Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

HappyBaby – Projet de Bachelor – 2021/2022

Rapport de projet

Cédric Mujynya

Experts : *Sylvain Egger, Dominique Marmy*
Professeur : *Pascal Bruegger*
Assistant : *Arton Hoxha*

Version 1.0

Historique de révisions :

N°	*Auteur*	*Date*	*Remarques*
1.0	Cédric Mujynya	15.07.2022	Publication initiale

Fribourg, le 15 juillet 2022

Sommaire

Avec l'apparition et la popularité d' applications tels que Facebook, Twitter ou encore Instagram, il existe aujourd'hui de multiples réseaux sociaux spécialisés. Il y a par exemple Tinder et toutes ces autres applications similaires, qui se focalisent sur l'amour. Dernièrement, Donald Trump a lui aussi sorti son propre réseau social, particulièrement adapté pour ses supporters.

D'un autre côté, des applications visant à mettre en relation des particuliers par le biais de services proposés par l'entreprise ont également vu le jour. Uber Eats ou encore Too Good To Go dans le domaine de l'alimentation, Airbnb pour l'hôtellerie, etc.

Actuellement en Suisse, il n'existe pas d'application spécialisée dans la mise en relation entre des parents cherchant une nounou pour s'occuper de leur(s) enfant(s). Si un parent cherche une maman de jour, que ce soit pour un soir ou de manière régulière, les techniques les plus utilisées restent le bouche-à-oreille ou encore des groupes spécialisés sur des réseaux sociaux.

De plus, les différentes crèches disponibles n'arrivent pas toujours à satisfaire toutes les demandes [1] et le contexte économique actuel pousse les deux parents à travailler simultanément. Le coronavirus a bien sûr changé la donne avec notamment la montée en popularité du télétravail, mais une part importante des parents dispose tout de même d'une activité professionnelle, comme le montre le graphique ci-dessous [2] :

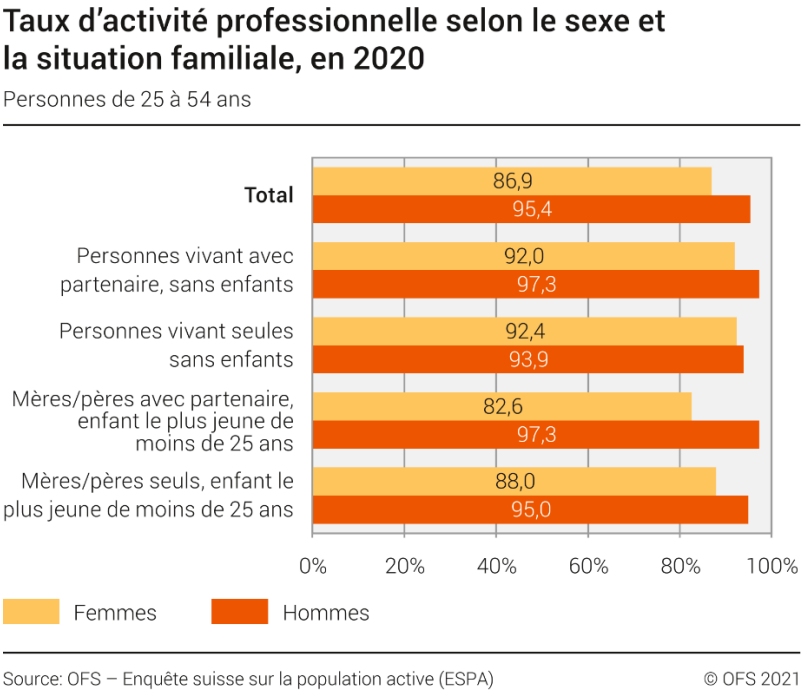

Figure 1 - Taux d'activité professionnelle selon le sexe et la situation familiale en Suisse, 2020 - repris de [2]

Ce projet de Bachelor a pour objectif de réaliser une application mobile permettant de connecter parents et nounous, à mi-chemin entre un réseau social et un service de mise en relation. Un prototype de l'application a déjà été réalisé durant le projet de semestre précédent, sous forme de *Proof of Concept* dont le but était de montrer l'intérêt et le potentiel d'une telle application.

Table des matières

1 Introduction

De nos jours, le smartphone est un outil tellement pratique qu'il est presque devenu indispensable au quotidien. En plus de la fonction de téléphonie et de messagerie, de nombreuses autres applications existent dans un tas de domaines plus variés les uns des autres.

Parmi ces services, certaines entreprises ont eu l'idée de réaliser un site internet compatible avec une navigation sur Smartphone permettant de rechercher une maman de jour (une nounou) pour un soir ou périodiquement. Avec le contexte économique actuel, ce sont des applications très intéressantes sur le papier car répondant à une demande sur le marché. En Suisse, des sites internet comme Yoopies [3] ou encore Topnanny [4] sont disponibles pour mettre en relation des parents et des nounous, cependant aucune plateforme n'a vraiment su s'imposer sur le marché.

1.1 Contexte

Actuellement, il n'existe pas d'application mobile spécifique à la mise en relation entre parents et nounous populaire en Suisse, que ce soit en Romandie ou du côté alémanique. Pour autant, il existe une demande sur ce marché.

Aujourd'hui, l'on retrouve principalement deux systèmes d'exploitation pour smartphone, Android et iOS. Afin de répondre au mieux à la demande, l'application se doit d'être déployée sur les deux systèmes et donc être « cross-plateforme ». Pour faciliter le développeur, l'on retrouve aujourd'hui divers environnements de développement permettant la création d'application cross-plateforme à partir d'un code commun, incluant même parfois la génération au format web.

Durant le projet de semestre précédent, un prototype de l'application utilisant les technologies Flutter [5] et Firebase [6] a été réalisé. Ce dernier intègre déjà certaines fonctionnalités essentielles au bon fonctionnement de l'application, comme la recherche d'utilisateurs dans un rayon proche ou encore la gestion du profil de l'utilisateur. Malheureusement, toutes les fonctionnalités présentes ne sont pas complètes et il est donc nécessaire de passer en revue et d'améliorer le prototype durant ce projet de Bachelor.

1.2 Etude du prototype

Comme mentionné précédemment, un prototype de l'application mobile HapyBaby a déjà été réalisé dans le cadre du projet de semestre 6. Grâce à cela, il n'est pas nécessaire de réaliser toutes les phases d'analyse, de conception et d'implémentation nécessaire à la réalisation d'une telle application durant ce projet de Bachelor. En effet, il aurait été impossible de délivrer un produit satisfaisant avec le nombre d'heures à disposition.

De plus, du fait d'une mauvaise planification notamment au niveau de l'anticipation du temps nécessaire à la réalisation de certains objectifs, toutes les activités principales du projet de semestre précédent n'ont pas pu être complètement réalisées. Le prototype n'intègre donc pas parfaitement tous les objectifs, comme le montre le tableau ci-dessous repris du projet de semestre précédent [7] :

Objectif	Statut	Remarques
Analyse Frontend	100%	Les technologies et applications similaires ont été étudiées.
Analyse Backend	100%	Les différentes plateformes disponibles ont été analysées.
Structure + implémentation DB	85%	La structure du profil utilisateur n'est pas la même que sur le modèle EA. Le profil des nounous n'a pas été implémenté.
Cas d'utilisation	30%	Les cas d'utilisation ont été présentés mais n'ont pour la plupart pas été étudiés.
Connexion/création du compte	50%	Les écrans de connexion n'ont pas été traduits. Il n'est pas possible de modifier ou de supprimer son compte.
Affichage/modification du profil	75%	Certaines informations au sujet des nounou ne sont pas présentes. Il n'est pas possible de créer plusieurs profils par type de service proposé pour une nounou. Le partage de la localisation par le biais de l'adresse ne fonctionne pas.
Recherche de nounou	95%	Il n'est pas possible de spécifier sa localisation à partir d'une adresse.
Demande de RDV	70%	La périodicité ainsi que le lieu du rendez-vous n'ont pas été implémentés.
Validation du RDV	60%	La nounou ne reçoit pas de notification lors d'un nouveau rendez-vous. Elle ne peut pas proposer de modification ou afficher le profil du parent ayant effectué la demande.
Scénarios de test	10%	Les seuls tests effectués sont ceux réalisés implicitement pour vérifier le bon fonctionnement. Aucun scénario n'a été réalisé. Les tests sont repoussés au projet de Bachelor.
Objectifs secondaires	0%	Aucun objectif secondaire n'a été réalisé.

Progression des objectifs à la fin du projet de semestre précédent [7]

Ces fonctionnalités ne peuvent pas rester partiellement implémentées dans le cadre de ce projet de Bachelor. Il est donc nécessaire dans un premier temps de finaliser l'implémentation des activités du prototype. Pour cela, les objectifs du projet de Bachelor ont été réévalués par rapport aux objectifs initialement prévus. De plus, l'architecture de l'application et la structure des données implémentés actuellement ne profite pas de tous les avantages apportés par Flutter et Firebase. Une réflexion doit donc être apportée sur l'architecture Frontend ainsi que sur la structure et les permissions au niveau du Backend.

En raison du temps supplémentaire requis pour finaliser et améliorer le prototype, certains objectifs initialement prévus pour le projet de Bachelor ont été contraints d'être déplacés en tant qu' objectifs secondaires. Notamment, la conception et l'implémentation des nouvelles fonctionnalités tels que l'intégration d'un outil de messagerie ou d'un système de feedback.

1.3 Objectifs

Ce projet de Bachelor a pour but de reprendre et d'améliorer les fonctionnalités du prototype déjà réalisé afin de produire une application prête à sortir sur le marché. Pour cela, il est dans un premier temps nécessaire de terminer la réalisation des objectifs qui n'ont pas pu être complétés dans le prototype. Dans un second temps, une réflexion devra être faite sur le modèle économique de l'application. Des tests utilisateurs devront également être réalisés en fin de projet.

1.3.1 Objectifs principaux

Cette sous-section liste les objectifs qu'il est impératif de réaliser afin de fournir un produit final répondant aux attentes de ce projet de Bachelor.

1.3.1.1 Refactoring du code au niveau du prototype

Un premier objectif pour ce projet est de revoir l'architecture et la communication entre les différents « objets » présents dans l'application mobile. Actuellement, une architecture se basant sur Bloc [8] est utilisée, mais sans pour autant intégrer le package en question à l'application. Au cours de l'implémentation du prototype, l'architecture n'a pas toujours été respectée et il n'y a donc pas de modèle générique pour récupérer les données. De plus, l'utilisation des flux de mise à jour en temps réel des données de Firebase [9] n'a également pas été implémentée.

Afin d'améliorer et de généraliser le code, il est donc nécessaire de concevoir et d'implémenter une architecture globale à toute l'application et de suivre l'architecture pour les futures implémentations. L'architecture choisie doit notamment profiter de l'avantage des mises à jour en temps réel « *snapshot* » fournie par Firebase lorsque cela est pertinent.

De plus, il n'est actuellement pas possible d'utiliser l'API de Google Maps car aucun compte de facturation n'était associé au projet et que les démarches pour en obtenir un par le biais de l'école avaient été réalisées trop tardivement. Comme un compte de facturation est maintenant disponible pour le projet de Bachelor, cet objectif inclut également l'intégration de l'API Google Maps au sein de l'application HappyBaby.

1.3.1.2 Conception et réorganisation des données Firebase avec permissions

Lors de l'implémentation du prototype, Firebase a été retenu comme gestionnaire de données Backend. Cet outil permet d'implémenter facilement et rapidement la lecture et l'écriture de données dans la mesure où toutes les méthodes permettant la communication entre l'application et la base de données ont déjà été implémentées par les équipes de Google. Des méthodes permettant l'écoute et la mise à jour en temps réel d'information sur la base de données Firebase par l'application sont également disponibles. De plus, Firebase et l'outil de développement Frontend choisi, Flutter, sont tous deux compatibles avec l'implémentation de notifications sur smartphones, fonctionnalité presque indispensable pour beaucoup d'applications, ici aussi par exemple lors de la réception d'un nouveau rendez-vous.

En contrepartie, il y a deux inconvénients notables à l'utilisation de Firebase : il s'agit d'un outil payant et l'outil n'offre pas une gestion relationnelle des données car il s'agit d'un noSQL. Pour le premier point, l'utilisation de Firebase reste gratuite en dessous de certains quotas d'utilisation. Dans le cadre du développement de l'application, cela ne devrait donc pas poser de problèmes.

Pour la partie noSQL, en revanche, ce problème inclut également la gestion des permissions sur les données. Actuellement, la structure et l'organisation des données mis en place ne permet pas d'attribuer correctement les droits sur les différents éléments présents dans Firebase.

Pour garantir une sécurité des données des utilisateurs, les permissions actuelles limitent certaines fonctionnalités du prototype, par exemple il n'est pas possible pour une nounou de consulter le profil du parent ayant effectué la demande d'un rendez-vous.

De ce fait, il est nécessaire de réfléchir et de concevoir une nouvelle structure des données afin que cette dernière réponde aux différentes requêtes de l'application tout en assurant une sécurité au niveau des données. Bien sûr, des changements devront également être appliqués au niveau Frontend pour correspondre avec la nouvelle architecture Firebase.

1.3.1.3 Amélioration du profil utilisateur pour les nounous

Le troisième objectif concerne une fonctionnalité en partie implémentée durant le projet de semestre précédent. Il est en effet déjà possible de gérer la photo de profil, les enfants ou encore certaines informations basiques concernant l'utilisateur. En revanche, pour les nounous, des informations complémentaires pourraient être apportées mais cela n'a pas été intégré au prototype. L'objectif est donc de permettre aux nounous de gérer les informations spécifiques les concernant, comme le type de nounou (maman de jour, baby-sitter du soir, garde partagée, etc.) ou encore indiquer si cette dernière dispose d'un diplôme professionnel.

Bien évidemment, l'implémentation de cette fonctionnalité inclut également le fait pour un parent de pouvoir consulter toutes ces informations lorsque ce dernier consulte le profil d'une nounou.

1.3.1.4 Amélioration de la gestion des rendez-vous

Actuellement, le prototype propose une gestion simplifiée des rendez-vous : une date, une heure de début et de fin, sans commentaires additionnels possible. Lorsqu'une nounou reçoit un rendez-vous, celle-ci n'est pas informée par l'application et ne peut que choisir d'accepter ou de refuser le rendez-vous. Du fait des règles actuelles de Firebase, la nounou ne peut même pas voir le profil du parent et des enfants qu'elle va garder !

Il est donc indispensable d'améliorer la prise et la gestion des rendez-vous. Pour cela, plus d'informations doivent être communiquées lors de la prise de rendez-vous : tarif, enfants concernés ou encore le lieu de garde. Une fois le rendez-vous envoyé, la nounou doit être informée qu'une nouvelle demande lui a été assignée, que ce soit par le biais de notifications et/ou de badges dans l'application. La nounou doit pouvoir consulter le profil du parent et des enfants concernés avant d'accepter ou de refuser l'évènement.

1.3.1.5 Réflexion sur le modèle économique de l'application

Tous les objectifs cités jusqu'à présent concernent le prototype implémenté lors du projet de semestre précédent. En effet, les réflexions sur l'architecture Flutter et la structure des données Firebase sont apparues suite aux problèmes rencontrés pendant l'implémentation du prototype. Le profil utilisateur et les rendez-vous sont également des fonctionnalités déjà en partie réalisées dans le prototype.

Or, ce projet de Bachelor doit également apporter des objectifs qui ne sont pas liés au prototype actuellement implémenté. Pour cela, une analyse des frais d'utilisation de Firebase et des outils de paiements intégrés sera premièrement effectuée.

Ensuite, une conception d'un modèle économique compatible avec notre application sera réalisée. Le modèle économique doit pouvoir à minima couvrir les frais d'utilisation de Firebase. Initialement, plusieurs approches devaient être étudiées, comme la prise d'une commission sur le paiement des nounous (ce qui inclut l'intégration de paiement au sein de l'application) ou encore l'affichage de publicités. Néanmoins, par manque de temps, seule la conception d'un modèle économique se basant sur l'intégration des paiements intégrés et la prise d'une commission sur les paiements sera étudiée.

Cet objectif ne concerne que l'analyse et la conception du modèle économique, dans le but d'informer sur le potentiel économique dont l'application HappyBaby pourrait bénéficier. L'implémentation concrète du modèle économique dans l'application est un objectif secondaire de ce projet de Bachelor et ne sera réalisé que si le temps le permet.

1.3.1.6 Tests utilisateurs

Un deuxième objectif, qui ne découle pas directement du projet de semestre précédent, concerne la réalisation de tests utilisateurs. Pour identifier la satisfaction et l'utilisabilité de notre application, il est nécessaire de concevoir et de réaliser différents scénarios de test avec de vrais personnes. Ces derniers devront ensuite compléter différents formulaires de satisfaction.

Durant cette phase, le choix de Firebase pourrait s'avérer problématique : pour réaliser des tests convaincants, il serait nécessaire qu'un certain nombre de parents et de nounous utilisent l'application dans un cadre pratique. Or, si les quotas d'utilisation de Firebase sont dépassés, il est possible que des frais viennent s'ajouter au projet. Il est donc nécessaire de bien étudier et anticiper le volume des requêtes lors de tests utilisateurs pour ne pas dépasser les quotas tout en proposant une phase de tests convaincante. Il est possible de reprendre les frais estimés dans l'analyse du modèle économique afin d'étudier la question.

1.3.2 Objectifs secondaires

Cette section liste les objectifs secondaires qui sont facultatifs pour ce projet de Bachelor. Ces objectifs seront réalisés uniquement si le temps le permet et si les objectifs principaux ont déjà été entièrement réalisés.

1.3.2.1 Implémentation du modèle économique

Durant ce projet de Bachelor, l'analyse et la conception du modèle économique sont deux éléments très importants pour apporter une réflexion sur des objectifs n'ayant pas ou peu de rapport avec le projet de semestre précédent. Néanmoins, l'implémentation et l'intégration du modèle économique dans l'application n'apporte que peu d'avantages dans le cadre de ce projet, du moins en comparaison avec les phases d'analyse et de conception. De plus, en fonction du modèle choisi, l'implémentation peut s'avérer plus ou moins complexe mais peut également engendrer des frais supplémentaires.

Pour ces raisons, l'implémentation du modèle économique ne sera réalisée que si le temps le permet et que la complexité n'est pas trop élevée.

1.3.2.2 Implémentation d'un outil de messagerie

L'implémentation d'un outil de messagerie permettant aux parents et aux nounous de communiquer directement par le biais de l'application est très importante pour garantir la satisfaction des utilisateurs. En effet, aucun parent n'a envie de confier son enfant à quelqu'un qu'il ne connaît pas ou très peu. Naturellement, les parents vont donc essayer de prendre contact avec la nounou avant la date du rendez-vous, voir même avant d'en proposer un.

Si le parent doit utiliser un outil externe pour communiquer avec la nounou, ce dernier aura de grandes chances d'être insatisfait car il doit effectuer plus d'actions et éventuellement créer un compte sur l'outil en question. De plus, il sera nécessaire que notre application fournisse les coordonnées nécessaires pour contacter la nounou, comme son numéro de téléphone, ce qui n'est pas non plus souhaitable pour toutes les nounous !

Pour ces raisons, il est donc recommandé de fournir un outil de communication directement intégrés à l'application. Initialement, l'ajout d'un outil de messagerie figurait dans la liste des objectifs principaux. Cependant, lors de la phase de planification, il a été décidé de ne pas inclure cet objectif afin d'éviter une surcharge et une mauvaise planification, de la même manière que durant le projet de semestre précédent.

Pour implémenter un outil de messagerie, il est dans un premier temps nécessaire de concevoir les cas d'utilisation possibles ainsi que l'intégration dans l'application. Ensuite, l'implémentation de la fonctionnalité au sein de l'application pourra être réalisée.

1.3.2.3 Amélioration de la gestion des rendez-vous (suite)

Si le temps le permet, il serait intéressant de donner la possibilité aux parents et aux nounous de modifier et/ou de proposer des modifications concernant un rendez-vous déjà établi. Par exemple, si l'heure de fin du rendez-vous ne convient pas à la nounou à 30 minutes près mais que toutes les autres indications lui conviennent, il serait intéressant que la nounou puisse proposer de modifier le rendez-vous afin que ce dernier se termine 30 minutes plus tôt.

De même, si un parent souhaite modifier un rendez-vous car il y a eu du changement dans son agenda, il devrait être possible de proposer une modification du rendez-vous à la nounou. Actuellement, le parent est obligé d'annuler le rendez-vous et de demander un nouveau rendez-vous avec les informations désirées.

1.3.2.4 Implémentation du feedback utilisateur

Une fonctionnalité très intéressante qui pourrait être implémentée, déjà présente sur les sites web « concurrents », est la possibilité de donner aux parents et nounous un retour sur l'expérience vécue. Pour cela, il est nécessaire de concevoir et d'implémenter un système de feedback et de notation pour les utilisateurs. Il ne serait possible de noter l'autre personne qu'une fois la garde de l'enfant terminée, afin d'éviter les abus.

Dans l'idéal, les enfants pourront eux aussi apporter des commentaires sur l'expérience vécue avec la nounou. Cela peut se faire par le biais d'un enregistrement audio de l'enfant lorsque celui-ci raconte son ressenti, à condition bien sûr que l'enfant sache déjà parler. Pour anonymiser l'enfant, il est possible de légèrement modifier l'enregistrement.

1.3.2.5 *Refonte graphique de l'application*

Enfin, un dernier objectif secondaire serait de concevoir une identité graphique propre à l'application. Jusqu'à présent, l'aspect visuel de l'application a été traité de sorte que l'application soit fonctionnelle et à peu près cohérente. Il n'y a donc pas vraiment d'efforts et de temps qui ont été accordés au design de l'application. Pour réaliser cet objectif, il est dans un premier temps nécessaire de concevoir un design pour les différents éléments que l'on retrouve dans notre application. Il est également nécessaire de concevoir une identité visuelle cohérente avec des couleurs unies.

Une fois la conception les différents *widgets* réalisés, l'application pourra être modifiée pour intégrer la nouvelle architecture graphique. De plus, l'implémentation actuelle se base uniquement sur des *widgets* de type « Material ». Or, bien que ce type de *widget* soit également disponible sur iOS, ce n'est pas le design par défaut présent sur ce système d'exploitation.

En effet, des *widgets* de type « Cupertino » sont disponibles sur Flutter pour reprendre les éléments de design propre à iOS. Dans l'idéal, l'application pourrait donc utiliser les vues *Cupertino* lorsque cette dernière est exécutée sur iOS et *Material* avec Android.

1.4 Organisation du document

Ce rapport suit les principes de rédaction d'un rapport scientifique. Il est divisé en 5 sections principales, chacune visant une thématique particulière du problème :

L'introduction vise à informer le lecteur sur la nature du problème en fournissant le contexte actuel ainsi que les objectifs souhaités visant à « résoudre » le problème.

Les méthodes se focalisent sur les moyens possibles pour résoudre le problème. Cela se fait tout d'abord part une analyse du contexte actuel et des outils à disposition, suivi d'une conception concluant l'analyse.

Le résultat explique l'implémentation réalisée pour résoudre le problème ainsi que les tests visant à valider que le problème est bien résolu.

La discussion est une section ouverte qui a pour but de discuter des méthodes et de l'implémentation choisie afin de stimuler la réflexion pour d'éventuels améliorations ou changements dans la résolution des problèmes.

La conclusion apporte une touche personnelle à la réalisation du projet et vise à conclure la rédaction sur le projet.

2 Méthodes

Cette section a pour but d'expliquer les méthodes disponibles et celles choisies pour la réalisation du projet. Une première sous-section porte sur l'analyse du prototype réalisé. Une deuxième sous-section vise à concevoir les différents environnements de chaque côté de l'application, par le biais de diagrammes et de maquettes.

2.1 Analyse

Cette sous-section regroupe les différentes analyses effectuées pour la réalisation de ce projet de Bachelor. En début de projet, une analyse du prototype déjà implémenté a été réalisée. Une fois l'application fonctionnelle et répondant aux objectifs liés à l'implémentation des fonctionnalités, une analyse des outils d'intégration de moyens de paiement au sein d'une application a été réalisée, dans le but de concevoir le modèle économique de l'application. Enfin, une analyse des outils de mesure de l'expérience utilisateur a été réalisée dans le but de concevoir des tests utilisateurs convaincants.

2.1.1 Prototype réalisé durant le projet de semestre 6

Cette section a pour objectif d'analyser et d'expliquer le prototype implémenté durant le projet de semestre précédent, afin de bien comprendre l'état actuel de l'application et des fonctionnalités disponibles.

2.1.1.1 Recherche d'utilisateurs

Le prototype intègre déjà le stockage de la position et la recherche d'utilisateur dans un rayon proche. Pour éviter de devoir traiter toutes les entrées présentes dans la base de données, un index doit être placé de sorte que seules les positions proches de la localisation donnée soient analysées. Pour cela, le prototype intègre le concept de *Geoqueries* Firebase [10] par le biais de l'utilisation du package Flutter *GeoFlutterFire* [11].

Lorsque l'utilisateur effectue une recherche, une requête est envoyée au serveur Firebase par le biais du plugin GeoFlutterFire. Ensuite, pour chaque résultat, une autre requête est envoyée afin de récupérer le profil de l'utilisateur. Une fois que tous les profils des utilisateurs recherchés ont été récupérés, ces derniers sont affichés sur l'application et il est possible de consulter les détails du profil en cliquant sur l'utilisateur.

Actuellement, il est possible d'utiliser uniquement la position actuelle du smartphone comme point central pour la recherche. Le rayon peut varier selon les besoins de l'utilisateur entre 1 et 50km. L'utilisation d'un lieu ou d'une adresse n'est pour l'instant pas possible car le prototype n'inclut pas l'utilisation de l'API Google Maps.

Les captures d'écran du prototype ci-dessous montrent les écrans implémentés pour la recherche d'utilisateurs :

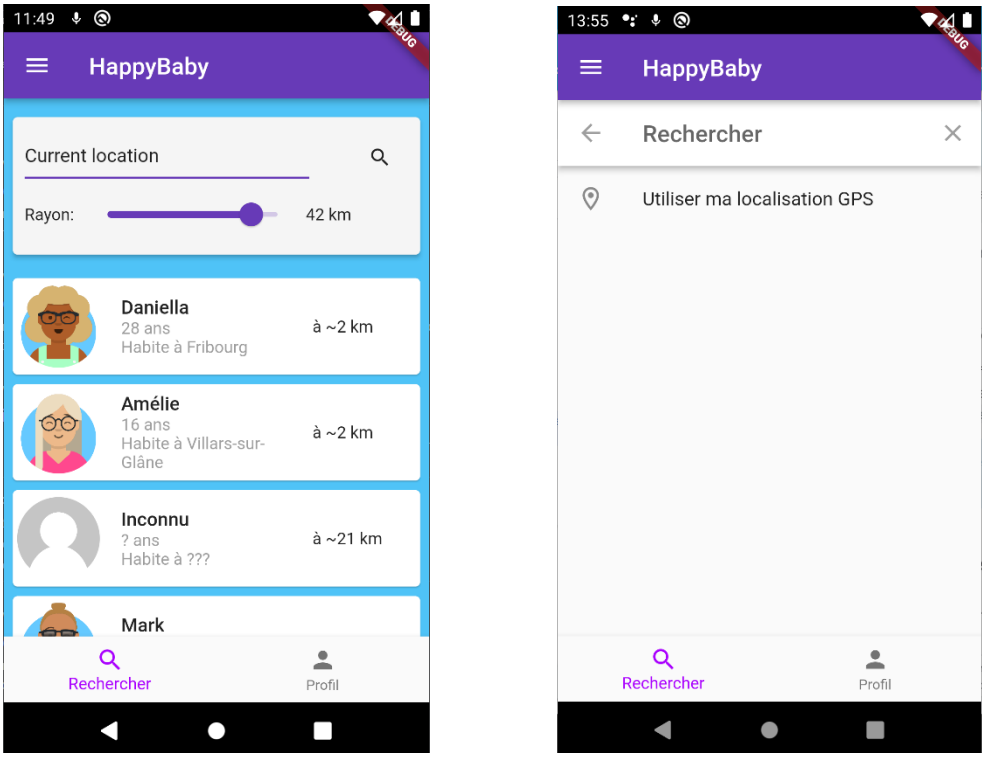

Figure 2 - Captures d'écran de la recherche d'utilisateurs implémenté dans le prototype du projet de semestre précédent

Cette fonctionnalité essentielle au fonctionnement de l'application est donc déjà presque entièrement implémentée. Lors de la phase de refactoring, seule l'intégration de l'API Google Maps devra être implémentée pour cette fonctionnalité. Néanmoins, lors de la phase d'amélioration du profil utilisateur, la recherche devra être modifiée afin d'inclure les nouveaux éléments indiqués par les nounous, sous forme de recherche avancée.

2.1.1.2 Profil utilisateur

Pour pouvoir être en mesure de rechercher des utilisateurs, un parent ou une nounou doit avoir la possibilité de créer, de consulter et de modifier son profil. Cette fonctionnalité est également déjà présente dans le prototype, mais de nombreuses améliorations peuvent être apportées.

Actuellement, un utilisateur authentifié peut consulter et modifier son profil en naviguant sur l'onglet du même nom. Sur cet écran, l'utilisateur peut voir et/ou changer ses informations de base, ses enfants ainsi que son statut de nounou.

Une vue est implémentée pour chaque section, comme le montre les captures d'écran ci-dessous :

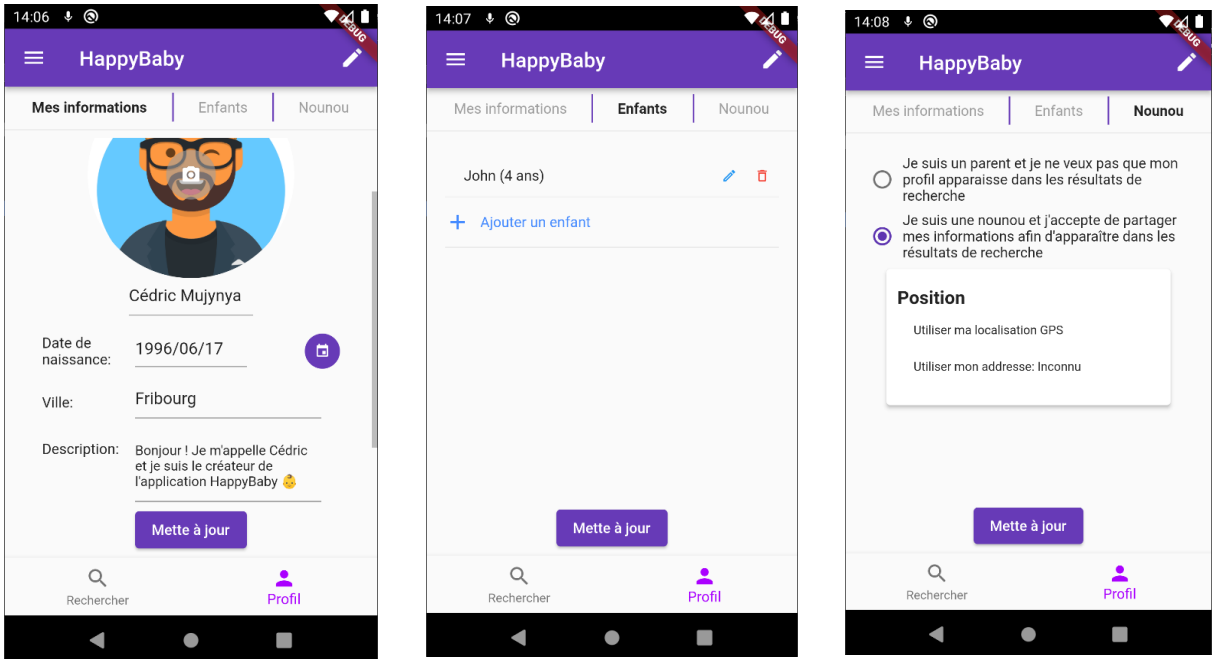

Figure 3 - Captures d'écran du profil utilisateur implémenté dans le prototype du projet de semestre précédent

Le prototype intègre déjà la gestion d'une bonne partie des informations de l'utilisateur, en particulier si ce dernier est un parent. La modification de l'adresse e-mail et du numéro de téléphone n'est cependant pas présente, de même qu'il n'est pas possible de supprimer son compte. L'utilisation de l'API Google Maps pour spécifier son emplacement n'est également pas disponible. Durant la phase de refactoring, ces fonctionnalités devront être implémentées.

Actuellement, les méthodes implémentées pour récupérer les informations de l'utilisateur ne profitent pas de la mise à jour en temps réel de Firebase, et le cheminement utilisé pour mettre à jour les vues ne suit pas non plus une logique particulière. Durant la phase de refactoring, une fois la conception de la nouvelle architecture Flutter réalisée, il sera donc également nécessaire de modifier les méthodes d'acquisition du profil de l'utilisateur.

De plus, presque toutes les informations concernant les nounous ne peuvent pas être indiquées dans l'implémentation actuelle du prototype. Le type de service proposé, le tarif ou encore l'obtention d'un diplôme professionnel dans le domaine ne peuvent pas être précisés par la nounou. Ces informations peuvent cependant être cruciales pour le parent lors du choix et de la recherche d'une nounou.

Dans la phase d'amélioration du profil utilisateur, toutes les informations citées devront pouvoir être entrées par l'utilisateur. Sur l'écran de recherche, il doit être possible filtrer les résultats de recherche en fonction des informations.

2.1.1.3 Prise de rendez-vous

La dernière fonctionnalité implémentée dans le prototype concerne la prise et la gestion des rendez-vous. Actuellement, quand un parent consulte le profil d'une nounou à la suite d'une recherche sur l'application, ce dernier peut effectuer une demande de rendez-vous. Lors de sa requête, le parent peut spécifier la date ainsi que l'heure de début et de fin du rendez-vous.

De son côté, la nounou peut consulter les horaires du rendez-vous proposé et choisir d'accepter ou non ce dernier. Actuellement, aucune notification n'est envoyée que ce soit lors de la demande d'un nouveau rendez-vous ou lors de la validation ou du refus du rendez-vous. Avec les règles de sécurité actuellement mises en place, la nounou ne peut pas consulter le profil du parent et de ses enfants. Les captures d'écran ci-dessous montrent l'implémentation actuelle des rendez-vous :

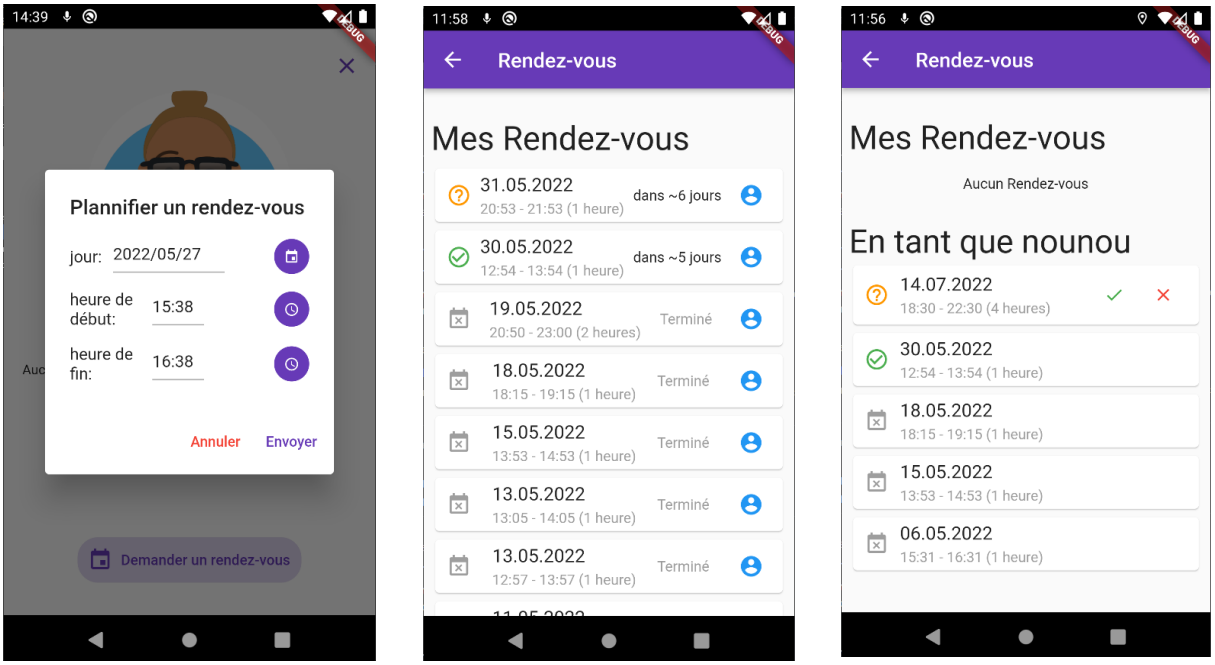

Figure 4 - Captures d'écran de la gestion des rendez-vous implémenté dans le prototype du projet de semestre précédent

De nombreuses améliorations peuvent être apportées au niveau de la gestion des rendez-vous. Premièrement, l'obtention des rendez-vous ne se fait que lors de l'affichage de la vue sur l'application. Lors de la phase de refactoring, cette partie pourra être améliorée afin que la vue s'actualise en temps réel avec l'état des données présentes sur le serveur Firebase. En effet, la mise à jour en temps réel des données et l'un des avantages à l'utilisation de Firebase.

Ensuite, le fait pour une nounou de ne pas pouvoir consulter le profil du parent et des enfants concernés par le rendez-vous n'est tout simplement pas acceptable. Dans la phase de réorganisation des données et des permissions dans Firebase, après avoir modifié les droits sur les profils utilisateur, cette fonctionnalité devra être implémentée.

De plus, les informations communiquées par le parent lors de la prise de rendez-vous ne sont pas suffisantes et d'autres détails pourraient être apportés. En particulier le lieu du rendez-vous (chez la nounou ou chez le parent), le tarif ou encore le moyen de paiement devraient pouvoir être indiqués.

Enfin, aucune notification n'est envoyée à l'autre utilisateur concerné lors de la prise ou de la modification du statut d'un rendez-vous. Pour autant, cette fonctionnalité serait très intéressante pour garantir une réponse plus rapide des nounous et assurer une meilleure satisfaction des parents mais aussi des nounous, qui ne seraient pas obligées de consulter régulièrement l'application pour savoir si de nouveau rendez-vous sont disponibles. Lors de la phase d'amélioration des rendez-vous, la prise en charge des notifications devra être implémentée.

2.1.2 Modèle économique

Dans un contexte de déploiement de l'application sur le marché des applications mobiles, il est indispensable d'évaluer les coûts engendrés lors de son utilisation, dans le but d'établir un modèle économique. Ce modèle doit permettre de générer des bénéfices à partir de notre application, à minima pour couvrir les frais d'utilisation. Bien évidemment, le mieux serait même d'engendrer suffisamment de revenus permettant à l'application de générer des bénéfices.

Pour cela, il est dans un premier temps nécessaire d'évaluer les coûts d'utilisation des services externes utilisés. Ces services sont tous proposés par la plateforme Google Developers, qui définit clairement les frais d'utilisation en fonction de l'usage des services, tels que les fonctionnalités de Firebase ou encore l'API de Google Maps. Concernant Firebase, il existe deux plans tarifaires différents:

- **Spark plan** : Utilisation gratuite de Firebase, mais avec des limitations sur l'usage
- **Blaze plan** : Utilisation complète de Firebase, mais avec une facturation à l'usage

Malheureusement, le choix de la formule Spark n'est pas envisageable car une fois le quota dépassé, il n'est plus possible d'effectuer des requêtes, ce qui n'est évidemment pas le comportement souhaité. Sur la page de facturation de Firebase [12], l'on retrouve la facturation à l'usage suivante (seules les fonctionnalités utilisées par l'application HapyBaby sont présentées) :

Service	Utilisation gratuite	Facturation
Authentification (email)	Illimitée	-
Stockage Firestore	Jusqu'à 1GiB	$0.117/GiB/mois
Trafic réseau Firestore (entrée)	Illimitée	-
Trafic réseau Firestore (sortie)	Jusqu'à 10GiB/mois	En moyenne $0.12/GiB *
Ecriture de documents Firestore	Jusqu'à 20'000/jour	$0.117 pour 100'000 écritures
Lecture de documents Firestore	Jusqu'à 50'000/jour	$0.039 pour 100'000 lectures
Suppression de documents Firestore	Jusqu'à 20'000/jour	$0.013 pour 100'000 suppressions
Appel d'une Cloud Functions	Jusqu'à 2M d'appels/mois	$0.40 pour 1M d'appels
Utilisation RAM en GB-secondes	Jusqu'à 400'000 GB-secondes	$0.0000035/GB-secondes
Utilisation CPU en GHz-secondes	Jusqu'à 200'000 GHz-secondes	$0.0000140/GHz-secondes
Trafic réseau Cloud Functions (entrée)	Illimitée	-
Trafic réseau Cloud Functions (sortie)	Jusqu'à 5GiB/mois	$0.12/GiB

Service	Utilisation gratuite	Facturation
Utilisation de FCM	Illimitée	-
Stockage Cloud Storage (photos)	Jusqu'à 5GB	$0.026/GB
Trafic réseau Cloud Storage (entrée)	Illimitée	-
Trafic réseau Cloud Storage (sortie)	Jusqu'à 1GB/jour	$0.12/GB
Ecriture de fichiers	Jusqu'à 20'000/jour	$0.05 pour 10'000 écritues
Lecture de fichiers	Jusqu'à 50'000/jour	$0.004 pour 10'000 lectures

** les plans tarifaires du réseau dépendent de la destination et de l'utilisation mensuelle*

En complément des fonctionnalités de Firestore, notre application utilise également l'API de Google Maps afin de récupérer une adresse à partir des coordonnées géographiques, appelé Geocoding. L'API est également utilisée afin de fournir une autocomplétion de l'adresse à partir du texte entré par l'utilisateur. Bien évidemment, ces deux requêtes ne sont pas gratuites et suivent une facturation à l'usage, de la même manière que la formule Blaze pour Firebase. Sur la page de facturation de l'API Google Maps [13], l'on retrouve les tarifications suivantes (seules les fonctionnalités utilisées par l'application sont présentées) :

Service	Utilisation gratuite	Facturation
Autocomplétion	Jusqu'à $200/mois (cumulé)	$2.83/1000 requêtes (session)*
Informations détaillées sur le lieu		$17.00/1000 requêtes (session)*
Geocoding		$5.00/1000 requêtes*

** Les frais de facturation décrit concernent une utilisation jusqu'à 100'000 requêtes par mois. Au-delà de cette limite, les frais d'utilisation changent.*

Pour estimer correctement les coûts d'utilisation en fonction du nombre d'utilisateurs présents, il est nécessaire de prévoir et/ou d'analyser l'utilisation moyenne d'un utilisateur. Afin d'anticiper au mieux les coûts d'utilisation, il serait préférable d'identifier au préalable les différents profils d'utilisateurs se servant de l'application et d'estimer une utilisation moyenne pour chaque profil différent. En effet, une nounou n'utilise pas forcément la même « quantité de services facturables » qu'un parent. Mais cela va plus loin encore : par exemple, parmi les parents utilisant l'application, l'on peut imaginer que certains parents utilisent de manière occasionnelle et non répétée l'application, tandis que d'autres parents sont bien plus actifs.

Néanmoins, pour identifier correctement les différents profils utilisateurs que compose notre application, il est nécessaire d'analyser dans le détail le public cible visé par l'application et/ou de réaliser des tests utilisateurs à grande échelle afin d'identifier les différents profils. Or, par manque de temps et de moyens, il a été décidé de ne pas séparer les différents profils d'utilisateurs existants pour estimer les coûts d'utilisation d'un utilisateur. Bien que cette méthode introduise un éventuel plus grand écart entre les coûts estimés et les coûts réels, cela permet tout de même d'avoir une estimation qui reflète une utilisation réelle.

Pour estimer les coûts moyens d'un utilisateur, il a été décidé d'analyser la « quantité de services » utilisé par un utilisateur à la fois parent et nounou au sein de l'application. Ainsi, si un écart entre les coûts théoriques et les coûts réels devait être détecté, il y aurait alors de grandes chances qu'il s'agisse d'une surestimation des coûts. Concrètement, cela revient à

dire que, dans un contexte de déploiement de l'application, cette dernière serait moins chère à entretenir qu'estimer, et il serait alors possible de générer un plus gros bénéfice ou de baisser les revenus générés tout en gardant un équilibre économique. Il s'agit donc d'un scénario bien plus favorable qu'une sous-estimation des coûts, ce qui risquerait d'engendrer des pertes. Pour chaque service payant utilisé, une estimation moyenne de l'utilisation de ce service par l'utilisateur a été réalisée.

Stockage Firestore :

Le stockage Firestore concerne les données propres à l'application de l'utilisateur (à l'exception de la photo de profil) ainsi que les données des rendez-vous. Pour estimer correctement le stockage moyen d'un utilisateur, il est nécessaire de connaître la taille des différents éléments présents dans un document ou une collection Firebase, ainsi que l'intégralité des champs présents dans notre application.

Pour le calcul de la taille des éléments, la documentation de Firebase [14] fournit des informations concernant le stockage des éléments en fonction de leur nature. D'un autre côté, l'objectif de ce projet de Bachelor *« Conception et réorganisation des données Firebase avec permissions »* a pour but d'identifier tous les champs présents dans les différents documents. Vous trouverez un schéma décrivant la structure des données dans Firebase et les différents champs dans la section « 2.2.2 – Structure Firebase », Figure 19 - Structure des données dans Firebase. Avec ces informations, il est alors possible d'estimer la taille des données pour un document d'une collection. Vous trouverez ci-dessous une estimation de la taille d'un document pour chaque collection, selon l'organisation Firebase en tant que parent + nounou.

Collection « profile-authorized » :

Taille du nom du document : 19 (nom collection) + 29 (nom document) + 16 = 64 octets

Taille des champs du document (en octets) :

Nom	Taille du nom	Taille de la valeur	Taille des indexes	Taille totale
Birthdate	10	24	254	288
Description	12	100	410	522
Languages	10	16	119	145
phoneNumber	12	14	238	264
Location	9	32	0	41
City (location)	5	11	218	234
Description (location)	12	40	290	342
Position (location)	9	32	0	41
Geohash (location)	8	10	222	240
Geopoint (location)	9	16	236	261
Total	96	295	1'987	2'378 octets

Taille des index composites : aucun index composite pour cette collection

Taille totale moyenne d'un document dans la collection *profile-authorized* : **2'474 octets**

Collection « profile-authorized/{uid}/kids »:

Taille du nom du document : 19 (nom sous-collection) + 29 (nom sous-document) + 5 (nom collection) + 21 (nom document) + 16 = 90 octets

Taille des champs du document (en octets) :

Nom	Taille du nom	Taille de la valeur	Taille des indexes	Taille totale
Birthdate	10	24	254	288
Description	12	100	410	522
DisplayName	12	10	228	250
Total	34	134	892	1'060 octets

Taille des index composites : aucun index composite pour cette collection

Taille totale moyenne d'un document dans la collection *kids* : **1'182 octets**

Collection « profile-private » :

Taille du nom du document : 16 (nom collection) + 29 (nom document) + 16 = 61 octets

Taille des champs du document :

Nom	Taille du nom	Taille de la valeur	Taille des indexes	Taille totale
AuthorizedUid	15	145	237	397
Email	5	23	242	270
FcmToken	10	164	534	708
Total	30	332	1013	1'375 octets

Taille des index composites : aucun index composite pour cette collection

Taille totale moyenne d'un document dans la collection *profile-private* : **1'468 octets**

Collection « profile-public » :

Taille du nom du document : 15 (nom collection) + 29 (nom document) + 16 = 60 octets

Taille des champs du document :

Nom	Taille du nom	Taille de la valeur	Taille des indexes	Taille totale
DisplayName	12	10	228	250
IsBabysit	11	1	208	220
Uid	4	29	250	283
BabysitRate	13	8	226	247
BabysitType	13	32	0	45
Babyitter (type)	10	1	206	217
DaytimeMother (type)	14	1	214	229
Hosting (type)	8	1	202	211
Nursery (type)	8	1	202	211
ShareChildcare (type)	15	1	208	224
Unique (type)	7	1	200	208
canClean	9	1	204	214
canCook	8	1	202	211
canHelpHomework	16	1	210	227
haveAidCourse	14	1	214	229
haveProfessionalLicence	24	1	234	259
havePersonalVehicle	20	1	226	247
profilePictureUrl	18	164	548	730
Total	224	256	3'982	4'462 octets

Taille des index composites : *IsBabysit* et *GeoHash*: 60 (nom) + 1 + 10 + 32 = 103 octets

Taille totale moyenne d'un document dans la collection *profile-private* : **4'657 octets**

Collection « meeting » :

Taille du nom du document : 8 (nom collection) + 21 (nom document) + 16 = 45 octets

Taille des champs du document :

Nom	Taille du nom	Taille de la valeur	Taille des indexes	Taille totale
AtBabysitterHouse	18	1	192	211
AtParentHouse	14	23	242	270
Status	7	9	186	202
BabysitterUid	14	29	240	283
ParentUid	10	29	232	271
EndDate	8	8	186	202
StartDate	10	8	190	208
KidsId	7	32	116	155
Description	12	100	378	490
Rate	5	8	180	193
PaymentMethod	14	10	202	226
Type	5	10	184	199
Scheduled	10	1	176	187
ScheduledDays	14	32	0	46
Monday (schedule)	7	1	170	178
Tuesday (schedule)	8	1	172	181
Wednesday (schedule)	10	1	176	187
Thursday (schedule)	9	1	174	184
Friday (schedule)	7	1	170	178
Saturday (schedule)	9	1	174	184
Sunday(schedule)	7	1	170	178
Total	205	307	3'910	4'413 octets

Taille des index composites :

- *BabysitterUid, parentUid* et *endDate* : 45 (nom) + 29 + 29 + 8 = 111 octets
- *BabysitterUid* et *startDate* : 45 (nom) + 29 + 8 = 82 octets
- *ParentUid* et *startDate* : 45 (nom) + 29 + 8 = 82 octets

Taille totale moyenne d'un document dans la collection *meeting* : **4'765 octets**

Si l'on considère qu'un utilisateur moyen possède deux enfants, cela nous donne une taille totale de : 2'474 (authorized) + 2 * 1'182 (kids) + 1'468 (private) + 4'657 (public) = 10'963 octets, soit environ **11 KB/utilisateur**.

En plus du stockage fixe concernant l'utilisateur, il y a également le stockage des rendez-vous à prendre en compte. Si l'on émet l'hypothèse qu'un utilisateur moyen effectue 3 propositions de rendez-vous par mois, cela nous donne 3 * 4'765 = 14'295 octets, soit environ **14.3 KB/utilisateur/mois**.

Il est néanmoins possible de drastiquement baisser le stockage de Firestore : nous observons sur les tableaux des pages précédentes que le stockage le plus conséquent concerne les indexes sur un champ unique. En effet, Firebase créé automatiquement des index pour chaque champ qui compose n'importe quel document. Ces index automatiques sont créés à la fois de manière croissante et de manière décroissante et il n'est pas possible de désactiver la création automatique d'index, comme le montre la capture d'écran ci-dessous :

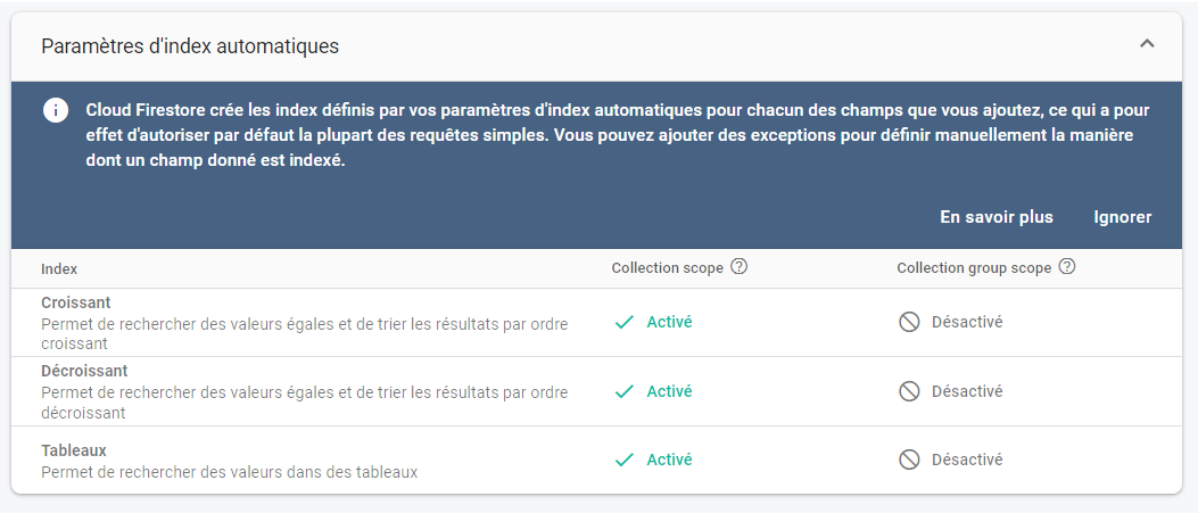

Figure 5 - Capture d'écran de la console Firebase concernant les index sur un champ individuel

En revanche, il est possible d'ajouter des exceptions afin de ne pas créer d'index automatique sur chaque champ qui compose les documents. Il faut néanmoins ajouter une exception pour chaque champ. Malheureusement, le temps restant avant l'échéance du projet de permet pas de créer des règles afin de supprimer les index automatiques et de vérifier si le fonctionnement de l'application n'a pas été perturbé.

Il est en revanche possible d'estimer le volume de stockage par utilisateur sans ces index :

Profile-authorized : 487 octets
Kids : 290 octets par enfant
Profile-private : 455 octets
Profile-public : 675 octets
Meeting : 864 octets

Si l'on reprend la même hypothèse d'un parent ayant deux enfants et effectuant 3 rendez-vous par mois, le stockage n'est plus que d'environ **2 KB/utilisateur** et de **2.5 KB/utilisateur/mois**.

Trafic réseau Firestore :

Maintenant que la taille des documents utilisés par l'application a pu être estimée, il est à présent possible d'évaluer le trafic engendré par un utilisateur. Au niveau du trafic entrant (donc des données envoyées à destination de Firebase), aucun frais n'est généré. Il n'y a donc pas de calculs à faire lors de la création ou de la mise à jour des données de l'utilisateur ou des rendez-vous.

En revanche, le trafic sortant, lui, est facturé. Il faut donc évaluer la charge réseau moyenne d'un utilisateur lorsque ce dernier réceptionne des informations à propos du profil d'un utilisateur ou d'un rendez-vous.

Pour estimer correctement le trafic sortant de Firestore à destination des applications, il est nécessaire de bien comprendre le fonctionnement interne de l'application :

- Automatiquement, dès que l'utilisateur est authentifié, son profil est téléchargé
- De même, tous les rendez-vous concernant l'utilisateur sont téléchargés
- Pour chaque résultat de recherche, le profil public de la nounou est téléchargé. Le profil est ensuite « transmis » à la vue s'occupant d'afficher le profil détaillé d'une nounou, il n'y a donc pas de deuxième téléchargement
- Quand un utilisateur affiche le détail d'un rendez-vous, les profils des enfants concernés par le rendez-vous sont téléchargés (le rendez-vous en lui-même, lui, n'est pas retéléchargé)
- Quand l'utilisateur souhaite afficher l'autre profil concerné par le rendez-vous, le profil public + autorisé de la personne est téléchargé

Afin d'estimer le trafic sortant de Firestore, les hypothèses suivantes ont été utilisées : Un parent ouvre en moyenne 5 fois l'application et effectue 3 recherches par mois, avec en moyenne 10 utilisateurs trouvés. Il possède 10 rendez-vous en moyenne et affiche les détails de 3 rendez-vous par mois et consulte le profil de la personne une fois sur deux, soit 1.5x/mois. De plus, il modifie en moyenne 1 fois par mois son profil, impliquant de retélécharger le profil. Avec ces estimations, il est possible de calculer le trafic :

Profile-public

Au total, le profil public est téléchargé 5 (authentification) + 3*10 (recherche) + 1.5 (rendez-vous) + 1 (profil) = 37.5 fois par mois. Avec l'estimation de 224 + 256 = 480 octets de données pour ce document (les colonnes « Taille du nom » et « Taille de la valeur » dans le tableau ci-dessus), nous avons une estimation d'environ **18 KB**/utilisateur/mois pour le profil public.

Profile-authorized

Au total, le profil autorisé est téléchargé 5 (authentification) + 1.5 (rendez-vous) + 1 (profil) = 7.5 fois par mois. Avec l'estimation de 96 + 295 = 391 octets de données pour ce document, nous avons une estimation d'environ **2.9 KB**/utilisateur/mois pour le profil autorisé.

Profile-authorized/{uid}/kids

Au total, le profil des enfants est téléchargé [5 (authentification) + 3 (détails)] * 2 (nombre d'enfants) = 16 fois par mois. Avec l'estimation de 34 + 134 = 168 octets de données pour ce document, nous avons une estimation d'environ **2.7 KB**/utilisateur/mois pour le profil des enfants.

Profile-private

Au total, le profil privé est téléchargé 5 (authentification) + 1 (profil) = 6 fois par mois. Avec l'estimation de 30 + 332 = 362 octets de données pour ce document, nous avons une estimation d'environ **2.2 KB**/utilisateur/mois pour le profil privé.

Meeting

Au total, les rendez-vous sont téléchargés 5 (authentification) * 10 (nb de rendez-vous) = 50 fois par mois. Avec l'estimation de 205+307 = 512 octets de données pour ce document, nous avons une estimation d'environ **25.6 KB**/utilisateur/mois pour les rendez-vous.

Au total, cela fait donc une estimation d'environ **51.4 KB**/utilisateur/mois pour le trafic sortant de Firestore.

Lecture de données Firestore :

Durant l'estimation précédente du trafic sortant de Firestore, il a été nécessaire de calculer le nombre de lecture de documents. En effet, il s'agit du nombre de fois que le document est lu par l'utilisateur. Pour rappel, voici les valeurs obtenues :

Profile-public : 37.5 lectures/mois

Profile-authorized : 7.5 lectures/mois

Profile-authorized/{uid}/kids : 16 lectures/mois

Profile-private : 6 lectures/mois

Meeting : 50 lectures/mois

Cependant, il ne faut pas oublier que Cloud Function (nécessaire à l'envoi des notifications) effectue également une lecture, afin de récupérer le jeton FCM de l'utilisateur. Comme nous partons de l'hypothèse qu'il y a en moyenne 3 rendez-vous effectués et que tous sont acceptés, refusés ou annulés, cela nous donne 6 notifications à envoyer au total, donc 6 lectures supplémentaires par mois.

Au total, il y a donc en moyenne **123 lectures**/utilisateur/mois effectuées.

Ecriture de données Firestore :

Pour estimer correctement le nombre d'écritures dans Firestore, il est nécessaire de bien comprendre le fonctionnement interne de l'application, en particulier quand est-ce que l'utilisateur envoie des données auprès de Firestore :

- Quand l'utilisateur crée/modifie son profil
- Quand l'utilisateur crée/modifie l'un de ses enfants
- Quand l'utilisateur demande un nouveau rendez-vous
- Quand l'utilisateur accepte/refuse/annule un rendez-vous

Pour rappel, nous partons du fait que l'utilisateur demande en moyenne 3 rendez-vous par mois qui sont presque toujours acceptés, refusés ou annulés (95% du temps, donc environ 2.9x). Nous estimons également que le parent a deux enfants et change son profil une fois par mois. Pour les enfants, l'hypothèse qu'un enfant n'est que très rarement créé/modifié soit 1 fois tous les 3 ans, donc environ 0.03x/mois en moyenne, a été faite. Avec ces hypothèses, il est possible d'estimer le nombre d'écritures au niveau de Firestore :

Profile-private, profile-authorized et *profile-public*: 1 écriture/mois

Profile-authorized/{uid}/kids : 0.03 écritures/mois

Meeting : 3 + 2.9 = 5.9 écritures/mois

Au total, il y a donc en moyenne **6.93 écritures**/utilisateur/mois effectuées.

Suppression de données Firestore :

Pour estimer le nombre de suppressions effectuées sur Firebase par chaque utilisateur, il est nécessaire de bien comprendre le fonctionnement de l'application. Dans la pratique, la suppression de document survient à deux moments : quand l'utilisateur décide de supprimer un de ses enfants ou alors quand ce dernier supprime son propre compte.

Ces deux actions peuvent être considérées comme rarissimes. Du côté des rendez-vous, aucune suppression de document n'est possible à travers l'application. Pour estimer le nombre de suppression moyen par utilisateur, l'hypothèse qu'un utilisateur ne supprime son compte ou l'un de ses enfants seulement une fois tous les 5 ans a été utilisée, ce qui nous donne un nombre de d'environ **0.017 suppressions**/utilisateur/mois.

Appel d'une Cloud Functions :

Tout le contexte permettant d'estimer le nombre d'appel mensuel moyen par utilisateur a déjà été donné dans les sections précédentes. En effet, nous savons qu'une fonction est appelée chaque fois qu'un rendez-vous est créé, ainsi que quand le rendez-vous est accepté, refusé ou annulé. D'un autre côté, nous estimons qu'un utilisateur moyen fait 3 demandes de rendez-vous par mois et que toutes ses demandes sont acceptées, refusées ou annulées. Il y a donc **6 appels**/utilisateur/mois.

Utilisation RAM/CPU des Cloud Functions :

À la vue de la très faible quantité de ressources utilisées par la fonction pour envoyer une notification, mais aussi de la difficulté à estimer précisément cette quantité, il n'est pas facile de prévoir par le biais d'estimation et de calcul l'utilisation en RAM et en CPU par les Cloud Functions.

De ce fait, une première solution serait de ne pas prendre en compte l'utilisation RAM/CPU en se disant que l'utilisation est si faible qu'elle en devient presque insignifiante. Du moins, l'on peut penser que la présence ou non de l'utilisation moyenne en CPU/RAM par rapport au nombre d'utilisateurs actifs pour estimer les frais d'utilisation ne change que très peu les valeurs obtenues.

Cependant, une autre solution plus fiable peut également être envisagée. En effet, il est possible de récupérer l'utilisation CPU/RAM lors des tests utilisateurs afin d'estimer l'utilisation moyenne. Vous trouverez ci-dessous les graphiques d'utilisation correspondant à la semaine de test, disponibles grâce à l'outil Google Monitoring [15] :

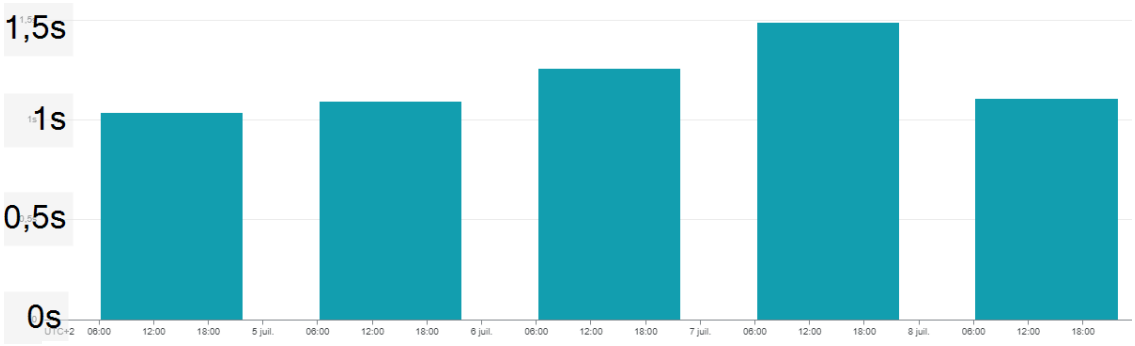

Figure 7 – Temps d'exécution CPU moyen des fonctions par jour, repris de [15]

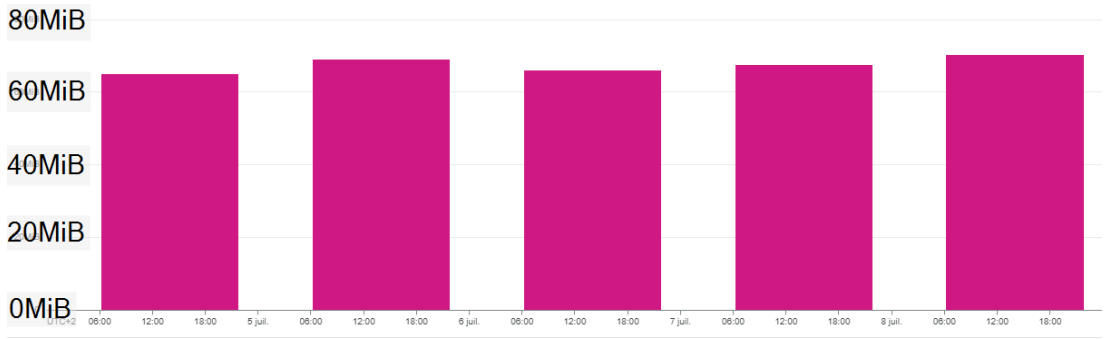

Figure 6 - Utilisation RAM moyenne des fonctions par jour, repris de [15]

La console Google Cloud permet également de déterminer que les fonctions sont exécutées sur un environnement disposant de 256MB de mémoire et un vCPU à 400MHz. Avec ces informations et le fait qu'il y avait 15 utilisateurs lors de tests, il est possible de déterminer l'utilisation moyenne par utilisateur :

- Temps d'exécution moyen : 1,19s
- GHz-secondes moyen : (400 / 1000) * 1,19 = 0,476GHz-s
- Utilisation RAM moyen : 67.6MiB ou 0.0709GB
- GB-secondes moyen : 0.0709 * 1.19 = 0.0843GB-s

Ensuite, avec une estimation de 6 invocations mensuelles par utilisateur on obtient les valeurs suivantes :

- CPU : 0.476 * 6 = **2.856 GHz-s**/utilisateurs/mois
- RAM : 0.0843 * 6 = **0.5058 GB-s**/utilisateur/mois

Trafic réseau Cloud Functions:

Du fait que le trafic en entrée est gratuit et que les fonctions communiquent uniquement avec Firestore et FCM, deux services proposés directement par Google, il n'y a aucun frais d'utilisation du réseau engendré par les Cloud Functions.

Stockage Cloud Storage (photos):

Le stockage des photos de profils des utilisateurs ne se fait pas via Firestore. En effet, seule l'url vers la photo de profil de l'utilisateur est stockée dans les documents Firestore. Les photos de profil des utilisateurs sont stockées dans Firebase Cloud Storage.

Or, il est très difficile d'estimer la taille que prendra la photo de profil d'un utilisateur, et ce pour une bonne raison : ni l'application, ni Firebase ne fait de compression sur la photo envoyée. L'espace de stockage utilisé par l'utilisateur peux donc énormément varier, comme le montre la capture d'écran ci-dessous réalisée durant la phase de test utilisateurs :

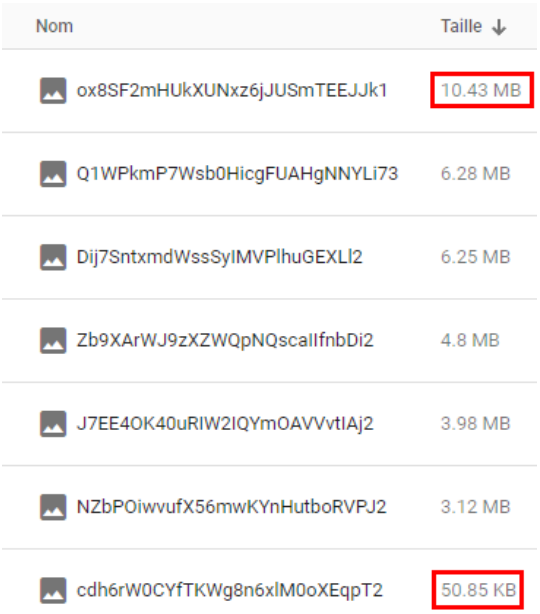

Figure 8 - Capture d'écran de la taille des photos de profil des utilisateurs durant la phase de test

De ce fait, l'utilisation des ressources observées durant la phase de test sera également utilisée afin de mesurer le stockage moyen d'un utilisateur :

- A la fin de la phase de test, il y avait 15 utilisateurs inscrits sur la plateforme
- Sur les 15 utilisateurs, 7 d'entre eux ont mis en ligne une photo de profil
- En moyenne, une photo de profil pèse environ 5MB

Cela nous donne donc un stockage moyen de (5* 7) / 15 = **2.333 MB**/utilisateur.

Trafic réseau Cloud Storage :

Tout comme le trafic Firebase ou concernant les Cloud Functions, le trafic en entrée de Cloud Storage est gratuit. Il n'y a donc pas de frais réseaux engendrés lorsque l'utilisateur met à jour sa photo de profil sur le serveur. En revanche, à chaque fois qu'une photo est demandée par l'application, du trafic sortant (facturable) est généré.

Pour calculer la charge réseau sortant de Cloud Storage, l'on peut reprendre les précédentes estimations et hypothèses que nous avons émises :

- Une photo pèse en moyenne 2.333MB (en prenant en compte les utilisateurs sans photo de profil)
- La photo est téléchargée à chaque fois qu'un profil est affiché
- L'utilisateur affiche en moyenne son propre profil 5 + 1 (modification) fois par mois
- L'utilisateur affiche le profil d'environ 30 autres utilisateurs par mois

Nous pouvons donc estimer un trafic sortant pour Cloud Storage de 2.333 * (6 + 30) = **84 MB**/utilisateur/mois.

Lecture/écriture de fichiers Cloud Storage :

Avec les hypothèses et estimations déjà présentes, le nombre de lectures et d'écritures de photo de profil a déjà été calculé :

- Lecture : **36 fichiers/**utilisateur/mois
- Ecriture : **1 fichier**/utilisateur/mois

Autocomplétion Google Maps API :

A chaque fois que l'utilisateur doit spécifier un lieu ou une adresse, que ce soit pour définir ou mettre à jour son adresse de domicile ou lors d'une recherche, il envoie différentes requêtes à l'API Google Maps. En particulier, à chaque fois que l'utilisateur entre un caractère supplémentaire, une requête de type « autocomplétion » est envoyée afin d'obtenir les lieux ou adresses répondant à la requête de l'utilisateur.

Pour implémenter et utiliser l'API, l'application doit également générer un jeton d'identification unique et inclure le jeton à travers toutes les requêtes. Grâce au jeton, l'API peut alors générer une session pour l'utilisateur et avoir accès aux précédentes requêtes effectuées par l'utilisateur. De plus, la facturation s'effectue en termes de sessions et non en termes de requêtes, générant ainsi moins de frais.

En revanche, le même jeton ne peut pas être utilisé entre deux recherches qui n'ont pas de lien entre elles. Ainsi, il n'est par exemple pas possible d'utiliser le même jeton à la fois lorsque l'utilisateur entre son adresse de domicile et lorsque ce dernier effectue une recherche, car les deux lieux peuvent être totalement différents.

Dans les estimations précédentes, nous sommes partis du principe que l'utilisateur effectue en moyenne trois recherches différentes par mois. Evidemment, si jamais la personne décide d'utiliser sa localisation GPS et non d'écrire un lieu, aucune requête de type « autocomplétion » n'est envoyée. Pour estimer le nombre de session moyen, l'hypothèse que l'utilisateur utilise une fois sur deux l'autocomplétion et l'autre fois le GPS pour rechercher un lieu ou une adresse a été faite.

De ce fait, le nombre de session d'autocomplétion de l'API Google Maps et de **1.5 sessions**/utilisateur/mois

Places Google Maps API :

En plus des requêtes d'autocomplétion, il est nécessaire d'effectuer une requête supplémentaire une fois que l'utilisateur a choisi son adresse, ce afin de récupérer les informations détaillées tels que les coordonnées GPS de l'adresse en question. En effet, les réponses d'autocomplétion ne contiennent pas cette information.

Pour estimer le nombre de requêtes de type « Place », il a été supposé que l'utilisateur effectue toujours un choix lorsque ce dernier effectue une recherche. Il y a donc également **1.5 sessions**/utilisateur/mois.

Geocoding Google Maps API :

A chaque fois que l'utilisateur décide d'utiliser le GPS pour entrer un lieu ou une adresse, une requête de type « Geocoding » est envoyée à l'API Google Maps. Cette requête permet de récupérer les informations sur un lieu telles que le nom de l'adresse à partir des coordonnées GPS de l'utilisateur.

Comme expliqué précédemment, l'utilisateur effectue en moyenne 3 recherches par mois et utilise le GPS une fois sur deux. Il y a donc **1.5 requêtes**/utilisateur/mois.

Résumé :

Grâce aux différentes estimations réalisées sur chacun des services utilisés par l'application, il est à présent possible de dresser un profil type d'utilisation moyenne d'un utilisateur. Le tableau suivant récapitule les estimations réalisées pour les services facturés :

Service	Utilisation moyenne mensuelle (1 utilisateur)
Stockage Firestore	11KB (unique) + 14KB (mensuel)
Trafic réseau Firestore (sortie)	51.4KB
Ecriture de documents Firestore	6.93 écritures
Lecture de documents Firestore	123 lectures
Suppression de documents Firestore	0.017 suppressions
Appel d'une Cloud Function	6 appels
Utilisation RAM en GB-secondes Cloud Functions	0.5058GB-s

Service	Utilisation moyenne mensuelle (1 utilisateur)
Utilisation CPU en GHZ-secondes Cloud Functions	2.856GHz-s
Trafic réseau Cloud Functions (sortie)	0B
Stockage Cloud Storage (photos)	2.333MB
Trafic réseau Cloud Storage (sortie)	84MB
Ecriture de fichiers Cloud Storage	36 fichiers
Lecture de fichiers Cloud Storage	1 fichier
Autocomplétion Google Maps API	1.5 requêtes
Place Google Maps API (informations détaillées)	1.5 requêtes
Geocoding Google Maps API	1.5 requêtes

Avec ces valeurs, il est à présent possible de calculer à partir de combien d'utilisateurs moyen mensuel l'application devient payante, mais aussi de quelle manière augmentent les frais en fonction du nombre d'utilisateurs. Les graphiques ci-dessous montrent l'évolution des frais d'utilisation en fonction du nombre d'utilisateurs :

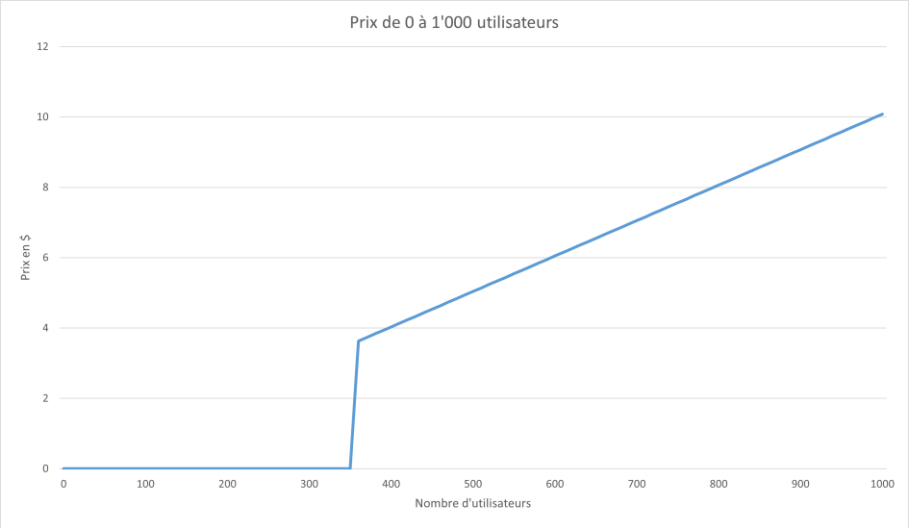

Figure 9 - Graphique d'estimation des frais Google de 0 à 1000 utilisateurs

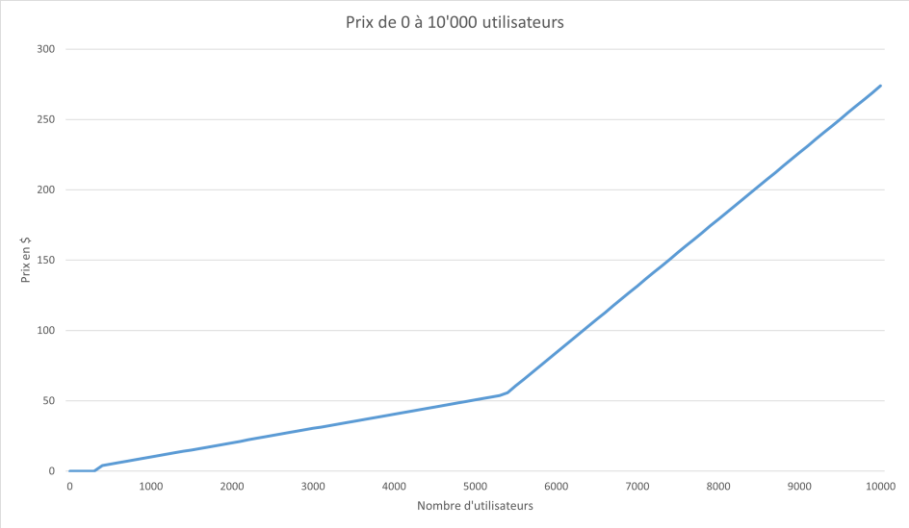

Figure 10 - Graphique d'estimation des frais Google de 0 à 10'000 utilisateurs

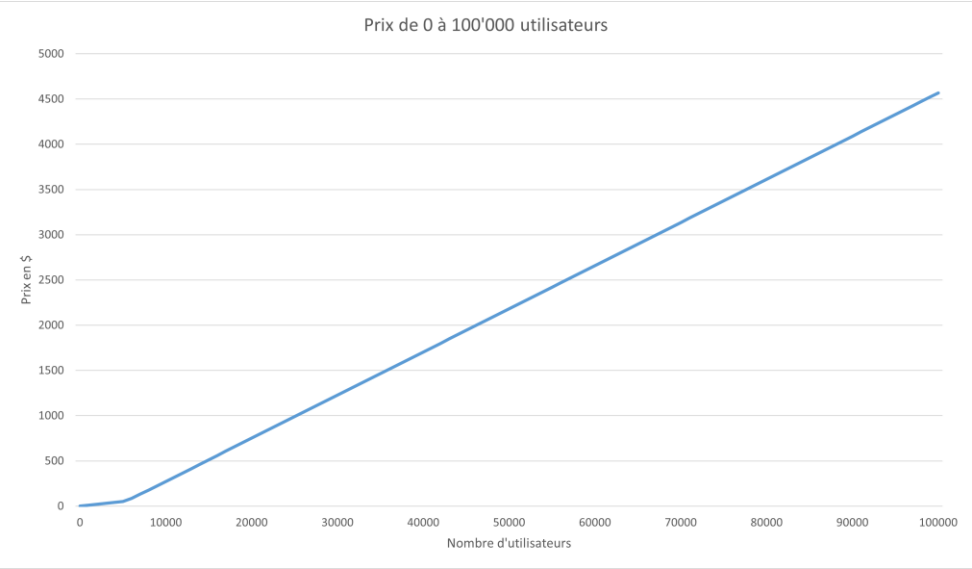

Figure 11 - Graphique d'estimation des frais Google de 0 à 100'000 utilisateurs

Sur ces graphiques, il est possible d'observer plusieurs points intéressants :

- En dessous d'environ 350 utilisateurs, il n'y a aucun frais généré.
- À partir d'environ 5'500 utilisateurs, la pente de la courbe change. Cela indique qu'il y a une augmentation des frais par utilisateur à ce moment donné.
- Au-delà de 5'500 utilisateurs, il n'y a plus de changement dans la courbe et l'on constate une augmentation linéaire (observée jusqu'à 1 million d'utilisateurs).
- Entre 350 et 5'500 utilisateurs, la pente est d'environ 1%. Cela veut dire que chaque utilisateur coûte en moyenne 0.01$ de frais d'utilisation des services de Google.
- À partir de 5'500 utilisateurs, la pente passe à environ 4.76%. Cela veut dire que chaque utilisateur coûte en moyenne 0.0476$ de frais d'utilisation.

Le changement de pente vers 5'500 utilisateurs s'explique par le fait que les 200$ mensuel offerts pour l'utilisation de l'API Google Maps sont dépassés à partir d'un tel nombre d'utilisateur, comme le montre la capture d'écran ci-dessous :

NB Utilisateurs	15. Autocomplétion Google Maps API		16. Place Google Maps API		17. Geocoding Google Maps API		15 + 16 + 17. Google Maps API
5000	7500	21,225	7500	127,5	7500	37,5	0
5100	7650	21,6495	7650	130,05	7650	38,25	0
5200	7800	22,074	7800	132,6	7800	39	0
5300	7950	22,4985	7950	135,15	7950	39,75	0
5400	8100	22,923	8100	137,7	8100	40,5	1,123
5500	8250	23,3475	8250	140,25	8250	41,25	4,8475
5600	8400	23,772	8400	142,8	8400	42	8,572
5700	8550	24,1965	8550	145,35	8550	42,75	12,2965
5800	8700	24,621	8700	147,9	8700	43,5	16,021
5900	8850	25,0455	8850	150,45	8850	44,25	19,7455
6000	9000	25,47	9000	153	9000	45	23,47

Cependant, une autre question se pose : avec pas moins de 17 services facturables différents utilisés, comment se fait-t-il qu'il n'y aie pas plus de variation sur la pente au fil de l'augmentation du nombre d'utilisateurs ?

En effet, à chaque fois qu'un quota est dépassé le service devient alors facturable ce qui engendre forcément une augmentation des frais, ce qui se traduit par une augmentation de la pente. Bien-sûr, si les frais du service en question ne forment qu'une infime partie des frais totaux, la variation de la pente sera elle aussi très faible.

Pour autant, ici il n'y a pas de variation ou alors une variation si faible qu'elle n'est même pas visible sur le graphique. Comment expliquer cela ?

Si l'on s'intéresse au pourcentage de frais facturé pour chaque service, l'on comprend très vite pourquoi : presque l'intégralité des frais d'utilisation est générée par ces deux services en particulier : le trafic réseau Cloud Storage (25% des frais en moyenne) et l'utilisation de l'API Google Maps (74% des frais en moyenne). A eux seuls, ces deux services sont à l'origine de 99% des frais d'utilisation de notre application !

Ces résultats montrent qu'il est nécessaire de concevoir un modèle économique permettant de couvrir les frais d'utilisation des services de Google à hauteur de 0.01$ en dessous de 5'500 utilisateurs et 0.0476$ en dessus d'un tel nombre d'utilisateurs, sans quoi le déploiement de l'application sur le marché risque d'être à perte !

Il ne faut cependant pas oublier que cette estimation est incomplète : en plus des nombreuses hypothèses émises afin de définir le profil d'utilisation d'un utilisateur moyen, il y a également certaines facilités qui ont été prises lors des calculs permettant d'arriver aux chiffres mentionnés ci-dessus. En effet, bien que la majorité des quotas d'utilisation gratuite des services de Google soit établis sur un mois, certains d'entre eux sont calculés et réinitialiser chaque jour. C'est notamment le cas des services suivants :

- Lecture, écriture et suppression de documents Firestore
- Trafic réseau Cloud Storage (photos de profil)
- Lecture, écriture et suppression de documents Cloud Storage

Afin de « traduire » ces quotas d'utilisation vers un quota mensuel, une multiplication par 30 (nombre de jours dans un mois) à été faite. Pour autant, ce changement ne reflète pas forcément une réelle utilisation de l'application.

2.1.3 Tests utilisateurs

Parmi les objectifs principaux de ce projet, le dernier d'entre eux concerne la réalisation de tests utilisateurs, qui consiste à évaluer les différentes fonctionnalités présentes mais également l'expérience utilisateur, abrégé UX (**U**ser e**X**perience). Néanmoins, pour comprendre comment mesurer ces différents aspects durant la phase de test, il est nécessaire d'analyser les outils et les méthodes permettant de réaliser des tests utilisateurs satisfaisants et de concevoir des scénarios de test et des questionnaires.

En premier lieu, il est intéressant d'étudier l'objectif visé en accomplissant des tests utilisateurs. Après quelques recherches sur différents articles, il y a trois avantages principaux que l'on retrouve sur différentes sources [16] [17] [18] :

- Identifier des problèmes (de design, de fonctionnement, etc.)
- Découvrir de nouvelles approches
- Comprendre le comportement et les préférences des utilisateurs

Ces éléments sont très importants si l'on souhaite réaliser une application répondant à la demande des utilisateurs. En effet, il existe de nombreux cas d'applications (mobile ou non) où les tests utilisateurs ont permis d'identifier de sérieux problèmes de design ou encore des

à priori sur les utilisateurs. Grâce aux tests utilisateurs, il sera alors possible de proposer des améliorations futures concrètes et permettant réellement d'avoir un impact sur ce projet.

Maintenant que les objectifs ont été identifiés, il est nécessaire de s'intéresser au déroulement des tests. Premièrement, l'on peut se demander quel sont les méthodes existantes permettant de réaliser des tests utilisateurs. Il en existe principalement quatre :

- Sur place, avec assistance humaine (p.ex : un développeur)
- Sur place, sans assistance
- À distance, avec assistance
- À distance, sans assistance

Dans le cadre de ce projet de Bachelor, les tests utilisateurs visent à « simuler » une utilisation réelle de l'application. Ainsi, le choix de réaliser des tests sur place, généralement avec des équipements permettant d'enregistrer le son et la vidéo autour de l'utilisateur de test, n'est pas le plus approprié. En effet, un déroulement à distance des tests par les utilisateurs se rapproche bien plus d'une réelle utilisation de l'application. Pour les mêmes raisons, il est préférable de ne pas fournir d'assistance humaine lors des tests utilisateurs. Néanmoins, il a été décidé que des procédures de réalisation des tests seront réalisées, dans le but d'aider l'utilisateur si jamais ce dernier se retrouve bloqué.

Une autre question dont il est important de répondre est combien d'utilisateurs il est nécessaire de prévoir pour la réalisation des tests. Selon une étude de Jackob Nielsen et Thomas K. Landauer [19], il apparaît que cinq utilisateurs arrivent en moyenne à identifier plus de 75% de tous les problèmes d'utilisabilité de l'application, comme le montre ce graphique ci-dessous :

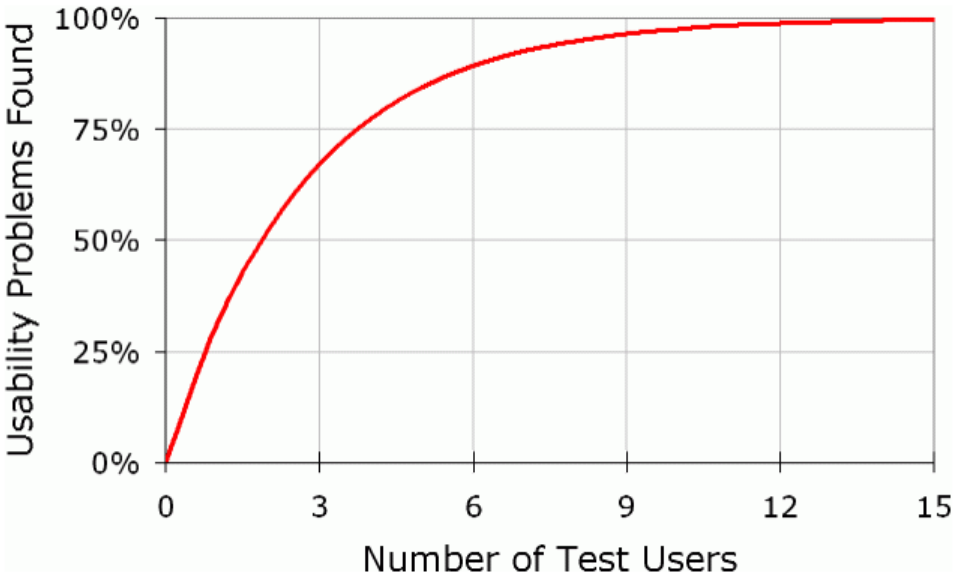

Figure 12 - Rapport entre l'identification de problème et le nombre d'utilisateurs de test, repris de [16]

Cette étude est très importante car elle montre qu'il n'est pas nécessaire d'avoir beaucoup d'utilisateurs pour réaliser une phase de test significative. Néanmoins, l'on peut se demander pourquoi il n'est pas optimal d'avoir une quinzaine d'utilisateurs et ainsi couvrir presque 100% des problèmes d'utilisabilité. Dans l'étude, cela est expliqué par une courbe entre les coûts/bénéfices et le nombre d'utilisateurs de tests :

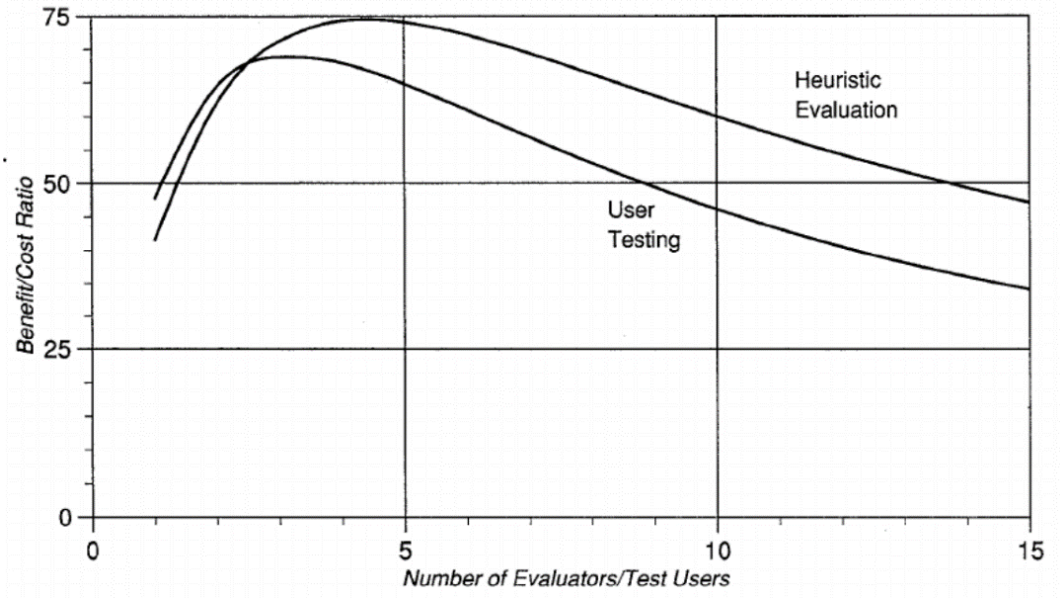

Figure 13 - Rapport entre les coûts/bénéfices et le nombre d'utilisateurs de test, repris de [19]

Néanmoins, pour arriver à ces résultats les chercheurs ont estimé entre 420$ et 520$ de frais par utilisateur. Bien évidemment, ce n'est pas le cas dans ce projet de Bachelor.

De plus, comme notre application vise à mettre en relation les différents utilisateurs entre eux, il est préférable d'avoir un maximum d'utilisateur. En effet, plus il y a d'utilisateurs inscris, plus il y a un vaste choix de recherche pour les parents désirant trouver une nounou. S'il n'y a pas assez d'utilisateurs, un nombre significatif de testeurs risquent de ne pas pouvoir explorer toutes les fonctionnalités de l'application, soit parce qu'ils n'ont pas trouvé de nounou dans leur région, soit car aucun parent ne leur a envoyé de demande de rendez-vous.

Pour ces raisons, il serait intéressant d'avoir plutôt 15 à 20 utilisateurs de test plutôt que 5, avec une part à peu près égale de parents et de nounous. Concernant l'âge des utilisateurs, comme l'application se veut être utilisée à la fois par des adolescents ou des jeunes adultes, par des jeunes parents ou encore par des personnes retraitées, il serait intéressant d'avoir une grande diversité sur l'âge des utilisateurs de test.

Grâce à l'analyse des outils et des méthodes permettant de réaliser des tests utilisateurs, il est à présent possible de concevoir les différents scénarios de test avec les objectifs que les testeurs devront réaliser. Vous trouverez de plus amples détails sur la conception des scénarios dans la section « 2.2.4 – Tests utilisateurs ».

D'un autre côté, pour s'assurer que les tests utilisateurs ne génèrent pas de frais liés à l'utilisation de Firebase et de l'API Google Maps, l'on peut reprendre les estimations d'utilisation des services de Firebase par utilisateur réalisées pendant la phase d'analyse du modèle économique. Il a été estimé qu'en dessous de 350 utilisateurs l'application ne génère pas suffisamment de trafic pour engendrer des frais. Avec 15 utilisateurs de test, il n'y a donc aucune chance qu'une utilisation dépassant le quota disponible gratuitement pour un service ne soit dépassé.

Maintenant qu'il est possible de réaliser des tests utilisateurs significatifs, il faut encore recueillir le *feedback* de l'utilisateur, que ce soit sa satisfaction, ses commentaires, etc. Pour ce faire, il est nécessaire d'établir un questionnaire à la suite de la réalisation de chaque test. Ce questionnaire doit être établi de façon à identifier les problèmes d'utilisabilité, de design, etc. par l'utilisateur, mais également à quantifier le problème.

Pour ce faire, plusieurs échelles de mesure de l'utilisabilité ont été étudiés [20], notamment les échelles ASQ (**A**fter **S**cenario **Q**uestionnaire) [21] et SEQ (**S**ingle **E**ase **Q**uestion) [22] qui ont été intégrées aux questionnaires pour ce projet.

Le questionnaire ASQ est principalement utilisé lorsque l'utilisateur doit résoudre un scénario, permettant d'identifier la difficulté relative à la réalisation de la tâche. C'est donc une échelle qui s'adapte parfaitement avec notre modèle de test utilisateur. Le questionnaire ASQ est composé de trois affirmations, avec une échelle de 1 à 7 permettant à l'utilisateur d'exprimer son accord (1 = pas du tout d'accord, 7 = tout à fait d'accord) :

1. De manière générale, je suis satisfait de la simplicité de résolution de ce scénario
2. De manière générale, je suis satisfait de la durée de résolution de ce scénario
3. De manière générale, je suis satisfait du support utilisateur disponible pour ce scénario

Le questionnaire SEQ, lui, porte uniquement sur un objectif du scénario. Il s'agit d'une question simple, également mesurée de 1 à 7 (1 = très difficile, 7 = très simple) :

- Quelle difficulté accordez-vous à la réalisation de cet objectif ?

Ainsi, il est possible de mesurer, pour chaque objectif du scénario, son utilisabilité dans le cadre de l'application. En effet, si une majorité de personnes ont rencontré des difficultés à réaliser la tâche, cela témoigne d'une mauvaise expérience utilisateur, probablement du fait d'un problème de design de l'application. Grâce aux échelles ASQ et SEQ, il est à présent possible de concevoir un questionnaire générique distribué pour chaque scénario que l'utilisateur devra réaliser.

Néanmoins, il est également intéressant d'évaluer le ressenti de l'utilisateur sur son utilisation globale de l'application. Il faut alors prévoir, en plus des questionnaires accompagnant les scénarios, un questionnaire de satisfaction globale que le testeur pourra remplir une fois l'intégralité des tests réalisés. Dans un tel contexte, il n'est pas vraiment possible de réutiliser les échelles ASQ et SEQ et d'autres échelles ont été analysées.

Il existe de nombreux modèles permettant de juger l'utilisabilité d'une application ou plus généralement d'un système tout entier. Parmi ces modèles, l'on retrouve notamment l'échelle SUS (**S**ystem **U**sability **S**cale), composé de 10 affirmations et dont l'analyse des résultats permet d'attribuer une « note d'utilisabilité » pour le système. Pour chaque affirmation, l'utilisateur peux répondre sur une échelle allant de 1 à 5, 1 signifiant « Pas du tout d'accord » et 5 « Tout à fait d'accord » [23] :

1. Je pense que je vais utiliser ce système fréquemment
2. Ce trouve que ce système est inutilement complexe
3. Je trouve que ce système est facile d'utilisation
4. Je pense avoir besoin d'une assistance technique pour utiliser ce système
5. Je trouve que les différentes fonctionnalités de ce système sont bien intégrées
6. Je trouve qu'il y a trop d'incohérence dans ce système
7. J'imagine que la plupart des gens apprendront rapidement à utiliser ce système
8. J'ai trouvé ce système très lourd à utiliser
9. Je me suis senti(e) très en confiance en utilisant ce système
10. J'ai dû apprendre beaucoup de choses avant de pouvoir utiliser ce système

La note obtenue varie entre 0 et 100. En fonction de la note, il est alors possible de mesurer l'utilisabilité global de notre application par rapport aux notes obtenues pour d'autres systèmes. En 2009, une étude a analysé et comparé les scores SUS de nombreux systèmes afin d'établir une échelle de la qualité du système en fonction de la note obtenue [24] :

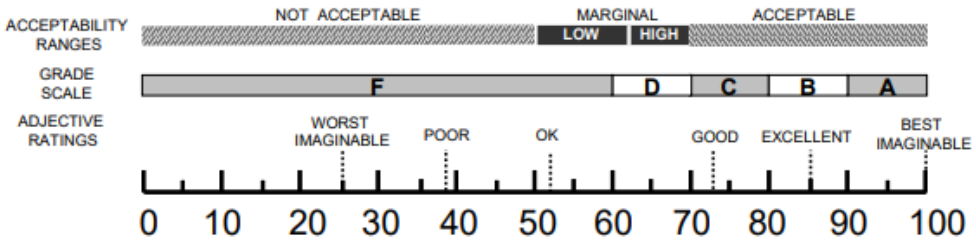

Figure 14 - Signification du score du SUS, repris de [24]

Pour autant, le modèle SUS n'est pas le seul modèle permettant de mesurer l'utilisabilité. Il existe par exemple le modèle SUPR-Qm, spécialement conçu pour mesurer l'utilisabilité d'applications mobiles. De plus, ce modèle sert également à mesurer l'expérience utilisateur de façon globale (évaluation subjective de l'utilisateur). Initialement, le modèle SUPR-Qm devait être présent dans le questionnaire, cependant deux problèmes ont été rencontrés :

Premièrement, certaines questions du SUS et du SUPR-Qm sont presque identiques voir complètement identiques. Si l'on pose ces questions plusieurs fois à travers le questionnaire, cela aurait pour conséquence de rendre l'utilisateur confus voir même de le vexer. Deuxièmement, l'étude et la conception du modèle SUPR-Qm a été réalisée par une entreprise privée, et les outils permettant d'analyser les réponses du formulaire SUPR-Qm ne sont pas disponibles gratuitement, mais pour 29$ [25].

Pour ces raisons, et comme ce n'est pas clairement expliqué si l'on est autorisé à utiliser le modèle sans pour autant acheter la licence, il a été décidé que le formulaire SUPR-Qm ne serait finalement pas présent dans le questionnaire final.

Dans un tout autre contexte, de la même manière que le modèle SEQ pour les questionnaires portant sur les scénarios, il existe également un modèle composé d'une unique question s'appliquant à une application ou un système en général. Il s'agit du modèle NPS (**N**et **P**romoter **S**core). Ce dernier ne sert pas à mesurer l'utilisabilité ou encore l'UX, mais la fidélité des utilisateurs. En effet, le formulaire NPS possède la question suivante, avec une échelle de 1 à 10 (1 = pas du tout, 10 = tout à fait) [26]:

- Quelle est la probabilité que vous recommandiez ce produit à un ami ou collègue ?

En fonction de la réponse donnée par l'utilisateur, ce dernier est classé selon trois catégories :

- Détracteurs *(réponse de 1 à 6)*
- Passifs *(réponse de 7 ou 8)*
- Promoteurs *(réponse de 9 ou 10)*

Afin d'obtenir la note NPS, il suffit de soustraire le pourcentage de détracteurs au pourcentage de promoteurs. La note varie donc entre -100 et 100. Il est ensuite possible de comparer la note obtenue avec d'autres services par le biais de « *Benchmark* » réalisés par différentes compagnies tels que Satmetrix, qui évalue le score NPS obtenue non pas pour un système ou un service mais pour une industrie tout entière. Pour la catégorie « *Sowtware & Apps* », en 2021, l'on retrouve les valeurs suivantes [27] :

- Score NPS moyen : 34
- Meilleur score obtenu : 54 par l'entreprise *Turbo Tax*

De manière générale, l'on peut considérer le système comme bon à partir d'un score supérieur à 0 et comme excellent à partir d'un score supérieur à 50 [28] :

Figure 15 - Spectre de classification du système en fonction du score NPS, repris de [28]

Maintenant qu'il est possible de mesurer l'utilisabilité et la fiabilité des utilisateurs, il reste encore à mesurer l'expérience utilisateur globale qu'a vécu l'utilisateur durant les tests. Pour cela, il était initialement prévu d'utiliser le modèle UEQ permettant d'effectuer des mesures sur différents aspects de l'UX. Il s'agit d'un modèle dont la traduction et les outils d'analyse sont disponibles publiquement, sur le site officiel du modèle UEQ [29].

Seul petit problème, le questionnaire est composé de 26 questions, ce qui est assez conséquent surtout si l'utilisateur doit également remplir les champs associés aux modèles SUS et NPS auparavant. Heureusement, il existe une version raccourcie du modèle, avec uniquement 8 questions, appelé UEQ-S.

Cependant, après quelques recherches, un autre modèle permettant de mesurer et de séparer différemment les aspects de l'expérience utilisateur a été étudié : le modèle AttrakDiff. Originellement composé de 28 questions, il existe également une version courte de seulement 10 questions, et les questions ont été traduites en français [30].

La question se pose alors : vaut-t-il mieux implémenter le formuler UEQ ou AttrakDiff pour mesurer l'expérience utilisateur ? Selon une étude, dont le but est justement de comparer les différents modèles de mesure de l'UX [31], il s'emblerait qu'il n'y ait pas vraiment de différences dans l'interprétation des résultats, du moment que le questionnaire n'est posé qu'une seule fois. Si en revanche le questionnaire est donné plusieurs fois au fur et à mesure de l'évolution du système, AttrakDiff semble donner des résultats plus stables que UEQ.

Pour cette raison et par préférence personnelle, il a donc été décidé que le modèle court d'AttrakDiff serait implémenté. Ce dernier est composé de 8 échelles différentes avec un score entre 1 et 7 [32]:

Items (dans l'ordre de passation)		
Simple	○ ○ ○ ○ ○ ○ ○	Compliqué
Laid	○ ○ ○ ○ ○ ○ ○	Beau
Pratique	○ ○ ○ ○ ○ ○ ○	Pas pratique
De bon goût	○ ○ ○ ○ ○ ○ ○	De mauvais goût
Prévisible	○ ○ ○ ○ ○ ○ ○	Imprévisible
Bas de gamme	○ ○ ○ ○ ○ ○ ○	Haut de gamme
Sans imagination	○ ○ ○ ○ ○ ○ ○	Créatif
Bon	○ ○ ○ ○ ○ ○ ○	Mauvais
Confus	○ ○ ○ ○ ○ ○ ○	Clair
Ennuyeux	○ ○ ○ ○ ○ ○ ○	Captivant

Figure 16 - Questions du formulaire AttrakDiff en version courte, repris de [32]

Pour résumer, avec les modèles ASQ et SEQ il est possible de mesurer individuellement l'utilisabilité de chaque fonctionnalité présente dans l'application, tandis que les modèles SUS, NPS et AttrakDiff permettent de mesurer l'utilisabilité, la fidélité ainsi que l'expérience utilisateur de l'application globalement.

2.2 Conception

Cette sous-section présente la conception des architectures réalisées dans le cadre de ce projet de Bachelor. Dans un premier temps, l'architecture déployée sur Flutter sera présentée afin de bien comprendre le fonctionnement et le cycle de vie de l'application. Ensuite, l'architecture déployée sur Firebase pour la gestion des données et les répercussions sur le code seront étudiées.

2.2.1 Architecture Flutter

Pour réaliser une bonne application mobile, il est nécessaire de concevoir une architecture du code qui sera utilisée pour l'implémentation de toutes les fonctionnalités de notre application. Ainsi, la structure généralisée de l'application offre plus facilement la possibilité d'ajouter de nouvelles fonctionnalités ou encore de réaliser des tests sur chaque fonctionnalité indépendamment des autres. Dans l'implémentation actuelle, aucune architecture n'a vraiment été réfléchie et l'on ne retrouve pas vraiment de généralité. Pour combler ce

problème, une nouvelle architecture a été imaginée en se basant sur les architectures MVVM (**M**odel-**V**iew-**V**iew**M**odel) et Bloc.

Cette nouvelle architecture profite également de la mise à jour en temps réel des données fournie par Firebase. En effet, grâce à la méthode `.snapshot()` il est possible d'obtenir un flux sur la référence indiquée, sous forme de d'objet `Stream<>` Flutter. Evidemment, cette fonctionnalité n'est pas utile pour toutes les fonctionnalités de notre application. Par exemple, pour la recherche d'utilisateur proche une méthode plus « classique », par le biais d'une requête et de l'attente de la réponse, est utilisée (sous forme d'objet `Future<>`).

Actuellement, comme expliqué précédemment, le prototype ne suit pas de structure particulière et les données sont traités différemment selon les vues et/ou le type de données. L'utilisation des Stream n'est pas non plus présente, hormis l'authentification de Firebase.

La figure ci-dessous représente l'organisation des différentes classes et objets présents dans l'application :

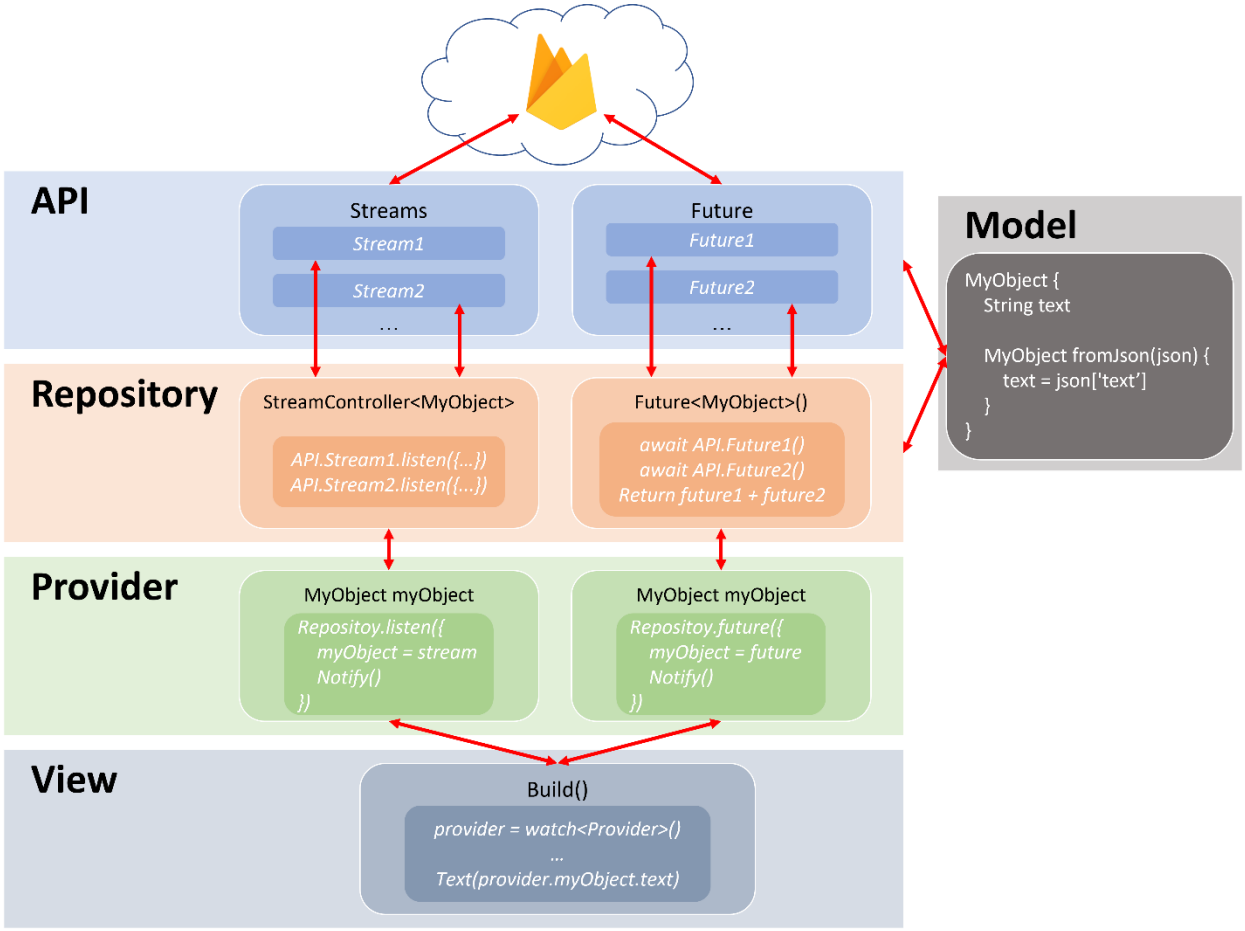

Figure 17 – Schéma de l'architecture de l'application Flutter

Sur cette figure, l'on distingue 5 catégories de classes différentes, à l'inverse d'une architecture MVVM qui ne contient généralement que trois catégories. Pour bien comprendre le fonctionnement de l'architecture, il est nécessaire de comprendre le but et le fonctionnement de chacune des catégories présentes :

API

Cette première couche a pour objectif de récupérer et de fournir les informations provenant de l'extérieur de l'application, comme c'est le cas pour les données Firebase ou les informations obtenues à partir de Google Maps. En fonction du type de données, ces dernières devront être accessibles par le biais d'une réponse unique sous forme de `Future<>` ou par le biais de plusieurs réponses pour garantir une mise à jour en temps réel, sous forme de `Stream<>`.

Dépendamment de la requête, c'est également l'API qui a la charge de traduire les données obtenues en objet reconnaissable et utilisable par les autres couches de notre application. Pour cela, les méthodes peuvent utiliser la fonction `fromJson(json)` présente dans les modèles implémentés. Il est parfois également nécessaire de réaliser directement la traduction car plusieurs types de requêtes aux structures différentes peuvent être retournées, comme pour l'API Google Maps. Pour cela, la méthode `jsonDecode(json)` permet de traduire la réponse en un objet Flutter plus facilement traitable.

Repository

Une fois les méthodes pour récupérer et écouter les données implémentées dans l'API, il est parfois nécessaire de traiter ou de regrouper certaines méthodes avant d'être utilisées. C'est notamment le cas pour les requêtes écoutant en temps réel les informations de l'utilisateur : si l'utilisateur actuel se déconnecte et change de compte, alors il faut que les flux arrêtent d'écouter sur l'ancien compte et se mettent à écouter les informations du nouveau compte.

Or, sans cette couche, chacune des couches suivantes qui utilise une méthode impliquant l'utilisateur actuel devraient gérer et traiter le changement de compte, ce qui amène à de la duplication de code et à une décentralisation. Pour éviter cela, le repository s'occupe de gérer toutes les méthodes présentes dans l'API. Quand c'est nécessaire, l'information n'est pas directement accessible mais il faut passer par un « objet » du repository qui s'occupe de gérer les requêtes en amont. Nous avons vu le cas avec les Stream, mais cela peut également être le cas quand plusieurs requêtes doivent être effectuées et combinées avant de retourner l'objet. Dans ces cas, ce n'est pas l'API mais le repository qui s'occupe de traduire les éléments en objets utilisables par notre application, via les modèles.

S'il n'est pas nécessaire d'apporter plus de traitement à la requête, alors une méthode se contentant simplement d'appeler et de retourner le résultat de l'API est disponible pour la couche supérieure.

Provider

La couche suivante s'occupe de notifier les vues lors d'une réponse obtenue par le repository. Pour cela, le provider dispose d'un état et la vue s'adapte en fonction de l'état et des données du provider. Du fait que la gestion des flux se fait au niveau du repository, le provider peut simplement se contenter d'écouter sur les flux obtenus du repository, et mettre à jour son état lors de la réception de données.

Pour ce faire, le provider utilise les méthodes `.listen()` des Stream ou encore `.then()` pour les objets `Future`. Pour communiquer avec les vues, tous les providers héritent de la classe `ChangeNotifier`, ce qui leur permet d'appeler la méthode `notify()`. A chaque fois que la méthode est appelée, toutes les vues utilisant le provider seront recalculées en fonction du nouvel état du provider.

A l'inverse d'un ViewModel, il n'y a pas forcément un provider implémenté pour chaque vue mais plutôt un provider implémenté pour chaque fonctionnalité. Ainsi, si plusieurs vues dépendent des mêmes données, elles peuvent utiliser le même provider tout en choisissant de présenter les données différemment. En effet, l'avantage d'un provider étant que plusieurs vues peuvent utiliser un même provider mais également qu'une vue peut écouter sur plusieurs providers simultanément.

View

Enfin, cette dernière couche s'occupe d'afficher les données présentes dans les différents providers à l'utilisateur. Pour cela, un `MultiProvider` est intégré à la racine de l'application, incluant tous les providers au contexte de l'application. Cela permet dans un premier temps d'assurer que toutes les vues de notre application utilisent les mêmes providers, et qu'aucun n'a été dupliqué au cours de l'utilisation.

De plus, les vues peuvent utiliser le contexte afin d'écouter sur un ou plusieurs providers. Si une vue écoute sur un provider et que ce dernier appelle la méthode `notify()`, alors la vue est recalculée avec les nouvelles données.

Bien sûr, il est aussi possible qu'une vue souhaite simplement appeler une méthode du provider sans pour autant écouter les mises à jour de ce dernier. Grâce au contexte et au `MultiProvider`, les vues ont en effet la possibilité d'utiliser le provider sans pour autant écouter les modifications.

Dans la pratique, en début de chaque méthode `build()` des vues, les providers utilisés sont récupérés, comme le montre l'extrait de code ci-dessous :

```
@override
Widget build(BuildContext context) {
  final profileProvider = context.watch<ProfileProvider>();
  final placeProvider = context.read<PlaceProvider>();
  ...
}
```

Dans cet extrais, le mot-clé « read » indique que la vue souhaite uniquement utiliser le provider mais ne pas être notifiée lors d'une modification de ce dernier, tandis que le mot-clé « watch » permet à la vue d'être actualisée. De cette manière, les vues implémentées n'ont pas à se soucier de l'état ou du traitement des données, mais simplement de récupérer et d'afficher les valeurs présentes dans le provider au moment de la construction de la vue.

Model

Cette dernière catégorie n'est pas vraiment une couche à part entière mais est utilisée par toutes les autres couches. Dans notre application, il est nécessaire des généraliser la structure des données et notamment de centraliser toutes les données communes. Pour cela, des modèles sont disponibles ainsi que des méthodes pour traduire ces modèle vers/hors de l'application.

Grâce aux modèles, les données restent cohérentes entre toutes les couches de l'application et il y a moins de risques qu'une erreur survienne car une donnée est manquante.

2.2.2 Structure Firebase

Afin d'assurer un accès optimal et sécurisé aux différentes données de l'application, il est nécessaire de concevoir une structure de la base de données. Or, la structure actuellement en place ne satisfait pas pleinement ces deux objectifs. La raison étant que la structure initialement conçue n'est pas compatible avec Firebase Firestore. En effet, cet outil a de nombreuses différences par rapport à une structure classique.

Premièrement, il existe seulement trois type d' « objets » différents au sein de Firestore :

- Collection
- Document
- Champ (clé-valeur JSON)

La figure ci-dessous montre l'organisation entre ces différents objets au sein de Firebase :

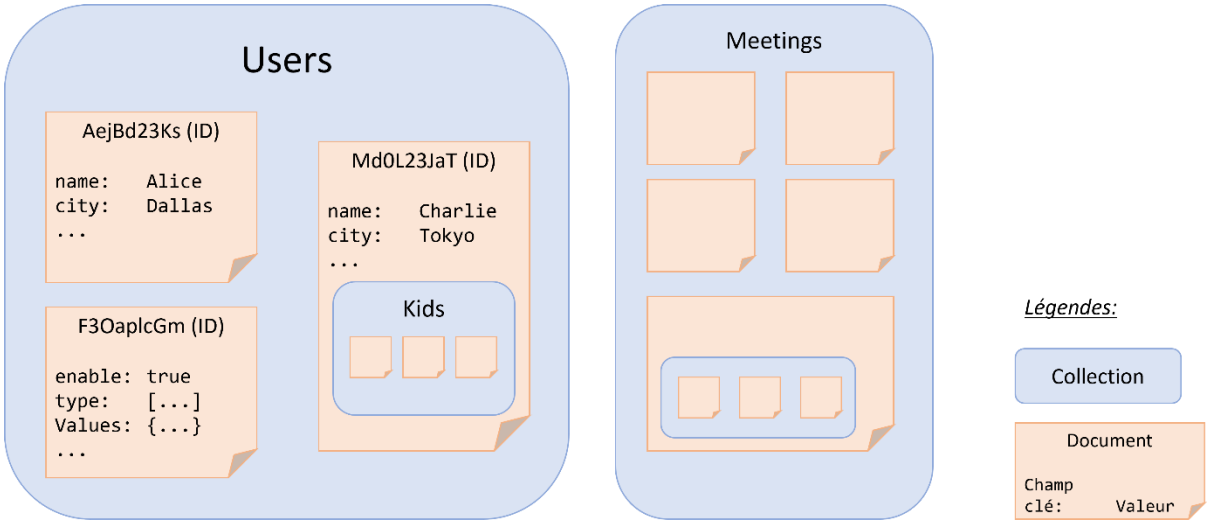

Figure 18 - organisation des données au sein de Firebase Firestore

À la racine, l'on trouve différentes collections qui se distinguent par leur nom. Chaque collection possède un ou plusieurs documents. Chaque document d'une collection possède un identifiant qui doit être unique au sein de la collection parente. Un document peut contenir différents champs et/ou d'autres collections.

En comparaison avec un outil de base de données SQL, une collection peux être associée à une table dont les colonnes sont les clés. Chaque ligne de la table correspond à un document, dont l'identifiant est la clé primaire. Il existe cependant des différences :

Dans SQL, chaque ligne d'une table doit contenir les mêmes champs que toutes les autres lignes. Or, avec Firestore, cette restriction n'est pas présente. Deux documents au sein d'une même collection peuvent avoir des champs totalement différents les uns des autres. De plus, avec SQL il n'est pas possible de créer directement des « sous-tables » ou tables enfant. Pour contourner le problème, il est nécessaire de créer une autre table et d'introduire une relation entre les deux tables. Dans Firestore, un document peut directement contenir une sous-collection. Il n'est d'ailleurs pas possible d'effectuer des requêtes relationnelles (avec des jointures entre les données) dans Firebase Firestore car il s'agit d'un noSQL.

Une autre différence majeure de Firestore concerne la sécurité et l'accès aux différentes données. Avec un SQL ou un Oracle, les droits sont établis sur les tables/vues/procédures/… en fonction de l'utilisateur authentifié et du type de requête demandé (SELECT, UPDATE, etc.). Avec Firebase, les droits peuvent se faire au niveau des collections mais également au niveau des documents. De même, il est possible de définir différents droits en fonction du type de requête (read, write, update, etc.). En revanche, les droits peuvent être bien plus « personnalisables » qu'avec un gestionnaire de données plus classique :

Premièrement, il est possible d'utiliser l'identifiant d'un document pour définir des droits. Par exemple, il est possible d'établir la règle suivante :

- Écriture autorisée si l'identifiant du document correspond à l'identifiant de l'utilisateur authentifié pour la requête.

D'un point de vue SQL, cela reviendrait à donner les droits UPDATE uniquement sur la ligne concernée par l'utilisateur, ce qui n'est pas possible sans passer par l'utilisation de procédures stockées.

Deuxièmement, de la même manière que l'identifiant du document peut être utilisé, les différents champs qui composent le document peuvent être utilisés, permettant d'établir des règles tels que :

- Lecture autorisée si la valeur du champ « public » du document vaut `true`.

Enfin, il est possible d'effectuer des requêtes intermédiaires sur des champs d'un autre document pour spécifier une règle. Ces requêtes supplémentaires seront effectuées directement par le gestionnaire de données Firebase possédant tous les droits de lecture, permettant la création de règles comme :

- Lecture autorisée si la valeur du champ « visible » dans le document /private/{id} vaut `true`. L'utilisateur n'a pas les accès en lecture au document /private/{id}.

En revanche, il existe aussi certaines limitations. En effet, il n'est pas possible de définir des règles au niveau des champs qui composent un document. Par exemple, la règle suivante n'est pas possible :

- Lecture sur les champs « name » et « city » du document autorisé.

Pour contourner cette absence de règles sur les gestionnaires de données plus classiques tels que SQL ou Oracle, il existe un intermédiaire supplémentaire, un serveur REST, qui s'occupe d'effectuer les requêtes sur le gestionnaire de données, puis de filtrer et de retourner les données en fonction des valeurs présentes dans les tables et des informations d'authentifications fournies. Dans ce projet, il a été décidé qu'aucun serveur REST ne serait implémenté. L'application mobile communique directement avec Firebase afin de profiter des avantages de mise à jour en temps réel des données et de la gestion des notifications.

En revanche, l'absence d'un serveur REST apporte également un risque de sécurité supplémentaire. En effet, si un utilisateur mal intentionné souhaite accéder à des données dont il n'est pas censé avoir accès, ce dernier doit trouver une faille au niveau de l'application Frontend. Or l'utilisateur peut directement analyser les communications entre notre application et le gestionnaire de données (Firebase, MySQL, etc.), ce qui n'est pas possible avec un serveur REST (sauf si l'on a réussi à s'introduire dans le réseau interne du serveur, évidemment).

Par exemple, si l'intégralité des données concernant un utilisateur sont transmises à l'application, y compris les données sensibles, et que ces dernières sont ensuite cachées à l'utilisateur par l'application, il « suffit » que l'utilisateur mal intentionné analyse la réponse obtenue pour récupérer toutes les informations sensibles de l'utilisateur.

Pour toutes ces raisons, il est nécessaire de concevoir une nouvelle structure des données de Firebase qui prend en compte les limitations et les différences de la plateforme. De même, la structure doit répondre à des règles de sécurité évitant qu'un utilisateur puisse accéder à des données confidentielles de l'utilisateur. La figure ci-dessous décrit la nouvelle structure de Firebase réalisée :

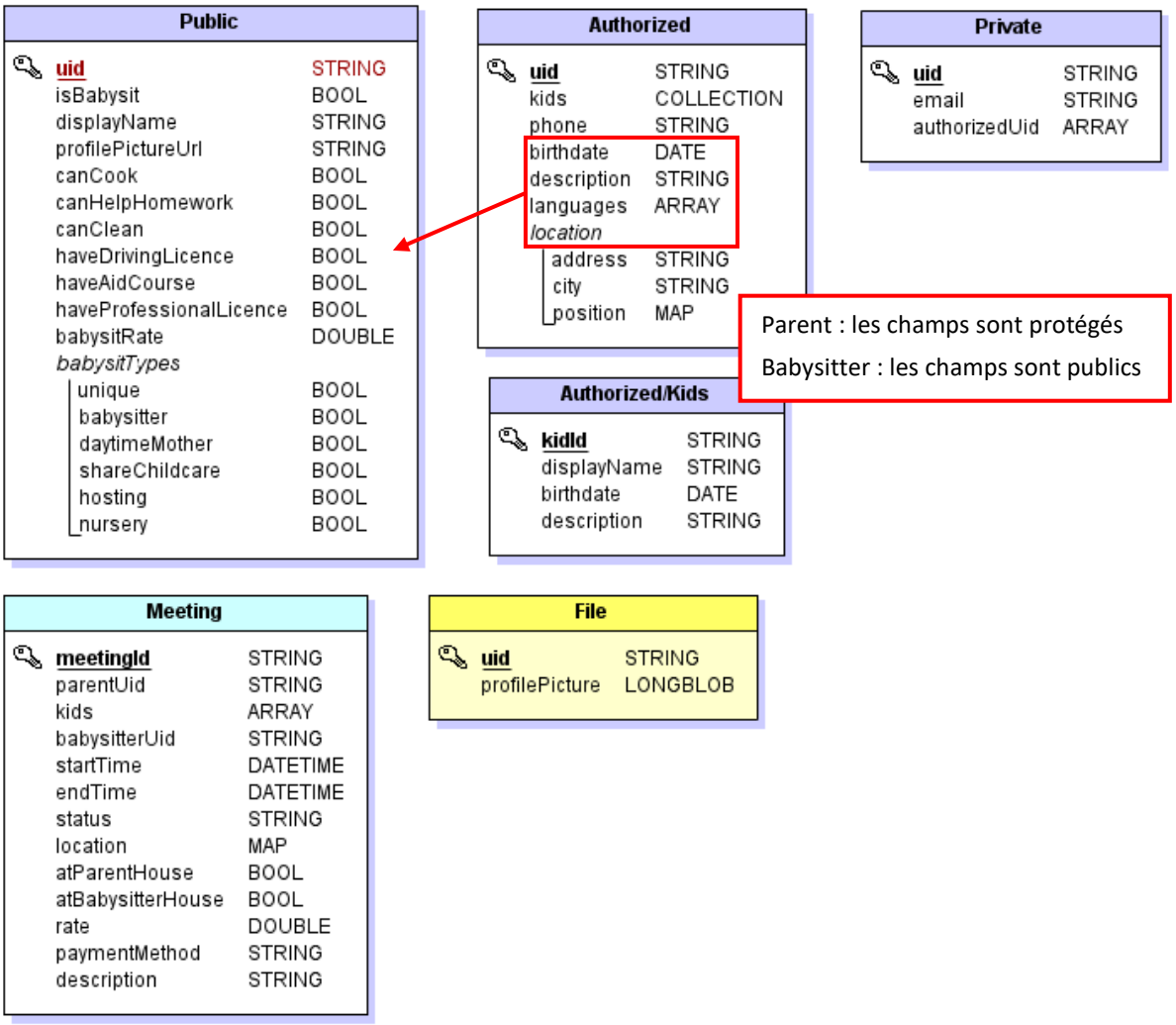

Figure 19 - Structure des données dans Firebase

Sur cette figure, l'on observe notamment que les informations concernant l'utilisateur ont dû être séparées en quatre collections différentes. De cette manière, des règles spécifiques peuvent être assignées à chaque collection indépendamment des autres, permettant ainsi de rendre des informations privées complètement invisibles aux yeux des autres utilisateurs. La collection « Authorized » permet de stocker des informations disponibles uniquement pour certains utilisateurs spécifiques, par exemple lors de la demande ou de la validation d'un rendez-vous. Les données encadrées en rouge naviguent entre la collection publique et la collection protégée, selon le type de profil utilisateur (parent ou nounou).

Pour garantir une confidentialité des données, les règles suivantes de Firebase ont été définies au niveau des documents :

Document	Lecture	Écriture
/Public/{userId}	Toujours	*1*
/Authorized/{userId}	*1* ou *2*	*1*
/Authorized/{userId}/Kids/*	1 ou 2	*1*
/Private/{userId}	*1*	*1*
/Meeting/{meetingId}	*3*	**Création :** si l'utilisateur est authentifié
		Édition : *3*
		Suppression : *3*
/File/{userId}	Toujours	*1*

Légendes :

1 : Si l'uid de la requête correspond à {userId}

2 : Si l'uid de la requête se trouve dans (/Private/userId).authorizedUid

3 : Si l'uid de la requête correspond à {meetingId}.parentUid ou babysitterUid

Avec cette structure et les règles de sécurité mises en place, il est possible d'assurer une disponibilité des données lorsque cela est nécessaire tout en garantissant une sécurité des données. Les données sensibles de l'utilisateur ne sont disponibles que si ce dernier a autorisé l'utilisateur effectuant la requête d'accéder à ses informations. Les données privées de l'utilisateur ne sont disponibles que pour lui-même. Les informations sur un rendez-vous ne peuvent être obtenues que par les utilisateurs concernés.

Il est également important de noter qu'en fonction du type de profil (parent ou nounou), certaines informations transitent entre la partie publique et la partie protégée de l'utilisateur. En effet, la disponibilité de cette information dépend du statut de l'utilisateur : Il doit être possible d'accéder à la localité d'une nounou afin de pouvoir effectuer une recherche avec ses coordonnées géographiques, mais pour un parent cette information ne doit être disponible que pour les nounous ayant un rendez-vous avec le parent. Pour éviter d'avoir une structure qui diffère légèrement selon le type d'utilisateur, il aurait été possible de mettre ces informations dans une nouvelle collection, de la même manière qu'une table spécifique aurait été réalisée avec SQL. Cependant, cette modification implique un accès supplémentaire à une ressource de Firebase, c'est-à-dire une opération en plus à prendre en compte dans la facturation, et ce à chaque fois que l'on désire récupérer un profil. De plus, Firebase autorise le fait que des documents au sein d'une même collection possèdent une structure différente, ce qui n'est pas le cas avec SQL ou Oracle.

Cette nouvelle structure comporte cependant un désavantage : en divisant les données en plusieurs collections, il est maintenant nécessaire de réaliser plusieurs opérations pour lire ou écrire les données concernant l'utilisateur. Or, la facturation de Firebase ce fait notamment en fonction du nombre d'opérations réalisées. En augmentant le nombre d'opérations requis pour récupérer les données, on augmente également les coûts d'utilisation de Firebase.

De plus, ces opérations successives seront directement réalisées par l'application HappyBaby. Mais il n'est pas impossible qu'une erreur de communication survienne lors de la réception d'une partie des informations, créant ainsi une incohérence entre les données locales et les données centrales.

Pour contourner ce problème, la mise en place d'un serveur Backend REST API serait possible, mais il serait aussi possible d'utiliser les Cloud Functions de Firestore [33]. Grâce aux fonctions, il est possible de « transférer » la charge de travail consistant à manipuler les informations des différentes collections de l'application Flutter vers Firestore, sans avoir à déployer le moindre serveur. Avec les fonctions, il y a donc moins de risques que des données inconsistantes apparaissent entre le Frontend et le Backend, et la charge de travail côté client se retrouve allégée. Pour le moment, aucune transformation conséquente des données n'est nécessaire, mais il n'est pas impossible que cela survienne durant le projet, réduisant les performances de l'application et donc l'expérience utilisateur en l'absence de fonctions.

Cependant, l'étude et la compréhension des Cloud Functions de Firestore n'est pas une tâche facile pouvant être réalisée rapidement. Il s'agit d'un outil très complexe et polyvalent et de nombreux tutoriels/exemples sont disponibles pour aider l'utilisateur à bien comprendre le fonctionnement des fonctions. Si l'outil devait donc être intégré au projet, il serait nécessaire de planifier une phase conséquente pour l'analyse, l'étude et la conception des fonctions, en plus du temps nécessaire à leur implémentation. Or, la planification actuelle est déjà bien chargée et de nombreux autres objectifs secondaires sont déjà présents. Pour ces raisons, il a été décidé que les Cloud Functions ne seraient pas étudiés dans ce projet de Bachelor.

Pendant la phase d'amélioration des rendez-vous, plus précisément lors de l'implémentation des notifications en utilisant le « protocole » **F**irebase **C**loud **M**essaging (abrégé FCM), l'utilisation des Cloud Functions s'est avérée être nécessaire car il est bien plus compliqué d'envoyer une notification FCM à partir de Flutter qu'à partir des Cloud Functions. Vous trouverez plus de détails sur ce sujet dans la section « 3.4 – Rendez-vous ».

2.2.3 Modèle économique

Suite à l'analyse des frais d'utilisation générés par l'application, nous savons qu'il est nécessaire de concevoir un plan permettant de générer en moyenne 0.05$ par utilisateur et par mois, peu importe sa fonction au sein de l'application.

Initialement, il était prévu que plusieurs approches différentes et plusieurs modèles soient présentés dans cette section. Malheureusement, le fait que cet objectif soit étudié sur la fin du projet et que le temps restant n'est pas aussi important qu'imaginé selon la planification initiale, seule une approche sera étudiée dans ce document : Utiliser l'achat intégré pour le paiement des rendez-vous et toucher une commission.

Parmi les autres approches qui aurait pu être étudiées, l'on retrouve l'utilisation de pages publicitaires au sein de l'application, le fait de rendre le téléchargement de l'application payante ou encore proposer un système d'abonnement mensuel payant donnant des avantages aux membres abonnés.

Premièrement, il est nécessaire d'étudier comment intégrer un service de paiement dans l'application HappyBaby. Après quelques recherches, il apparaît qu'un package Flutter développé par les équipes de Google est disponible, permettant ainsi rapidement l'intégration de Google Pay et Apple Pay [34]. En revanche, ce n'est pas la seule chose qu'il est nécessaire d'effectuer.

En effet, ces services se basent sur une passerelle (*gateway* en anglais) pour effectuer réellement la transaction. C'est en partie pour cette raison que ni Google Pay, ni Apple Pay ne prennent de commissions additionnelles sur les paiements : ce ne sont pas vraiment ces services qui effectuent le travail.

Mais alors quel est l'utilité d'Apple Pay et de Google Pay ? Tout simplement de proposer une méthode de paiement unique. En effet, peu importe si l'utilisateur passe par sa carte de crédit, son compte bancaire, un portefeuille virtuel (E-wallet) ou une carte cadeau achetée en physique, la transaction est généralisée et il n'y a qu'un seul type de transaction à gérer. En plus, la popularité internationale des entreprises Apple et Google fait que de nombreuses banques ont intégrés la possibilité de lier son compte ou sa carte de crédit aux applications en question. Par exemple, en Suisse, les deux services sont compatibles avec [35][36][37]:

- UBS
- Raiffeisen (cartes de crédit uniquement pour Google Pay, via Viseca)
- BCV, BCF et presque toutes les banques cantonales (carte de crédit uniquement)
- Etc.

En plus des banques « traditionnelles », ces outils sont également compatibles avec des banques ou des services proposants des cartes de crédit éphémères tels que Revolut ou encore N26.

L'utilisation de ces services apporte donc un avantage conséquent : celui de proposer au client un large choix d'options de paiement différents sans pour autant devoir implémenter chaque option séparément. Pour autant, comme cité en amont, une passerelle doit être liée à Apple Pay ou Google Pay pour fonctionner correctement. En effet, il n'est pas possible de lier son compte bancaire en tant que compte de réception directement dans ces services. Les outils ne font que communiquer les informations sur la transaction à effectuer auprès des passerelles qui s'occupent d'identifier la nature du paiement et de demander l'autorisation et l'authentification de l'expéditeur auprès de sa banque pour le paiement. La figure ci-dessous résume les étapes de fonctionnement des services proposés :

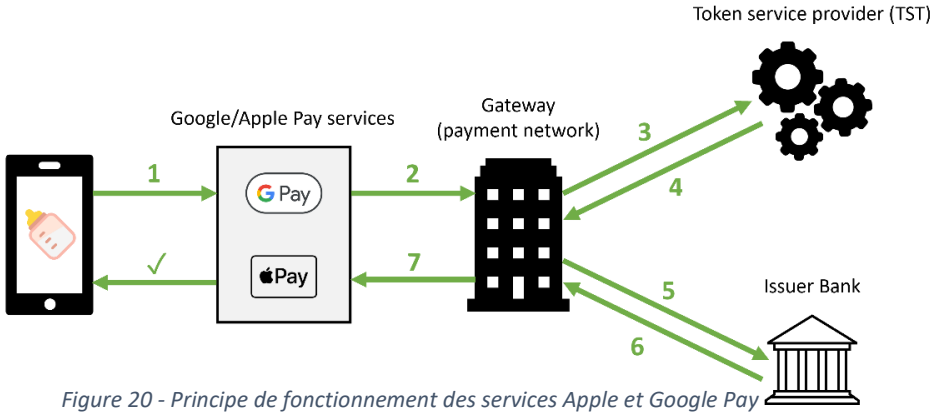

Figure 20 - Principe de fonctionnement des services Apple et Google Pay

Mais alors quelles sont les passerelles disponibles en Suisse permettant l'intégration de ces services ? Si l'on regarde la documentation, seuls peu de services sont disponibles à la fois pour Apple Pay et Google Pay en Suisse [38][39]. Dans ce projet, l'une des plateformes les plus populaires sur le marché, Stripe, a été étudiée.

Stripe se définit comme « *Une infrastructure de paiement pour le commerce en ligne* » [40] et propose de multiples outils pour assister les développeurs dans l'implémentation d'une plateforme de paiement sécurisé au sein de leur application. Concernant Flutter, un package est également disponible permettant une intégration rapide de Stripe en proposant des modèles de vues que l'on peut directement utiliser pour les paiements [41]. Si en revanche on souhaite utiliser le package Pay précédemment cité, Stripe propose un support complet du package et une intégration en seulement quelques lignes supplémentaires de code.

Stripe est donc un outil permettant à la fois de s'intégrer rapidement au sein de notre application en proposant des vues et des modèles déjà implémentés, mais fait également office de passerelle pour l'utilisation d'Apple Pay et Google Pay. De plus, Sprite étant un outil beaucoup utilisé et démocratisé, de nombreux autres moyens de paiement sont disponibles à travers cet outil. Par exemple, l'utilisateur n'a pas besoin de passer par les services d'Apple ou de Google si jamais ce dernier décide de payer avec sa carte de crédit.

Bien évidemment, tous ces avantages ont un coût. A chaque paiement réussi, une commission de 2,9% de la transaction + 0,30CHF sera perçue par Stripe, que ce soit pour une transaction effectuée via Apple Pay, Google Pay ou en utilisant directement une carte de crédit [42]. Le modèle économique doit donc prendre en compte ces frais additionnels pour proposer une solution économiquement rentable. Des frais additionnels peuvent être ajoutés si des changements de devises sont effectués lors du paiement. Néanmoins, comme l'application n'a pas pour vocation d'être disponible ailleurs qu'en Suisse (du moins pour ce projet), aucun frais additionnel ne sera considéré.

En revanche, une dernière interrogation se pose : une fois l'argent disponible sur notre compte, comment faire pour le redistribuer aux nounous ? Heureusement, il se trouve que Stripe propose également des solutions particulièrement adaptées à cette situation. En effet, Stripe Connect [43] est un service permettant entre-autre à une entreprise d'envoyer rapidement des virements à différents prestataires et/ou marchands. Il correspond donc parfaitement à notre volonté de pouvoir gérer et envoyer simplement de l'argent aux différentes nounous présentes dans l'application, tout en touchant une commission au passage.

Bien sûr, l'utilisation de Stripe Connect n'est pas gratuite. Pour chaque virement effectué à une nounou, des frais de 0,25% du virement + 0,55CHF seront perçus. De plus, pour chaque client ayant reçu au moins un virement dans le mois, 2CHF additionnels seront perçu du solde Stripe. Ces frais doivent également être pris en compte. Avec toutes ces informations, nous pouvons à présent concevoir un modèle économique rentable se basant sur l'intégration d'achat intégrés avec une commission perçue pour chaque virement effectué.

Afin d'étudier les bénéfices engendrés par l'application, il est nécessaire de connaître le prix moyen d'un rendez-vous effectué. Pour cette conception, l'hypothèses qu'un rendez-vous est payé en moyenne 30CHF, indépendamment de sa durée. Le schéma ci-dessous explique le fonctionnement du modèle économique utilisé :

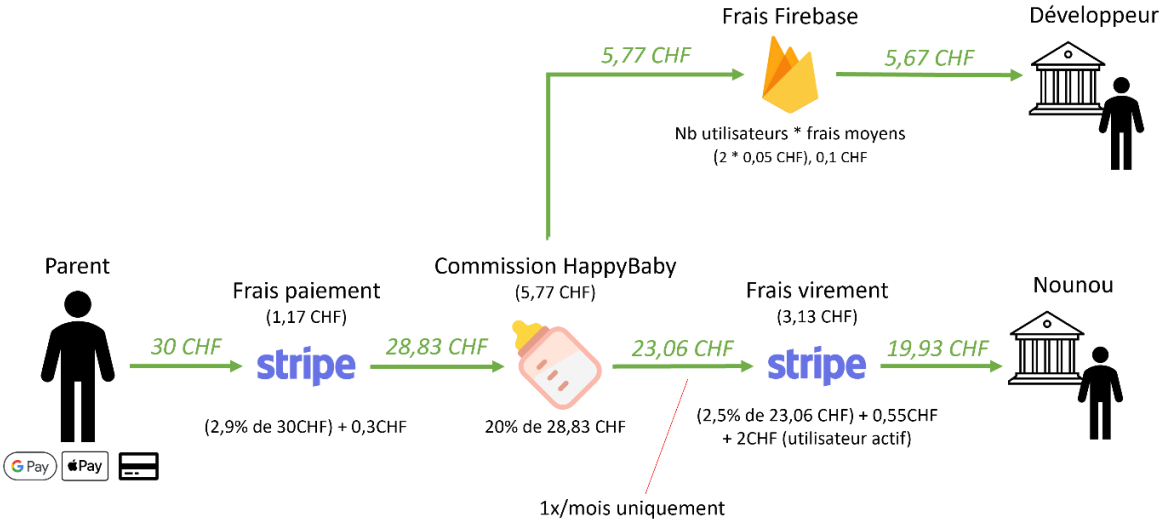

Figure 21 - Conception du modèle économique avec une commission perçue de 20%

Sur cette figure, nous pouvons observer que l'argent gagné grâce à la commission touchée ne sert qu'à rembourser uniquement les frais engendrés par Firebase. En effet, les frais de commissions perçus par Stripe ne doivent pas être remboursés à partir du solde Stripe (sinon on paierait à double). En revanche, sur les 30.- payés par le parent, seulement ~20.- sont touchés par la nounou, soit environ 66% du paiement initial. Sur cette même figure, nous pouvons également observer que la commission touchée par HappyBaby est de 20%. Pour autant, est-ce qu'une autre commission, plus faible, pourrait être envisagée ?

La réponse à cette question dépend de différents facteurs, notamment le nombre d'utilisateurs actifs sur l'application, le nombre de rendez-vous payés ou encore le prix du rendez-vous (estimé à 30 CHF). Les calculs effectués montrent que mêmes dans des situations extrêmes, c'est-à-dire un grand nombre d'utilisateurs actifs mais pour autant une faible quantité de rendez-vous payés, une commission touchée de moins de 10% suffit à couvrir les frais d'utilisation de Firebase, comme le montre le graphique ci-dessous :

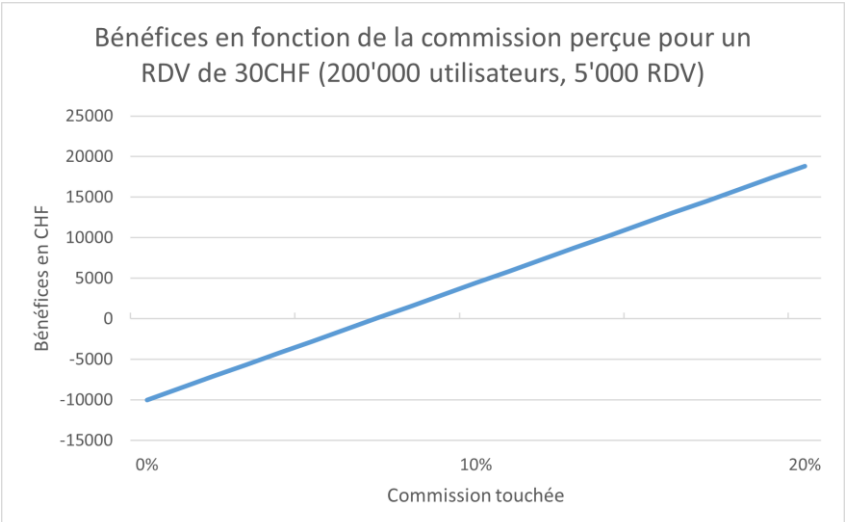

Figure 22 - Bénéfices obtenus en fonction de la commission perçue pour un rendez-vous de 30CHF

2.2.4 Tests utilisateurs

Pour réaliser des tests utilisateurs significatifs et pouvoir tirer des conclusions sur l'utilisabilité de notre application, il est nécessaire de concevoir un modèle de réalisation des tests. Nous savons que les tests visent à représenter au maximum une utilisation réelle de l'application tout en mesurant la satisfaction des utilisateurs et l'utilisabilité de chaque fonctionnalité. Pour ce faire, il a premièrement été nécessaire de concevoir différents scénarios de test permettant de séparer les différentes fonctionnalités présentes dans l'application afin que l'utilisateur teste ces fonctionnalités une à une. De plus, les scénarios garantissent que l'utilisateur découvre toutes les fonctionnalités le concernant, il n'y a pas ou peu de risques qu'un utilisateur passe « à côté » d'une fonctionnalité. Bien évidemment, les fonctionnalités propres aux nounous ne seront pas étudiées par les parents et vice-versa durant la phase de test. Ainsi, l'on peut dire que seules les personnes ayant à la fois le rôle de parent et de nounou utilisent l'intégralités des fonctionnalités offertes par l'application.

2.2.4.1 Scénarios de test

Afin de concevoir des scénarios de test, il est important de se rappeler que les utilisateurs acceptent de participer gratuitement aux tests utilisateurs et n'ont donc pas beaucoup de temps à consacrer à la participation des tests. Ainsi, les objectifs à atteindre pour les utilisateurs ont été séparés en différents scénarios, avec dans l'idée que l'utilisateur complète un scénario par jour, avec le questionnaire associé. Durant la préparation à la semaine de test utilisateur, les objectifs suivants ont été identifiés permettant une utilisation complète de l'application :

Type	*Objectifs*
Commun	- Ouvrir l'application à partir de l'écran d'accueil - Créer un compte au sein de l'application - Remplir ses informations de base - Modifier son mot de passe - Se connecter à l'application - Consulter les informations détaillées d'un rendez-vous - Afficher un profil à partir d'un rendez-vous
Nounou	- Remplir ses informations de nounou - Consulter ses rendez-vous - Accepter/refuser un rendez-vous
Parent	- Remplir ses informations de parent - Chercher une nounou - Consulter le profil de l'utilisateur via la recherche - Demander un rendez-vous - Annuler un rendez-vous

Les objectifs ont donc été séparés dans différents scénarios. Pour les parents et les nounous, il y a quatre scénarios différents que les utilisateurs devront réaliser durant la semaine de test. Les personnes ayant à la fois le rôle de parent et de nounou, eux, devront réaliser cinq scénarios. Vous trouverez en annexe les scénarios distribués aux utilisateurs.

2.2.4.2 *Questionnaire concernant les scénarios*

Afin de mesurer l'utilisabilité de l'application Hapybaby, des questionnaires de satisfaction ont été élaborés. Après chaque scénario, l'utilisateur était invité à remplir un questionnaire, réalisé et hébergé par le service Google Forms [44]. En plus des questions ASQ et SEQ présentées dans la phase d'analyse, d'autres questions complémentaires sont posées aux utilisateurs : pour chaque objectif, la personne indique si elle a utilisé la procédure afin de réaliser l'objectif et pourquoi. Ainsi, si beaucoup de personnes ont utilisé la procédure d'utilisation car ils ne savaient pas comment résoudre l'objectif, on peut considérer que les fonctionnalités couvertes par l'objectif ne sont pas ergonomiques et instinctives pour un utilisateur moyen.

De plus, l'utilisateur indique également son âge et le système d'exploitation qu'il utilise. Ainsi, il est possible de séparer les résultats des questionnaires selon l'un des deux facteurs et de comparer les résultats, afin de détecter s'il y a des différences majeures dans les réponses en fonction de l'âge ou du système d'exploitation utilisé. La structure suivante pour les questionnaires concernant la réalisation d'un scénario par l'utilisateur a été réalisée :

Pour chaque objectif couvert par le scénario :

Question	Format de la réponse
Quelle difficulté accordez-vous à la réalisation de cet objectif ? *	Échelle de 1 à 7 (1 = très difficile,7 = très facile)
Comment jugez-vous l'utilité des fonctionnalités couvertes par cet objectif ? *	Échelle de 1 à 7 (1 = totalement inutiles, 7 = indispensables)
Comment jugez-vous l'intégration des fonctionnalités couvertes par cet objectif ? *	Échelle de 1 à 7 (1 = très mal intégrées, 7 = parfaitement intégrées)
Quelles améliorations pourraient être apportées sur les fonctionnalités présentées ?	Texte libre
Avez-vous des remarques portant sur la réalisation de cet objectif ?	Texte libre
Avez-vous des remarques portant sur les fonctionnalités couvertes par cet objectif ?	Texte libre
Avez-vous suivi la procédure afin de réaliser cet objectif ? *	Oui ou non
Uniquement si l'utilisateur répond 'Oui' à la question précédente : Pour quelles raisons avez-vous regardé la procédure ? *	« J'étais bloqué sur une ou plusieurs étapes nécessaires à la réalisation de cet objectif », « J'ai préféré suivre la procédure pour réaliser cet objectif » ou « Je pensais qu'il était obligatoire de suivre la procédure » (plusieurs réponses acceptées)

Pour chaque scénario :

Question	Format de la réponse
Avec quel type de téléphone avez-vous réalisé ce scénario ? *	Android ou iOS
Quel est votre âge ? *	« -18 ans », « 18-25 ans », « 26-35 ans », « 36-49 ans », « 50-70 ans » ou « +70 ans »
De manière générale, vous êtes satisfait de la simplicité de résolution des objectifs : *	Échelle de 1 à 7 (1 = pas du tout d'accord, 7 = tout à fait d'accord)
De manière générale, vous êtes satisfait de la durée de résolution des objectifs : *	Échelle de 1 à 7 (1 = pas du tout d'accord, 7 = tout à fait d'accord)
De manière générale, vous êtes satisfait du support utilisateur (procédure, email) disponible pour la résolution des objectifs: *	Échelle de 1 à 7 (1 = pas du tout d'accord, 7 = tout à fait d'accord)
Avez-vous des informations complémentaires sur la réalisation de ce scénario ?	Texte libre

* Les questions marquées d'un astérisque sont obligatoires

Vous trouverez en annexe de ce document les questionnaires utilisés par les utilisateurs.

2.2.4.3 Questionnaire de satisfaction globale

De même que les questionnaires portant sur les scénarios, le questionnaire de satisfaction global utilise le service Google Forms afin de recueillir les résultats des participants. Dans la phase d'analyse des tests utilisateur, les modèles SUS, NPS et AttrakDiff ont été retenus pour mesurer l'utilisabilité et l'expérience utilisateur de l'application HappyBaby. En complément, des questions génériques (comme l'âge ou la profession de la personne) seront également recueillies, afin d'étudier s'il existe des différences dans les résultats. De même, des questions additionnelles seront posées afin d'avoir une idée plus précise de la meilleure, respectivement de la moins bonne, fonctionnalité proposée par l'application.

La structure suivante pour le questionnaire de satisfaction a été réalisée :

Page	**Questions**	**Format de la réponse**
1	Questions SUS *	Échelle de 1 à 5 (1 = Pas du tout, 5 = Tout à fait)
	Question NPS *	Échelle de 1 à 10 (1 = Pas du tout, 5 = Tout à fait)
2	Questions AtrakkDiff court *	Échelle de 1 à 7 (multiples critères)
3	Quel était votre rôle au sein de l'application durant la période de test ?*	« Parent », « Nounou » ou « Parent & Nounou »
	Avez-vous réalisé tous les scénarios de tests envoyés par mail ? *	Oui ou non
	Uniquement si l'utilisateur répond 'non' à la question précédente : Pourquoi n'avez-vous pas réalisé tous les scénarios de test ? *	« Je manquais de temps », « J'ai rencontré un problème qui m'a complètement bloqué(e) », « J'ai oublié de réaliser les scénarios restant » et/ou « Je n'avais pas envie de réaliser les scénarios restants », plusieurs réponses possibles
4	Globalement, l'application a répondu à mes attentes	Échelle de 1 à 5 (1 = Pas du tout, 5 = Tout à fait)
	Globalement, je suis satisfait de mon expérience avec l'application *	Échelle de 1 à 5 (1 = Pas du tout, 5 = Tout à fait)
	S'il y a une chose qui doit être améliorée au niveau de cette application, qu'est-ce que ce serait ? *	Texte libre
	Qu'avez-vous préféré dans l'utilisation de cette application ? *	Texte libre
	Avez-vous des remarques complémentaires sur l'application HappyBaby ?	Texte libre
	Avez-vous des remarques complémentaires sur les scénarios réalisés ou sur les procédure mises à disposition ?	Texte libre
	Avez-vous des remarques complémentaires sur les questionnaires distribués ?	Texte libre
5	Quel est votre statut professionnel ? *	Liste déroulante (étudiant, employé, etc.)
	Dans quelle région habitez-vous ? *	« Fribourg et alentours », « Lausanne et alentours » ou « Autre… (libre) »
	Quel est votre âge ? *	« -18 ans », « 18-25 ans », « 26-35 ans », « 36-49 ans », « 50-70 ans » ou « +70 ans »

* Les questions marquées d'un astérisque sont obligatoires

Vous trouverez en annexe de ce document les questionnaires utilisés par les utilisateurs.

3 Résultats

Cette section a pour objectif de décrire les différentes étapes de l'implémentation du projet. Pour cela, une description de chaque fonctionnalité sera donnée, accompagnée d'éventuels problèmes rencontrés et de leur résolution.

3.1 Refactoring du code

Le premier objectif de ce projet de Bachelor concerne le code du prototype déjà implémenté. En raison d'une mauvaise architecture et du manque de temps lors du projet de semestre précédent, le code implémenté ne suivait pas une logique générique tout au long de l'application. Il en résulte un code difficilement compréhensible et surtout peu générique.

Un premier élément concerne l'utilisation de l'API Google Maps. Bien que les méthodes étaient déjà implémentées dans le prototype, il n'était pas possible d'utiliser l'API car aucun compte de facturation n'était associé au projet. Pour le Bachelor, un compte a été mis à disposition afin de résoudre le problème. Bien évidemment, il a fallu faire quelques modifications sur les méthodes et classes réalisées afin de correspondre avec la nouvelle architecture utilisée.

De plus, maintenant qu'il est possible d'utiliser l'API de Google Maps, il est également possible de récupérer les informations à partir des coordonnées géographiques d'une position (adresse, ville, etc.). Une nouvelle méthode permettant d'effectuer une requête sur l'API pour obtenir les informations à partir des coordonnées de l'utilisateur a été implémentée. Enfin, l'architecture de l'API Google Maps et la gestion des données au sein de l'application ne respectaient pas du tout la nouvelle structure décrite dans la section « 2.2.1 – Architecture Flutter ». Il a donc été nécessaire d'adapter le code pour que ce dernier soit conforme avec l'architecture réalisée.

Ensuite, il était nécessaire de s'attaquer aux flux de données Firebase car aucun d'entre eux n'étaient implémentés. Jusqu'à présent, le profil, la localisation et les rendez-vous de l'utilisateur étaient obtenus une seule fois et il était nécessaire de refaire une requête pour mettre à jour les données (p.ex : il était nécessaire de fermer et de réouvrir la page de rendez-vous pour afficher les éventuels nouveaux rendez-vous). Le code a donc été modifié pour intégrer les flux `Stream<>` comme expliqué dans la section « 2.2.1 – Architecture Flutter ». Pour autant, aucune modification (ou uniquement des petites modifications) n'a dû être effectuée sur les vues car ces dernières ne sont pas concernées par le traitement des données, ce qui est un des buts de la mise en place d'une telle architecture. Les méthodes permettant la modification ou la suppression des données Firebase ayant également déjà été réalisées durant le projet de semestre, aucune, ou de très légères modifications, n'a dû être réalisée. Vous trouverez à la page suivante une capture d'écran de l'organisation du code.

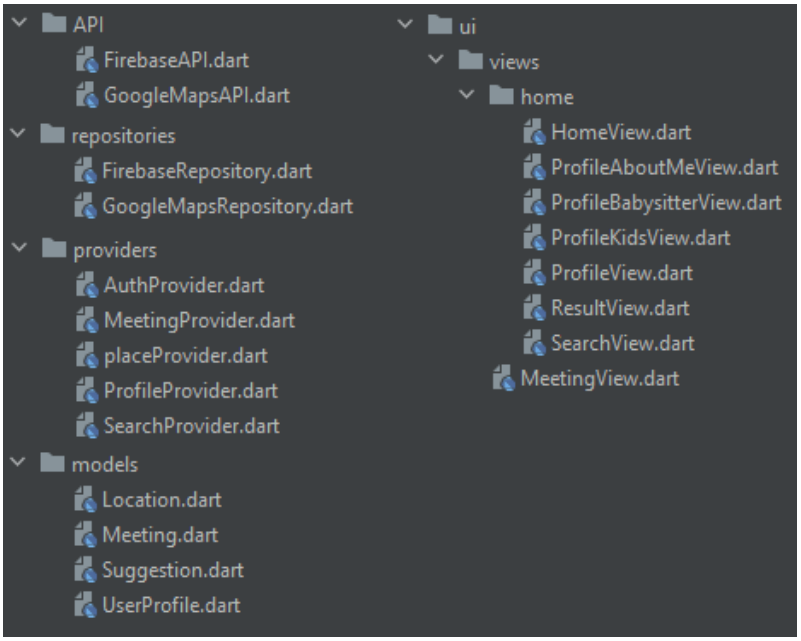

Figure 23 - Organisation du code Flutter

En plus de l'intégration de l'API Google Maps et des Streams, la phase de refactoring a également permis de vérifier la qualité du code. L'analyseur Dart et l'IDE Android Studio ont tous les deux identifié plusieurs mauvaises pratiques au niveau du code. Pour l'identification de code dupliqué, le logiciel PMD [45] a analysé le code réalisé.

Le code a ensuite été rassemblé de sorte que l'on ne retrouve que très peu de code dupliqué, les quelques résultats restants sur la figure ci-dessous étant des bouts de code qu'il est difficilement possible de rassembler sans pour autant complexifier la compréhension du code:

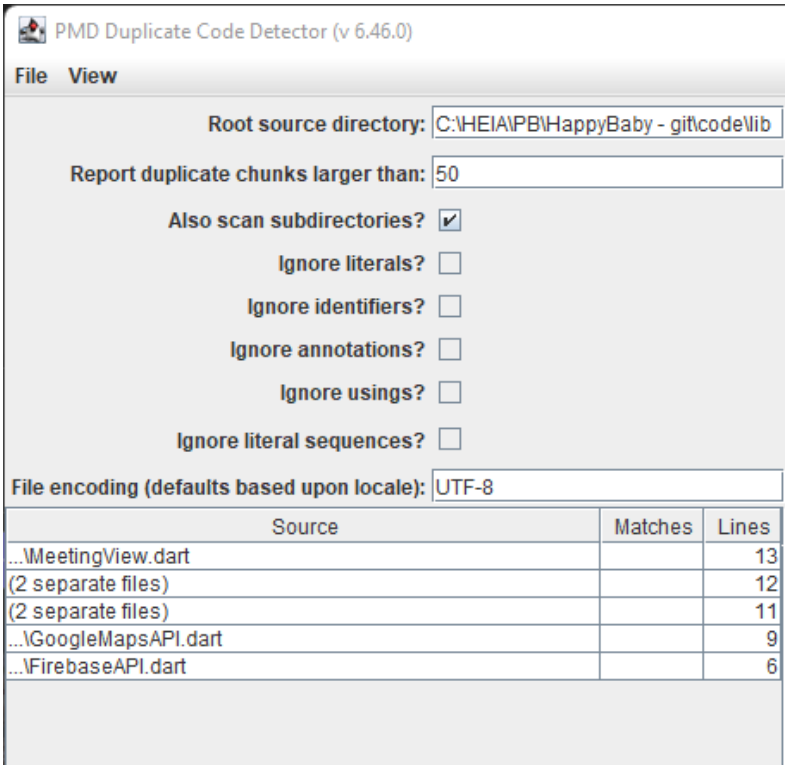

Figure 24 -Capture d'écran des résultats d'analyse du code par PMD [45]

3.1.1 Problèmes rencontrés et résolution

Lors de la phase de refactoring, aucun problème majeur n'est venu perturber le planning du projet. En revanche, des petits problèmes ont été rencontrés durant les différentes phases de refactoring du code.

3.1.1.1 Impossible de mettre à jour la vue lors de la recherche d'adresse

Lors de l'utilisation de l'API Google Maps, une vue spécifique à la recherche d'adresse est utilisée. Cette vue n'hérite pas des classes *StatelessWidget* ou *StatefulWidget* comme les autres vues, mais hérite à la place de la classe *SearchDelegate*. Or, cette classe n'intègre pas la présence de la méthode *setState()* permettant de forcer la mise à jour de la vue.

L'idée originelle était que lorsque l'utilisateur clique sur « Utiliser ma localisation GPS », la vue effectuait une requête et se modifiait afin d'afficher une indication de chargement. Une fois les coordonnées GPS obtenues et éventuellement les informations de ces coordonnées auprès de l'API Google Maps, la vue était alors à nouveau mise à jour.

Malheureusement, comme la vue ne profite pas des méthodes *setState()* et que le provider ne peut pas être observé, il aurait été nécessaire d'utiliser des widgets supplémentaires et de complexifier la classe pour obtenir le résultat espéré. Il a donc été décidé de ne pas faire de changement et de garder l'implémentation actuelle du prototype : si l'utilisateur clique sur « Utiliser ma localisation GPS », la recherche se ferme en indiquant une suggestion particulière indiquant que l'utilisateur souhaite utiliser le GPS. C'est alors à la vue parente qui elle profite de la méthode *setState()* de devoir récupérer les coordonnées GPS ainsi que les éventuelles informations concernant l'emplacement de l'utilisateur.

Par exemple, si l'utilisateur choisi d'utiliser sa position GPS lors de la recherche, c'est la vue gérant la recherche qui récupère la position de l'utilisateur et effectue une requête sur l'API Google Maps. En revanche, si l'utilisateur choisi d'utiliser sa position GPS pour définir son adresse, alors c'est la vue gérant le profil utilisateur qui s'occupe d'effectuer le travail. Les vues gèrent également l'affichage d'erreur en cas de non-récupération des coordonnées GPS.

3.1.1.2 Multiples écoutes sur les flux du Repository

Un autre problème est survenu lors de l'implémentation des Stream. En effet, par défaut il n'est pas possible que plusieurs éléments différents écoutent sur le même flux. Or, cela peut parfois arriver dans notre application, par exemple si plusieurs providers doivent écouter sur le flux d'authentification.

Pour corriger le problème, il est possible d'utiliser un flux de type *broadcast*, mais dans ce cas les anciens évènements qui surviennent avant l'écoute sur le flux ne sont plus transmis, ce qui pose d'autres problèmes.

Après quelques recherches, l'utilisation des contrôleurs de flux de type `BehaviorSubject<>` à la place de `StreamController<>` a permis de résoudre le problème. Comme le problème survenait principalement au niveau de la liste des rendez-vous, il aurait été possible de ne pas utiliser de *StreamProvider* mais d'avoir directement la liste des rendez-vous dans le provider et que cette dernière écoute sur le flux à la place de la vue et mette à jour cette dernière lors

d'une nouvelle information. Si l'on regarde bien la conception de l'architecture, cette solution respecte encore mieux les principes de l'architecture choisie. Durant la phase d'amélioration des rendez-vous, cette amélioration a été effectuée.

3.2 Structure Firebase

Le deuxième objectif de ce projet de Bachelor concerne la nouvelle architecture réalisée pour la gestion des données dans Firebase. Comme expliqué, la première architecture n'était pas compatible avec les objets et les règles présentes dans Firestore. Pour concevoir la nouvelle structure des données, des tests ont été réalisés dans un premier temps pour bien comprendre ce qu'il est possible ou non de faire avec les règles de Firebase. A l'aide de la documentation officielle [46] et des tests réalisés, il a été possible d'identifier que les données de l'utilisateur devaient être séparées en plusieurs collections. De plus, les enfants peuvent être une sous-collection du document concernant le parent, comme expliqué dans la section « 2.2.2 – Structure Firebase ».

Une fois les tests réalisés et le résultat répondant aux attentes du projet, il a été nécessaire d'adapter le code présent afin que ce dernier soit compatible avec la nouvelle structure. Premièrement, les modèles présents ont dû être modifiés pour accueillir les nouvelles valeurs présentes pour l'utilisateur. Comme la réception des informations de l'utilisateur se fait maintenant en plusieurs parties, il est nécessaire d'implémenter une méthode `update()` permettant à l'objet de mettre à jour ses valeurs en fonction des données passées en paramètre.

Si la valeur n'est pas présente, alors ce dernier reste inchangé, comme le montre le schéma ci-dessous :

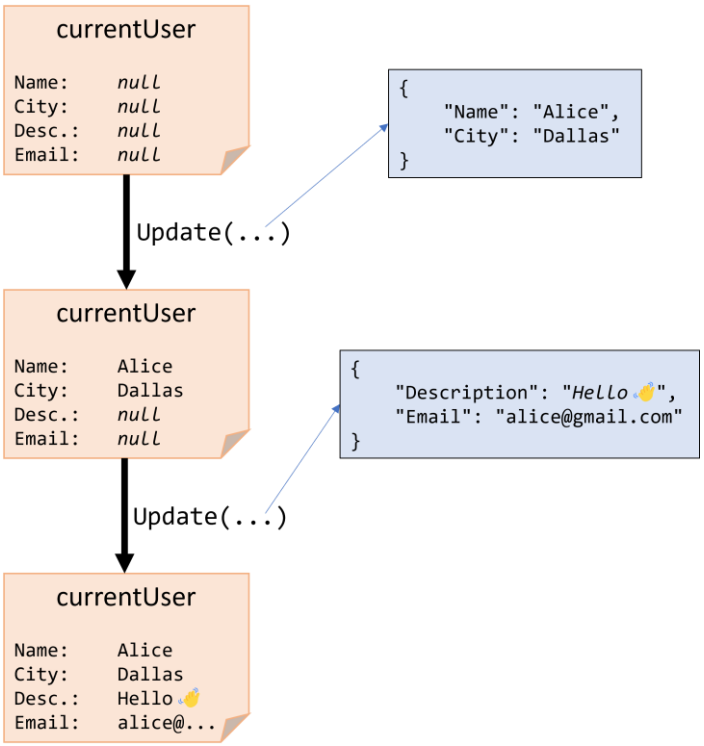

Figure 25 - Mise à jour des données de l'utilisateur en fonction des données reçues

Pour ce faire, la méthode `fromJSON()` a été modifiée afin que la nouvelle méthode utilise ses propres valeurs. De même, la méthode `toJson()` a été séparée en trois méthodes distinctes, une pour chaque partie de l'utilisateur (private, authorized, public).

Une fois le modèle de l'utilisateur modifié, il était à présent nécessaire de modifier ou d'implémenter les autres modèles restants. Concernant les modèles « location » et « suggestion », peu de modifications ont dû être apportées. Pour le modèle des enfants, il est nécessaire d'ajouter l'id de l'enfant car ce dernier est maintenant un document à part entière d'une sous-collection du parent. Le modèle « BabysitType » a dû être réalisé, afin que l'utilisateur puisse spécifier le type de nounou qu'il est. Enfin, le modèle « Meeting » a pu être modifié. Néanmoins, comme les nouvelles informations ne sont pour l'instant pas utilisées car la gestion des rendez-vous n'a pas encore été retravaillée, il est possible que des problèmes apparaissent à l'avenir de suites du changement du modèle. La figure ci-dessous représente la composition des différents modèles au sein de l'utilisateur :

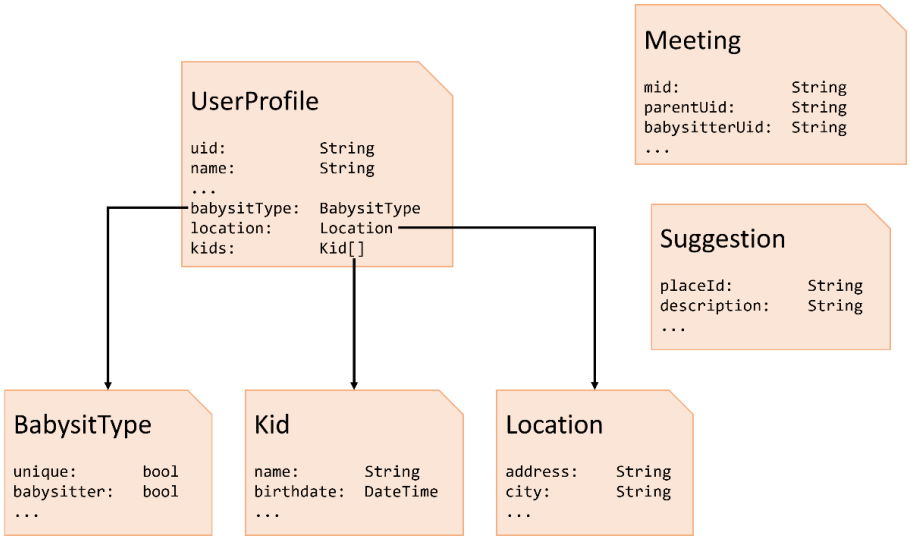

Figure 26 – Composition, interaction des modèles de l'application

Comme nous pouvons le constater, le modèle « UserProfile » inclue d'autres modèles parmi ses valeurs. De ce fait, il est nécessaire que les modèles enfants héritent des mêmes méthodes que le modèle parent. Ainsi, lors de l'appel de la méthode parente, ce dernier peut appeler la méthode enfant pour traiter l'information au niveau de la valeur. Par exemple, dans la méthode `update()` permettant de mettre à jour les informations à partir de JSON, si la clé est « location », alors le modèle parent peux directement faire appel à la méthode `Location.fromJson()` avec les valeurs du JSON concernant la localisation de l'utilisateur pour récupérer un objet de type « Location » conforme aux données reçues.

Une fois les modèles retravaillés, il était dorénavant nécessaire d'adapter les couches « API » et « Repository » de notre architecture. En effet, ce sont ces deux couches qui s'occupent de travailler avec Firebase et avec les modèles de l'application pour fournir des données aux différentes vues. En plus de devoir adapter le code avec les nouvelles informations du modèle, il était également nécessaire de séparer en trois (voir en quatre avec les enfants) les méthodes de réception et d'envoi de données concernant l'utilisateur. Notamment, quatre flux de données en temps réel sont maintenant utilisés pour le profil utilisateur (public, autorisé, privé et enfants) contre uniquement deux flux auparavant (profile et location).

3.2.1 Problèmes rencontrés et résolution

Durant la phase d'implémentation de la nouvelle structure, certains aspects qui n'avaient pas été anticipés sont apparu. Heureusement, le traitement et la résolution de ces problèmes a pu se faire assez rapidement.

3.2.1.1 Id et suppression d'un enfant

Dans l'ancienne structure, les enfants étaient stockés sous forme d'un tableau JSON d'objet dans Firebase. De ce fait, toute la liste des enfants était mise à jour lors d'une opération d'écriture de l'utilisateur. Or, dans la nouvelle structure, il s'agit d'une sous-collection de l'utilisateur. De ce fait, une opération est nécessaire pour chaque enfant que l'on désire traiter.

Un premier problème est survenu pour la gestion des identifiants de l'utilisateur : comment créer les identifiants tout en s'assurant de l'unicité entre tous les téléphones possédant l'application ? Pour cela, il est possible de générer un identifiant unique au niveau de Flutter en utilisant le package Uuid [47] ou alors de laisser Firebase générer un identifiant unique en utilisant la méthode `.add({…})` au lieu de `.doc(id).set({…})` lorsqu'un nouvel enfant est déclaré. Comme le package Uuid génère des identifiants plus longs et que cela ne fait pas vraiment de sens que l'application Frontend génère un identifiant qui se doit d'être unique au niveau Backend, la méthode `add()` a été utilisée.

Un autre problème, toujours lié aux enfants, est également apparu : comment informer le serveur Firebase si un enfant a été supprimé ? Auparavant, toute la liste des enfants était réécrite à chaque fois donc il suffisait de prendre les valeurs présente dans la liste Flutter et de les traduire au format JSON. Or, maintenant que les enfants sont une collection, il est nécessaire de détecter précisément quels enfants ont été supprimés lors de la mise à jour de l'utilisateur. Plusieurs solutions sont possibles pour corriger ce problème :

Premièrement, il est possible de comparer la liste obtenue avant les modifications avec la nouvelle liste. En effet, toutes les modifications non envoyées au serveur sont stockées dans un deuxième utilisateur, temporaire. Néanmoins, pour effectuer une recherche optimisée des différences, il serait probablement nécessaire de transformer la liste d'enfant en objet `Map<id,Kid?>` et de générer l'identifiant au niveau Flutter.

Deuxièmement, il est possible de modifier le modèle de l'utilisateur de sorte que ce dernier dispose des informations des enfants qui doivent être supprimés. Par exemple, une autre liste d'enfant nommée `deletedKids[]` pourrait être ajoutée. De cette manière, si des enfants sont présents dans cette liste lors de la mise à jour de l'utilisateur, on sait que ces enfants doivent être supprimés.

Enfin, une dernière solution pour résoudre le problème serait de modifier le modèle de l'enfant pour y ajouter une nouvelle valeur `isDeleted`. De cette manière, il est également possible de détecter et de supprimer de Firebase les enfants concernés. Il ne faut cependant pas oublier de modifier les vues des enfants, de sorte qu'un enfant qui doit être supprimé n'apparaisse plus dans la vue. C'est la solution qui a été choisie pour résoudre le problème.

3.2.1.2 Mise à jour des vues

Avec la modification de certains attributs des modèles présents dans notre application, presque toutes les vues ont dû être adaptées pour correspondre aux nouveaux modèles. Heureusement, dans la plupart des cas, il suffisait de changer le nom de l'attribut pour corriger le problème.

En revanche, pour les vues utilisant les informations de localisation de l'utilisateur, d'autres modifications ont été apportées : Auparavant, il existait deux types de localisation : celle du parent (son adresse) et celle en tant que nounou (utilisé pour la recherche). Mais après réflexion, il n'y avait pas vraiment de logique de séparer ces deux informations. Dans la nouvelle structure, une seule localisation est utilisée, incluant l'adresse et la ville de l'utilisateur.

Les vues utilisant la localisation ont alors été modifiées afin de toujours utiliser la nouvelle localisation, que ce soit pour définir/afficher l'adresse de l'utilisateur ou son emplacement en tant que nounou.

3.2.1.3 Optimisation des requêtes Firebase

Un autre problème qui a pu être identifiant durant l'implémentation de la nouvelle structure de Firebase concerne le nombre d'opérations effectuées. Comme expliqué dans la section « 2.2.2 – Structure Firebase », cette nouvelle architecture apporte indéniablement plus d'opérations sur Firebase et donc plus de potentiels coûts d'utilisation. En revanche, pour limiter les opérations, il est possible d'analyser au niveau Frontend si effectivement des modifications doivent être apportées avant d'effectuer la requête.

En effet, imaginons que l'utilisateur modifie uniquement son nom d'affichage. Comme cette information se trouve dans la partie publique de l'utilisateur, une seule opération sur le document de la collection *public* de l'utilisateur doit être effectuée. Il n'est en effet pas nécessaire de mettre à jour les documents concernant l'utilisateur situés dans les collections *authorized*, *kids* et *private*.

Néanmoins, ce n'est pas le comportement qui est implémenté actuellement. Si la moindre information concernant l'utilisateur se retrouve modifiée, alors tous les documents de toutes les collections concernant l'utilisateur sont mis à jour avec les nouvelles données, même s'il n'y a pas de changement à réaliser. De ce fait, des opérations « inutiles » sont exécutées sur Firebase.

Actuellement, ce problème ne pause aucune contrainte en pratique car même si certaines opérations sont exécutées inutilement, le nombre total d'opérations journalières effectuées reste bien en dessous des quotas d'utilisation à partir duquel l'utilisation de Firebase devient payante. En revanche, il n'est pas impossible que cet aspect pose un problème lors de la phase de tests utilisateurs.

Pour ces raisons, il a été décidé que le problème serait analysé et éventuellement résolu durant la phase d'analyse et de conception des tests utilisateurs.

3.3 Profil utilisateur

Une fois l'architecture aussi bien au niveau Frontend que Backend retravaillée, il est maintenant possible d'améliorer et d'implémenter les fonctionnalités manquantes dans notre application. Le troisième objectif de ce projet de Bachelor consiste à améliorer le profil utilisateur en intégrant toutes les données le concernant. Jusqu'à présent, certaines informations ne pouvaient pas être renseignées par l'utilisateur et n'étaient pas visible lors de la consultation du profil. Ces informations supplémentaires peuvent également servir de filtre pour la recherche d'utilisateurs.

Premièrement, pour améliorer la généricité entre les différents écrans présents concernant le profil utilisateur, un modèle de « ligne » (au sens de *Row* Flutter) a été réalisé. Ce modèle est ensuite réutilisé pour la majorité des champs concernant l'utilisateur, que ce soit pour afficher ou pour modifier l'information en question. Cela permet également d'éviter grandement la duplication de code.

Ensuite, au niveau des informations de base de l'utilisateur, il n'était pas possible de renseigner son numéro de téléphone ou encore de modifier ses identifiants de connexion. Pour éviter de surcharger l'écran avec trop de champs à disposition, un écran supplémentaire pour gérer les identifiants de connexion a été réalisé, comme le montre les captures d'écran ci-dessous (la traduction n'a pas encore été réalisée) :

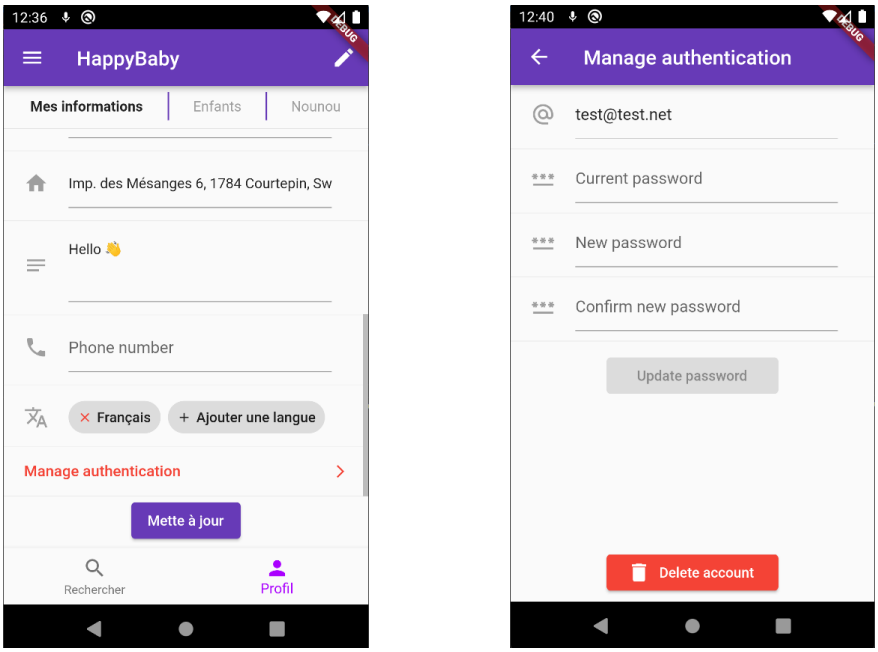

Figure 27 - Captures d'écran des écrans permettant de gérer les identifiants de connexion

Nous pouvons observer qu'il est également possible de supprimer son compte par le biais de cet écran. Lorsque le compte est supprimé, les informations concernant l'utilisateur sont déplacées dans une collection spéciale, en écriture seule, pour garder un historique. Le compte d'authentification de Firebase, en revanche, est complètement et définitivement supprimé.

Concernant l'écran de gestion des enfants, ce dernier était déjà intégralement réalisé et satisfaisant. Seule quelques légères modifications visuelles ont été réalisées.

Enfin, le dernier écran concerne les informations des nounous. Il s'agit de l'écran où le plus de modifications doivent être apportées, car actuellement aucune information qui ne concerne que les nounous ne peuvent être renseignée et/ou visible. De la même manière que pour les identifiants de l'utilisateurs, il a été décidé que des écrans supplémentaires devront être implémentés afin de ne pas surcharger l'écran principal d'informations et de rendre l'application moins ergonomique. Plus précisément, deux écrans supplémentaires seront implémentés: un écran permettant de fournir les informations sur le type de nounou supporté et un deuxième écran permettant de fournir les informations additionnelles concernant la nounou. Le tarif proposé par la nounou, lui, est présent directement sur l'écran principal. Ensuite, toutes ces informations sont présentes dans le profil de la personne. Vous trouvez ci-dessous des captures d'écran des écrans implémentés :

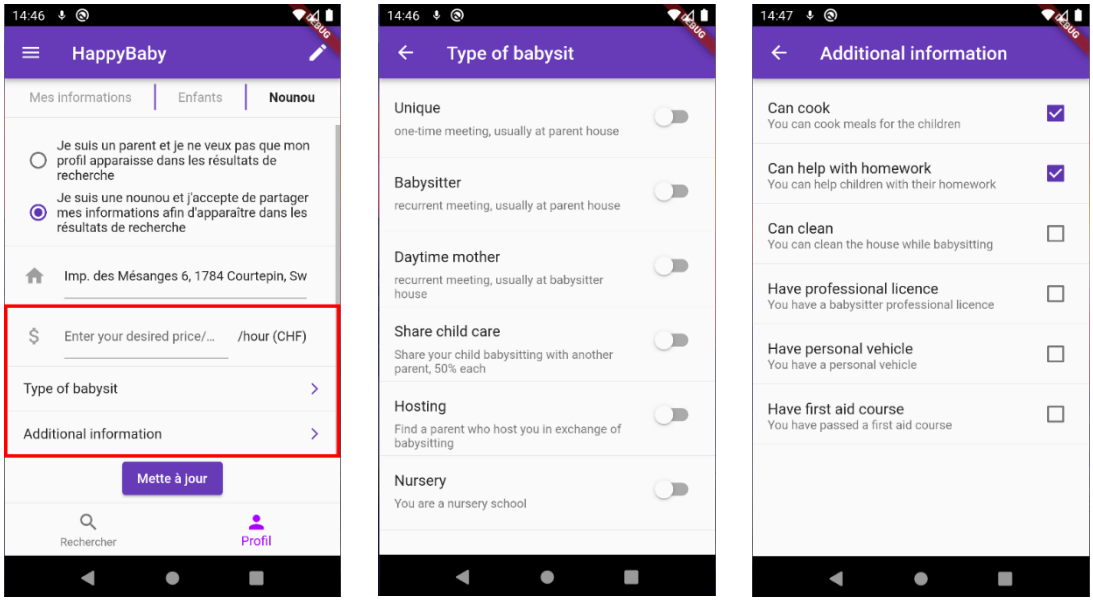

Figure 28 – Captures d'écran des écrans permettant l'édition d'informations concernant les nounous

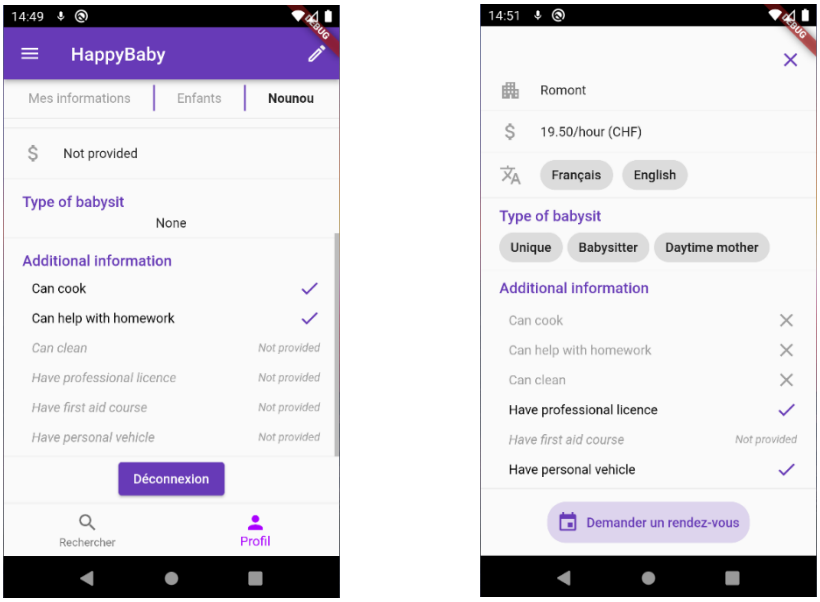

Figure 29 - Captures d'écran de l'affichage des informations d'une nounou sur son propre profil (à gauche) ou lors de la recherche d'utilisateurs (à droite)

A présent, toutes les informations concernant le profil d'un utilisateur peuvent être renseignées et affichées. La présence de ces informations supplémentaires apporte déjà une meilleure expérience utilisateur en permettant aux parents de trouver une nounou correspondant à ses critères de recherches plus simplement. Cependant, cela n'est pas suffisant: Si un parent recherche une nounou de type « Maman de jour » uniquement, il doit lui-même aller consulter chaque profil et regarder si ce dernier est bien du bon type. Idéalement, le parent devrait pouvoir filtrer les résultats de la recherche afin de n'afficher que les nounous qui acceptent le type « Maman de jour ». Pour ce faire, un écran supplémentaire permettant de spécifier des critères supplémentaires lors de la recherche a été implémenté. Idéalement, le tri des données doit se faire directement au niveau de Firebase. Cependant, un problème est survenu lors de la mise en place des indexes pour effectuer des requêtes optimisées et il n'a pas été possible d'effectuer le tri directement sur Firebase. Les données sont donc filtrées par l'application Frontend, à l'exception du tri en fonction de la position qui, lui, se fait bien de manière optimisée directement sur Firebase. Vous trouverez plus de détails concernant ce problème dans la section « 3.3.1.2 – Optimisation de la recherche avancée ». Les captures d'écran ci-dessous représentent les vues implémentées pour la recherche avancée :

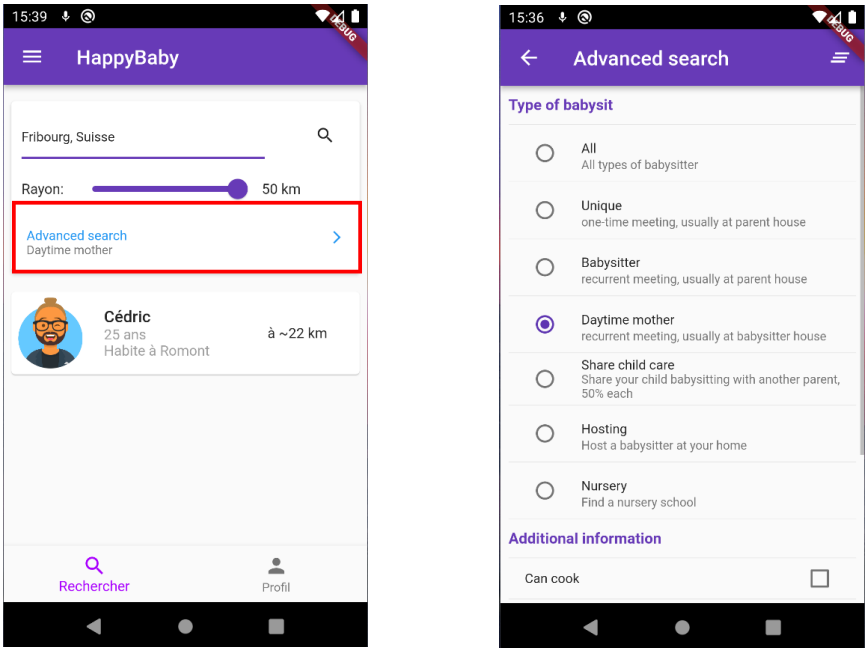

Figure 30 - Captures d'écran des vues permettant la recherche avancée

Ainsi, le parent peut facilement trouver une nounou répondant à ces critères de recherche. En plus des informations présentes, le prix pourrait également être considéré comme un critère de recherche, par exemple en spécifiant une plage tarifaire de recherche ou encore en proposant de trier les résultats en fonction du prix (croissant ou décroissant). Cependant, la mise en place d'un tel système peut prendre du temps et ce n'est pas une fonctionnalité indispensable au bon fonctionnement de l'application. Du fait que le temps initial accordé à l'amélioration du profil utilisateur était déjà dépassé, il a été décidé de ne pas inclure le prix dans les critères de recherche. Si le temps le permet, cette amélioration pourra être apportée durant la phase de réserve de temps ou alors en parallèle de la phase de tests utilisateurs.

3.3.1 Problèmes rencontrés et résolution

Plusieurs problèmes sont apparus durant la phase d'amélioration du profil utilisateur. Certains d'entre eux ont rapidement pu être corrigés, tandis que pour d'autres problèmes il a fallu réfléchir plus longuement à leur résolution. Parfois, la « meilleure » résolution du problème n'était pas facilement réalisable et impliquait l'utilisation de nouveaux outils tels que les Cloud Function de Firebase. Pour éviter de passer trop de temps sur la résolution du problème, une solution moins optimale au problème mais plus rapide à corriger a été réalisée.

3.3.1.1 Modification et suppression des identifiants Firebase

Un premier problème rencontré concerne la gestion de l'utilisateur Firebase. Sur la documentation de Firebase [48], il est précisé que pour modifier ou supprimer un utilisateur de Firebase, il est absolument nécessaire que ce dernier se soit récemment connecté. Or, ce n'est pas forcément le cas dans le cadre de notre application. Pour éviter ce problème, Firebase retourne une erreur spécifique si une réauthentification est nécessaire. Ensuite, il est possible d'utiliser la méthode `reauthenticateWithCredential()` afin de corriger le problème, pour peu que l'on connaisse les identifiants de l'utilisateurs. Ainsi, une première résolution du problème a pu être imaginée :

1. L'utilisateur entre ses nouveaux identifiants de connexion
2. La requête est effectuée auprès de Firebase
3. Si la requête ne retourne pas de message d'erreur, alors il n'y a rien à faire de plus
4. Si la requête indique que l'utilisateur doit être réauthentifié, un écran s'affiche permettant à l'utilisateur d'entrer son mot de passe actuel (l'adresse e-mail de l'utilisateur étant déjà connue)
5. L'utilisateur entre son mot de passe
6. La méthode `reauthenticateWithCredential()` est appelée
7. Si la méthode retourne une erreur (mauvais mot de passe), un message d'erreur s'affiche et l'on revient au point 4
8. Si la réauthentification a pu se faire sans problème, la requête initiale est à nouveau effectuée auprès de Firebase

Cette méthode convient tout à fait pour la modification du mot de passe de l'utilisateur. C'est d'ailleurs la méthode recommandée par Firebase. Cependant, pour la suppression de l'utilisateur ce n'est pas aussi simple. Avant de supprimer un utilisateur, on souhaite déplacer toutes les informations le concernant dans une collection spéciale servant d'historique. Le déplacement se fait donc avant d'appeler la méthode permettant de supprimer les identifiants de Firebase. Or, si la suppression échoue car l'utilisateur doit être réauthentifié, il est nécessaire d'annuler le déplacement dans la mesure où la personne décide finalement de ne pas supprimer son compte pour une quelconque raison. Bien sûr, à chaque fois cela engendre plusieurs opérations sur Firebase et donc des coûts potentiellement plus élevés. Pour corriger ce problème, il serait par exemple possible de déléguer le déplacement des informations du profil auprès d'une Cloud Function : à chaque fois qu'un utilisateur Firebase est supprimé, la fonction s'occupe de déplacer les informations de l'utilisateur en question dans la collection servant d'historique. Comme la fonction utilise ses propres identifiants, le déplacement peux se faire même après la suppression du compte.

Cependant, l'étude et l'utilisation des Cloud Function n'est pas prévue pour ce projet de Bachelor. Une autre solution a donc été imaginée :

1. L'utilisateur fourni en amont son mot de passe actuel, même si ce dernier c'est authentifié récemment
2. La méthode `reauthenticateWithCredential()` est appelée
3. Si une erreur survient, cette dernière est affichée et l'on revient au point 1
4. Si la méthode ne retourne pas d'erreur, alors on peut effectuer les requêtes suivantes (déplacement des données + suppression de l'utilisateur)

Cette méthode a pour défaut de toujours demander le mot de passe de l'utilisateur même si ce dernier s'est authentifié récemment, mais a pour avantage d'éviter l'utilisation des Cloud Function tout en évitant de faire des requêtes inutiles auprès de Firebase comme déplacer des données pour ensuite les remettre au même endroit.

3.3.1.2 Optimisation de la recherche avancée

Comme déjà expliqué brièvement en amont, un problème majeur est survenu durant l'implémentation de la recherche avancée avec les nouvelles informations présentes dans le profil utilisateur. Pour effectuer une recherche optimisée, il serait nécessaire que l'intégralité du tri de la recherche soit effectuée au niveau de Firebase. Dans la pratique, cela veut dire que plusieurs méthodes *where()* seront appelées en cascade avant d'effectuer la requête, comme le montre le pseudo-code ci-dessous :

```
// Get all babysitter of type hosting
var data = await _publicProfileRef
    .where('isBabysit', isEqualTo: true)
    .where('babysitType.hosting', isEqualTo: true)
    .where('...')
    .get(); // Request query to Firebase
```

Pour effectuer la requête de manière optimisée, Firebase crée automatiquement un index pour chaque champ des documents d'une collection, comme expliqué dans la documentation de Firebase [49]. En revanche, si notre requête utilise une comparaison de plage (`<`,`<=`,`>`,`>=`) ou qu'un tri doit être effectué, il est nécessaire de créer un index composite avec les différents champs concernés par la requête. Or, pour effectuer une recherche optimisée en fonction de la position de l'utilisateur, le package *GeoFlutterFire* utilise justement un tri sur le hash de la position. Il est donc absolument nécessaire de créer un index composite. Mais alors, un autre problème survient :

Comme l'utilisateur peux choisir de spécifier ou non un critère pour chaque champ de la recherche, il est nécessaire de créer un index composite pour chaque type de requête différente ! Par exemple, si l'utilisateur effectue une recherche simple sans spécifier de termes avancés, alors l'index composite sera : `isBabysit + location.position.geohash`. En revanche, si l'utilisateur spécifie en plus qu'il recherche uniquement une fille au père qui sait cuisiner et faire le ménage, l'index composite sera : `isBabysit + babysitType.hosting + canCook + canClean + location.position.geohash`.

Il est important de bien comprendre qu'il n'est pas possible d'utiliser le deuxième index composite pour une simple recherche car les champs `babysitType.hosting`, `canCook` et

`canClean` n'apparaissent pas dans la requête. Si l'on souhaite donc effectivement créer un index composite pour chaque type de requête, comme il y a 6 types de nounous et 6 informations additionnelles, il faudrait créer des millions d'indexes composites pour couvrir tous les différents types de requêtes possibles ! Bien sûr, cela n'est pas envisageable.

Une autre solution serait donc de créer un seul index composite incluant tous les champs concernés pour la recherche. Lors de la requête, tous les champs sont utilisés dans une méthode `where()` même si l'utilisateur n'a pas spécifier d'indications pour le champ en question. Dans ce cas, la méthode `where()` doit prendre en compte toutes les valeurs possibles. Heureusement, comme chaque champ est un booléen, il n'y a que trois valeurs possibles : `true`, `false` ou `null`. Il serait donc possible d'utiliser la méthode `where()` avec le paramètre `whereIn : [true, false, null]`.

Cependant, les limitations de Firebase ne permettent d'utiliser qu'une seule clause *whereIn* par requête. Or, il suffit que l'utilisateur ne spécifie pas deux informations lors de sa recherche pour que plus d'une clause *whereIn* soit utilisée. Ce n'est donc également pas une solution envisageable. Après plusieurs recherches sur le sujet, il n'existe pas de solution « miracle » permettant d'effectuer des requêtes optimisées en utilisant dynamiquement certains champs. Il s'agit d'une des différences majeures avec SQL. En effet, cela ne cause aucun problème si seule une partie des indexes réalisés sont utilisés dans une requête de type SQL.

Quels sont donc les solutions possibles permettant tout de même d'effectuer une recherche ? Une première solution serait à nouveau d'utiliser les Cloud Functions en créant un nouveau point d'entrée pour les requêtes. Le filtrage du résultat sera ensuite réalisé par la fonction. Bien sûr, cela veut dire que toutes les données devront être passées à la fonction, mais ce n'est pas vraiment un problème car les fonctions sont dans le même environnement que Firebase Firestore. Une deuxième solution, est de faire le filtrage au niveau du Frontend. Cette solution est loin d'être idéale car certains résultats inintéressants transitent tout de même sur le réseau et sur internet. Néanmoins, c'est la seule solution qu'il est possible de mettre en place sans avoir à étudier les Cloud functions ou à changer complètement l'organisation au niveau du Backend (utiliser les real-time databases de Firebase voir SQL).

De plus, la nouvelle organisation de Firebase ne sépare plus la localité dans une collection propre. Or, si l'on demande à Firebase de trier selon un champ qui n'existe pas dans le document (ou si la valeur vaut *null*), ce dernier est tout de même retourné. C'est par exemple le cas des profils publics des parents, qui ne disposent pas du champs « *location.position.geohash* » utilisé car ce dernier se trouve dans la collection profil autorisés. Pour résoudre ce problème, un index composite a tout de même été établi entre les champs « *isBabysit* » et « *location.position.geohash* ». Ainsi, il est possible de récupérer uniquement les nounous proches et ne plus prendre en compte les parents.

3.4 Rendez-vous

Ce quatrième objectif vise à améliorer les fonctionnalités concernant la demande et l'affichage des rendez-vous. En effet, le prototype implémenté durant le projet de semestre précédent ne permet pas du tout de créer et de gérer les rendez-vous comme souhaité : Seul la date de l'évènement et l'heure de début et de fin peuvent être renseignées. Il n'est pas possible de spécifier les enfants concernés, le type de rendez-vous, le tarif, etc. Pourtant, ce sont des informations qui peuvent être essentielles pour qu'une nounou puisse prendre la décision d'accepter ou de refuser le rendez-vous.

En complément des informations manquantes, d'autres fonctionnalités ne sont également pas présentes : Il n'est pas possible de consulter les détails d'un rendez-vous, les utilisateurs concernés et aucune notification n'est envoyée lors de la création ou du changement du statuts du rendez-vous. Afin d'avoir un système de gestion des rendez-vous répondant aux objectifs de ce projet, chaque fonctionnalité a été améliorée une par une.

Premièrement, comme évoqué dans la section « 3.1.1.2 – Multiples écoutes sur les flux du Repository », l'utilisation des *StreamBuilder* dans les vues ne respecte pas l'architecture Flutter implémenté. En effet, c'est au provider d'écouter sur le flux et de notifier éventuellement les vues lorsqu'un nouvel élément survient. La grande différence étant que le provider est unique au sein de l'application, alors que les vues peuvent être détruites et reconstruites plusieurs fois en fonction de la navigation de l'utilisateur. Une première correction a du pu être apportée en modifiant le provider et la vue afin d'être conforme à l'architecture Flutter du projet.

3.4.1 Demande de rendez-vous

Ensuite, il est nécessaire d'implémenter une nouvelle vue, plus détaillée, pour la demande de rendez-vous. Pour garder une cohérence visuelle dans l'application, les mêmes lignes (au sens de *Row* Flutter) que celles présentes dans les vues concernant le profil utilisateur ont été utilisées. Toutes les informations concernant le rendez-vous peuvent être décrites directement grâce à cette vue. Cela comprend :

- La périodicité ou non du rendez-vous
- Les jours de la semaines concernés (dans le cadre d'un rendez-vous périodique)
- La date de début
- L'heure de début
- L'heure de fin
- La date de fin (dans le cadre d'un rendez-vous périodique)
- Le type de rendez-vous (garde simple, garde partagée, fille au père ou garderie)
- Le lieu du rendez-vous (libre uniquement dans le cadre d'une garde simple)
- Les enfants concernés (paris les enfants du parent)
- Le tarif à l'heure (CHF)
- Le moyen de payement (Cash, TWINT, virement bancaire ou carte bancaire)
- Une description (facultatif)

Pour chaque champ, le parent peut définir la valeur qu'il souhaite. Une fois que tous les champs obligatoires sont replis, ce dernier peut envoyer sa demande de rendez-vous. Les captures d'écran ci-dessous représentent les vues implémentées :

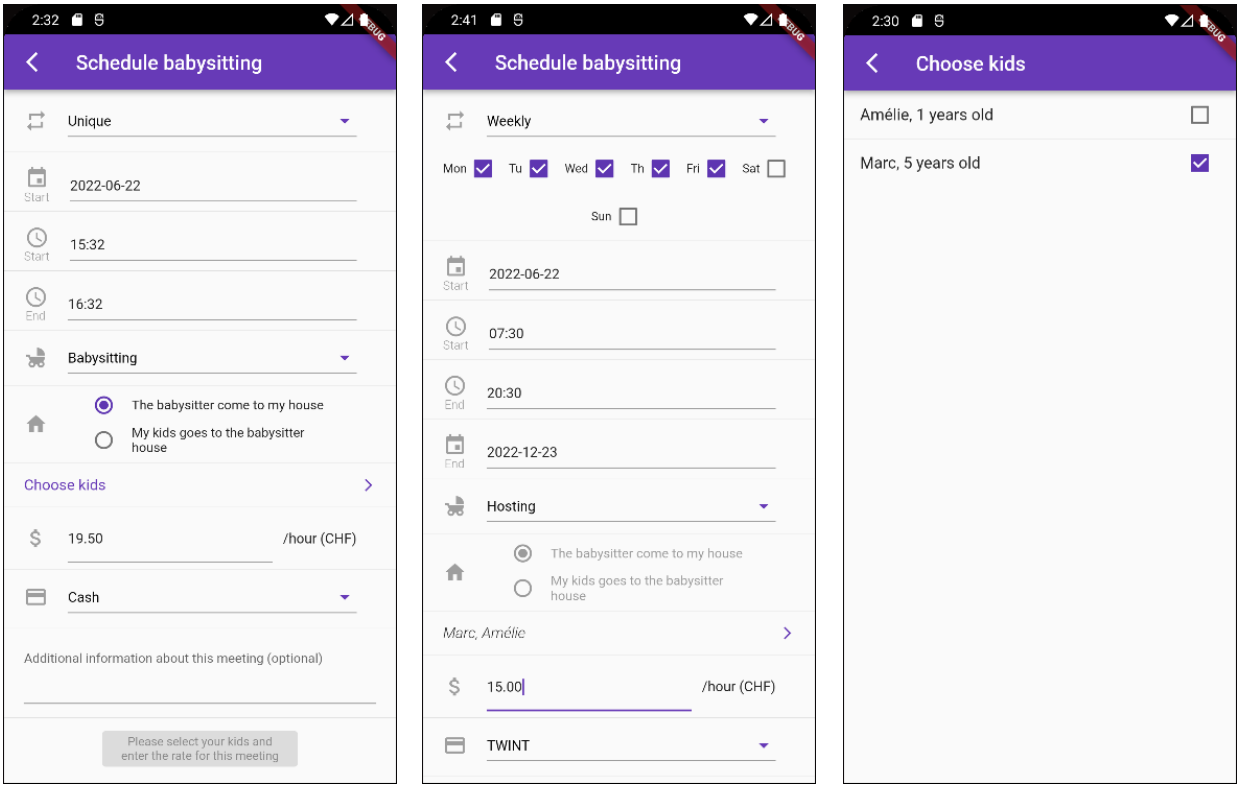

Figure 31 - Captures d'écran des vues pour la prise de rendez-vous unique (à gauche) ou hebdomadaire (au milieu) et la sélection des enfants concernés (à droite)

Les écrans implémentés sont assez denses en raison du nombre d'éléments associés au rendez-vous. Pour alléger l'écran, il a été décidé qu'une autre vue serait implémenté pour le choix des enfants. Une chose importante à prendre en compte est que sur la vue, la date et l'heure du début ne constituent pas deux informations séparées mais une seule est même information au format `DateTime`. C'est également le cas pour la date et l'heure de fin. Quand l'utilisateur change la date de départ, par exemple, il convient alors de modifier l'objet *startDate* (au format `DateTime`) en prenant en compte à la fois la date et l'heure de départ présents dans la vue.

De plus, il est nécessaire de limiter certaines actions de l'utilisateur pour éviter que ce dernier entre une date/heure de fin se situant avant le début du rendez-vous. Pour bien comprendre, imaginons un petit scénario tout à fait reproductible :

Situation initiale :

- L'objet *startDate* vaut : 1er janvier 2022 à 21:00
- Le champ « date de départ » affiche le 1er janvier 2022
- Le champ « heure de départ » affiche 21:00
- L'objet *endDate* vaut : 1er janvier 2022 à 22:00
- Le champ « heure de fin » affiche 22:00
- Le champ « date de fin » n'est pas visible car le rendez-vous est unique

Maintenant, imaginons que l'utilisateur change l'heure de fin pour 00:30. Si l'on se contente de modifier le champ *endDate* en prenant en compte les informations de la vue, sa valeur sera alors : 1er janvier 2022 à 00:30, soit avant la date de départ !

Pour corriger ce problème, il est nécessaire de comparer les valeurs *startDate* et *endDate*. Si la date de fin se trouve être avant la date de début, alors il faut rajouter 1 jour à la date de fin. Dans le scénario, le champ *endDate* vaudra alors : 2 janvier 2022 à 00:30.

3.4.2 Affichage d'un rendez-vous

Une fois les vues permettant la création et la demande de rendez-vous implémentées, il est à présent nécessaire de réaliser les vues permettant d'afficher les informations du rendez-vous. Tout d'abord, pour que la nounou puisse consulter les informations sensibles du parent, tels que ses enfants et éventuellement son domicile, il est nécessaire que l'uid de la nounou se situe dans la liste *authorizedUid* du parent dans Firebase. Pour cela, avant d'envoyer le nouveau rendez-vous au serveur, l'application vérifie si la nounou se trouve bien dans la liste des utilisateurs autorisés. Si ce n'est pas le cas, une première requête est envoyée à Firebase afin d'ajouter d'uid de la nounou dans la liste des utilisateurs autorisés, avant d'envoyer la demande de rendez-vous.

Maintenant que la nounou peut disposer de toutes les informations concernant le rendez-vous, il est possible de s'attaquer à l'implémentation de la vue. Les captures d'écran ci-dessous représentent les vues implémentées pour l'affichage d'informations d'un rendez-vous :

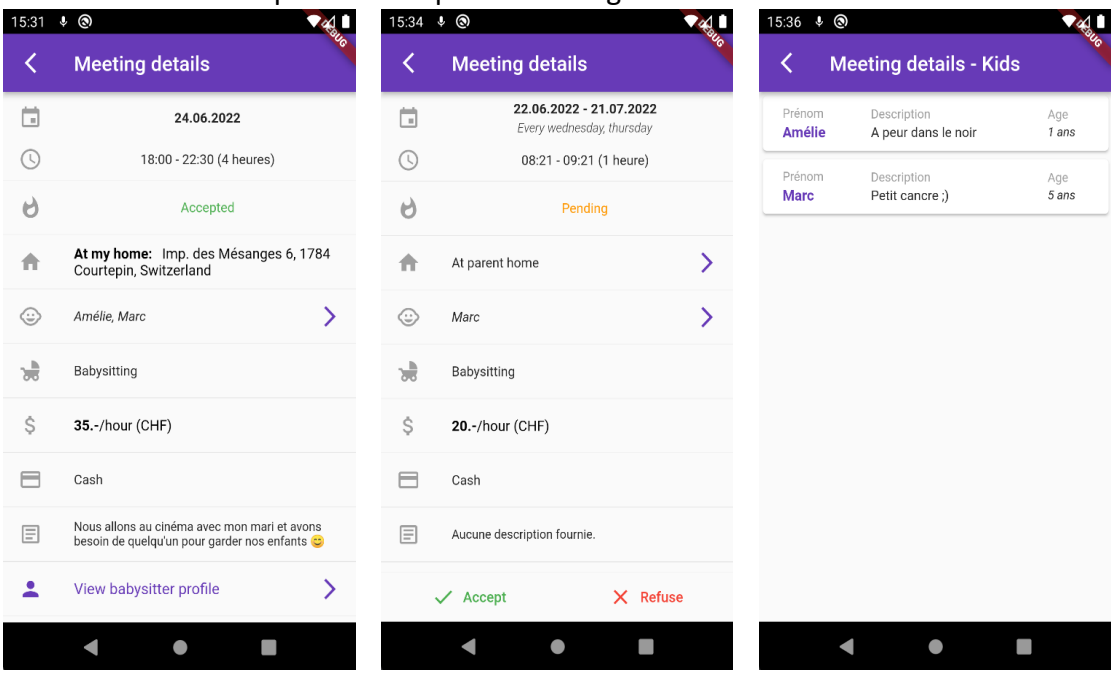

Figure 32 - Captures d'écran des vues permettant l'affichage détaillé des informations d'un rendez-vous unique (à gauche) ou hebdomadaire (au milieu) et des enfants concernés (à droite)

En plus des informations détaillées du rendez-vous, cette vue doit permettre à la nounou d'accepter ou de refuser les rendez-vous qui sont encore en attente. De même, un parent peut décider d'annuler un rendez-vous si jamais ce dernier n'a pas encore été confirmé. Pour cela, des boutons « accepter » et « refuser » sont disponibles pour la nounou, comme nous pouvons le voir sur la capture d'écran du milieu. De la même manière, un bouton « annuler » est disponible pour le parent, à condition que le rendez-vous n'ait pas encore été validé.

En complément des informations concernant le rendez-vous, cet écran permet également de consulter les enfants concernés par le rendez-vous et le lieu. Or, au niveau de Firebase, ces informations ne sont pas disponibles dans le document qui constitue le rendez-vous.

Pour les enfants concernés, du fait que la nounou est autorisée à accéder à la collection Firebase des enfants du parent et que chaque enfant est identifié par son id (disponible dans les informations du rendez-vous), il est possible d'effectuer une requête pour obtenir les informations. En revanche, pour le lieu, cela pose un problème car cela impliquerait de récupérer toutes les informations concernant l'autre utilisateur. Or, il n'est pas optimal de récupérer ces informations dans cette vue. Vous trouverez plus de détail concernant ce problème dans la section « 3.4.5.1 – Affichage des informations de l'utilisateur pour un rendez-vous ».

3.4.3 Liste des rendez-vous

Enfin, il reste une dernière vue concernant les rendez-vous qui n'a pas encore été mentionnée. Il s'agit de la vue permettant l'affichage simplifié de tous les rendez-vous concernant l'utilisateur. Cette vue étant déjà présente dans le prototype, seule quelques légères améliorations ont été effectuées. Par exemple, Si le rendez-vous est hebdomadaire, une petite icône est affichée afin d'informer l'utilisateur. Les captures d'écran ci-dessous représentent les vues réalisées :

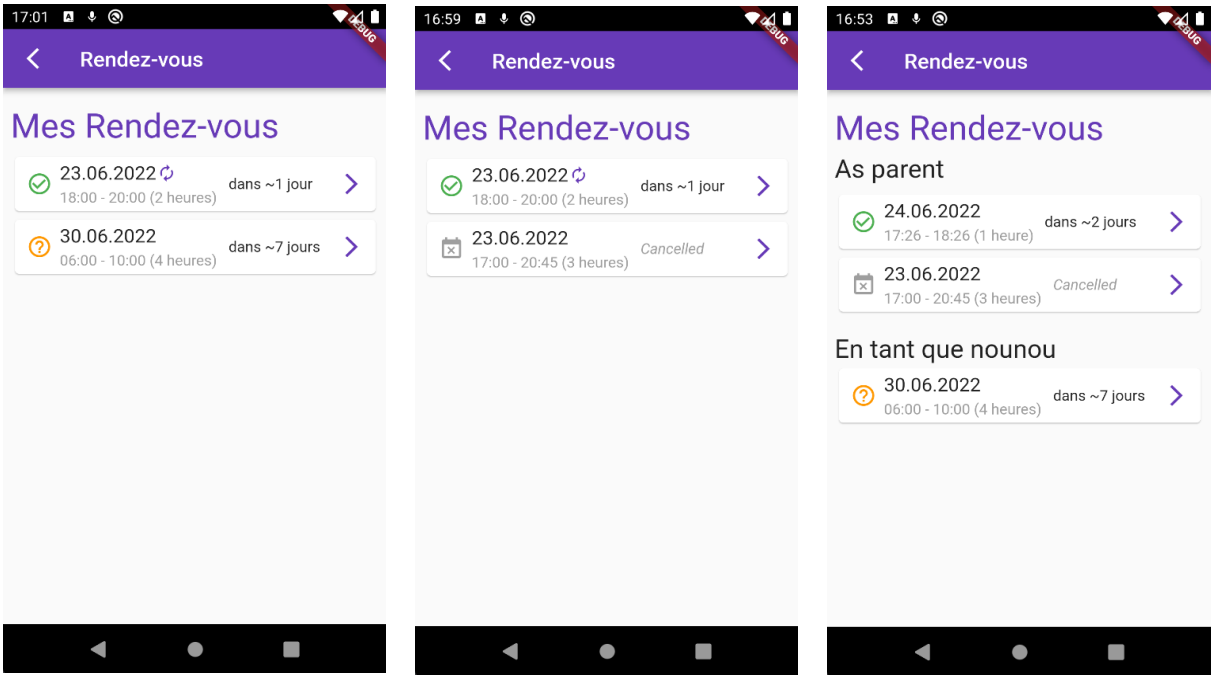

Figure 33 - Captures d'écran de l'affichage simplifié en tant que parent (à gauche), nounou (au milieu) ou les 2 (à droite)

Sur ces captures d'écran, l'on remarque que l'affichage est le même peu importe si l'on est un parent ou une nounou. En revanche, l'affichage change légèrement dans le cas où l'on a à la fois des rendez-vous en tant que parent et des rendez-vous en tant que nounou. Lors de rendez-vous hebdomadaires, c'est la date et l'heure du prochain rendez-vous planifié qui est affichée, comme on peut le voir sur le premier rendez-vous des deux premiers écrans. Comme précédemment, le statut du rendez-vous est également affiché. Il est possible de cliquer sur le rendez-vous pour naviguer sur l'écran détaillé de l'évènement.

3.4.4 Notification des rendez-vous

Enfin, une dernière fonctionnalité concernant la prise de rendez-vous est l'envoi de notifications à l'aide du « protocole » **F**irebase **C**loud **M**essaging, abrégé FCM. Jusqu'à présent, aucune notification n'est envoyée à la nounou quand cette dernière reçoit un nouveau rendez-vous. De même, aucune notification n'est envoyée au parent après la réponse de la nounou. Or, pour garantir une meilleur expérience utilisateur et pour donner plus de disponibilité, il est impératif de mettre en place un tel système.

Tout d'abord, il était prévu d'utiliser le protocole HTTP de FCM afin d'envoyer directement les notifications à partir du téléphone. Malheureusement, après de nombreux problèmes expliqués dans la section « 3.4.5.3 – Envoi de notifications FCM », l'utilisation des Cloud Function s'est avérée nécessaire du moins pour l'envoi de notifications FCM. La fonction en question écoute sur la création et la mise à jour de documents dans la collection des rendez-vous. Si un rendez-vous est détecté, alors la fonction récupère le jeton FCM de l'utilisateur associé et envoie une notification. Du fait de l'utilisation des Cloud Functions, il n'est à présent plus nécessaire pour les autres utilisateurs de connaître le jeton FCM. Il est alors possible, au niveau du profil utilisateur, de déplacer ce champ dans les informations privées de l'utilisateur. En effet, la fonction possède tous les droits, que ce soit en lecture ou en écriture, dans toutes les ressources de Firebase Firestore.

3.4.5 Problèmes rencontrés et résolution

Durant l'implémentation des diverses fonctionnalités nécessaires à l'amélioration des rendez-vous, plusieurs problèmes ont été rencontrés. Des petits problèmes mineurs, tels que la cohérence entre la date de début et la date de fin d'un évènement ont déjà été évoqués. Néanmoins, ce problème a rapidement pu être résolu. Cependant, d'autres problèmes plus complexes ont également été rencontrés.

3.4.5.1 Affichage des informations de l'utilisateur pour un rendez-vous

Durant la phase d'implémentation de la vue concernant les informations détaillées d'un rendez-vous, un problème est apparu : Comment faire pour afficher le lieu du rendez-vous ?

En effet, l'adresse de domicile correspondant au rendez-vous, que ce soit chez la nounou ou chez le parent, n'est pas stockée dans le document Firebase concernant l'évènement. Bien sûr, comme le rendez-vous possède les identifiants de la nounou et du parent et que la nounou est autorisée à accéder aux informations protégées du parent (dont son adresse), il est possible d'effectuer une requête Firebase pour récupérer le profil de l'autre utilisateur et ainsi afficher son adresse.

Cependant, comme il est également possible de naviguer sur le profil de la personne à partir des détails du rendez-vous, la réception du profil utilisateur se ferait alors deux fois : une première fois lors de l'affichage du rendez-vous afin d'afficher l'adresse de domicile, et une deuxième fois lorsque l'utilisateur affiche le profil de l'autre.

Or, plus de requêtes signifie également plus de potentiels coûts de facturation Firebase, en plus que ce n'est pas une solution optimale. Pour contourner cela, il a été décidé de ne pas récupérer les informations de l'utilisateur lors de l'affichage détaillé du rendez-vous et donc de ne pas afficher le lieu si ce dernier se trouve être chez l'autre utilisateur. A la place, le message « chez le parent » ou « chez la nounou » est affiché. Une autre solution possible serait de dupliquer l'information de la localité dans le rendez-vous, mais cette solution n'est pas vraiment meilleure…

3.4.5.2 Récupération et tri selon la date du prochain rendez-vous

Durant la phase d'implémentation de la vue concernant l'affichage de la liste de tous les rendez-vous concernés par l'utilisateur, un problème supplémentaire est survenu maintenant que les rendez-vous peuvent être hebdomadaires : Comment faire pour récupérer et ainsi afficher la date du prochain rendez-vous planifié ?

Si le rendez-vous est unique, ou alors si la première occurrence du rendez-vous hebdomadaire n'a pas encore eu lieu, il est très simple de déterminer la date du prochain rendez-vous. Il s'agit tout simplement de la date de début de l'évènement. De même, si la date de fin des rendez-vous se trouve être dépassée, alors la date du « prochain » ou plutôt du dernier rendez-vous se trouve être la date de fin de l'évènement. En revanche, si l'on se trouve actuellement entre la date de début et la date de fin d'un évènement hebdomadaire, alors il est nécessaire de prendre en compte les jours concernés par le rendez-vous afin de calculer la date du prochain rendez-vous. Mais alors, comment faire pour calculer cette date ? Pour rappel, voici les informations dont nous disposons :

- Le jours de la semaine concernés par le rendez-vous hebdomadaire
- La date de début et la date de fin de l'évènement
- La date d'aujourd'hui

De plus, les objets Flutter de type `DateTime` permettent également de comparer deux dates, ou encore d'ajouter une certaine durée à notre date (en minutes, heures, jours, etc.). Une première solution imaginée consistait à partir de la date de départ et à itérer de jours en jours jusqu'à trouver le prochain jour où l'évènement a lieu. Néanmoins, cette solution n'est pas la plus optimale car le nombre d'itération risque d'être très élevé.

Une deuxième solution imaginée consistait à itérer de semaines en semaines. En effet, comme les rendez-vous sont hebdomadaires, il est possible de naviguer de semaine en semaine et de vérifier tous les jours de la semaine acceptée par le rendez-vous lors d'une itération. Néanmoins, pour cela, il est nécessaire de récupérer le premier lundi. Pour ce faire, après quelques recherches, il est possible de modifier la date à l'aide de l'attribut *weekday* afin de soustraire les éventuels jours de la semaine et ainsi récupérer le 1er jour de la semaine (lundi). Le pseudo code ci-dessous montre comment il est possible d'obtenir cette date :

```
Monday = startDate.substract(days: startDate.weekday)

While (Monday < endDate):
    Check for each scheduled day if the date is after today and return it.
    If no valid date found, Monday += 7 days and next iteration
```

Cette solution est satisfaisante, cependant après quelques réflexions supplémentaires on remarque tout de même qu'une partie du calcul est inutile : Comme l'on connait la date d'aujourd'hui et que l'on est sûr que le prochain évènement ne peut pas se situer dans le passer, pourquoi itérer à partir de la date de début du rendez-vous ? Le mieux à faire serait d'itérer à partir de la semaine courante. Cette modification est possible en changeant simplement la première ligne du pseudo-code :

```
Monday = today.substract(days: today.weekday)
```

De cette manière, la recherche de la date du prochain évènement planifié est bien plus rapide. En effet, le nombre d'itération se retrouve largement réduit : seule 1 ou 2 itérations sont possible, dépendamment du contexte :

- S'il y a un jour de la semaine courante planifié selon l'évènement et que ce jour se trouve être après la date d'aujourd'hui, alors seule 1 itération sera effectuée pour trouver la date. Par exemple : nous sommes mercredi et le rendez-vous est planifié tous les vendredis.
- Si par contre tous les jours planifiés datent d'avant la date d'aujourd'hui, alors il est nécessaire de sauter à la semaine suivante en effectuant une deuxième itération pour trouver la date du prochain rendez-vous. Par exemple : nous sommes mercredi et le rendez-vous est planifié tous les mardis. Ce mardi étant déjà passé, c'est le mardi d'après qui doit être retourné.

Avec cette solution, cela permet à l'application de calculer et de connaître rapidement la date du prochain rendez-vous. Bien sûr, il est important de rappeler que tout cela ne concerne que les rendez-vous de type hebdomadaires qui ont déjà eu lieu et ne sont pas encore terminés. Maintenant que la date est connue, il est encore nécessaire de s'intéresser au tri des rendez-vous. Jusqu'à présent, les rendez-vous obtenus dans Firebase étaient classés selon la date de départ de l'évènement. Le tri est effectué lors de la requête, au moyen d'un index composite. Si la personne ne possède que des rendez-vous uniques, les évènements sont donc déjà triés selon nos besoins. En revanche, si l'utilisateur possède un rendez vous hebdomadaire, il est possible de devoir retrier la liste en prenant en compte pour cet évènement non pas la date de départ mais la date de la prochaine occurrence.

Afin d'effectuer le tri, il est possible d'ajouter au niveau Frontend la date du prochain rendez-vous planifié directement dans le modèle du rendez-vous. Ainsi, il est possible d'utiliser la méthode `sort()` comme le montre l'exemple de pseudo code ci-dessous :

```
For each meeting in the list:
   If it's unique or not started yet, set meeting.nextDate = meeting.startDate
   If it's not, determine meeting.nextDate using the previous pseudo-code

If we need to sort again because there were weekly meetings:
   Meetings.sort(meeting1.nextDate.compareTo(meeting2.nextDate))
```

Pour optimiser cette méthode, il est possible de vérifier au préalable si meeting.nextDate est déjà défini ou non. Si tel est le cas, pas besoin de recalculer la date et donc il n'y a potentiellement pas besoin non plus de trier la liste à nouveau.

3.4.5.3 Envoi de notifications FCM

Enfin, le problème ayant pris le plus de temps à résoudre concerne l'envoi de notifications FCM. Cette fonctionnalité n'avait pas du tout été explorée durant le projet de semestre précédent par manque de temps et il restait donc quelques doutes quant à l'implémentation de cette fonctionnalité.

Initialement, il était prévu d'utiliser directement le package Flutter « *firebaseMessaging* » [50] pour envoyer des notifications à l'autre utilisateur. Après quelques recherches dans la documentation du package, il semblerait que la méthode `sendMessage()` peux convenir pour l'envoi de notifications. Cependant, après quelques recherches plus approfondies, cette méthode ne permet pas directement l'envoi de notification à un autre utilisateur mais à un serveur FCM qui implémente le protocole XMPP, comme expliqué dans la documentation Firebase [51]. Or, nous ne cherchons pas ici à mettre en place un tel serveur.

Une deuxième possibilité qui a été étudié est le fait d'utiliser directement des requêtes HTTP pour l'envoi de notifications. En effet, la documentation de Firebase explique qu'il est possible d'envoyer une requête POST sur l'url https://fcm.googleapis.com/fcm/send afin d'envoyer des notifications à d'autres utilisateurs. Cependant, un autre problème survient : Pour autoriser les requêtes sur le serveur, ces dernières doivent être authentifiées à l'aide d'un jeton d'enregistrement OAuth v2.0 ou le package Firebase Admin SDK [52]. Or, ces deux options ne sont pas vraiment disponibles avec Flutter, ou du moins il n'est pas recommandé d'utiliser Flutter pour effectuer directement les requêtes. Il serait alors nécessaire de mettre en place un autre serveur tourant par exemple avec Node.js s'occupant de gérer l'envoi de notifications. Cependant, cette solution est assez couteuse à la fois en temps et en ressource.

Enfin, une dernière solution envisagée qui se trouve être l'option choisie pour résoudre ce problème consiste à passer par les Cloud Functions. Durant beaucoup de problèmes rencontrés, l'utilisation des Cloud Functions a été prise en compte. Souvent, il s'agissait de la solution la plus optimal pour résoudre le problème, cependant cela nécessitait l'étude et l'apprentissage d'un tout nouvel outil que sont les Cloud Functions. Or, pour les problèmes précédemment évoqués il existait également une autre solution, moins optimale, mais plus rapide à mettre en place et n'utilisant pas de nouvelles technologies.

En revanche, dans le cadre de l'envoi des notifications FCM, l'alternative a l'utilisation des Cloud Functions consisterait à mettre en place un serveur HTTP pour gérer l'envoi des notifications, ou alors de demander des jetons d'enregistrement directement depuis Flutter. De ce fait, ces solutions alternatives sont tout aussi complexes et n'apportent pas vraiment d'avantages que ce soit en temps ou en performances, par rapport à l'utilisation des Cloud Functions.

Pour ces raisons, il a été décidé après quelques réflexions de tout de même utiliser les fonctions Firebase dans le cadre de ce projet de Bachelor. En revanche, les fonctions ne serviront uniquement qu'à l'envoi des notifications FCM. Les autres problèmes cités ont amont qui ont déjà été résolu sans l'utilisation des Cloud Functions ne seront pas réevalués. De cette manière, seule l'étude des fonctionnalités des base des Cloud Functions et leur utilisation dans un contexte d'envoi de notification seront étudiées.

Pour l'étude et la mise en place des notifications FCM, la documentation de Firebase sur les Cloud Functions fourni un exemple basique d'environnement et de fonctions permettant l'envoi de notification FCM lors d'un évènement précis au niveau de la base de données [53] :

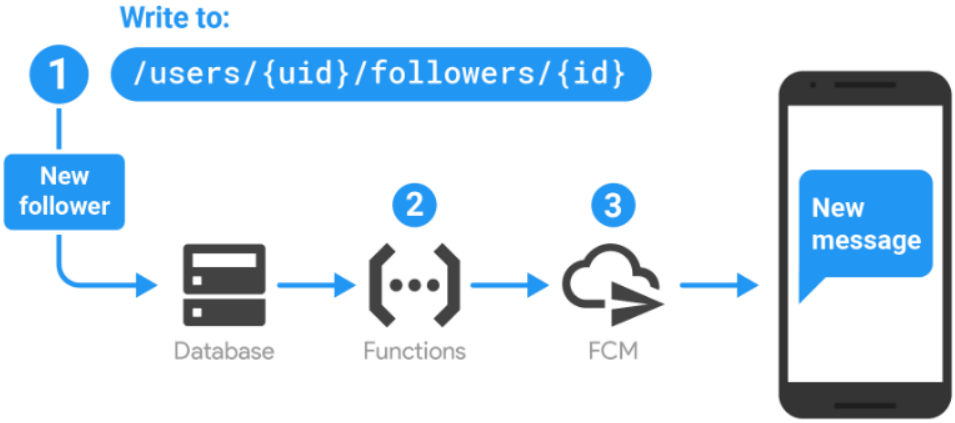

Figure 34 - Schéma d'envoi de notification FCM par le biais d'une Cloud Functions lors d'un évènement Firebase - repris de [53]

1. Un utilisateur modifie un document, ce qui déclenche la fonction
2. La fonction compose le message FCM à envoyer
3. FCM envoie la notification à l'appareil de l'utilisateur

Dans le code d'exemple disponible, il est également expliqué comment récupérer d'autres informations en provenance de Firebase avant d'envoyer la notification. Cette étape est également nécessaire dans notre projet car il faut que la fonction récupère le jeton FCM qui se situe dans les documents concernant l'utilisateur. Pour connaître l'identifiant de l'utilisateur, la fonction utilise les champs *parentUid* et *babysitterUid* présents dans le rendez-vous, en fonction du contexte.

Avec cet exemple, il est donc tout à fait possible d'adapter le code fourni pour qu'il corresponde avec notre environnement Firebase. Premièrement, il est nécessaire d'utiliser deux fonctions distinctes, l'une écoutant sur la création de rendez-vous et l'autre sur la modification d'un rendez-vous. Lors de la création d'un nouvel évènement, c'est la nounou qui doit être notifiée, alors que pour la modification d'un évènement c'est le parent qui doit être notifié (lorsque le statut passe à « accepté » ou « refusé »). Pour ce faire, une requête à Firebase est effectuée sur le profil de l'utilisateur afin de récupérer son jeton FCM. L'avantage étant que comme la fonction dispose de droits administrateurs, il est à présent possible de stocker le jeton dans les données privées de l'utilisateur, s'assurant ainsi que les autres utilisateurs ne pourront jamais accéder à cette information. Une fois le jeton FCM obtenu, il est alors possible d'envoyer la notification à l'utilisateur.

Cependant, deux problèmes complémentaires sont apparus avec cette solution : Premièrement, que se passe-t-il si l'utilisateur est authentifié sur plusieurs appareils avec son compte et qu'une notification doit être envoyée ? Sur quel appareil faut-t-il envoyer la notification ? Deuxièmement, comment faire pour traduire la notification, sachant qu'au moment de l'envoi de la notification par la fonction cette dernière ignore complètement la langue du téléphone de l'utilisateur.

Pour le premier problème, il a été décidé que seule une notification serait envoyée sur l'appareil le plus récemment utilisé. Une autre solution envisageable serait de transformer le champ *fcmToken* en une liste de chaine de caractères, afin de pouvoir stocker plusieurs jetons FCM pour un même utilisateur.

Pour le second problème, après quelques recherches dans la documentation de Firebase et plus particulièrement de FCM, il est possible d'utiliser les champs *titleLocKey* et *bodyLocKey* afin de spécifier non pas un texte pour le titre et le corps de la notification mais une clé, sous forme de chaîne de caractère. Ensuite, cette clé sera utilisée localement par le téléphone pour récupérer le texte correspondant, en fonction de la langue du téléphone. Pour ce faire, sur Android, il est nécessaire de fournir les traductions des clés dans les ressources de l'application, comme le montre l'image ci-dessous :

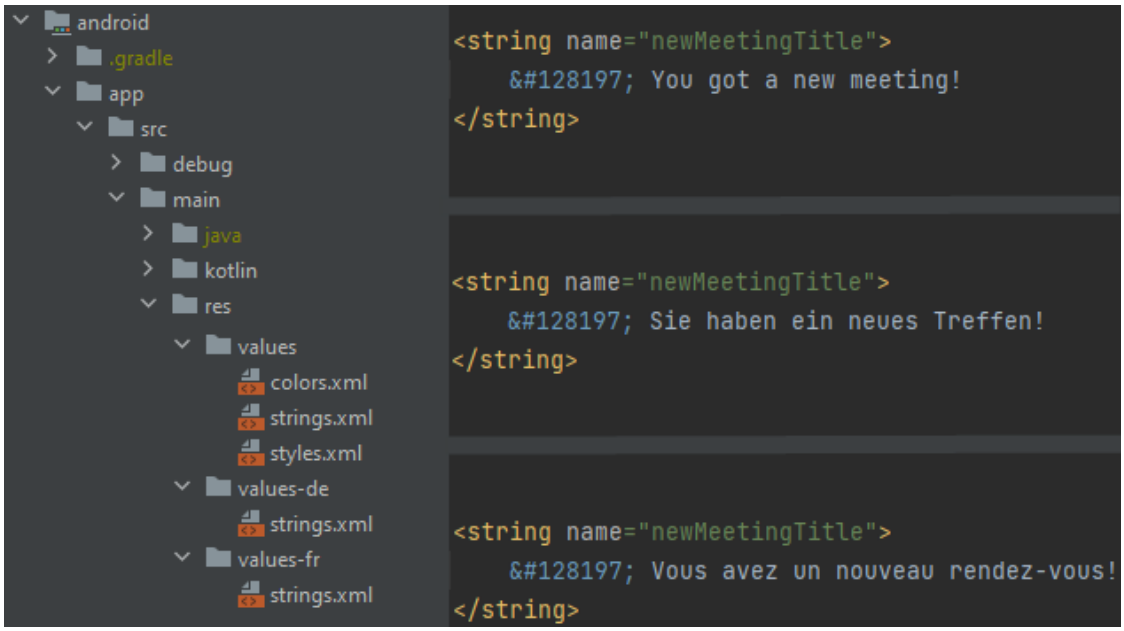

Figure 35 - Capture d'écran des ressources Android nécessaires à la traduction des notifications

Sur cette capture d'écran, l'on remarque qu'un fichier *strings.xml* est présent pour chaque langue supportée, dans le dossier `values-xx` correspondant à la langue. Sur la droite de l'image, l'on constate que chaque fichier *strings.xml* contient sa propre valeur pour la même clé *newMeetingTitle*, avec le message de la notification traduit selon la langue. Sur iOS, en revanche, il est nécessaire de définir les traductions dans le fichier `Localizable.strings` :

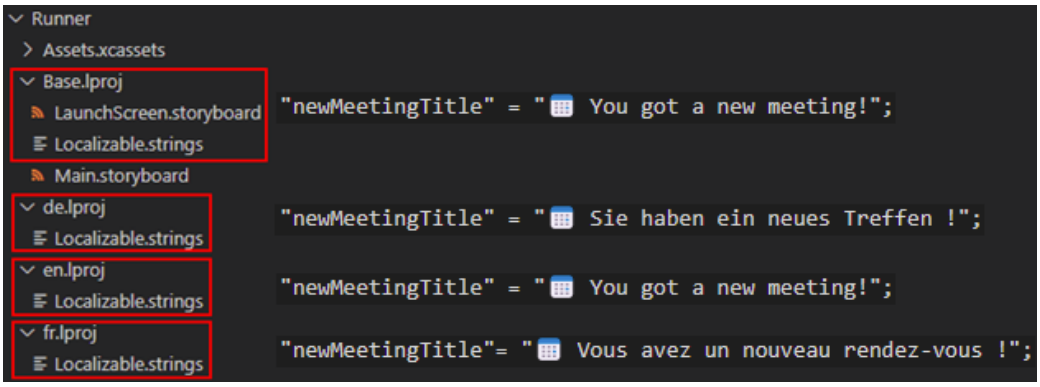

Figure 36 - Captures d'écran des ressources iOS nécessaires à la traduction des notifications

3.5 Tests utilisateurs

En plus des objectifs se focalisant directement sur le développement de l'application, des tests utilisateurs ont également été réalisés. Bien sûr, cela ne sert à rien de soumettre des questionnaires de satisfactions si les résultats ne sont pas par la suite étudiés.

Pour rappel, durant la semaine de test deux questionnaires différents ont été réalisés : Un premier type de questionnaires se focalisant sur la réalisation d'un scénario en particulier (au nombre d'un questionnaire par scénario), ainsi qu'un questionnaire de satisfaction global permettant à l'utilisateur de s'exprimer sur l'intégralité de l'application.

3.5.1 Résultats des questionnaires sur les scénarios

Les questionnaires basés sur les scénarios reprennent les modèles ASQ et SEQ expliqués dans la section d'analyse « 2.1.3 – Tests utilisateurs ». Pour chaque fonctionnalité de l'application, l'utilisateur pouvait noter de 1 à 7 la difficulté d'utilisation, l'utilité et l'intégration de la fonctionnalité. Concernant la difficulté d'utilisation, une étude réalisée avec 286 tâches et un nombre de participant variant entre 8 et 601 [54] a identifié que le score moyen était d'environ 5.5, comme le montre la figure ci-dessous :

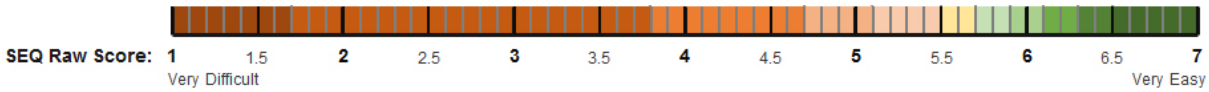

Figure 37 - Classification du score SEQ, repris de [54]

Voici les résultats obtenus concernant l'application HappyBaby. En moyenne 9 personnes différentes ont répondu aux questionnaires :

Difficulté		
Score moyen	**6,336**	
	Score	*Objectif(s) / tâche(s)*
Meilleur score moyen	7	Afficher les détails d'un rendez-vous
Pire score moyen	5,57	Rechercher une nounou
Score le plus bas	1	Gérer un rendez-vous (accepter/refuser/annuler)

Nous pouvons constater que le score moyen donné à la difficulté d'utilisation des fonctionnalités présentes dans l'application est supérieur à 6. Globalement, l'application n'est donc pas difficile à utiliser. Le tableau indique également la meilleur et la pire moyenne obtenue pour la réalisation d'une tâche, ainsi que la plus mauvaise note donnée par un utilisateur. Nous pouvons constater que l'objectif avec le score moyen le plus bas reste tout de même légèrement au-dessus de la moyenne de 5.5. En revanche, nous pouvons également constater qu'au moins un utilisateur a attribué la note de 1 pour la gestion des rendez-vous. Bien que la moyenne pour cette tâche soit de 5,6 il serait tout de même intéressant de mentionner cette fonctionnalité pour des améliorations futures.

Si l'on compare les résultats obtenus selon le système d'exploitation du téléphone, nous n'obtenons pas vraiment de différences dans les scores. En revanche, si l'on sépare les résultats selon l'âge des participants, on constate que les personnes âgées de 50 à 70 ont eu légèrement plus de difficultés à utiliser l'application, comme le montre la figure ci-dessous:

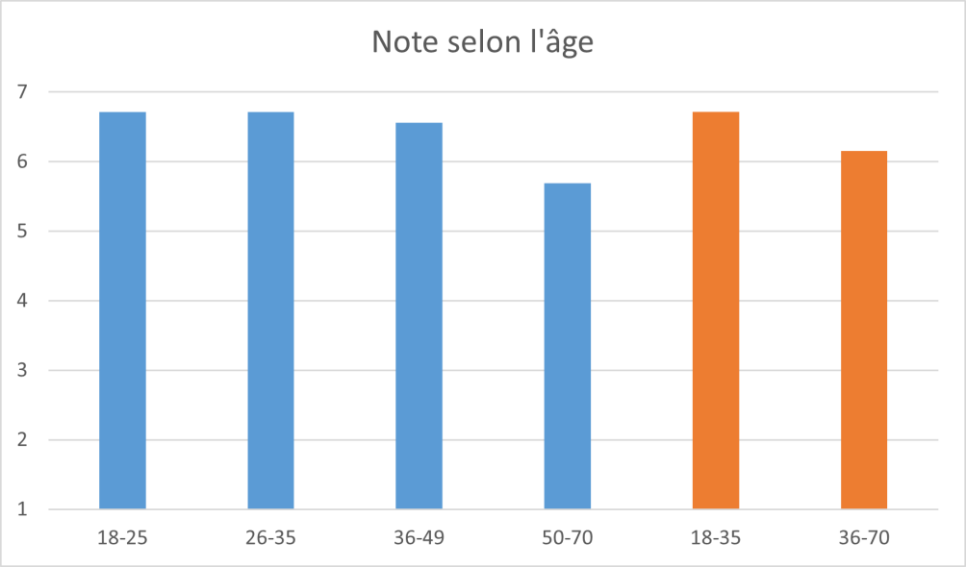

Figure 38 - Score obtenu pour la difficulté de réalisation selon la tranche d'âge de l'utilisateur

Sur cette figure, on observe que le score moyen attribué par les personnes de 18 à 49 ans se situe aux alentours de 6.5, alors que pour les personnes âgées de 50 à 70 ans le score est plutôt de 5.5. Bien sûr, l'âge lui-même explique en partie cette différence, car ces personnes n'ont connu le smartphone que récemment dans leur vie et ont généralement plus de difficultés à s'adapter aux technologies actuelles. Cependant, cela s'explique certainement aussi par le fait que l'application a été réalisées par un jeune étudiant.

Concernant les scores attribués à l'utilité ou à l'intégration des fonctionnalités, il n'existe pas d'échelle standard ni d'études permettant de comparer le résultat obtenu avec d'autres notes. Il est tout de même intéressant d'observer les résultats obtenus :

Utilité		
Score moyen	**6,557**	
	Score	*Objectif(s) / tâche(s)*
Meilleur score moyen	7	4 objectifs différents ont eu la note moyenne de 7
Pire score moyen	6	Modifier son mot de passe
Score le plus bas	3	Créer un compte, modifier son mot de passe

Intégration		
Score moyen	**6,45**	
	Score	*Objectif(s) / tâche(s)*
Meilleur score moyen	7	Consulter le profil d'une nounou (via recherche)
Pire score moyen	5,55	Modifier son mot de passe
Score le plus bas	1	Gérer un rendez-vous

Sur ces tableaux, nous pouvons remarquer que la manière actuelle qu'à l'utilisateur de modifier son mot de passe soit la fonctionnalité la moins utile et la plus mal intégrée. Nous pouvons également remarquer que le score moyen est dans les deux cas très bon et même supérieur au score moyen de difficulté de réalisation de la tâche. Des comparaisons en

fonction de l'âge ou encore du système d'exploitation ont été réalisées, cependant il ne semble pas y avoir de différences significatives pour ce qui concerne l'utilité et l'intégration.

En plus des tâches, le questionnaire présentait également des questions se focalisant sur la réalisation du scénario en entier, en utilisant le modèle ASQ. La encore, il n'existe pas vraiment d'échelle permettant de mieux interpréter les résultats, bien que ce modèle soit populaire. Cependant, même sans échelle il est possible d'interpréter les résultats obtenus :

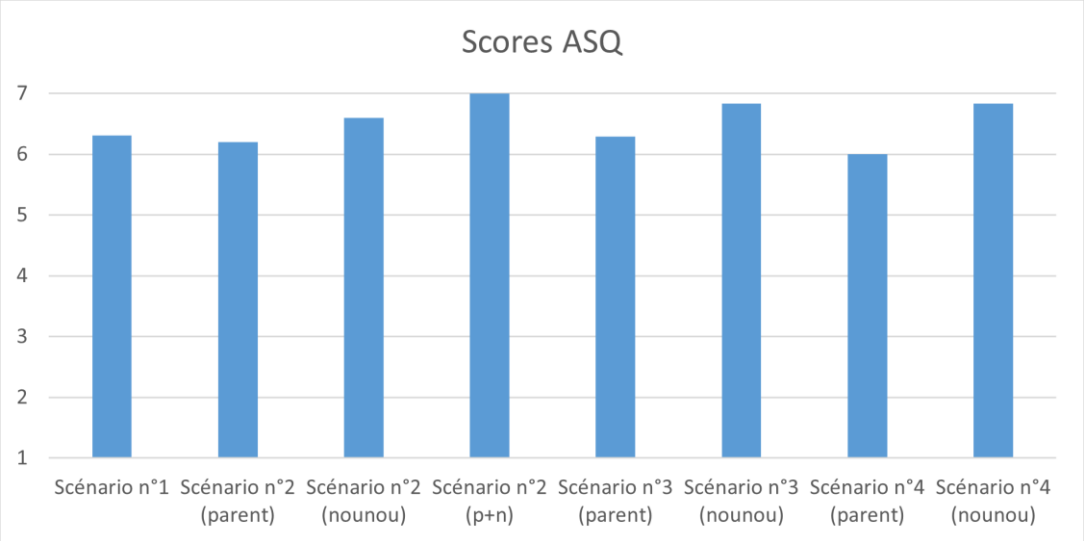

Figure 39 - Scores ASQ obtenus pour chaque scénario réalisé

Premièrement, l'on constate que le score moyen obtenu pour chaque scénario est très élevé, ce dernier n'allant jamais en dessous de 6. Les scénarios réalisés ont donc globalement convenu aux utilisateurs et les tâches à réalisées n'étaient pas trop longues ou trop fastidieuses.

Deuxièmement, l'on constate que les scénarios complétés par les nounous ont en général une moyenne légèrement plus élevée que ceux réalisés par les parents. Si l'on affiche ces résultat côte-à-côte on observe encore mieux l'écart, qui varie entre 0.5 jusqu'à presque 1 pour le scénario n°4 :

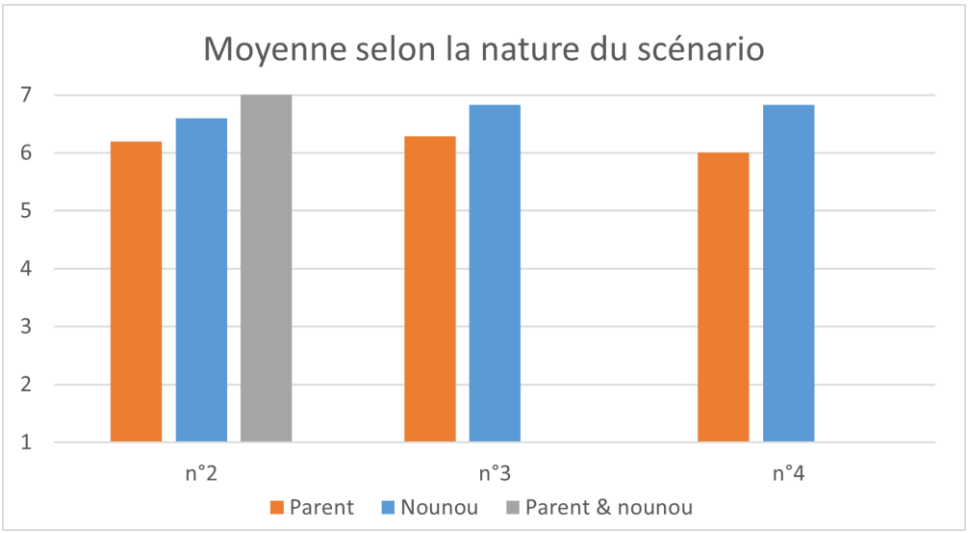

Figure 40 - Scores ASQ obtenus selon la nature de l'utilisateur (parent ou nounou)

Cette différence s'explique par deux points : Premièrement, les utilisateurs ayant le rôle de nounou sont en moyenne plus jeunes que ceux ayant le rôle de parent. L'on peut donc imaginer qu'ils ont plus de facilité à résoudre les objectifs demandés. Mais il y a également un autre point : Les scénarios réalisés par les parents sont plus longs et plus compliqués que ceux réalisés par les nounous. En effet, une fois les tâches communes réalisés, le parent utilise les fonctionnalités de recherche et de demande de rendez-vous, alors que les nounous n'ont pas besoin d'utiliser ces fonctionnalités dans aucun scénarios présentés.

Enfin, à propos de l'utilisation de la procédure pour réaliser les objectifs, les résultats des participants montrent que peu d'utilisateurs ont utilisés la procédure pour réaliser les scénarios, à l'exception du premier scénario ou beaucoup de personnes pensaient qu'il était obligatoire de suivre la procédure pour compléter les objectifs, comme le montre la figure ci-dessous :

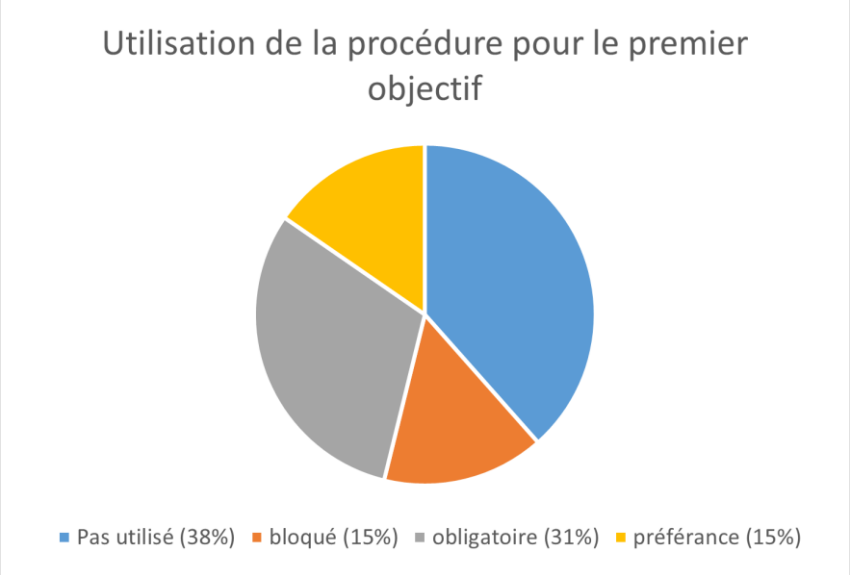

Figure 41 – Parts d'utilisation de la procédure pour le premier objectif à réaliser

L'information comme quoi les procédures n'étaient pas obligatoires n'a donc pas bien été communiquée aux participants, car une bonne partie d'entre eux n'avaient pas compris cela durant la réalisation des premiers objectifs. Une fois que la personne remplissait le premier questionnaire, elle comprenait par le biais de la question « Avez-vous utilisé la procédure ? » que l'aide distribuée était facultative. Pour les objectifs des autres scénarios, il n'y avait en moyenne que 13% des utilisateurs qui suivaient la procédure pour réaliser les objectifs, principalement car ces derniers préféraient faire ainsi.

L'objectif qui a suscité le plus d'utilisation de la procédure en raison de blocage est la gestion des rendez-vous. En plus des résultats précédemment obtenus qui, eux aussi, mettait en avant le fait que la gestion des rendez-vous était la fonctionnalité la moins appréciée.

3.5.2 Résultats du questionnaire d'évaluation global

Une fois la semaine de test terminée et les scénarios réalisés, un questionnaire de satisfaction global à été soumis aux participants. Ce questionnaire vise à évaluer l'utilisabilité et l'expérience utilisateur vis-à-vis de l'application dans sa globalité.

Les premiers résultats concernent le modèle SUS utilisé pour mesurer l'utilisabilité d'un produit. Pour rappel, les 10 questions posées permettent d'attribuer un score SUS pour chaque utilisateur, compris entre 0 et 100. Plus la note est élevée, mieux l'application est pratique d'utilisation. En général, il est recommandé d'avoir un score d'au moins 68. La figure ci-dessous montre les résultats des 13 utilisateurs obtenus pour l'application HappyBaby :

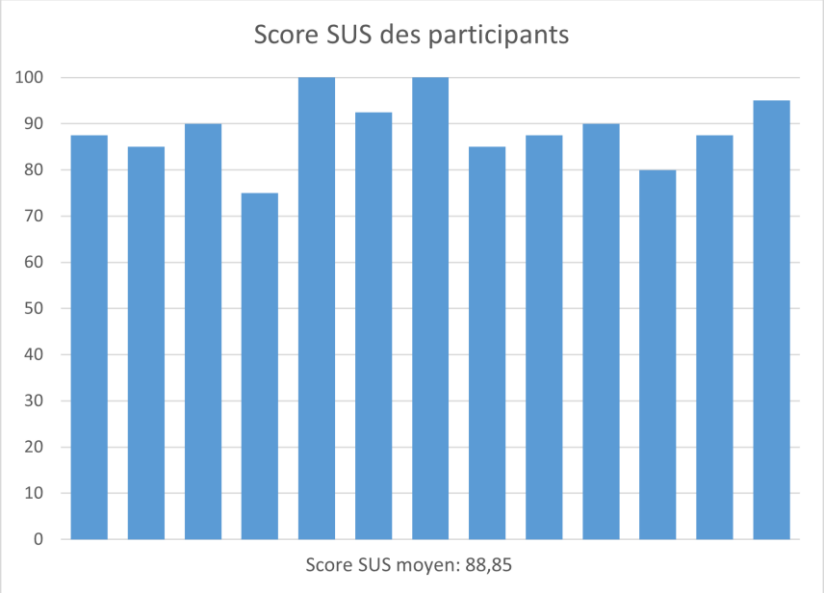

Figure 42 - Score SUS obtenu pour chaque participant

Sur cette figure, nous pouvons constater que le score SUS varie entre 75 et 100, avec une moyenne de 88,85. Nous pouvons donc en déduire que l'application est facile à utiliser, assez intuitive et que peu de fonctionnalités sont incomprises. Les commentaires laissés par les utilisateurs confirment un tel résultat, en indiquant pour la plupart que l'application est simple, fluide et/ou intuitive.

Deuxièmement, une question basée sur le modèle NPS avait pour objectif d'identifier la fiabilité des utilisateurs. Pour rappel, le modèle NSP consiste à séparer en 3 catégories différentes les participants, en fonction du résultat donnée par la personne. En comparant les pourcentages de participants obtenus pour chaque catégorie, il est possible d'attribuer un score NPS au service désiré. La note varie entre -100 et 100. Le graphique ci-dessous indique les parts obtenues pour chaque catégorie d'utilisateur et le score obtenu :

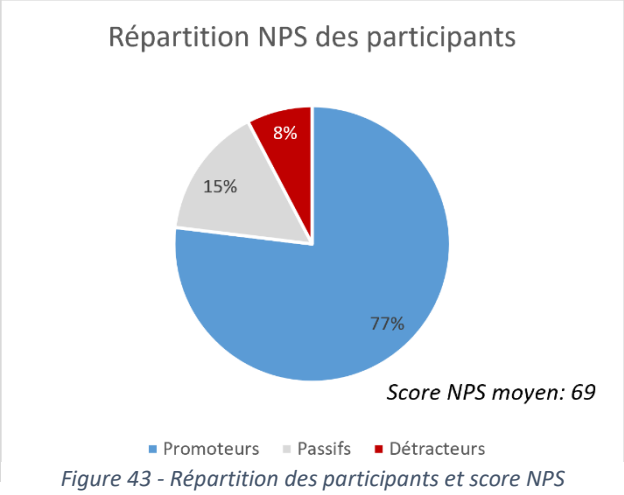

Figure 43 - Répartition des participants et score NPS

Sur cette figure, nous pouvons observer que plus de 75% des participants sont catégorisés comme promoteurs, signifiant que si l'application venait effectivement à sortir sur le marché, ces derniers serait très fidèles et participeraient indirectement à la démocratisation de l'application. Le score NPS de 69 et, en comparaison avec d'autres applications mobiles ou programmes informatiques, largement supérieur à au score moyen de 34.

Enfin, un dernier modèle de questionnaire, nommé AttrakDiff, a été soumis aux participants dans sa version courte, qui ne comporte que 10 questions. Ce dernier sert à évaluer l'expérience utilisateur en attribuant un score pour quatre aspects différents de l'UX.Pour rappel, l'échelle varie entre -3 et +3 pour chaque catégorie. Le graphique ci-dessous montre les scores moyens obtenus dans chaque catégorie :

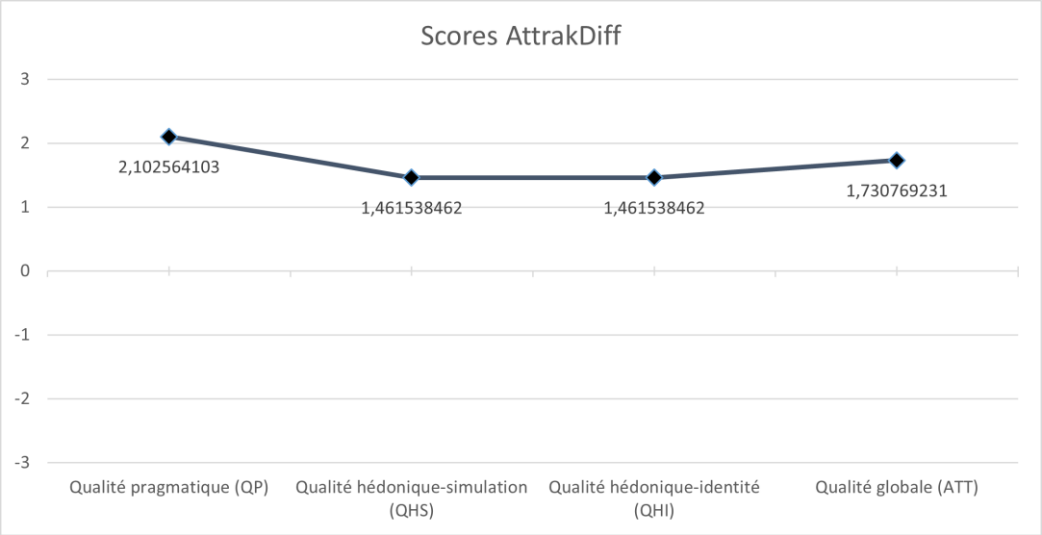

Figure 44 – Scores AttrakDiff obtenu dans chaque catégorie

Une première bonne nouvelle que nous pouvons constater est qu'aucune catégorie de l'UX n'est inférieur à 0, ce qui indique une expérience moyenne positive dans chaque domaine. Deuxièmement, l'on remarque que les scores moyens obtenus varient entre ~1.5 et 2.1. Il n'y a donc pas de grande variation du score entre les différents éléments de l'UX mesurés. Cela indique donc que l'expérience de l'utilisateur est la même pour tous les aspects présentés.

Enfin, deux questions supplémentaires étaient posées pour savoir si l'application et les tests avaient répondu aux attentes et avait satisfait l'utilisateur. Avec une moyenne d'environ 4.7/5, nous pouvons donc affirmer que l'application et les tests proposés ont été globalement bien reçus par les utilisateurs, et que ces derniers ont pris du plaisir à utiliser l'application.

Initialement, il était prévu de comparer les résultats obtenus en fonction de l'âge ou encore du statut professionnel (employé, étudiant, etc.) de l'utilisateur. Malheureusement, cette étude ne pourra finalement pas être faite dans le cadre de ce projet, car les délais et la charge de travail restant avant le rendu du projet ne permettent pas de comparer et d'étudier convenablement les résultats.

4 Discussion

Cette section vise à discuter des différents résultats obtenus ou réalisés dans le cadre de ce projet de Bachelor. Cette section a également pour objectif de discuter des améliorations possibles de l'implémentation finale produite.

4.1 Structure Firebase

Premièrement, bien que la structure des données dans Firebase ait déjà été réévaluée dans le cadre de ce projet de Bachelor, par rapport au projet de semestre précédent, l'on peut se demander si la structure choisie est bien adaptée à notre application.

En effet, la restructuration des données à été faite dans ce projet afin de résoudre différents problèmes d'optimisation et/ou d'utilisation entre Firebase et Flutter. Avec l'ancien modèle, il n'était par exemple pas possible de définir des droits différents pour certains champs de l'utilisateur, ce qui est à présent possible. Or, de nouvelles limitations ont été rencontrées durant l'amélioration des fonctionnalités, en particulier pour la recherche avancée. Pour pouvoir effectuer une recherche avec différents champs concernés de manière optimisée en noSQL, il faut en effet bien étudier les limitations qu'apporte cet outil en ce qui concerne l'équivalent des clauses « *WHERE* » en SQL.

Malheureusement, du fait que ces limitations n'étaient pas bien connues avant de réaliser le projet, la structure actuellement conçue ne convient pas pour effectuer des recherches avancées de manière optimisée ! Pour améliorer les performances lors de la recherche, il serait intéressant d'étudier plus en profondeurs les limitations de Firestore au niveau des requêtes et de réaliser une nouvelle structure des données plus adaptée.

4.2 Utilisation de Cloud Storage et Google Maps API

Avec l'analyse des frais d'utilisation des services de Google, il a pu être identifié que les deux services responsables à 99% de la facturation concerne l'utilisation de Cloud Storage et de l'API Google Maps. Il est donc tout naturel de se questionner sur l'utilité de ces services et s'il n'existerait pas des alternatives moins onéreuses.

Au niveau de Cloud Storage, une première amélioration évidente qui peux être apportée est le fait de compresser (avec ou sans perte) la photo choisie par l'utilisateur au niveau Frontend, pour ensuite l'envoyer et la stocker dans Cloud Storage. Ainsi, il est possible de drastiquement réduire le trafic réseau généré par Cloud Storage lors de la réception des photos de profils des utilisateurs. Une deuxième amélioration consisterait à stocker en local les photos des utilisateurs téléchargés (notamment sa propre photo de profil). Ainsi, si le profil doit être réaffiché ultérieurement, il ne serait pas nécessaire de retélécharger la photo à partir de Cloud Storage. Pour savoir si la photo en local correspond toujours avec la photo hébergée, il serait par exemple possible de stocker le « hash » (en quelques sortes l'empreinte digitale de la photo) en plus de la photo. Ainsi, seul le hash devrait être téléchargé et comparé avec le hash obtenu à partir de la photo en locale. Si les deux chaînes sont identiques, alors la photo n'a pas été changée et il n'est pas nécessaire de la télécharger à nouveau. Si cependant il y a une différence, alors cela veut dire qu'une nouvelle photo est disponible et doit être téléchargée.

Enfin, une dernière solution qui pourrait être apportée serait de ne tout simplement pas utiliser Cloud Storage mais un autre serveur d'hébergement d'image. Il serait par exemple possible d'utiliser les services d'autres compagnies tels que Amazon AWS ou Microsoft Azure, voir même de réaliser son propre serveur d'hébergement de fichiers. Bien sûr, un tel changement implique au préalable d'estimer les coûts pour être sûr que c'est un changement bénéfique.

Concernant l'utilisation de l'API Google Maps, il n'est pas vraiment possible de réduire le volume de requêtes en compressant d'une manière ou d'une autre l'information ou encore en garant dans une sorte de cache certaines données. Le seul moyen de réduire les coûts d'utilisation de Google Maps et donc de ne pas utiliser l'API mais des alternatives. Pour rappel, il y a trois utilisations possibles de l'API :

- Lorsque l'utilisateur entre une lettre dans la recherche d'adresse (autocomplétion)
- Lorsque l'utilisateur sélectionne une adresse parmi les résultats (geocoding inversé)
- Lorsque l'utilisateur utilise son GPS pour définir son adresse (geocoding)

Concernant l'autocomplétion, après quelques recherches sur internet il existe certains projets utilisant non pas les services de Google Maps mais d' OpenStreetMap [55] proposant une autocomplétion d'adresse, comme c'est par exemple le cas pour le projet « *Leaflet Autocomplete* » [56] ou encore « *MOTIS Project* » [57]. En revanche, contrairement à l'API Google Maps, ces services ne fournissent pas l'hébergement du serveur permettant d'effectuer les requêtes d'autocomplétion.

Concernant le geocoding, il est tout à fait possible de ne pas passer par l'API de Google Maps. En effet, à la fois Google (Android) et Apple (iOS) proposent un service de Geoconding gratuit permettant de traduire des coordonnées GPS en une adresse et vice-versa. En prime, un package Flutter se basant sur ces fonctionnalités a déjà été réalisé [58]. Il y a cependant une difficulté supplémentaire à prendre en compte : En utilisant partout les services de Google Maps API, on était assurés d'obtenir un geocoding inversé. Or, ce n'est plus forcément le cas si l'on utilise un premier service tels que MOTIS pour obtenir l'adresse puis le package Geocoding de Flutter pour traduire l'adresse en coordonnées GPS. En effet, rien ne garanti que le package Geocoding reconnaisse l'adresse fournie par MOTIS.

4.3 Estimation des frais d'utilisation des services de Google

Durant la phase d'analyse des frais engendrés par l'utilisation des services de Google, il a été décidé d'estimer les frais selon chaque service utilisé en se basant sur les quotas d'utilisation gratuite, la structure et le fonctionnement actuel de l'application mais également en émettant plusieurs hypothèses sur le comportement moyen d'un utilisateur de l'application. Or, ces hypothèses sont pour la plupart infondées, dans le sans ou il n'y a pas spécialement d'arguments justifiant pourquoi une telle hypothèse à été émise par rapport à une autre.

Cela revient à dire qu'en réévaluant les calculs avec d'autres hypothèses, il est possible d'avoir une estimation des frais par utilisateur complètement différente de celle obtenu. Cependant, comme l'utilisation de l'API de Google Maps est de loin la plus chère et que sans compression ni cache la bande passante utilisée par Cloud Storage reste très importante, il est très probable

que même avec de nouvelles hypothèses ces services restent les services ayant la plus grosse part facturée chaque fois. En revanche, bien que les part restent à peu près équivalentes, il est possible que les nouvelles hypothèses arrivent à la conclusion que les frais par utilisateur sont plus conséquents que ceux estimés dans ce document, ce qui se traduit par une pente plus importante sur les différents graphiques présentés à la fin de la section « **Erreur ! Source du renvoi introuvable.** – Utilisation des services Google ».

Quels autres moyens permettent donc d'estimer les frais d'utilisation ? Premièrement, il est possible de garder le fait d'émettre des hypothèses mais cette fois-ci en s'appuyant de différents arguments expliquant pourquoi une telle hypothèse a été faite. Par exemple, plusieurs questionnaires pourraient être distribués à des parents, des nounous et d'autres personnes afin d'identifier leur mode de vie, le temps d'utilisation moyen passé devant son téléphone, etc. Deuxièmement, il est possible de réaliser une étude à plus petite échelle et d'analyser ensuite l'utilisation moyenne afin d'estimer les frais. Par exemple, il aurait été possible d'étudier l'utilisation des différents services de Google durant la semaine de test. Comme on connait le nombre de participants, on peut alors utiliser ces valeurs pour prédire des frais d'utilisation à plus grande échelle.

Cette méthode semble plus fiable et s'appuie sur des éléments concrets pour estimer les frais engendrés. Pourquoi n'a-t-elle donc pas été utilisée ici ? Il y a trois raisons principales :

- L'analyse des frais a commencé avant le début des tests utilisateurs
- Il y avait peu de participants pour une tel étude et ces derniers ne reflétaient pas forcément le profil type d'un utilisateur, car ils avaient des scénarios précis à réaliser.
- Avec la planification réalisée, il aurait fallu durant la dernière semaine 1) Analyser l'utilisation lors des tests pour estimer les frais, 2) Concevoir ou adapter le modèle pour correspondre à la nouvelle estimation, 3) analyser et interpréter les résultats des questionnaires liés aux tests utilisateurs et 4) peaufiner les détails sur le rapport ou le flyer. En seulement 5 jours, c'est un pari très risqué.

Pour ces raisons, l'analyse des frais ne s'est pas faite en se basant sur l'utilisation des services pendant la semaine de test utilisateur. Avec plus de temps à disposition, il serait tout de même intéressant d'effectuer une telle analyse afin de comparer les estimations obtenues avec celles présentes dans ce document. S'il n'y a pas ou peu de différences alors il y aura deux modèles différents prédisant les mêmes résultats, ce qui est un argument supplémentaire. Si en revanche il y a de grandes différences avec les prédictions, alors des analyses plus poussées devront être effectuées.

4.4 Améliorations apportées par les tests utilisateurs

Les résultats des questionnaires et les remarques laissés par les utilisateurs ont permis de mettre en avant les fonctionnalités le moins bien implémentées ou encore celles qui devrait être ajoutées en priorité. Ces résultats viennent confirmer les idées d'améliorations déjà envisagées au préalable.

Pour chaque fonctionnalité (implémentée ou non), vous trouverez sur les pages suivantes une liste des améliorations futures possibles.

Ouverture de l'application :

- Lors de la première ouverture de l'application, des aides sur le fonctionnement de l'application doivent s'afficher, par exemple en mettant en évidence les différents éléments présents sur l'écran les uns après les autres avec des phrases courtes expliquant l'utilité de l'élément en question [59] :

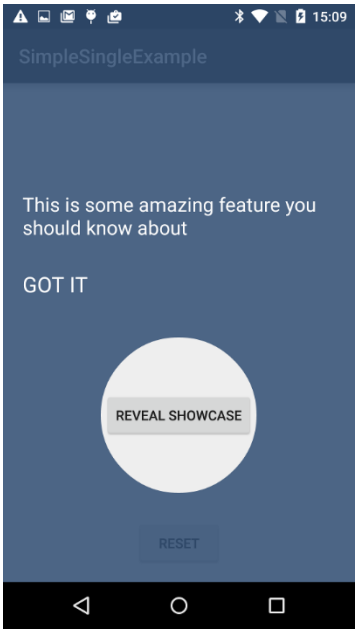

Figure 45 - Exemple de mise en évidence d'un élément de l'application, repris de [59]

Créer un compte / se connecter :

- Après la création d'un compte, regarder si l'adresse e-mail de la personne est vérifiée et envoyer un mail de vérification si ce n'est pas le cas. Ne pas autoriser la prise de rendez-vous et le passage en mode nounou si l'e-mail de l'utilisateur n'est pas vérifié.
- Renforcer les règles de sécurité minimales pour la création d'un mot de passe
- Proposer une double authentification lors de la connexion (F2A)

Modifier son mot de passe :

- Indiquer plus clairement la méthode permettant de modifier son mot de passe
- Envoyer un mail à l'utilisateur à chaque changement de mot de passe pour l'informer

Remplir ses infos de base :

- Laisser la possibilité à l'utilisateur d'entrer textuellement sa date de naissance
- Ajouter plus de traductions sur les langues disponibles

Remplir ses infos de nounou :

- Limiter le montant désiré maximal
- Donner la possibilité aux nounous de spécifier les jours et heures de la semaine ou elles sont disponibles, et éventuellement d'autres infos (place de jeu, lits, etc.)

Demander un rendez-vous :

- Améliorer le design afin de rendre plus compréhensif les éléments modifiables
- Donner la possibilité de planifier des rendez-vous mensuels, journaliers, etc.

Rechercher une nounou :

- Rechercher et afficher les nounous proches automatiquement lorsque la vue est affichée pour la première fois
- Ajouter une option « Utiliser mon adresse » pour effectuer la recherche
- Optimiser la recherche avancée (voir 4.1)

Afficher les informations détaillées d'un rendez-vous :

- Afficher directement la vue du rendez-vous concerné lors de l'ouverture de l'application par le biais d'une notification

Gérer un rendez-vous :

- Donner la possibilité d'annuler un rendez-vous déjà accepté
- Implémenter la modification du rendez-vous (par le parent et/ou la nounou)
- Lors de rendez-vous récurrents, laisser la possibilité d'annuler 1 occurrence sans pour autant annuler tous les rendez-vous

Communiquer avec les autres utilisateurs :

- Implémenter un système de messagerie permettant aux parents et aux nounous de communiquer directement à travers l'application
- Implémenter le fait de pouvoir commenter/annoter un rendez-vous (visible par les deux utilisateurs)

Authenticité des nounous :

- Donner la possibilité aux nounous d'être un « utilisateur vérifié »
- Donner la possibilité aux nounous de pouvoir certifier leur statut professionnel (p.ex : scan des documents et affichage possible lors de la consultation du profil)
- Implémenter un système de *feedback* permettant aux parents d'évaluer le service proposé par une nounou après le rendez-vous. L'évaluation moyenne sera visible par les autres utilisateurs sur le profil de la nounou.

Design de l'application :

- Concevoir et implémenter une identité graphique pour l'application
- Utiliser la librairie graphique

Les commentaires laissés dans les questionnaires par les utilisateurs parlent principalement du fait de pouvoir annuler un rendez-vous déjà accepté et de pouvoir communiquer avec la nounou ou le parent après la prise de rendez-vous.

4.5 Structure du rapport

Enfin, un dernier point de discussion concerne l'organisation de ce document. En effet, le document suit une structure particulièrement adaptée à la publication d'articles scientifiques. Cette structure est très utile quand il s'agit de répondre à une question ou encore de prouver ou du moins d'appuyer les preuves d'une théorie en particulier. Cependant, ce projet, lui, consiste à l'étude et à l'implémentation d'une application d'une application mobile. Il n'y a donc pas vraiment de liens entre une étude scientifique et le rapport de ce projet.

Bien-sûr, l'on retrouve des éléments communs : Dans les deux cas, il est nécessaire d'analyser et d'étudier ce qui a déjà été produit auparavant mais également de concevoir différents modèles qui servent à expliquer la logique ou les méthodes de fonctionnement de notre étude. Cependant, avec le développement d'une application mobile, certains aspects se retrouvent dans plusieurs sections. Prenons par exemple le modèle économique : tout d'abord, il est nécessaire d'analyser toutes les sources générant des frais afin d'estimer les coûts de déploiement et de maintenance de l'application. Une fois l'analyse effectuée, il est alors possible de concevoir un modèle économique qui, en théorie, couvre les frais générés par l'application. Enfin, il est nécessaire d'implémenter les différentes fonctionnalités requises par le modèle économique afin de pouvoir réellement commencer à générer des entrées d'argent. En plus de toutes ces étapes, des discussions peuvent également être apportées à posteriori afin de réévaluer le modèle économique réalisé jusqu'à présent.

Nous remarquons donc qu'avec la structure choisie pour le rapport, les différentes parties expliquant le modèle économique se retrouvent séparés à travers les différentes sections de ce projet. C'est également le cas pour les tests utilisateurs ou pour les objectifs d'implémentation.

Or, le fait de séparer chaque objectif en différentes parties et de mélanger entre-elles toutes ces différents objectifs ne facilitent pas la lecture de ce document. En effet, lorsque les résultats des tests utilisateurs sont présentés, il est nécessaire de connaître les modèles utilisés par les questionnaires et la structure des questions. Si ces informations viennent d'êtres lues par le lecteur, il y a de grandes chances que ce dernier se souvienne de la définition et de l'utilité des modèles utilisés. En revanche, si l'information a été donnée 40 pages en arrière, on peut s'attendre à ce que le lecteur ne se souvienne plus parfaitement du rôle et du fonctionnement des différents modèles.

Pour éviter ce problème, une amélioration pourrait être faite sur le rapport en abandonnant la structure scientifique et en proposant une structure plus adaptée à ce rapport. Par exemple, le document pourrait être divisé en 6 sections principales : Introduction, application mobile, modèle économique, tests utilisateurs, conclusion et références. Pour chaque section, les phase d'analyse, de conception, de résultat et de discussion seront séparées en différentes sous-sections, par exemple :

2. Application mobile 2.1. Analyse (prototype, structure Flutter, données Firebase) 2.2. Conception (Flutter, Firebase) 2.3. Résultats (rapport couvrant l'implémentation)	3. Modèle économique 3.1. Analyse (frais) 3.2. Conception (plan économique permettant de générer un bénéfice théorique) 3.3. Discussion (réévaluation)

5 Conclusion

En conclusion, ce projet de Bachelor permet de montrer sous différents angles l'intérêt et les possibilités offertes à la réalisation et au déploiement d'une application mobile de mise en relation entre parents et nounous. Ce projet permet également de se rendre compte de la diversité des éléments à prendre en considération et étudier pour déployer sur le marché une telle application. Ce projet de Bachelor offre également une bonne « maquette » à présenter aux potentiels entrepreneurs intéressés du fait des différents domaines étudiés. En effet, une étude sur le plan économique à été réalisée en concevant un modèle économique, mais également sur le plan ergonomique avec des tests utilisateurs et une interprétation des résultats, ou encore sur le plan technique avec une application mobile crosse-plateforme fonctionnelle.

En comparaison avec le projet de semestre précédent, ce projet de Bachelor montre également qu'avec une bonne anticipation de la charge de travail nécessaire à la réalisation des objectifs et une planification adaptée, il est tout à fait possible de réaliser un travail conséquent tout en respectant les délais et les objectifs principaux fixés. En effet, les objectifs du projet précédents ayant été sous-évalués, tous n'avaient pas pu être terminés avant les échéances de rendu.

Sur le plan personnel, ce projet de Bachelor m'a permis de renforcer mes connaissances sur de multiples environnements/outils de travails. Tout d'abord, il y a bien sûr Flutter et Firebase dont je maîtrise encore mieux l'utilisation car ce sont les outils utilisés pour réaliser l'application. Mais ce projet m'a également permis de renforcer mes connaissances sur l'utilisation des autres fonctionnalités offertes par la console Google Cloud, tels que l'API de Google Maps ou l'analyseur de métriques d'utilisation. Ce projet de semestre m'a aussi fait découvrir certaines fonctionnalités d'Apple, en particulier de l'App Store Connect et de TestFlight, nécessaire au déploiement sur iOS de l'application. J'ai également découvert Stripe pour l'intégration de paiements, Google Forms, etc.

J'ai beaucoup apprécié réaliser ce travail de Bachelor. Le défi de proposer le développement d'une application aussi complexe et utilisant beaucoup de fonctionnalités diverses m'a vraiment mis à l'épreuve durant ces dernières semaines. Néanmoins, si je devais ressortir un point négatif, j'ai trouvé que le nombre de tâches à réaliser dans des domaines parfois très éloignés les uns des autres m'a donné l'impression de réaliser « une version courte de plusieurs projets de Bachelor différents ». Ce que je veux dire par là, c'est que chaque domaine étudié pourrait selon moi, si l'étude était poussée jusqu'au bout, faire l'objet d'un projet de Bachelor à part entière (pas forcément dans l'informatique). Par exemple, le fait de développer une solution permettant de rechercher de manière optimisée des utilisateurs en fonctions des coordonnées géographiques et d'autres champs facultatifs (fourchette de prix, statut, etc.) le tout en utilisant un système de gestion des données noSQL pourrait, à mon avis et si l'étude est poussée jusqu'au bout, faire l'objet d'un projet de Bachelor. Or, dans ce projet, ce n'est qu'une fonctionnalité parmi bien d'autres.

Avec la démocratisation des SaaS tels que Google Cloud Console et l'utilisation de plus en plus fréquente de Flutter pour le développement d'application mobile, j'espère que les outils et les technologies apprises durant la réalisation de ce projet de Bachelor me seront utiles dans ma vie professionnelle future.

5.1 Remerciements

Je tiens à remercier particulièrement l'enseignant Pascal Bruegger pour avoir accepté mon idée d'application en tant que projet de semestre et projet de Bachelor, mais également pour m'avoir suivi durant sa réalisation.

Je tiens également à remercier un collaborateur et assisant pour ce projet, Arton Hoxha, pour avoir pris le temps de répondre à certaines questions techniques au sujet de Flutter ou de Firebase. Son aide m'a été précieuse et m'a permis d'économiser de perdre trop de temps sur un problème.

Je remercie également les experts assignés sur ce projet, Monsieur Sylvain Egger et Monsieur Dominique Marmy pour avoir participé activement aux séances réalisées en leur compagnie. Les idées évoquées et la vision plus professionnelles apportée m'ont été d'une grande aide.

Je tiens aussi à remercier mon ancienne supérieure Rebecca Beiro qui m'a donnée l'idée il y a de cela 5 ans de réaliser cette application, ce qui paraissait pour moi très difficile à réaliser avec mes connaissances de l'époque.

Enfin, je tiens à remercier tous les participants des tests utilisateurs réalisés. Sans eux, je n'aurais pas pu mesurer à quel point l'application répondait à la demande et il aurait été plus difficile d'évaluer les vrais besoins des utilisateurs. Les commentaires ont également apporté d'autres idées d'amélioration.

5.2 Déclaration d'honneur

Je, soussigné, Cédric Mujynya, déclare sur l'honneur que le travail rendu est le fruit d'un travail personnel. Je certifie ne pas avoir eu recours au plagiat ou à toutes autres formes de fraudes. Toutes les sources d'information utilisées et les citations d'auteur ont été clairement mentionnées.

Cédric Mujynya

6 Références

[1] La Liberté, ATS. 2022. « Près de 70 000 places de crèches créées ». Consulté le 24.05.2022. https://www.laliberte.ch/news/suisse/pres-de-70-000-places-de-creches-creees-636864

[2] Office fédéral de la statistique. OFS. 2021. « Activité professionnelle, tâches domestiques et familiales ». Consulté le 24.05.2022. https://www.bfs.admin.ch/bfs/fr/home/statistiques/population/familles/activite-professionnelle-taches-domestiques-familiales.html

[3] Yoopies by Worklife. 2022. Site internet de l'application Yoopies. Consulté le 25.05.2022. https://yoopies.ch/

[4] Topnanny. 2022. Site internet de l'application Topnanny. Consulté le 25.05.2022. https://topnanny.ch/fr

[5] Google Developers. 2022. Page d'accueil du Framework Flutter. Consulté le 25.05.2022. https://flutter.dev/

[6] Google Developers. 2022. Produits proposés par Firebase. Consulté le 25.05.2022. https://firebase.google.com/products-build

[7] Cédric Mujynya. 2022. « HappyBaby – projet de semestre 6 – rapport de projet ». Consulté le 31.05.2022. https://gitlab.forge.hefr.ch/cedric.mujynya/happybaby

[8] The Bloc Community. 2022. Bloc, a predictable state management library for Dart. Consulté le 31.05.2022. https://bloclibrary.dev/#/

[9] Documentation Firebase. 2022. « Obtenez des mises à jour en temps réel avec Cloud Firestore ». Consulté le 31.05.2022. https://firebase.google.com/docs/firestore/query-data/listen

[10] Documentation Firebase. 2022. « Geo queries ». Consulté le 31.05.2022. https://firebase.google.com/docs/firestore/solutions/geoqueries

[11] Noeatsleepteam.com. 2022. GeoFlutterFire – Flutter packages. Consulté le 31.05.2022. https://pub.dev/packages/geoflutterfire

[12] Google Developers. 2022. Tarifs d'utilisation de Firebase. Consulté le 24.06.2022. https://firebase.google.com/pricing

[13] Google Developers. 2022. Tarifs d'utilisation de Google Maps. Consulté le 24.06.2022. https://mapsplatform.google.com/intl/fr/pricing/

[14] Documentation Firebase. 2022.« Calculs de la taille de stockage ». Consulté le 10.07.2022. https://firebase.google.com/docs/firestore/storage-size

[15] Google Cloud. 2022. Introduction à Cloud Monitoring. Consulté le 10.07.2022. https://cloud.google.com/monitoring

[16] BuildFire. 2022. « How to Perform User Testing for your App ». Consulté le 30.06.2022. https://buildfire.com/how-to-perform-user-testing-for-your-app/

[17] ARKA Softwares. 2022. « Usability Testing of Mobile App ». Consulté le 30.06.2022. https://www.arkasoftwares.com/blog/mobile-app-usability-testing/

[18] Neha Bharati. 2022. « How to Perform Usability Testing for Mobile Apps ». Consulté le 30.06.2022. https://www.browserstack.com/guide/usability-testing-for-mobile-apps

[19] Jackob Nielsen, Thomas K. Landauer. 1993. « A mathematical model of the finding of usability problems ». *INTERCHI93 – Conference on Human Factors in Computing*, p. 206-213. Consulté le 30.06.2022. https://dl.acm.org/doi/10.1145/169059.169166

[20] Carine Lallemand, Guillaume Gronier. 2015. « Méthodes de design UX ». p. 352-364. Consulté le 30.06.2022. https://tecfa.unige.ch/tecfa/maltt/ergo/articles/P3/echelles_utilisabilite_(Lallemand2016).pdf

[21] Anastacia Valdespino. 2020. « After-Scenario Questionnaire ». Consulté le 30.06.2022. https://help.qualaroo.com/hc/en-us/articles/360039070552-After-Scenario-Questionnaire-ASQ-

[22] UX Hints. 2022. « SEQ – Single Ease Question ». Consulté le 30.06.2022. https://uxhints.com/user-testing/ux-metric-single-ease-question-seq/

[23] U.S. Services Administration. 2022. « System Usability Scale ». Consulté le 06.07.2022. https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

[24] Aaron Bangor, Philip Kortum, James Miller. 2009. « Determining what individual SUS scores mean: adding an adjective rating scale ». *Journal of Usability Studies v.4*, p. 114-123. Consulté le 07.07.2022. https://dl.acm.org/doi/10.5555/2835587.2835589

[25] MeasuringU. 2022. « SUPR-Qm™ Package ». Consulté le 06.07.2022. https://measuringu.com/product/supr-qm/

[26] Qualitrics. 2022. « Guide du Net Promoter Score ». Consulté le 06.07.2022. https://www.qualtrics.com/fr/gestion-de-l-experience/client/nps/

[27] NICE Satmetrix. 2020. « Net Promoter Benchmarks ». Consulté le 06.07.2022. https://www.satmetrix.com/wp-content/uploads/2020/07/net-promoter-score-benchmarks.pdf

[28] Inés Roldós. 2021. « What is a Good NPS Score? ». Consulté le 06.07.2022. https://monkeylearn.com/blog/what-is-a-good-nps-score/

[29] Andreas Hinderks, Martin Schrepp, Jörg Thomaschewski. 2018. Page d'accueil d'UEQ. Consulté le 07.07.2022. https://www.ueq-online.org/

[30] Carine Lallemand. 2015. « Questionnaire UX AttrakDiff version française ». Consulté le 07.07.2022. https://carinelallemand.files.wordpress.com/2015/09/version-franc3a7aise-attrakdiff_lallemand_2015.pdf

[31] Walter Takashi Nakamura, Iftekhar Ahmed, David Redmiles, Edson Oliveira,3 David Fernandes, Elaine H. T. de Oliveira, Tayana Conte. 2021. « Are UX Evaluation Methods Providing the Same Big Picture? ». Consulté le 07.07.2022. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8156257/

[32] Guillaume Gronier. 2019. « Échelles d'utilisabilité et UX ». Consulté le 08.07.2022. http://www.guillaumegronier.com/cv/resources/EchellesUtilisabiliteUX_v1.pdf

[33] Documentation Firebase. 2022. « Que puis-je faire avec Cloud Functions ? ». Consulté le 21.06.2022. https://firebase.google.com/docs/functions/use-cases

[34] Google Developers. 2022. Pay – Flutter Package. Consulté le 11.07.2022. https://pub.dev/packages/pay

[35] Aide Google Pay. 2022. Modes de paiement acceptés en Suisse. Consulté le 11.07.2022. https://support.google.com/pay/answer/9095913

[36] Support Apple. 2022. Banques partenaires d'Apple Pay en Europe. Consulté le 11.07.2022. https://support.apple.com/fr-ch/HT206637

[37] Viseca. 2022. Liste des banques et des fournisseurs de cartes. Consulté le 11.07.2022. https://www.viseca.ch/fr/services-numeriques/mobile-payment/liste-des-banques-participantes

[38] Apple Pay. 2022. Fournisseurs de services de paiement (PSP). Consulté le 11.07.2022. https://developer.apple.com/apple-pay/payment-platforms/

[39] Google Pay. 2022. Fournisseurs de services de paiement (PSP). Consulté le 11.07.2022. https://developers.google.com/pay/api#participating-processors

[40] Stripe. 2022. Page d'accueil de Stripe. Consulté le 11.07.2022. https://stripe.com/fr-ch

[41] FlutterStripe.io. 2022. Flutter Stripe – Flutter Package. Consulté le 11.07.2022. https://pub.dev/packages/flutter_stripe

[42] Stripe. 2022. Tarifs d'utilisation de Stripe. Consulté le 11.07.2022. https://stripe.com/fr-ch/pricing

[43] Stripe. 2022. Effectuer des virements avec Stripe Connect. Consulté le 11.07.2022. https://stripe.com/fr-ch/connect/payouts

[44] Google. 2022. Page d'accueil de Google Frorms. Consulté le 30.06.2022. https://www.google.com/intl/fr_ch/forms/about/

[45] PMD. 2022. « An extensible cross-language static code analyzer ». Consulté le 09.06.2022. https://pmd.github.io/

[46] Documentation Firebase. 2022. « Structurer les règles de sécurité Cloud Firestore ». Consulté le 09.06.2022. https://firebase.google.com/docs/firestore/security/rules-structure

[47] Yuli. 2022. Uuid – Flutter Package. Consulté le 09.06.2022. https://pub.dev/packages/uuid

[48] Documentation Firebase. 2022. « Ré-authentifier un utilisateur ». Consulté le 09.06.2022. https://firebase.google.com/docs/auth/web/manage-users#re-authenticate_a_user

[49] Documentation Firebase. 2022. « Types d'index dans Cloud Firestore ». Consulté le 09.06.2022. https://firebase.google.com/docs/firestore/query-data/index-overview

[50] Google Developers. 2022. Firebase Messaging – Flutter Package. Consulté le 21.06.2022. https://pub.dev/packages/firebase_messaging

[51] Documentation Firebase. 2022. « Envoi de messages en amont sur Android (FCM) ». Consulté le 21.06.2022. https://firebase.google.com/docs/cloud-messaging/android/upstream

[52] Documentation Firebase. 2022. « Autoriser les demandes d'envoi (FCM) ». Consulté le 21.06.2022. https://firebase.google.com/docs/cloud-messaging/auth-server

[53] Documentation Firebase. 2022. « Que puis-je faire avec Cloud Functions ? ». Consulté le 24.06.2022. https://firebase.google.com/docs/functions/use-cases

[54] MeasuringU. 2018. « Using Task Ease (SEQ) to Predict Completion Rates and Times ». Consulté le 12.07.2022. https://measuringu.com/seq-prediction/

[55] OpenStreetMap. 2022. A propos d'OSM. Consulté le 13.07.2022. https://www.openstreetmap.org/about

[56] Grzegorz Tomicki. 2022. Leaflet.Autocomplete – projet GitHub. Consulté le 13.07.2022. https://github.com/tomik23/Leaflet.Autocomplete

[57] MotisProject. 2022. « Address Autocompletion Based on OpenStreetMap Data ». Consulté le 13.07.2022. https://motis-project.de/docs/api/endpoint/address.html

[58] Baseflow. 2022. Geocodinf – Flutter Package. Consulté le 13.07.2022. https://pub.dev/packages/geocoding

[59] Deno 2390. 2021. Exemple d'image avec Showcase. Consulté le 13.07.2022. https://imhypee.xyz/add-showcase-view-for-new-users-intro-android-studio/

7 Annexes

Vous trouverez en annexe de ce document, dans l'ordre énoncé :

- Les différents scenarios d'utilisation distribués aux utilisateurs durant les tests
- Un exemple parmi les questionnaires portant la réalisation des scénarios
- Le questionnaire portant sur l'application au global