IMT Nord Europe
École Mines-Télécom
IMT-Université de Lille

« Module Mobile »

« Développement d'une application mobile »

Projet « Cht'i Face Bouc »

Par Didier JUGE-HUBERT

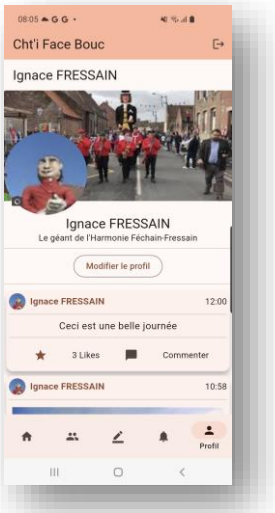
 Version 1.1/2025

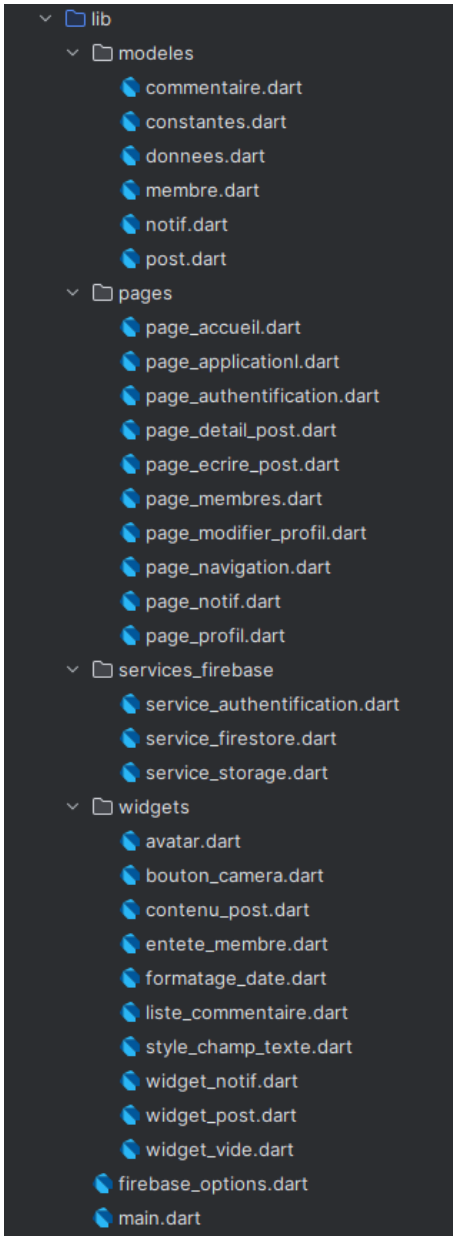
Sommaire

Introduction

Parmi les applications mobiles phares de ces dernières années, nous pouvons citer les réseaux sociaux.

Dans cette application, vous allez essayer de reproduire cette application :





REMARQUE IMPORTANTE : Dans votre projet, je veux retrouver la même structure de fichiers et les mêmes noms de fichiers.

1 Etape 1 : Création d'un projet Firebase et d'un projet Flutter

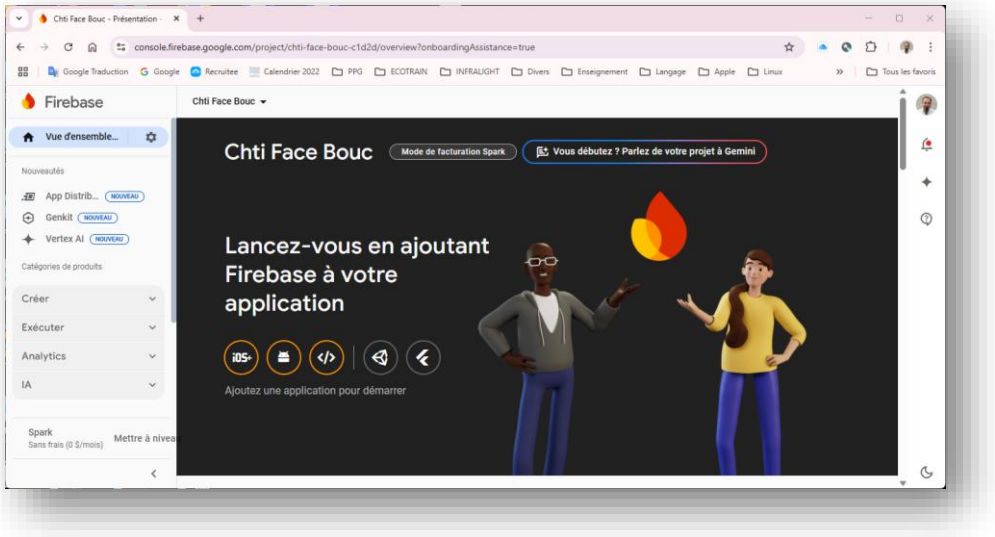
1.1 Création d'un projet Firebase

A partir de votre compte Google et à l'aide de la console Firebase (https://console.firebase.google.com/), vous allez créer un nouveau projet Firebase nommé « *Chti Face Bouc* » sans *Google Analytics*.

Dans votre projet Firebase, vous allez ajouter les services suivants :

- Authentication
- Firestore Database
- Storage

Vous devez obtenir l'affichage ci-dessous :



1.2 Création d'un projet Flutter

Sur votre machine, vous créez un nouveau projet Flutter nommé « *chti_face_bouc* » pour votre cible Android ou IOS suivant votre smartphone ou simulateur. Simplifiez le projet créé en vous inspirant du code ci-joint et en structurant votre projet (page d'accueil dans un sous-répertoire « *pages* ») :

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

import 'pages/page_accueil.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Chti Face Bouc',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.brown),
        useMaterial3: true,
      ),  // ThemeData
      debugShowCheckedModeBanner: false,
      home: const PageAccueil(title: "Cht'i Face Bouc"),
    );  // MaterialApp
  }
}
```

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

class PageAccueil extends StatefulWidget {
  const PageAccueil({super.key, required this.title});
  final String title;
  @override
  State<PageAccueil> createState() => _PageAccueilState();
}

class _PageAccueilState extends State<PageAccueil> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Theme.of(context).colorScheme.inversePrimary,
        title: Text(widget.title),
      ),  // AppBar
      body: const Center(
        child: Text("Cht'i Face Bouc"),
      ),  // Center
    );  // Scaffold
  }
}
```

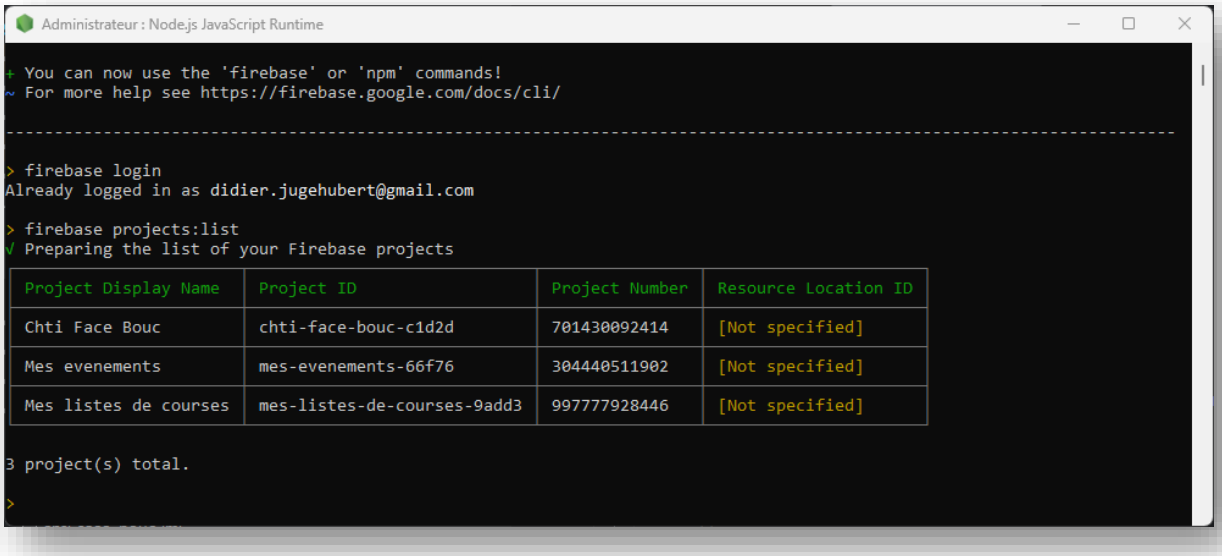
1.3 Configuration du projet Firebase à l'aide de CLI

Retournez sur la console Firebase (https://console.firebase.google.com/), sur votre projet Firebase « *Chti Face Bouc* », cliquez sur « Ajouter Firebase à votre application Flutter » (encadrer en rouge)

1.3.1 Installation du CLI Firebase

Si cela n'a pas déjà été fait, est d'installer le « *CLI Firebase* » sur votre machine via le lien ci-dessous qui est sur la page : https://firebase.google.com/docs/cli?hl=fr

N'oubliez pas à la fin de l'installation du CLI Firebase et dans celui-ci, de vous connecter à Firebase (`firebase login`) et de tester la connexion (`firebase projets:list`). Vous devriez obtenir un affichage équivalent à ci-dessous :



Vous pouvez passer au point suivant.

1.4 Configuration du projet Flutter

Retournez dans votre éditeur Flutter, dans votre projet sur le terminal, exécutez la commande suivante pour installer et activer le *FlutterFire_CLI* :

```
dart pub global activate flutterfire_cli
```

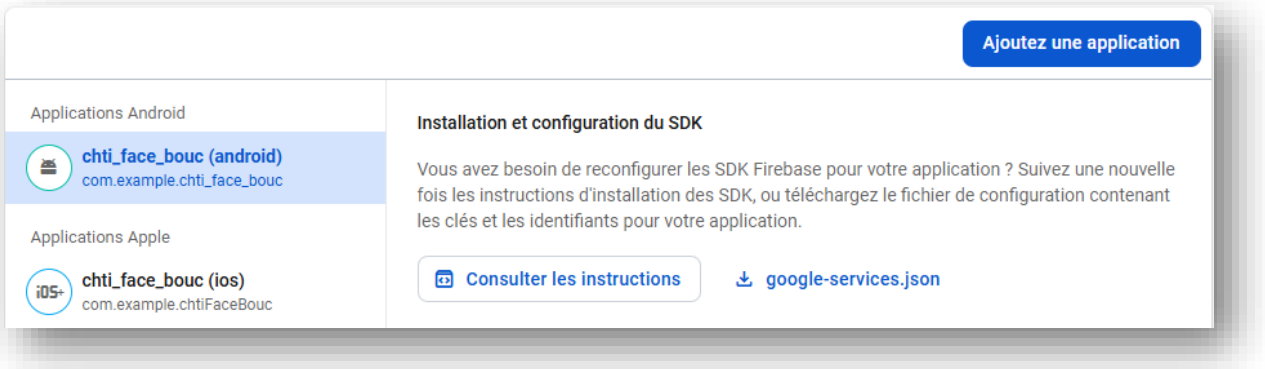
Puis, toujours dans le terminal, configurez votre projet Flutter pour utiliser Firebase avec la commande suivante :

```
flutterfire configure --project=<id de votre projet>
```

Lors de l'exécution de cette commande, vous aurez à choisir la plateforme pour laquelle vous développez. Vous choisissez votre plateforme (android ou ios ou les deux) à l'aide des flèches et de la barre espace pour sélectionner, puis validez par « *entrée* ».

Après exécution de la commande précédente, vous devriez :

- dans votre répertoire lib de votre projet, un nouveau fichier *firebase_options.dart*.
- dans votre projet Firebase, une ou deux applications suivant les OS sélectionnés.



Vous devez ensuite ajouter les packages Flutter dont vous allez avoir besoins par les commandes suivantes :

```
flutter pub add firebase_core
flutter pub add firebase_auth
flutter pub add cloud_firestore
flutter pub add firebase_storage
flutter pub add image_picker
flutter pub add intl
```

Modifiez votre *main* de la manière suivante,

a) importez le fichier de firebase_core.dart
b) importez le fichier avec vos options firebase
c) passez le main en fonction asynchrone
d) ajoutez la ligne de code pour attendre initialisation de votre application
e) ajoutez la ligne de code pour initialiser la connexion à Firebase.

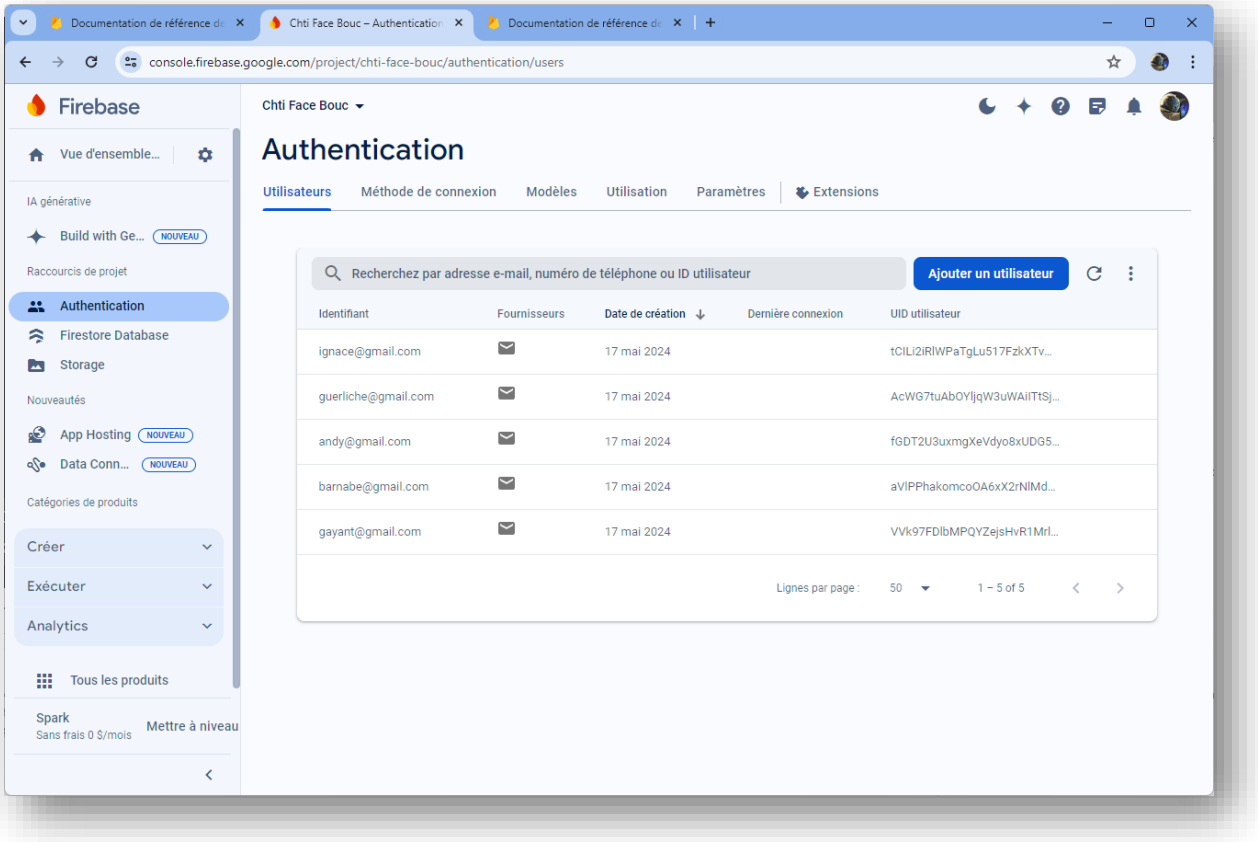
2 Etape 2 : Authentification sous Firebase

2.1 Créer le fournisseur d'accès et des utilisateurs

Retournez sur la console Firebase (https://console.firebase.google.com/), sur votre projet Firebase « *Chti Face Bouc* », cliquez sur « *Authentication* » dans les raccourcis de projet.

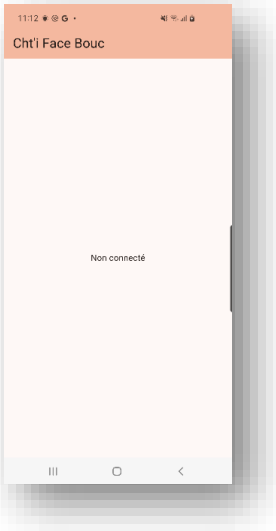
Choisissez le fournisseur « *Adresse e-mail/Mot de passe* » puis cliquez sur « Enregistrer » sans activer le choix « *Lien envoyé par e-mail* ».

Allez sur l'onglet « *Utilisateurs* », puis créez quelques utilisateurs.



2.2 Vérification est-ce que l'authentification fonctionne ?

Vous allez commencer à mettre en œuvre authentification. Pour cela, vous allez ajouter à votre page d'accueil un *StreamBuilder* qui va tester l'état de la connexion avec Firebase.



Pour cela, vous devez importer le package '*firebase_auth.dart*', puis remplacer le code de votre paramètre body par le code ci-dessous :

```
body: StreamBuilder<User?>(
  stream: FirebaseAuth.instance.userChanges(),
  builder: (BuildContext context, AsyncSnapshot snapshot) {
    return (snapshot.hasData)
        ? const Center(child: Text("Connecté"))
        : const Center(child: Text("Non connecté"));
  },
), // StreamBuilder
```

Vous devez obtenir à l'exécution l'écran ci-contre.

2.3 Création de la classe service authentification

Maintenant que l'authentification est validée, vous devez créer une nouvelle page qui va permettre de s'authentifier. Pour cela, vous allez travailler avec des services dédiés à Firebase. Dans le répertoire « *lib* », vous pouvez créer un sous-répertoire « *services_firebase* » qui contiendra tous les fichiers dart gérant les services Firebase.

Le premier service à créer est ***service_authentification*** au travers d'un fichier dart : *service_authentification.dart.* Dans ce fichier vous allez créer une classe *ServiceAuthentification* important le package *firebase_auth.dart* et définissant la propriété et les méthodes ci-dessous qui seront à compléter :

```
import 'package:firebase_auth/firebase_auth.dart';

class ServiceAuthentification {
  //Récupérer une instance de auth
  final instance = FirebaseAuth.instance;

  // Connecter à Firebase
  Future<String?> signIn({required String email, required String password}) async {
    String result = "";
    return result;
  }

  // Créer un compte sur Firebase
  Future<String?> createAccount({
    required String email,
    required String password,
    required String surname,
    required String name
  }) async {
  }

  // Déconnecter de Firebase
  Future<bool> signOut() async {
    bool result = false;
    return result;
  }

  // Récupérer l'id unique de l'utilisateur
  String? get myId => instance.currentUser?.uid;

  // Voir si vous etes l'utilisateur
  bool isMe(String profileId) {
    bool result = false;
    return result;
  }
}
```

2.4 Création d'une page authentification

Maintenant que nous avons écrit les méthodes précédentes, nous allons les appeler dans une nouvelle page. Pour organiser votre travail, dans le répertoire « *pages* » dans *lib,* vous allez créer un nouveau fichier dart « *page_authentification.dart* » qui contiendra une classe StateFullWidget « *PageAuthentification* ».

Dans cette classe vous ajouterez :

a) Les variables suivantes
   a. *bool accountExists :* détermine si le compte existe ou pas
   b. *TextEditingController mailController* : champ texte pour la saisie du mail
   c. *TextEditingController passwordlController* : champ texte pour la saisie du mot de passe
   d. *TextEditingController surnameController* : champ texte pour la saisie du prénom
   e. *TextEditingController nameController* : champ texte pour la saisie du nom

b) Les méthodes *initState()* et *dispose()* pour les controlleurs précédents

c) Dans la méthode *build(BuildContext context)*

```
Scaffold
   body: SafeArea
      SingleChildScrollView
         Column
            Image (logo)
            SegmentedButton
            Card
               Container
               Column
                  TextField (adresse mail)
                  TextField (mot de passe)
                  TextField (prénom)
                  TextField (nom)
```

d) Une fonction *_onSelectedChanged* qui sera appelée par le bouton *SegmentedButton*

```
_onSelectedChanged(Set<bool> newValue) {
  setState(() {
    accountExists = newValue.first;
  });
}
```

e) Une fonction *_handleHauth* qui en fonction de l'existence ou non du compte *accountExists* permettra de se connecter *signIn* ou de créer un compte avec *createAccounts* de la librairie *service_authentification.dart.*

REMARQUE : Pour vous déconnecter, vous pouvez appeler dans votre main la méthode *signOut* de la librairie *auth_service.dart.*

3 Etape 3 : Création d'une base de données sous Firestore

La gestion des utilisateurs dans Firebase n'est pas simple. De plus, nous devons pouvoir gérer une liste de membres de notre réseau social. Pour pallier à cela, nous allons créer une base de données qui nous permettra de gérer les utilisateurs de notre réseau social, les posts et les notifications avec des informations complémentaires (photo, nom, prénom, …).

3.1 Création d'une base de données Firestore Database dans Firebase

Retournez sur la console Firebase (https://console.firebase.google.com/), sur votre projet Firebase « *Chti Face Bouc* », cliquez sur « *Firestore Database* » dans les raccourcis de projet.

Vous devez créer une nouvelle base de données avec les caractéristiques suivantes :

- Mode production
- Zone de stockage : europe (eur3)

Comme nous sommes en mode production, vous devez rajouter une règle dans l'onglet « Régles ». La règle a créé accorde l'accès si l'on est authentifié (voir copie écran ci-dessous).

N'oubliez pas de « ***Publier*** » votre nouvelle règle avant de sortir.

3.2 Création des modèles d'accès aux données

Je vous propose de créer un fichier *constantes.dart* dans un nouveau répertoire *modeles* qui contiendra tous les noms des collections et des champs de Firestore. Je vous donne le fichier *constantes.dart* ci-après :

```
// Collection
String memberCollectionKey = "members";
String postCollectionKey = "posts";
String commentCollectionKey = "comments";
String notificationCollectionKey = "notifications";

// Member
String memberIdKey = "memberID";
String nameKey = "name";
String surnameKey = "surname";
String profilePictureKey = "profilePicture";
String coverPictureKey = "coverPicture";
String descriptionKey = "description";
```

```dart
// Post
String textKey = "text";
String postImageKey = "image";
String dateKey = "date";
String likesKey = "likes";

// Notif
String fromKey = "from";
String isReadKey = "read";
String postIdKey = "postID";
```

A partir de ce fichier, vous allez définir un premier fichier dart gérant les information utilisateurs. Vous ne pourrez pas appeler cette classe *User* car Firebase défini déjà une classe *User*. Je vous propose de l'appeler *Membre* afin d'éviter les conflits de nom dans le fichier *membre.dart* dans le répertoire *modeles*. Vous trouverez ci-dessous le contenu de ce fichier :

```dart
import 'package:cloud_firestore/cloud_firestore.dart';

import 'constantes.dart';

class Membre {
  DocumentReference reference;
  String id;
  Map<String, dynamic> map;

  Membre({required this.reference, required this.id, required this.map});

  String get name => map[nameKey] ?? "";
  String get surname => map[surnameKey] ?? "";
  String get profilePicture => map[profilePictureKey] ?? "";
  String get coverPicture => map[coverPictureKey] ?? "";
  String get description => map[descriptionKey] ?? "";
  String get fullName => "$surname $name";
}
```

3.3 Création de la classe service storage

Maintenant que le modèle de données pour les membres est créé, vous devez créer une nouvelle classe pour gérer les accès à *FirebaseStorage*. Pour cela vous allez créer dans le répertoire *services_firebase* un fichier *service_storage.dart* qui fournira une classe *ServiceStorage*.

Cette classe assurera la sauvegarde des images de l'application dans la zone de stockage de Firebase et aura une méthode *addImage* qui permettra de sauvegarder une image. Ce fichier aura comme base :

```dart
import 'dart:io';
import 'package:firebase_storage/firebase_storage.dart';

class ServiceStorage {
  static final instance = FirebaseStorage.instance;
  Reference get ref => instance.ref();

  Future<String> addImage({required File file, required String folder, required String userId, required String imageName,}) async {
    final reference = ref.child(folder).child(userId).child(imageName);
    UploadTask task = reference.putFile(file);
    TaskSnapshot snapshot = await task.whenComplete(() => null);
    String imageUrl = await snapshot.ref.getDownloadURL();
    return imageUrl;
  }
}
```

3.4 Création de la classe service firestore

Maintenant que le modèle de données pour les membres est créé, vous devez créer une nouvelle classe pour gérer les accès à *Firestore*. Pour cela vous allez créer dans le répertoire *services_firebase* un fichier *service_firestore.dart* qui fournira une classe *ServiceFirestore*.

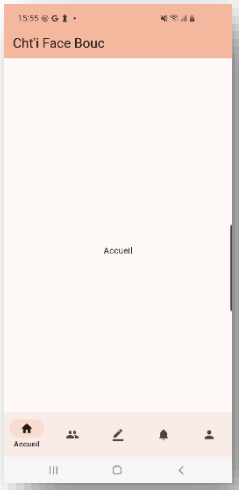
Cette classe assurera l'accès à *Firestore* et des méthodes *addMember* et *updateMember* qui permettront d'ajouter ou de modifier un membre dans notre collection. Ce fichier aura comme base :

```dart
import 'dart:io';
import 'package:cloud_firestore/cloud_firestore.dart';
import '../modeles/constantes.dart';
import 'service_storage.dart';

class ServiceFirestore {

  //Accès a la BDD
  static final instance = FirebaseFirestore.instance;

  //Accès spécifique collection
  final firestoreMember = instance.collection(memberCollectionKey);

  //Ajouter un membre
  addMember({required String id, required Map<String, dynamic> data}) {
    firestoreMember.doc(id).set(data);
  }

  // Mettre à jour un membre
  updateMember({required String id, required Map<String, dynamic> data}) {
    firestoreMember.doc(id).update(data);
  }

  // stockage et mise à jour d'une image
  updateImage(
      {required File file,
        required String folder,
        required String memberId,
        required String imageName}) {
    ServiceStorage()
        .addImage(
        file: file, folder: folder, memberId: memberId, imageName: imageName)
        .then((imageUrl) {
      updateMember(id: memberId, data: {imageName: imageUrl});
    });
  }
}
```

3.5 Appel de la méthode addMember

Dans la classe authentification, vous allez modifier la méthode *createAccount* pour appeler, avec les bons paramètres la méthode *addMember*.

4 Etape 4 : Page Navigation



Dans cette partie vous allez créer la structure de navigation dans votre application. Pour cela vous devez créer un nouveau *StateFull* widget dans un fichier *page_navigation.dart*.

Je vous propose de créer des widgets qui ne contiendront rien et qui pourront être réutilisés sur plusieurs pages. Vous faites cela dans un dossier *widgets* sous *lib* puis dans un fichier *widget_vide.dart*.

```dart
class EmptyBody extends StatelessWidget {
  const EmptyBody({super.key});

  @override
  Widget build(BuildContext context) {
    return const Center(
      child: Text('Aucune données'),
    );  // Center
  }
}

class EmptyScaffold extends StatelessWidget {
  const EmptyScaffold({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(),
      body: const EmptyBody(),
    );  // Scaffold
  }
}
```

Dans la méthode la classe *_PageNavigationState :*

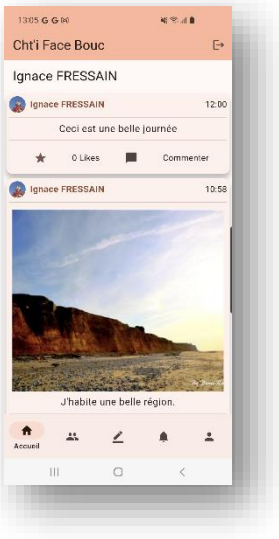
a) La variable suivante
   a. *Int index = 0 :* entier index de la page afficher
b) Dans la méthode *build(BuildContext context)*

```dart
@override
Widget build(BuildContext context) {
  final memberId = ServiceAuthentification().myId;
  return (memberId == null)
      ? const EmptyScaffold()
      : StreamBuilder<DocumentSnapshot>(
      stream: ServiceFirestore().specificMember(memberId),
      builder: (BuildContext context, AsyncSnapshot<DocumentSnapshot> snapshot) {
        if (snapshot.hasData) {
          final data = snapshot.data!;
          final Membre member = Membre(
              reference: data.reference,
              id: data.id,
              map: data.data() as Map<String, dynamic>
          );  // Membre
          List<Widget> bodies = [
            const Center(child: Text("Accueil")),
            const Center(child: Text("Membres")),
            const Center(child: Text("Ecrire un post")),
            const Center(child: Text("Notifications")),
            const Center(child: Text("Profil")),
          ];
          return Scaffold(
            appBar: AppBar(
              title: Text(member.fullName),
            ),  // AppBar
            bottomNavigationBar: NavigationBar(
              labelBehavior:
              NavigationDestinationLabelBehavior.onlyShowSelected,
              selectedIndex: index,
              onDestinationSelected: (int newValue) {
                setState(() {
                  index = newValue;
                });
              },
```

```
            destinations: const [
              NavigationDestination(
                icon: Icon(Icons.home),
                label: "Accueil",
              ),  // NavigationDestination
              NavigationDestination(
                icon: Icon(Icons.group),
                label: "Membres",
              ),  // NavigationDestination
              NavigationDestination(
                icon: Icon(Icons.border_color),
                label: "Ecrire",
              ),  // NavigationDestination
              NavigationDestination(
                icon: Icon(Icons.notifications),
                label: "Notification",
              ),  // NavigationDestination
              NavigationDestination(
                icon: Icon(Icons.person),
                label: "Profil",
              ),  // NavigationDestination
            ],
          ),  // NavigationBar
          body: bodies[index],
        );  // Scaffold
      } else {
        return const EmptyScaffold();
      }
    }
  );  // StreamBuilder
}
```

5 Etape 5 : affichage de tous les posts sur la page d'accueil

Pour rappel, un post est constitué de plusieurs éléments comme nous l'avons vu dans la définition de la classe modèle « *post.dart* » :



- Un auteur
- Une date
- Un texte
- Une photo
- Un nombre de likes

Sur la page d'accueil, nous allons afficher tous les posts du plus récent au plus ancien.

5.1 Modification du service Firestore

Pour pouvoir récupérer tous les posts, vous devez créer dans le service Firestore, une nouvelle méthode *allPosts*() en vous inspirant du code ci-dessous :

```
// Lire la liste de tous les posts
allPosts() => firestorePost.orderBy(dateKey, descending: true).snapshots();
```

5.2 Création de la page d'accueil

Maintenant que nous avons écrit la méthode précédente, nous allons l'appeler dans une nouvelle page. Pour organiser votre travail, dans le répertoire « *pages* » dans *lib,* vous allez créer un nouveau fichier dart « *page_accueil.dart* » qui contiendra une classe StateLessWidget « *PageAccueil* ».

Vous appellerez cette page à partir de la page de navigation dans l

Cette page, comme les précédentes, est réalisée autour d'un *StreamBuilder* pour la créer.

Cette page pourra être basée sur la structure suivante :

```
StreamBuilder
   stream: allPosts
   builder: (BuildContext context,
AsyncSnapshot<QuerySnapshot> snapshot) {
      return (snapshot.hasData)
         ? ListView.builder
             itemCount: doc.length,
             itemBuilder: (context, index) {
               return WidgetPost(post: post);
             },
         : EmptyBody()
```

5.3 Appel de la page d'accueil

Pour afficher cette page, vous appellerez à partir de la page navigation dans le 1er bouton en remplacement du widget *Text(« Accueil »)*.

6 Etape 6 : La page de profil

6.1 Création du modèle de données et de l'accès pour les posts

Vous allez avoir besoin de créer un modèle de données pour les posts avec les identifiants des champs correspondants dans le répertoire *modeles* dans un fichier *post.dart*.

```dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'constantes.dart';

class Post {
  DocumentReference reference;
  String id;
  Map<String, dynamic> map;

  Post({required this.reference, required this.id, required this.map});

  String get member => map[memberIdKey] ?? "";
  String get text => map[textKey] ?? "";
  String get imageUrl => map[postImageKey] ?? "";
  int get date => map[dateKey] ?? 0;
  List<dynamic> get likes => map[likesKey] ?? [];
}
```

6.2 Modifier le service Firestore

Dans le fichier *service_firestore.dart*, vous devez créer le chemin d'accès à la collection des posts *firestorePost* :

```dart
  // Accès a la BDD
  static final instance = FirebaseFirestore.instance;

  // Accès spécifique aux collections
  final firestoreMember = instance.collection(memberCollectionKey);
  final firestorePost = instance.collection(postCollectionKey);
```

Et vous devez ajouter une méthode *postForMember(String id)* permettant de récupérer tous les posts d'un membre. Vous pouvez vous inspirer du code ci-dessous :

```dart
// Lire des posts d'un utilisateur
postForMember(String id) => firestorePost.where(memberIdKey, isEqualTo: id).snapshots();
```

6.3 Création du fond de page

Dans cette partie vous allez créer la page de profil dans votre application. Pour cela vous devez créer un nouveau *StateFul* widget dans un fichier *page_profil.dart*. Cette page, comme la précédente, est réalisée autour d'un *StreamBuilder* pour créer la page. Une fois que vous avez créer votre page, vous l'appellerez à partir de la page de navigation en modifiant la ligne correspondante dans la liste de widgets *bodies :*

```dart
          List<Widget> bodies = [
            const Center(child: Text("Accueil")),
            const Center(child: Text("Membres")),
            const Center(child: Text("Ecrire un post")),
            const Center(child: Text("Notifications")),
            PageProfil(member: member),
          ];
```

Pour votre page de profil, vous pouvez vous inspirer du code ci-dessous comme trame de base :

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

import '../services_firebase/service_authentification.dart';
import '../services_firebase/service_firestore.dart';
import '../modeles/membre.dart';

class PageProfil extends StatefulWidget {
  final Membre member;
  PageProfil({super.key, required this.member});

  @override
  State<PageProfil> createState() => _PageProfilState();
}
```

```dart
class _PageProfilState extends State<PageProfil> {
  @override
  Widget build(BuildContext context) {
    return StreamBuilder<QuerySnapshot>(
      stream: ServiceFirestore().postForMember(widget.member.id),
      builder: (BuildContext context, AsyncSnapshot<QuerySnapshot> snapshot) {
        final data = snapshot.data;
        final docs = data?.docs;
        final length = docs?.length ?? 0;
        final isMe = ServiceAuthentification().isMe(widget.member.id);
        final indexToAdd = (isMe) ? 2 : 1;
        return ListView.builder(
            itemCount: length + indexToAdd,
            itemBuilder: (context, index) {
              if (index == 0) {
                return Column(
                  children: [
                    Container(
                      child: Stack(
                        alignment: Alignment.bottomLeft,
                        children: [
                          Column(
                            mainAxisSize: MainAxisSize.max,
                            children: [
                              Stack(
                                children: [
                                  //Image
                                  Container(
                                    height: 200,
                                    width: MediaQuery.of(context).size.width,
                                    color: Theme.of(context).colorScheme.primaryContainer,
                                    child: const Center(),
                                  )  // Container
                                  //Button
                                ],
                              ),  // Stack
                              const SizedBox(height: 25)
                            ],
```

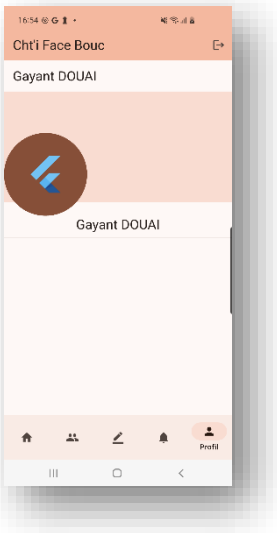
```
                    ),  // Column
                    Stack(
                      alignment: Alignment.bottomRight,
                      children: [
                        //ImageCirculaire
                        //Button
                      ],
                    )  // Stack
                  ],
                ),  // Stack
              ),  // Container
              Text(
                widget.member.fullName,
                style: Theme.of(context).textTheme.titleLarge,
              ),  // Text
              const Divider(),
              Text(widget.member.description)
            ],
          );  // Column
        }
      }
    );  // ListView.builder
  },
);  // StreamBuilder
}
}
```

6.4 Appel de la page de profil

Pour afficher cette page, vous appellerez à partir de la page navigation dans le 5ème bouton en remplacement du widget *Text(« Profil »)*.

6.5 Création de l'image de votre avatar

Dans cette étape, vous devez créer une classe qui retournera une image circulaire de votre photo d'avatar. Pour cela vous pouvez créer un nouveau fichier *avatar.dart* dans le dossier *widgets*. Cette classe sera basée sur un *StatelessWidget* qui retournera un widget *CircleAvatar* avec l'image réseau dont *radius* est le rayon et *url* de l'image sont passées comme paramètres. Si l'url est vide alors vous afficherez le logo flutter à l'aide du widget *FlutterLogo* avec une taille égale au rayon.



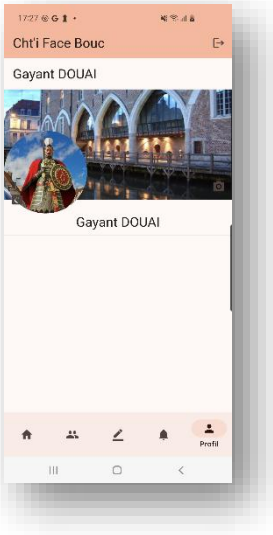
L'appel de ce widget se fera dans la page de profile à la place du commentaire *//ImageCirculaire* de la manière suivante :

```
children: [
  Avatar(
    radius: 75,
    url: widget.member.profilePicture,
  ),  // Avatar
  //Button
],
```

6.6 Ajout et sauvegarde d'une image

Dans cette étape, vous devez créer une classe qui retournera une image à partir de votre appareil.

Pour cela vous pouvez créer un nouveau fichier *bouton_camera.dart* dans le dossier *widgets*. Cette classe sera basée sur un *StatelessWidget* qui retournera un widget *IconButton* nommé *BoutonCamera* avec comme paramètres le type de l'image et l'id du membre.



Ce bouton fera appel à une méthode *_takePicture* d'acquisition d'image à partir d'un type et d'une source. Vous pouvez vous inspirez du code suivant :

```dart
_takePicture(ImageSource source, String type) async {
  final XFile? xFile =
      await ImagePicker().pickImage(source: source, maxWidth: 500);
  if (xFile == null) return;
  ServiceFirestore().updateImage(
    file: xFile,
    folder: memberCollectionKey,
    userId: id,
    imageName: type,
  );
}
```

L'appel de ce widget se fera à deux endroits dans la page de profil à la place des commentaires *//Button* au niveau de la photo de couverture avec un alignement en bas à droite et au niveau de la photo du profil avec un alignement en bas à gauche :

```dart
(isMe)
  ? BoutonCamera(
    type: coverPictureKey,
    id: widget.member.id
  )  // BoutonCamera
  : Center()
```

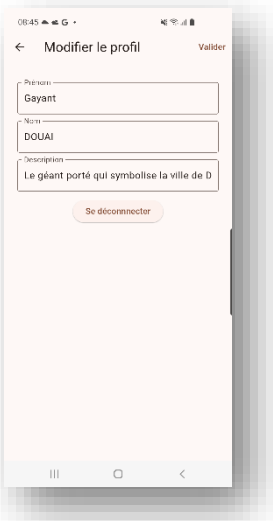
Photo de couverture

```dart
(isMe)
    ? BoutonCamera(
    type: profilePictureKey,
    id: widget.member.id
    )  // BoutonCamera
    : Center()
```

Photo du profil

6.7 Modifier le profil

En premier lieu, vous devez créer une page à partir d'un *StatefulWidget permettant de modifier votre profil.* Dans cette classe vous ajouterez :

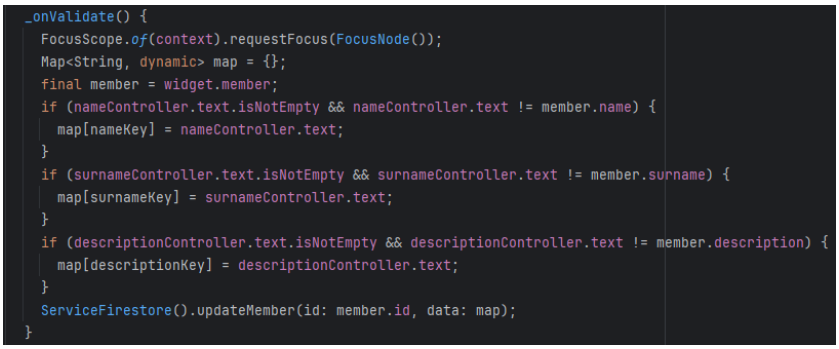


a) Les variables suivantes
   a. *TextEditingController surnameController* : champ texte pour la saisie du prénom
   b. *TextEditingController nameController* : champ texte pour la saisie du nom
   c. *TextEditingController descriptionController* : champ texte pour la saisie d'une description
b) Les méthodes *initState()* et *dispose()* pour les controlleurs précédents

c) Dans la méthode *build(BuildContext context)*

```
Scaffold
   appBar: AppBar
      title
      actions (valider les modifications => appel de la méthode _onValidate)
   body: SingleChildScrollView
      Container
         Column
            TextField (prénom)
            TextField (nom)
            TextField (description)
            ElevatedButton (se déconnecter)
               AlertDialog (Voulez vous vous déconnecter => OUI / NON)
                  Si OUI => déconnecter
```

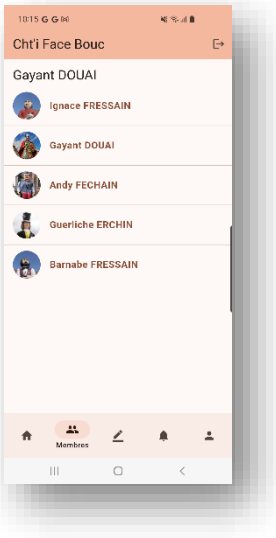
d) Une fonction *_onValidate* qui sera appelée par l'action



```dart
_onValidate() {
  FocusScope.of(context).requestFocus(FocusNode());
  Map<String, dynamic> map = {};
  final member = widget.member;
  if (nameController.text.isNotEmpty && nameController.text != member.name) {
    map[nameKey] = nameController.text;
  }
  if (surnameController.text.isNotEmpty && surnameController.text != member.surname) {
    map[surnameKey] = surnameController.text;
  }
  if (descriptionController.text.isNotEmpty && descriptionController.text != member.description) {
    map[descriptionKey] = descriptionController.text;
  }
  ServiceFirestore().updateMember(id: member.id, data: map);
}
```

Puis, ajoutez un bouton de type *OutlinedButton* permettant de modifier le profil « *Modifier le profil* » sur la page *PageProfil* en appelant la page précédente.

7 Etape 7 : La page des membres



7.1 Modifications de la classe ServiceFirestore

Vous devez créer dans cette classe, une nouvelle méthode *allMembers* permettant de récupérer la liste de tous les membres (voir ci-dessous) :

```
// Lire la liste de tous les membres
allMembers() => firestoreMember.snapshots();
```

7.2 Création du fond de page

Dans cette partie vous allez créer la page de profil dans votre application. Pour cela vous devez créer un nouveau *StateFul* widget dans un fichier *page_membres.dart*. Cette page, comme les précédentes, est réalisée autour d'un *StreamBuilder* pour la créer.

Cette page pourra être basée sur la structure suivante :

```
StreamBuilder
   stream: allMembers
   builder: (BuildContext context,
AsyncSnapshot<QuerySnapshot> snapshot) {
      return (snapshot.hasData)
         ? ListView.separated
              return ListTile
         : EmptyBody()
```

Cette nouvelle page sera appelée à partir de la barre de bouton en bas de la page Navigation sur le deuxième bouton « *Membres* ».

7.3 Profil d'un membre

Lors vous cliquez sur un membre de la liste, vous devez renvoyer sur sa page de profil. Pour cela vous pouvez adapter et réutiliser la classe *PageProfil* réaliser précédemment.

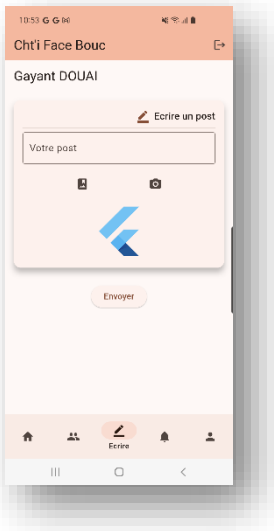
7.4 Appel de la page de la liste des membres

Pour afficher cette page, vous appellerez à partir de la page navigation dans le 2ème bouton en remplacement du widget *Text(« Membres »)*.

8 Etape 8 : La page d'écriture d'un post

8.1 Modifications de la classe FirestoreService

Vous devez créer dans cette classe, une nouvelle méthode *createPost* permettant d'écrire un post dans ServiceFirebase (voir ci-dessous) :




```
createPost(
    {required Membre member,
      required String text,
      required XFile? image}) async {
  final date = DateTime.now().millisecondsSinceEpoch;
  Map<String, dynamic> map = {
    memberIdKey: member.id,
    likesKey: [],
    dateKey: date,
    textKey: text,
  };
  if (image != null) {
    final url = await ServiceStorage().addImage(
      xfile: image,
      folder: postCollectionKey,
      memberId: member.id,
      imageName: date.toString(),
    );
    map[postImageKey] = url;
    firestorePost.doc().set(map);
  } else {
    firestorePost.doc().set(map);
  }
}
```

8.2 Création du fond de page

Vous devez créer dans le fichier *page_ecrire_post.dart* une page à partir d'un *StatefulWidget PageEcrirePost* permettant d'écrire un post et de rajouter ou pas une photo venant de l'appareil ou de la gallerie. Dans cette classe vous ajouterez :

a) Les variables suivantes
   a. *TextEditingController textController* : champ texte pour la saisie du texte du post
b) Les méthodes *initState()* et *dispose()* pour le contrôleur précédent
c) Dans la méthode *build(BuildContext context)*

```
SingleCHildScrollview
   Column
      Card
         Row
            Icon(border_color)
            Text(Ecrire un post)
         Divider
            TextField
         Row
            Icon(gallery)
            Icon(camera)
      ElevatedButton(envoyer)
```

d) Une fonction *_sendPost* qui sera appelée lors de l'appuis sur le bouton « *envoyer* »

```
_sendPost() {
  FocusScope.of(context).requestFocus(FocusNode());
  if (xFile == null && textController.text.isEmpty) return;
  ServiceFirestore().createPost(
    member: widget.member,
    text: textController.text,
    image: xFile,
  );
  widget.newSelection(0);
}
```

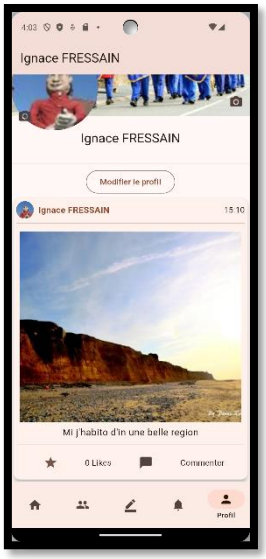
e) Une fonction *_takePic* qui sera appelée avec la bonne source lors de l'ajout d'une image

```
_takePic(ImageSource source) async {
  XFile? newFile =
  await ImagePicker().pickImage(source: source, maxWidth: 500);
  setState(() {
    xFile = newFile;
  });
}
```

8.3 Appel de la page d'écriture d'un post

Pour afficher cette page, vous appellerez à partir de la page navigation dans le 3ème bouton en remplacement du widget *Text(« Ecrire un post »)*.

9 Etape 9 : affichage de mes posts sur ma page de profil



Un post est constitué de plusieurs éléments comme nous l'avons vu dans la définition de la classe modèle « *post.dart* » :

- Un auteur
- Une date
- Un texte
- Une photo
- Un nombre de likes

Pour afficher ce post, nous allons définir une classe modèle pour les commentaires et une classe d'affichage pour la date.

9.1 Création d'un modèle d'affichage pour la date/heure

Dans ce même dossier « *modeles* », vous allez créer un fichier « *formatage_date.dart* » qui contiendra une classe modèle d'affichage de la date.

```dart
import 'package:intl/intl.dart';

class FormatageDate {
  String formatted(int timeStamp) {
    DateTime postTime = DateTime.fromMillisecondsSinceEpoch(timeStamp);
    DateTime now = DateTime.now();
    DateFormat format;
    if (now.difference(postTime).inDays > 0) {
      format = DateFormat.yMMMd();
    } else {
      format = DateFormat.Hm();
    }
    return format.format(postTime).toString();
  }
}
```

9.2 Création d'un widget pour afficher un post

Pour faciliter l'affichage des posts, je vous propose de créer un widget *WidgetPost* qui sera un *Stateless Widget* dans un fichier *post_widget.dart*.

Cette classe recevra un post en paramètre et aura comme structure dans la méthode *build* :

```dart
class WidgetPost extends StatelessWidget {
  final Post post;
  const WidgetPost({super.key, required this.post});

  @override
  Widget build(BuildContext context) {
    return Card(
      elevation: 8,
      child: Container(
        padding: const EdgeInsets.all(5),
        child: Column(
          children: [
            ContenuPost(post: post),
            const Divider(),
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
              children: [
                IconButton(
                  onPressed: () {},
                  icon: Icon(
                    Icons.star,
                    color: (post.likes.contains(ServiceAuthentification().myId!))
                        ? Theme.of(context).colorScheme.primary
                        : Theme.of(context).colorScheme.secondary,
                  ),  // Icon
                ),  // IconButton
                Text('${post.likes.length} Likes'),
                IconButton(
                  onPressed: () {                },
                  icon: const Icon(Icons.messenger),
                ),  // IconButton
                const Text('Commenter'),
              ],
            )  // Row
          ],
        ),  // Column
      ),  // Container
    );  // Card
  }
}
```

```
Card
   Container
      Column
         Row
            Avatar(rayon de 15 et photo de l'émetteur du post)
            Text(nom complet de l'émeteur du post)
            DateHandler(date d'émission du post)
         Si imageUrl alors
            Image.network
         Sinon
            Container(vide)
         Text(texte du post)
         Row
            IconButton(Icon.star)
            Text(nombre de likes)
            IconButton(Icon.messenger)
            Text(commenter)
```

9.3 Modification de la page de profil

Enfin, modifiez la page de profil pour afficher les posts du membre en vous inspirant du code ci-dessous et en utilisant le widget créé précédemment :

```
            final doc = snapshot.data!.docs;
            final current = doc[index - indexToAdd];
            final post = Post(
              reference: current.reference,
              id: current.id,
              map: current.data() as Map<String, dynamic>,
            );  // Post
            return WidgetPost(
              post: post,
            );  // WidgetPost
```

9.4 Trier les posts à l'affichage

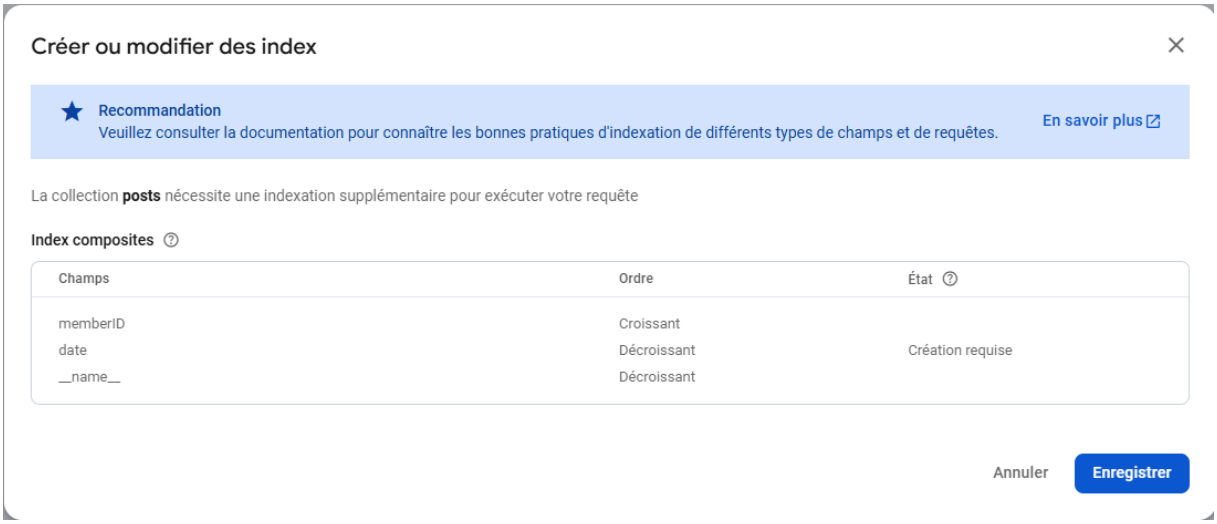
Il serait mieux d'afficher les posts du plus récent au plus ancien. Pour cela vous devez modifier la méthode *postForMender(String id)* dans le service Firestore pour ajouter un ordre de tri *orderBy* :

```
// Lire des posts d'un utilisateur
postForMember(String id) => firestorePost.where(memberIdKey, isEqualTo: id).orderBy(dateKey, descending: true).snapshots();
```

Vous devriez obtenir un message d'information comme ci-dessous qui vous indique qu'il manque un index dans votre collection :

```
W/Firestore(11267): (25.1.3) [Firestore]: Listen for Query(target=Query(posts where memberID==7kfeXBGrOGakC8nQ1eeHhuz7MUm2 order by -date, -__name__);limitType=LIMIT_TO_FIRST) failed:
 Status{code=FAILED_PRECONDITION, description=The query requires an index. You can create it here: https://console.firebase.google
 .com/v1/r/project/chti-face-bouc-c1d2d/firestore/indexes?create_composite
 =Cl1wcm9qZWNOcy9jaHRpLWZhY2UtYm91Yy1jMWQyZC9kYXRhYmFzZXMvKGRlZmF1bHQpL2Nvb6xlY3Rpb25Hcm91cHMvc69zdHMvaW5kZXhlcy9DSUNBZ09qW6oRUsQARoMCghtZW1iZXJJRBABGqgKBGRhdGUQAhoMCghfX25hbWVfXxAC, cause=null}
I/ViewRootImpl(11267): updatePointerIcon pointerType = 1000, calling pid = 11267
```

Cliquez sur le lien, puis cliquez sur « *Enregistrer* » sur la page apparaissant sur votre console firebase (voir ci-dessous), ceci a pour but de créer un index composite sur id (en croissant) puis sur la date (en décroissant) :



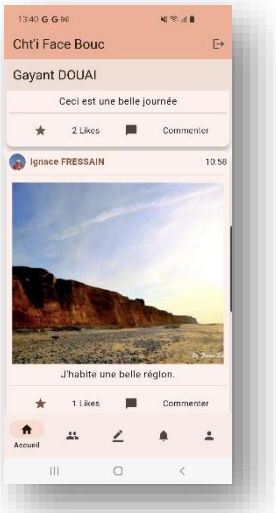
Si vous relancez votre application, vous devriez avoir l'affichage des posts du profil sélectionné par ordre du plus récent au plus ancien.

10 Etape 10 : Ajouter des « like »

Comme dans tout réseau social, vous devez donner la possibilité d'aimer ou pas un post. Pour cela lorsque le membre cliquera sur « likes », il ajoutera un « like » au post.

10.1 Modification du service Firestore

Dans un premier temps, vous devez créer une méthode *addLike({required String memberID, required Post post})* dans le service Firestore en vous inspirant du code ci-dessous :




```dart
// Ajouter un "j'aime" sur le post
addLike({required String memberID, required Post post}) {
  if (post.likes.contains(memberID)) {
    post.reference.update({likesKey: FieldValue.arrayRemove([memberID])});
  } else {
    post.reference.update({likesKey: FieldValue.arrayUnion([memberID])});
  }
}
```

10.2 Modification du widget post

Dans un deuxième temps, vous devez modifier le widget post pour appeler la méthode précédente lorsque l'utilisateur clique sur l'étoile (*IconStar*).

11 Etape 11 : Ajouter des commentaires

11.1 Création d'un modèle de données pour les commentaires

Dans le dossier « *modeles* », vous allez créer un fichier « *commentaire.dart* » qui contiendra une classe modèle pour stocker les commentaires dans Firebase.

```dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'constantes.dart';

class Commentaire {
  DocumentReference reference;
  String id;
  Map<String, dynamic> map;

  Commentaire({required this.reference, required this.id, required this.map});

  String get member => map[memberIdKey] ?? "";
  String get text => map[textKey] ?? "";
  int get date => map[dateKey] ?? 0;
}
```

11.2 Modification du service Firestore

Vous devez créer une méthode *addComment({required Post post, required String text})* dans le service Firestore en vous inspirant du code ci-dessous :

```dart
// Ajouter un commentaire sur un post
addComment({required Post post, required String text}) {
  final memberId = ServiceAuthentification().myId;
  if (memberId == null) return;
  Map<String, dynamic> map = {
    memberIdKey: memberId,
    dateKey: DateTime.now().millisecondsSinceEpoch,
    textKey: text,
  };
  post.reference.collection(commentCollectionKey).doc().set(map);
}
```

Vous devez ajouter une deuxième méthode *postComment({required String PostId })* qui permet de récupérer les commentaire d'un post. Pour cela, dans le service Firestore, vous devez créer une méthode en vous inspirant du code ci-dessous :

```dart
// Lire un commentaire sur un post
postComment(String postId) =>
    firestorePost.doc(postId).collection(commentCollectionKey).orderBy(dateKey, descending: true).snapshots();
```

11.3 Création d'un widget ListeCommentaire

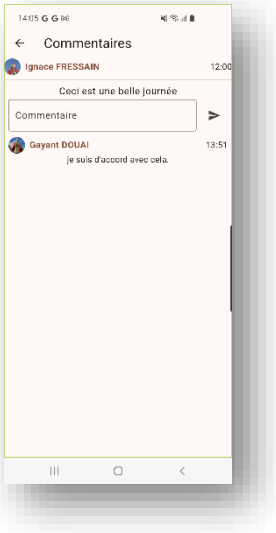
Pour faciliter l'affichage des commentaires, je vous propose de créer un widget ListeCommentaire qui sera un *Stateless Widget* dans un fichier *liste_commentaire.dart*.

Cette classe recevra un post en paramètre et la structure dans la méthode *build*, comme les précédentes, est réalisée autour d'un *StreamBuilder* pour la créer. Cette page pourra être basée sur la structure suivante :

```
StreamBuilder
   stream: allMembers
   builder: (BuildContext context,
AsyncSnapshot<QuerySnapshot> snapshot) {
      return (snapshot.hasData)
         ? ListView.separated
              return ListTile
         : EmptyBody()
```

11.4 Création d'un page commentaire

Vous devez définir une nouvelle classe *StatefulWidget* nommée Page*DetailPost* dans un fichier *page_detail_post_page.dart* qui permettra de saisir et d'afficher les commentaires sur un post. Dans cette classe vous ajouterez :



a) La variable suivante
   a. *TextEditingController commentController* : champ texte pour la saisie du commentaire
b) Les méthodes *initState()* et *dispose()* pour le contrôleur précédent
c) Dans la méthode *build(BuildContext context)*

```
    Scaffold
      AppBar: 'Commentaires'
      body: SingleChildScrollView
        Column
          WidgetPost
          Padding
            Row
              Expanded
                TextField,
                IconButton => addComment
          SizedBox(
            ListeCommentaire
```

11.5 Modification du widget post

Vous devez modifier le widget post pour appeler la page précédente lorsque l'utilisateur clique sur la bulle (*IconMessenger*).

12 Etape 12 : Les notifications

Nous allons mettre en place un système de « pseudo notification » car le système de notification sous android ou ios est devenu payant depuis l'année dernière.

12.1 Création d'un modèle de données pour les notifications

Dans le dossier « *modeles* », vous allez créer un fichier « *notification.dart* » qui contiendra une classe modèle pour définir les notifications dans Firebase.

```dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'constantes.dart';

class Notification {
  DocumentReference reference;
  String id;
  Map<String, dynamic> data;
  Notification({required this.reference, required this.id, required this.data});

  String get from => data[fromKey] ?? "";
  String get text => data[textKey] ?? "";
  int get date => data[dateKey] ?? 0;
  bool get isRead => data[isReadKey] ?? false;
  String get postId => data[postIdKey] ?? "";
}
```

12.2 Modification du service Firestore

Vous devez créer une première méthode *sendNotification({required String to, required String text, required String postID})* dans le service Firestore en vous inspirant du code ci-dessous :

```dart
// Envoyer une notification
sendNotification({required String to, required String text, required String postID}) {
  final memberId = ServiceAuthentification().myId;
  if (memberId == null) return;
  Map<String, dynamic> map = {
    dateKey: DateTime.now().millisecondsSinceEpoch,
    isReadKey: false,
    fromKey: memberId,
    textKey: text,
    postIdKey: postID
  };
  firestoreMember.doc(to).collection(notificationCollectionKey).doc().set(map);
}
```

Vous devez ajouter une deuxième méthode *markRead(DocumentReference reference)* qui permet de marquer une notification comme lue. Pour cela, dans le service Firestore, vous devez créer une méthode en vous inspirant du code ci-dessous :

```dart
// Marquer une notification qu a été lue
markRead(DocumentReference reference) {
  reference.update({isReadKey: true});
}
```

Vous devez ajouter une troisième méthode *notificationForUser(String id)* qui permet de lire les notifications envoyées à un membre. Pour cela, dans le service Firestore, vous devez créer une méthode en vous inspirant du code ci-dessous :

```dart
// Liste des notifications pour un membre
notificationForUser(String id) {
  firestoreMember.doc(id).collection(notificationCollectionKey).orderBy(dateKey, descending: true).snapshots();
}
```

12.3 Création d'un widget Notif

Pour faciliter l'affichage des notifications, je vous propose de créer un widget Notif qui sera un *Stateless Widget* dans un fichier *widget_notif.dart*. Cette classe recevra une notification en paramètre et aura comme structure dans la méthode *build* la structure suivante :

```dart
Widget build(BuildContext context) {
  return InkWell(
    onTap: () {
      ServiceFirestore().firestorePost.doc(notif.postId).get().then((snapshot) {
        ServiceFirestore().markRead(notif.reference);
        final post = Post(
          reference: snapshot.reference,
          id: snapshot.id,
          map: snapshot.data() as Map<String,dynamic>,
        );  // Post
        Navigator.of(context).push(
            MaterialPageRoute(builder: (context) {
              return PageDetailPost(post: post);
            })  // MaterialPageRoute
        );
      });
    },
    child: Container(
      color: (notif.isRead)
          ? Colors.green.withValues(alpha: 0.3)
          : Colors.red.withValues(alpha: 0.3),
      width: MediaQuery.of(context).size.width,
      margin: const EdgeInsets.only(
        left: 5,
        right: 5,
      ),  // EdgeInsets.only
      padding: const EdgeInsets.all(8),
      child: Column(
        children: [
          MemberHeader(
            memberId: notif.from,
            date: notif.date,
          ),  // MemberHeader
          Text(notif.text),
        ],
      ),  // Column
    ),  // Container
  );  // InkWell
}
```

12.4 Création d'un page Notif

Vous devez définir une nouvelle classe *StatefulWidget* nommée *PageNotif* dans un fichier *page_notif.dart* qui permettra de saisir et d'afficher les notifications pour un membre.

Cette classe recevra un membre en paramètre et la structure dans la méthode *build*, comme les précédentes, est réalisée autour d'un *StreamBuilder* pour la créer. Cette page pourra être basée sur la structure suivante :

```
StreamBuilder
   stream: notifForUser
   builder: (BuildContext context, AsyncSnapshot<QuerySnapshot> snapshot) {
      return (snapshot.hasData)
         ? ListView.separated
              return WidgetNotif
         : EmptyBody()
```