Issam Toure
Edouardo Rodrigues
Tarek Hallalel

# Rapport Mobistory

## Programmation Mobile M2 Info

M Chilowicz
Mars 2024

# Sommaire

# Constitution de la base d'événements historiques

## Introduction

Dans le cadre de ce projet d'application mobile, notre équipe, composée de trois membres, a adopté une stratégie de répartition efficace des tâches afin de faciliter le suivi de l'avancement du projet. Cette méthode de travail nous a permis de nous spécialiser sur différents aspects du développement, optimisant ainsi notre productivité et notre efficacité. En outre, nous avons mis en place une pratique de "shadow coding", où chaque membre avait l'occasion de suivre en temps réel le codage effectué par les autres. Cette approche collaborative nous a non seulement aidés à maintenir une cohérence dans notre base de code, mais a également favorisé un environnement d'apprentissage enrichissant, où chaque membre de l'équipe pouvait bénéficier de l'expertise et des perspectives des autres.

# Importation de la base

Pour l'importation de la base de données dans notre projet d'application mobile, nous avons adopté une approche prudente et efficace concernant le fichier 'events.txt'. Le processus commence par vérifier l'existence préalable du fichier pour éviter des téléchargements redondants. Si le fichier n'est pas déjà présent, nous procédons alors à son téléchargement. Après l'acquisition du fichier, une étape cruciale est l'analyse et le traitement (parsing) du code JSON qu'il contient afin de le rendre compatible avec notre schéma de base de données.


Figure 1: téléchargement du fichier events.txt


Figure 2: Parsing du Json

Au commencement du projet, notre base de données se composait principalement de deux tables : 'Events' et 'Claims', contenant respectivement les événements et les revendications associées à ces événements. Cependant, à mesure que le projet progressait, la nécessité d'une organisation plus détaillée et d'une segmentation plus fine des données est devenue évidente. En réponse à cette évolution des besoins, nous avons introduit de nouvelles tables telles que 'GEO' pour les informations géographiques et 'DATE' pour les dates spécifiques liées aux événements. De plus, au sein de la table 'Events', nous avons enrichi nos données avec l'ajout de nouvelles variables comme 'Favoris', permettant aux utilisateurs de marquer certains événements comme favoris, et 'Etiquette', offrant la possibilité d'annoter des evenements. Ces améliorations ont grandement contribué à accroître la fonctionnalité et l'utilisabilité de notre application mobile.

```kotlin
data class Event(
    val id: Int,
    val label: String,
    val aliases: String,
    val description: String,
    val wikipedia: String,
    val popularity: Popularity,
    val claims: List<Claim>,
    val date : Date? = null,
    val geo : Geo? = null,
    val isFavorite: Int = 0,
    val etiquette: String? = null

)

new *
data class Claim(
    val id : Int,
    val verboseName: String,
    val value: String,
    val item: Item?
)
```

Figure 3: Class Events et Claims

```kotlin
override fun onCreate(db: SQLiteDatabase) {
    val createEventsTableStatement = """
    CREATE TABLE $TABLE_EVENTS (
        $COLUMN_ID INTEGER PRIMARY KEY,
        $COLUMN_LABEL TEXT,
        $COLUMN_ALIASES TEXT,
        $COLUMN_DESCRIPTION TEXT,
        $COLUMN_WIKIPEDIA TEXT,
        $COLUMN_POPU_FR INTEGER,
        $COLUMN_POPU_EN INTEGER,
        $COLUMN_FAVORITE INTEGER,
        $COLUMN_ETIQUETTE TEXT

    )
    """.trimIndent()

    val createClaimsTableStatement = """
    CREATE TABLE $TABLE_CLAIMS (
        $COLUMN_CLAIMS_ID INTEGER PRIMARY KEY AUTOINCREMENT,
        $COLUMN_CLAIMS_EVENTID INTEGER,
        $COLUMN_CLAIMS_VERBOSE TEXT,
        $COLUMN_CLAIMS_VALUE TEXT,
        $COLUMN_ITEM_LABEL TEXT,
        $COLUMN_ITEM_DESCRITPION TEXT,
        $COLUMN_ITEM_WIKI TEXT,
        FOREIGN KEY ($COLUMN_CLAIMS_EVENTID) REFERENCES $TABLE_EVENTS($COLUMN_ID)
    )
    """.trimIndent()
```

Figure 4: Table Events et Claims

# Affichage de la liste d'événements

Dans notre application mobile, nous avons mis en place plusieurs méthodes pour trier et rechercher des événements, afin de maximiser l'expérience utilisateur en offrant une flexibilité et une efficacité optimales dans l'accès à l'information.

## la date de l'événement ou date de début et de fin

Notre approche commence par extraire la date de l'événement à partir des informations fournies dans l'un des claims associés. Cette date est ensuite insérée dans une table spécifique nommée Date, qui est initialement mise à null lors de sa création. Cette stratégie a plusieurs avantages significatifs. Tout d'abord, elle permet de centraliser et de normaliser les informations relatives aux dates dans la base de données, facilitant ainsi les requêtes et les filtrages basés sur les dates. En ayant directement accès aux dates des événements dans une table dédiée, nous évitons de devoir ré-analyser (re-parser) les claims à chaque redémarrage de l'application pour extraire à nouveau ces informations.

```
private fun extractEventDates() {
    if (!dbHelper.hasValidDateData()) {
        val eventDateMap = mutableMapOf<Int, Date>()
        itemList.forEach { event ->
            val firstDateClaim = event.claims.firstOrNull { it.value.startsWith( prefix: "date:") }
            firstDateClaim?.let { it: Claim ->
                val dateString = it.value.removePrefix( prefix: "date:")
                val year = extractYearFromDate(dateString)
                if (year != null) {
                    val (yearStr, month, day) = dateString.split( ...delimiters: "-")
                        .map { it.toIntOrNull() ?: 0 }
                    val date = Date(event.id, year, month, day)
                    eventDateMap[event.id] = date
                }
            }
        }
        dbHelper.addDateAllEvent(eventDateMap)
    }
}
```

Figure 5: extrait les dates des evenements

La fonction 'extractEventDates' extrait les dates des événements à partir des claims et les stocke dans une map liant l'ID de chaque événement à sa date. Elle parcourt tous les événements, extrait la première claim de date, la traite pour obtenir les composants de la date, crée un objet 'Date', et met à jour la base de données avec cette nouvelle information. Cette procédure est exécutée uniquement si les dates ne sont pas déjà validées dans la base.

```
fun hasValidDateData(): Boolean {
    val db = this.readableDatabase
    var hasData = false

    // Requête pour vérifier l'existence de lignes avec des valeurs dif
    val query = """
    SELECT EXISTS (
        SELECT 1
        FROM $TABLE_DATE
        WHERE $COLUMN_YEAR != 0 OR $COLUMN_MONTH != 1 OR $COLUMN_DAY !=
        LIMIT 1
    )
""".trimIndent()

    db.rawQuery(query, selectionArgs: null).use { cursor ->
        if (cursor.moveToFirst()) {
            // Si le résultat est 1, cela signifie qu'il existe au moin
            hasData = cursor.getInt( columnIndex: 0) == 1
        }
    }

    db.close()
    return hasData
}
```

Figure 6: verifie que la table dates ne contient pas deja des donnée

```
fun addDateAllEvent(eventDateMap: Map<Int, Date>) {
    val db = this.writableDatabase
    db.beginTransaction()
    try {
        eventDateMap.forEach { (eventId, date) ->
            val cursor = db.query(
                TABLE_DATE,
                arrayOf(COLUMN_DATE_EVENTID),
                 selection: "$COLUMN_DATE_EVENTID = ?",
                arrayOf(eventId.toString()),
                 groupBy: null,  having: null,  orderBy: null
            )
            val exists = cursor.moveToFirst()
            cursor.close()

            val eventValues = ContentValues().apply {  this: ContentValues
                put(COLUMN_DATE_EVENTID, eventId)
                put(COLUMN_YEAR, date.year)
                put(COLUMN_MONTH, date.month)
                put(COLUMN_DAY, date.day)
            }

            if (exists) {
                // Mise à jour de l'entrée si elle existe déjà
                db.update(
                    TABLE_DATE,
                    eventValues,
                     whereClause: "$COLUMN_DATE_EVENTID = ?",
                    arrayOf(eventId.toString())
                )
            } else {
                // Insertion de la nouvelle entrée
                db.insert(TABLE_DATE,  nullColumnHack: null, eventValues)
            }
        }
        db.setTransactionSuccessful()
    } finally {
        db.endTransaction()
    }
    db.close()
}
```

Figure 7: ajoute les dates a la table Date

Ce processus assure un accès aisé aux dates des événements partout dans l'application, évitant ainsi de devoir répéter inutilement l'analyse des données à chaque redémarrage de l'application.

## la localisation géographique de l'événement

L'implémentation de cette fonctionnalité commence par la collecte périodique de la localisation de l'appareil de l'utilisateur, effectuée toutes les 30 secondes. Cette approche garantit que nous disposons toujours d'une information de localisation à jour, essentielle pour fournir des résultats de recherche pertinents basés sur la proximité géographique. Une fois la localisation de l'utilisateur obtenue, nous utilisons une formule mathématique précise pour calculer la distance entre cette position et les localisations des différents événements enregistrés dans notre base de données.

Pour gérer efficacement les données de localisation associées aux événements, nous avons créé une table Geo dédiée au sein de notre base de données. Cette table contient les coordonnées géographiques de tous les événements qui disposent d'une localisation spécifiée dans leurs claims. Il est important de noter que certains événements peuvent ne pas avoir de localisation attribuée. Dans de tels cas, ces événements sont traités de manière appropriée pour s'assurer qu'ils ne perturbent pas l'expérience de recherche basée sur la localisation.


Figure 8: extrait les coordonnées géographiques des evenements

La fonction 'extractEventGeo' extrait les coordonnées géographiques des événements à partir des claims et les enregistre dans une map qui associe l'ID de l'événement à ses coordonnées géographiques. Elle ne procède à l'extraction que si les données géographiques ne sont pas déjà présentes, afin d'optimiser les performances de l'application. Une fois l'extraction terminée, la base de données est mise à jour avec les nouvelles informations géographiques.

tout le reste du processus est éxactement comme pour Date

Figure 9: trier les evenement pour les utiliser plus tard

Cette fonction maîtresse trie les événements en fonction du mode d'affichage sélectionné: par popularité, ordre alphabétique, temporalité, ou proximité géographique, en utilisant la latitude et la longitude actuelles de l'utilisateur. Elle utilise une chaîne de comparaisons conditionnelles pour déterminer le critère de tri approprié et met à jour la liste des événements avec l'ordre trié.

## le nom de l'événement

nous avons mits la possibilité de rechercher/trier les evenement par charactere

## la popularité de l'événement

Pour réaliser cela, nous avons intégré une variable de popularité dans notre classe d'événements. Cette popularité est affichée en français et en anglais, et le tri des événements s'effectue d'abord selon la popularité en français, puis en anglais. La règle de tri privilégie les événements ayant la popularité la plus faible, à l'exception de ceux dont la popularité est égale à zéro, qui sont considérés comme non classés. Il est à noter que certains événements peuvent ne pas avoir de valeur de popularité attribuée.

## favoris

Pour accomplir cette fonctionnalité, nous avons ajouté une variable booléenne nommée 'favoris', qui est par défaut réglée sur 0 (faux). Lorsqu'un utilisateur clique sur l'icône en forme de cœur sur la carte de l'événement, cette variable bascule à 1 (vrai), transformant ainsi l'icône de cœur vide en un cœur plein. Pour gérer ce changement, nous avons mis à jour la base de données de manière à ce que, suite à un clic, la valeur de la variable 'favoris' soit inversée (passant de vrai à faux ou inversement, grâce à l'opérateur '!favoris').

## étiquettes

Pour les étiquettes, nous avons développé un système intégré aux cartes qui, une fois qu'on clique sur l'icône d'édition, permet d'ajouter une note directement visible sur la carte de l'événement. Pour mettre en œuvre cette fonctionnalité, nous avons ajouté une variable de type chaîne de caractères (String) dans la classe événement et dans la Table Events. Cette variable est mise à jour dans la base de données chaque fois que le texte est modifié, assurant ainsi que les notes ou étiquettes personnalisées sont conservées et affichées correctement sur les cartes des événements.

# Frise chronologique

dans ce projet il est possible de passer directmeent du mode frise au mode card elle comprend les meme parametre de trie que le mode Card (Normal)

## temporel

Pour le mode temporel, nous avons mis en œuvre une frise chronologique innovante. Cette frise affiche les dates par incréments de cent ans, et chaque événement est attaché à une date spécifique, clairement marquée sur la frise, à la manière des frises chronologiques classiques. Cette approche permet aux utilisateurs de visualiser facilement l'ordre et le contexte historiques des événements, facilitant ainsi une compréhension plus profonde des périodes historiques.

## Geo

Le mode Géo, dans notre application, organise les événements sur la frise selon leur proximité à l'utilisateur, offrant ainsi une visualisation unique qui relie la distance géographique à l'expérience de navigation.

## Alphabetique et popularité

Dans les modes Alphabétique et Popularité, la frise dispose les éléments les uns après les autres dans une séquence horizontale, offrant une navigation linéaire et intuitive basée sur l'ordre alphabétique ou le niveau de popularité des événements.

# Événement du jour

Pour la fonctionnalité "Événement du jour", nous avons mis en place une notification qui fonctionne même lorsque l'application est fermée. Elle se déclenche en sélectionnant aléatoirement un événement parmi ceux qui partagent le même mois et jour que la date actuelle.

# Événements proches

Pour la fonctionnalité d'événements proches, nous avons choisi de ne pas l'intégrer dans notre application pour une raison spécifique. Le calcul des événements proches nécessiterait de mesurer la distance entre la localisation actuelle de l'utilisateur et celle de tous les événements dans la base de données à un instant donné. Cependant, cette opération s'est avérée trop exigeante en ressources pour notre application, entraînant des problèmes de performance et de stabilité.

# Quiz

Pour la fonctionnalité du quiz, nous avons conçu un jeu interactif où quatre événements sont sélectionnés aléatoirement dans la base de données à chaque manche. L'objectif pour l'utilisateur est de les classer par ordre chronologique, de la date la plus ancienne à la plus récente. Le quiz se déroule en cinq rounds, à l'issue desquels l'utilisateur reçoit sa note. Un point est attribué pour chaque round où tous les événements sont correctement classés, permettant ainsi d'évaluer la connaissance de l'utilisateur sur la séquence historique des événements.

# Implémentions GEO

Pour la géolocalisation, nous employons la même API que celle utilisée dans le TP5, identifiée par ‘com.google.accompanist:accompanist-permissions:0.33.2-alpha@aar‘.

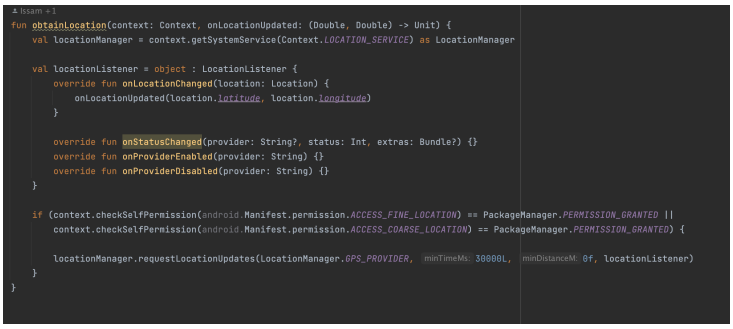


Figure 10: recuperer la localisation

Cette dernière est récupérée, comme illustré sur la figure 10, toutes les 30 secondes, puis intégrée dans l’application à l’aide d’un ‘LaunchedEffect‘.

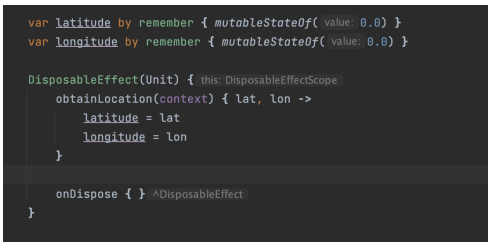


Figure 11: recuperer la géo de l’appareil

# Organisation Pour le Développement

Pour ce projet, nous avons initialement commencé par établir la base de données, télécharger et analyser le fichier event.txt. Cette phase a été réalisée collectivement par tous les membres de l'équipe, en adoptant parfois la méthode de shadow coding (où une personne programme pendant que les autres observent et fournissent des conseils) pour certaines de ces tâches. Après avoir complété cette étape, nous avons ensemble développé un ViewModel pour faciliter l'interaction entre les classes et les composants Composable. Une fois cette partie achevée, nous avons chacun commencé à travailler sur nos tâches spécifiques, ce qui a contribué à une progression rapide du projet.

Par exemple :

Issam a été responsable de la gestion des éléments favoris et des étiquettes pour chaque événement. Il a également pris en charge la non redondance au démarrage. De plus, il a conçu la frise chronologique selon sa vision et a été chargé d'intégrer la fonctionnalité de géolocalisation de manière efficace.

Eduardo a géré les ajouts dans la base de données pour assurer la persistance des données et leur intégration dans le ViewModel. Il s'est également occupé de la fonctionnalité de recherche d'événements et de certains filtrages. Pour conclure, il a été responsable de la conception du quiz.

Tarek a fourni l'affichage des cartes d'événements, ainsi que plusieurs méthodes de tri et des calculs de distance pour la fonctionnalité de localisation.

# Difficulté et Solution

La principale difficulté du projet a résidé dans la structuration initiale des tables. En raison de nombreux changements et reconceptions en début de projet, nous avons décidé de commencer avec deux tables principales, events et claims, avant d'en ajouter d'autres par la suite. Une autre difficulté rencontrée a été de se familiariser avec l'utilisation des ViewModels ainsi qu'avec le cycle de vie de l'application. Pour surmonter cela, nous nous sommes mutuellement aidés à comprendre quand et comment les utiliser et les mettre en place.

En outre, nous avons rencontré plusieurs problèmes liés aux différentes versions d'Android. La solution que nous avons trouvée a été de limiter certaines fonctionnalités aux versions plus récentes d'Android.

# Conclusion

Pour conclure, ce projet a représenté un véritable défi pour notre groupe de trois personnes. Tout d'abord, la nécessité de structurer efficacement les bases de données dès le départ a été une leçon importante, soulignant l'importance d'une planification et d'une conception préalables. La décision initiale d'opter pour deux tables principales, events et claims, a évolué au fil du temps avec l'ajout de nouvelles tables pour répondre aux besoins croissants du projet, reflétant notre capacité à s'adapter et à réagir aux défis de conception.

La maîtrise des ViewModels et la compréhension du cycle de vie des applications Android ont également constitué un défi majeur. En travaillant ensemble et en partageant nos connaissances et notre expertise, nous avons surmonté ces obstacles et aussi renforcé notre cohésion d'équipe et notre compétence collective en développement d'applications.

Les difficultés techniques, notamment les problèmes liés aux diverses versions d'Android, ont mis en lumière notre capacité à trouver des solutions créatives, comme restreindre certaines fonctionnalités aux versions les plus récentes d'Android pour assurer la compatibilité et l'expérience utilisateur.

Chaque membre a joué un rôle clé dans la réussite du projet. Issam, avec sa gestion des éléments favoris, des étiquettes, et du téléchargement et parsing des données; Eduardo, en assurant la persistance des données et le développement du quiz; et Tarek, en fournissant l'affichage des cartes d'événements et en intégrant des fonctionnalités de localisation. Cette collaboration a permis de développer une application riche en fonctionnalités.

En somme, ce projet a été une aventure enrichissante, pour nous tous. Il a renforcé notre compréhension des technologies Android, de la gestion de projet, et surtout, de l'importance du travail d'équipe.