Projet Individuel BAC 3

SmartSplit

Bénimédourène Charles

2021-2022 (Q2)

Table des matières

Environnement

Cette section aborde les différents composants utilisés lors de l'implémentation de SmartSplit, allant des prérequis fondamentaux à toute maintenance ou modification du programme aux bonnes pratiques et outils utilisés mais n'étant pas essentiels.

Framework

Flutter (fondamental)

Description

SmartSplit a été créé avec Flutter, un framework de Google utilisant Dart comme langage de programmation et présentant comme principal attrait la possibilité de concevoir des applications et web-apps facilement déployables sur une multitude de plateformes (Windows, Mac, Android, Linux, Web).

Avantages

En plus de pouvoir être compilé afin de tourner sur une multitude de plateforme sans devoir changer une seule ligne de code, Flutter offre les avantages suivants:

- **Hot-reload**: Les changements apportés au code se reflètent directement sur l'application sans nécessiter une recompilation du code ce qui permet de gagner du temps et de débugger les éventuels problèmes plus rapidement.
- **Plugins**: Flutter étant open-source et possédant une grande communauté active, de nombreux utilisateurs partagent leurs créations sous forme de plugins. Ces plugins peuvent aller de widgets réutilisables tels que des affichages d'étoiles pour des revues à des fonctionnalités internes liées au backend du programme et à son fonctionnement.
  La liste des plugins utilisés est disponible ici.
- **Documentation**: Flutter est documenté de manière très exhaustive, la documentation (disponible ici) est produite par Google et agrémentée de vidéos et d'exemples afin de rapidement faire comprendre aux utilisateurs les bases du framework et ses spécificités. La documentation est donc extrêmement concise et agréable à naviguer avec un soin presque artistique porté à son aspect et aux vidéos d'explication présentées par des développeurs de Google.

Fonctionnement

Flutter utilise le langage de programmation Dart qui est extrêmement similaire à JavaScript et avec comme spécificité pour Flutter de reposer sur le système de “Widgets”. Les widgets représentent basiquement les blocs de construction pour l’interface et reposent sur un système d’imbrication.
Par exemple, le widget principal pourrait être une page blanche et dedans, nous pourrions ajouter un widget “Column” listant lui-même 5 widgets “Text” avec comme valeur “Hello World”. Ainsi le résultat final serait une page blanche avec au centre une colonne de 5 “Hello World”.

Installation

Il est nécessaire d’avoir installé Flutter sur sa machine afin de pouvoir apporter des modifications au programme. Pour plus d’informations regardant l’**installation** voici un lien vers la documentation officielle détaillant le procédé afin d’installer la version la plus récente en fonction de son OS.

Plugins (fondamentaux)

Description

Les plugins de Flutter sont des extensions créées par les développeurs et utilisateurs afin de faciliter la vie de la communauté ou de partager des créations intéressantes.

Installation

Les plugins utilisés par l’application seront installés automatiquement lors du lancement de l’application en mode debug (si ce n’est pas le cas, naviguez jusqu’au fichier pubspec.yaml et sauvegardez le).

L’installation d’autres plugins est très simple et revient uniquement à utiliser une ligne de commande (“flutter pub add x”) ou à ajouter une entrée au fichier pubspec.yaml. Par précaution, la page des plugins documente également la commande à utiliser et/ou l’entrée à rajouter au fichier pubspec.yaml.

Plugins utilisés

Hive
Hive est un plugin offrant un système de gestion des données NoSQL rapide, performant et facile d'utilisation. Le plugin sert donc de base de donnée/backend à SmartSplit et permet de faire persister les informations des étudiants.

Notes:
J'ai décidé d'utiliser ce type de database afin de découvrir une nouvelle manière de faire persister les données qui n'a pas été vue en cours (et donc malheureusement, la base de donnée finale présente dans l'application n'est pas des plus optimales et peut prêter à confusion à cause de la redondance des données). Ainsi la base de données pourrait être modifiée en plusieurs "boites" afin d'être plus optimale et ne plus présenter de redondances.

Dans la version "application" de SmartSplit, les fichiers sont sauvegardés localement sur la machine de l'utilisateur mais dans la version "web", la base de données est contenue dans le navigateur!

File Picker
File Picker est un plugin permettant à l'application d'utiliser le gestionnaire de fichiers de la plateforme locale afin de sélectionner des fichiers. Dans ce cas-ci, File Picker est utilisé afin d'importer les fichiers csv contenant la liste des élèves et leurs informations.

Notes:
File Picker est également utilisé afin de sauvegarder des fichiers ou de les télécharger (si l'application est en mode web et tourne sur navigateur). Ainsi l'export en format csv de la base de données est gérée par File Picker.

BitsDojo Window (version app)
Ce plugin est utilisé afin de se débarrasser de la barre d'application par défaut du système d'opération de l'utilisateur et de pouvoir ensuite créer sa propre barre d'application (de manière similaire aux applications telles que Docker, Spotify, Chrome, etc).

Notes:
Ce plugin n'est utilisé que par la version application/desktop de SmartSplit et n'offre que des fonctionnalités visuelles et esthétiques. Ce plugin pourrait donc être retiré si dans le futur il n'était pas mis à jour correctement ou commencerait à poser des problèmes.

Path Provider (version app)
Path Provider est un plugin permettant d'utiliser des chemins afin de naviguer dans l'arborescence de fichier sur la machine de l'utilisateur.

Notes:
Utilisé par SmartSplit pour localiser le dossier des documents afin d'y enregistrer la base de données et les fichiers exportés.

Provider (version app)
Provider est un plugin simplifiant entre autres la gestion des états des widgets.

Notes:
Provider n'a au final presque pas été utilisé dans SmartSplit étant donné que j'ai par la suite découvert d'autres manières de gérer les états des widgets et a été retiré de la version web du projet.

IDE

Visual Studio Code et extensions (optionnel)

J'ai personnellement travaillé avec Visual Studio Code (VS code, à ne pas confondre avec Visual Studio Community) qui est un IDE assez commun, léger et versatile avec une forte communauté active produisant des extensions afin de rendre l'expérience de programmation plus agréable et rapide.

Je n'ai utilisé qu'une seule extension lors de l'implémentation de SmartSplit nommée "Flutter". Cette extension permet de débugger la majeure partie des erreurs avant le runtime ainsi que d'auto compléter les entrées pour travailler plus rapidement (linter et intellisense pour Flutter (et Dart)).

Pour ouvrir le projet SmartSplit avec Visual Studio Code, il suffit de faire un clic droit sur le fichier *"smart_split"* dans *"src"* et de cliquer sur "ouvrir avec visual studio code".

Bien entendu, n'importe quel IDE peut être utilisé pour travailler sur ce projet, la documentation officielle de Flutter contenant un guide pour VScode, IntelliJ et Android Studio.

OS

Windows (optionnel)

J'ai personnellement travaillé sous windows étant donné qu'il reste l'operating system avec lequel je suis le plus familier. Si vous souhaitez utiliser un autre OS que Windows, **Mac** et **Linux fonctionnent** parfaitement pour travailler sur le projet (à noter que lors de la compilation en une application, l'application ne sera compilée que pour votre os, par exemple si vous décidez de compiler SmartSplit sur une machine Linux, l'exécutable SmartSplit ne fonctionnera que sur les machines Linux).
La seule version desktop disponible sur Github est la version Windows actuellement mais la compilation en une version Linux ou Mac ne requiert aucune modification du code ou de l'implémentation.

Testing

Assert

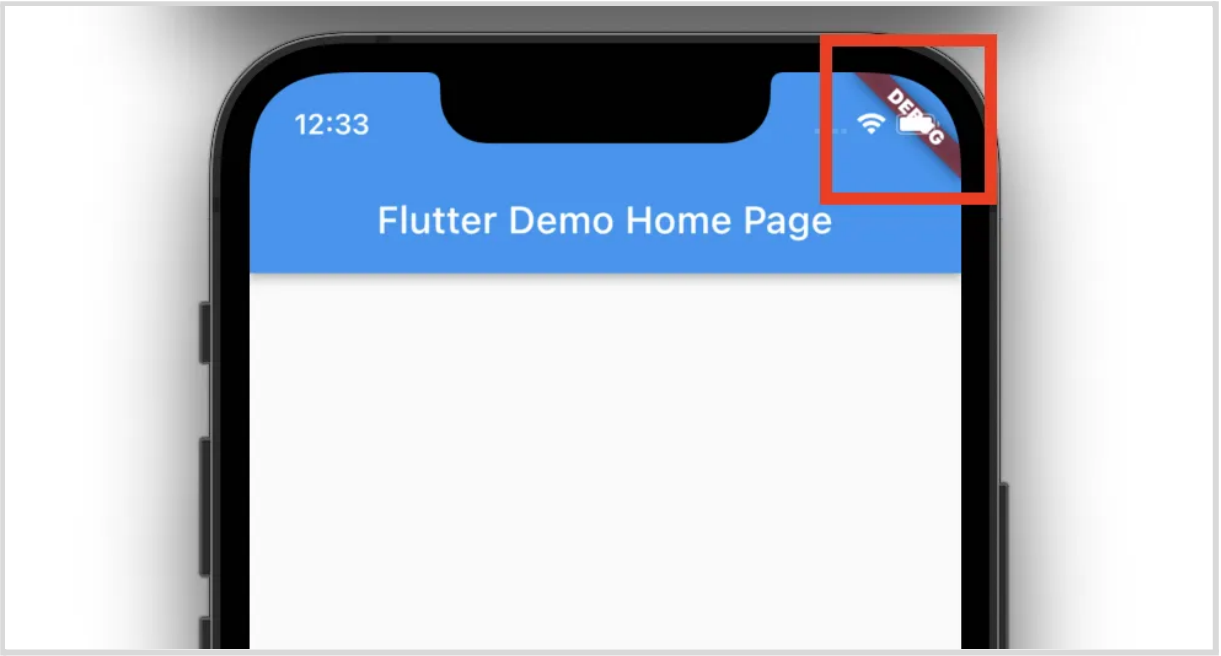
Flutter et Dart supportent les asserts sous cette forme: “assert(condition, “message à afficher”)”. Pour plus d’information se référer à ce guide dans la documentation officielle.

Unit Testing

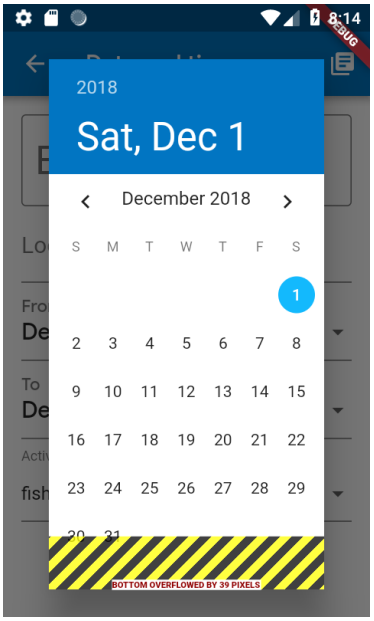
Flutter supporte également les tests unitaires, autant sur l’interface que sur les données. Pour se faire, il suffit d’écrire ses tests dans les fichiers contenus dans le dossier test (*“\src\smart_split\test”*).
Ici encore, le langage et framework étant toujours sujet à changement, je préfère renvoyer vers la documentation officielle au cas où l’implémentation des tests serait modifiée dans le futur: documentation.

Debug Mode

Debug Mode est l’affichage par défaut de l’application durant son développement et la version à laquelle vous serez le plus confronté en tant que développeur.
Ce mode est tout de suite reconnaissable à la bannière présente en haut à droite:



Cet affichage de testing affiche également les overflows de l'interface et les erreurs. Ces éléments ne seront plus affichés une fois SmartSplit compilé afin de ne pas déranger l'utilisateur dans le cas où il resterait des overflows et autres erreurs que la bonne pratique recommanderait de corriger (il n'y a aucune erreur dans la version actuelle).



Dans cet exemple, le calendrier overflow en bas de l'écran, causant l'apparition de la bannière jaune et noire.

Compilation et déploiement

Une fois le projet terminé, vous pouvez créer une version exécutable pour votre OS grâce à la commande "flutter build x" où x peut être windows, macos ou linux en fonction de votre OS. Pour plus d'informations voici la documentation pour le support desktop de Flutter.
Vous pouvez également créer une version web (attention, certains des plugins ne fonctionnent pas sur web!) grâce à la commande "flutter build web". Se référer à cette documentation web pour plus d'informations.

Une fois une version desktop créée, naviguez jusqu'au dossier respectif (cf documentation) pour trouver le .exe et les dossiers du programme, directement exécutable et déplaçable. Pour la version web, le processus est similaire mais avec des dossiers .html (cf documentation). La documentation détaille plusieurs manières de lancer un serveur (j'ai personnellement utilisé python avec la commande "python -m http.server 8000").

Architecture

Backend et Frontend

Le frontend et backend sont mélangés (tout est conservé et manipulé côte client), la base de données en elle-même est gérée par le plugin Hive (cf présentation des plugins utilisés) et les modèles sont répertoriés dans le dossier "*Data*".

Dossiers

Le projet est séparé en plusieurs dossiers contenant chacun des fichiers ".dart", ces fichiers dart contiennent le code de l'application (widgets, aspect logique et modèle de base de données) et sont groupés ensemble par dossier en fonction de leurs rôles.

Dossier principal

Le dossier principal contient les sous-dossiers "Data", "Utils" et Widgets" en plus de contenir le fichier *"main.dart"* qui est à la base de l'application, faisant appel aux autres fichiers et widgets.

Data

Le dossier data contient les fichiers liés aux modèles stockés dans la base de donnée.
Il existe un fichier ".dart" et un fichier ".g.dart" pour chaque objet de la base de donnée.
Le fichier ".dart" est l'implémentation en Dart de la classe et "extends" toujours HiveObject ce qui nous permet de générer le fichier ".g.dart". Pour plus d'information sur les objets Hive, se référer à la documentation du plugin.

Utils

Utils contient quelques widgets que je réutilise souvent, *"decorations.dart"* contient mon widget pour afficher les barres de séparation tandis que *"dialog_utils.dart"* contient les fenêtres pop-up s'affichant lorsque l'utilisateur clique sur certains boutons ou effectue certaines actions.

Widgets

Widgets contient un dossier pour chaque onglet principal de SmartSplit, nommément "File Manager", "Project Manager" et "Student Manager". Chaque dossier contient à son tour les fichiers ".dart" correspondants à l'onglet et à l'affichage de ses widgets.

Pistes d’amélioration et autre

Flutter fonctionne avec une gestion des états et l’interface est uniquement composée de widgets imbriqués les uns dans les autres. Je recommande donc vivement de s’entraîner et de comprendre ces concepts avant de modifier SmartSplit. Si vous êtes déjà un programmeur chevronné dans l’utilisation de Flutter et/ou des états, je pense qu’il est également possible d’améliorer SmartSplit et de simplifier certaines parties du programme (tout particulièrement les premiers widgets que j’ai implémenté étant donné que certains ne respectent pas spécialement les règles de “bonne programmation”). Cependant le code ne devrait pas poser énormément de problèmes à comprendre et maîtriser et les améliorations envisageables ne se limitent qu’aux modèles de la base de données (avoir plusieurs boîtes Hive au lieu d’une seule), à la factorisation des styles et à l’utilisation d’un autre système de gestion des états.