Haute école de gestion
Genève

Développement d'une application d'évaluation de prescription médicale avec Flutter

Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES

par :

John-Michael QUINTO

Conseiller au travail de Bachelor :

Rolf HAURI, Professeur HES

Carouge, le 8 février 2021

Haute École de Gestion de Genève (HEG-GE)

Filière Informatique de Gestion

Hes·so GENÈVE
Haute Ecole Spécialisée de Suisse occidentale

Déclaration

Ce travail de Bachelor est réalisé dans le cadre de l'examen final de la Haute école de gestion de Genève, en vue de l'obtention du titre Bachelor en informatique de gestion.

L'étudiant a envoyé ce document par email à l'adresse remise par son conseiller au travail de Bachelor pour analyse par le logiciel de détection de plagiat URKUND, selon la procédure détaillée à l'URL suivante : https://www.urkund.com .

L'étudiant accepte, le cas échéant, la clause de confidentialité. L'utilisation des conclusions et recommandations formulées dans le travail de Bachelor, sans préjuger de leur valeur, n'engage ni la responsabilité de l'auteur, ni celle du conseiller au travail de Bachelor, du juré et de la HEG.

« J'atteste avoir réalisé seul le présent travail, sans avoir utilisé des sources autres que celles citées dans la bibliographie. »

Fait à Genève, le 7.02.2021

Quinto John-Michael

Remerciements

Je tiens tout d'abord à remercier M. Rolf Hauri pour le suivi et les conseils qu'il a pu apporter durant ce travail de Bachelor, ainsi que pour les cours qu'il a enseigné pendant ma formation à l'HEG.

Je remercie Mabelle Cepeda et Megan Quinto d'avoir lu mon travail et apporté plusieurs commentaires qui m'ont permis de l'améliorer.

J'aimerai également remercier ma famille de m'avoir soutenu tout au long de mes études.

Résumé

La création d'applications de qualité est associée à un effort de développement conséquent, effort qui doit être poursuivi pour la maintenance tant que l'application est en production. Pour le développement mobile, produire une version pour toutes les plateformes existantes est primordial afin de rendre notre application accessible au plus grand public. Cela signifie que l'effort de développement et de maintenance sont multipliés par le nombre de plateforme et environnement de développement.

Ce problème a été résolu il y a quelques années grâce à l'émergence de frameworks cross-platforms, permettant de n'avoir qu'un seul code source à travers toutes les différentes plateformes. Ceci est possible en développant notre application avec un langage qui sera compilé vers un autre afin d'être exécutable indépendamment de la plateforme. Flutter est l'un des frameworks les plus récents et celui dont nous étudierons dans ce travail.

Développé par Google et publié en 2017, Flutter donne la possibilité de développer rapidement des applications performantes et visuellement attirantes. Le langage de programmation utilisé est Dart, qui est lui aussi un produit de Google. Ce dernier est l'une des raisons principales derrière la performance de leurs applications.

Dans le cadre de ce travail de Bachelor, je développerai une application médicale avec Flutter, dont le but est d'identifier les prescriptions inappropriées chez les enfants, afin de les éviter et de réduire le risque d'erreur médicamenteuse. À travers le développement de cet outil, nous pourrons bien comprendre le fonctionnement de ce framework et d'évaluer son utilisation.

Table des matières

Développement d'une application d'évaluation de prescription médicale avec Flutter 1
Déclaration i
Remerciements ii
Résumé iii
Table des matières iv
Liste des tableaux vi
Liste des figures vi
1. Introduction 1
2. Introduction à Flutter 2
2.1 Historique 2
2.2 Positionnement sur le marché par rapport 3
2.2.1 Aux technologies 3
2.2.2 Aux produits proches 4
2.2.3 Aux recherches des développeurs 4
2.2.4 Aux produits « d'avenir » 6
3. Fonctionnement de Flutter 7
3.1 Architecture 7
3.2 Widgets 8
3.3 Widget Lifecycle Events 11
3.4 Widgets Stateless 11
4. Développer avec Flutter 13
4.1 Environnements de développement 13
4.1.1 Android Studio 13
4.2 Outils 15
4.2.1 Flutter SDK 15
4.2.2 Flutter doctor 15
4.2.3 Emulateur Android / IOS 16
4.3 Documentation 17
4.3.1 Documentation in code 17
4.3.2 Développeurs venant d'une autre plateforme 18
4.3.3 Repo GitHub 19
4.4 Langage 19
4.5 Processus du code source au store 19
4.5.1 Google Play Store 20
4.5.2 App Store 23
5. Développer en Dart 24

5.1 Introduction **24**
5.2 Le Dart en bref/Essentiel du Dart **24**
5.2.1 Dart Web (JavaScript) 24
5.2.2 Dart Native 25
6. Présentation de l'application **26**
6.1 Fonctionnalités de haut-niveau **26**
6.1.1 Créer un compte 27
6.1.2 Se connecter 28
6.1.3 Réinitialiser son mot de passe 29
6.1.4 Modifier ses informations générales 30
6.1.5 Ajouter / supprimer un diplôme 31
6.1.6 Créer un cas pour un nouveau patient 32
6.1.7 Modifier / supprimer un cas 33
6.2 Autres fonctionnalités **34**
6.2.1 Affichage des cas 34
6.2.2 Affichage des pathologies 35
6.2.3 Affichage des prescriptions 36
6.2.4 Affichage du menu compte 37
6.2.5 Affichage des crédits de l'application 38
7. Développement de l'application **39**
7.1 Difficultés et solutions **39**
7.2 État actuel **39**
7.3 Rapport de tests **40**
7.4 Améliorations futures **43**
8. Analyse et conclusion **44**
Bibliographie **45**

Liste des tableaux

Tableau 1 : Définition des widgets 9
Tableau 2 : StatefulWidget Lifecycle Events 13

Liste des figures

Figure 1 : Frameworks cross-platform utilisés par les développeurs dans le monde entier en 2019 et 2020 4
Figure 2 : Frameworks, librairies et outils les plus appréciés en 2019 5
Figure 3 : Frameworks, librairies et outils les plus appréciés en 2020 5
Figure 4 : Frameworks, librairies et outils les plus recherchés en 2019 6
Figure 5 : Frameworks, librairies et outils les plus recherchés en 2020 6
Figure 6 : Architecture du framework Flutter 7
Figure 7 : Application affichant “Hello world” 8
Figure 8 : Hiérarchie des widgets d’une application de base 9
Figure 9 : Hiérarchie des widgets d’une application de base 10
Figure 10 : Lifecycle du StatelessWidget 11
Figure 11 : Structure de base d’un StatefulWidget 12
Figure 12 : Lifecycle d’un StatefulWidget 12
Figure 13 : Logo d’Android Studio 13
Figure 14 : Logo d’IntelliJ 14
Figure 15 : Logo de Visual Studio Code 14
Figure 16 : Résultat du diagnostic par Flutter doctor 15
Figure 17 : Fenêtre d’AVD Manager 16
Figure 18 : Fenêtre “Manage Devices” 17
Figure 19 : Documentation du widget “Navigator” 18
Figure 20 : Documentation pour développeurs expérimentés 18
Figure 21 : Résultat de recherche sur le repo de Flutter 19
Figure 22: Dossier contenant les fichiers pour l’icône de l’application 20
Figure 23 : Fichier “AndroidManifest.xml 20
Figure 24 : Création du keystore 21
Figure 25 : Référence du keystore dans l’application 21
Figure 26 : Configuration du fichier “build.gradle” 21
Figure 27 : Vérification du fichier “build.gradle” 22
Figure 28 : Construction de l’App bundle / APK 22
Figure 29 : Dossier contenant les fichiers pour l’icône de l’application 23
Figure 30 : Création du “build archive” 23
Figure 31 : Logo Dart 24
Figure 32 : Compilateur Dart 25
Figure 33 : Diagramme de use-case de l’application POPI 26
Figure 34 : Ecrans de création de compte 27
Figure 35 : Ecran de connexion 28
Figure 36 : Ecran de réinitialisation de mot de passe 29
Figure 37 : Ecran de modification des informations générales 30
Figure 38 : Ecran de modification des diplômes 31
Figure 39 : Ecran d’ajout d’un nouveau cas 32
Figure 40 : Ecran de modification d’un cas 33
Figure 41 : Ecran d’affichage des cas 34
Figure 42 : Ecran d’affichage des pathologies 35
Figure 43 : Ecran d’affichage des prescriptions 36
Figure 44 : Ecran du menu du compte 37

Figure 45 : Ecran des crédits de l'application 38
Figure 46 : Détail manquant dans l'application 39

1. Introduction

Aujourd'hui, la conception d'une application mobile se fait de plusieurs manières. La plus classique étant celle dite native, qui permet d'utiliser l'intégralité des capacités du système d'exploitation et d'obtenir de meilleures performances. Cependant, un énorme désavantage de cette dernière est la duplication du code pour la rendre disponible sur les deux grandes plateformes (Android et IOS), et les autres existantes mais moins utilisées. Ceci va donc multiplier le temps de travail suivant le nombre de plateformes et va engendrer des coûts supplémentaires pour le développement ainsi que la maintenance de l'application.

L'émergence du développement hybride et cross-platform grâce à des frameworks a su répondre à ce problème en permettant aux développeurs de réaliser des applications déployables sur le Play Store de Google et l'App Store d'Apple, tout en ayant un seul code source. Cette nouvelle méthode est donc dorénavant la solution la plus favorisée pour les équipes qui font face aux limites du développement natif.

Le but de ce travail de Bachelor n'est pas d'étudier tous les frameworks existants, mais plutôt d'un en particulier qui se nomme Flutter, ceci est l'un des derniers à avoir vu le jour. Présenté par Google en 2017, de plus en plus de développeurs adoptent cette technologie et compte aujourd'hui plus de 2 millions d'utilisateurs. Il est donc pertinent de s'intéresser par ce dernier et de déterminer la raison derrière ce succès.

L'apprentissage Flutter se fera en deux parties. La première consistera en la recherche de cette nouvelle technologie, c'est-à-dire l'architecture et le fonctionnement de celui-ci, et la deuxième du développement d'une application dans le domaine médical avec Flutter. En effet, le développement permettra la mise en pratique des connaissances acquises lors de la partie recherche.

2. Introduction à Flutter

Avant d'entrer dans le vif du sujet, il est nécessaire de faire une petite introduction à Flutter dans lequel on prendra connaissance de son histoire et de son émergence sur le marché. Par la suite, nous verrons son positionnement sur le marché par rapport aux différentes technologies existantes. Finalement, nous aborderons le sujet de Flutter et de son fonctionnement.

2.1 Historique

Flutter a premièrement été dévoilé en 2015 lors du « Dart Developer Summit » et a été présenté par Eric Seidel, co-fondateur, et qui dirige actuellement l'équipe Flutter chez Google, sous le nom « Sky ». Lors de ce sommet, Eric Seidel a précisé que c'était un projet expérimental et a procédé par effectuer la démo d'une application qu'il a développé en utilisant Sky et publié sur le Play Store d'Android.

Avant de commencer la démo, il a précisé qu'il n'y avait pas de Java dans l'application et que le tout était dessiné en Dart. Ce dernier est un langage de programmation développé par Google, il permet la création d'applications exécutables sur de multiples plateformes. Il a ensuite procédé par montrer plusieurs fonctionnalités dont le défilement rapide et le multitouche sur l'application. La présentation avait donc pour but de montrer le niveau de rapidité, de responsivité, et de performance globale que cette application pouvait atteindre grâce à ce nouveau framework. Il a également montré à quel point il était facile de construire une application avec Sky, ainsi que le niveau de contrôle que celui-ci pouvait avoir sur l'appareil qui la lance. Finalement, les utilisateurs et développeurs ont conclu que Sky avait la meilleure expérience mobile. (NAPOLI 2019, p. XXI)

En mai 2017, Google a publié Sky mais sous un autre nom, Flutter. Cependant, sa première version stable (v.1.0.0) a été annoncé en décembre 2018, lors de l'évènement Flutter Live. Durant un interview avec Eric Seidel à cet évènement, il a expliqué que Flutter avait initialement pour but de rendre le web meilleur, en essayant d'avoir une version plus petite et rapide de Chrome. Le résultat obtenu était donc un navigateur nettement supérieur en termes de rapidité, mais cela n'était pas suffisant. Ils ont donc rajouté plus de fonctionnalités, réécrit le système plusieurs fois, jusqu'à finalement obtenir le framework Flutter. De plus, en réécrivant le système avec plusieurs langages, dont Javascript et C++, ils se sont rendu compte que Dart était le plus optimal pour eux. De ce fait, ils ont procédé par n'utiliser que celui-ci. (SEIDEL, 2018)

2.2 Positionnement sur le marché par rapport

2.2.1 Aux technologies

En ce qui concerne la conception des applications mobiles, nous avons plusieurs approches : native, hybride et cross-platform. Comme mentionné précédemment, le développement natif est la conception d'une version de l'application pour chaque plateforme, en utilisant le langage dédié pour celle-ci. Par exemple, pour Android nous avons Java et Kotlin, puis Swift et Objective-C pour IOS. Le principal avantage de cette approche est l'accessibilité au hardware de l'appareil (camera, microphone, GPS, etc.) Flutter ne rentre pas dans cette catégorie, mais plutôt dans le développement cross-platform. (SIDDHARTH, 2020)

Beaucoup de personnes se trompent et pensent qu'hybride et cross-platform ne sont que deux termes différents signifiant la même chose. En effet, leur objectif est plus ou moins pareil, néanmoins on trouve tout de même quelques différences.

La méthode hybride permet de développer des applications qui sont à moitié natives et web. Le code source est écrit avec les langages de développement web (HTML, CSS, et JavaScript) et est ensuite encapsulé dans une application native, comportant son propre navigateur, grâce à des solutions comme le framework Apache Cordova. Notre application ne s'affichera donc plus dans un navigateur normal, mais à travers une application native. Ce dernier fourni une "webview", qui est le composant de visualisation web de chaque plateforme, dans lequel s'affichera notre application web. Celui-ci porte le nom de "WKWebView" pour iOS et "WebView" pour Android. En utilisant les frameworks, nous pouvons bénéficier de plugins qui nous permettent de dépasser les limites d'un navigateur simple, en ayant accès à certaines fonctionnalités du téléphone (camera, géolocalisation, etc.). Cependant, l'accès reste encore limité et le niveau de performance est inférieur à celui des applications natives. (GRIFFITH)

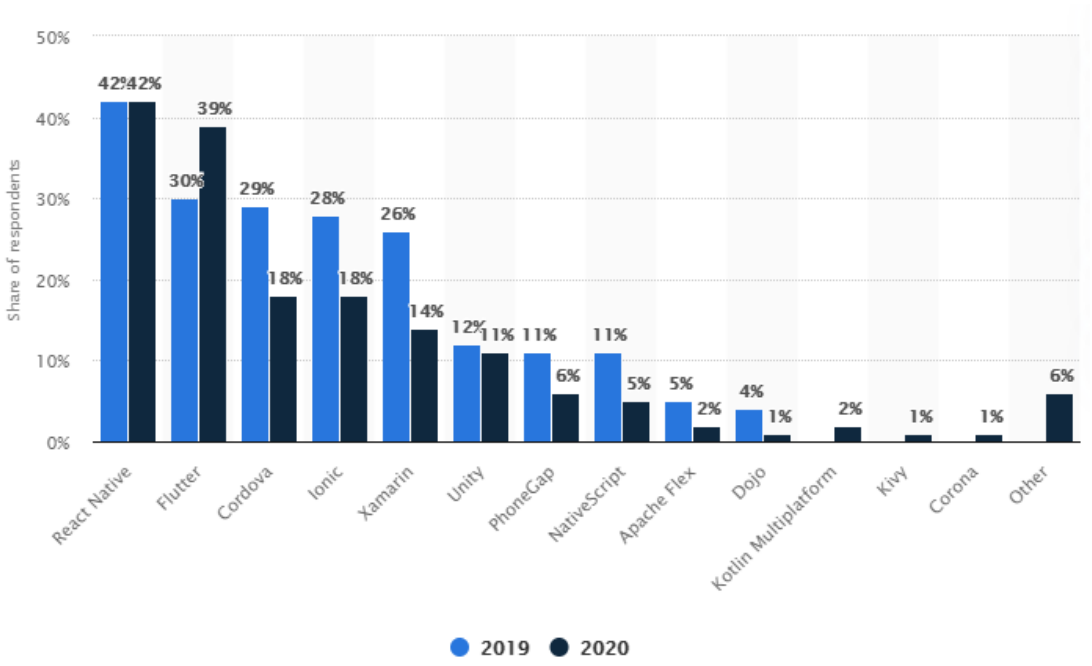
Finalement, nous avons la dernière méthode qui est le développement cross-platform et qui concerne donc Flutter. Contrairement à la méthode hybride, celle-ci permet d'utiliser des composants natifs afin de donner à l'utilisateur une meilleure expérience. Nous avons un seul code source, mais chaque plateforme peut avoir sa partie spécifique. Même si les applications natives restent supérieures, la performance des applications cross-platform s'en rapproche tout en ayant comme principal avantage le gain de temps et d'argent. (KIDECHA, 2020)

2.2.2 Aux produits proches

Flutter est plus récent que ses concurrents majeurs, ce qui signifie qu'il est moins mature et est encore en train de se développer. En revanche, il a été conçu en prenant en compte les faiblesses de ses concurrents, ce qui lui donne un avantage.

Le graphique ci-dessous représente la répartition de l'utilisation des frameworks en 2019 et 2020, selon un sondage effectué auprès de 19'696 développeurs en juin 2020.

Figure 1 : Frameworks cross-platform utilisés par les développeurs dans le monde entier en 2019 et 2020



(Statista, 2020)

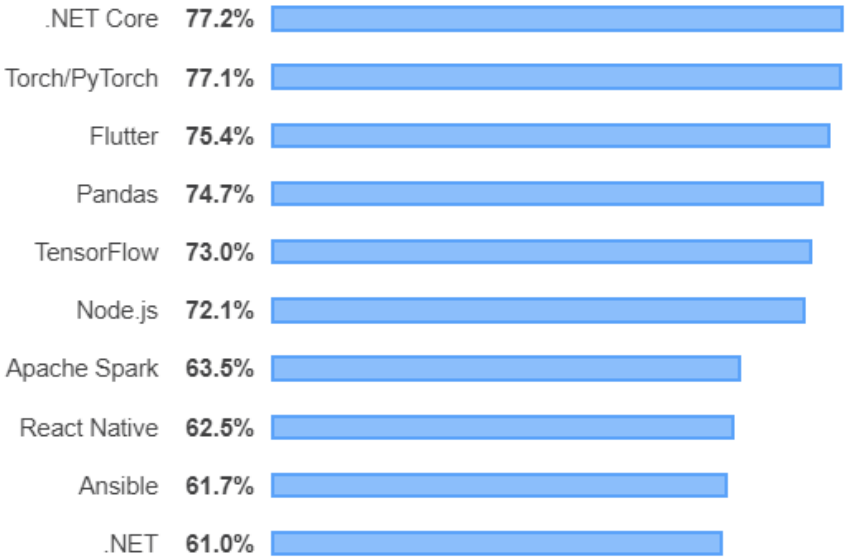
On constate que malgré le fait que la première version stable de Flutter soit apparue en décembre 2018, il se retrouve en deuxième place en 2019 et 2020. Ses principaux concurrents sont :

- React Native
- Cordova
- Ionic
- Xamarin

2.2.3 Aux recherches des développeurs

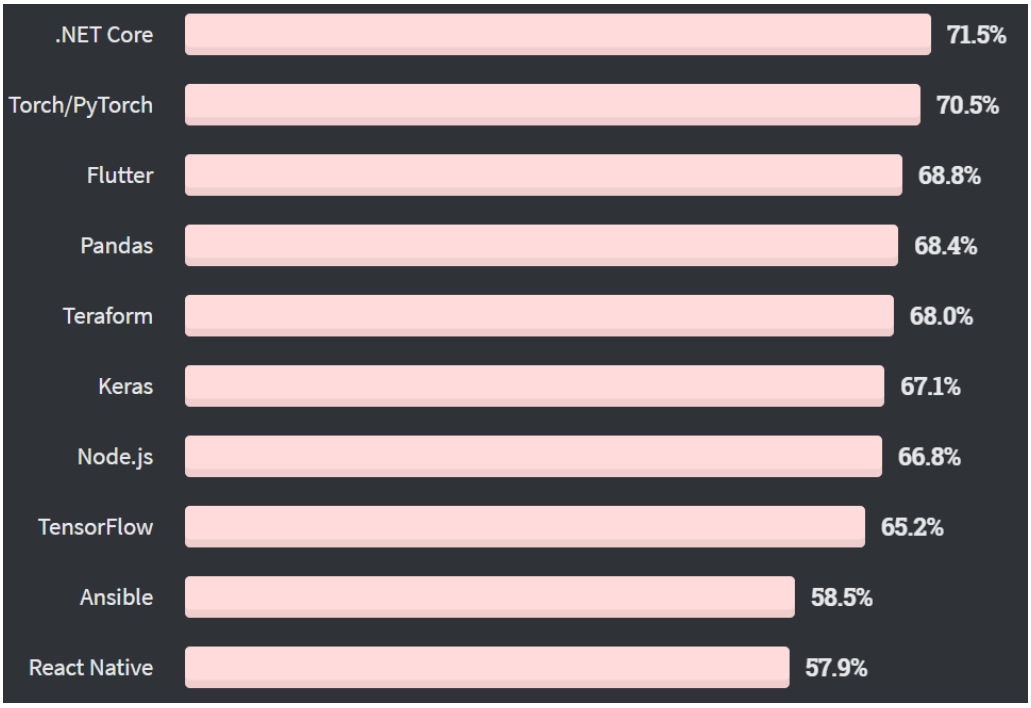
Après avoir étudié les frameworks les plus utilisés, il serait maintenant utile de voir les avis des développeurs. Stack Overflow réalise chaque année un sondage auprès des développeurs couvrant tout, de leurs technologies préférées à leurs préférences professionnelles. Les deux graphiques ci-dessous représentent les frameworks les plus appréciés en 2019 et 2020 :

Figure 2 : Frameworks, librairies et outils les plus appréciés en 2019



(Stack Overflow, 2019)

Figure 3 : Frameworks, librairies et outils les plus appréciés en 2020

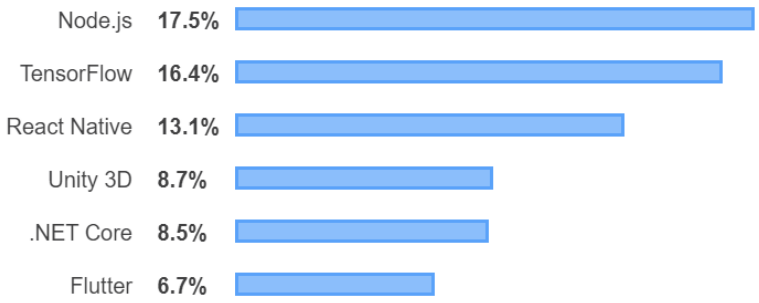


(Stack Overflow, 2020)

Selon les graphiques présentés, il est évident que Flutter est un framework qui plaît aux développeurs. En effet, il occupe la troisième place pour les deux années avec un taux d'appréciation de 75.4% en 2019 et 68.8% en 2020.

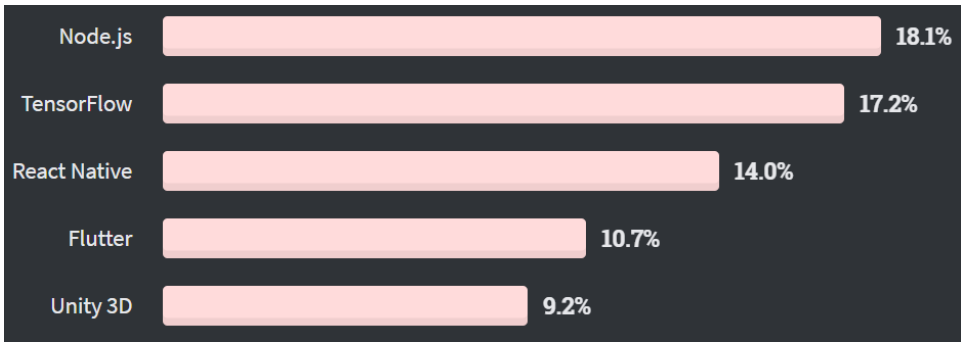
Les deux graphiques suivants représentent la part des développeurs n'utilisant pas Flutter mais qui sont intéressés par ce dernier.

Figure 4 : Frameworks, librairies et outils les plus recherchés en 2019



(Stack Overflow, 2019)

Figure 5 : Frameworks, librairies et outils les plus recherchés en 2020



(Stack Overflow, 2020)

D'après ces graphiques, les développeurs sont de plus en plus intéressés par Flutter. En 2019, le taux était de 6.7% seulement contre 10.7% en 2020. Ce n'est qu'une augmentation de 4%, mais qui est beaucoup plus significative que ses concurrents dont la plupart ont connu une augmentation inférieure à 1%.

2.2.4 Aux produits « d'avenir »

En 2015, Google a créé le terme PWA (Progressive Web Apps). Les PWAs sont des applications web qui prennent en compte certains avantages des applications natives, comme leur accessibilité depuis une icône se trouvant dans le téléphone sans avoir à écrire l'URL dans un navigateur. Ceci enlève également la nécessité de les télécharger depuis un store (App ou Google Play) et d'effectuer les mises à jour régulières, ce qui est un avantage pour les utilisateurs n'ayant peu ou pas d'espace de stockage. Vu que ce sont des applications web, cela signifie qu'elles sont accessibles sur toutes les plateformes et leur développement est plus simple car nous utilisons les langages HTML, CSS et JavaScript, qui sont connus par la plupart des développeurs. Malgré cela, elles ne peuvent pas encore accéder à toutes les fonctionnalités du téléphone comme les applications natives le font, surtout sur IOS. (Topflight, 2020)

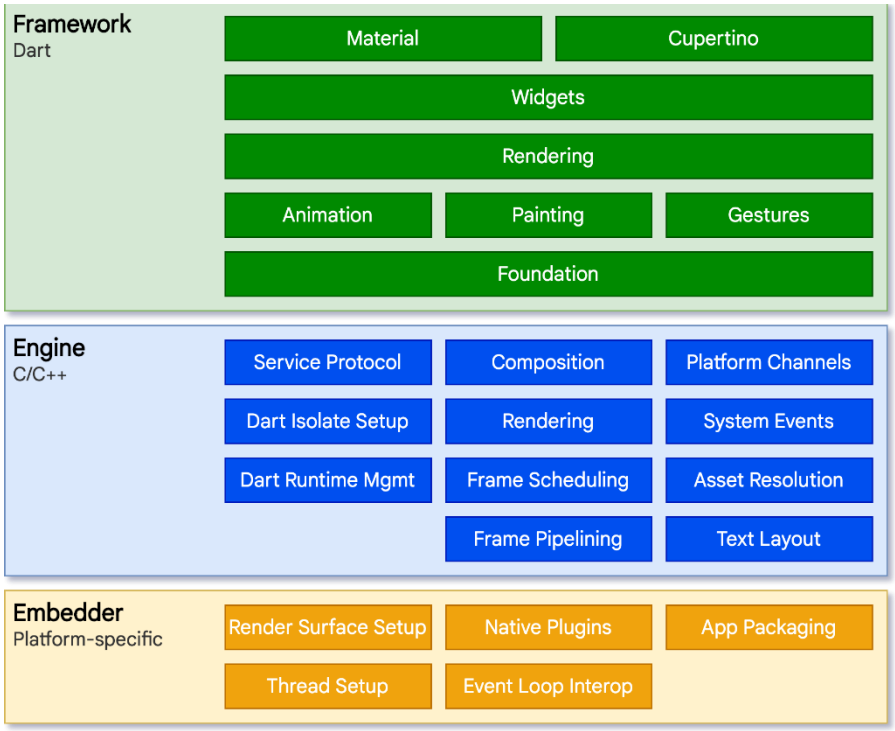
3. Fonctionnement de Flutter

Flutter est un framework avec lequel nous pouvons créer des applications de haute performance tout en permettant la réutilisabilité de son code pour les déployer sur différentes plateformes tels que mobile et desktop.

Le concept principal de Flutter est qu'on considère tout comme étant un « widget ». En effet, une phrase que l'on entend souvent en parlant de Flutter est « Everything is a widget ». Les widgets sont des composants d'interface qui sont utilisés pour créer l'interface utilisateur d'une application. Ils peuvent être comparés à des briques LEGO que nous rassemblons pour créer une structure plus grande, et en ajoutant différents types de briques, nous modifions l'apparence et le comportement de celui-ci. Une application créée avec Flutter est donc un ensemble de widgets personnalisés qui ont été combinés. Il existe énormément de widgets, y compris ceux imitant l'apparence IOS. (NAPOLI 2019, p. 5)

3.1 Architecture

Figure 6 : Architecture du framework Flutter



(https://flutter.dev/docs/resources/architectural-overview)

La figure ci-dessus représente l'architecture de Flutter qui est divisé en trois parties. La première partie est le framework, écrit dans le langage Dart, dans lequel nous retrouvons tous les composants que nous pouvons intégrer dans une application Flutter. Il est donc possible qu'une application ne contienne pas tous les éléments de cette couche.

La couche du bas, appelé « Embedder », est spécifique à chaque plateforme et communique avec l'OS. Ainsi, il nous permet d'accéder aux fonctionnalités du système natif. Ce dernier est écrit en Java et C++ pour Android, Objective-C / Objective-C++ pour IOS et MacOS, puis C++ pour Windows et Linux.

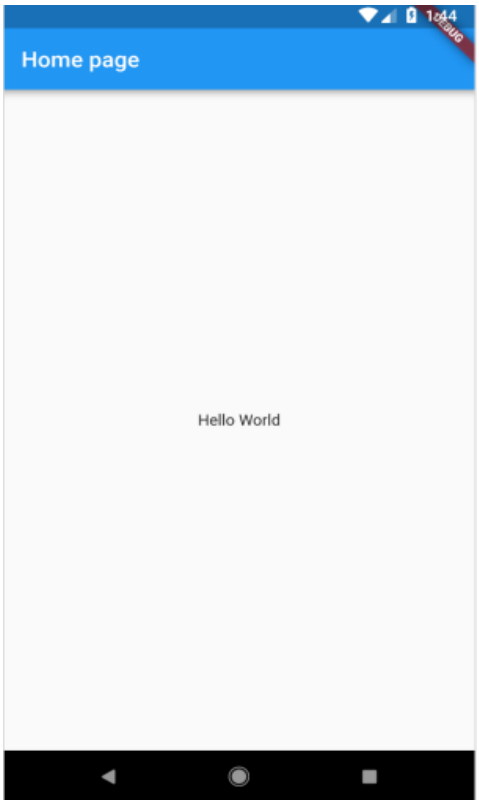
Finalement nous retrouvons le Flutter « Engine » qui est au cœur du Framework. Ce dernier, principalement écrit en C++, s'occupe de rastériser (transformer une image décrite en un format vectoriel en une série de pixels afin de créer une image) à chaque fois qu'un écran doit être « dessiné ». Il implémente les librairies principales de Flutter, y compris les animations et graphiques (à travers Skia), la mise en page du texte, les entrées et sorties de fichiers et de réseau, ainsi que la chaîne d'outils pour l'exécution et la compilation de Dart. (Flutter)

Skia est une librairie graphique 2D open source fournissant des API qui marchent sur diverses plateformes. Elle est utilisée par Google Chrome, Chrome OS, Android, Mozilla Firefox, Firefox OS et plein d'autres produits. (Skia)

3.2 Widgets

Afin de mieux comprendre le concept des widgets, nous décomposerons une application simple en widgets, puis nous analyserons comment ces derniers sont structurés dans le code source.

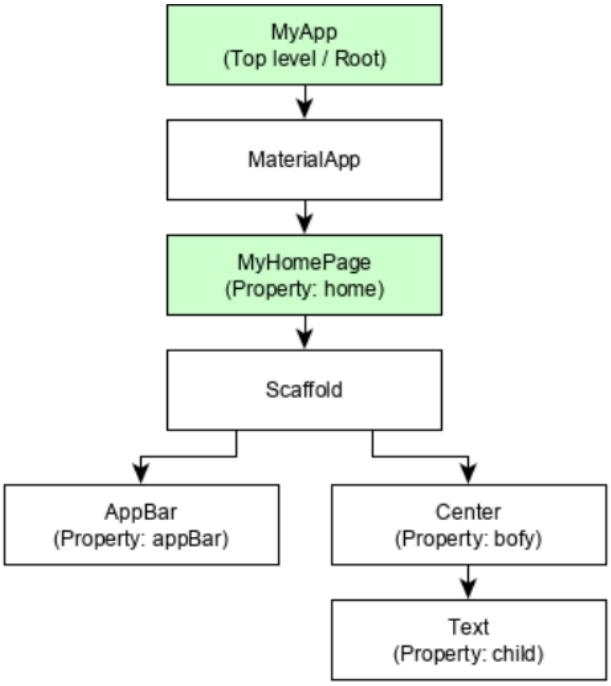
Figure 7 : Application affichant "Hello world"



(Tutorials Point 2019, p. 11)

Voici l'application que nous étudierons. Elle a pour seul but d'afficher un text « Hello world ». Ce que nous pouvons constater est que le style utilisé est le Material Design développé par Google en 2014.

Figure 8 : Hiérarchie des widgets d'une application de base



(Tutorials Point 2019, p. 12)

Le diagramme ci-dessus représente la hiérarchie des widgets pour notre application. Chaque case est un widget et chaque flèche indique quel autre widget il enveloppe. Nous avons donc MyApp, le widget de plus haut niveau, qui enveloppe MaterialApp, et ainsi de suite. Scaffold, quant à lui, contient deux widgets, dont Appbar et Center. L'ensemble de ces widgets forment le « Widget Tree ». Le tableau suivant donne une explication pour chaque widget :

Tableau 1 : Définition des widgets

Widget	**Définition**
MyApp	Widget qui est appelé lors du lancement de l'application
MaterialApp	Classe prédéfinie permettant d'accéder aux widgets de Flutter
MyHomePage	Widget qui représente une page de l'application
Scaffold	Widget qui fournit un cadre implémentant la structure de mise en page visuelle pour le Material design
AppBar	Widget qui définit l'entête de l'interface utilisateur
Center	Widget qui permet de centrer les widgets à l'intérieur
Text	Widget qui permet d'afficher du texte

Inspiré de (Tutorials Point 2019, p.13)

Figure 9 : Hiérarchie des widgets d'une application de base

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Hello World Demo Application',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(title: 'Home page'),
    );
  }
}

class MyHomePage extends StatelessWidget {
  MyHomePage({Key key, this.title}) : super(key: key);

  final String title;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(this.title),
      ),
      body: Center(
        child:
            Text(
              'Hello World',
            )
      ),
    );
  }
}
```

(Tutorials Point 2019, p. 31)

Le code présenté ci-dessus est celui qui se trouve dans le fichier « main.dart » qui est le point d'entrée du projet. Au démarrage de l'application, la méthode « main » est appelé et va ensuite lancer « runApp », qui prend en paramètre notre widget de plus haut niveau s'intitulant MyApp. Finalement, runApp attachera notre widget à l'écran de notre appareil.

Nous remarquons plusieurs éléments :

- Les widgets que nous créons doivent étendre de la classe « StatelessWidget » (ou « StatefulWidget »)
- Les widgets que nous créons doivent contenir une méthode « build », prenant en paramètre un « BuildContext », qui s'occupe de leur instanciation dans le « widget tree »
- La méthode build retourne un autre widget dont on définit les propriétés
- MaterialApp possède trois propriétés :
  - Title : titre de l'application
  - Theme : styles de l'application
  - Home : indique quel widget utiliser en tant que page d'accueil
- Scaffold possède deux propriétés :

- AppBar : l'en-tête de l'application
- Body : corps de l'application. Il contient un « child », qui est le widget de type « Text », où se trouve notre « Hello world »

Après avoir étudié comment une application Flutter gère son interface utilisateur, nous comprenons mieux pourquoi la phrase « Everything is a widget » résume bien ce framework.

3.3 Widget Lifecycle Events

La notion de « Lifecycle Events » correspond aux différents évènements qui surviennent au cours du cycle de vie d'un widget, comme par exemple lorsqu'il a été créé ou modifié. Ces événements nous permettent de définir le comportement du widget au moment où ils se déclenchent.

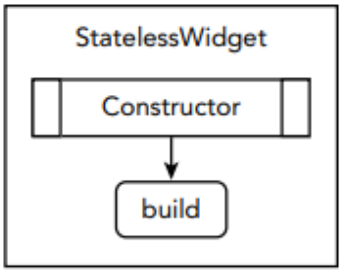
Dans Flutter, nous avons deux types de widgets pour créer l'interface utilisateur : les StatelessWidgets et les StatefulWidget. Comme leur nom le suggère, la différence principale entre ces deux est la gestion d'états. En effet, nous utilisons un StatelessWidget lorsque l'état de notre widget ne change pas. À l'inverse, un StatefulWidget nous permet de gérer les changements d'états de notre widget. Nous pouvons donc nous attendre à ce que les StatefulWidgets aient un cycle de vie plus complexe.

La méthode commune aux deux types de widget est « build » qui s'occupe de l'emplacement du widget dans le Widget Tree. Ce dernier prend en paramètre un BuildContext qui est un objet « Element ». La différence avec un widget est qu'un Element est une instanciation d'un widget dans l'arbre. (NAPOLI 2019, p. 5-6)

3.4 Widgets Stateless

Les StatelessWidgets ne change pas dynamiquement. Prenons par exemple notre application qui affiche un texte « Hello World ». L'écran n'affichera que ce texte et ceci ne changera pas. La figure suivante représente le lifecycle du StatelessWidget.

Figure 10 : Lifecycle du StatelessWidget



(Beginning Flutter 2020, p. 6)

Sa méthode « build » ne peut qu'être appelé à trois scénarios différents : lorsque le widget est créé, si un widget parent ou un « InheritedWidget » change. Un InheritedWidget est un type de widget permettant de passer des paramètres à tous ses descendants.

3.4.1.1 Widgets Stateful

Les StatefulWidgets, quant à eux, peuvent changer dynamiquement. Prenons par exemple, un écran qui affiche une liste de produits. L'utilisateur peut ajouter ou enlever des produits dans cette liste et nous observerons notre liste changer en fonction des actions de l'utilisateur.

Pour déclarer un widget de type « Stateful », nous devons le faire avec les deux classes suivantes : « StatefulWidget » et « State ». La classe State va contenir toutes les configurations qui peuvent changer au fil du temps. Le StatefulWidget sera reconstruit à chaque fois que le State subira des changements afin d'afficher le nouvel état.

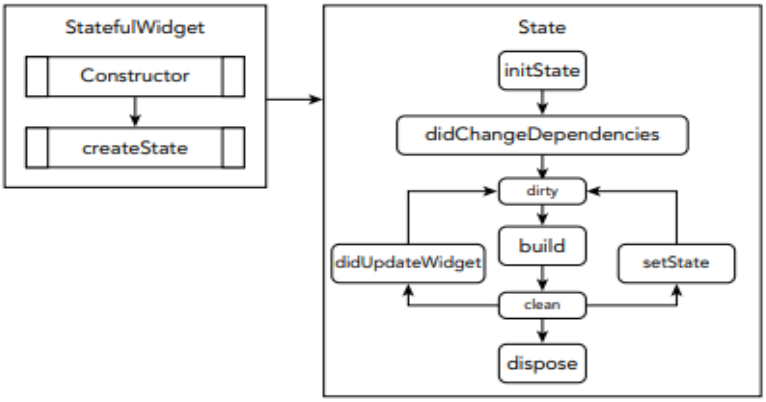
Figure 11 : Structure de base d'un StatefulWidget

```
class JournalEdit extends StatefulWidget {
  @override
  _ JournalEditState createState() => _ JournalEditState();

}

class _JournalEditState extends State<JournalEdit> {
  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

(Beginning Flutter 2020, p. 6-7)

L'extrait de code ci-dessus représente la déclaration d'un StatefulWidget avec son State. Le StatefulWidget « JournalEdit » utilise sa méthode « createState » pour créer l'état muable « _JournalEditState » qui lui est associé.

Figure 12 : Lifecycle d'un StatefulWidget



(Beginning Flutter 2020, p. 7)

La figure ci-dessus nous montre que contrairement au StatelessWidget, il est possible de manipuler notre widget à plusieurs moments de son cycle de vie. Le tableau ci-dessous décrit ces différents moments.

Tableau 2 : StatefulWidget Lifecycle Events

Méthode	**Description**
InitState	Appelé lorsque l'objet est inséré dans le Widget Tree
Dispose	Appelé lorsque l'objet est enlevé du Widget Tree de façon permanente
DidChangeDependencies	Appelé lorsque State change
DidUpdateWidget	Appelé lorsque la configuration du widget change
Deactivate	Appelé lorsque l'objet est enlevé du Widget Tree
SetState	Notifie le framework que l'état de l'objet a changé. Face à ce changement, la méthode build doit être appelé

Inspiré de (Beginning Flutter 2020, p. 7-8)

4. Développer avec Flutter

Lorsque l'on envisage d'utiliser un nouveau framework, il est important de connaître les environnements de développement, les outils à notre disposition, mais également si la documentation contient le nécessaire pour intégrer toutes les fonctionnalités de notre application. En bref, savoir si le framework nous permettra de construire et de publier notre application sans qu'on ait de problèmes pendant le développement.

4.1 Environnements de développement

Avant de commencer à développer avec Flutter, il faut tout d'abord procéder par l'installation des prérequis, dont l'environnement de développement.

4.1.1 Android Studio

Figure 13 : Logo d'Android Studio

(https://developer.android.com/studio)

D'après le site officiel de Flutter, l'installation d'Android Studio est nécessaire car Flutter dépend d'Android Studio pour récupérer les dépendances d'Android. C'est également l'IDE (Integrated Development Environment), ou environnement de développement, officiel pour les applications Android. Son utilisation est gratuite et marche également pour concevoir des applications IOS avec Flutter, ce qui fait de lui un très bon choix. Toutefois, nous ne sommes pas dans l'obligation de l'utiliser pour développer nos applications Flutter.

4.1.1.1 IntelliJ

Figure 14 : Logo d'IntelliJ

(https://hownot2code.files.wordpress.com/2019/01/intellij-519-d4ff21c469.png)

Tout comme Android Studio, IntelliJ est un IDE pouvant être utilisé pour le développement d'applications Flutter. Pour cela, il suffit d'installer les plugins Flutter et Dart, dont l'installation prend 5 minutes. Il existe une version gratuite et payante, mais dans notre cas, la version gratuite est suffisante. Il a été développé par JetBrains et est exécutable sur Windows, Linux et MacOS.

4.1.1.2 Visual Studio Code

Figure 15 : Logo de Visual Studio Code

(https://commons.wikimedia.org/wiki/File:Visual_Studio_Code_1.35_icon.svg)

Visual Studio Code est un éditeur de texte gratuit créé par Microsoft et qui est conçu pour Windows, Linux et MacOS. Par rapport à ses alternatives, il est plus léger, c'est-à-dire qu'il s'exécute plus rapidement et utilise moins de mémoire. L'installation des plugins Flutter et Dart prend moins de 5 minutes également.

4.2 Outils

4.2.1 Flutter SDK

Le Flutter SDK (Software Development Kit) contient tous les packages et outils de ligne de commande indispensables pour développer des applications Flutter sur toutes les plateformes. Celui-ci est disponible par téléchargement sur le site officiel : https://flutter.dev/docs/get-started/install ou par Git, en faisant un clone depuis le Flutter repo sur GitHub, si l'on ne souhaite pas installer une version fixe.

4.2.2 Flutter doctor

« Flutter doctor » est également inclus dans le Flutter SDK et, comme son nom l'indique, va vérifier et diagnostiquer ce qui est bon ou non dans notre environnement de développement Flutter. Il fournira un résumé contenant étapes à suivre afin de réparer les erreurs, comme le montre la figure ci-dessous.

Figure 16 : Résultat du diagnostic par Flutter doctor

```
C:\Users\venka>flutter doctor

    WARNING: your installation of Flutter is 55 days old.

    To update to the latest version, run "flutter upgrade".



                    Welcome to Flutter! - https://flutter.io

    The Flutter tool anonymously reports feature usage statistics and crash
    reports to Google in order to help Google contribute improvements to
    Flutter over time.

    Read about data we send with crash reports:
    https://github.com/flutter/flutter/wiki/Flutter-CLI-crash-reporting

    See Google's privacy policy:
    https://www.google.com/intl/en/policies/privacy/

    Use "flutter config --no-analytics" to disable analytics and crash
    reporting.


Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel beta, v0.5.1, on Microsoft Windows [Version 10.0.17134.165], locale en-US)
[X] Android toolchain - develop for Android devices
    X Unable to locate Android SDK.
      Install Android Studio from: https://developer.android.com/studio/index.html
      On first launch it will assist you in installing the Android SDK components.
      (or visit https://flutter.io/setup/#android-setup for detailed instructions).
      If Android SDK has been installed to a custom location, set $ANDROID_HOME to that location.
[√] Android Studio (version 3.1)
    X Flutter plugin not installed; this adds Flutter specific functionality.
    X Dart plugin not installed; this adds Dart specific functionality.
[!] IntelliJ IDEA Community Edition (version 2018.1)
    X Flutter plugin not installed; this adds Flutter specific functionality.
    X Dart plugin not installed; this adds Dart specific functionality.
[!] VS Code, 64-bit edition (version 1.21.1)
[!] Connected devices
    ! No devices available
```

(CLOW, p. 121)

4.2.3 Emulateur Android / IOS

Les émulateurs sont essentiels pour tester et visualiser nos applications. Un des avantages principaux d'émuler est le coût. En effet, se procurer tous les différents appareils sur lesquels on veut tester notre application nous reviendra beaucoup plus cher que si on les émule.

4.2.3.1 Android

Pour créer un émulateur Android, il existe plusieurs façons de le faire, mais celle qui nous permet d'avoir le plus de contrôle (type d'hardware et version d'Android) est la création avec Android Studio. La figure ci-dessous représente l'AVD (Android Virtual Device) Manager qui affiche les émulateurs que nous avons créé.

Figure 17 : Fenêtre d'AVD Manager

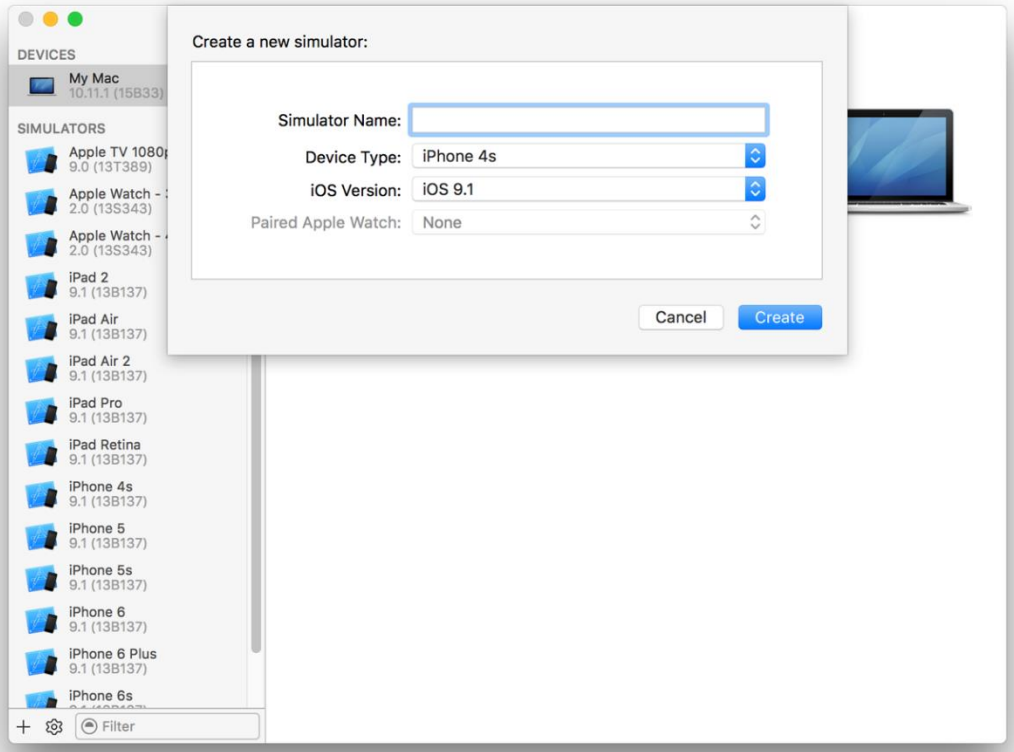
(Learn Google Flutter Fast 65 Example Apps, p. 128)

Une fois que nos émulateurs ont été créés, nous pourront les exécuter à travers Visual Studio Code ou la ligne de commande, sans passer par Android Studio.

4.2.3.2 IOS

Pour créer un émulateur IOS, nous devons utiliser l'IDE d'Apple, XCode. Malheureusement, il est uniquement disponible sur MacOS. Pour tester notre application sur IOS, il faudrait donc être en possession d'un Mac ou d'un appareil IOS. Il est également possible d'utiliser une machine virtuelle et d'exécuter une image IOS. Depuis XCode, il faut lancer le « Simulator » et d'aller sur « Manage Devices ». La figure suivante représente la fenêtre permettant d'afficher et de gérer nos différents émulateurs.

Figure 18 : Fenêtre “Manage Devices”



(https://developer.apple.com/library/archive/documentation/IDEs/Conceptual/iOS_Simulator_Guide/GettingStartedwithiOSSimulator/GettingStartedwithiOSSimulator.html)

4.3 Documentation

Lire la documentation d’un framework est indispensable afin de l’utiliser comme il le faut et d’adapter correctement les composants disponibles à notre code. Ceci permet d’éviter de nombreux problèmes qui sont essentiellement dus aux dépendances inhérentes à la structure du framework et aux éléments constituant l’application.

4.3.1 Documentation in code

La documentation est disponible depuis « flutter.dev/docs ». Cependant, il est possible d’obtenir la documentation de chaque élément directement depuis le code, sans passer par le site officiel. Pour y accéder, tout dépend de notre environnement de développement. Par exemple, avec Visual Studio Code, il suffit de passer la souris par-dessus le widget qui nous intéresse. La figure ci-dessous représente la documentation du widget “Navigator” qui est directement affiché depuis l’IDE.

Figure 19 : Documentation du widget “Navigator”

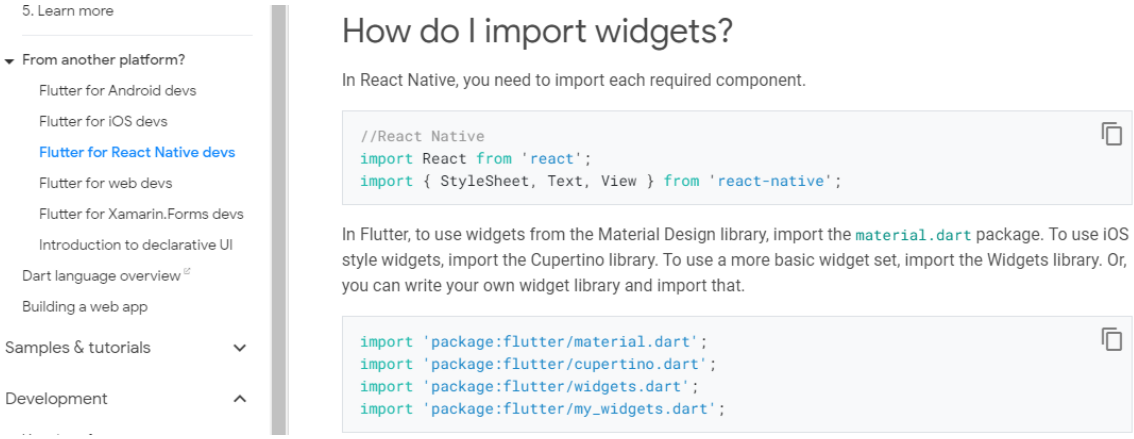
```
class Navigator extends StatefulWidget

package:flutter/src/widgets/navigator.dart

A widget that manages a set of child widgets with a stack discipline.

Many apps have a navigator near the top of their widget hierarchy in order to
display their logical history using an [Overlay] with the most recently visited pages
visually on top of the older pages. Using this pattern lets the navigator visually
transition from one page to another by moving the widgets around in the overlay.
Similarly, the navigator can be used to show a dialog by positioning the dialog
widget above the current page.

## Using the Navigator API

Mobile apps typically reveal their contents via full-screen elements called
"screens" or "pages". In Flutter these elements are called routes and they're
managed by a [Navigator] widget. The navigator manages a stack of [Route]
objects and provides two ways for managing the stack, the declarative API
```

(John-Michael Quinto, 2020)

4.3.2 Développeurs venant d’une autre plateforme

Pour les développeurs ayant déjà de l’expérience avec une autre plateforme de développement mobile, Flutter leur propose une section dans la documentation permettant d’utiliser facilement leur expérience comme point de départ. Pour chaque plateforme, on explique les points communs et les différences, puis comment les implémenter avec Flutter.

Figure 20 : Documentation pour développeurs expérimentés

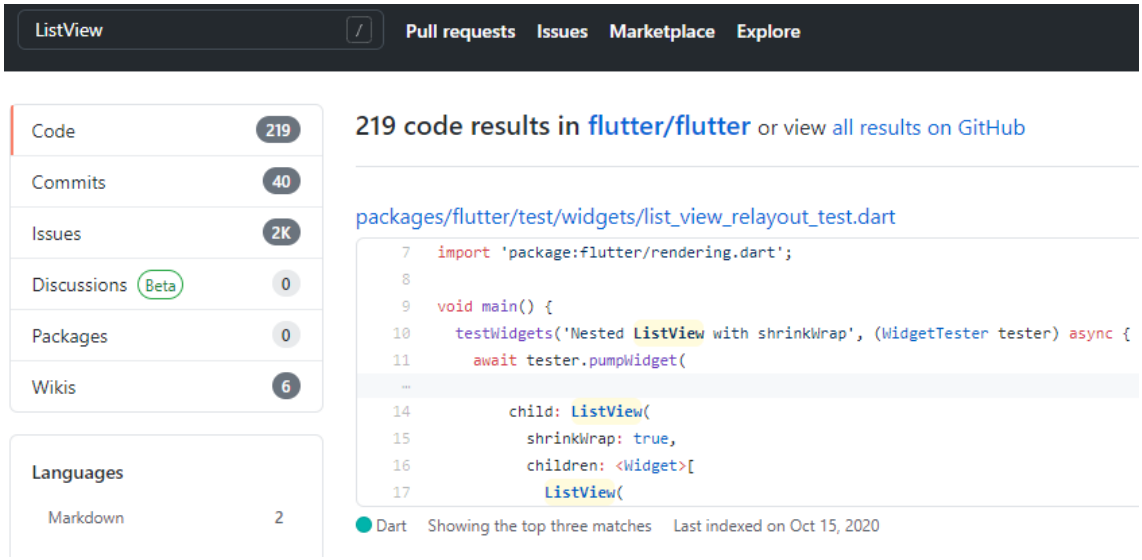


(https://flutter.dev/docs/get-started/flutter-for/react-native-devs)

4.3.3 Repo GitHub

Dans la documentation, il manque parfois un exemple pour avoir une idée de comment implémenter un widget dans notre code. Dans ce cas, le repo de Flutter sur GitHub nous est très utile. En effet, nous pouvons rechercher un widget dans le repo, puis nous retrouverons plusieurs exemples dans lesquels notre widget est utilisé. Dans le cas ci-dessous, le widget recherché est « ListView » (une liste déroulante contenant d'autres widgets), et nous obtenons 219 exemples de code.

Figure 21 : Résultat de recherche sur le repo de Flutter



(https://github.com/flutter/flutter/search?q=ListView)

4.4 Langage

Les applications Flutter sont écrites en Dart. Créé en 2011 par Google, c'est un langage de programmation orienté objet, open-source, adoptant une syntaxe similaire à Java et C#. Alors que ce n'est pas un des langages les plus populaires et qui est très peu utilisé en production, nous pouvons nous demander pourquoi Google l'utilise pour son nouveau framework. Est-ce seulement car Dart lui appartient? Par la suite, nous verrons plus en détail la réponse à cette question.

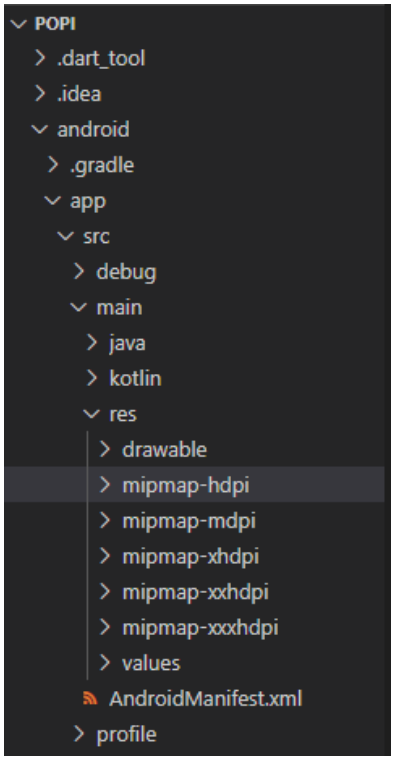
4.5 Processus du code source au store

Pendant la phase de développement, nous testons notre application avec les options « Run » et « Debug » de notre IDE, puis le résultat obtenu est une version « Debug » (test) de celle-ci. Une fois terminée, nous devons préparer la version « Release » (production) qui sera publiée sur le Google Play Store et l'App Store. Mais avant cela, plusieurs choses doivent être mises en place.

4.5.1 Google Play Store

La première étape consiste à créer l'icône de notre application en utilisant le générateur de notre choix. Ce dernier va générer des fichiers nécessaires pour la modification de son apparence et que nous déplacerons dans le dossier « /android/app/src/main/res » de notre projet.

Figure 22: Dossier contenant les fichiers pour l'icône de l'application



(John-Michael Quinto, 2020)

Dans le fichier qui se nomme "AndroidManifest.xml" du dossier « main », il faudra changer « ic_launcher » avec le nom choisi lors de la génération de l'icône. Le champ « android:label » comporte le nom qui figurera en dessous de l'icône.

Figure 23 : Fichier "AndroidManifest.xml

```
    FlutterApplication and put your custom class here. -->
<user-premission android.name="android.permission.INTERNET" />
<application
    android:name="io.flutter.app.FlutterApplication"
    android:label="popi"
    android:icon="@mipmap/ic_launcher">
    <activity
```

(John-Michael Quinto, 2020)

Une fois que ces configurations sont faites, notre icône devrait changer d'apparence. Nous pourrons également voir ce changement sur notre émulateur ou notre appareil Android lors du débogage.

La deuxième étape est la signature de l'application avec le « keystore ». Avant d'effectuer cette étape, il faut s'assurer que Java JDK est installé sur notre machine. Il suffit ensuite d'exécuter les commandes ci-dessous dans notre ligne de commande.

Figure 24 : Création du keystore

If you have an existing keystore, skip to the next step. If not, create one by running the following at the command line:

On Mac/Linux, use the following command:

```
keytool -genkey -v -keystore ~/key.jks -keyalg RSA -keysize 2048 -validity 10000 -alias
```

On Windows, use the following command:

```
keytool -genkey -v -keystore c:\Users\USER_NAME\key.jks -storetype JKS -keyalg RSA -key
```

(https://flutter.dev/docs/deployment/android)

Par la suite, le keystore devra être référencé dans le fichier « key.properties » que nous créerons manuellement dans le dossier « /android ».

Figure 25 : Référence du keystore dans l'application

Create a file named `<app dir>/android/key.properties` that contains a reference to your keystore:

```
storePassword=<password from previous step>
keyPassword=<password from previous step>
keyAlias=key
storeFile=<location of the key store file, such as /Users/<user name>/key.jks>
```

(https://flutter.dev/docs/deployment/android)

La troisième étape est la configuration du fichier « build.gradle » se trouvant dans le dossier « /android/app/ ». Dans cette partie, il faudra simplement copier-coller les bouts de code ci-dessous.

Figure 26 : Configuration du fichier "build.gradle"

```
def keystoreProperties = new Properties()
def keystorePropertiesFile = rootProject.file('key.properties')
if (keystorePropertiesFile.exists()) {
    keystoreProperties.load(new FileInputStream(keystorePropertiesFile))
}

android {
      ...
}
```

(https://flutter.dev/docs/deployment/android)

La quatrième étape est la vérification du fichier « build.gradle ». Il faut s'assurer que les valeurs pour les éléments ci-dessous sont correctes.

Figure 27 : Vérification du fichier "build.gradle"

Review the default Gradle build file file, `build.gradle`, located in `<app dir>/android/app` and verify the values are correct, especially the following values in the `defaultConfig` block:

`applicationId`
Specify the final, unique (Application Id)appid

`versionCode` & `versionName`
Specify the internal app version number, and the version number display string. You can do this by setting the `version` property in the pubspec.yaml file. Consult the version information guidance in the versions documentation.

`minSdkVersion`, `compilesdkVersion`, & `targetSdkVersion`
Specify the minimum API level, the API level on which the app was compiled, and the maximum API level on which the app is designed to run. Consult the API level section in the versions documentation for details. `buildToolsVersion`

Specify the version of Android SDK Build Tools that your app uses. Alternatively, you can use the [Android Gradle Plugin] in Android Studio, which will automatically import the minimum required Build Tools for your app without the need for this property.

(https://flutter.dev/docs/deployment/android)

Une fois que la vérification a été effectuée, nous pouvons passer à l'étape finale, qui est la construction de la version production de l'application. Pour cela, nous avons deux formats possibles: App bundle ou APK. L'App bundle est la version préférée pour le Google Play Store, mais il existe d'autre stores qui ne la prennent pas encore en charge. Voici les commandes à exécuter dans notre ligne de commande pour les construire.

Figure 28 : Construction de l'App bundle / APK

1. Enter `cd <app dir>`
   (Replace `<app dir>` with your application's directory.)
2. Run `flutter build appbundle`
   (Running `flutter build` defaults to a release build.)

The release bundle for your app is created at `<app dir>/build/app/outputs/bundle/release/app.aab`.

1. Enter `cd <app dir>`
   (Replace `<app dir>` with your application's directory.)
2. Run `flutter build apk --split-per-abi`
   (The `flutter build` command defaults to `--release`.)

This command results in three APK files:

- `<app dir>/build/app/outputs/apk/release/app-armeabi-v7a-release.apk`
- `<app dir>/build/app/outputs/apk/release/app-arm64-v8a-release.apk`
- `<app dir>/build/app/outputs/apk/release/app-x86_64-release.apk`

(https://flutter.dev/docs/deployment/android)

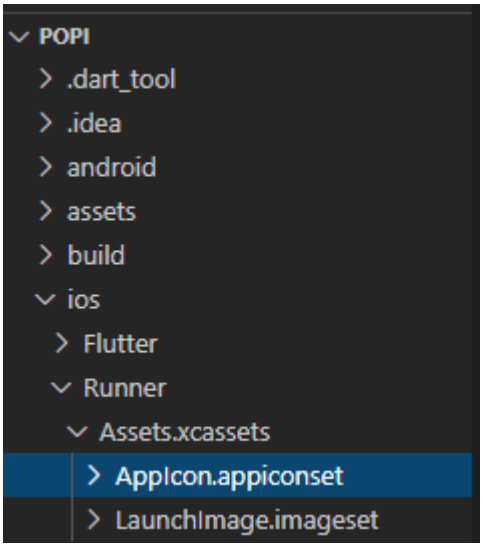
En suivant les instructions sur https://developer.android.com/distribute/best-practices/launch, vous pourrez maintenant déployer votre App bundle / APK sur le Google Play store. (Flutter)

4.5.2 App Store

Pour préparer notre application afin de le publier sur l'App Store, il y a moins de manipulations à faire sur notre projet que pour le Google Play Store. Cependant, nous devons tout d'abord être en possession d'une machine qui tourne sous MacOS et du logiciel XCode.

La première étape est la modification de l'icône et pour cela, nous devons d'abord générer les fichiers nécessaires depuis le générateur de notre choix. Ces fichiers seront ensuite déplacés vers le dossier « /ios/Runner/Assets.xcassets/ » de notre projet.

Figure 29 : Dossier contenant les fichiers pour l'icône de l'application



(John-Michael Quinto, 2021)

Finalement, il restera à créer le « build archive » qui sera téléchargé sur l'App Store Connect. Le « build archive » est la version « Release » de l'application et est créé en exécutant la commande suivante sur notre ligne de commande depuis le répertoire de notre projet.

Figure 30 : Création du "build archive"

On the command line, follow these steps in your application directory:

1. Run `flutter build ios` to create a release build (`flutter build` defaults to `--release`).
2. To ensure that Xcode refreshes the release mode configuration, close and re-open your Xcode workspace. For Xcode 8.3 and later, this step is not required.

(https://flutter.dev/docs/deployment/ios)

Nous pouvons maintenant déployer notre application sur l'App Store en suivant les instructions sur le site : https://developer.apple.com/documentation/xcode/preparing_app_for_distribution. (Flutter)

5. Développer en Dart

Figure 31 : Logo Dart

(https://upload.wikimedia.org/wikipedia/commons/f/fe/Dart_programming_language_logo.svg)

5.1 Introduction

Dart a été publié par Google en 2011, mais il est devenu populaire seulement après l'annonce de son nouveau framework Flutter. Ce dernier, écrit en Dart, a obligé les développeurs d'apprendre le langage afin de l'utiliser. Néanmoins, Dart adopte une syntaxe similaire à celui de C, rendant son apprentissage plus facile pour les développeurs connaissant Java, JavaScript et les autres langages influencés par C. Pour les débutants en programmation, sa prise en main est également simple et rapide.

5.2 Le Dart en bref/Essentiel du Dart

Dart est un langage à usage général, c'est-à-dire qu'il peut être utilisé pour créer des applications web, server, desktop et mobile. Pour développer nos applications Dart, l'installation du Dart SDK est nécessaire (inclus dans le Flutter SDK). Ce dernier contient les commandes pour compiler notre code, ainsi que sa propre VM (machine virtuelle) dans lequel nous pouvons les exécuter.

5.2.1 Dart Web (JavaScript)

Initialement, Dart avait pour but d'être une alternative à JavaScript pour ceux qui ne voulaient plus utiliser ce langage de programmation. Malheureusement, la plupart des navigateurs ne supportaient pas Dart et la plupart ont continué à utiliser JavaScript. Finalement, Google a développé ses propres compilateurs de Dart vers du JavaScript. Désormais, les développeurs peuvent coder en Dart et être certain que leurs applications fonctionneront sur n'importe quel navigateur. (Ionos, 2020)

Lors du développement, le compilateur « dartdevc » est très utile pour ne pas perdre de temps. En effet, il compile seulement les modules qui ont subi des changements au lieu de tous les packages dont notre application dépend. De ce fait, les modifications sont observables presque instantanément.

Dès que notre application est prête à être déployée, le compilateur « dart2js » s'occupera de compiler la totalité de notre code vers du JavaScript. Étonnamment, le résultat produit est plus rapide que si le code était écrit en JavaScript à la base. (COWL, 2019)

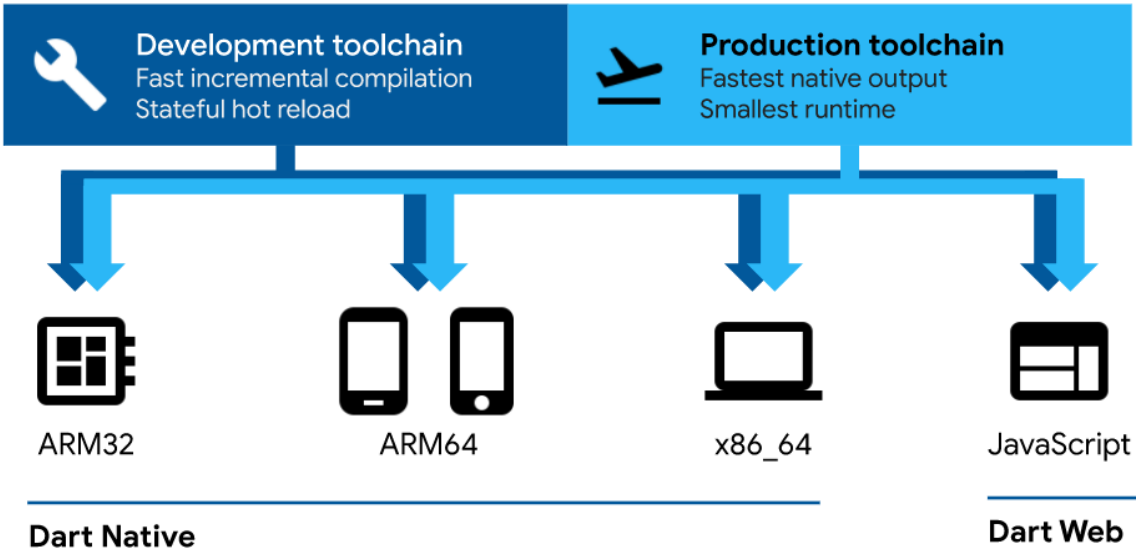
5.2.2 Dart Native

Pour les applications mobiles, desktop et server, Dart VM possède deux types de compilateur: JIT (just-in-time) et AOT (ahead-of-time).

Le JIT est utilisé lors du développement car il compile juste les changements au code source et permet au développeur de coder et tester en même temps. C'est ce qu'on appelle le « Hot Reloading / Restarting » . Pendant cette phase, nous pouvons exécuter et tester notre application depuis le Dart VM, sans qu'il ne communique directement avec l'OS.

À la fin du développement, notre application est compilée vers du code natif ARM ou X64 avec la méthode AOT. Nous aurons donc la version production qui pourra être déployée et lancé de manière native. Le résultat sera une application rapide car le code ne devra plus être interprété d'un langage à un autre pour que notre machine puisse le comprendre. (Dart)

Figure 32 : Compilateur Dart



(https://dart.dev/platforms)

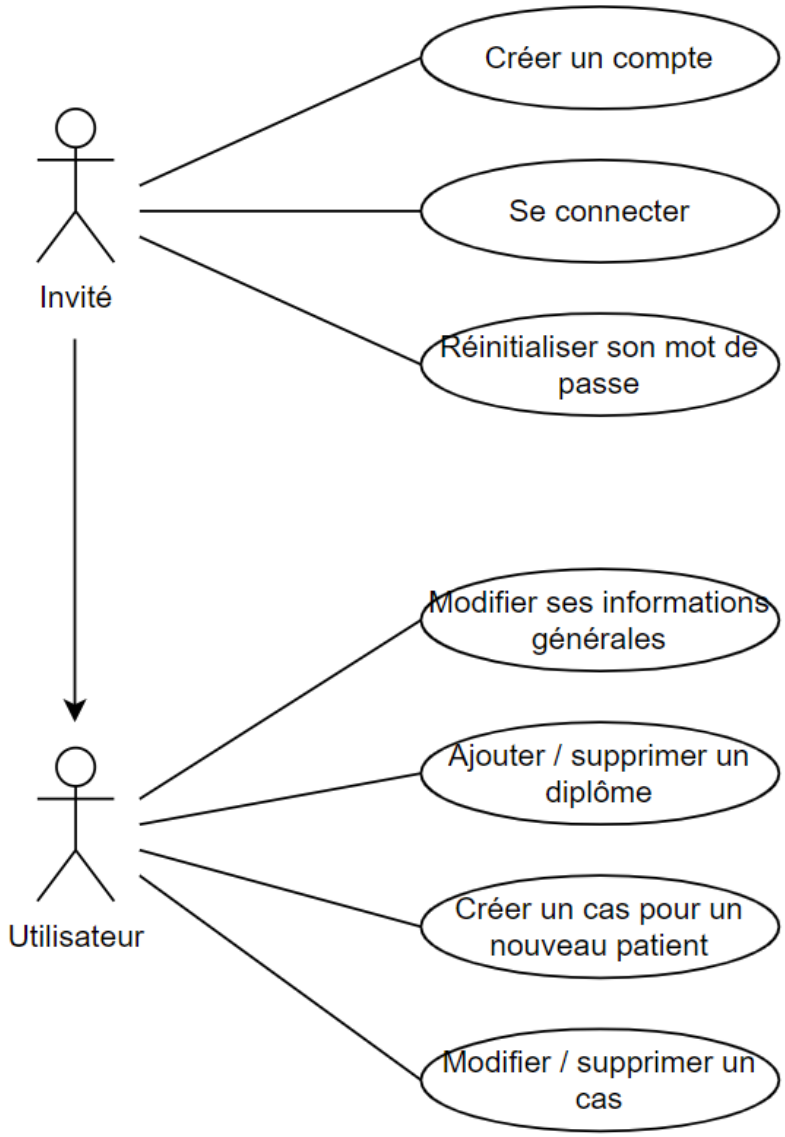
La figure ci-dessus représente les deux types de compilateur : « Development toolchain » correspond au JIT utilisé pendant la phase de développement, et « Production toolchain », l'AOT qui compile notre code afin qu'il soit déployable sur les différentes plateformes. En dessous, nous retrouvons les deux plateformes : Native et Web.

6. Présentation de l'application

L'erreur médicamenteuse est une des causes d'iatrogénie médicamenteuse (ensemble des effets indésirables provoqués par la prise d'un ou plusieurs médicaments) chez les enfants. POPI (pédiatrie: omissions et prescriptions inappropriées) est un outil de d'identification des prescriptions inappropriées chez l'enfant, dont l'objectif serait l'acquisition de réflexes et d'habitudes de prescription afin de réduire les prescriptions inappropriées et les omissions de prescriptions.

6.1 Fonctionnalités de haut-niveau

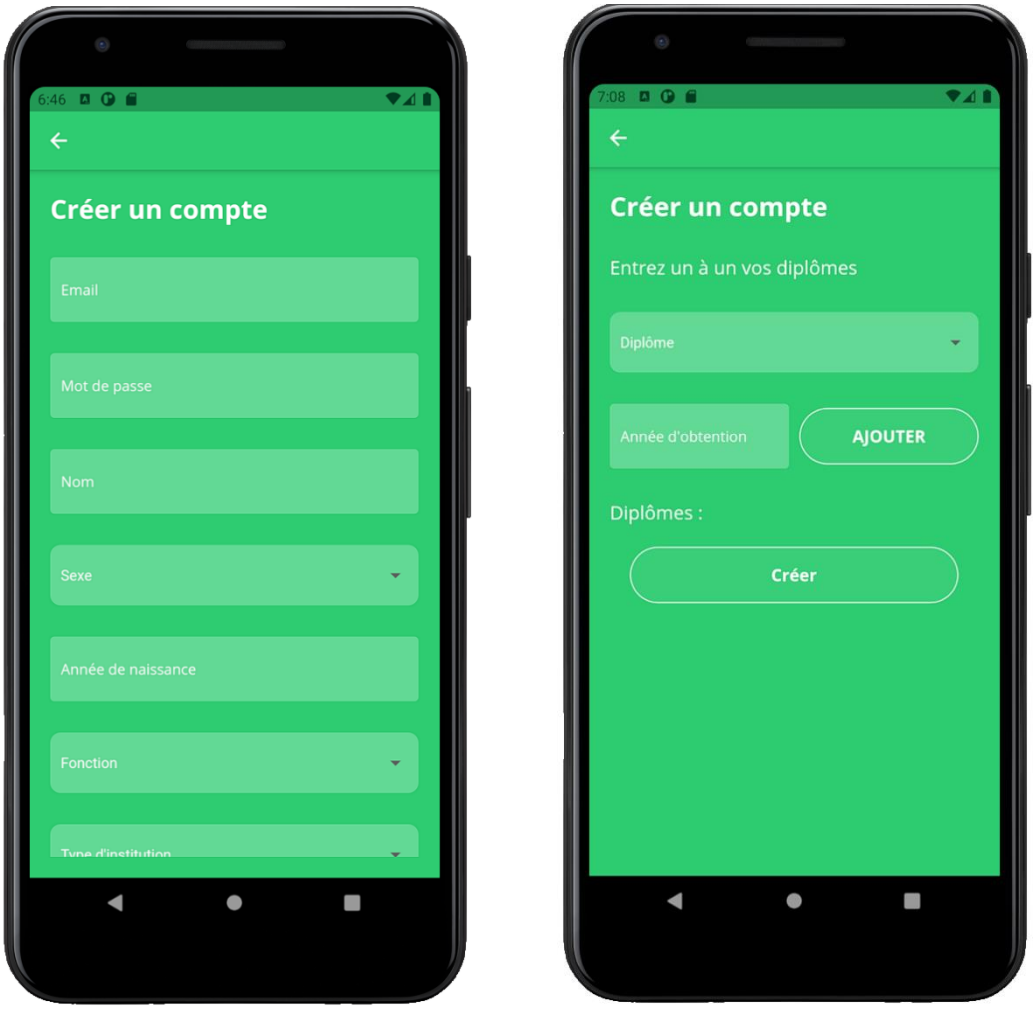
Figure 33 : Diagramme de use-case de l'application POPI



(John-Michael Quinto, 2020)

6.1.1 Créer un compte

Figure 34 : Ecrans de création de compte



(John-Michael Quinto, 2020)

En lançant l'application, l'utilisateur arrive sur la page de connexion. Depuis cette page, il devra appuyer sur le bouton « S'inscrire » pour pouvoir créer un compte. Il atterrira sur la page « Créer un compte », qui figure ci-dessus, et pourra ensuite insérer ses informations et celles liées à son travail. Une fois terminé, il pourra appuyer sur le bouton « Suivant » qui l'emmènera vers la prochaine étape qui consiste à ajouter ses diplômes. Chaque champ ne doit pas être vide et doit respecter le format attendu. Dans le cas contraire, un message d'erreur s'affichera en dessous de chaque champ erroné. Finalement, le bouton « Créer » enregistra le nouveau compte dans la base de données et l'utilisateur sera automatiquement connecté.

6.1.2 Se connecter

Figure 35 : Ecran de connexion

(John-Michael Quinto, 2020)

La page de connexion s'affiche après avoir lancé l'application. En insérant l'email et le mot de passe liés à notre compte et en appuyant sur le bouton « Se connecter », nous serons redirigés vers la page d'accueil qui affiche les cas traités par l'utilisateur. Si les identifiants ne sont pas corrects, un message d'erreur apparaîtra pour informer l'utilisateur que la connexion a échoué.

6.1.3 Réinitialiser son mot de passe

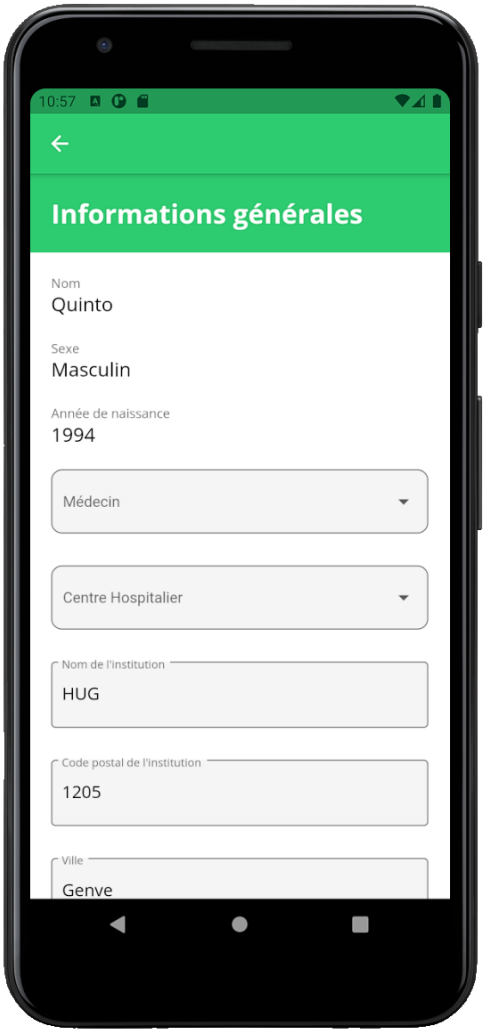
Figure 36 : Ecran de réinitialisation de mot de passe

(John-Michael Quinto, 2020)

Si l'utilisateur a oublié son mot de passe, il peut le réinitialiser depuis cette page qui est accessible en appuyant sur le bouton « Mot de passe oublié ? » de la page de connexion. L'utilisateur doit simplement insérer l'email avec lequel il a créé son compte et appuyer sur le bouton « Valider » pour qu'un email contenant son nouveau mot de passe lui soit envoyé. Il se peut que l'email se retrouve dans les spams, il ne faut donc pas oublier de vérifier ce dossier avant de renvoyer un autre mot de passe. Si aucun compte n'a été créé avec l'email inséré, un message d'erreur s'affichera.

6.1.4 Modifier ses informations générales

Figure 37 : Ecran de modification des informations générales



(John-Michael Quinto, 2020)

Si l'utilisateur le souhaite, il peut modifier toutes les informations qu'il a indiqué lors de la création de son compte, sauf le nom, sexe et année de naissance. Une fois les informations modifiées, le bouton « Valider » enregistre les modifications dans la base de données et affiche un message indiquant le succès de l'opération.

6.1.5 Ajouter / supprimer un diplôme

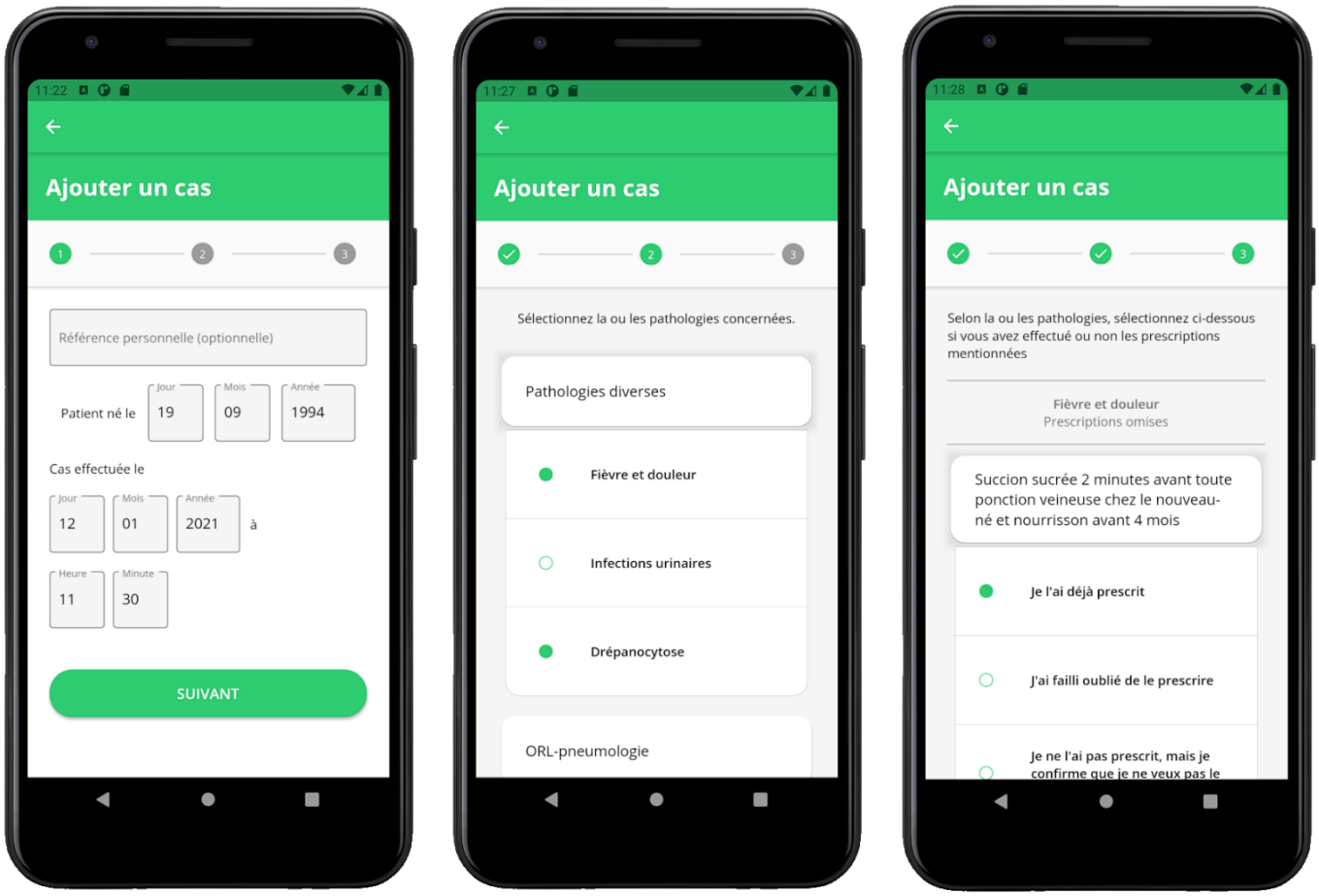
Figure 38 : Ecran de modification des diplômes

(John-Michael Quinto, 2020)

Si l'utilisateur le souhaite, il peut ajouter ou supprimer ses diplômes sur cette page. Lorsqu'un diplôme est ajouté ou supprimé, la base de données est directement mise à jour sans que l'utilisateur n'ait besoin d'appuyer sur le bouton pour confirmer les changements effectués. Pour ajouter un diplôme, il doit sélectionner le type de diplôme depuis la liste déroulante, ou préciser le nom du diplôme dans le cas où l'option choisie est « Autre ». Il doit également indiquer l'année d'obtention du diplôme avant d'appuyer sur « Ajouter » pour l'ajouter à sa liste de diplômes. Pour chaque changement effectué, un message indiquant le succès de l'opération est affiché.

6.1.6 Créer un cas pour un nouveau patient

Figure 39 : Ecran d'ajout d'un nouveau cas

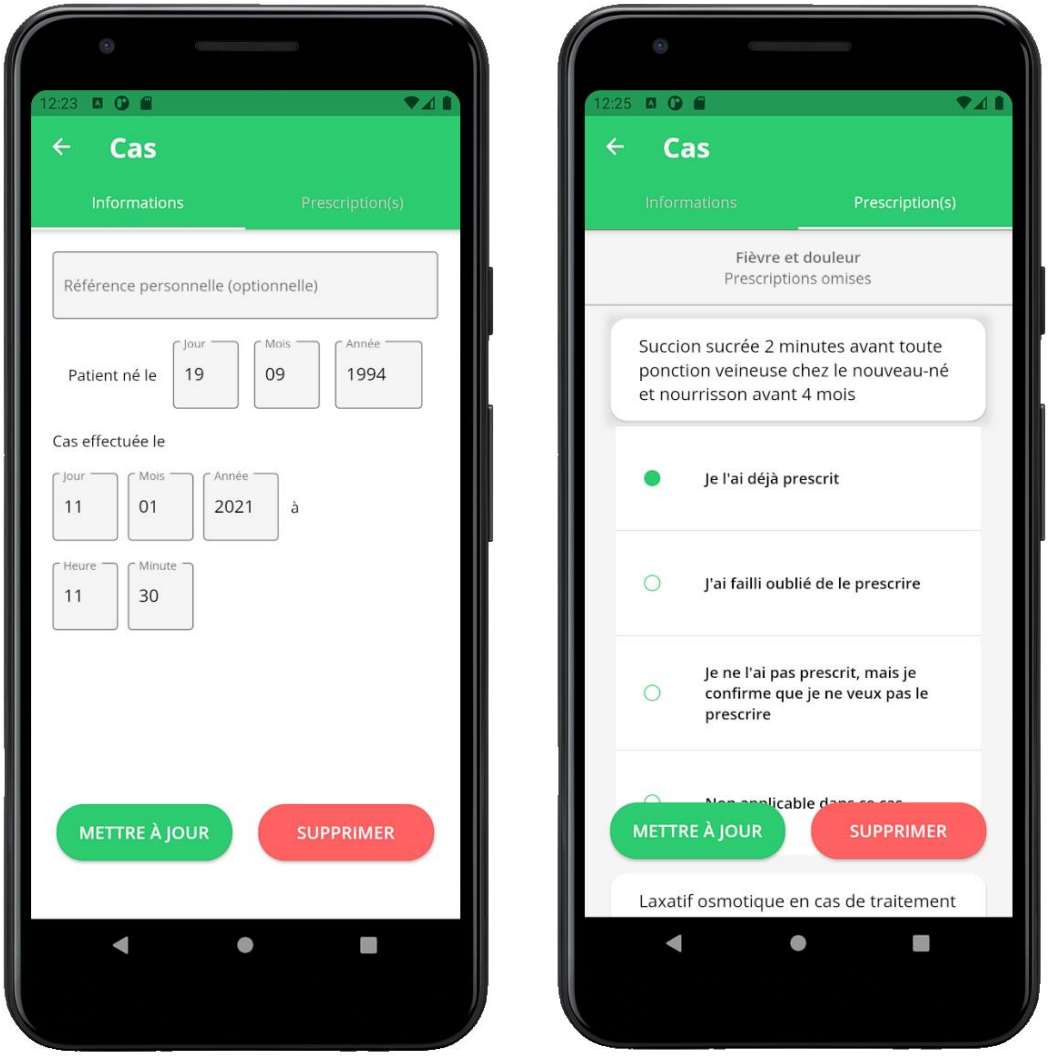


(John-Michael Quinto, 2020)

L'ajout d'un cas pour un nouveau patient se fait en trois étapes. Premièrement, l'utilisateur doit indiquer la date de naissance du patient et la date de consultation. La référence personnelle est optionnelle mais l'application vérifie que les dates sont valides et au bon format. Le bouton « Suivant » emmène l'utilisateur vers la deuxième étape, où il doit sélectionner au moins une pathologie pour le patient concerné. Une fois le ou les pathologie(s) sélectionnée(s), le bouton « Suivant » redirige l'utilisateur vers la dernière étape. Lors de cette étape, toutes les prescriptions omises et inappropriées pour les pathologies choisies sont affichées et l'utilisateur doit indiquer sa décision pour chacune d'elles. Le bouton « Valider » enregistre le nouveau cas dans la liste de l'utilisateur et il est ensuite redirigé vers la page des cas.

6.1.7 Modifier / supprimer un cas

Figure 40 : Ecran de modification d'un cas



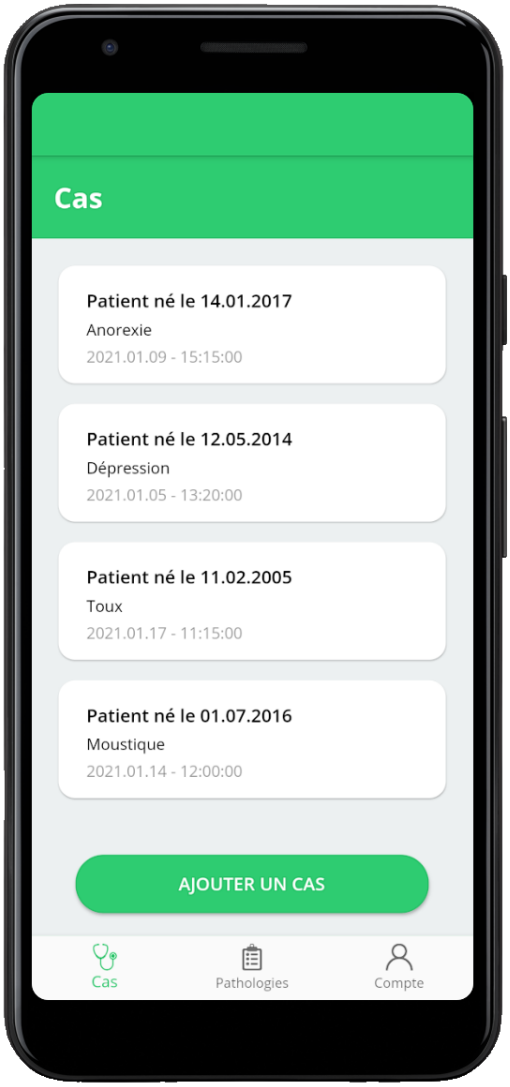
(John-Michael Quinto, 2020)

Lorsque l'utilisateur se trouve sur la page des cas, en sélectionner un le redirigera vers la page du cas concerné où les informations et prescriptions pour les pathologies du patient seront affichées. Il pourra modifier ce dont il voudra et appuyer sur le bouton « Mettre à jour » pour sauvegarder les changements. L'application vérifie également que les dates sont valides et au bon format. Le bouton « Supprimer » affichera un dialogue de confirmation pour permettre à l'utilisateur de vérifier son choix avant de supprimer le cas. Une fois supprimé, l'utilisateur est redirigé vers la page des cas.

6.2 Autres fonctionnalités

6.2.1 Affichage des cas

Figure 41 : Ecran d'affichage des cas

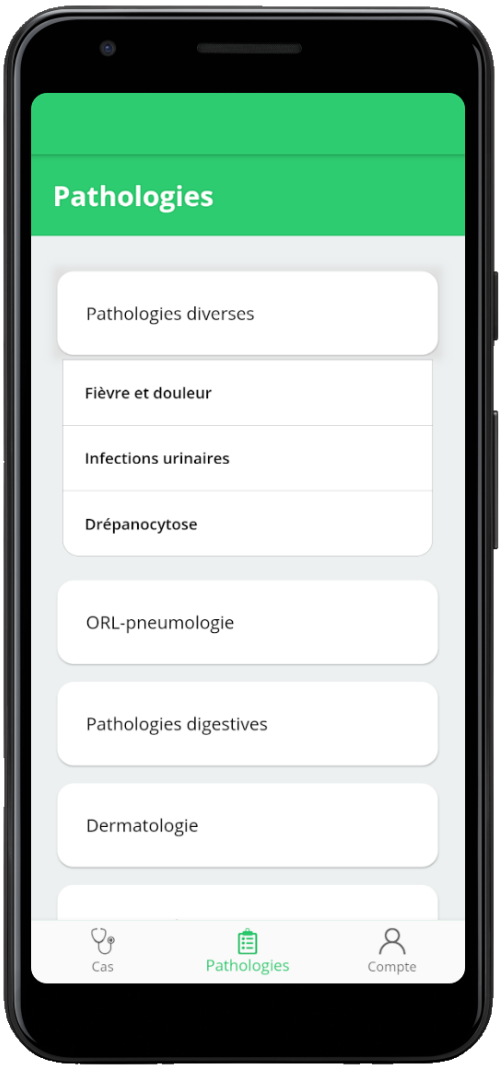


(John-Michael Quinto, 2020)

La page des cas est la première page qui s'affiche après la connexion d'un utilisateur et montre les cas traités par ce dernier. À travers cette page, nous pouvons ajouter un cas en appuyant sur le bouton « Ajouter un cas » ou nous diriger vers la page des informations d'un cas en le sélectionnant depuis la liste. Le navigateur d'onglet qui se trouve en bas de l'écran nous permet de basculer entre la page des pathologies et le menu du compte.

6.2.2 Affichage des pathologies

Figure 42 : Ecran d'affichage des pathologies



(John-Michael Quinto, 2020)

La page des pathologies affiche premièrement les familles de pathologies. Ensuite, en appuyant sur l'une d'elles, celle-ci affichera sous forme d'une liste déroulante les pathologies faisant partie de cette famille. Si l'on appuie une deuxième fois, la liste disparaîtra.

L'utilisateur peut également appuyer sur une des pathologies pour se rendre vers la page de ses prescriptions omises et inappropriées.

6.2.3 Affichage des prescriptions

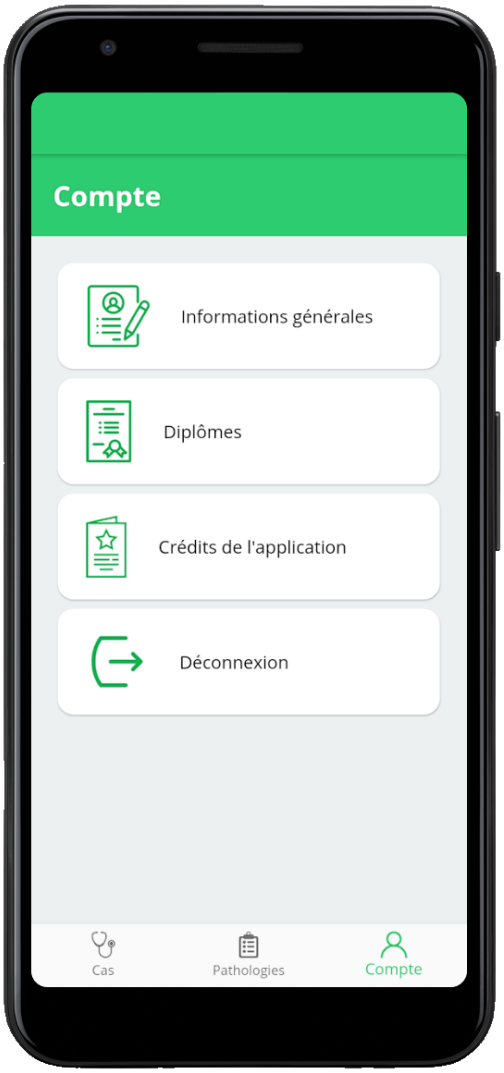
Figure 43 : Ecran d'affichage des prescriptions

(John-Michael Quinto, 2020)

La page des prescriptions est spécifique à chaque pathologie et est atteignable depuis la page des pathologies. Sur la figure ci-dessus, les prescriptions omises et inappropriées concernent la pathologie « Fièvre et douleur ».

6.2.4 Affichage du menu compte

Figure 44 : Ecran du menu du compte

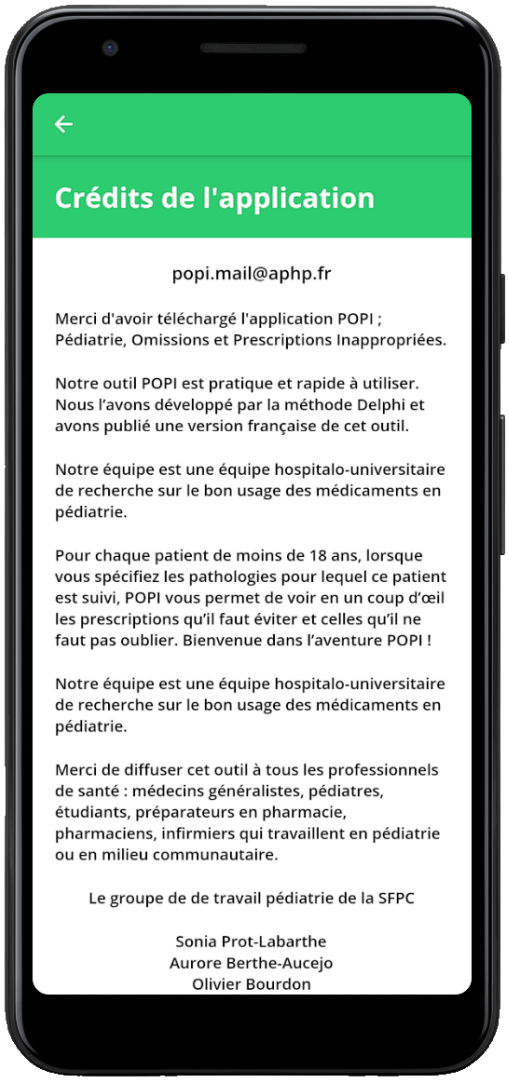


(John-Michael Quinto, 2020)

Depuis la page du menu du compte, nous pouvons accéder à plusieurs pages telles que la page des informations générales, des diplômes, et des crédits de l'application. L'utilisateur peut également se déconnecter en appuyant sur le bouton « Déconnexion », ce qui l'emmènera vers la page de connexion.

6.2.5 Affichage des crédits de l’application

Figure 45 : Ecran des crédits de l’application



(John-Michael Quinto, 2020)

La page des crédits de l’application inclut les remerciements de l’équipe ayant créée l’application et une simple description de l’objectif de son usage.

7. Développement de l'application

Dans cette partie, je vous partagerais mon expérience avec le framework Flutter, quelles étaient les difficultés rencontrées et comment j'ai pu les surmonter, puis nous verrons l'état actuel de l'application avec les améliorations qu'elle pourra peut-être implémenter dans le futur.

7.1 Difficultés et solutions

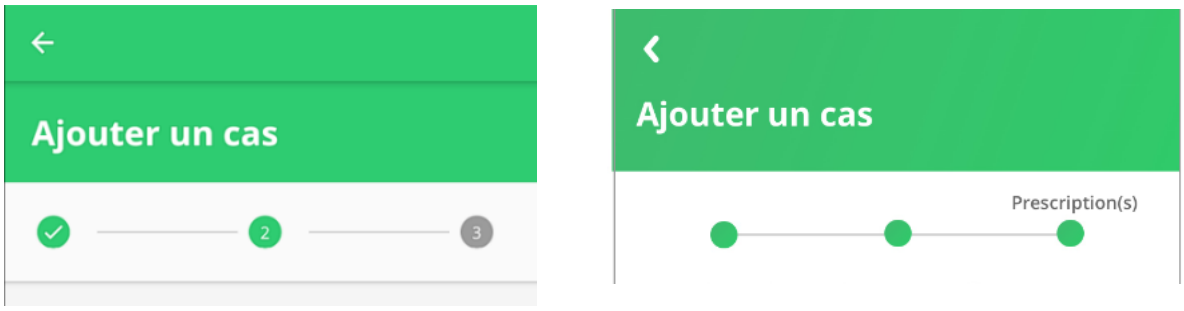
La prise en main de Flutter était pour moi assez compliqué et j'ai donc pris du temps pour bien démarrer. Son installation était relativement simple et il suffisait de suivre les instructions sur le site officielle pour le faire, mais la plus grosse difficulté était l'utilisation des widgets et comment les connecter entre eux. En lisant bien la documentation Flutter et surtout grâce à la communauté de développeurs, j'ai enfin fini par comprendre comment démarrer. À partir de ce moment, le développement de l'application était plus rapide. Malgré le fait que Flutter soit encore jeune, je n'ai pas eu de difficultés pour chercher des solutions à mes problèmes en ligne, ce qui montre le grand intérêt des développeurs pour ce framework.

7.2 État actuel

Actuellement, l'application implémente toutes les fonctionnalités attendues, à part pour quelques détails, et son utilisation est très fluide. Après avoir créé un compte, l'utilisateur peut directement accéder à toutes les fonctionnalités proposées par l'application.

En ce qui concerne les détails, nous avons par exemple le libellé qui figure au dessus de l'étape du formulaire sur lequel nous nous trouvons (figure de droite), mais qui est absent dans l'application (figure de gauche). Ceci est dû au widget « Steppers » permettant d'ajouter un formulaire avec des étapes mais dont l'apparence est difficilement modifiable.

Figure 46 : Détail manquant dans l'application



(John-Michael Quinto, 2020)

7.3 Rapport de tests

Numéro	Action	Résultat attendu	Résultat obtenu
1	Appuyer sur l'icône de l'application	Afficher la page de login	Affichage de la page de login
2	Appuyer sur le bouton « S'inscrire »	Afficher le formulaire pour créer un nouveau compte	Affichage du formulaire pour créer un nouveau compte
3	Saisir des informations non valides (par exemple : champ nul ou mot de passe trop court), et appuyer sur le bouton « Suivant »	Empêcher de procéder à l'étape suivante et afficher un message d'erreur	Affichage d'un message d'erreur sous chaque champ non valide
4	Saisir des informations correctes et appuyer sur le bouton « Suivant »	Afficher le formulaire pour ajouter les diplômes	Affichage du formulaire pour ajouter les diplômes
5	Appuyer sur le bouton « Créer » sans avoir ajouté au moins un diplôme	Empêcher la création et afficher un message d'erreur	Affichage d'un message d'erreur indiquant l'obligation d'ajouter au moins un diplôme
6	Appuyer sur le bouton « Créer » après avoir ajouté au moins un diplôme	Finaliser la création et connecter le nouvel utilisateur. Afficher la page contenant les cas du nouveau compte (vide à la création)	Connexion de l'utilisateur et redirection vers la page des cas
7	Connexion avec des informations non valides (par exemple : champ nul ou format de l'email incorrect) ou avec un compte inconnu	Afficher un message d'erreur	Affichage d'un message d'erreur pour chaque champ non valide et un message d'erreur si le compte est inconnu
8	Appuyer sur le bouton « Mot de passe oublié? »	Afficher le formulaire de demande d'un nouveau mot de passe	Affichage du formulaire de demande d'un nouveau mot de passe

9	Saisir un email non valide ou n'étant lié à aucun compte et appuyer sur le bouton « Valider »	Afficher un message d'erreur	Affichage d'un message d'erreur sous le champ de l'email ou d'un message indiquant qu'aucun compte avec l'email n'a été retrouvé suivant l'erreur
10	Saisir un email valide et appuyer sur le bouton « Valider »	Envoyer un email contenant le nouveau mot de passe	Envoi de l'email contenant le mot de passe et affichage d'un message de succès
11	Connexion avec des informations valides	Connecter l'utilisateur et afficher la page des cas	Connexion de l'utilisateur et redirection vers la page des cas
12	Appuyer sur le bouton « AJOUTER UN CAS »	Afficher le formulaire d'ajout pour un nouveau cas (avec indication du nom de l'étape)	Affichage du formulaire pour un nouveau cas (le nom des étapes ne sont pas indiqué)
13	Saisir des informations non valides (exemple : date non valide) et appuyer sur le bouton « Suivant »	Empêcher de procéder à l'étape suivante	Affichage d'un message d'erreur sous chaque champ non valide
14	Saisir des informations valides et appuyer sur le bouton « Suivant »	Afficher la prochaine étape pour l'ajout d'un cas (sélection des pathologies)	Affichage de l'étape suivante (sélection des pathologies)
15	Dans l'étape 2, ne sélectionner aucune pathologie et appuyer sur le bouton « Suivant »	Empêcher de procéder à l'étape suivante	Affichage d'un message d'erreur indiquant l'obligation de sélectionner au moins une pathologie
16	Appuyer sur une pathologie	Ajouter la pathologie dans la liste des pathologies du patient / l'enlever si on rappuie dessus	Affichage d'un cercle vert à côté de la pathologie pour indiquer qu'elle est sélectionnée / cercle blanc lorsqu'on le désélectionnee

17	Dans l'étape 2, appuyer sur le bouton « Suivant » après sélection du ou des pathologie(s)	Procéder à la dernière étape (affichage des prescriptions omises / inappropriées pour les pathologies sélectionnées)	Affichage de la dernière étape (sélection des prescriptions omises / inappropriées pour chaque pathologie)
18	Appuyer sur une option pour une des prescriptions	Sauvegarder l'option pour la prescription concernée	Un cercle vert indique que l'option est sélectionnée pour la prescription (par défaut l'option 1 est sélectionnée)
19	Appuyer sur le bouton « Valider »	Sauvegarder le nouveau cas et rediriger vers la page des cas	Sauvegarde du cas et redirection vers la page des cas (où se trouve le nouveau cas)
20	Appuyer sur un cas dans la page des cas	Afficher les informations du cas	Affichage de la page où se trouvent les informations du cas
21	Modifier les informations / options pour les prescriptions et appuyer sur le bouton « METTR À JOUR »	Mettre à jour le cas	Affichage d'un message indiquant que les informations ont été sauvegardées
22	Appuyer sur le bouton « SUPPRIMER »	Supprimer le cas	Affichage d'un message de confirmation. L'option "Continuer" entraîne la suppression du cas. Redirection vers la page des cas.
23	Appuyer sur l'onglet « Pathologies »	Afficher les familles de pathologies et leurs pathologies	Affichage des familles de pathologie et de leurs pathologies
24	Appuyer sur une pathologie	Afficher les prescriptions omises / inappropriées de cette pathologie	Affichage des prescriptions omises / inappropriées pour cette pathologie

25	Appuyer sur l'onglet « Compte »	Afficher le menu du compte	Affichage du menu du compte
26	Appuyer sur le bouton « Informations générales »	Afficher les informations du compte	Affichage des informations du compte
27	Modifier les informations du compte et appuyer sur le bouton « VALIDER »	Sauvegarder les nouvelles informations	Affichage d'un message indiquant que les informations ont été modifies
28	Appuyer sur le bouton « Diplômes »	Afficher les diplômes de l'utilisateur	Affichage des diplômes de l'utilisateur
29	Saisir les informations d'un nouveau diplôme et appuyer sur le bouton « AJOUTER »	Ajouter le nouveau diplôme dans la liste de diplômes de l'utilisateur	Affichage d'un message indiquant que les informations ont été mises à jour
30	Appuyer sur le bouton « X » à côté d'un diplôme	Supprimer le diplôme de la liste	Affichage d'un message indiquant que les informations ont été mises à jour
31	Appuyer sur le bouton « Crédits de l'application »	Afficher les crédits de l'application	Affichage des crédits de l'application
32	Appuyer sur le bouton « Déconnexion »	Déconnecter l'utilisateur	Déconnexion de l'utilisateur et redirection vers la page de login

7.4 Améliorations futures

L'objectif de l'application est simple et toutes les fonctionnalités implémentées permettent de l'atteindre. Cependant, une fonctionnalité qui pourrait être ajoutée serait la possibilité d'ajouter ou de supprimer une pathologie pour un cas déjà créé. En effet, la modification d'un cas se limite à celle des informations concernant la date de naissance et de consultation, et des choix pour chaque prescription des pathologies sélectionnées.

8. Analyse et conclusion

De nos jours, le fait de développer avec un framework cross-platform est plus favorable pour l'équipe de développement et le client car il permet de réduire significativement la durée et les coûts, tout en obtenant des applications de qualité. L'expérience utilisateur et l'accès aux fonctionnalités de chaque plateforme, qui n'étaient disponible qu'avec le développement natif, le sont maintenant avec qu'un seul code source.

Flutter propose une méthode de développement original avec ses widgets et une augmentation de la productivité grâce au Hot-Reload du Dart VM, permettant de compiler et d'observer les changements de manière presque instantanée.

Flutter propose énormément de widgets afin de développer rapidement ce dont nous avons besoin. En effet, pour l'application POPI, toutes les fonctionnalités ont pu être implémentées et son design correspond à celui qui a été fourni au début du projet. De plus, son utilisation est rapide et fluide. Cependant, certains widgets sont difficilement modifiables ce qui peut poser problème lorsque l'on souhaite suivre au détail près les maquettes de l'application.

La partie recherche m'a permis d'aller plus loin que simplement utiliser un framework. En effet, en plus de comprendre son fonctionnement, j'ai pu connaître ses origines et les éléments qui ont aboutis à sa création.

J'ai beaucoup apprécié travailler avec Flutter pour plusieurs raisons. Premièrement, j'ai pu développer d'une manière très unique et intéressante. Deuxièmement, j'ai eu l'opportunité de mettre en pratique les connaissances que j'ai acquises durant mes années à l'HEG, comme l'utilisation des services REST. Finalement, Flutter et le langage Dart sont de plus en plus utilisés et commencent à prendre de l'ampleur sur le marché. Je me réjouis donc d'avoir pu bénéficier de cette expérience.

Bibliographie

CLOW, Mark, 2019. *Learn Google Flutter Fast : 65 Example Apps.* Kindle Edition. 31 mars 2019. [Consulté le 19 novembre]. ASIN : B07NX3WM6L. Disponible à l'adresse : https://www.amazon.fr/Learn-Google-Flutter-Fast-Example-ebook/dp/B07NX3WM6L/ref=tmm_kin_swatch_0?_encoding=UTF8&qid=&sr= [accès par abonnement]

Dart, pas de date. Platforms [en ligne]. Pas de date. [Consulté le 16 janvier 2021]. Disponible à l'adresse : https://dart.dev/platforms

Flutter, pas de date. Flutter architectural overview. [en ligne]. Pas de date. [Consulté le 11 novembre 2020]. Disponible à l'adresse : https://flutter.dev/docs/resources/architectural-overview

Flutter, pas de date. Build and release an Android app. Flutter.dev [en ligne]. Pas de date. [Consulté le 9 décembre 2020]. Disponible à l'adresse : https://flutter.dev/docs/deployment/android

Flutter, pas de date. Build and release an IOS app. Flutter.dev [en ligne]. Pas de date. [Consulté le 9 décembre 2020]. Disponible à l'adresse : https://flutter.dev/docs/deployment/ios

GRIFFITH, Chris, pas de date. What is Hybrid App Development?. Ionic.io [en ligne]. Pas de date. [Consulté le 5 février 2021]. Disponible à l'adresse : https://ionic.io/resources/articles/what-is-hybrid-app-development

Ionos, 2020. Dart programming language: an introduction. Ionos.com [en ligne]. 15 octobre 2020. [Consulté le 15 janvier 2021]. Disponible à l'adresse : https://www.ionos.com/digitalguide/websites/web-development/dart-programming-language/

KIDECHA, Sanjay, 2020. Native vs. Hybrid vs. Cross-Platform: How and What to Choose?. Dzone.com [en ligne]. 27 May 2020. [Consulté le 19 octobre 2020]. Disponible à l'adresse: https://dzone.com/articles/native-vs-hybrid-vs-cross-platform-how-and-what-to#:~:text=The%20unmatched%20user%20interface%20will,on%20the%20framework%20you%20choose.

NAPOLI, Marco L, 2019. *Beginning Flutter* [en ligne]. Indianapolis : Wrox. Septembre 2019. ISBN 978-1-119-55085-3. Disponible à l'adresse : https://www.wiley.com/en-us/Beginning+Flutter%3A+A+Hands+On+Guide+to+App+Development-p-9781119550853 [accès par achat]

SEIDEL, Eric, 2018. Interview with Eric Seidel (Flutter Live) [enregistrement vidéo]. Youtube.com. 8 décembre 2018. [Consulté le 17 octobre 2020]. Disponible à l'adresse : https://www.youtube.com/watch?v=7b1OJjug3MI

SIDDHARTH, Sharma, 2020. Native vs Hybrid vs Cross-Platform – What to Choose?. Medium.com [en ligne]. 9 août 2020. [Consulté le 19 octobre 2020]. Disponible à l'adresse : https://medium.com/flutterdevs/native-vs-hybrid-vs-cross-platform-what-to-choose-3221130f7cc5

Skia, pas de date. Skia Graphics Library. [en ligne]. Pas de date. [Consulté le 11 novembre 2020]. Disponible à l'adresse : https://skia.org/

Stack overflow, 2019. Developer Survey Results 2019. Stackoverflow.com [en ligne]. Mars 2019. [Consulté le 27 octobre 2020]. Disponible à l'adresse : https://insights.stackoverflow.com/survey/2019

Stack overflow, 2020. 2020 Developer Survey. [en ligne]. Février 2020. [Consulté le 27 octobre 2020]. Disponible à l'adresse : https://insights.stackoverflow.com/survey/2020

Statista, 2020. Cross-platform mobile frameworks used by software developers worldwide in 2019 and 2020. Stackoverflow.com [en ligne]. 2 juillet 2020. [Consulté le 27 octobre 2020]. Disponible à l'adresse : https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/

Topflight, 2020. Progressive Web Apps vs. Native Apps in 2021: Pros and Cons. Topflight.com [en ligne]. 18 aôut 2020. [Consulté le 3 novembre 2020]. Disponible à l'adresse : https://topflightapps.com/ideas/native-vs-progressive-web-app/

Tutorials Point, 2019. *Flutter Tutorial* [en ligne]. 2019. Disponible à l'adresse : https://www.tutorialspoint.com/flutter/flutter_tutorial.pdf