DÉVELOPPEMENT D'APPLICATIONS MOBILES

Par : Alexandre Caron

Antonin Lenoir

Projet Synthèse

FRIENDS WAVE

L'application pour se faire une vague d'amis

Présenté à Monsieur Jean-Christophe Demers

Dans le cadre du cours de projet sythèse

Technique de l'informatique

Cégep du Vieux Montréal

2 mars 2023

DONNÉES ET SERVICES:

Pour cette application nous utiliserons plusieurs services afin d'offrir une expérience de qualité aux utilisateurs. Parmi ces services nous retrouverons :

- ***Firebase Realtime Database:*** c'est un service de base de données en NoSQL et qui permet de stocker les informations sur les utilisateurs, les événements et les commentaires de façon synchroniser et en temps réel. Les données sont stockées dans un format JSON, ce qui les rend facilement accessibles et modifiables. Les modifications apportées à la base de données sont immédiatement disponibles sur tous les appareils connectés. Firebase utilise la synchronisation en temps réel pour mettre à jour les données sur tous les appareils connectés. Firebase offre une sécurité de base des données en utilisant des règles de sécurité simples et intuitives. Les règles peuvent être définies pour contrôler l'accès aux données et les autorisations de modification.

- ***Firebase Authentication:*** nous utiliserons ce service pour gérer l'authentification des utilisateurs, ce qui inclut la gestion des comptes, la vérification de l'identité et la gestion des sessions. C'est un service de Firebase qui permet aux développeurs de gérer facilement l'authentification des utilisateurs de leur application. Il prend en charge plusieurs méthodes d'authentification, comme l'authentification par courriel et mot de passe, l'authentification avec des comptes Google, Facebook, Twitter, etc. Il offre également des fonctionnalités de gestion des comptes d'utilisateurs, telles que la création de comptes, la réinitialisation de mot de passe, la vérification de courriel, etc. Firebase Authentication offre une sécurité des données en utilisant des algorithmes de cryptage standard pour protéger les informations d'identification des utilisateurs. Les informations d'identification sont stockées de manière sécurisée dans la base de données Firebase.

- ***Google Maps API:*** pour fournir une carte interactive et des fonctionnalités de géolocalisation pour les événements, permettant aux utilisateurs de trouver et de s'inscrire à des événements près de leur emplacement actuel. Ce service permet d'afficher des cartes avec des informations sur les routes, les bâtiments, les zones d'intérêt, etc. Nous utiliserons ces informations pour créer une application qui intègre une cartographie interactive. Il est aussi possible de rechercher des lieux précis en fonction de l'emplacement de l'utilisateur ou de leur nom. Il prend en charge la recherche de différents types de lieux, tels que des restaurants, des magasins, des hôtels. On peut trouver des instructions étape par étape pour se rendre à un lieu, ainsi que des informations en temps réel sur les conditions de circulation.

- ***Firebase Cloud Messaging:*** peut être utilisé pour envoyer des notifications push aux utilisateurs lorsque des mises à jour sont disponibles pour leurs événements. FCM permet d'envoyer des notifications push en temps réel à des applications mobiles. Les notifications peuvent être déclenchées par des événements spécifiques, tels que la mise à jour de données dans Firebase Realtime Database ou Firebase Cloud Storage, ou par des requêtes manuelles depuis un serveur. On peut aussi envoyer des notifications push ciblées à des utilisateurs spécifiques. Ce service sera utile pour informer les utilisateurs d'un changement sur les événements inscrits ou tout simplement pour leur envoyer des rappels ou des suggestions.

STRUCTURE DE DONNÉES INTERNES ET EXTERNES:

Voici les structures de données internes qui pourraient être utilisées dans ce projet:

Nous utiliserons très certainement des ArrayList pour stocker les utilisateurs, les messages et les événements en les déclarant comme des listes d'objets de la classe User, Message et Évent. Nous pourrons par la suite afficher à l'usager soit sa liste de contacts personnels, ses messages, un choix d'éléments, etc.

Nous utiliserons un dictionnaire pour stocker différentes catégories d'événements que notre application propose (par exemple, sport, musique, conférences, etc.). Les clés du dictionnaire seraient les noms des catégories (par exemple, "sport", "musique", "conférences", etc.) et les valeurs seraient une liste d'objets événements appartenant à cette catégorie. Ainsi, lorsqu'un utilisateur recherche des événements dans une catégorie donnée, on peut simplement récupérer la liste d'objets événements associés à cette catégorie à partir du dictionnaire.

Pour faciliter la recherche d'événements par les utilisateurs de notre application, nous utiliserons un dictionnaire pour stocker les événements en fonction de leur emplacement géographique. Les clés du dictionnaire seraient les noms des lieux (par exemple, villes, quartiers) et les valeurs seraient une liste d'objets événements associés à ce lieu. Lorsque les utilisateurs recherchent des événements à proximité de leur emplacement actuel, nous pouvons simplement récupérer la liste d'objets événements associés au lieu correspondant à leur position à partir du dictionnaire.

L'une de ces structures de données sera entièrement programme par nous. Il s'agira d'une liste chainée. Les listes chainées peuvent être redimensionnées dynamiquement en ajoutant ou en supprimant des éléments à l'intérieur de celle-ci. Elles utilisent donc un espace en mémoire pour les éléments présents à la différence des tableaux qui doivent réserver un espace en mémoire fixe. Il est bien important de garder en

référence la tête de la liste, ainsi que chaque élément a une référence à l'élément qui le suit.

Nous utiliserons cette liste pour stocker les événements d'un utilisateur. Nous stockerons d'abord dans la liste chainée les événements en ordre chronologique selon l'ordre dans lequel ils seront ajoutés. Chaque nouvel événement sera rajouté à la fin de la liste. Nous créerons des méthodes pour ajouter et supprimer de nouveaux événements, ainsi que pour rechercher des événements selon certains critères. Nous implémenterons des méthodes nous permettant de trier les événements selon leur type, le lieu et selon la date de ceux-ci.

Nous tenterons d'implémenter une fonction "mergeSort()" qui divise la liste en deux sous-listes égales, triant chacune de manière récursive et fusionne les éléments triés pour obtenir une liste triée finale. Cette méthode est très efficace pour trier une grande quantité de données.

Voici les structures de données externes qui pourraient être utilisées dans ce projet:

Firebase Realtime Database: Firebase Realtime Database est une base de données en temps réel développée par Google qui permet de stocker et de synchroniser les données en temps réel entre les clients et le serveur. Cela sera utile pour stocker les données sur les événements, les informations utilisateur et les conversations.

Nous utiliserons l'API Google Maps pour afficher les lieux d'événements sur une carte, en utilisant les coordonnées géographiques stockées dans Firebase pour chaque événement. Cela permettra aux utilisateurs de visualiser facilement où se trouvent les événements et de planifier leur itinéraire en conséquence. Pour implémenter cette fonctionnalité, nous stockerons les coordonnées géographiques de chaque événement dans Firebase. Ensuite, nous utiliserons l'API Google Maps pour récupérer ces coordonnées et les afficher sur une carte interactive dans l'application. Pour cela, nous utiliserons la librairie Google Maps SDK pour Android. Nous ajouterons un marqueur pour chaque événement sur la carte. Vous pouvez également permettre aux utilisateurs de zoomer et de faire défiler la carte pour explorer d'autres lieux. Enfin, nous pourrons ajouter des informations supplémentaires sur chaque événement en cliquant sur le marqueur, tel que le nom de l'événement, la date et l'heure, et un lien vers la page de l'événement dans l'application.

ALGORITHMES:

Algorithme de Jaccard : cet algorithme permet de comparer la similarité et la diversité entre deux échantillons. C'est un algorithme de statistique, qui repose sur l'intersection et l'union de deux matrices. Le calcul de l'union des deux matrices nous permet de récupérer la taille des valeurs uniques des deux matrices et l'intersection avec le

nombre de valeurs en commun. En divisant l'intersection par l'union, on obtient alors un chiffre compris entre 0 et 1. Ce chiffre est ensuite facilement transportable en pourcentage.
Dans notre application, il permettra de comparer les points d'intérêts d'un utilisateur avec ceux d'un autre utilisateur. Ainsi on obtiendra un pourcentage de correspondance, qui nous permettra de proposer de meilleures suggestions.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Calcul des matrices

Algorithme de Boyer-Moore : L'algorithme de Boyer-Moore est un algorithme de recherche de sous-chaine de caractères (modèle) qui peut être utilisé pour vérifier si une chaine de caractères contient un ou plusieurs modèles appartenant à une liste de mots. Cet algorithme de recherche est très efficace en termes de performances. Contrairement à un algorithme plus naïf, cet algorithme effectue une recherche avec un décalage, ce qui lui permet de faire moins de comparaisons. L'algorithme parcourt les chaines de caractères de droite à gauche et non de gauche à droite, avec la possibilité d'avoir un pas plus important que de parcourir la liste lettre par lettre. Elle utilise deux tables de saut pour accélérer la recherche, en évitant de parcourir le texte pour des positions qui ne peuvent pas correspondre au modèle recherché. Dans notre application, son utilisation nous permettrait de vérifier le contenu des descriptifs des évents créés par les utilisateurs. Ainsi on pourrait facilement comparer le contenu avec une liste de mots interdits et cela même si l'utilisateur venait à écrire sans utiliser d'espaces.

Nous utiliserons un algorithme pour donner des suggestions d'usager en fonction de leur proximité. Pour cet algorithme, nous utiliserons la formule de Haversine pour mesurer de distance géodésique entre deux points sur une sphère.

Finalement nous tenterons d'implémenter l'algorithme mergesort(), que nous avons mentionné plus haut. Cet algorithme a une complexité temporelle de O(n log n) dans le pire des cas, ce qui le rend très efficace pour trier une grande quantité de données. Toutefois, il nécessite un espace en mémoire supplémentaire pour stocker les deux moitiés de liste lors de la récursivité, ce qui pourrait être un inconvénient dans un certain cas.

DIFFÉRENT PARADIGME DE PROGRAMMATION DONT L'ORIENTÉ OBJET ET SES CONSTITUANTS:

Dans le contexte de notre projet de réseau social, la POO sera très utile pour modéliser les différents éléments de notre application. Par ses différents concepts, la POO répond aux attentes pour rendre une application maintenable, plus sécurisée et plus testable. De plus elle permettra à chaque membre de l'équipe de travailler sur des fonctionnalités différentes sans impact sur le reste du code.

Dans les concepts fondamentaux de la POO, on retrouve notamment :

- L'encapsulation : Lors de la création de nos différents objets, certaines données ne doivent pas être connues de tous. Pour rendre notre code plus sécurisé, certaines données doivent rester privées et doivent être accessibles via des accesseurs. Une bonne pratique pour éviter la manipulation non autorisée de certaines données sensibles comme le nom ou l'adresse de l'utilisateur.

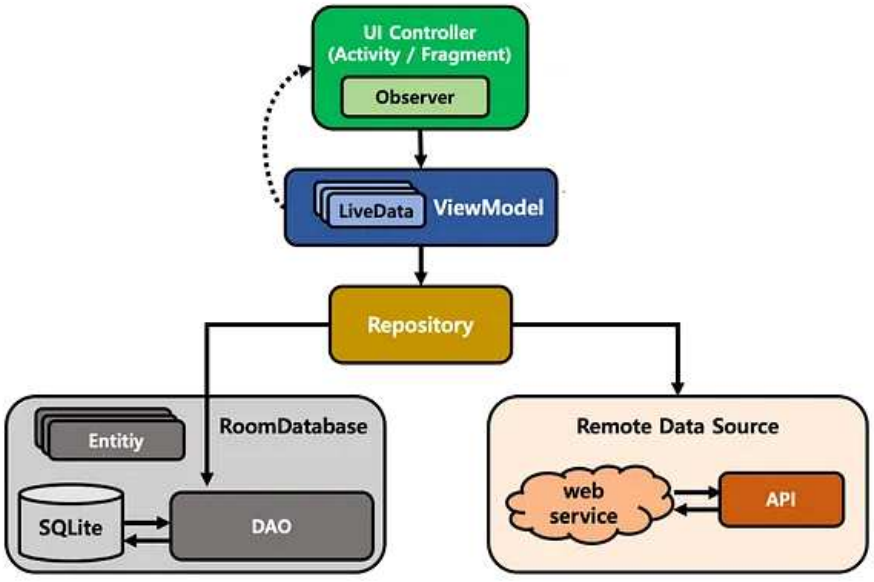
- L'abstraction : Pour notre application, différentes logiques internes peuvent être masquées pour une programmation plus simple. Par exemple, nous pourrons englober la logique entourant FireBase dans une classe et pouvoir la rendre accessible aux autres éléments de notre application. L'abstraction nous permettra de séparer les fonctionnalités des différents composants de notre application. On pourra réutiliser le code ou les algorithmes dans des contextes différents. Par exemple, la logique pour les suggestions d'amis pourra s'appliquer aussi bien dans une autre application. De plus, l'abstraction permettra de mieux maintenir notre code, en pouvant changer des parties de notre logique sans qu'elle impacte le reste du programme.

- L'héritage : ce concept nous permettra de faire hériter d'une classe mère, des attributs ou des méthodes à une classe enfant et éviter une redondance du code. Dans le cadre de notre application de réseau social, la classe mère contiendra toutes les fonctionnalités communes et les autres classes pourront hériter de ces méthodes et les rendre plus spécifiques en fonction des cas d'usages. Par exemple pour une classe utilisateur on pourrait créer une base User avec des attributs et des méthodes communes à tous les utilisateurs. Ainsi nous pourrons créer à partir de cette classe différent type d'utilisateur, par exemple : super User, suspendu, normal, etc. cela réduira la duplication de code et cela nous permettra d'avoir une structure plus modulaire et claire. De plus, la centralisation des attributs dans une super classe permet une meilleure maintenance.

- Polymorphisme : cette notion va nous permettre de générer des interfaces qui seront implémentées dans nos classes. Le but est d'avoir des méthodes communes qui seront définies au sein de chaque classe. Le but est que le programme ne se soucie pas de savoir à quelle classe appartient la méthode. Dans notre projet, nous utiliserons essentiellement du polymorphisme hérité. Nous créerons ainsi un GeneriqueListAdapter qui nous permettra d'afficher différents recyclerView dans les activités de notre application. Ces listes se répètent, mais se distinguent puisqu'elles affichent différents types d'information. En créant une GeneriqueListAdapter, nous évitons une redondance de code tout en appliquant le concept de polymorphisme.

PATRONS DE CONCEPTION ET ARCHITECTURE LOGICIELLES:

Patron MVVM (Modèle-Vue-Vue Modèle) : C'est un modèle architectural qui permet de séparer la logique du modèle et de la vue dans une application. Cette architecture a été adaptée par Google pour répondre aux attentes des développeurs Android. Sous le nom JetPack, Google préconise l'utilisation de librairies pour mieux s'adapter à l'environnement Android. Nous utiliserons ce design pour améliorer la structure, la maintenabilité et la testabilité de notre code et quelques librairies de l'écosystème JetPack.



Concept de base du MVVM

Le Modèle est la couche qui gère les données et la logique applicative de l'application. Dans notre cas, le modèle contiendra toute l'information sur les événements, les utilisateurs, les commentaires, etc.

La vue est la couche qui présentera les informations à l'utilisateur. C'est la partie visible pour l'utilisateur et c'est elle qui reçoit toutes ses entrées. Notre vue sera composée des composants graphiques d'Android comme les fragments, les zones de textes ou les boutons. Elle affichera l'information provenant du ViewModel par rapport aux événements, a la messagerie, aux demandes d'amis, etc.

Le ViewModel s'occupera de relier le Modèle et la Vue. Il contient la logique métier de l'application. Il est responsable de mettre à jour la Vue lorsqu'il y a des changements dans le Modèle, et pour envoyer les entrées utilisateur au Modèle.

LiveData : C'est une classe qui observe les données, mais contrairement aux autres observateurs, il s'adapte aux applications Android. En effet, le LiveData permet de tenir compte du cycle de vie des activités ou des fragments des applications. En tenant compte du cycle de vie, il permet d'observer que les composants actifs. Il suit le patron de conception Observateur, ce qui permet de ne plus avoir à besoin de mettre à jour l'interface graphique à chaque changement de données.

Data Binding : c'est une librairie faisant partie de JetPack, elle permet de faire un lien entre les composants graphiques Android en XML et notre activité codé en Kotlin. Grâce à cette librairie, on ne déclare plus les composants graphiques en XML dans la logique codée en Kotlin. En instanciant une simple variable, on peut accéder aux valeurs de nos composants graphiques. Ainsi les layouts peuvent gérer leurs propres données et être facilement réutilisables grâce au ViewModel.

Design Pattern :

Patron Observer: ce patron sera utilisé pour envoyer des notifications concernant les évents, les messages, ou les demandes d'amitiés.

Ce patron sera utilisé pour créer une interface utilisateur dynamique et réactive. Par exemple pour toutes les opérations de login, sign in, édit profil, édit évent, les données seront ensuite rafraîchies automatiquement sans qu'il ait besoin de rafraichir manuellement la page.

En utilisant le patron observer on pourra notifier automatiquement l'utilisateur lorsqu'un nouveau message est reçu.

Patron Stratégie (Strategy): Ce modèle est utile pour permettre à l'application de choisir dynamiquement la méthode de traitement appropriée en fonction des besoins de l'utilisateur. Dans une application de rencontre, cela peut aider à sélectionner les critères de correspondance en fonction des préférences de l'utilisateur.

De plus, le Design Pattern Stratégie permet de mettre en place des fonctionnalités de personnalisation pour les utilisateurs, qui peuvent choisir entre différentes stratégies de correspondance en fonction de leurs préférences. Cette approche peut aider à améliorer l'expérience utilisateur en offrant un choix plus personnalisé et en augmentant la satisfaction globale de l'utilisateur.

On utilisera ce Design Pattern afin de proposer une différente stratégie de suggestion d'amis aux utilisateurs. Il y aura des stratégies différentes, avec un ensemble distinct de critères de correspondance, d'algorithmes de recommandation et de processus de mise en correspondance.

Lors de la configuration de son profil, l'utilisateur pourra choisir une stratégie de correspondance basée sur les intérêts et les habitudes de navigation. Autrement s'il cherche à faire des rencontres en se basant sur des critères plus généraux tels que l'âge, la localisation géographique ou les centres d'intérêt, d'autres stratégies pourraient être utilisés.

Patron État (State) : Ce design permet à un objet de modifier son comportement en fonction de son état interne. Dans le contexte de notre application, nous utiliserons ce design pour différencier les différents états d'un utilisateur tels que Super User, Normal User, SuspendUser...

Le modèle État repose sur une abstraction commune qui permet de définir les différents états possibles d'un objet. Dans notre cas, nous utiliserons une interface *State* qui définira les différentes méthodes et propriétés que chaque état d'utilisateur doit implémenter. Par la suite, chaque état possible peut être représenté par une classe concrète qui implémentera l'interface. Ces classes concrètes contiendront la logique spécifique de chaque utilisateur telles que les fonctionnalité et restriction associées à son état.

Ainsi, notre classe User, utilisera une instance de l'interface pour encapsuler son état interne et déléguer les appels de méthode appropriée a l'objet *State.* Cela permettra à la classe Users de modifier son comportement en fonction de son état interne sans avoir à écrire une logique conditionnelle complexe.

ÉLÉMENTS DE L'ERGONOMIE LOGICIELLE:

Dans le contexte de notre projet, l'ergonomie logicielle est un élément important pour assurer une expérience utilisateur agréable et intuitive. Voici quelques éléments d'ergonomie logicielle à considérer :

- La lisibilité : Il est important d'utiliser une typographie lisible et des couleurs de texte contrastées pour permettre une lecture facile des informations affichées sur l'interface utilisateur.
- La navigation : notre application offrira une navigation simple et intuitive, avec une hiérarchie claire des menus et des fonctionnalités.
- La cohérence : Nous maintiendrons une cohérence graphique et de design entre les différentes pages et fonctionnalités de l'application.
- La réactivité : notre application sera réactive et répondra rapidement aux actions de l'utilisateur, afin de donner l'impression d'une utilisation fluide et rapide.
- La rétroaction : notre application doit fournir une rétroaction claire à l'utilisateur pour l'informer de l'état de la tâche en cours d'exécution, ainsi que des erreurs éventuelles.
- La simplicité : L'interface utilisateur devra être simple et épurée, avec un nombre minimal d'éléments affichés à l'écran pour ne pas surcharger l'utilisateur.
- L'accessibilité : notre logiciel sera accessible à tous les types d'utilisateurs, y compris ceux ayant des besoins spécifiques en termes d'accessibilité.
- L'intuitivité : L'application doit être conçue de manière à être facilement compréhensible et utilisable par l'utilisateur, sans avoir besoin de consulter un manuel d'utilisation.

En appliquant ces principes d'ergonomie logicielle à notre application de réseau social, nous pourrons améliorer l'expérience utilisateur et rendre l'application plus facile à utiliser pour nos utilisateurs.

STRATÉGIES DE GESTION DE PROJET:

Pour ce projet nous utiliserons la méthodologie Agile. C’est une méthode qui est beaucoup utilisée dans le développement de logiciel. Cette méthode permet une planification flexible, des changements de direction rapides en réponse aux rétroactions. Puisque l’on fonctionne par sprints divisés en courte sous tache, cela permet une livraison plus rapide des fonctionnalités et des mises à jours. De plus cette méthode encourage grandement la collaboration pour identifier et faire face aux

problématiques de développement. Elle permet de gagner du temps dans la production d'un logiciel et offrir une amélioration continue sur le produit final.

Pour gérer notre projet à l'aide de la méthode Agile nous débuterons par une planification globale du projet. Nous établirons une vision claire du projet, des objectifs et des attentes. Nous définirons la portée du projet et nous ferons la création d'une feuille de route.

Ensuite, nous définirons les sprints qui permettront de préciser les fonctionnalités spécifiques sur lesquelles nous travaillerons sur une période donnée. Les sprints sont généralement d'une durée de 1 à 4 semaines, dans notre cas elles seront d'une durée de 3 semaines. Au début de chaque sprint, nous établirons un objectif clair à accomplir. Pendant la durée du sprint, nous tiendrons des réunions quotidiennes afin de discuter des progrès, des problèmes, des difficultés et des priorités. À la fin du sprint nous reviendrons sur les fonctionnalités terminées afin d'obtenir une rétroaction. Nous discuterons aussi des points positifs et de ceux à améliorer.

En utilisant des sprints, nous pourrons nous concentrer sur des objectifs clairs et mesurables tout en fournissant des résultats concrets à la fin de chaque sprint. Cela nous permettra de nous adapter rapidement aux changements et aux commentaires reçus tout en s'assurant de la qualité de notre produit final.

Nous répéterons ce cycle jusqu'au terme de notre projet tout en nous adaptant aux problèmes rencontrés. Il sera important de suivre les données de performance pour comprendre les progrès et les lacunes de notre projet.

EXPRESSION RÉGULIÈRE:

Afin de valider certains champs dans notre projet, nous utiliserons des expressions régulières notamment :

- *Pour les Courriels :*

"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$"

Cette expression régulière vérifie si l'adresse couriel est valide en vérifiant si elle contient un ou plusieurs caractères alphanumériques, suivi d'un symbole "@", suivi d'un nom de domaine qui contient également des caractères alphanumériques et des traits d'union, suivi d'un point, suivi d'un suffixe de domaine (comme .com, .net, .fr, etc.)

- *Pour les Mots de passe :*

"^(?=.["A-Za-z])(?=.*\d)[A-Za-z\d]{8,}$"*

On vérifie si le mot de passe contient au moins 8 caractères, au moins une lettre et au moins un chiffre.

- *Pour les Événements :*

"^[a-zA-Z0-9\s.'-]{1,100}$"

Pour vérifier si le nom de l'événement contient entre 1 et 100 caractères, qui peuvent être des lettres, des chiffres, des espaces, des apostrophes, des tirets et des points.

- *Pour le Nom et prénom :*

"^[a-zA-Z\s.'-]{1,50}$"

Cette expression régulière vérifie si le nom ou le prénom contient entre 1 et 50 caractères, qui peuvent être des lettres, des espaces, des apostrophes, des tirets et des points.

- *Pour les Adresses :*

"^[a-zA-Z0-9\s.,'-]{1,200}$"

Cette expression régulière vérifie si l'adresse contient entre 1 et 200 caractères, qui peuvent être des lettres, des chiffres, des espaces, des virgules, des points, des apostrophes, des tirets et des traits d'union.

MATHÉMATIQUE:

Pour ce projet nous utiliserons au moins une équation mathématique qui utilise plus que les quatre opérateurs fondamentaux (addition, soustraction, multiplication et division).
Notamment dans la formule de Haversine qui utilise des fonctions trigonométriques pour calculer deux points sur une sphère en prenant en compte la courbe de la surface.
Elle prend en entrée les coordonnées géographiques de deux points, exprimées en latitude et longitude, et donne en sortie la distance en kilomètres entre ces deux points.
La formule est définie comme suit :

$$d = 2 * R * arcsin(sqrt(sin^2((lat_2 - lat_1)/2) + cos(lat_1) * cos(lat_2) * sin^2((lon_2 - lon_1)/2)))$$

Où :

d : distance entre les deux points en kilomètres
R : rayon de la sphère (en général, le rayon moyen de la Terre est utilisé, soit environ 6 371 km)
lat_1 et lat_2 : latitudes des deux points en radians
lon_1 et lon_2 : longitudes des deux points en radians

VEILLE TECHNOLOGIQUE :

Pour ce projet, nous allons explorer la technologie Kotlin. Il s'agit d'un langage de programmation conçu par JetBrains et qui est une alternative au langage Java pour le développement d'applications Android.

Depuis 2019, Google propriétaire d'Android recommande et souhaite que les développeurs d'applications utilisent Kotlin plutôt que Java. Désormais, la documentation officielle met l'accent sur ce nouveau langage. Après quelques exemples, on comprend mieux l'intérêt de Google pour ce langage. En effet, bien que Java soit performant, il est également très verbeux. Kotlin permet de réduire considérablement le nombre de lignes codées pour le même résultat. Ainsi, Kotlin permet d'éviter des erreurs courantes comme le NullPointerException, en proposant de déclarer une propriété comme nulle ou non et de gérer l'exception à la compilation. On réduit notre code pour se concentrer sur l'essentiel et être davantage productif. À la lecture, cela se remarque aussi par un code plus lisible et plus concis.

De plus, Kotlin et Java sont compatibles, ce qui signifie qu'on peut coder en Java dans un programme Kotlin et inversement. Bien que nous éviterons de mélanger les deux langages dans notre projet, nous aurons quand même l'avantage de pouvoir nous documenter aussi bien en Java qu'en Kotlin pour l'implémentation de fonctionnalités.