide pas avec celle d'une autre. Si l'application ne valide pas un de ces critères, elle renvoie l'utilisateur à la programmation de l'activité.

Une fois ces tests validés, les champs vont être stockés dans la base de données en enregistrant également un champ supplémentaire permettant de définir si l'activité est terminée ou non. Chaque activité est identifiée dans la base de données dans un document par un nom unique composé de la concaténation du nom de sport, de la date ainsi que de l'heure de début de l'activité. On enregistre sur les champs la date, l'heure, la durée le nom de l'activité pratiqué ainsi qu'un statut qui sera soit « planifiée » soit « terminée ».

Ensuite, l'activité va lancer ce que l'on appelle un service de premier plan, un composant qui permet d'effectuer des tâches sans que l'application ne soit ouverte. Dans notre cas, le service va servir à l'affichage d'une notification informant l'utilisateur qu'une activité est programmée et par la suite qu'elle est terminée. Voici l'explication étape par étape de ce point technique.

Point technique : Service de premier plan

Extrait de ProgrammationActivity.kt

En annexe, l'extrait du code qui sera expliqué ci-dessous :

Sur ProgrammationActivity.kt qui est le code en Kotlin de l'activité de planification d'activité, on crée 5 variables en lateinit au début du fichier (que l'on initialisera plus tard, ici après l'algorithme de recommandation et le formulaire de l'activité). On y associe la durée de l'activité que l'on convertit en String et en Int, on associe également la date et l'heure en String, un statut défini en « planifiée » et enfin le nom de l'activité.

Après avoir créé la variable activiteAdd qui sera le nom de l'activité comme expliqué ci-dessus, on crée une référence de document Firestore (documentReference) avec comme collection l'userID et comme document activiteAdd. userID est une variable qui récupère l'identifiant de l'utilisateur courant avec la fonction getcurrentuser() suivie de getUid(). Ensuite, on crée une table de hachage qui va associer chacun des champs à envoyer au document par leurs valeurs respectives. Par exemple, la clé « nomActivite » est reliée à la valeur du nom de l'activité.

On utilise ensuite la fonction set() pour envoyer la table de hachage sur le document correspondant sur Firestore et addOnSuccessListener se déclenche si l'opération se solde par un succès.

Par la suite, il faudra placer une liste d'actions qui se déclenche lorsque l'activité planifiée est terminée, c'est-à-dire lorsque la date de l'activité ainsi que sa durée sont dépassées. On va donc effectuer ces actions dans un service. Il s'agit d'un processus qui peut s'exécuter même quand l'application est fermée ou réduite. Pour cela, on va créer un mécanisme d'échange entre activités appelé Intent. Avec la fonction putExtra(), on peut ajouter des variables en paramètre qui seront récupérées dans le service. On met les attributs de l'activité, son nom ainsi que l'échéance. Cette dernière est la date à partir de laquelle l'activité est terminée.

Elle est calculée en fonction des « milliseconds since epoch ». Il s'agit de la date en milliseconde depuis le 1 janvier 1970. En effet, Kotlin calcule la date actuelle avec la fonction System.currentTimeMillis() (date et heure précise) en comptant le nombre de millisecondes entre le 01/01/1970 et maintenant.

Ainsi, pour calculer notre échéance, il suffit de faire la somme de la date en format "yyyy-MM-dd-HH:mm"(fulldate) issue du date picker et du time picker convertie en millisecondes avec la durée de l'activité qui est à la base en heure elle aussi convertie en millisecondes (pour passer des heure au milliseconde, on multiplie 2 fois par 60 puis par 1000).

Enfin, Il nous reste plus qu'à lancer le service en premier plan avec la fonction startForegroundService().

Service.kt

Le fichier Service.kt étant très long et très complexe, on ne donnera pas son code source.

A la création du service (onCreate), il faut déjà appeler la fonction createNotification et l'intégrer à notre service en premier plan.

Ensuite, sur la fonction onStartCommand (à chaque fois que le service créé est appelé), on va tout d'abord récupérer les variables passées en paramètre de l'intent sur programmationActivity.kt avec intent.getTypeExtra (on remplace Type par le String, Int , Long…). Pour certains types comme les int ou les Long, il faut préciser une valeur par défaut dans le cas où la valeur récupérée est null. Par exemple, pour l'échéance, si jamais la valeur récupérée est null, elle est alors égale à la date et heure actuelle convertie en millisecondes que l'on a avec System.currentTimeMillis() (on veut que l'activité se termine instantanément en cas d'erreur).

Il faut par la suite créer une alarme qui va se déclencher lorsque la date actuelle est supérieure à l'échéance. On va alors se servir de la fonction onAlarm incluse dans l'alarmManager. En paramètre, il suffit de mettre l'échéance comme valeur à atteindre.

Lorsque l'activité devient terminée, on va alors modifier le status sur « valide » sur la base de données. De plus, on va aussi incrémenter le compteur du nombre d'activités terminées de 1 (avec la fonction FieldValue.increment()), générer une notification et enfin finir le service avec onDestroy().

Pour ce qui est des notifications, il faut bien distinguer 3 fonctions. On appelle déjà createNotification() qui va envoyer une première notification lorsqu'on lance le service (pour signifier que l'activité est planifiée). Cette fonction retourne donc un builder avec les paramètres de la notification tels que son titre, sa description ou son icone. Cette notification est persistante, c'est-à-dire qu'elle ne peut pas être détruite jusqu'à la fin de l'activité. Il faut savoir que l'on est obligé de mettre une notification persistante pour commencer un service en premier plan. Le service en premier plan est la seule solution pour que le système ne détruise pas le processus si jamais on quitte l'application. Par exemple, si jamais le système Android veut faire de la place dans la mémoire du téléphone, il ne viendra pas détruire le service en cours.

Ensuite, la fonction createNotificationChannel va créer une chaine de transmission nécessaire à la diffusion de la seconde notification. Celle-ci est différente de la première car étant déclenché à la fin de l'activité, elle va être non persistante.

Enfin sur la fonction notifActiviteTermine() on vient toujours créer le Builder avec les paramètres de la notification.

Recycler View

Pour afficher les données des utilisateurs, nous avons opté pour un RecyclerView, plus adapté que le listView pour notre demande.

Qu'est-ce qu'un RecyclerView ?

Le RecyclerView sert à afficher des données et les ordonner comme on le souhaite. Cela sert dans notre cas à afficher les sports qu'ont déjà fait nos utilisateurs par exemple (dans la partie historique des activités).

Le RecyclerView fonctionne avec 3 composants :

- On a tout d'abord le LayoutManager, qui permet de positionner les données dans notre liste.
- L'Adapter, qui permet de faire la liaison entre notre RecyclerView et nos données.
- Enfin, le Dataset, qui est l'ensemble des données en entrée. Il en existe deux types : le Dataset qui fonctionne avec une liste d'objets, celle-ci stockée localement et utile dans l'affichage des succès par exemple. On a également celui qui fonctionne avec notre base de données Firestore, qui récupère et affiche les données de notre BDD. On l'utilise ici pour l'historique des activités planifiées ou terminées.

Historique des activités

Une fois que l'utilisateur a planifié une activité, elle doit être consultable sur une liste. De plus, une fois que cette activité est considérée comme terminée, elle doit être archivée sur une autre liste.

Ainsi, pour l'affichage des activités planifiées et terminées, nous avons utilisé un Recycler View qui récupère les données en fonction d'une requête NoSQL. Par exemple, ***pour récupérer toutes les activités avec un statut « valide »,*** on choisit comme collection l'identifiant de l'utilisateur, et on vient chercher tous les documents qui ont un champ appelé « status » et une valeur « valide ».

Ensuite, on utilise le FirestoreRecyclerOptions.Builder pour établir la requête, on initialise le layout manager et on appelle l'adapter pour afficher sur les conteneurs les infos de la requête. Il faut savoir que Firestore dispose d'un adapter spécial pour afficher les résultats des requêtes NoSQL.

De plus, Firestore dispose de plusieurs fonctions pour faire utiliser correctement ces Recycler View. On a tout d'abord OnStart() et onMove() qui déterminent quand l'adapter doit commencer à « écouter » (listen) la requête Firestore. On a aussi onSwiped issue de la classe ItemTouchHelper qui permet de gérer ce qui arrive si on « swipe » avec notre doigt d'un côté ou de l'autre.

Dans notre cas, lorsque l'on swipe de la droite ou de la gauche, on veut pouvoir supprimer l'activité swipée. Il faut donc que l'on appelle getAdapterPosition() sur l'adapter et que l'on supprime l'item swipé comme ceci**.**

Extrait du code en Java pour supprimer une activité en swipant l'élément

Succès

Nous avons pensé qu'il serait sympathique de récompenser les utilisateurs avec des trophées symboliques. Ils sont générés à partir du nombre de connexions et du nombre d'activités terminées.

Comme l'historique des activités planifiées ou terminées, on utilise un Recycler View mais cette fois-ci, on ne récupère pas les données d'une requête pour les afficher. On crée les données à afficher localement à partir des 2 compteurs qui, eux, proviennent de la base de données.

Ainsi, il faut toujours initialiser le layout manager et appeler l'adapter mais on n'a pas besoin d'adapter spécial pour afficher des requêtes NoSQL.

Tout d'abord, on crée un constructeur addToList() qui va instancier à partir de l'adapter des objets succès avec un titre, une description ainsi qu'une image. Ces 3 attributs figurent donc sur chaque conteneur.

Ensuite, on crée une fonction postToList() dans laquelle on vient d'abord récupérer les 2 compteurs. Pour cela, après avoir défini le document à rechercher (documentReference), on vient récupérer sur deux variables de type Long les valeurs avec documentSnapshot !!.get(). Android Studio nous conseille d'ajouter des non null assertions « !! ». Cela permet d'assurer au logiciel que la valeur est non nulle et cela évite ainsi que l'application crashe sur des exceptions comme NullPointerException. Il s'agit d'une exception qui se lance si on tente d'utiliser une valeur null alors qu'une valeur est nécessaire.

Si les deux valeurs sont inférieures à 1, on affiche un message d'erreur, sinon pour chaque succès, on vient placer des conditions. Si elles sont vérifiées, on crée l'objet succès avec une image de trophée. Sinon on met une image de cadenas à la place.

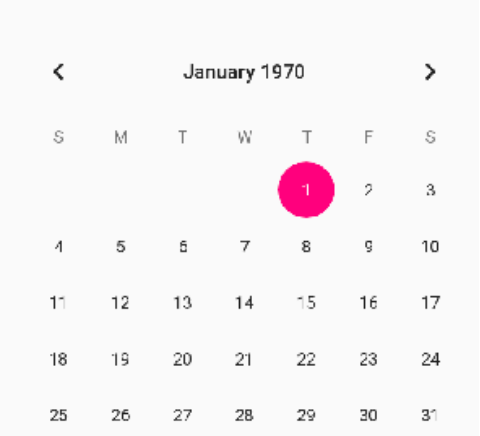
Par exemple, prenons le succès avec le titre « bonnes résolutions » et la description « Se connecter 10 fois ». Si le nombre de connexions est supérieur ou égale à 10, on vient alors créer un objet Succès avec donc le constructeur addToList et on précise en paramètre les attributs avec comme image un trophée. Sinon, on fait la même chose mais on met cette fois-ci une image de cadenas.

Problèmes rencontrés

Permettre à l'utilisateur de visualiser ses activités dans le temps

Un des plus gros problèmes que nous ayons rencontré était au niveau de l'enregistrement d'une activité. Cela se passait bien au niveau de la base de données mais nous voulions également quelque chose qui permette à l'utilisateur d'avoir un rappel ou une vue d'ensemble sur les activités qu'il a programmé. Notre idée de base était donc d'intégrer un calendrier directement dans l'application.

En effet, ayant repéré une vue de type Calendrier dans les outils d'Android Studio, nous pensions qu'il était possible d'intégrer un calendrier fonctionnel en se basant sur cette vue. Malheureusement, après plusieurs essais, nous avons découvert qu'il s'agissait d'un simple affichage et qu'il n'était pas possible d'y insérer des événements comme nous pourrions le faire dans un agenda numérique classique. Nous avons donc abandonné l'idée et nous nous sommes tournés vers un enregistrement directement dans le calendrier du téléphone.



La vue Calendrier présente dans Android Studio

Après plusieurs recherches, nous avons trouvé que sur Android cela se faisait en passant par un intent. Au moment de la validation de l'activité, l'application générait un intent de type Calendrier dont les attributs correspondaient aux différents champs que l'on peut trouver dans un calendrier (date de début, de fin, durée, etc…). On convertissait donc la date de l'activité qu'on convertissait en milliseconds since Epoch puis on y ajoutait la durée en millisecondes pour obtenir la date de fin.

Cependant cette méthode n'a jamais fonctionné. Notre application créait bien l'intent mais l'erreur se produisait au niveau de l'insertion dans le calendrier. Nous l'avons testé sur plusieurs marques de smartphones Android différents mais à chaque fois le système nous retournait qu'aucun calendrier Google n'était détecté. Après plusieurs recherches et

tentatives de modifications de paramètres nous avons fini par abandonné cette méthode et nous nous sommes tournés vers les services de premier plan dont nous vous avons parlé dans une partie antérieure.

Conflit de session

Cela a été un problème majeur lors de la réalisation de notre application et finalement jamais vraiment compris. En effet, étant donné que la connexion de l'utilisateur à Firebase n'est pas réellement encadrée par Android Studio, l'application peut, par erreur, ne pas savoir qui est connecté et bien souvent retourner null sur l'identifiant de l'utilisateur que nous essayons de récupérer pour effectuer nos opérations. Par exemple, lorsque que l'on se déconnectait, l'application crashait une fois sur deux en jetant une nullpointerException. Les fois où la déconnexion se passait sans encombre, c'était la connexion qui faisait des siennes : l'identifiant de l'utilisateur renvoyait constamment null. Ce problème s'accentuait lorsque l'on a décidé d'incrémenter de 1 le nombre de connexions à chaque fois.

Pour pallier ce problème, nous avons décidé de placer une activité qui sert de passerelle entre l'authentification (ou l'inscription) et l'activité d'accueil. Cette activité passerelle appelle Firebase et incrémente le nombre de connexions. De cette manière, on n'a plus eu ces types d'erreur par la suite.

Bilan global

En conclusion, l'objectif de l'application est globalement respecté. On a réussi à proposer à l'utilisateur un planificateur d'activité pour l'aider à rester motiver à faire du sport. De plus, on le récompense avec des trophées symboliques et sympathiques.

Cependant, on pourra noter que toutes les fonctionnalités décrites en conception n'ont malheureusement pas pu être réalisés soit par manque de temps, soit par manque de compétences. On retiendra surtout le fait d'avoir pu produire une application livrable sur le marché en un temps bien souvent court et avec des technologies au départ inconnues du DUT informatique.

Remerciements

Nous souhaitons tout d'abord remercier M.Karouri pour nous avoir accompagné tout au long du 2ème semestre. Ses connaissances ainsi que son expérience en planification de projet nous ont permis d'acquérir un rythme de travail professionnel qui nous a bien aidé pour finir ce projet. De plus, il a joué le rôle de client à merveille pour nous aider à rédiger le cahier des charges.

Nous souhaitons également remercier Mme Faci pour avoir choisi de rejoindre l'aventure au 3ème semestre. Elle nous a beaucoup aidé à nous orienter vers les choix les plus judicieux pour notre projet dans les plus brefs délais. Elle nous a également été d'une grande aide sur la fin de la partie conception de l'application.

Amine

Ce projet a été une expérience très enrichissante pour moi et m'a apporté beaucoup d'autonomie dans ma façon de travailler. En effet, les technologies utilisées pour ce projet m'étaient, au départ, inconnues. Il a fallu que j'apprenne à bien lire les documentations et à chercher les solutions intelligemment sur les nombreux forums et tutoriels.

De plus, j'ai appris à travailler en équipe, à partager les idées des autres et à communiquer de manière efficace.

Benoit

La réalisation de ce projet tutoré m'a beaucoup appris.

En effet, j'ai tout d'abord appris le travail en équipe. Dans un groupe de 4, il y aura forcément des idées qui divergent et des idées qui convergent. Ce que j'ai retenu de ces séances de PTUT, c'est d'exprimer nos idées, qu'il ne faut pas se butter sur son idée, et qu'il faut voir plus loin et accepter des points de vue différents. Il faut aussi avoir le cran de proposer ses idées même si elles paraissent, en premier lieu pas exceptionnelles. Il faut arriver à filtrer les propositions de chacun, pour obtenir un résultat satisfaisant. J'ai aussi appris à ne pas me refermer sur moi-même lorsque je rencontre un problème. Il y aura surement un de mes camarades pour m'épauler lorsque je rencontre une difficulté. Nous allons peut-être passer plus de temps sur la recherche de solution, mais nous irons plus loin.

De plus, Ce projet m'a aussi appris les bases de la programmation mobile. Nous ne connaissions rien à Android Studio et Kotlin au début de notre projet, mais grâce aux nombreux tutoriels disponible sur Youtube, et Stackoverflow (pour les plus connus), nous avons petit à petit réussi à prendre en main le logiciel qui pour ma part, n'est vraiment pas le logiciel le plus facile à prendre en main. Nous avons aussi utilisé de l'XML intégré à Android Studio, ce qui était encore une fois, une première mais qui était tout de même beaucoup moins compliqué à comprendre que Kotlin.

Enfin, si je devais refaire ce projet, je participerais bien plus à la partie développement de l'application, car j'ai trouvé que comparé à certains de mes camarades, je n'osais pas assez découvrir de nouvelles facettes d'Android Studio. Je serais donc ravi, à l'avenir, d'ajouter de nouvelles fonctionnalités à notre application.

Antoine

Je pense que ce projet m'a énormément apporté sur les aspects technique et social.

D'un point de vue technique, il m'a permis de découvrir le développement mobile. Je n'aurais sûrement jamais fait de développement mobile au cours de mes études et c'est pourquoi je suis content d'avoir pu m'y essayer. Je sais aujourd'hui que ce n'est pas le domaine dans lequel je veux travailler mais il s'agit quand même d'une expérience intéressante. De plus, la découverte du langage Kotlin est selon moi un énorme plus pour mon avenir professionnel. En effet, c'est un langage encore très récent mais je suis convaincu qu'il va à termes être extrêmement demandé puisqu'il est déjà recommandé par Google.

Niveau social maintenant, il s'agissait pour moi du premier « gros » projet que je réalisais en équipe. Cette gestion d'équipe était très intéressante à mettre en place sur ce projet car il combinait un aspect purement technique combiné à une partie graphique. Il a donc fallu apprendre à se répartir les tâches, à collaborer aux travers de différentes versions mais également à partager aux autres le savoir dont nous faisions l'acquisition. En effet, devant très fréquemment effectuer des recherches pour solutionner des problèmes, il fallait par la suite

être capable d'expliquer aux autres les nouveautés que nous ajoutions pour qu'ils ne soient pas perdus.

Je suis finalement très content de la manière avec laquelle nous avons conduit ce projet et je suis plus que satisfait du résultat final que nous avons obtenu.

Danovan

Ce projet sur la longue durée m'a beaucoup appris car on l'a fait de A à Z, ce qui se rapproche déjà plus d'un projet réel. Nous avons eu le temps de connaitre les points forts et les points faibles de chacun, afin de se compléter en cas de problème car nous avons tous commencés et découvert ce langage. Le travail en équipe et la communication ont beaucoup été demandés et je pense avoir acquis de l'expérience pour mes futurs projets.

Annexes

Schéma de l'architecture de l'application

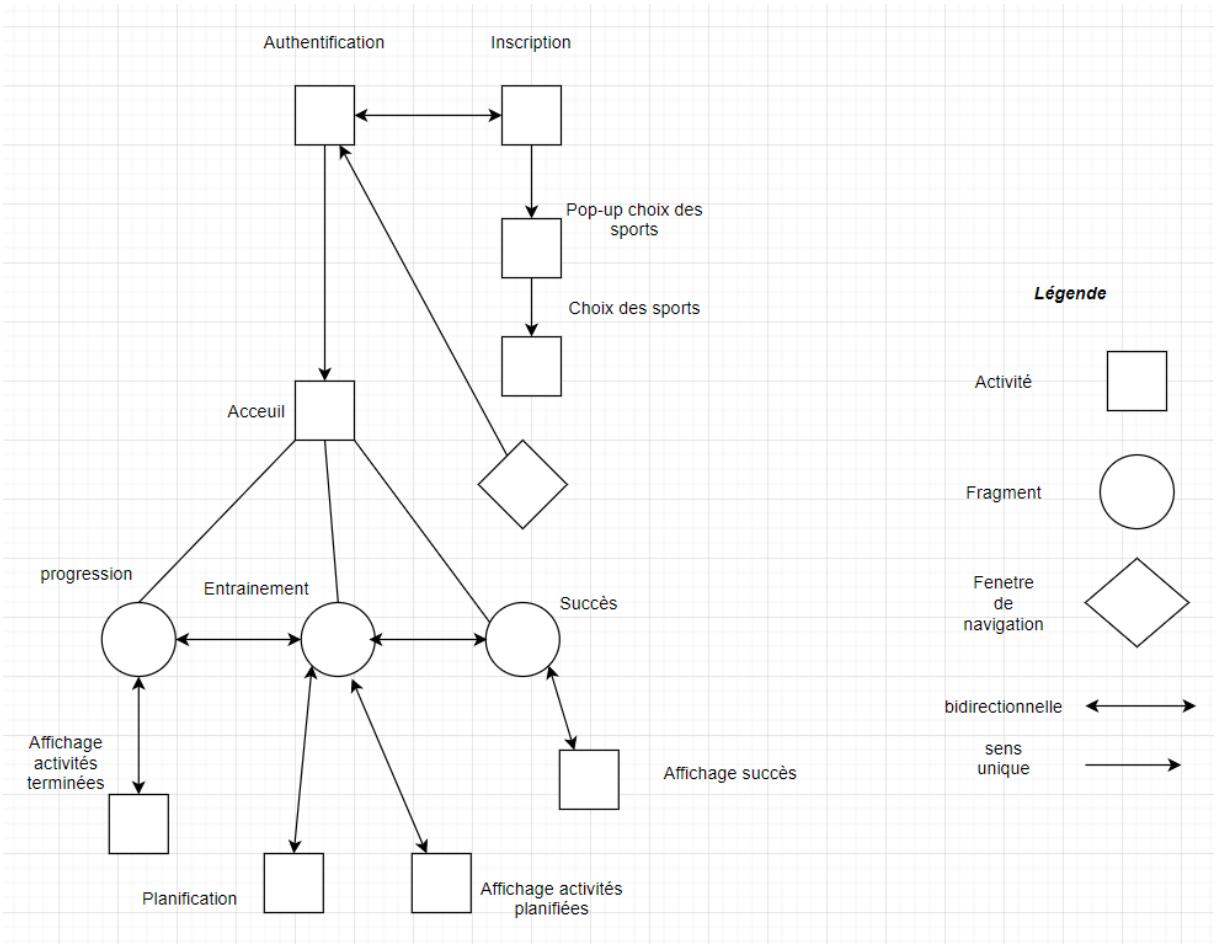
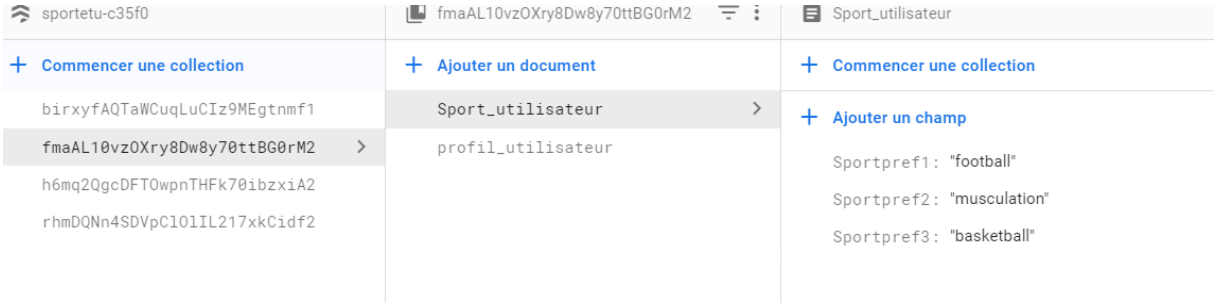
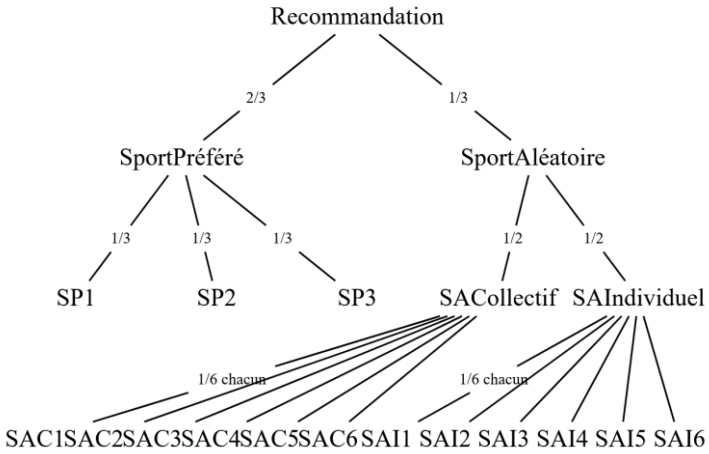


Photo illustrant la structure de la BDD



Fenêtre de dialogue du choix des sports

Arbre de probabilité de l'algorithme



Extrait de code en Kotlin de ProgrammationActivity.kt

```
buttonProgrammer.setOnClickListener {

    var activiteChoisie =nomActiviteTF.getText().toString()
    var SportConseille = sportConseiller.getText().toString()
    if(TextUtils.isEmpty(activiteChoisie)){
        activite=sportConseiller.getText().toString()
    } else {
        activite = nomActiviteTF.getText().toString()
    }


    duree = durreHourActiviteTf.getText().toString()
    durreInt=duree.toInt()
    status="planifie"
    date = timeTv.getText().toString()
    heure = hourTv.getText().toString()



    var activiteAdd :String
    activiteAdd = activite+date+heure

    val documentReference = mStore.collection(userID).document(activiteAdd)

    val Activite = hashMapOf<String, Any>(
        "nomActivite" to activite,
        "date" to date,
        "heuredebut" to heure,
        "duree" to duree,
        "status" to status
    )

    documentReference.set(Activite)
        .addOnSuccessListener {
            Log.d(
                "TAG",
                "onSuccess: activite ajoute$userID"
            )
        }.addOnFailureListener(OnFailureListener { e ->
            Log.w(
                "TAG",
                "Error writing document",
                e
            )
        })
```

Requête récupérant toutes les activités avec un statut "valide"

```
Query query = db.collection(userID).whereEqualTo( field: "status",  value: "valide");
```

Extrait du code en Java pour supprimer une activité en swipant l'élément

```
  @Override
    public void onSwiped(@NonNull RecyclerView.ViewHolder viewHolder, int
direction) {
        adapter.deleteItem(viewHolder.getAdapterPosition());
    }
}).attachToRecyclerView(recyclerView);
```