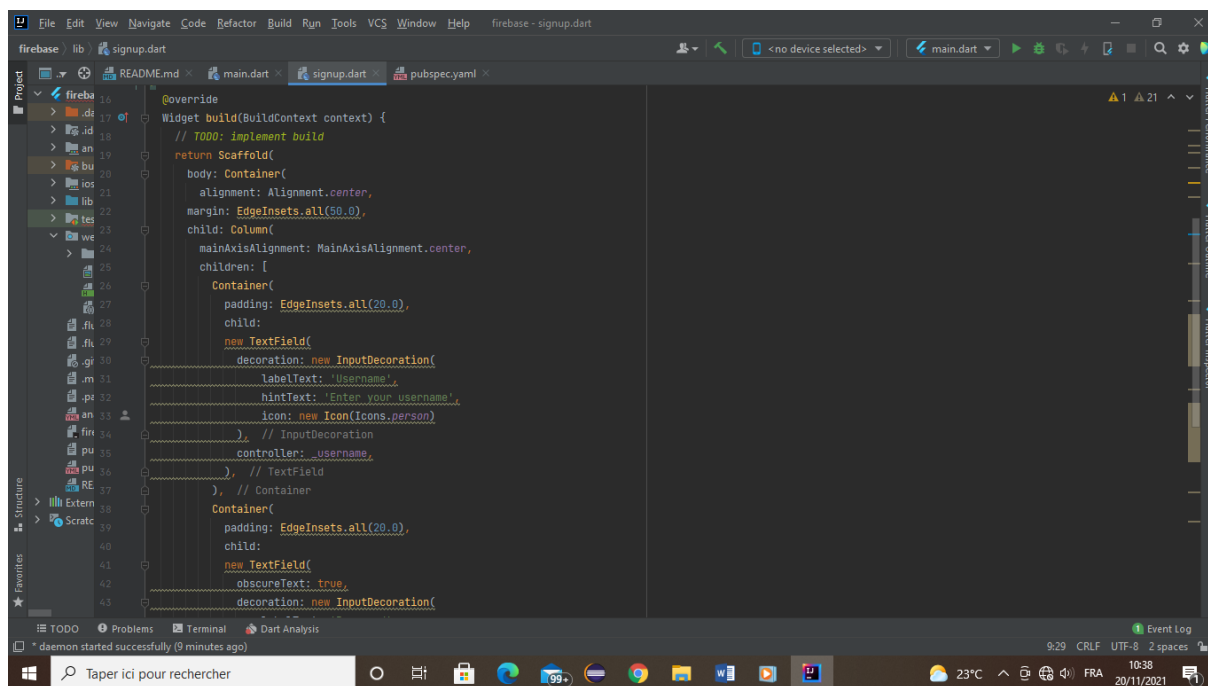


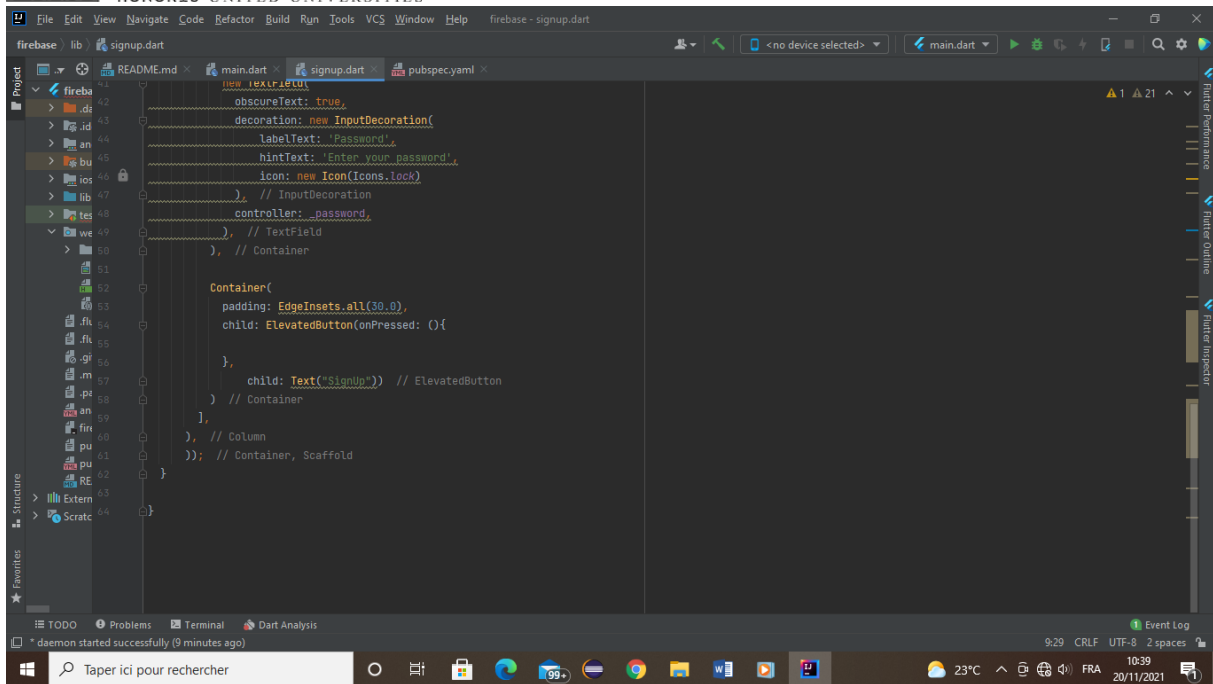
Firestore & Flutter

Etape1 :

On crée notre fonction main qui contient un formulaire d'inscription (vous choisissez les données que vous voulez).



```
@override
Widget build(BuildContext context) {
  // TODO: implement build
  return Scaffold(
    body: Container(
      alignment: Alignment.center,
      margin: EdgeInsets.all(50.0),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Container(
            padding: EdgeInsets.all(20.0),
            child:
              new TextField(
                decoration: new InputDecoration(
                  labelText: 'Username',
                  hintText: 'Enter your username',
                  icon: new Icon(Icons.person)
                ), // InputDecoration
                controller: _username,
              ), // TextField
          Container(
            padding: EdgeInsets.all(20.0),
            child:
              new TextField(
                obscureText: true,
                decoration: new InputDecoration(
```



Etape 2 : Utilisation des controllers

Afin de contrôler et récupérer les champs de texte, on va utiliser les 'TextEditingController', au lieu de onChanged() et onSubmitted().

Premièrement, il faut déclarer les contrôleurs.

```

TextEditingController _username=new TextEditingController();
TextEditingController _password=new TextEditingController();

```

Ensuite, attribuez à chaque champ de texte le contrôleur correspondant.

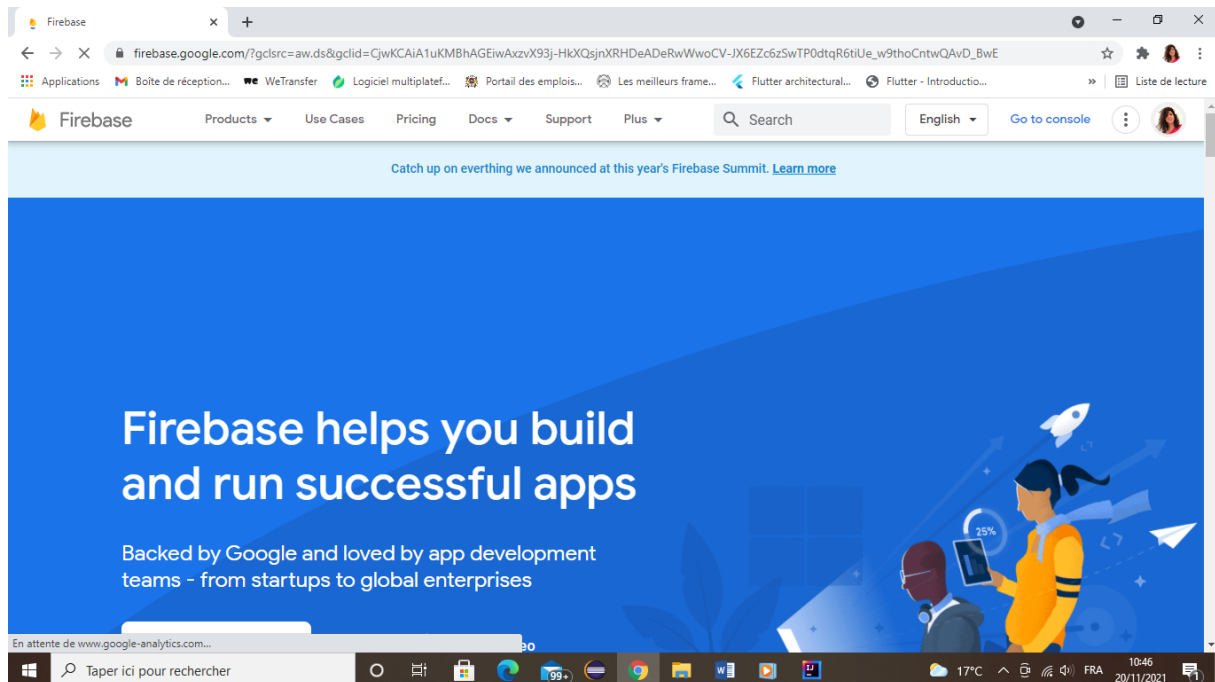
```

new TextField(
  decoration: new InputDecoration(
    labelText: 'Username',
    hintText: 'Enter your username',
    icon: new Icon(Icons.person)
  ), // InputDecoration
  controller: _username,
), // TextField
), // Container

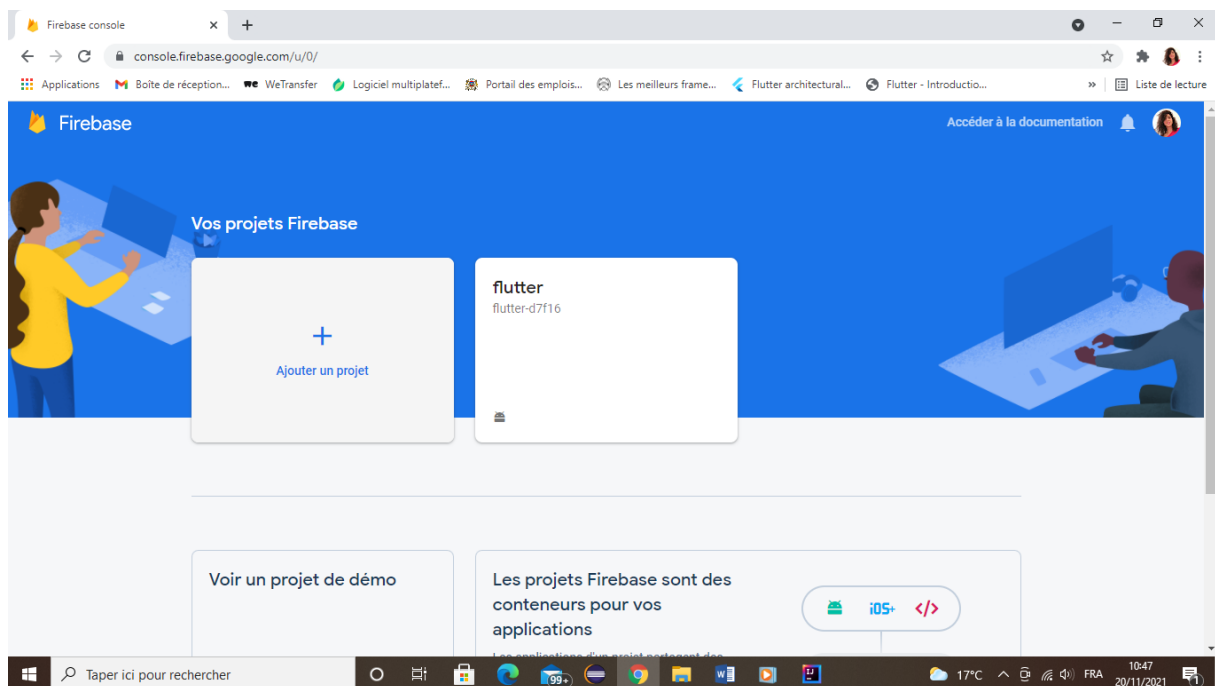
```

Etape3 : Créer un projet Firebase

Se connecter à Firebase.



Dans la console Firebase, cliquez sur Ajouter un projet (ou Créer un projet) et nommez votre projet.

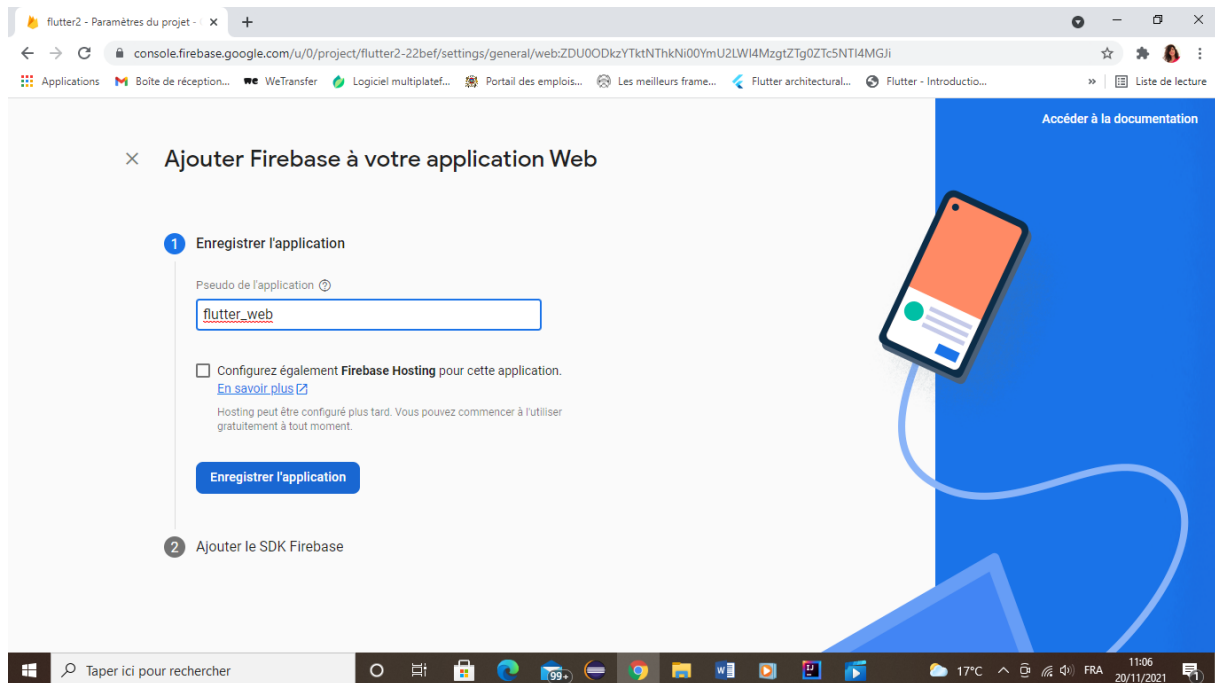


Etape 4 : Liaison du projet flutter avec Firebase

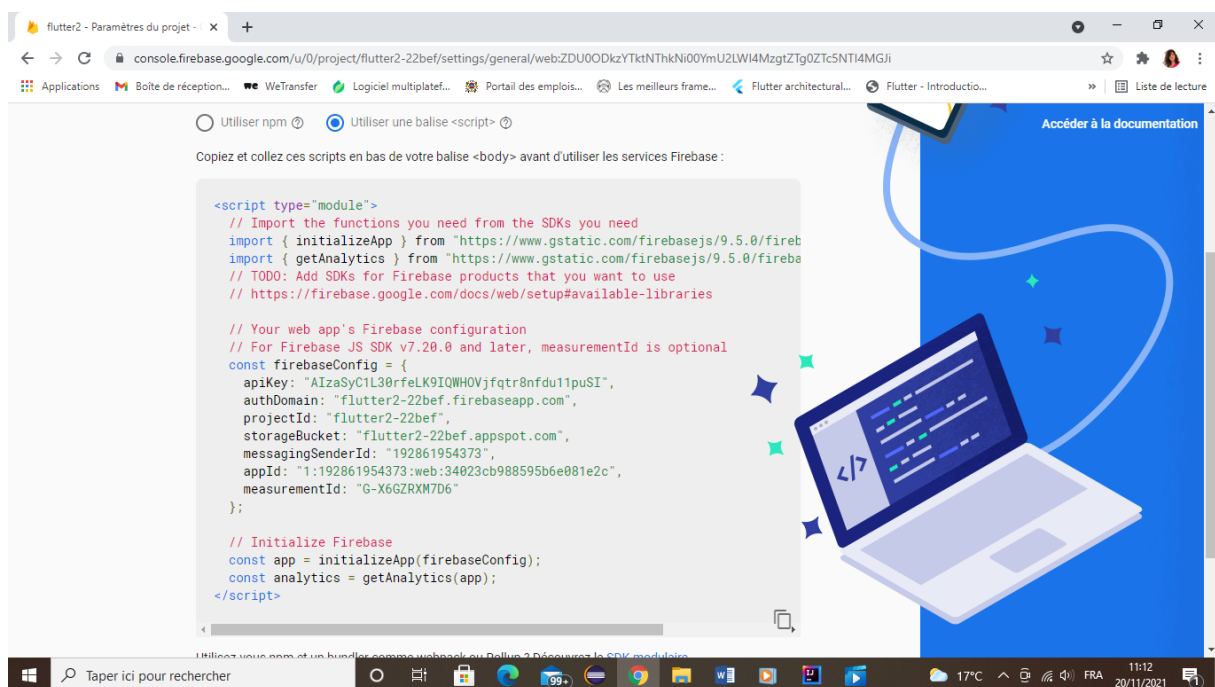
Ajouter le projet flutter à votre projet Firebase, on choisissant la nature de la plateforme sur laquelle vous compilez (android/iOS/Web).

Exemple : web

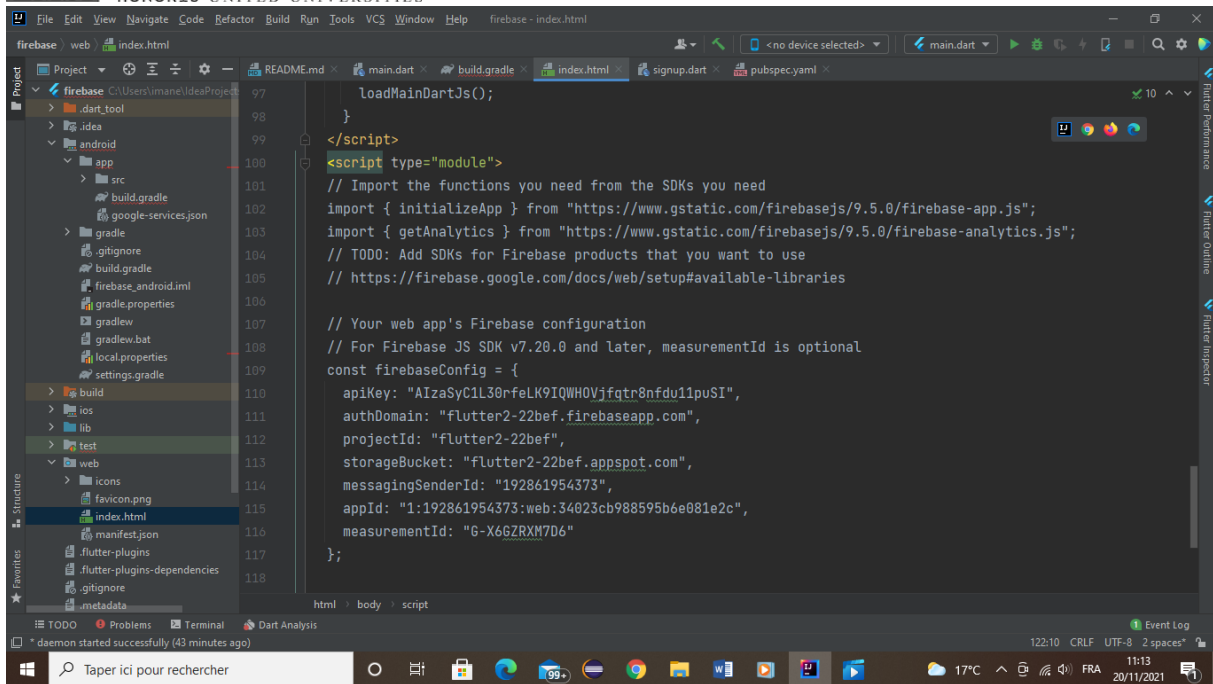
Enregistrez votre application avec un pseudo de votre choix.



Copiez le script généré afin de l'ajouter dans notre application.



Collez le script dans le fichier index.html du dossier web de votre projet flutter. Après la balise script.

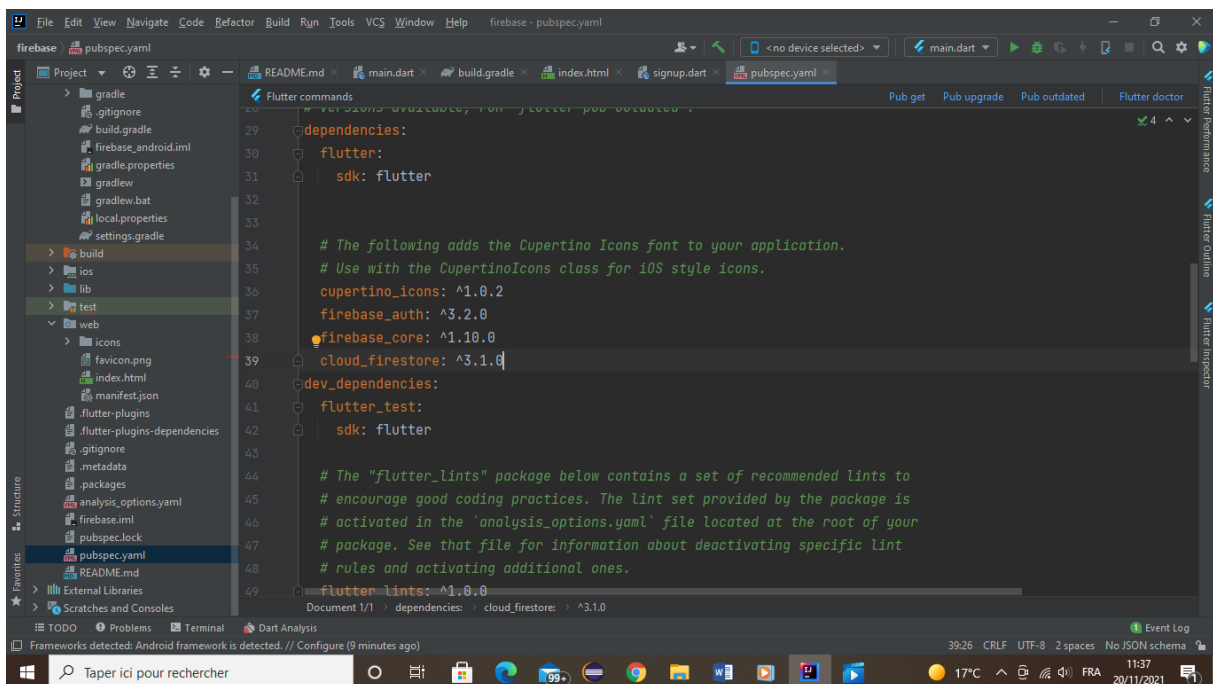


```

loadMainDartJs();
}
</script>
<script type="module">
// Import the functions you need from the SDKs you need
import { initializeApp } from "https://www.gstatic.com/firebasejs/9.5.0/firebase-app.js";
import { getAnalytics } from "https://www.gstatic.com/firebasejs/9.5.0/firebase-analytics.js";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
  apiKey: "AIzaSyC1L30rfelK9IQWH0Vjfqtr8nfdU11puSI",
  authDomain: "flutter2-22bef.firebaseio.com",
  projectId: "flutter2-22bef",
  storageBucket: "flutter2-22bef.appspot.com",
  messagingSenderId: "192861954373",
  appId: "1:192861954373:web:34023cb988595b6e081e2c",
  measurementId: "G-X6GZRXM7D6"
};
  
```

Direction le fichier pubspec.yaml, et ajouter les dépendances de Firebase_auth, Firebase_core et cloud_firestore. Vous pouvez trouver tous les packages dont vous avez besoin sur le site <https://pub.dev/>.



```

dependencies:
  flutter:
    sdk: flutter

  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.2
  firebase_auth: ^3.2.0
  firebase_core: ^1.10.0
  cloud_firestore: ^3.1.0

dev_dependencies:
  flutter_test:
    sdk: flutter

  # The "flutter_lints" package below contains a set of recommended lints to
  # encourage good coding practices. The lint set provided by the package is
  # activated in the 'analysis_options.yaml' file located at the root of your
  # package. See that file for information about deactivating specific lint
  # rules and activating additional ones.
  flutter_lints: ^1.0.0
  
```

Exemple2 : Android

Dans la console Firebase , sélectionnez Aperçu du projet dans la barre de navigation de gauche, puis cliquez sur le bouton Android sous Commencez par ajouter Firebase à votre application.

× **Add Firebase to your Android app**

1 **Register app**

Android package name ⓘ

App nickname (optional) ⓘ

Debug signing certificate SHA-1 (optional) ⓘ

Required for Dynamic Links, Invites, and Google Sign-In or phone number support in Auth. Edit SHA-1s in Settings.

Dans votre répertoire d'applications Flutter, ouvrez le fichier `android/app/src/main/AndroidManifest.xml`.

Dans le manifest élément, trouver la valeur de chaîne du package attribut. Cette valeur est le nom du package Android (quelque chose comme `com.yourcompany.yourproject`). Copiez cette valeur.

Dans la boîte de dialogue Firebase, collez le nom du package copié dans le champ Nom du package Android.

Vous n'avez pas besoin du certificat de signature de débogage SHA-1 pour cette codelab. Laissez ce champ vide.

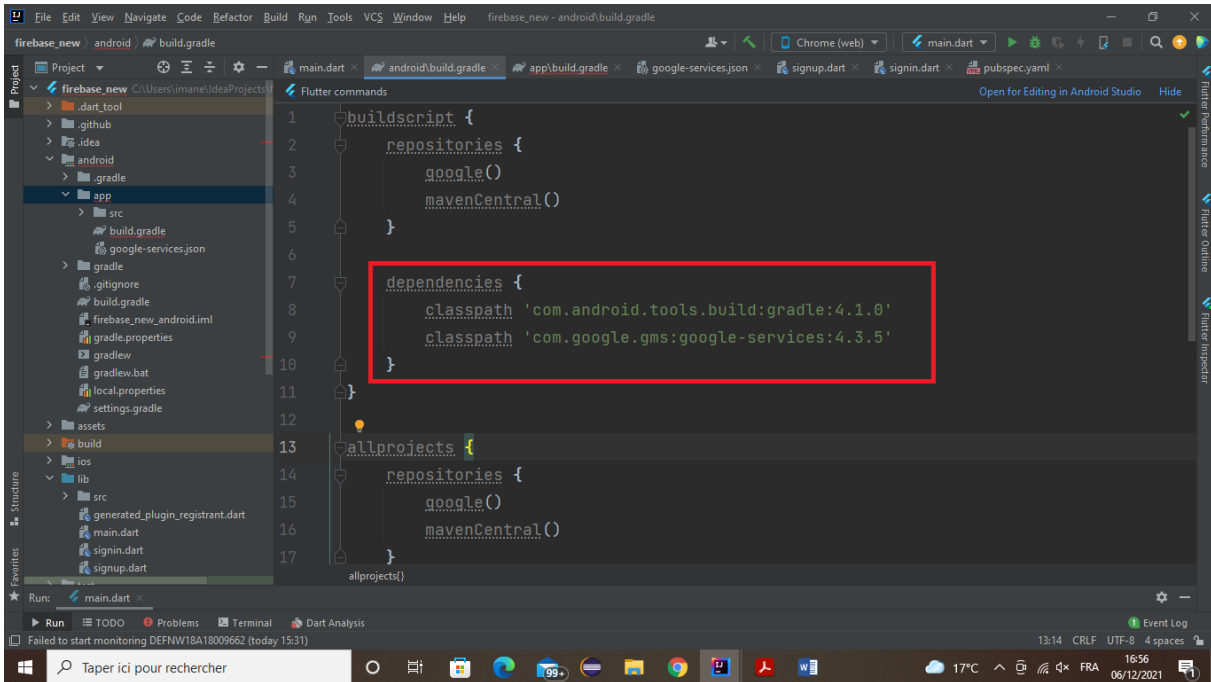
Cliquez sur Enregistrer App.

En continuant à Firebase, suivez les instructions pour télécharger le fichier de configuration `google-services.json`.

Accédez à votre répertoire d'applications Flutter, et déplacer le `google-services.json` fichier (que vous venez de télécharger) dans l' `android/app` répertoire.

De retour dans la console Firebase, ignorez les étapes restantes et revenez à la page principale de la console Firebase.

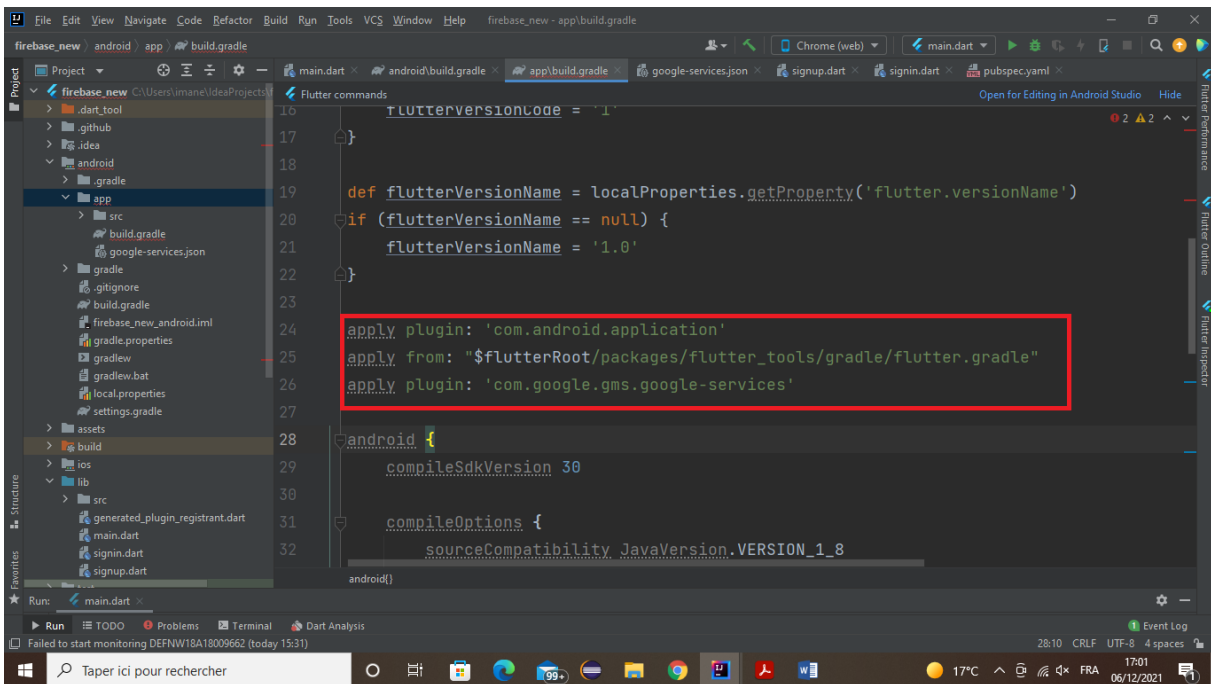
Modifier votre `android/build.gradle` pour ajouter les google-services plug - in dépendance:



```

1 buildscript {
2     repositories {
3         google()
4         mavenCentral()
5     }
6
7     dependencies {
8         classpath 'com.android.tools.build:gradle:4.1.0'
9         classpath 'com.google.gms:google-services:4.3.5'
10    }
11 }
12
13 allprojects {
14     repositories {
15         google()
16         mavenCentral()
17     }
18 }
  
```

Modifier votre android/app/build.gradle pour permettre aux google-services plugin:



```

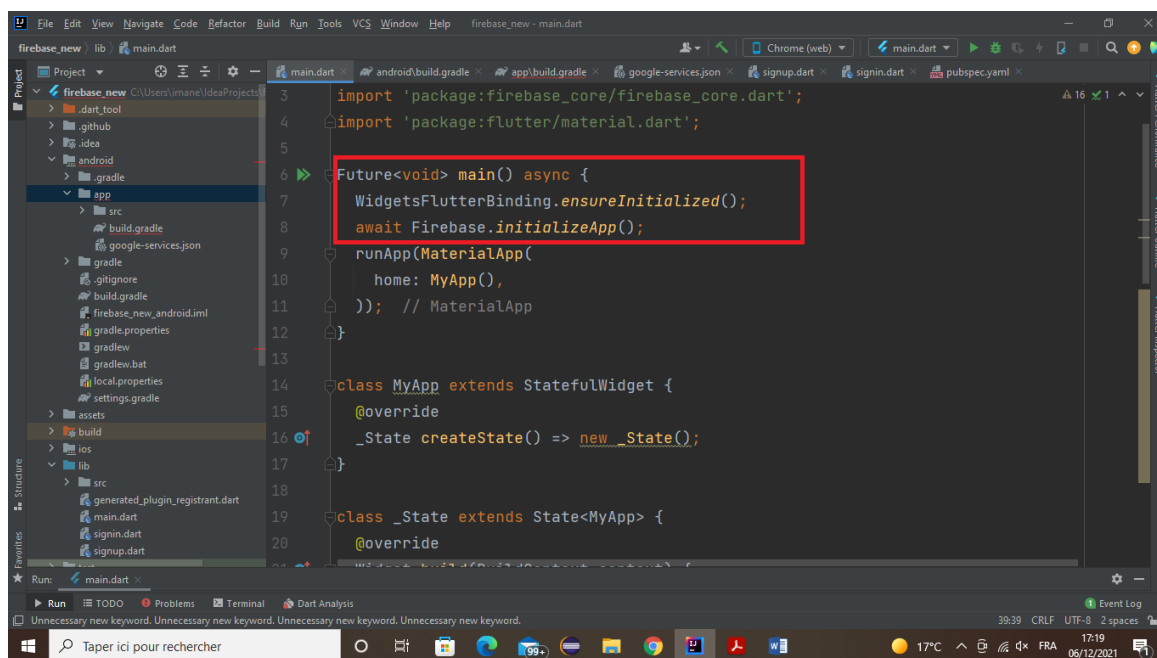
16 flutterVersionCode = 1
17
18
19 def flutterVersionName = localProperties.getProperty('flutter.versionName')
20 if (flutterVersionName == null) {
21     flutterVersionName = '1.0'
22 }
23
24 apply plugin: 'com.android.application'
25 apply from: "$flutterRoot/packages/flutter_tools/gradle/flutter.gradle"
26 apply plugin: 'com.google.gms.google-services'
27
28 android {
29     compileSdkVersion 30
30
31     compileOptions {
32         sourceCompatibility JavaVersion.VERSION_1_8
33     }
34 }
  
```

Firebase nécessite que Multidex soit activé, et une façon de le faire est de définir le SDK minimum pris en charge sur 21 ou plus. Modifier votre android/app/build.gradle mise à jour minSdkVersion :

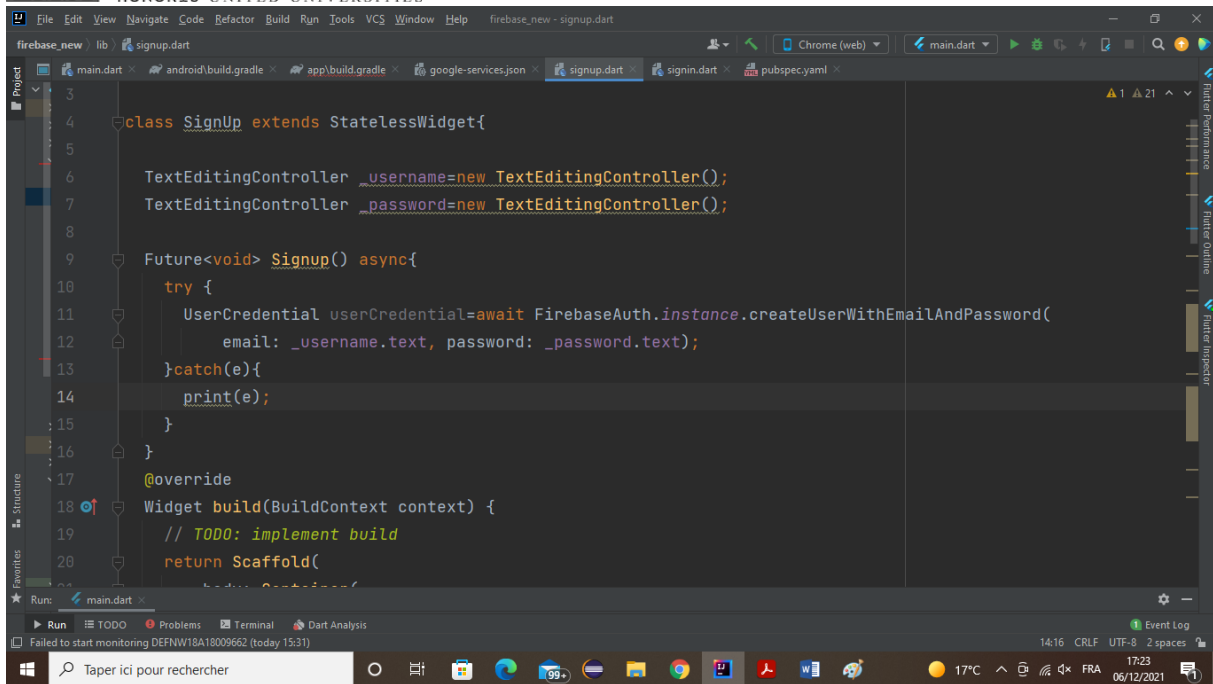
```
defaultConfig {
    applicationId "com.example.gtk_flutter"
    minSdkVersion 21 // Updated
    targetSdkVersion 30
    versionCode flutterVersionCode.toInteger()
    versionName flutterVersionName
}
```

Etape 5 : SignUp avec Firebase depuis l'application

Tout d'abord, il faut initialiser la base de données.



On fait appel à la fonction « **FirebaseAuth.instance.createUserWithEmailAndPassword** » pour pouvoir créer un nouvel utilisateur. Elle prend en paramètre l'email et le password saisie dans les champs de texte et récupéré en utilisant les controllers.



```

class Signup extends StatelessWidget{
  TextEditingController _username=new TextEditingController();
  TextEditingController _password=new TextEditingController();

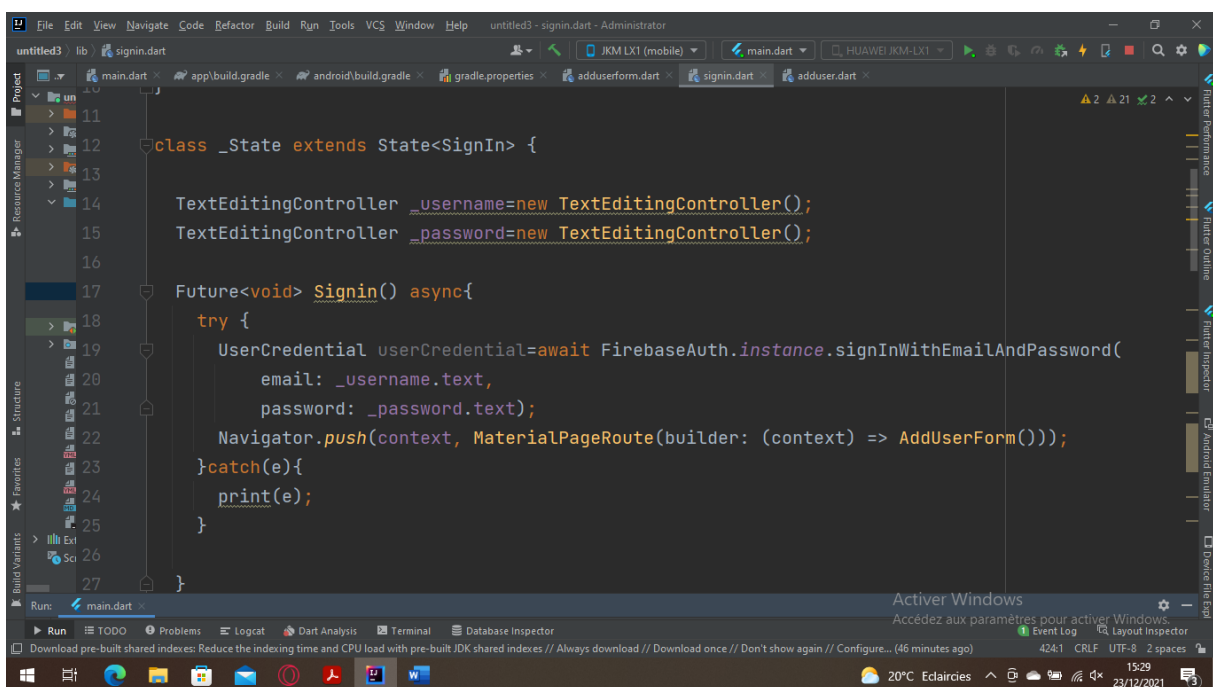
  Future<void> Signup() async{
    try {
      UserCredential userCredential=await FirebaseAuth.instance.createUserWithEmailAndPassword(
        email: _username.text, password: _password.text);
    }catch(e){
      print(e);
    }
  }

  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return Scaffold(
      body: Container(

```

Etape 6 : SignIn avec Firebase

On fait appel à la fonction « **FirebaseAuth.instance.signInWithEmailAndPassword** » pour pouvoir créer un nouvel utilisateur. Elle prend en paramètre l'email et le password saisie dans les champs de texte et récupéré en utilisant les controllers.



```

class _State extends State<SignIn> {
  TextEditingController _username=new TextEditingController();
  TextEditingController _password=new TextEditingController();

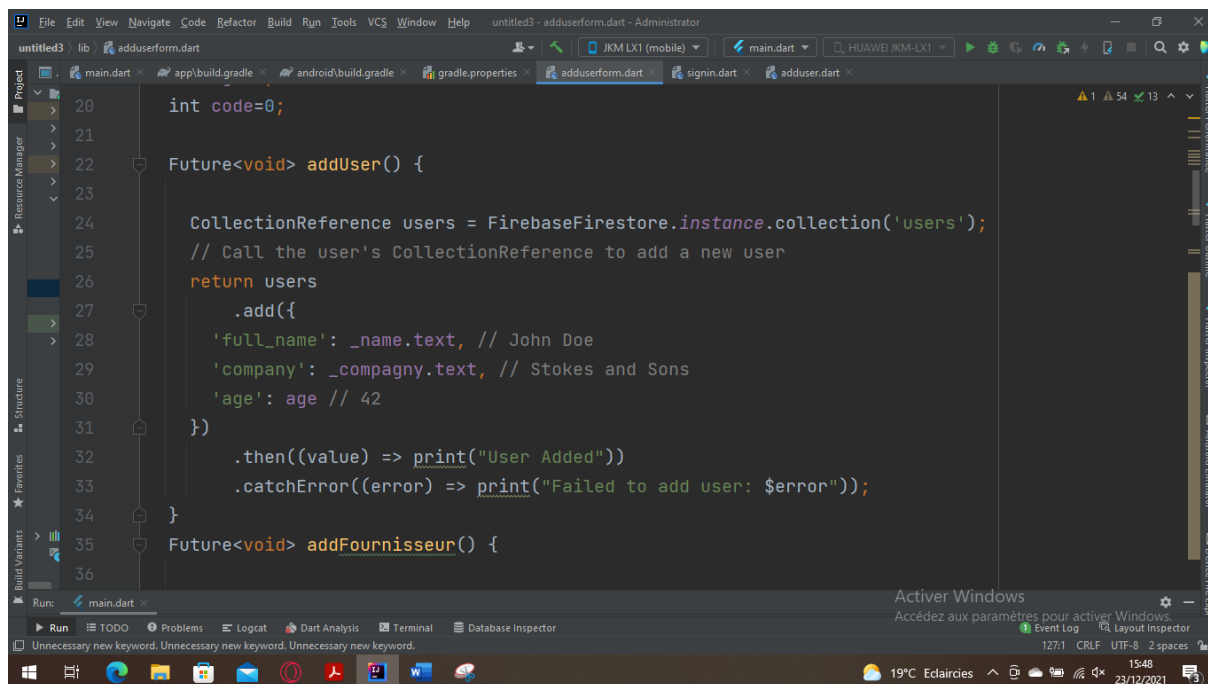
  Future<void> Signin() async{
    try {
      UserCredential userCredential=await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: _username.text,
        password: _password.text);
      Navigator.push(context, MaterialPageRoute(builder: (context) => AddUserForm()));
    }catch(e){
      print(e);
    }
  }
}

```

Etape 7 : Utilisation de Firestore. Ajouter un document.

Firestore stocke les données dans des « documents », qui sont contenus dans des « collections ». Les documents peuvent également contenir des collections imbriquées. Par exemple, nos utilisateurs auraient chacun leur propre "document" stocké dans la collection "Utilisateurs". La méthode collection nous permet de référencer une collection au sein de notre code.

Dans l'exemple ci-dessous, nous pouvons référencer les utilisateurs de la collection et créer un nouveau document utilisateur.



```
int code=0;

Future<void> addUser() {

    CollectionReference users = FirebaseFirestore.instance.collection('users');
    // Call the user's CollectionReference to add a new user
    return users
        .add({
            'full_name': _name.text, // John Doe
            'company': _compagny.text, // Stokes and Sons
            'age': age // 42
        })
        .then((value) => print("User Added"))
        .catchError((error) => print("Failed to add user: $error"));
}

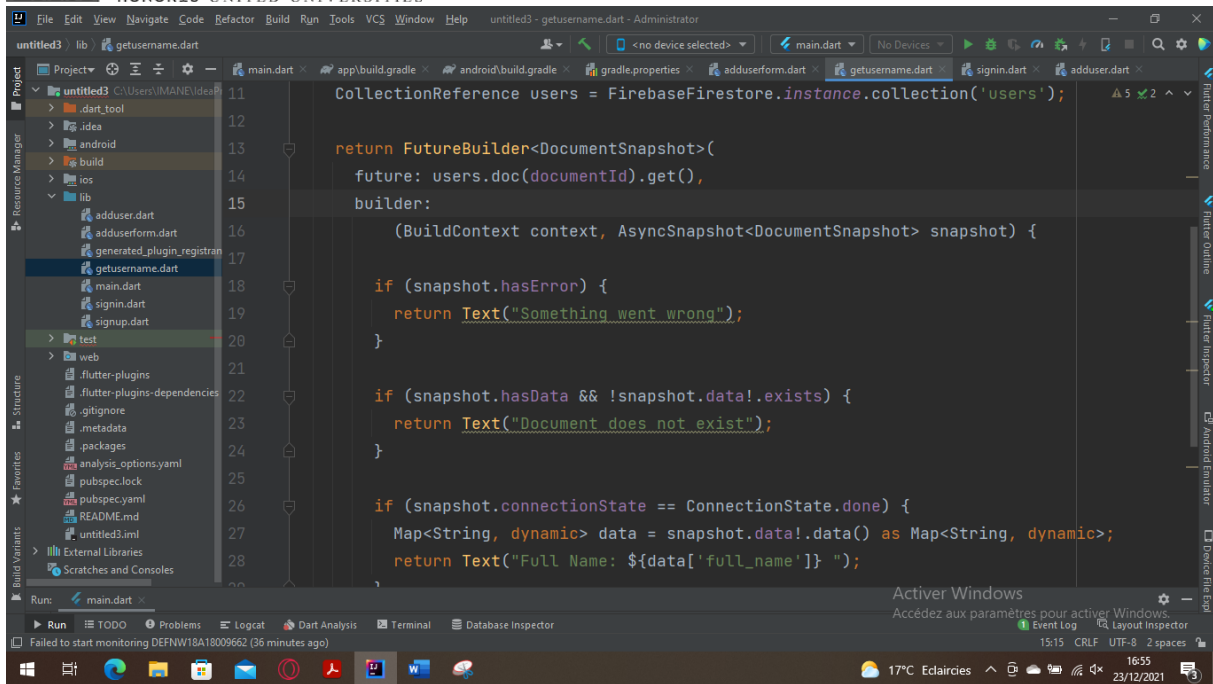
Future<void> addFournisseur() {
```

Etape 8 : Afficher les champs d'un document.

Cloud Firestore vous donne la possibilité de lire la valeur d'une collection ou d'un document. Il peut s'agir d'une lecture unique ou fournie par des mises à jour en temps réel lorsque les données d'une requête changent.

Lecture Unique :

Pour lire une collection ou un document une fois, appelez les méthodes Query.get ou DocumentReference.get.



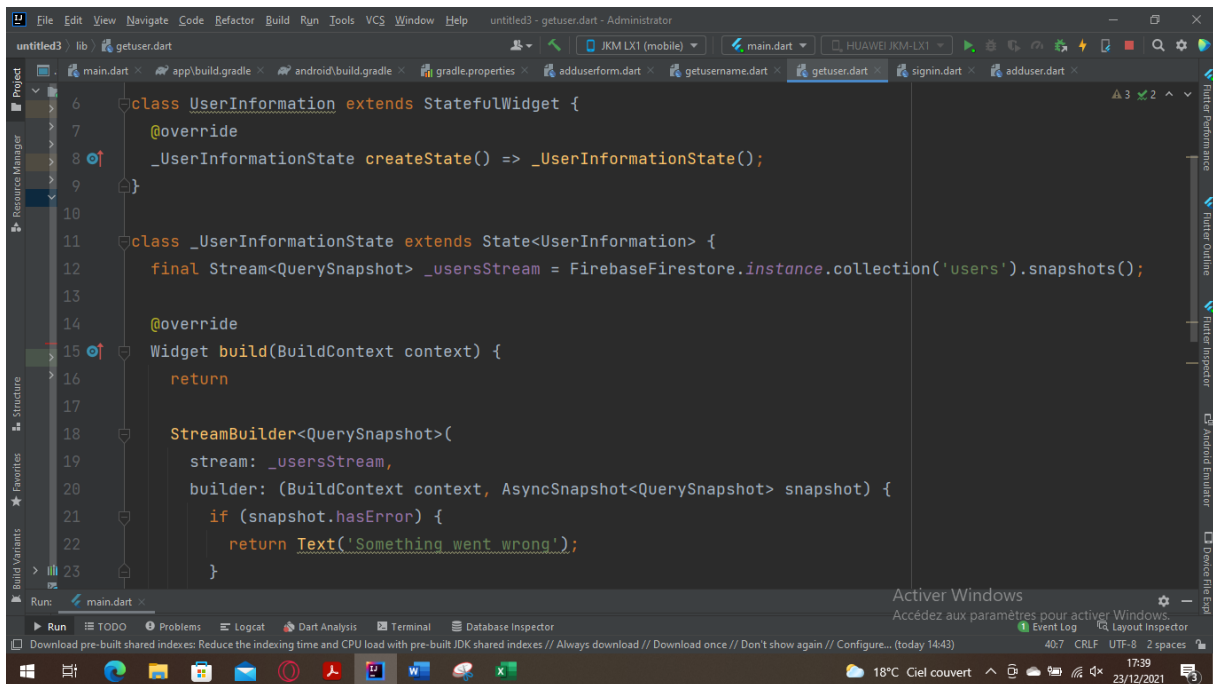
```

11 CollectionReference users = FirebaseFirestore.instance.collection('users');
12
13
14 return FutureBuilder<DocumentSnapshot>(
15   future: users.doc(documentId).get(),
16   builder:
17     (BuildContext context, AsyncSnapshot<DocumentSnapshot> snapshot) {
18
19       if (snapshot.hasError) {
20         return Text("Something went wrong");
21       }
22
23       if (snapshot.hasData && !snapshot.data!.exists) {
24         return Text("Document does not exist");
25       }
26
27       if (snapshot.connectionState == ConnectionState.done) {
28         Map<String, dynamic> data = snapshot.data!.data() as Map<String, dynamic>;
29         return Text("Full Name: ${data['full_name']} ");
30       }
31     }
32 );
  
```

Modifications en temps réel :

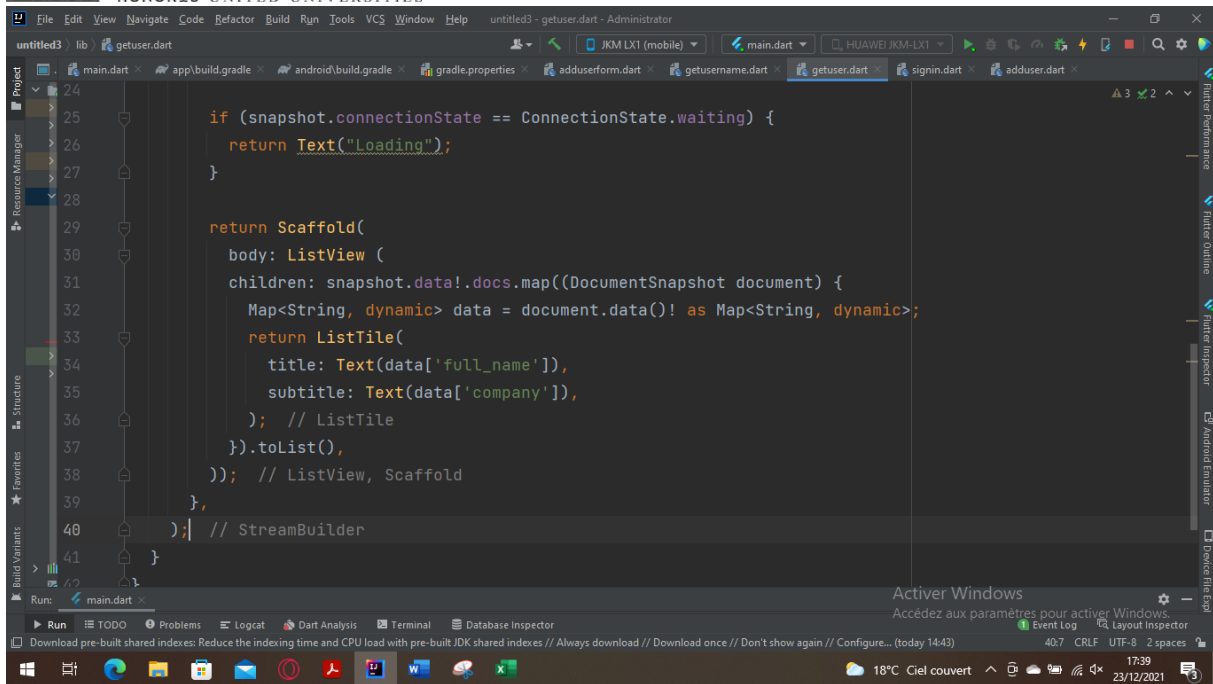
FlutterFire prend en charge la gestion des modifications en temps réel des collections et des documents. Un nouvel événement est fourni lors de la demande initiale, et toute modification ultérieure de la collection/du document chaque fois qu'un changement se produit (modification, suppression ou ajout).

CollectionReference et DocumentReference fournissent une méthode snapshots() qui renvoie un Stream.



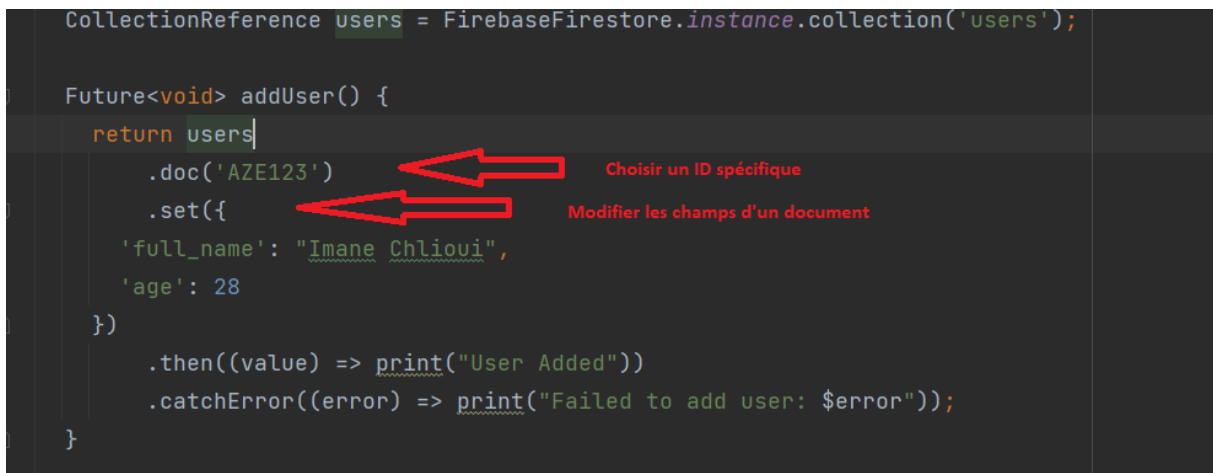
```

6 class UserInformation extends StatefulWidget {
7   @override
8   _UserInformationState createState() => _UserInformationState();
9 }
10
11 class _UserInformationState extends State<UserInformation> {
12   final Stream<QuerySnapshot> _usersStream = FirebaseFirestore.instance.collection('users').snapshots();
13
14   @override
15   Widget build(BuildContext context) {
16     return
17
18     StreamBuilder<QuerySnapshot>(
19       stream: _usersStream,
20       builder: (BuildContext context, AsyncSnapshot<QuerySnapshot> snapshot) {
21         if (snapshot.hasError) {
22           return Text('Something went wrong');
23         }
24       }
25     );
  
```

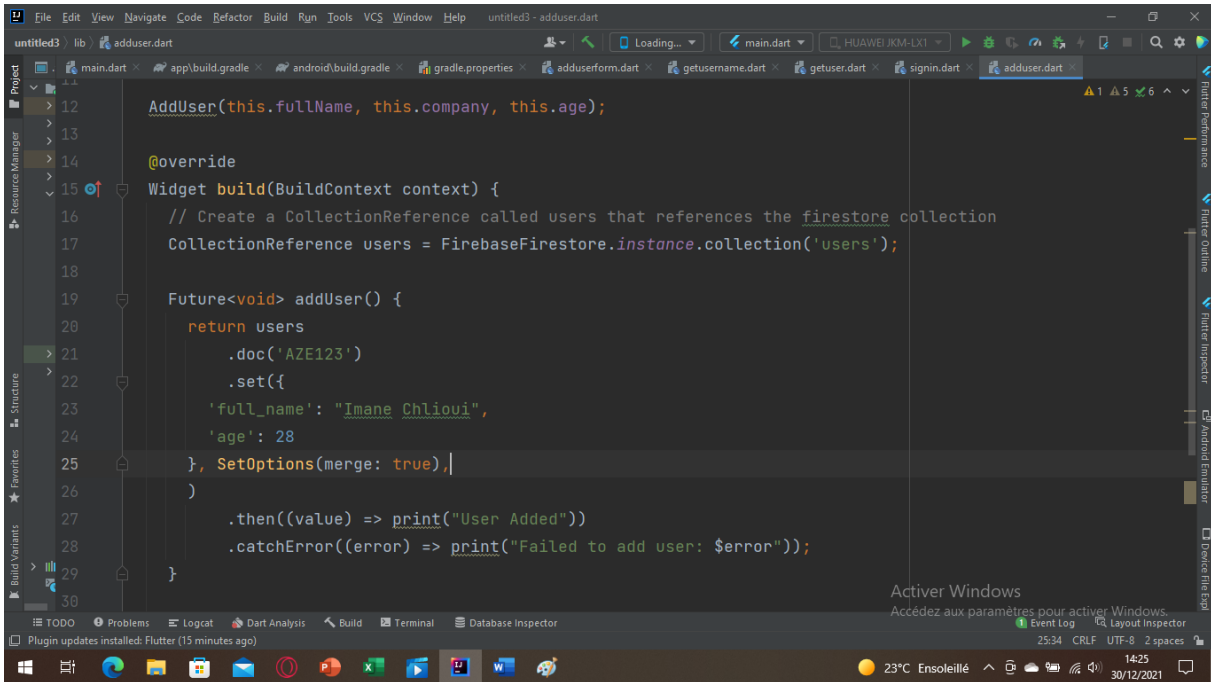


Etape 9 : Modification d'un document

La méthode `add` ajoute le nouveau document à votre collection avec un ID unique généré automatiquement. Si vous souhaitez spécifier votre propre ID, appelez plutôt la méthode `set` sur un `DocumentReference` :



L'appel de `set` avec un identifiant qui existe déjà sur la collection remplacera toutes les données du document. Vous pouvez également spécifier `SetOptions(merge : true)` sur la requête, et cela fusionnera le document existant avec les données transmises dans le `set()` :



```

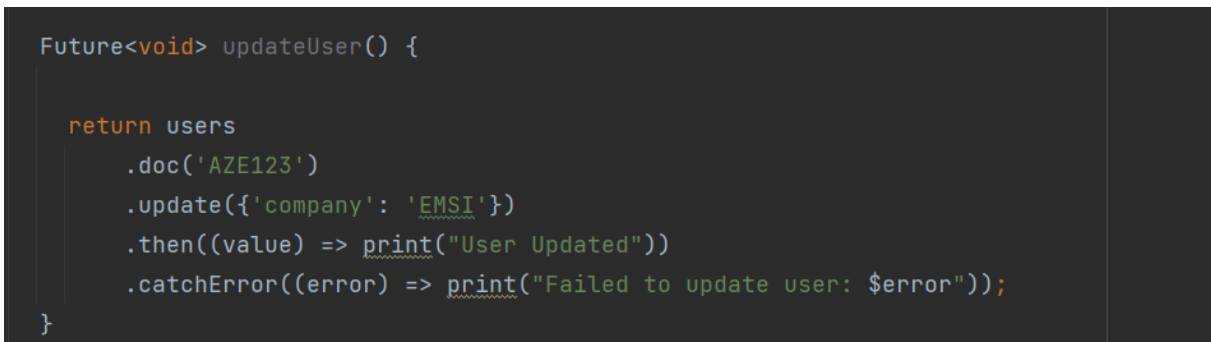
AddUser(this.fullName, this.company, this.age);

@override
Widget build(BuildContext context) {
  // Create a CollectionReference called users that references the firestore collection
  CollectionReference users = FirebaseFirestore.instance.collection('users');

  Future<void> addUser() {
    return users
      .doc('AZE123')
      .set({
        'full_name': "Imane Chlioui",
        'age': 28
      }, SetOptions(merge: true),
    )
      .then((value) => print("User Added"))
      .catchError((error) => print("Failed to add user: $error"));
  }
}

```

La méthode set ci-dessus remplace toutes les données existantes sur un DocumentReference donné. Si vous souhaitez plutôt mettre à jour un document, utilisez la méthode Update :



```

Future<void> updateUser() {
  return users
    .doc('AZE123')
    .update({'company': 'EMSI'})
    .then((value) => print("User Updated"))
    .catchError((error) => print("Failed to update user: $error"));
}

```

Etape 10 : stocker une image dans un document

Pour stocker une image, fournissez une Uint8List via le widget rootBundle, et ensuite on fait appel à la méthode Blob qui vous permet de stocker les fichiers multimédia.



```

Future<void> updateUser() {
  return rootBundle
    .load('assets/images/sample.jpg')
    .then((bytes) => bytes.buffer.asUint8List())
    .then((avatar) {
      return users
        .doc('ABC123')
        .update({'info.avatar': Blob(avatar)});
    })
    .then((value) => print("User Updated"))
    .catchError((error) => print("Failed to update user: $error"));
}

```

Etape 11 : Supprimer un document

Pour supprimer des documents avec Cloud Firestore, vous pouvez utiliser la méthode `delete` sur un `DocumentReference` :

```
CollectionReference users = FirebaseFirestore.instance.collection('users');  
  
Future<void> deleteUser() {  
    return users  
        .doc('AZE123')  
        .delete()  
        .then((value) => print("User Deleted"))  
        .catchError((error) => print("Failed to delete user: $error"));  
}
```

Si vous devez supprimer des propriétés spécifiques d'un document plutôt que du document lui-même, vous pouvez utiliser la méthode `delete` avec la classe `FieldValue` :

```
Future<void> deleteField() {  
    return users  
        .doc('ABC123')  
        .update({'age': FieldValue.delete()})  
        .then((value) => print("User's Property Deleted"))  
        .catchError((error) => print("Failed to delete user's property: $error"));  
}
```