Documentation:

PS : mettre à jour le projet au fur et a mesure et corriger les fautes que vous trouverez.

------

Structure des dossiers:

Dossier

* Assets: contient les fonts, les images et les dictionnaires et les fichiers .json
* Lib : contient les dossiers du projet
  + Database : dossier contenant les fichier pour une base de données Hive NOSQL et les fichiers automatiquement générés par hive. Il existe également une base de données SQLITE et SharedPreferences. 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

Nous avons plusieurs tables/boxes sur hive

TABLES : ‘scheduledmsg’ qui possède des objets de type Scheduledmsg\_hive,   
‘rapportmsg’ qui possède des objets de type Rapportmsg\_hive,   
‘user’ qui possède des objets de type User\_hive,  
‘group’ qui possède des objets de type GroupContact,   
‘alert qui possède des objets de type Alert,

‘alertkey’ qui possède des objets de type AlertKey.

Chacune d’entre elles possèdent des champs spécifiques que l’on a créer selon nos besoins (c.f. fichier hive\_database.dart)

* + Model : Plusieurs modèle d’objet utilisés dans l’application (couleurs ,notifications, expressions)   
      
    \*(Avec SharedPreferences : Quelques modèles proposent des fonctionnalités comme la conversion des données du modèle en json ou le décodage du format json au format du modele.)\*
  + Pages : Contient toute les pages de l’application
  + Utils : Utilisé essentiellement pour la sauvegarde des donnes et certaines fonctions utilisés à travers le programme  
    Contient la classe fonctions avec la fonction pour montrer une notification sur le pied de la page  
    Contient la classe boxes qui propose les fonctions d’accès aux boxes hive  
    (getScheduledmsg, getRapportmsg, getGroupContact, getAutoAlert, getAlertKey, getUser..)
  + Widget : Contient les widgets utilisés dans les pages tels que la bar de navigation inférieur et supérieur, le logo, le burger menu, la photo du profil etc.

Pubspec.yaml contient les dépendances, les packages utilisés, la déclaration des ressources utilisés dans le l’application comme les images, font etc.

Les Pages :

* accueil\_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 20% 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 bouton retour sur la page d’accueil

Le bouton pour ajouter une nouvelle alerte   
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 et changer entre les pages de message programmé ou message automatique  
Dans ce fichier on trouve les fonctions

* sendSms() :  
  Vérifie si le message possède des options avant envoi
* send() :  
  Envoi un message avec la librairie telephony
* 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, aucune etc.)
* 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
* saveMsgToRappor(Scheduledmsg\_hive message) :  
  sauvegarde dans le box des rapport un message
* 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 le formulaire avec les informations d’un message sauvegarder   
  Nous pouvons faire les modifications et sauvegarder si elles sont correctes 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 boites/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 des widgets sur les autres pour créer le bouton d’ajout 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