UNIVERSITÉ DE LORRAINE
INSTITUT UNIVERSITAIRE DE TECHNOLOGIE

Département Informatique

Projet de Remplacement :

Compte rendu sur Flutter

Réalisé par :
Sélène VIOLA

Enseignant tuteur :
Samuel CRUZ-LARA

ANNÉE UNIVERSITAIRE 2019-2020

SOMMAIRE

Introduction

Flutter est un framework de développement mobile open source créé par Google en 2017. Grâce à ce framework, nous pouvons développer des applications multiplateforme pour Android ou iOS.
Le langage utilisé par Flutter est un langage de programmation web et mobile développé lui aussi par Google et appelé Dart.
En 2013, Lars Bak et Gilad Bracha sortent la première version de Dart. Cette dernière offrirait une alternative à JavaScript et aurait de meilleures performances, une meilleure sécurité ainsi que la capacité à être plus facilement utilisable pour des projets à grande échelle que JavaScript.
Dart garde cependant une syntaxe familière à celle de JavaScript ou encore de Java, ce qui facilite donc son apprentissage. De plus, il est possible de voir instantanément les modifications effectuées grâce au “rechargement à chaud” (“hot reload”).
Il existe aussi un éditeur en ligne “DartPad”.
Pour résumer, Dart est un langage assez simple, orienté objet avec des héritages, des interfaces, des classes concrètes et abstraites et les fonctions sont des objets. L’utilisation de Flutter permet de garder la nature dynamique du JavaScript, tout en offrant un langage et des outils facilitant le développement de grosses applications.

1. Différents fichiers

Lorsque nous créons un nouveau projet Flutter, plusieurs répertoires et plusieurs fichiers se créés. Dans le répertoire lib, nous avons le fichier main.dart. Ce dernier est la classe principale qui fait appel à la méthode runApp(myApp).

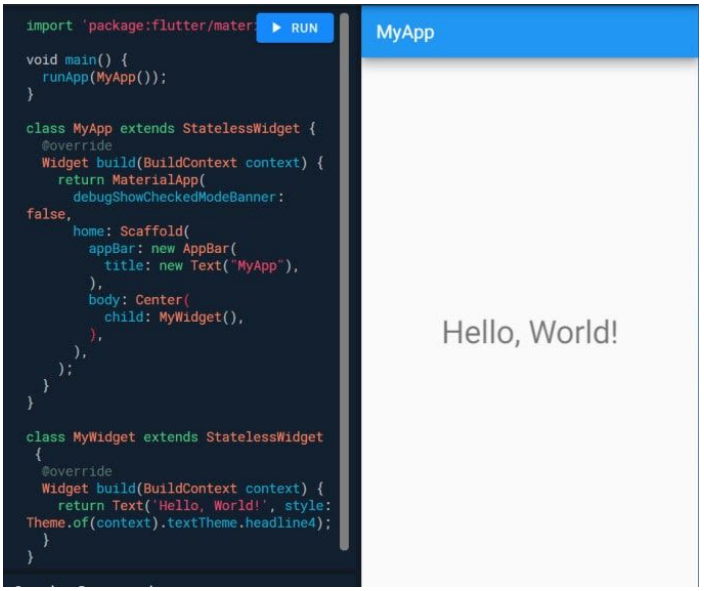
pubspec.yaml est le fichier qui permet d'avoir accès à certaines informations sur l'application comme son nom ou encore sa description. Il permet aussi d'avoir accès à l'environnement et à la version ainsi qu'à certaines fonctionnalités comme ajouter des icônes, des images ou encore de télécharger certains packages comme "english_words" qui regroupe une centaine des mots les plus utilisés en anglais par exemple.

AndroidManifest.xml permet de changer les paramètres de l'application sur Android comme son icône ou son nom.

Info.plist permet de changer les paramètres de l'application sur iOS comme son icône ou son nom.

2. Fonctionnement général de Flutter

La syntaxe de Flutter reste similaire à celle de JavaScript. De plus, Flutter fonctionne sous forme de classes comme Java ainsi qu'avec des méthodes, des variables et des widgets.



2.1 Les variables

final : permet d'interdire aux autres classes de modifier cette variable.
const : permet d'initialiser une variable avec une valeur constante ou pour déclarer des constructeurs qui créés des valeurs constantes.
var : autorise les affectations multiples.

Pour montrer qu'une variable est privée, on ajoute "_" avant son nom (ex : String _nom;).

Dans certains cas, nous avons forcément besoin d'une variable (ou d'une méthode) dans une classe pour continuer. Nous devons donc ajouter l'annotation "maClasse({@required this.variable});".

2.2 Les widgets

Un widget est construit à l'aide de la méthode build(BuildContext context) et peut ne pas avoir d'état ou en avoir un. Dans ce dernier cas, le widget peut gérer son propre état, dans le cas d'une utilisation plus esthétique, ou déléguer la gestion à ses parents, dans le cas d'une utilisation qui nécessite plus de sécurité comme pour la gestion des données utilisateurs par exemple.

2.2.1 Utilisation des widgets sans état

On a une classe widget qui hérite de StatelessWidget.

2.2.2 Utilisation des widgets avec état

Pour créer un widget avec état, on doit créer deux classes, une classe widget et une classe widgetState. La classe widget hérite de la classe StatefulWidget qui nous permet de créer l'état à l'aide de la méthode createState().
La classe widgetState hérite de la classe State<widget>. Cette dernière nous permet de mettre à jour la valeur du widget ainsi que son affichage en notifiant "Flutter" que l'état a changé à l'aide de la méthode setState().

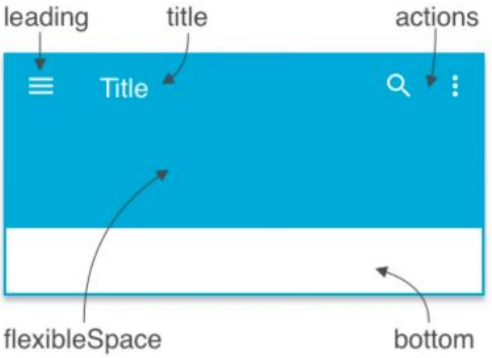
3. Quelques éléments fréquemment utilisés

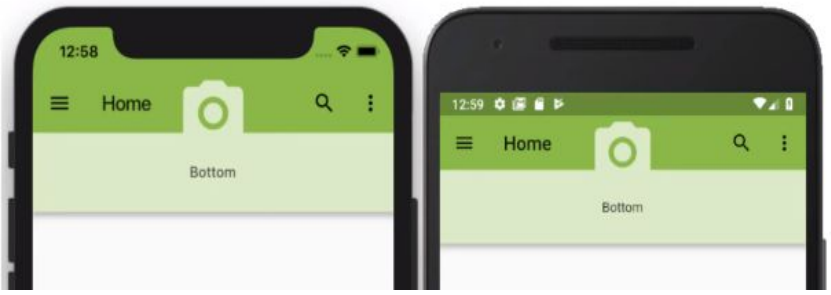
3.1 Scaffold

Cette classe permet de mettre en place la structure visuelle de la page comme les actions, les titres, les menus, …
Voici quelques exemples de structure utiles.

3.1.1 AppBar

La propriété appBar est utilisée pour ajouter une barre en haut de l'application. Dans cette barre on peut y mettre un titre, y mettre des widgets ou encore des icônes ou des boutons.





3.1.2 Body

Le body affiche ses enfants qui seront le contenu principal de la page. On peut y ajouter des champs de texte, des images, des icônes, des boutons, des listes, …

3.1.3 Les boutons

Il y a plusieurs sortes de boutons. Les plus couramment utilisés sont les boutons flottants ("FloatingActionButton") qui viennent recouvrir une partie du body et qui sont eux aussi une propriété de la classe Scaffold. Ils sont utilisés pour des actions principales comme pour créer quelque chose ou partager quelque chose par exemple.

Les boutons sont en général composés d'icônes ou encore de texte ou les deux.

Lorsque nous avons plusieurs choix possibles, nous pouvons ajouter un menu déroulant appelé "DropdownButton". C'est en fait une liste d'items ("DropdownMenuItem") avec une valeur et un nom. Nous avons ensuite une méthode qui, à chaque changement, envoie une notification signalant que l'état du bouton a changé ("setState()") et indiquant les actions à effectuer en fonction de la valeur du bouton.

3.1.4 Les listes

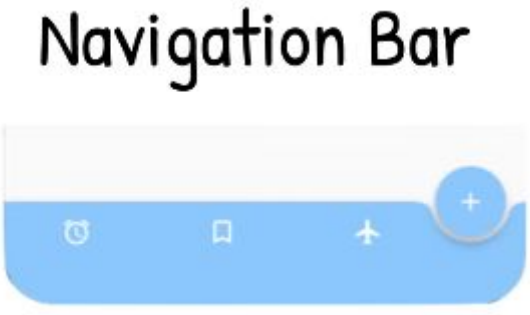
Comme pour les boutons, il existe beaucoup de types de listes différents. Celle qui nous intéressait le plus dans notre situation est la "Listview". Cette dernière permet de créer une liste de widgets disposés de façon linéaire et qui charge les éléments au fur-et-à-mesure du défilement de la liste pour avoir un affichage constant et rapide.
On peut aussi citer les listes de type ListTile qui permettent d'afficher une seule ligne à hauteur fixe avec du texte et des icônes. On aperçoit aussi souvent les listes de type CheckboxListTile qui permettent d'avoir les mêmes propriétés qu'une ListTile, mais avec une case à cocher.

3.1.5 Drawer

Cette propriété nous permet d'avoir un menu coulissant qui s'affiche sur le côté de l'écran. On peut accéder à ce menu à l'aide d'une icône ou en glissant son doigt de gauche à droite (ou de droite à gauche).

3.1.6 BottomNavigationBar

Cette propriété est utile pour afficher une barre avec trois à cinq éléments en bas de l'application. En général, les éléments sont composés de texte, d'icônes ou des deux. Cette barre est une aide ergonomique pour faciliter la navigation entre plusieurs pages.



4. MaterialApp

Cette classe regroupe des widgets utiles à la conception matérielle et visuelle de l'application, c'est un peu la charte graphique de Google sur ces applications web et mobile.
C'est aussi dans cette dernière que l'on crée les routes de l'application pour faciliter la navigation entre les différentes vues.

4.1 Routes

À partir du moment où l'on commence à avoir plusieurs pages dans notre application mobile, nous devons créer des routes vers ces pages. Pour ce faire, nous devons nommer les différentes pages à l'aide des propriétés initialRoute et routes de MaterialApp. "initialRoute" est la page d'accueil et routes est une liste de routes nommées. Chaque route est représentée par son nom et les widgets à créer lors de la navigation vers ces pages.
Lorsque l'on presse un bouton pour changer de page, on doit le lier à la nouvelle page. Pour ce faire, on utilise la méthode "onPressed" qui fera appel à "Navigator.pushNamed(context, nomDeLaPage)".
On peut aussi utiliser "Navigator.push(context)" pour créer une route non nommée et "Navigator.pop(context)" pour revenir à la route précédente.

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'To do list',
    initialRoute: '/',
    routes: {
    '/': (context) => ListeTODO(taches: tachesCrees, tacheComplete: tacheFaite),
    '/create': (context) => ListeCreation(creationTache : creationNouvelleTache),
    },
  ); // MaterialApp
}
```

5. Quelques widgets intéressants

Voici une liste de quelques widgets que Flutter fournit :

- Structure d'Application et navigation: Scaffold, Appbar, BottomNavigationBar, TabBar, TabBarView, Drawer
- Buttons: RaisedButton, FloatingActionButton, FlatButton, IconButton, PopupMenuButton, ButtonBar
- Input et Sélection: TextField, Checkbox, Radio, Switch, Slider, Date & Time Pickers, FormField, Form
- Dialogues, alertes et panneaux: SimpleDialog, AlertDialog, BottomSheet, ExpansionPanel, SnackBar
- Information displays: Image, Icon, Chip, Tooltip, DataTable, Card, LinearProgressIndicator, GridView
- Layout: ListTile, Stepper, Divider
- Basic: Container, Row, Column, Text, RichText
- Support des Canvas
- Cupertino (iOS -style): ActivityIndicator, AlertDialog, Button, Dialog, DialogAction, Slider, Switch, Picker, PageTransition, FullscreenDialogTransition, NavigationBar, TabBar, PageScaffold, TabScaffold, TabView
- Layout: Container, Padding, Center, Align, FittedBox, AspectRatio, ConstrainedBox, Baseline, FractionallySizedBox, SizedBox, OverfloxBox, Transform, Stack, GridView, Flow, Table, Wrap, ListBody, ListView…
- Animation: AnimatedContainer, AnimatedCrossFade, Hero, DecoratedBoxTransition, FadeTransition, PositionedTransition, RotationTransition, ScaleTransition, SizeTransition, AnimatedOpacity…
- Interaction: Draggable, LongPressDraggable, GestureDetector, DragTarget, Dismissible, IgnorePointer, AbsortPointer, Scrollable
- Routing: Hero, Navigator
- Styling: Theme, MediaQuery
- Painting and Effect: Opacity, Transform, DecoratedBox, FractionalTranslation, RotatedBox, ClipOval, ClipPath, ClipRect, CustomPaint, BackdropFilter
- Scrolling
- Accessibility

6. Après la théorie, la pratique !

Pour ce travail de remplacement, j'ai choisi de faire une application de type To Do List. En effet, durant le module d'application mobile, nous avons déjà expérimenté ce genre de projet à l'aide de templates et d'Onsen UI. J'ai trouvé que recommencer l'application, cette fois-ci avec Flutter pouvait être très intéressant.

6.1 Choix du template

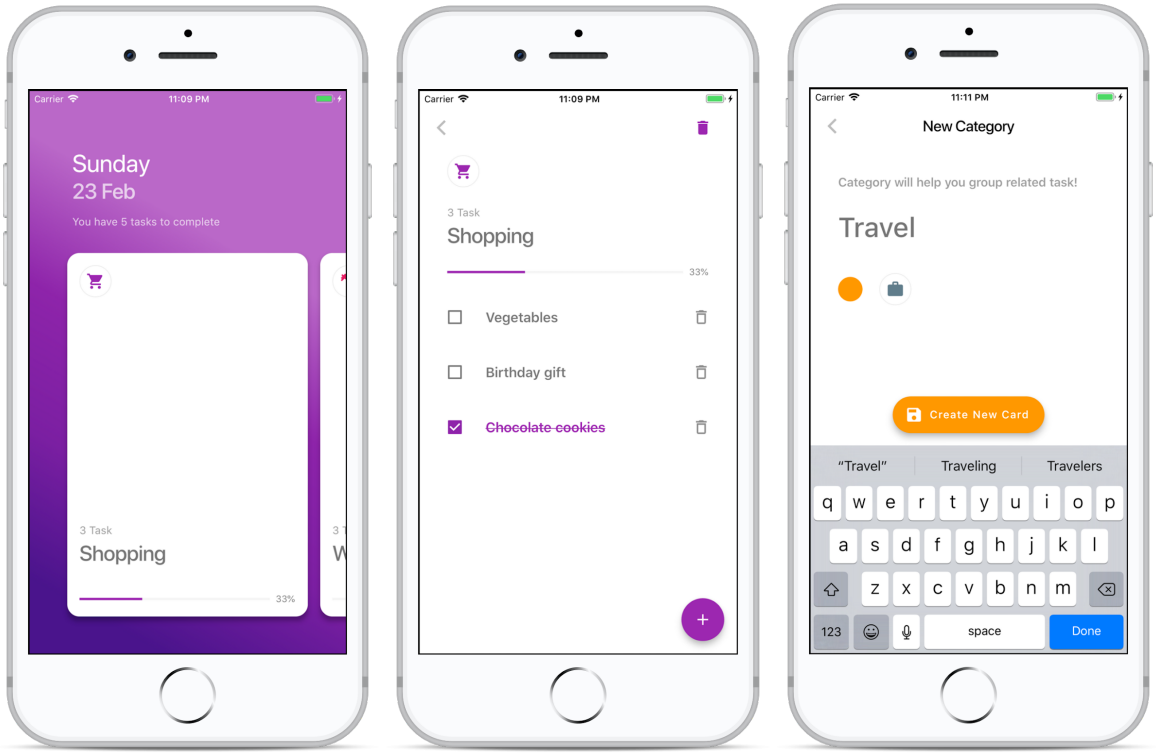


Image 1 : écran d'accueil et gestion des cartes de catégorie.
Image 2 : détails et gestion des tâches présentes dans la catégorie "Shopping".
Image 3 : création d'une nouvelle catégorie.

Il y a beaucoup de template disponibles pour faire une application To Do list et la plupart se ressemblent tous. Je voulais quelque chose de dynamique et de parlant. J'ai donc choisi un template utilisant beaucoup d'icônes, de formes et de couleurs.

Voici le lien du dépôt git du template :
https://github.com/sabinbajracharya/fluttery-todo

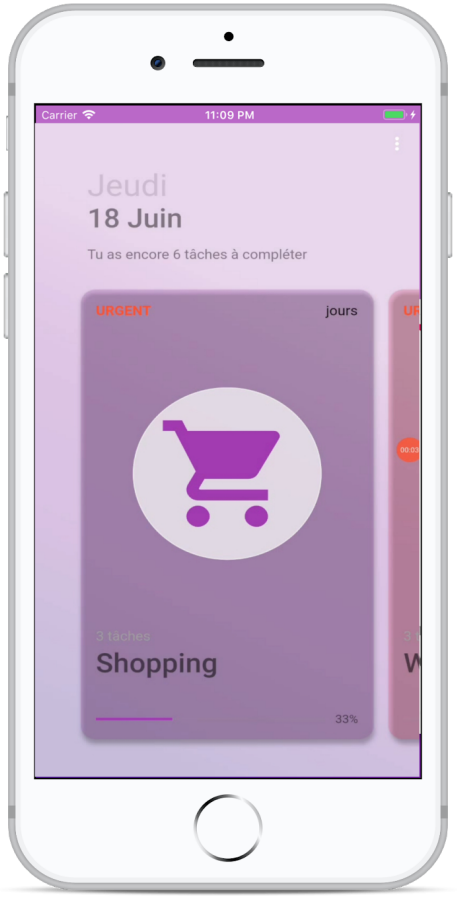
6.2 Les modifications effectuées

Tout d'abord, j'ai étudié le code page par page pour me créer des repères et comprendre la structure de l'application. Dès que j'ai apprivoisé la structure principale, j'ai pu commencer les modifications.
La première chose que j'ai faite a été de traduire toute l'application qui de base était en anglais. J'ai étudié les différentes classes de Dart qui gèrent la date et l'heure pour avoir une date locale. Or, je n'ai pas réussi à l'intégrer à l'application. Donc, par manque de temps, j'ai traduit la date à l'aide d'un switch que j'ai ajouté à la classe DateTimeUtils du fichier lib/utils/datetime_utils.dart.
Ensuite, j'ai agrandi les icônes représentant la catégorie et modifié les couleurs d'arrière-plan de l'icône et de la catégorie.
Puis, j'ai changé le nom de l'application ainsi que son icône. Grâce à la fonctionnalité "Image Asset" d'Android Studio, j'ai pu créer plusieurs tailles pour l'icône de l'application.
Pour finir, j'ai ajouté un bouton switch 'Urgent' et un bouton calendrier pour la deadline lors de la création de catégories. Tous deux étaient fonctionnels, mais je n'ai pas réussi à récupérer et sauver les données dans la base de données. J'ai donc supprimé ces deux fonctionnalités.

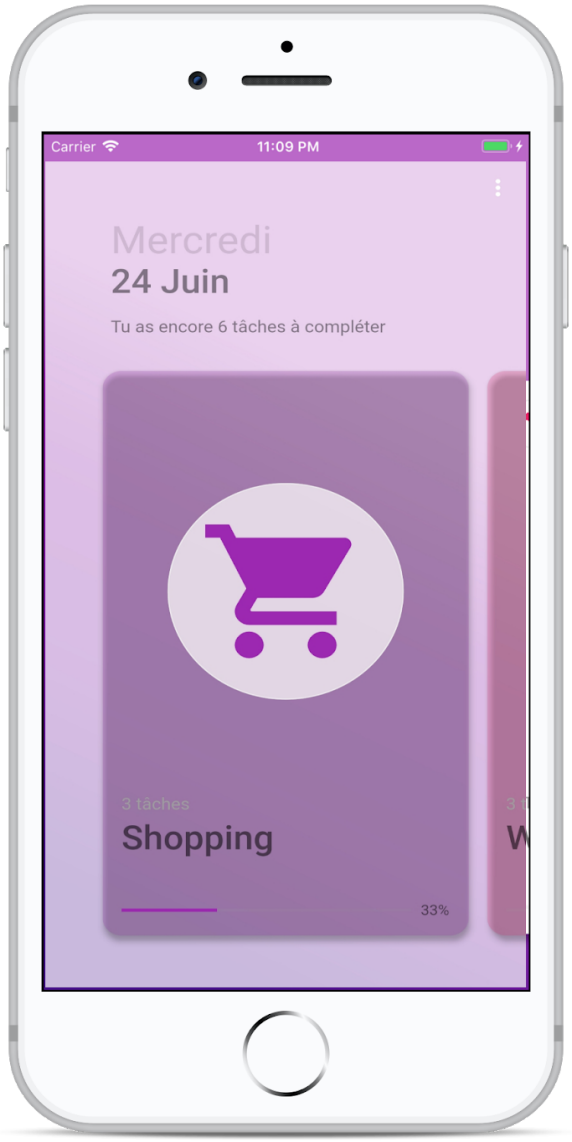
6.3 Les changements non concrétisés

J'ai commencé à implémenter deux autres fonctionnalités qui n'ont pas abouties par manque de temps. La première est une notion d'urgence pour les catégories. Le but étant de signaler en haut à gauche d'une carte catégorie si elle est urgente ou non, à l'aide d'une icône "danger" ou du mot "URGENT" par exemple.
La deuxième fonctionnalité est l'ajout d'une deadline présente en haut à droite de la carte catégorie.
Ces deux fonctionnalités auraient pu être implémentées sur les tâches en elle-même et non pas leur catégorie.
Pour finir, comme dit précédemment, pouvoir modifier le code pour avoir une date locale à chaque utilisateur serait aussi un plus.

6.4 Photos des modifications non concrétisées



6.5 Photo de l’écran d’accueil après modifications



Conclusion

Pour cette conclusion, j'ai choisi de vous exprimer mon ressenti concernant Flutter et Dart.
Tout d'abord, au vu de la situation et du temps très limité que nous avons eu pour étudier ce nouveau langage, je n'ai pu voir que les bases de Flutter et de Dart.
Dart est un langage structuré et orienté objet assez proche de JavaScript et de Java. Ayant déjà des bagages avec ces derniers, la prise en main des bases fût assez simple. J'ai pu rapidement programmer des applications simples et fonctionnelles. De plus, Flutter et Dart sont deux langages ayant énormément de documentation et de tutoriels ce qui facilite d'autant plus l'apprentissage.
Pour en finir avec les avantages, la fonctionnalité "Hot Reload" est vraiment très intéressante. En effet, elle nous permet de voir quasiment instantanément après la sauvegarde les changements sur le mobile sans avoir à attendre aussi longtemps que d'ordinaire.

En revanche, ce qui fait la force de Flutter est aussi sa faiblesse. Il est facile de se perdre à cause de l'emboîtement de widgets. Le principe des widgets est très bien car chaque fonctionnalités est représentée par un widget ce qui rend le code plus facilement lisible. Mais dès lors où l'on souhaite ajouter pleins de widgets, changer leurs paramètres, leurs tailles, leur couleur, cela devient très vite illisible.

Pour conclure, malgré le manque de temps, j'ai vraiment apprécié découvrir Flutter et Dart, ces nouveaux langages qui m'étaient jusqu'à présent inconnu.
J'ai appris à les utiliser en suivant des tutoriels et en m'aidant de la documentation. Grâce à cela, j'ai donc pu finaliser ce projet en programmant une application mobile de "to do list".
Enfin, cette expérience m'a donc conforté dans l'idée de continuer mes études vers une licence orientée applications mobiles.