RAPPORT ANDROID STUDIO

VCAL

CHARRIER Killian

VERPOORTE William

VCAL

SOMMAIRE

PRÉSENTATION

L’APPLICATION

VCAL est une application de génération de mots de passe robustes et sécurisés qui agrège les informations provenant des divers capteurs de votre téléphone pour créer un mot de passe unique. L'application exploite notamment les conditions météorologiques de l'endroit où vous vous trouvez, ainsi que l'heure à laquelle vous avez branché votre téléphone pour la dernière fois. Pour ajouter encore de la complexité au mot de passe, l'application, à l'aide d'une notification permanente, collecte une fois par heure des informations du système, telles que la quantité de mémoire RAM disponible. Enfin, la dernière variable dans la génération des mots de passe est un mot que l'utilisateur renseigne, car, en fin de compte, il constitue la plus grande source d'aléatoire.

L'application se veut simple. Il suffit de renseigner un mot aléatoire et de lancer la génération du mot de passe, qui vient remplacer les deux éléments précédents. Cette simplicité découle du fait qu'il est toujours pénible de créer un nouveau mot de passe, et l'application ne doit pas ajouter à cette contrainte. De plus, elle n'utilise que des éléments d'Android, ce qui la rend plus fiable avec un meilleur temps de réponse.

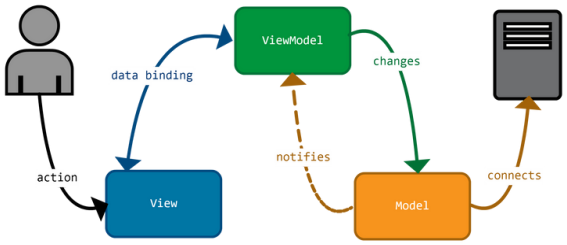
VCAL

Organisation du code

Le projet a été développé selon l'architecture MVVM (Modèle-Vue-Modèle), qui est préconisée par Google comme l'architecture la plus utilisée dans le développement d'applications Android, notamment en raison de ses nombreux avantages. Dans notre projet, cette architecture se distingue par sa facilité de compréhension des rôles de chaque composant. Bien qu'elle soit souvent présentée comme la plus utilisée, les exemples de projets l'implémentant sont rares, et la plupart ne sont plus fonctionnels. Cependant, notre implémentation, bien que probablement non optimale, a le mérite de fonctionner.

Dans le cadre de notre projet, nous utilisons des vues qui sont des layouts classiques. Une vue peut également avoir un modèle de vue (ViewModel), notamment lorsque la vue doit afficher des informations provenant de la base de données ou vice versa. Les ViewModel récupèrent les informations depuis la base de données, les traitent pour les adapter au format approprié, et les transmettent à la vue. De plus, nous utilisons le databinding, ce qui simplifie les échanges entre les vues et les ViewModels. En effet, dans la vue, nous renseignons directement les attributs du ViewModel que nous voulons afficher. Cette relation fonctionne dans les deux sens : la vue se met à jour automatiquement si l'attribut change, et elle peut modifier automatiquement un attribut avec un EditText, par exemple. Le databinding nous permet de nous affranchir totalement des listeners et de la mise à jour manuelle des TextView, ces actions étant effectuées implicitement, simplifiant ainsi grandement le code.

Pour la relation entre le ViewModel et le modèle, nous utilisons la bibliothèque Room. Cette bibliothèque nous permet de créer des modèles de données à enregistrer dans la base de données, et elle offre également des DAO (Data Access Objects) nous permettant de spécifier les requêtes nécessaires pour notre modèle. Les ViewModels peuvent ainsi appeler les méthodes des DAO pour afficher des valeurs de la base de données ou inversement. Avec tous ces éléments, la boucle est bouclée, et nous avons effectivement une architecture MVVM.



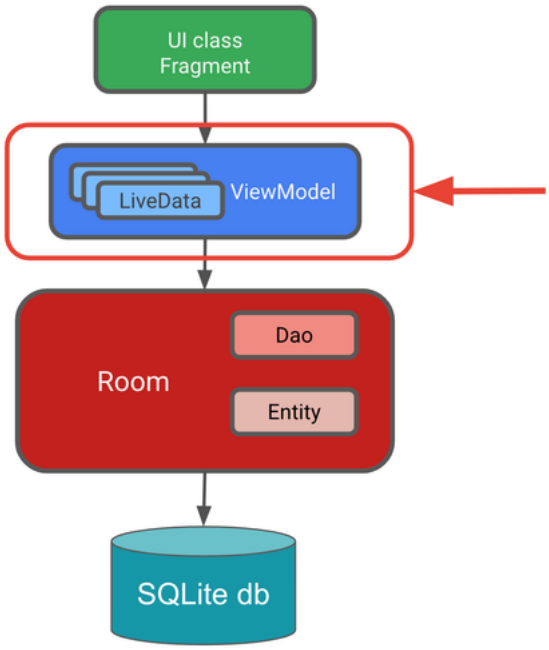
SQLite Android

Comme mentionné dans la section précédente, notre projet fonctionne avec la bibliothèque Room, qui simplifie l'accès à la base de données MySQL de l'application à tous les niveaux, de la création de l'instance de la base aux tables, etc. Notre base de données est simple, comprenant seulement deux tables avec leurs modèles et DAO respectifs, tous définis dans le fichier principal de la base.

Le premier modèle traduit le moment où le téléphone a été branché au secteur. Il est donc composé seulement d'un timestamp et d'un identifiant. Le deuxième modèle représente les relevés des informations système effectués toutes les heures, traduisant spécifiquement la mémoire libre à un moment donné. La table ne contient donc que deux éléments accompagnés d'un identifiant.

En ce qui concerne les DAO de ces modèles, ils sont tout aussi simples avec une méthode pour insérer une entrée et la supprimer, ainsi qu'une méthode pour récupérer la dernière entrée. Cette dernière méthode utilise le timestamp pour déterminer la dernière entrée, et c'est d'ailleurs celle qui revêt le plus d'importance.

La base de données nous permet de stocker des données de manière persistante dans notre projet, mais elle est surtout nécessaire car nous devons conserver des informations même lorsque l'application n'est pas lancée, ce qui aurait été impossible autrement.



Service Android

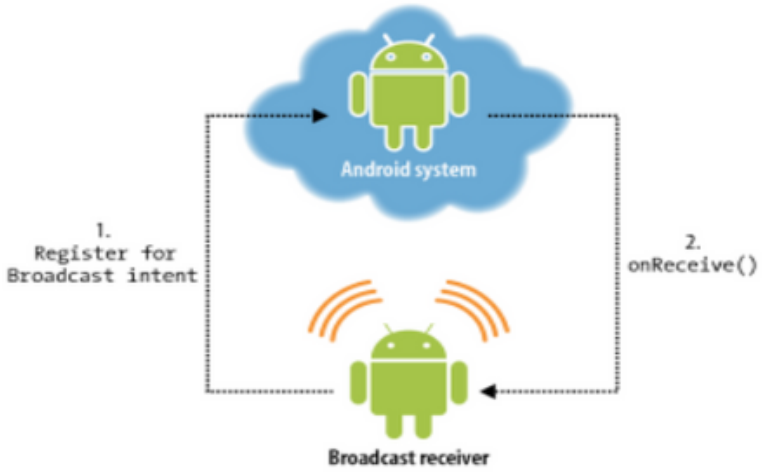
Notre application doit utiliser un service Android, car nous souhaitons récupérer toutes les heures la quantité de mémoire RAM disponible, indépendamment du fait que l'application soit lancée ou non. Ce scénario correspond parfaitement à un service, car il permet d'exécuter du code en dehors de notre application principale tout en ayant la possibilité d'ajouter des informations à la base de données. Cependant, après avoir créé notre premier service, nous avons été confrontés à un problème. En effet, depuis les dernières mises à jour d'Android, l'utilisation des services a été restreinte, et nous avons été obligés d'utiliser un autre type de service appelé un "foreground service". Ce type de service est conçu pour fonctionner en continu sur une période plus longue. Comparé à un service classique, il affiche une notification continue informant de son existence à l'utilisateur.

Avec ce nouveau service, tout fonctionne parfaitement. Au début, nous avions simplement un journal (log) qui était envoyé, mais nous avons ensuite mis en place une tâche qui se lance toutes les heures. Il s'agit d'un handler : il nous suffit de lui donner une tâche et une durée pendant laquelle il va attendre avant de refaire la tâche. Notre tâche est simple : nous récupérons simplement la taille de la RAM libre et l'ajoutons dans notre base de données. Cependant, lors de l'utilisation de notre base, une erreur survient, indiquant qu'on ne peut pas effectuer ce type d'opération dans le thread principal. Pour résoudre ce problème, nous avons utilisé un executor fourni par Java, car il est très simple à utiliser et a fonctionné du premier coup, contrairement à tout le reste lors du développement. Enfin, à l'arrêt du service, nous arrêtons également le handler pour éviter toute exécution non souhaitée du point de vue d'Android.

Récupération de l'heure de la mise en charge

Notre application récupère et stocke l'heure à laquelle le téléphone a été branché sur le secteur, ce qui constitue un moyen très simple d'obtenir une donnée aléatoire. Cependant, dans les faits, cette fonctionnalité qui aurait dû être simple à développer nous a posé quelques difficultés. Pour la réaliser, nous avons dû créer un BroadcastReceiver, un composant fourni par Android qui, une fois enregistré dans le système, reçoit automatiquement des intentions du système et les traite selon notre implémentation. Dans notre cas, nous avons créé un BroadcastReceiver qui, lorsqu'il reçoit une intention de type "power connected", enregistre dans la base de données le moment de la réception (toujours à l'aide d'un executor).

Théoriquement, il suffit ensuite d'enregistrer notre BroadcastReceiver dans notre manifeste d'application avec le bon filtre d'intention pour que le système nous les envoie automatiquement. C'est là que les choses se corsent. En effet, quelle que soit la situation, le BroadcastReceiver ne recevait rien et ne montrait aucun signe de vie. Après avoir fouillé dans les logs, nous avons découvert une erreur indiquant que notre BroadcastReceiver n'était pas autorisé à recevoir l'intention "power connected". Cette erreur est apparue avec les dernières versions d'Android qui limitent les tâches en arrière-plan, et c'est la cause de notre blocage. Pour résoudre ce problème, il faut lancer le BroadcastReceiver depuis notre code, ici depuis notre activité principale, et tout fonctionne. Enfin, sa suppression se fait automatiquement par le système.

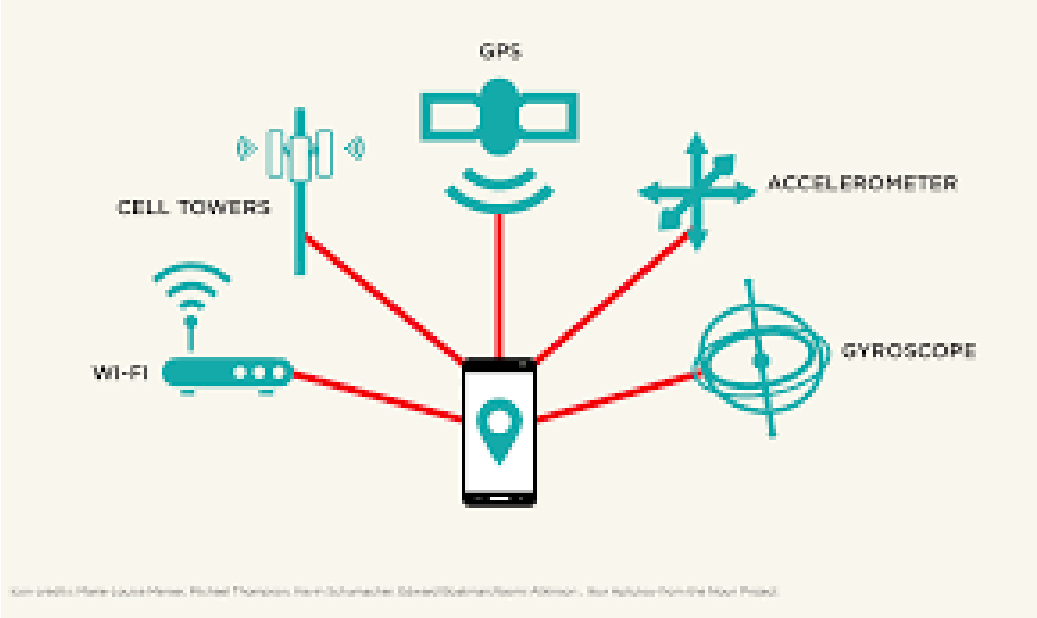


Utilisation des coordonnées

La géolocalisation du téléphone représente une composante essentielle de notre application, notamment en ce qui concerne la récupération précise de la longitude et de la latitude. Toutefois, pour parvenir à cette fonctionnalité, il est impératif d'obtenir préalablement l'accès aux permissions de géolocalisation du téléphone, un processus qui s'effectue en sollicitant l'autorisation de l'utilisateur.

Une fois les permissions acquises, notre application tire parti de la classe FusedLocationProviderClient, une interface fournie par Google Play spécifiquement conçue pour simplifier et optimiser la gestion des données de localisation. Cette classe exploite diverses sources, telles que le GPS, le Wi-Fi et les données cellulaires, pour garantir une localisation fiable et précise.

En utilisant cette approche, nous sommes en mesure de récupérer avec efficacité la longitude et la latitude du téléphone. Ces données géographiques constituent un élément clé pour nos futures fonctionnalités, permettant une personnalisation et une adaptabilité accrues de l'application en fonction de la position de l'utilisateur.



Récupération des données météo

La récupération des données météo à l'emplacement se réalise à travers deux fonctionnalités essentielles. Tout d'abord, nous exploitons les coordonnées GPS préalablement récupérées du téléphone. Ensuite, nous faisons appel à une API nommée OpenWeather. Cette API offre une richesse de données météorologiques complètes pour des localisations spécifiques. Il nous suffit donc d'utiliser les coordonnées GPS obtenues et de les intégrer dans l'API OpenWeather.

Ce processus s'effectue en utilisant l'URL fournie par l'API. En soumettant cette URL, l'API OpenWeather nous renvoie les données météorologiques souhaitées sous forme de fichier JSON. À ce stade, il ne nous reste plus qu'à utiliser ces données de manière asynchrone ou à travers des threads afin d'accéder aux informations météo spécifiques à notre localisation.
Cependant, cette requête, bien que fonctionnelle, a posé problème plus tard dans le développement de l'application. En effet, lorsque l'utilisateur souhaite générer un mot de passe, nous avons besoin des données météo, même si cela implique d'attendre la fin de la requête. Pour résoudre ce défi, nous avons opté pour l'utilisation de CompletableFuture. Cela nous permet d'appeler la méthode `get` sur le CompletableFuture, laquelle est bloquante. Ainsi, nous avons réussi à obtenir les données météo de manière synchrone, répondant ainsi aux besoins spécifiques de la génération de mots de passe.

Nous avons également rencontré des problèmes similaires lorsqu'il a été nécessaire d'accéder aux données de la base de données, car les requêtes doivent être effectuées dans un autre thread pour éviter de bloquer l'interface utilisateur. Pour résoudre ce problème, nous avons fait usage de notre propre executor en conjonction avec un CountDownLatch. Ce dernier permet de mettre en pause l'exécution jusqu'à ce que la requête soit terminée. Lorsque la requête est accomplie, le CountDownLatch est décrémenté à zéro, débloquant ainsi la fonction await qui a été appelée sur le CountDownLatch. Ce mécanisme nous a permis de gérer de manière efficace les opérations asynchrones et synchrones dans notre application, assurant un flux de travail fluide et réactif pour l'utilisateur.

Comment fonctionne l’application

L'application adopte une structure composée d'une activité divisée en deux fragments distincts. Le premier fragment se présente sous la forme d'un bouton, déclenchant le processus de génération du mot de passe à partir des données météo obtenues et des informations stockées dans la base de données. L'objectif est de créer un mot de passe entièrement aléatoire, spécifique à l'appareil de l'utilisateur.

Le second fragment, quant à lui, est une zone de texte qui affiche le résultat après la génération du mot de passe. Ainsi, l'utilisateur peut facilement utiliser le mot de passe généré par l'application à sa convenance.

La logique de génération du mot de passe est délibérément simpliste, étant principalement utilisée comme un prétexte pour explorer les fonctionnalités d'Android. Pour générer le mot de passe, l'application récupère les informations météo, le dernier taux de mémoire RAM libre du téléphone, l'heure à laquelle le téléphone a été branché pour la dernière fois, et enfin, un mot saisi par l'utilisateur. Ces données sont ensuite concaténées pour former une chaîne de caractères, qui constitue le mot de passe affiché.

Il est important de souligner que le mot de passe se génère lors du clic sur le bouton du premier fragment. Une fois généré, le mot de passe est transmis au deuxième fragment à travers un mécanisme, souvent appelé "fragment résultat", pour être affiché à l'utilisateur.

Merci pour votre attention !

VCAL

CHARRIER Killian
VERPOORTE William

VCAL