

# Application Mobile

Préparé par:

Hadjer YKHLEF

Département d'Informatique

[ykhlef.hadjer@gmail.com](mailto:ykhlef.hadjer@gmail.com)

# Persistance des données



# CONTENT

## 01

### Introduction

## 02

### Firestore

Authentication, Real-Time database.

## 03

### SQLite

Création d'une base de données, Ouverture, Lecture, Ecriture,...

## 04

### Content Provider

Utilisations, Types, Fournisseurs intégrés & personnalisés.



**01**

# Introduction



# Introduction

- Plusieurs techniques sont à la disposition du développeur pour faire persister des données dans le temps, chacune offrant un compromis entre facilité de mise en œuvre, rapidité et flexibilité.
  - Persistance des activités
  - Système de fichiers
  - Préférence partageable
  - Fournisseurs de contenu (Content Provider)
  - Données sur le Cloud (online)
  - Base de données (SQLite)
  - ORM (Object-Relational Mapping)

# Introduction

- Le choix de la solution idéale dépend des besoins spécifiques:
  - Si les données doivent être **privées** ou **accessibles** par **d'autres applications**.
  - Combien **d'espace** disponible est-il requis?
  - Les données sont-elles **structurées**, **semi-structurées** ou **non structurées**?
- Aussi, chaque technique de persistance de données a ses avantages et inconvénients, il est important de choisir la méthode la plus adaptée aux besoins en termes de sécurité, performances, complexité et portabilité.

**02**

**Firebase**



# Introduction

- En 2011, Firebase Inc. crée une plateforme de développement d'application mobiles et web appelé : Firebase. Puis en 2014 Google acquiert Firebase.
- Firebase est une plateforme d'application mobile et Web avec des outils et une infrastructure conçus pour aider les développeurs à créer des applications de haute qualité.
- Firebase est compatible avec de nombreux frameworks et outils populaires tels que Angular, React, Vue.js, Unity, et bien plus encore.



# Services

Les services proposés par **Firestore** comprennent notamment :

- **Firestore Analytics** : Un service qui permet aux développeurs de suivre et d'analyser le comportement des utilisateurs dans leur application, tels que le nombre d'utilisateurs actifs, la durée de session moyenne, les événements clés, etc.
- **Firestore AdMob** : Une fonctionnalité qui permet aux développeurs de monétiser leur application en affichant des publicités.
- **Firestore Authentication**: Firestore offre des fonctionnalités d'authentification des utilisateurs, notamment l'authentification par e-mail et mot de passe, l'authentification par téléphone, l'authentification avec Google, Facebook, Twitter, GitHub, etc.

# Services

Les services proposés par **Firestore** comprennent notamment :

- **Firestore Cloud Messaging** : Un service de messagerie en temps réel qui permet aux développeurs d'envoyer des notifications push à leurs utilisateurs même lorsque l'application n'est pas en cours d'exécution.
- **Firestore Hosting** : Un service d'hébergement qui permet aux développeurs de déployer leur application et de la mettre à disposition des utilisateurs sur le web.

# Services

Les services proposés par **Firestore** comprennent notamment :

- **Base de données en temps réel** : Firestore offre **deux** services de base de données en temps réel : **Cloud Firestore** et **Realtime Database**.
- Ces services permettent de stocker et de synchroniser les données en **temps réel** entre les clients et les serveurs.

# Cloud Firestore vs Realtime Database

Voici les principales différences entre les deux :

- **Structure des données : Cloud Firestore** utilise une structure de données de type **document-collection**, tandis que **Realtime Database** utilise une structure de données de type **arbre JSON**.
- **Requêtes : Firestore** offre des **requêtes plus puissantes et flexibles** que Realtime Database. Par exemple, Firestore prend en charge les requêtes de **filtrage**, les requêtes de **tri**, les requêtes de pagination et les requêtes de jointure. Realtime Database n'offre pas toutes ces fonctionnalités.

# Cloud Firestore vs Realtime Database

Voici les principales différences entre les deux :

- **Évolutivité** : **Firestore** est capable de **gérer de grandes quantités** de données et de trafic en temps réel, tandis que **Realtime Database** peut parfois être **plus lent** lorsqu'il gère de grandes quantités de données.
- **Sécurité** : Les deux services de base de données offrent des fonctionnalités de sécurité telles que l'authentification des utilisateurs et les règles de sécurité. Cependant, **Firestore** offre une sécurité **plus fine-grain** avec un système de règles de sécurité plus avancé et granulaire.

# Services

Les services proposés par **Firestore** comprennent notamment :

- **Firestore ML** : Un service de Firestore qui permet aux développeurs d'ajouter des fonctionnalités de machine learning à leurs applications Android. Firestore ML offre des modèles de machine learning pré-entraînés ainsi que des outils pour entraîner des modèles personnalisés.

# Services: Firebase ML

Les fonctionnalités de Firebase ML comprennent :

- **Firebase ML Kit** : Une bibliothèque de développement logiciel (SDK) qui fournit des modèles de machine learning **pré-entraînés** pour des tâches telles que la **reconnaissance d'image, la détection de texte et la traduction de langues**.
  - Firebase ML Kit est facile à intégrer dans une application Android et ne nécessite aucune expertise en machine learning.

NotificationPrjt ▼

# Machine Learning


APIs Custom AutoML

 Le 3 juin 2020, nous avons commencé à proposer les API de ML Kit sur appareil via un nouveau SDK autonome. Les API Cloud et le modèle de déploiement personnalisé resteront disponibles via Firebase Machine Learning.

[Ignorer](#) [En savoir plus](#)

Toutes les API ▼

 Mettez votre projet à jour pour passer à la formule Blaze et activer les API Cloud [Changer de formule](#)



**Reconnaissance optique de caractères**

Détecter du texte dans les images et l'extraire

[Premiers pas](#)



**Ajout de libellés à des images**


Identifiez des objets, des lieux, des activités, des espèces animales, des produits et plus encore

[Premiers pas](#)



**Détection des visages**

Détectez les visages et les repères faciaux à l'aide de la fonctionnalité Contours de visage

[Premiers pas avec ML Kit](#) 



**Détection d'objets et suivi**


Détecter, suivre et classer des objets dans des images statiques ou en direct de l'appareil photo





**Lecture de codes-barres**

Analysez et traitez des codes-barres



**ID de langue**

Détectez la langue du texte



# Services: Firebase ML

Les fonctionnalités de Firebase ML comprennent :

- **AutoML Vision Edge** : Un outil qui permet aux développeurs de **créer des modèles de classification d'images personnalisés à partir des données d'entraînement personnelles.**
  - Les modèles entraînés peuvent être déployés sur des appareils Android en utilisant Firebase ML Kit.



## Services: Firebase ML

Les fonctionnalités de Firebase ML comprennent :

- **TensorFlow Lite** : Une bibliothèque open-source pour la mise en œuvre de modèles de machine learning sur des appareils mobiles et intégrée dans Firebase ML.
  - Elle offre une grande flexibilité pour les développeurs qui souhaitent construire leurs propres modèles de machine learning.



# TensorFlow Lite

# Minimum requirements

**Firestore** nécessite :

- Un appareil tournant sous Android 2.3 minimum.
- **Google Play service 9.8.0 ou plus, l'émulateur doit disposer des services Google play.**
- Android Studio 1.5 ou plus.



**01**

# **COMMENT AJOUTER FIREBASE À L'APPLICATION?**

# 1. Créer un Projet

1. Créer un nouveau projet **Firebase** dans la **console Firebase** (<https://console.firebase.google.com/u/0/>). Suivre les étapes de configuration du projet.



Commençons par donner un nom à votre projet?

Nom du projet

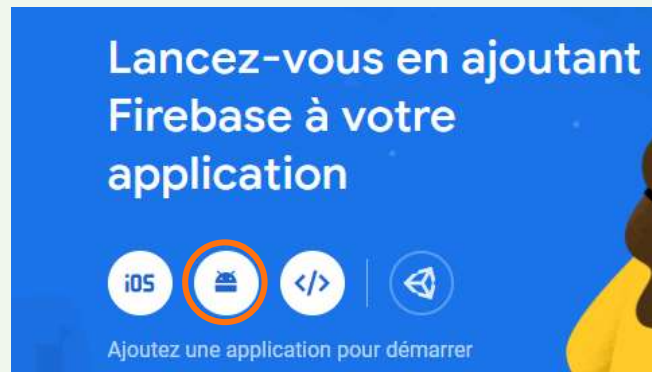
FirebaseExemple

 fir-exemple-a1133

Continuer

## 2. Ajouter Firebase à une application Android

1. Cliquer sur Ajouter Firebase à une application Android:



## 2. Ajouter Firebase à une application Android

2. Entrer le nom du package de l'application (Manifest.xml). Il est important de saisir le nom complet du package utilisé par l'application.

[illegible]

## 2. Ajouter Firebase à une application Android

3. Télécharger le fichier google-services.json.

2

Télécharger le fichier de configuration

Instructions relatives à Android Studio ci-dessous: | [Unity](#) [C++](#)

Télécharger google-services.json

Accédez à la page **Projet** dans Android Studio pour consulter le répertoire racine de votre projet.

Déplacez le fichier google-services.json que vous venez de télécharger dans le répertoire racine du module de votre application Android.

google-services.json

Précédent

Suivant

Project Packages Scratch

MyApplication (~/.Desktop/MyApplication)

Structure

.gradle

.idea

app

build

libs

src

.gitignore

app.iml

build.gradle

google-services.json

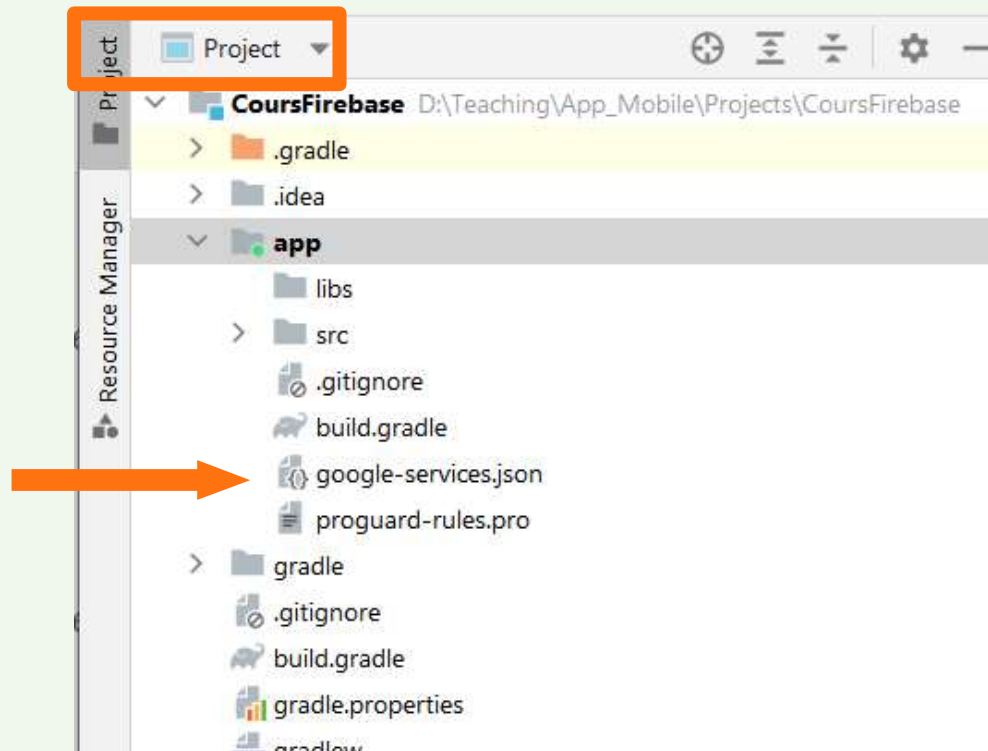
proguard-rules.pro

gradle



## 2. Ajouter Firebase à une application Android

4. Placer le fichier **google-services.json** dans le dossier du module du projet, dossier app/ .

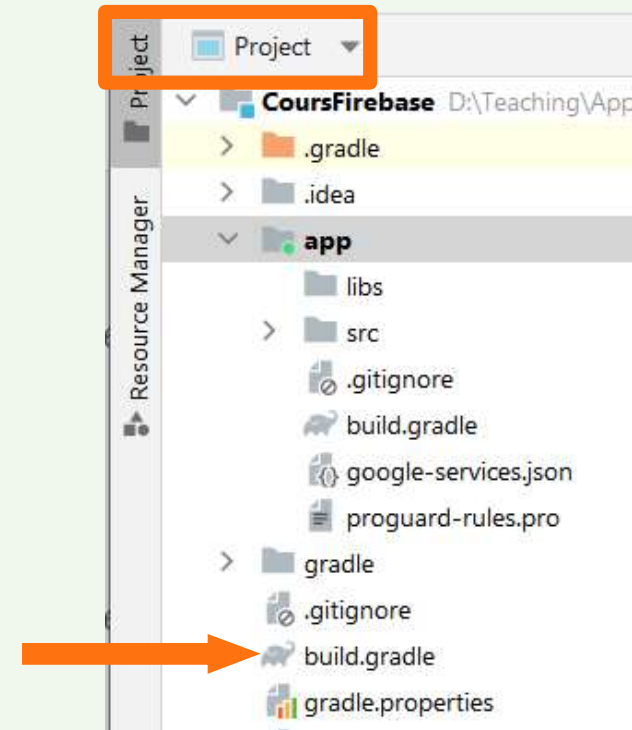


## 2. Ajouter Firebase à une application Android

5. Pour intégrer les bibliothèques Firebase dans un projet (**Android Studio 1.5.1**):
- ajouter des règles au fichier build.gradle au niveau du projet (**<project>/build.gradle**) pour inclure le plug-in google-services :

1

```
buildscript {  
    repositories {  
        jcenter()  
        maven {  
            url 'https://maven.google.com'  
        }  
    }  
}
```

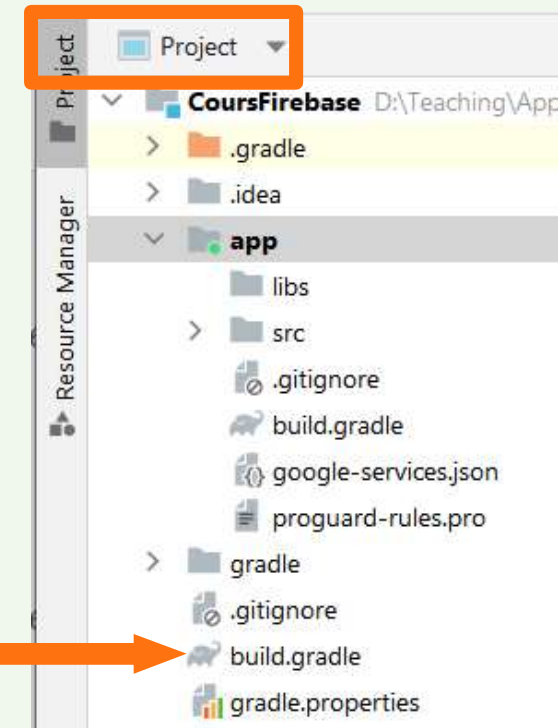


## 2. Ajouter Firebase à une application Android

5. Pour intégrer les bibliothèques Firebase dans un projet (**Android Studio 1.5.1**):
- ajouter des règles au fichier build.gradle au niveau du projet (**<project>/build.gradle**) pour inclure le plug-in google-services :

2

```
dependencies {  
    classpath 'com.android.tools.build:gradle:1.5.0'  
    classpath 'com.google.gms:google-services:3.0.0'  
  
    // NOTE: Do not place your application dependencies here  
    // in the individual module build.gradle files  
}
```

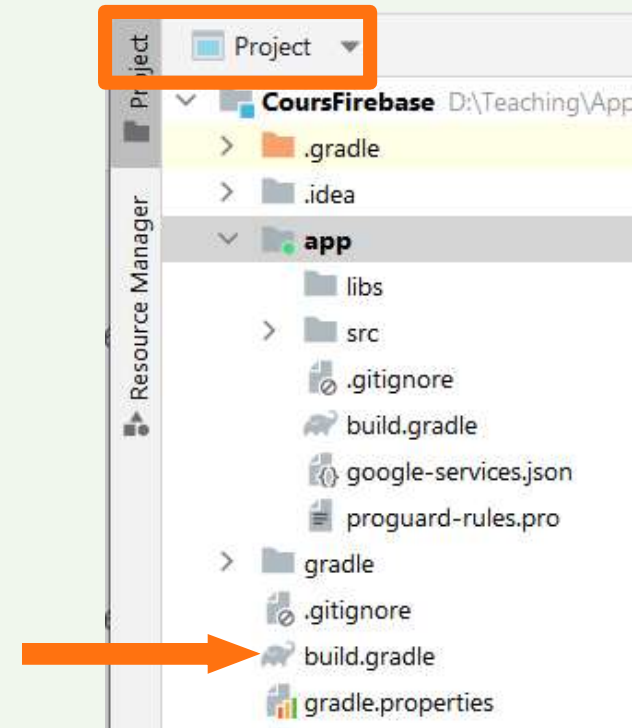


## 2. Ajouter Firebase à une application Android

5. Pour intégrer les bibliothèques Firebase dans un projet (**Android Studio 1.5.1**):
- ajouter des règles au fichier build.gradle au niveau du projet (**<project>/build.gradle**) pour inclure le plug-in google-services :

3

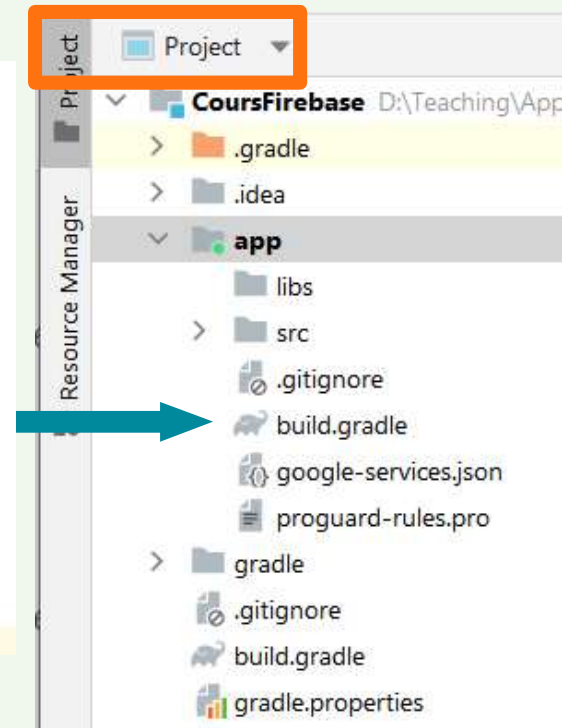
```
allprojects {  
    repositories {  
        jcenter()  
        maven {  
            url 'https://maven.google.com'  
        }  
    }  
}
```



## 2. Ajouter Firebase à une application Android

6. Pour intégrer les bibliothèques Firebase dans un projet (**Android Studio 1.5.1**):
- ajouter des règles au fichier build.gradle au niveau du projet (**<app>/build.gradle**) pour inclure le plug-in google-services :

```
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    testCompile 'junit:junit:4.12'  
    compile 'com.android.support:appcompat-v7:23.1.1'  
    compile 'com.google.firebase:firebase-auth:9.0.2'  
    compile 'com.google.firebase:firebase-core:9.0.2'  
    compile 'com.google.firebase:firebase-database:9.0.2'  
}  
  
apply plugin: 'com.google.gms.google-services'
```



## 2. Ajouter Firebase à une application Android

7. **Synchroniser le projet**, ça peut prendre **quelques minutes** pour **télécharger** les **libraries**.

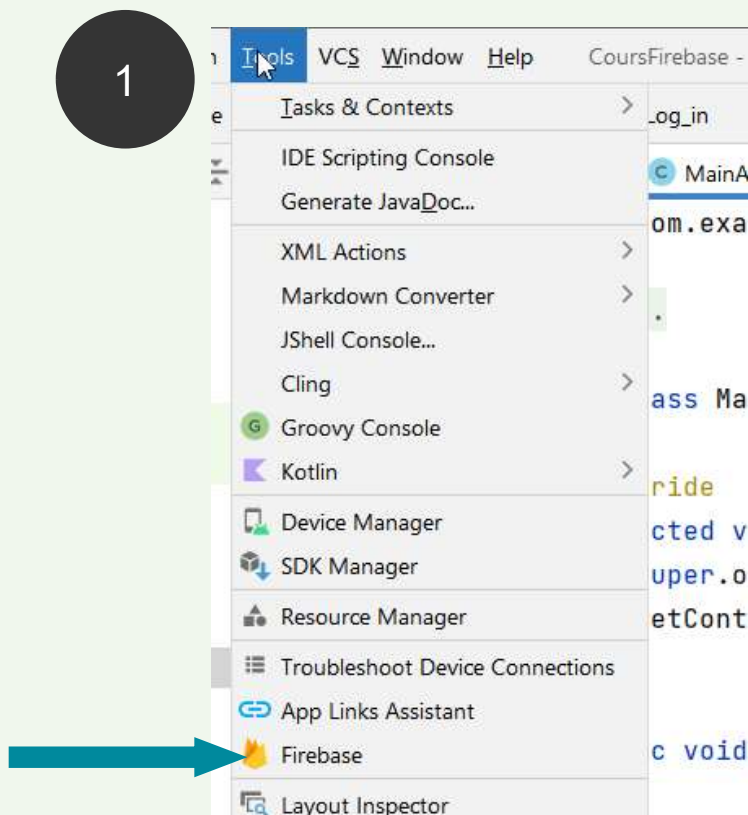
# Bibliothèques Firebase compatibles avec Android Studio 1.5.1

- Plusieurs bibliothèques sont disponibles pour les différentes fonctionnalités Firebase.

Ligne de dépendance Gradle	Service
com.google.firebase: firebase-core: 9.0.2	Analytique
com.google.firebase: firebase-database: 9.0.2	Base de données en temps réel
com.google.firebase: firebase-storage: 9.0.2	Espace de rangement
com.google.firebase: crash-firebase: 9.0.2	Rapport de collision
com.google.firebase: firebase-auth: 9.0.2	Authentification
com.google.firebase: firebase-messaging: 9.0.2	Cloud Messaging / Notifications
com.google.firebase: firebase-config: 9.0.2	Configuration à distance
com.google.firebase: invite-firebase: 9.0.2	Invites / Liens dynamiques
com.google.android.gms: play-services-appindexing:9.0.2	Indexation des applications

## 2. Ajouter Firebase à une application Android

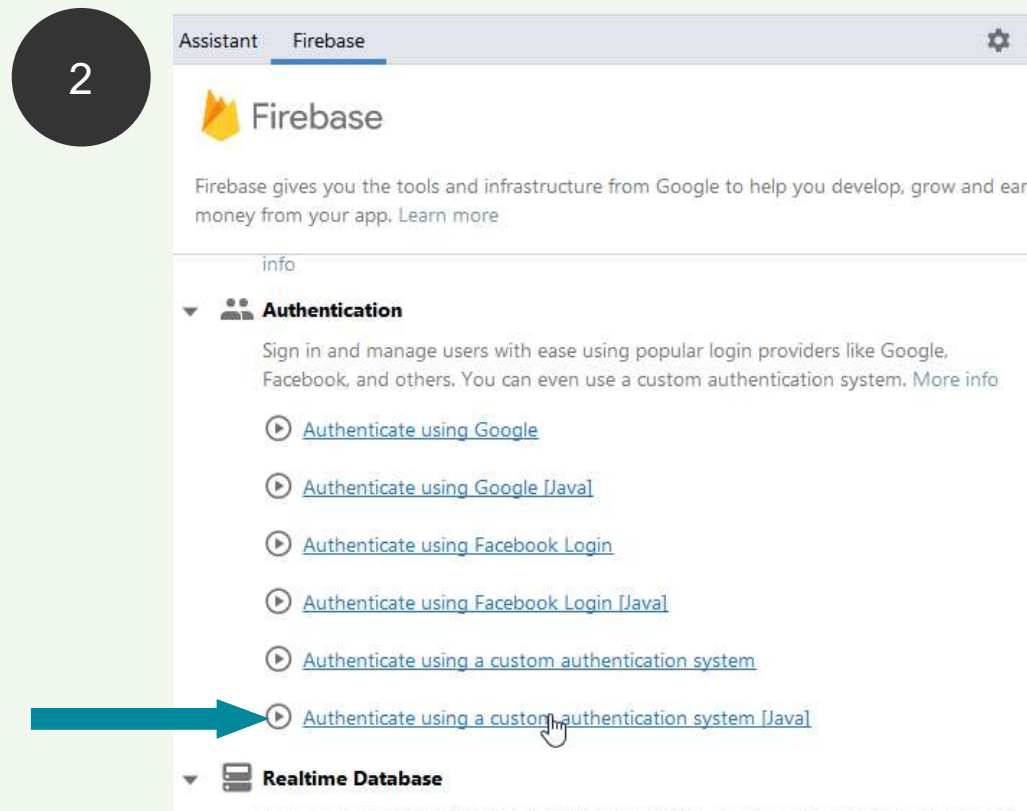
5. Pour intégrer les bibliothèques Firebase dans un projet (**Android Studio Chipmunk**):
- o Aller à Tools -> Firebase





## 2. Ajouter Firebase à une application Android

5. Pour intégrer les bibliothèques Firebase dans un projet (**Android Studio Chipmunk**):
- Sélectionner **Authentication using a custom authentication system (Java)**



## 2. Ajouter Firebase à une application Android

5. Pour intégrer les bibliothèques Firebase dans un projet (**Android Studio Chipmunk**):
- Cliquer sur **Connect** puis **Add the Firebase authentication SDK to your app**. Cela permettra d'intégrer automatiquement les bibliothèques Firebase dans le projet.

3

### Authenticate using a custom authentication

You can integrate Firebase Authentication with a custom authentication system by modifying your server-side tokens when a user successfully signs in. Your app receives this token and uses it to authenticate with the Firebase Authentication SDK.

[Launch in browser](#)

- 1 Connect your app to Firebase  
✓ Connected
- 2 Add the Firebase Authentication SDK to your app  
[Add the Firebase Authentication SDK to your app](#)
- 3 Get your Firebase project's server keys

**NOTE:** After adding the SDK, here are some other helpful configurations to consider:

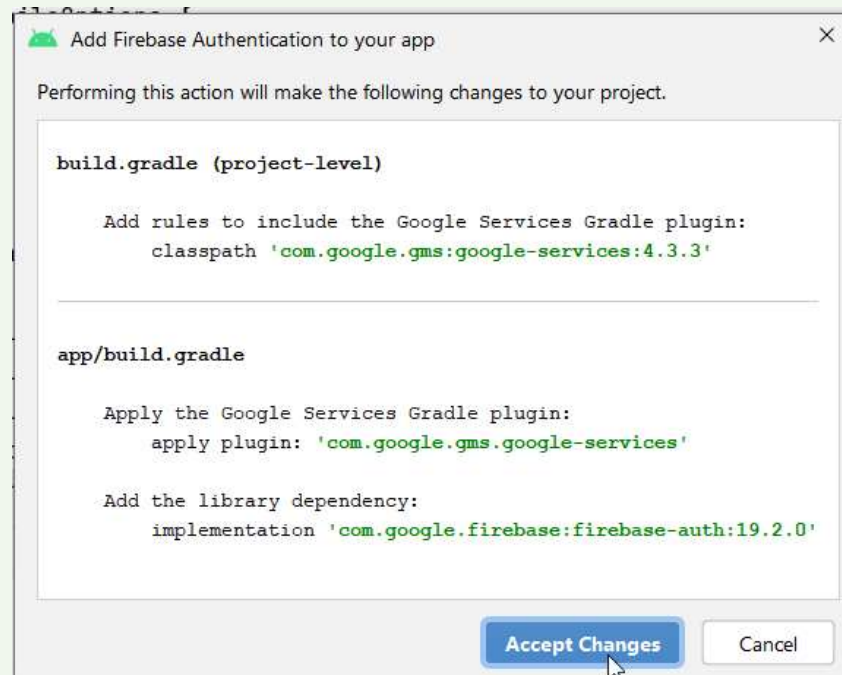
- **Do you want an easier way to manage library versions?**  
You can use the [Firebase Android BoM](#) to manage your Firebase library versions and ensure you are using compatible library versions.

1. Go to the [Service accounts](#) tab in your **Project settings** in the Firebase console.

## 2. Ajouter Firebase à une application Android

5. Pour intégrer les bibliothèques Firebase dans un projet (**Android Studio Chipmunk**):
- o Un message de confirmation s'affiche. Ces modifications (les rules ajoutées) **dépendent de la version d'Android Studio**.

4





**02**

# **FIREBASE AUTHENTICATION**

# Firebase Authentication

- **Firebase Authentication** est un **service d'authentification** facile à utiliser et entièrement géré qui permet aux utilisateurs de se connecter à l'application Android en utilisant leurs comptes Google, Facebook, Twitter et bien plus encore.
- Il offre une **sécurité solide**, une intégration **facile** et une personnalisation étendue.
- **La classe FirebaseAuth** est la principale classe de Firebase Authentication qui permet d'authentifier les utilisateurs dans votre application Android.
- Elle fournit des méthodes pour gérer les utilisateurs, les sessions d'authentification, les fournisseurs d'authentification et bien plus encore.

# Ajouter une authentification par email (Firebase console)

AVANT DE COMMENCER A CODER: Activer un moyen d'authentification via l'interface web de Firebase:

The screenshot displays the Firebase Authentication console interface. On the left is a dark sidebar with navigation options: 'Vue d'ensemble du p...', 'Authentification', 'Créer', 'Publier et surveiller', 'Analytics', 'Engager', and 'Tous les produits'. The main area is titled 'Authentication' and includes tabs for 'Users', 'Sign-in method' (which is active), 'Templates', 'Usage', 'Settings', and 'Extensions' (marked 'NOUVEAU'). Below the tabs, a table titled 'Fournisseurs de connexion' is visible, with columns for 'Fournisseur' and 'État'. A modal window titled 'Sélectionner un fournisseur de connexion (étape 1 sur 2)' is open, showing a list of providers. The 'Adresse e-mail/Mot de passe' provider is highlighted with a checkmark and a mouse cursor. Other providers listed include 'Téléphone', 'Anonyme', 'Google', 'Facebook', 'Play Jeux', 'Game Center', 'Apple', 'GitHub', 'Microsoft', 'Twitter', 'Yahoo!', 'OpenID Connect', and 'SAML'. A button 'Ajouter un fournisseur' is located in the top right of the modal.

# Initialisation de l'instance FirebaseAuth

- Afin d'utiliser la classe **FirebaseAuth** dans l'application Android, il est primordial d'initialiser l'instance FirebaseAuth en appelant la méthode **getInstance()** :

```
FirebaseAuth mAuth = FirebaseAuth.getInstance();
```

# Fonctionnalités principales

Voici les principales fonctionnalités de la classe **FirebaseAuth** :

- Création d'un utilisateur;
- Connexion d'un utilisateur;
- Vérification de la session d'authentification;
- Déconnexion d'un utilisateur;
- Réinitialisation du mot de passe;
- Suppression d'un utilisateur;
- Authentification anonyme;
- Envoyer des e-mails de vérification;
- Ajout de fournisseurs d'authentification;



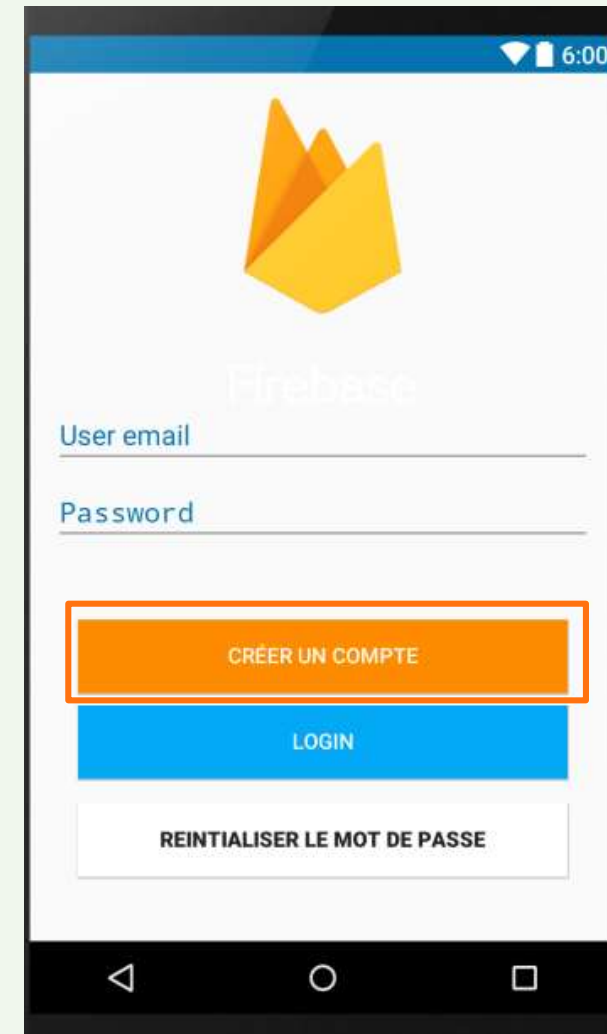
# Création d'un utilisateur

```
public void signup(View view){
    String user = ((EditText)findViewById(R.id.user_Text)).getText().toString();
    String pass = ((EditText)findViewById(R.id.pass_text)).getText().toString();

    if(user.isEmpty()) {
        Toast.makeText(AuthActivity.this, "Entrer une @ email", Toast.LENGTH_SHORT).show();
        return;
    }
    if(pass.isEmpty()) {
        Toast.makeText(AuthActivity.this, "Entrer un password", Toast.LENGTH_SHORT).show();
        return;
    }
}
```

```
FirebaseAuth auth = FirebaseAuth.getInstance();
auth.createUserWithEmailAndPassword(user,pass).addOnCompleteListener(
    new OnCompleteListener<AuthResult>() {
        @Override
        public void onComplete(Task<AuthResult> task) {
            if (task.isSuccessful()) {
                Toast.makeText(AuthActivity.this, "Compte créer avec succès, " +
                    "Login pour continuer", Toast.LENGTH_SHORT).show();
            } else
                Toast.makeText(AuthActivity.this, "Echec:" + task.getException().getMessage(),
                    Toast.LENGTH_SHORT).show();
        }
    });
}
```

Afficher le msg d'erreur



# Création d'un utilisateur

```
FirebaseAuth auth = FirebaseAuth.getInstance();  
auth.createUserWithEmailAndPassword(user, pass);
```

- La librairie Firebase fournit un objet Singleton **FirebaseAuth.getInstance()** bien pratique, contenant plusieurs mécanismes:
  - Une méthode permettant de créer un compte utilisateur **createUserWithEmailAndPassword(user,pass)**.
  - Une méthode permettant de connecter/déconnecter un utilisateur **signInWithEmailAndPassword(user,pass)**, et **signOut()**.
  - Une méthode permettant de récupérer l'utilisateur actuellement connecté à l'application **getCurrentUser()**.

# Création d'un utilisateur

```
auth.createUserWithEmailAndPassword(user,pass).addOnCompleteListener(  
    new OnCompleteListener<AuthResult>() {  
        @Override  
        public void onComplete(Task<AuthResult> task) {  
            if (task.isSuccessful()) {  
                Toast.makeText(AuthActivity.this, "Compte créer avec succès, " +  
                    "Login pour continuer", Toast.LENGTH_SHORT).show();  
            } else  
                Toast.makeText(AuthActivity.this, "Echec:" + task.getException().getMessage(),  
                    Toast.LENGTH_SHORT).show();  
        }  
    }  
);
```

- **onComplete(task)** est appelée après l'exécution de la tâche « création d'un compte utilisateur ».
  - Cas de succès: afficher un message;
  - Cas d'échec: afficher l'erreur.

# Création d'un utilisateur

- Une fois inscrit, le compte **de l'utilisateur est créé** sur Firebase:

Raccourcis de projet

 Authentication


Catégories de produits

Créer ▾

Publier et surveiller ▾

Analytics ▾




Engager ▾

 Tous les produits

Users Sign-in method Templates Usage Settings Extensions **NOUVEAU**

Recherchez par adresse e-mail, numéro de téléphone ou ID utilisateur

**Ajouter un utilisateur**

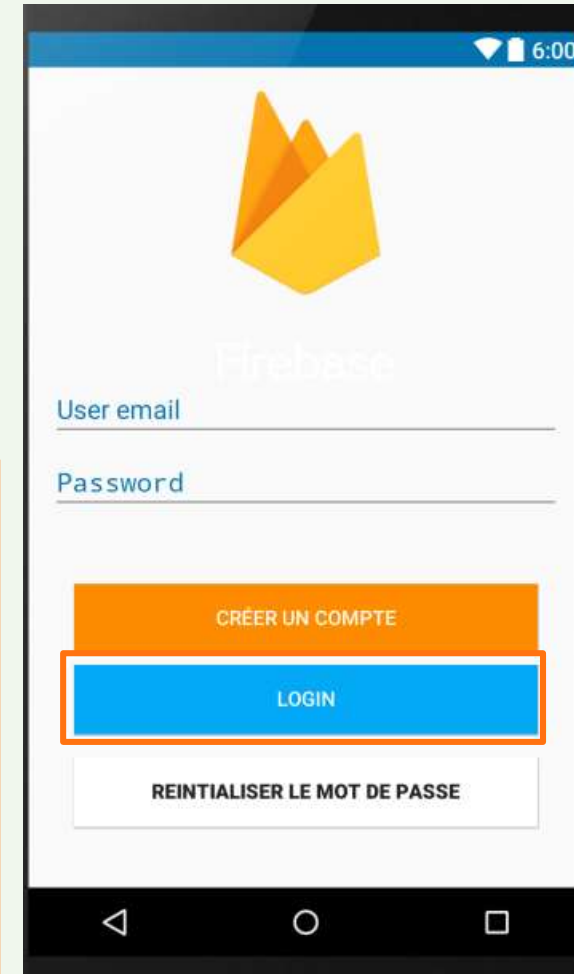
Identifiant	Fournisseurs	Date de création	↓	Dernière connexion	UID utilisateur
ykhlef_hadjer@univ-blida.dz		28 mars 2023		28 mars 2023	Ik7eWweM8RRLi3Ud9u9LBhf9kTz1
l3siq2019@gmail.com		28 mars 2023		28 mars 2023	itHLPB9cQJdKIAR43Ly2m8NoBVA2
ykhlef.monitoring@gmail.c...		28 mars 2023		28 mars 2023	o0WHjbQ0ZASMTcNc5lk05W8340...

Lignes par page : 50 1 – 3 of 3

# Connexion d'un utilisateur

```
public void login(View view){
    String user = ((EditText)findViewById(R.id.user_Text)).getText().toString();
    String pass = ((EditText)findViewById(R.id.pass_text)).getText().toString();
    if(user.isEmpty()) {
        Toast.makeText(AuthActivity.this, "Entrer une @ email", Toast.LENGTH_SHORT).show();
        return;
    }
    if(pass.isEmpty()) {
        Toast.makeText(AuthActivity.this, "Entrer un password", Toast.LENGTH_SHORT).show();
        return;
    }
    FirebaseAuth auth = FirebaseAuth.getInstance();
    auth.signInWithEmailAndPassword(user, pass).addOnCompleteListener(
        new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(Task<AuthResult> task) {
                if (task.isSuccessful()) {
                    startActivity(new Intent(getApplicationContext(), AnotherActivity.class));
                    finish();
                } else {
                    Toast.makeText(AuthActivity.this, "Echec:" + task.getException().getMessage(),
                        Toast.LENGTH_SHORT).show();
                }
            }
        }
    );
}
```

Afficher le msg d'erreur



# Vérification de la session d'authentification

- La méthode `getCurrentUser()` de FirebaseAuth permet de **vérifier** si un utilisateur est **connecté** à l'application Android.
- Cette méthode renvoie un objet **FirebaseUser** qui représente l'utilisateur connecté.
- Si aucun utilisateur n'est connecté, la méthode renvoie **null**.

```
FirebaseUser currentUser = mAuth.getCurrentUser();
if (currentUser != null) {
    // Utilisateur connecté
    // ...
} else {
    // Utilisateur non connecté
    // ...
}
```



# Déconnexion d'un utilisateur

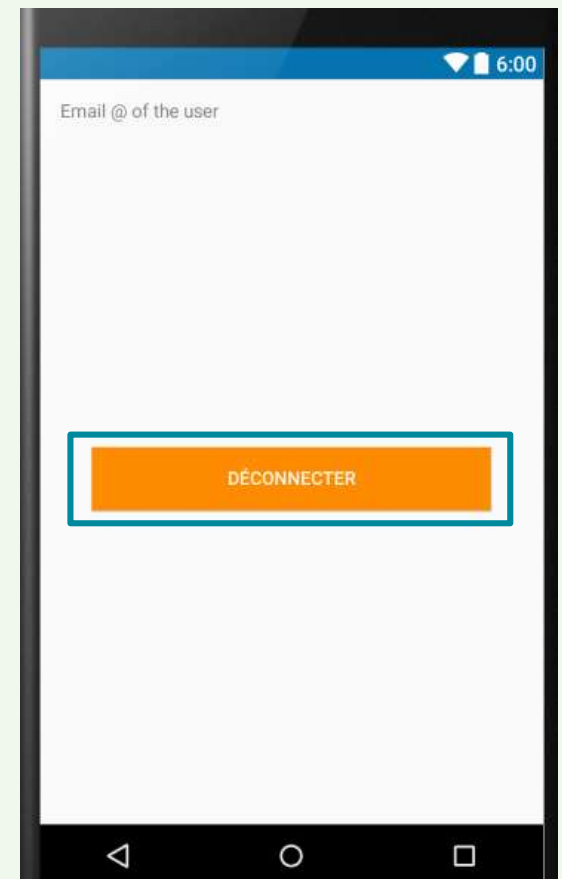
- Une fois l'utilisateur se connecte, l'activité **AnotherActivity** est affichée:

```
public class AnotherActivity extends AppCompatActivity {  
    FirebaseAuth auth;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_another);  
  
        auth = FirebaseAuth.getInstance();  
        FirebaseUser user = auth.getCurrentUser();  
        TextView textView = (TextView) findViewById(R.id.textView);  
        textView.setText(user.getDisplayName()+" "+user.getEmail());  
    }  
}
```

Profile de l'utilisateur

```
public void signout(View view){  
    auth.signOut();  
    startActivity(new Intent(getApplicationContext(), AuthActivity.class));  
    finish();  
}
```

Logout



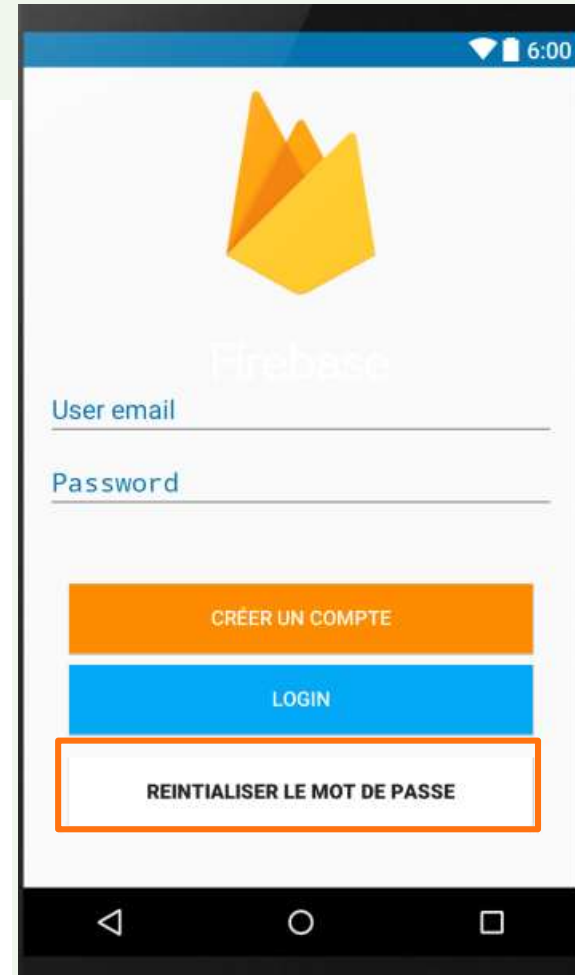
# Réinitialisation du mot de passe

```
public void resetpass(View view){
    String user = ((EditText)findViewById(R.id.user_Text)).getText().toString();

    if(user.isEmpty()) {
        Toast.makeText(AuthActivity.this, "Entrer une @ email", Toast.LENGTH_SHORT).show();
        return;
    }

    FirebaseAuth auth = FirebaseAuth.getInstance();

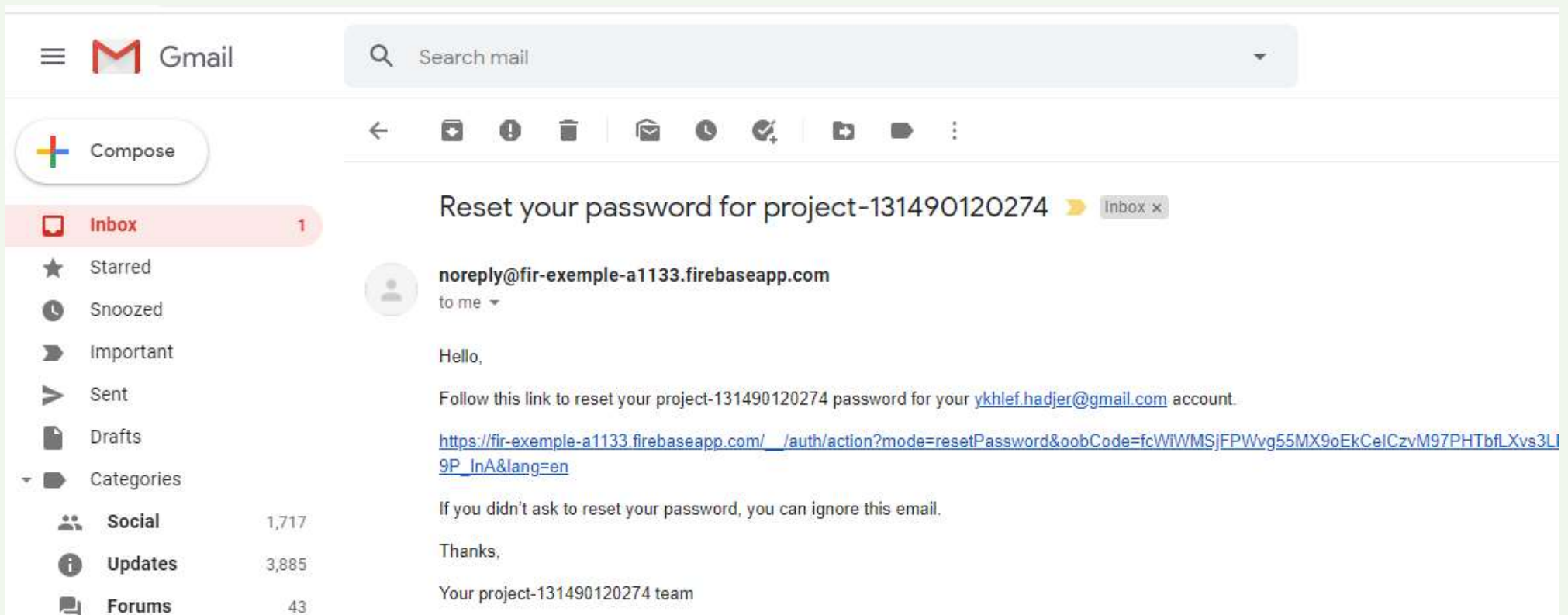
    auth.sendPasswordResetEmail(user).addOnCompleteListener(
        new OnCompleteListener() {
            @Override
            public void onComplete(Task<Void> task) {
                if (task.isSuccessful()) {
                    Toast.makeText(AuthActivity.this,
                        "Reset email envoyé avec succès", Toast.LENGTH_SHORT).show();
                } else
                    Toast.makeText(AuthActivity.this,
                        "Echec:" + task.getException().getMessage(), Toast.LENGTH_SHORT).show();
            }
        }
    );
}
```





# Réinitialisation du mot de passe

- Une fois la méthode **sendPasswordResetEmail** est exécutée, le backend **Firestore** génère un email de réinitialisation de mot de passe:



# Suppression d'un utilisateur

- La méthode `delete()` de `FirebaseUser` permet de supprimer un utilisateur.

```
FirebaseUser user = mAuth.getCurrentUser();
user.delete()
    .addOnCompleteListener(new OnCompleteListener<Void>() {
        @Override
        public void onComplete(@NonNull Task<Void> task) {
            if (task.isSuccessful()) {
                // Utilisateur supprimé avec succès
                // ...
            } else {
                // Erreur lors de la suppression de l'utilisateur
                // ...
            }
        }
    });
```

# Authentication anonyme

- La méthode `signInAnonymously()` de `FirebaseAuth` crée un compte d'utilisateur anonyme pour l'utilisateur actuel, qui est stocké dans `Firebase Authentication`.
- L'utilisateur peut ensuite se connecter à son compte anonyme ultérieurement en appelant la méthode `signInAnonymously()` à nouveau.

```
FirebaseAuth.getInstance().signInAnonymously()
    .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
        @Override
        public void onComplete(@NonNull Task<AuthResult> task) {
            if (task.isSuccessful()) {
                // Connexion anonyme réussie
            } else {
                // Erreur de connexion anonyme
            }
        }
    });
```

The background is a solid teal color. On the left side, there is a large, dark grey, thick-lined circle. Inside this circle, there is a smaller teal circle. A small white circle is positioned at the bottom left of the dark grey circle. A small circle with diagonal hatching is located at the top left of the dark grey circle. In the top right corner, there is a small orange circle and a larger orange shape that resembles a stylized exclamation mark or a thick, slanted line.

**03**

# **FIREBASE REALTIME DATABASE**

# Firestore Realtime Database

- **Firestore Realtime Database** est une base de données **NoSQL** en temps réel hébergée dans le cloud, fournie par Firebase.
- Elle permet de **stocker** et de **synchroniser** des données en **temps réel** entre les **utilisateurs d'une application**.
- Firestore Realtime Database est une base de données **orientée clé-valeur**, où les données sont stockées sous forme de **paires clé-valeur** en temps réel.

# Base de données NoSQL

- Une **base de données NoSQL (Not Only SQL)** est un système de gestion de base de données qui diffère des systèmes de gestion de bases de données relationnelles (RDBMS) traditionnels en ce sens qu'elle **ne repose pas** sur un **modèle de données tabulaire fixe**.
- Contrairement aux RDBMS, les bases de données NoSQL permettent de stocker des données **non structurées, semi-structurées et structurées de manière flexible**.
- Ces bases de données sont souvent utilisées pour gérer des **données volumineuses**, car elles sont plus **évolutives** et plus performantes.
- Elles sont également utilisées pour les **applications web et mobiles**, les applications de réseaux sociaux et les applications de **l'Internet des objets (IoT)**.

# Configurer la base de données

- Activer la base de données dans la console Firebase



# Configurer la base de données

- Configurer les règles d'accès à la base de données :

### Configurer une base de données

Options de la base de données

2 Règles de sécurité

Après avoir défini la structure de vos données, vous devrez spécifier des règles pour les sécuriser.  
[En savoir plus](#)

☐ Commencer en mode verrouillé  
Par défaut, vos données sont privées. L'accès client en lecture/écriture ne sera autorisé qu'en fonction de vos règles de sécurité.

☒ Démarrer en mode test  
Par défaut, vos données sont publiques pour permettre une configuration rapide. Toutefois, vous devez mettre vos règles de sécurité à jour dans les 30 jours pour autoriser l'accès client en lecture/écriture sur le long terme.

```
{  "rules": {    ".read": "now < 1682722800000",    // 2023-4-29    ".write": "now < 1682722800000",    // 2023-4-29  } }
```

! Par défaut, les règles de sécurité en mode test autorisent tout utilisateur disposant de la référence de votre base de données à afficher, modifier et supprimer toutes les données qu'elle contient pendant les 30 prochains jours

Annuler

Activer



## **Configurer la base de données**

**Pas besoin de créer la structure de la base de données (colonnes, etc.), elle sera automatiquement créée lorsqu'un élément est inséré à la base de données.**

# Premier Exemple

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);
```

1

```
    // Initialisation de la base de données Firebase  
    FirebaseDatabase database = FirebaseDatabase.getInstance();  
    DatabaseReference myRef = database.getReference("message");
```

2

```
    // Lire depuis la base de données  
    myRef.addValueEventListener(new ValueEventListener() {  
        @Override  
        public void onDataChange(DataSnapshot dataSnapshot) {  
            // This method is called once with the initial value and again  
            // whenever data at this location is updated.  
            String value = dataSnapshot.getValue(String.class);  
            Log.i("onDataChange", "Valeur: " + value);  
        }  
        @Override  
        public void onCancelled(DatabaseError error) {  
            // Failed to read value  
            Log.i("onCancelled", "Impossible de lire la donnée.", error.toException());  
        }  
    });
```

3

```
    // Ecrire dans la base de données  
    myRef.setValue("Hello, World!");  
}
```

# Premier Exemple

- Résultat après Ecriture:



The screenshot displays the Firebase console interface. On the left, a dark sidebar contains the 'Firebase' logo and a navigation menu with options: 'Vue d'ensemble du projet', 'Développer', 'Authentication', 'Database' (highlighted in blue), 'Storage', 'Hosting', 'Functions', and 'ML Kit'. The main content area is titled 'Database' and shows 'Realtime Database' selected. Below this, there are tabs for 'Données', 'Règles', 'Sauvegardes', and 'Utilisation'. A URL bar at the top right shows 'https://fir-exemple-a1133.firebaseio.com/'. Below the URL bar, a console window titled 'fir-exemple-a1133' displays the JSON message: `{ "message": "Hello, World!" }`.

# Premier Exemple

- **Résultat après Ecriture:**



## Exemple 02

```
// Initialisation de la base de données Firebase
FirebaseDatabase database = FirebaseDatabase.getInstance();
DatabaseReference myRef1 = database.getReference("N1");
DatabaseReference myRef2 = database.getReference("N2");

// Ecrire dans la base de données
myRef1.setValue("Hello, World!");
myRef2.setValue("Salam, Alikom!");
```

Résultat après Ecriture



<https://fir-exemple-a1133.firebaseio.com/>

**fir-exemple-a1133**

```
{
  N1: "Hello, World!",
  N2: "Salam, Alikom!"
}
```

# Initialisation de l'instance FirebaseDatabase

- **getInstance():** cette méthode statique retourne une instance de la classe **FirebaseDatabase** qui peut être utilisée pour interagir avec la base de données Firebase.

```
public static FirebaseDatabase getInstance()
```

# DatabaseReference

- La classe **DatabaseReference** dans Firebase Realtime Database pour Android est une **référence** à un emplacement spécifique de la base de données en temps réel.
- Elle permet de **lire, écrire, supprimer et écouter les données** de la base de données.

# getReference()

- **getReference()** : cette méthode retourne une référence à la racine de la base de données Firebase.

```
public DatabaseReference getReference()
```

- **getReference(String path)** : cette méthode retourne une référence à un nœud spécifique de la base de données Firebase en utilisant le chemin spécifié.

```
public DatabaseReference getReference(String path)
```



# getReference() et child()

- **DatabaseReference child(String path)** : Cette méthode renvoie une référence à l'emplacement spécifié en tant qu'enfant de la référence actuelle.
- Elle prend en paramètre une chaîne de caractères représentant le nom de l'enfant à récupérer.

```
DatabaseReference ref = FirebaseDatabase.getInstance().getReference();  
DatabaseReference usersRef = ref.child("users");
```

- Ce code crée une instance de la classe **DatabaseReference** à la racine de la base de données en temps réel de Firebase.
- Puis, il crée une **nouvelle référence** à l'emplacement "**users**" de la base de données, qui est un enfant de la référence racine créée précédemment.

# DatabaseReference: Lecture de données

- **addListenerForSingleValueEvent()** : Cette méthode permet de lire une fois les données à l'emplacement spécifié. Elle prend un objet **ValueEventListener** en paramètre qui contient les données lues.
- **addValueEventListener()** : Cette méthode permet d'écouter les changements de données à l'emplacement spécifié en temps réel. Elle prend un objet **ValueEventListener** en paramètre qui est appelé chaque fois que les données sont modifiées.

# DatabaseReference: Ecriture de données

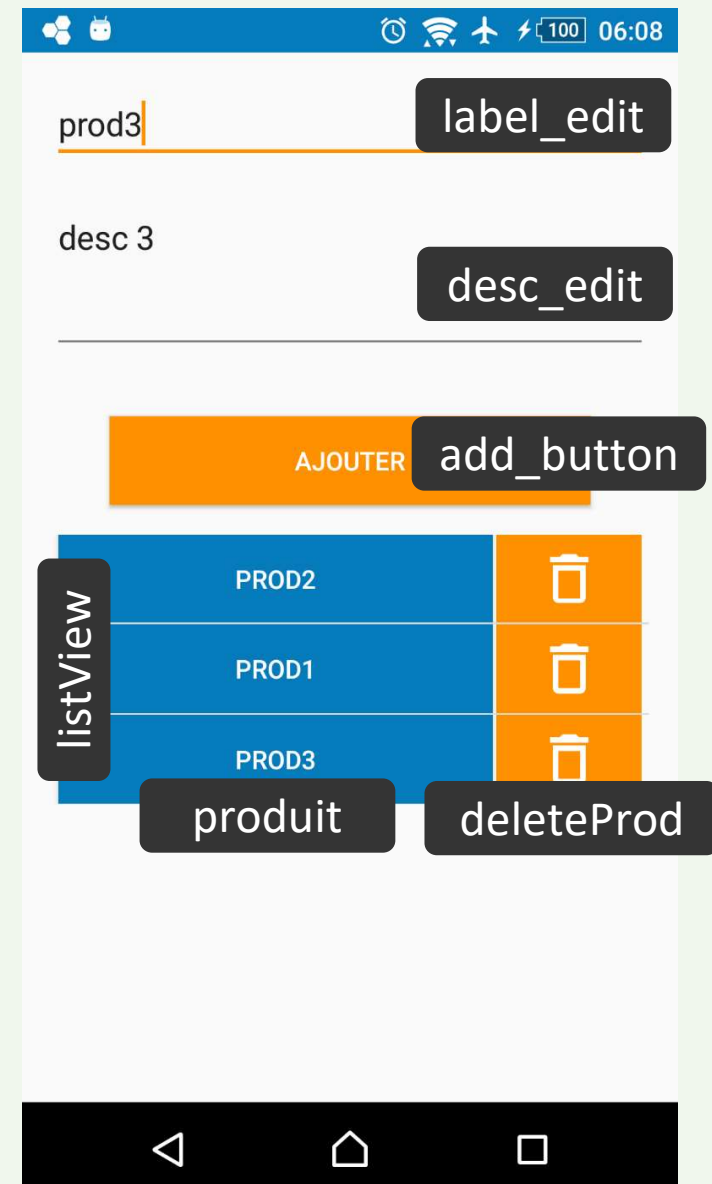
- **setValue()** : Cette méthode permet d'écrire des données à l'emplacement spécifié. Elle prend un objet de données en paramètre.
- **updateChildren()** : Cette méthode permet de mettre à jour les données à l'emplacement spécifié. Elle prend un objet Map en paramètre, qui contient les données à mettre à jour.
- **push()** : Cette méthode génère une nouvelle clé unique pour chaque objet de données que vous écrivez. Elle est souvent utilisée pour stocker plusieurs enregistrements de données sous le même nœud.

# DatabaseReference: Supprimer de données

- **removeValue()** : Cette méthode permet de supprimer les données à l'emplacement spécifié.

# Exemple Complet

- L'objectif est de réaliser une app permettant de gérer une **liste des produits**.
- Un produit est caractérisé par un **id (String)**, **label (String)**, **description (String)**.
- Pour stocker la liste des produit, l'app utilise une Firebase RealTime DataBase.



# Exemple Complet

Voici un **exemple de JSON** qui montre comment les données sont stockées dans la base de données:

- Contrairement aux bases de données relationnelles, Firebase stocke les données au **format JSON**.
- Chaque ligne d'une base de données relationnelle comme un objet JSON (qui est essentiellement une **paire clé-valeur non ordonnée**).
- Le nom de la **colonne** devient **clé** et la **valeur stockée dans cette colonne pour une ligne particulière est la valeur**. De cette manière, la ligne entière est représentée comme un objet JSON et une liste de ceux-ci représente une table de base de données entière.



# Exemple Complet: Classe Produit

```
public class Produit {  
    private String id;  
    private String label;  
    private String description;  
  
    public Produit() {}  
  
    public Produit(String id, String label, String description) {  
        this.label = label;  
        this.description = description;  
        this.id = id;  
    }  
  
    public Produit(String label, String description) {  
        this.label = label;  
        this.description = description;  
    }  
}
```

```

public class MainActivity extends AppCompatActivity {
    private DatabaseReference productdataset;
    private ArrayList<Produit> listProduit;
    private MyListAdapter myListAdapter;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        listProduit = new ArrayList<Produit>();
        myListAdapter = new MyListAdapter(MainActivity.this, listProduit);
        ListView listProdView = (ListView)findViewById(R.id.listView);
        listProdView.setAdapter(myListAdapter);

        productdataset = FirebaseDatabase.getInstance().getReference("Produits");
        productdataset.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                listProduit.clear();
                for (DataSnapshot d : dataSnapshot.getChildren()) {
                    Produit produit = d.getValue(Produit.class);
                    listProduit.add(produit);
                }
                myListAdapter.notifyDataSetChanged();
            }
            @Override
            public void onCancelled(DatabaseError databaseError) { }
        });
    }
}

```

## Exemple Complet: MainActivity : onCreate



# Exemple Complet: Écoute des mises à jour de la base Firebase

```
productdataset.addValueEventListener(new ValueEventListener() {  
    @Override  
    public void onDataChange(DataSnapshot dataSnapshot) {  
        listProduit.clear();  
        for (DataSnapshot d : dataSnapshot.getChildren()) {  
            Produit produit = d.getValue(Produit.class);  
            listProduit.add(produit);  
        }  
    }  
});
```

- Une application de chat ou une application de liste de produit collaborative souvent nécessite la synchronisation entre les utilisateurs.

# Exemple Complet: Écoute des mises à jour de la base Firebase

```
productdataset.addValueEventListener(new ValueEventListener() {  
    @Override  
    public void onDataChange(DataSnapshot dataSnapshot) {  
        listProduit.clear();  
        for (DataSnapshot d : dataSnapshot.getChildren()) {  
            Produit produit = d.getValue(Produit.class);  
            listProduit.add(produit);  
        }  
    }  
});
```

- Si on utilise une **base de données Firebase** et on ajoute **un écouteur d'événement** de valeur au nœud parent ou au nœud parent de liste de produits, on finira **avec toute la structure de la base** (chaque fois qu'un nœud est ajouté).

# Exemple Complet: Écoute des mises à jour de la base Firebase

```
productdataset.addValueEventListener(new ValueEventListener() {  
    @Override  
    public void onDataChange(DataSnapshot dataSnapshot) {  
        listProduit.clear();  
        for (DataSnapshot d : dataSnapshot.getChildren()) {  
            Produit produit = d.getValue(Produit.class);  
            listProduit.add(produit);  
        }  
    }  
});
```

On utilise ChildEventListener ou ValueEventListener:

- **onDataChange ()** donne la valeur enfant entière avec les propriétés modifiées (nouvelles).
- L'objet **dataSnapshot** regroupe toutes les entrées de la table Produits.
- **dataSnapshot.getChildren()** permet de parcourir la liste des items de la table.

# Exemple Complet: Ajouter un nouveau produit

```
public void ajouter_firebase(View view) {
    String label= ((EditText)findViewById(R.id.label_edit)).getText().toString();
    String desc= ((EditText)findViewById(R.id.desc_edit)).getText().toString();

    if(label.isEmpty() || desc.isEmpty()) {
        Toast.makeText(MainActivity.this, "Merci de remplir les champs",
            Toast.LENGTH_SHORT).show();
        return;
    }

    //Recupérer un id unique pour le nouv produit
    String id = productdataset.push().getKey();

    //Insérer le nouv produit
    productdataset.child(id).setValue(new Produit(id,label,desc)).
        addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(Task<Void> task) {
                if (task.isSuccessful())
                    Toast.makeText(MainActivity.this, "Produit ajouté avec succès"
                        , Toast.LENGTH_SHORT).show();
                else
                    Log.i("onComplete", task.getException().getMessage());
            }
        });
}
```

# Exemple Complet: Ajouter un nouveau produit

<https://fir-exemple-a1133.firebaseio.com/>

fir-exemple-a1133

Produits

Clé (key) du prod2

Valeur  
du  
prod2

-M6T4B00\_AXG7HuUM\_m-

description: "desc 2"  
id: "-M6T4B00\_AXG7HuUM\_m"  
label: "prod2"

-M6T4MmcjEKXSUpTJR0o

description: "desc 3"  
id: "-M6T4MmcjEKXSUpTJR0"  
label: "prod1"

-M6T401SsJFutcueLyVQ

description: "desc 3"  
id: "-M6T401SsJFutcueLyV"  
label: "prod3"

1

```
//Recupérer un id unique pour le nouv produit  
String id = productdataset.push().getKey();
```

**push()** génère un nom nouveau et unique du nœud qui renverra une autre DatabaseReference , qui à son tour pointe sur un nœud nouvellement formé pour insérer le produit.

2

```
//Insérer le nouv produit  
productdataset.child(id).setValue(new Produit(id,label,desc))
```

**child(id)** permet d'obtenir une référence au nœud nouvellement crée.

**setValue(Object)** permet d'insérer un objet, ou mettre à jour sa valeur si id existe déjà dans la base.

# Exemple Complet: Supprimer un produit

```
public void supp_firebase(View view) {  
    int pos = (int)view.getTag();  
    productdataset.child(listProduit.get(pos).getId())  
        .removeValue(new DatabaseReference.CompletionListener() {  
            @Override  
            public void onComplete(DatabaseError databaseError,  
                                   DatabaseReference databaseReference) {  
                if (databaseError == null)  
                    Toast.makeText(MainActivity.this, "Produit supp avec succès",  
                                   Toast.LENGTH_SHORT).show();  
                else  
                    Log.i("onComplete", databaseError.toString());  
            }  
        });  
    listProduit.remove(pos);  
    myListAdapter.notifyDataSetChanged();  
}
```



# Exemple Complet: MyArrayAdapter

```
class MyListAdapter extends ArrayAdapter<Produit> {
    private Activity context;
    private ArrayList<Produit> listProduits;

    public MyListAdapter(Activity context, ArrayList<Produit> listProduits) {
        super(context, R.layout.ma_ligne, listProduits);
        this.context = context;
        this.listProduits = listProduits;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        if (convertView == null) {
            LayoutInflater inflater = context.getLayoutInflater();
            convertView = inflater.inflate(R.layout.ma_ligne, null);
        }

        Button produit_button = (Button) convertView.findViewById(R.id.produit);
        produit_button.setText(listProduits.get(position).getLabel());

        ImageButton deleteProd = (ImageButton) convertView.findViewById(R.id.deleteProd);

        produit_button.setTag(position);
        deleteProd.setTag(position);

        return convertView;
    }
}
```