Documentation:

------

Structure des dossiers:

Dossier

* Assets: contient les fonts, les images et les fichiers .json
* Lib : contient les dossiers du projet
  + Database : dossier contenant les fichier pour manipuler une base de donnés SQFLITE de type SQL sur le mobile, contient les fonctions standard CRUD (Create Read Update Delete)  
    une base de données Hive NOSQL (c’est celle qu’on utilise car c’est la plus rapide)
  + Model : Plusieurs modèle d’objet utilisés dans le l’application (Alert, User, Scheduledmsg etc.) Quelques modèles proposent des fonctionnalités comme la conversion des données du modèle en json ou le décodage du format json au format du modelé.  
    Hive database est utilisé dans ce projet car c’est un moyen optimal d’utiliser une base de données NOSQL plus rapide que SQFLITE et SharedPreferences dans l’écriture et la lecture de données
  + Pages : Contient toute les pages de l’application
  + Utils : Utilisé essentiellement pour la sauvegarde des donnes d’un utilisateur sur SharedPreferences mais susceptible d’être modifier selon les besoins
  + Widget : Contient les widgets utilisés dans les pages tels que la bar de navigation inférieur et supérieur, le logo etc.

Les Pages :

* Home\_page.dart :  
  Cette page contient  
  La bar de navigation supérieur avec un menu de type burger qui se superpose a l’écran lorsqu’on clique dessous

Le logo qui occupe 25% de l’écran  
Bouton message programmé qui nous renvoie vers la page des messages programmés

Bouton message automatique qui nous renvoie vers la page des messages auto

La bar de navigation inférieure qui permet de naviguer de la page d’accueil a la page des rapports, la page d’aide, et la page des paramètres

* Sms\_prog\_page.dart :  
  Cette page contient  
  La bar de navigation supérieur avec un menu de type burger qui se superpose a l’écran   
  le titre de la page, le bouton d’ajout d’alertes  
  un message qui est affiché si on ne possède aucune alerte   
  La bar de navigation inférieure qui permet de revenir à la page d’accueil  
  Dans ce fichier on trouve les fonctions
* sendSms() :  
  Vérifie si le message possède l’option de confirmation avant envoi
* updateDate(Scheduledmsg\_hive msg) :   
  Prend en paramètre un message et met à jour la date d’envoie du message programmé en fonction de la récurrence choisi (tout les ans, tout les mois et
* Bool canBeSent(Scheduledmsg\_hive msg) :  
  Retourne vrai si l’heure actuelle est supérieure ou égale a l’heure programmé pour envoi
* confirmSend(Scheduledmsg\_hive msg) :  
  Créer un pop up pour demander si l’utilisateur souhaite envoyer le msg si non un délai de 5 minutes est ajouter a l’heure d’envoie
* buildListOfMsg(List<Scheduledmsg\_hive> messages):  
  Prend en paramètre la liste de messages de type scheduledmsg\_hive et construit la liste de messages avec ListView.Builder
* buildMsg(BuildContext context, Scheduledmsg\_hive message):  
  Créer le modèle pour un message
* buildButtons(BuildContext context, Scheduledmsg\_hive message):  
  Créer les boutons affichés dans le message (modifier supprimer)
* buildPopupDialog(Scheduledmsg\_hive message) :  
  Créer un pop up pour demander si l’utilisateur souhaite supprimer ou non l’alerte
* formulaire\_alerte\_prog\_page.dart :  
  Cette page contient le formulaire de création pour une alerte programmé, il est accessible lorsqu’on clique sur ajouter une alerte dans la page des messages programmés  
  La bar de navigation supérieure avec le titre de la page  
  un bouton retour   
  les champs de texte pour :

Nom de l’alerte,  
 Numéro du contact avec icone à droite pour aller sélectionner des contacts du téléphone  
 Le message à envoyer  
La sélection de la date et de l’heure  
La sélection de la récurrence  
L’option compte à rebours  
L’option de confirmation avant envoi  
L’option de notification lorsqu’un message est envoyé  
Dans ce fichier on trouve les fonctions

* saveToHive() :

Permet de sauvegarder le message sur notre box intitulé ‘scheduledmsg’ de hive database si tous les champs sont remplis correctement

* buildTextField(String labelText, String placeholder, TextEditingController controller, int nbLines) :  
  Contruit un champ pour rentrer du texte avec le nom du champ(labeltext), le texte d’indice(placeholde) le controleur de ce champ(controller) et le nombre de lignes du champ
* buildDatePicker() :  
  Construit un sélectionneur de date avec le format de la librairie Cupertino, date initiale étant la date actuelle, format 24h
* buildRepeatOptions() :  
  Construit un sélectionneur de récurrence avec la librairie Cupertino selon les options de repetition
* showSnackBar(BuildContext context, String s):  
  Montre un message sur le pied de la page pendant une courte durée, utilisé par exemple pour notifier l’utilisateur que tous les champs n’ont pas été remplis
* edit\_scheduledmsg\_page.dart :  
  Cette page contient les mêmes champs que la formulaire avec les informations d’un message sauvegarder   
  Nous pouvons faire les modifications et sauvegarder grâce la fonction
* saveChanges() :  
  Permet de sauvegarder les modifications des champs qui sont lu à l’aide des contrôleur de texte sur la box ‘scheduledmsg’

Le dossier widget contient un ensemble de widget réutilisables selon nos besoin dans nos pages, pour ce faire nous n’avons qu’à l’importer et l’appeler avec les paramètres nécessaires

* appbar\_widget.dart :  
  widget responsable pour la bar de navigation supérieure

Pour appeler ce widget

* TopBar(String title) :   
  Retourne in widget avec le titre donné en paramètre
* boxes.dart :  
  class responsable de la récupération des boxes de hive database  
  On trouve la méthode getScheduledmsg() qui peut être utilisé pour récupérer tous les messages sauvegarder sur la boîte ‘scheduledmsg’ example (final allMessages = Boxes.getScheduledmsg() )
* button\_widget.dart :  
  widget responsable pour la création d’un bouton avec un titre et une fonction sur le clique

Pour utilise ce widget

* ButtonWidget(String text, VoidCallback onClicked) :  
  permet d’afficher un button avec le titre donné en paramètre et la fonction à exécuter sur le clique donné également en paramètre
* hamburgermenu\_widget.dart :  
  widget responsable pour la création d’un bouton de type burger  
  le menu avec les options compte, rapport, paramètres et aide s’affichent lorsqu’on clique  
  et nous redirectionne vers l’une des pages sélectionnée

Pour utilise ce widget

* HamburgerMenu() :  
  permet d’afficher un button de type burger  
  widget de type drawer
* logo\_widget.dart :  
  widget responsable pour la création du logo avec une taille fixe de 25% de l’écran

Pour utilise ce widget

* Logo() :  
  permet d’afficher un containeur avec le logo a l’intérieure
* navbar\_widget.dart :  
  widget responsable pour la création des bar de navigation inférieure  
  ce fichier comporte les différentes bar de navigation utilisés dans l’application

Pour utilise ce widget

* BottomNavigationBarSmsAutoTwo() :
* BottomNavigationBarSmsProgTwo() :
* BottomNavigationBarSection() :
* profile\_widget.dart :  
  widget responsable pour la création de l’image du profile avec l’icone pour modification

Pour utilise ce widget

* ProfileWidget(String imagePath, bool isEdit, VoidCallback onClicked) :  
  permet de superposes plusieurs widgets sur les autres pour créer l’effet d’un bouton de modification sur l’image
* logo\_widget.dart :  
  widget responsable pour la création du logo avec une taille fixe de 25% de l’écran

Pour utilise ce widget

* Logo() :  
  permet d’afficher un containeur avec le logo a l’intérieure