Firebase Authentication

Firebase のサービスには、Firestore Database や Realtime Database のようにデータの管理以外にも、アプリ作成に伴う様々なサービスが提供されています。
今回は、そのサービスの中から認証処理を行う「Authentication」について学習を進めます。

・Firebase Authentication ドキュメント
https://firebase.google.com/docs/auth?hl=ja&authuser=0

・Firebase のサービス（一部抜粋

カテゴリ	サービス名	概要
構築(ビルド)	Authentication	認証のバックエンドサービス
構築(ビルド)	Storage	オブジェクトストレージサービス
構築(ビルド)	Cloud Messaging	プッシュメッセージサービス
リリースと モニタリング	Analytics	ユーザエンゲージメントの測定と分析

フェデレーションの必要性

ユーザ情報が必要なシステムの場合、ログイン機能はほぼ必須で実装しなければなりません。しかしながら、ユーザ情報は個人情報そのもので、エンジニアはユーザの管理とともにセキュリティを意識する必要があります。

また、利用者はサービスごとにアカウント登録とパスワード管理に煩わしさを感じるため近年では、Google などフェデレーション(インターネットサービス間のユーザ認証連携)を提供している企業がいくつかあります。

Firebase Authentication はパスワード、電話番号、Google、Facebook、Twitter などの一般的なフェデレーション ID プロバイダーを使用した認証をサポートしています。

具体的には、FirebaseUI もしくは、Firebase Authentication SDK を使用してアプリケーションのサインインを実装します。
FirebaseUI は、Firebase が用意している画面を呼び出して認証処理をお任せすることが出来て、Firebase Authentication SDK は、複数のサインイン方法を手動で認証サービスの統合をすることが出来ます。

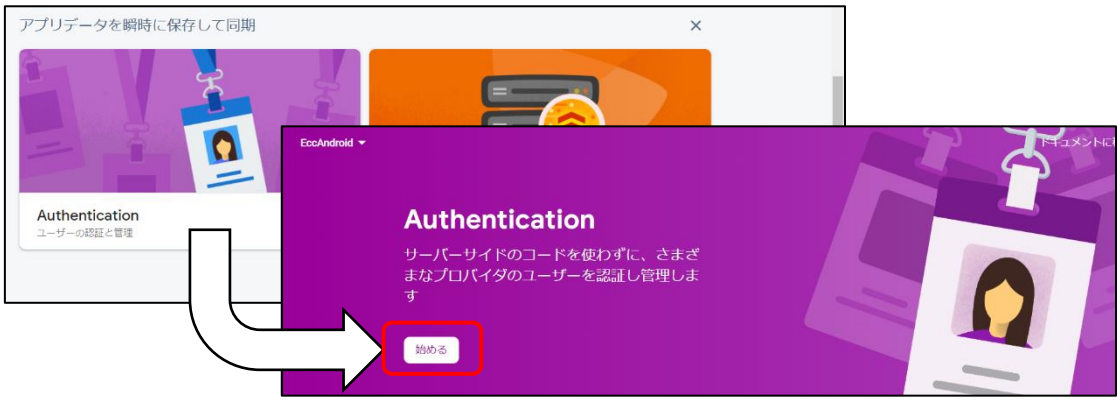
・FirebaseUI 認証と Firebase SKD 認証について

FirebaseUI 認証	
ドロップイン認証ソリューション	Firebase 推奨のサインイン実装方法。 メールアドレスとパスワード、電話番号、および Google や Facebook ログインなどの一般的なフェデレーション ID プロバイダーを使用してユーザをサインインさせるための UI フローを処理するドロップイン認証ソリューションを提供します。
Firebase SDK 認証	
メール、パスワード認証	メール アドレスとパスワードを使用してサインインするユーザを作成および管理する方法を提供します。
フェデレーション ID プロバイダーの統合	Firebase Authentication SDK は、ユーザが Google、Facebook、Twitter、および GitHub アカウントでサインインできるようにするメソッドを提供します。
電話番号認証	SMS メッセージを電話に送信してユーザを認証します。
カスタム認証システムの統合	既存のサインインシステムを Authentication SDK に接続し、Realtime Database やその他の Firebase サービスにアクセスします。
匿名認証	一時的な匿名アカウントを作成して、ユーザが最初にサインインする必要なく、認証を必要とする機能を提供します。

また、従量制にアップグレードをすることにより、「多要素認証」や「ブロッキング機能」も利用することが出来ます。

ハンズオン　Firebase Authentication を有効化する

1．Firebase のコンソール画面から Authentication ページに遷移する



2．メール/パスワード認証を有効化する

FirebaseUI

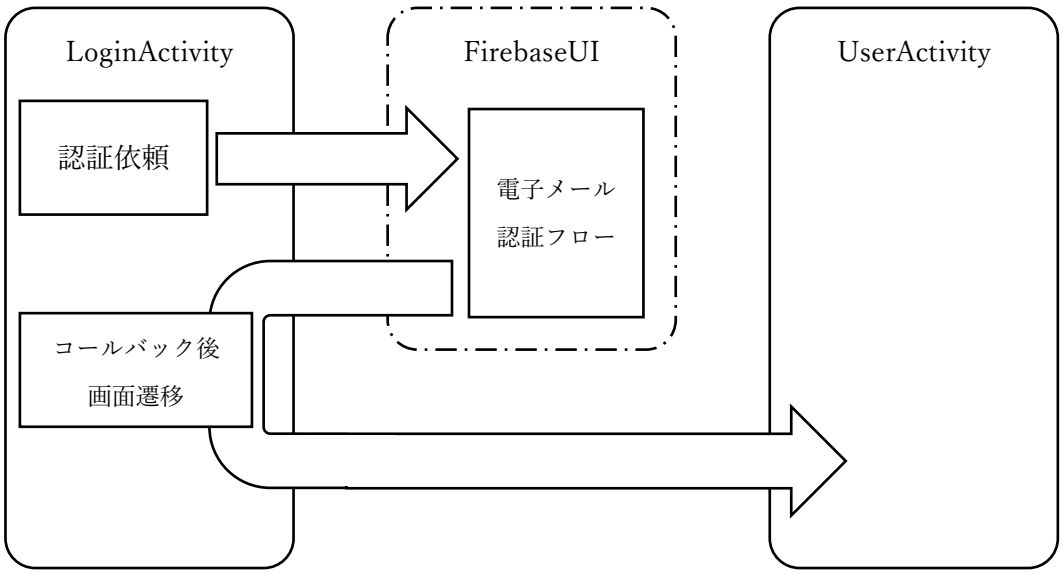
FirebaseUI は、Authentication SDK の上に構築されたライブラリであり、アプリで使用するためのドロップイン UI フローを提供します。 FirebaseUI には、次の利点があります。

メリット	説明
複数プロバイダー対応	電子メール/パスワードだけでなく、電子メールリンク、電話認証、Google サインインなどのサインフローを提供
アカウント管理	アカウントの作成やパスワードのリセットなどのアカウント管理タスクを処理するためのフローを提供
アカウントリンク	ID プロバイダー間でユーザーアカウントを安全にリンクするためのフローを提供
匿名ユーザアップデート	匿名ユーザーを安全にアップグレードするためのフローを提供
カスタムテーマ	アプリに合わせて FirebaseUI の外観をカスタマイズすることが可能
Smart Lock for Passwords	Smart Lock for Passwords との自動統合により、デバイス間の高速サインインが可能になります

Android で FirebaseUI を導入する場合は、アプリレベルの build.gradle に依存関係を追記することで利用することが出来ます。

ハンズオン　FirebaseUI で電子メール/パスワード認証を実装する

・ハンズオン実装イメージ



1．FirebaseUI の依存関係をアプリレベルの build.gradle に追記する

※HandsOnFirebase プロジェクトで作業してください。

```
implementation 'com.google.android.gms:play-services-auth:20.7.0'
implementation 'com.firebaseui:firebase-ui-auth:8.0.2'
```

・build.gradle(:app)

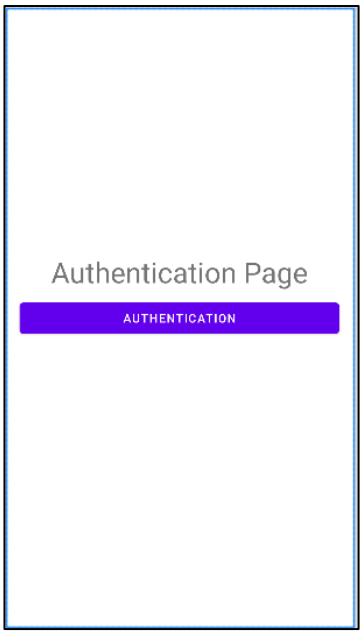
```
dependencies {

    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.5.1'
    implementation 'com.google.android.material:material:1.7.0'
    implementation 'com.google.android.gms:play-services-auth:19.2.0' // SDK 31以上の場合必要
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    implementation platform('com.google.firebase:firebase-bom:31.0.3') // Firebase基本セット
    implementation 'com.google.firebase:firebase-analytics-ktx'        // Analytics
    implementation 'com.firebaseui:firebase-ui-auth:7.2.0'             // FirebaseUI
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```

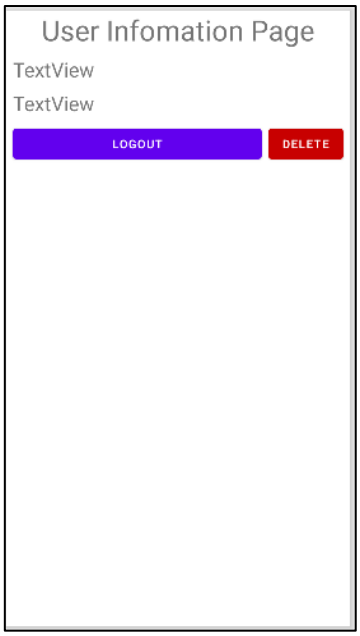
FirebaseUI Auth SDK は、FirebaseSDK と GooglePlay サービス SDK に推移的に依存しています。

2．LoginActivity および UserActivity を生成する。
3．画面デザインは配布データにある activity_login.xml、activity_user.xml を適用させる。

・LoginActiviy



・UserActivity



4．LoginActivity 内で、FirebaseUI を起動させるためのランチャー(発射台)の設定を行う

```
// FirebaseUI のランチャー設定
private val signInLauncher = registerForActivityResult(
    FirebaseAuthUIActivityResultContract()) { res ->
        // this.onSignInResult(res)
    }
```

※onSignInResult はまだ実装していないため、コメントで記載

このランチャー内で、FirebaseUI の起動を行っており、正常に処理が完了すれば
onSignInResult メソッドを呼び出すようにプログラムしています。

5．onCreate メソッドに認証ボタンのクリックイベントを実装する。
6．クリックイベントで、FirebaseUI に処理してもらう認証プロバイダーの設定を行う

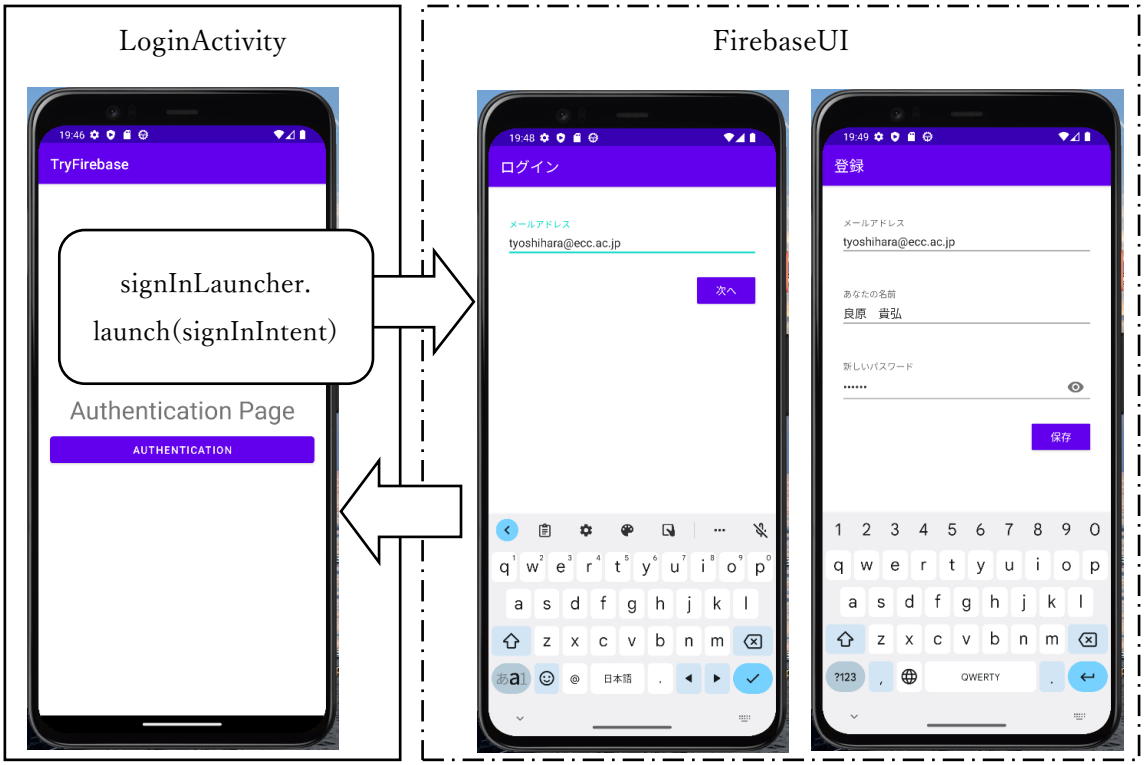
```
// 認証プロバイダー設定
val providers = arrayListOf(
        AuthUI.IdpConfig.EmailBuilder().build()
)
```

※FirebaseUI に電子メール認証の依頼を設定しています。

7．サインインインテントの生成を行い、ランチャーにセットして FirebaseUI を起動する。

```
// サインインインテントをランチャーにセットして FirebaseUI を起動
val signInIntent = AuthUI.getInstance()
    .createSignInIntentBuilder()
    .setAvailableProviders(providers)
    .build()
signInLauncher.launch(signInIntent)
```

8．AndroidManifest を修正して起動画面を LoginActivity に変更する



ここまでの処理で、FirebaseUI を使った認証を実装することが出来ました。
今のままだと認証後もログイン画面が表示されるので正常に認証出来れば、ユーザ情報画面に遷移する処理をを実装します。

ハンズオン　認証完了後に画面遷移を行う

1．LoginActivity 内で FirebaseUI による認証処理完了後に呼び出される
コールバックメソッドを実装する

```
// FirebaseUI 処理後、呼び出されるメソッド
private fun onSignInResult(result: FirebaseAuthUIAuthenticationResult) {

}
```

2．引数のリザルトコードをチェックして正常終了の場合、ユーザ情報を取得して
UserActivity の画面遷移を行う。

```
if (result.resultCode == RESULT_OK) {
    // 認証ユーザ情報の取得
    val user = FirebaseAuth.getInstance().currentUser

    // 認証出来ていれば次の画面遷移する
    user?.let {
        val nextIntent = Intent(this, UserActivity::class.java)
        nextIntent.putExtra("userName", it.displayName)
        nextIntent.putExtra("email", it.email)
        startActivity(nextIntent)
    }

}
```

３．リザルトコードが正常終了以外の場合は、エラー対応の処理を記述する

```
} else {
    // サインインに失敗しました。応答が null の場合、
    // ユーザーは [戻る] ボタンを使用してサインイン フローをキャンセルしました。
    // それ以外の場合は、response.getError().getErrorCode() を確認してエラーを
    // 処理してください。
    val response = result.idpResponse
    if(response == null){
        Toast.makeText(applicationContext, "認証がキャンセルされました",
    Toast.LENGTH_SHORT).show()
    }else{
        response.error?.let{
            Log.e("err",it.toString())
        }
    }
}
```

※else 文は if 分の続きで記述すること！！

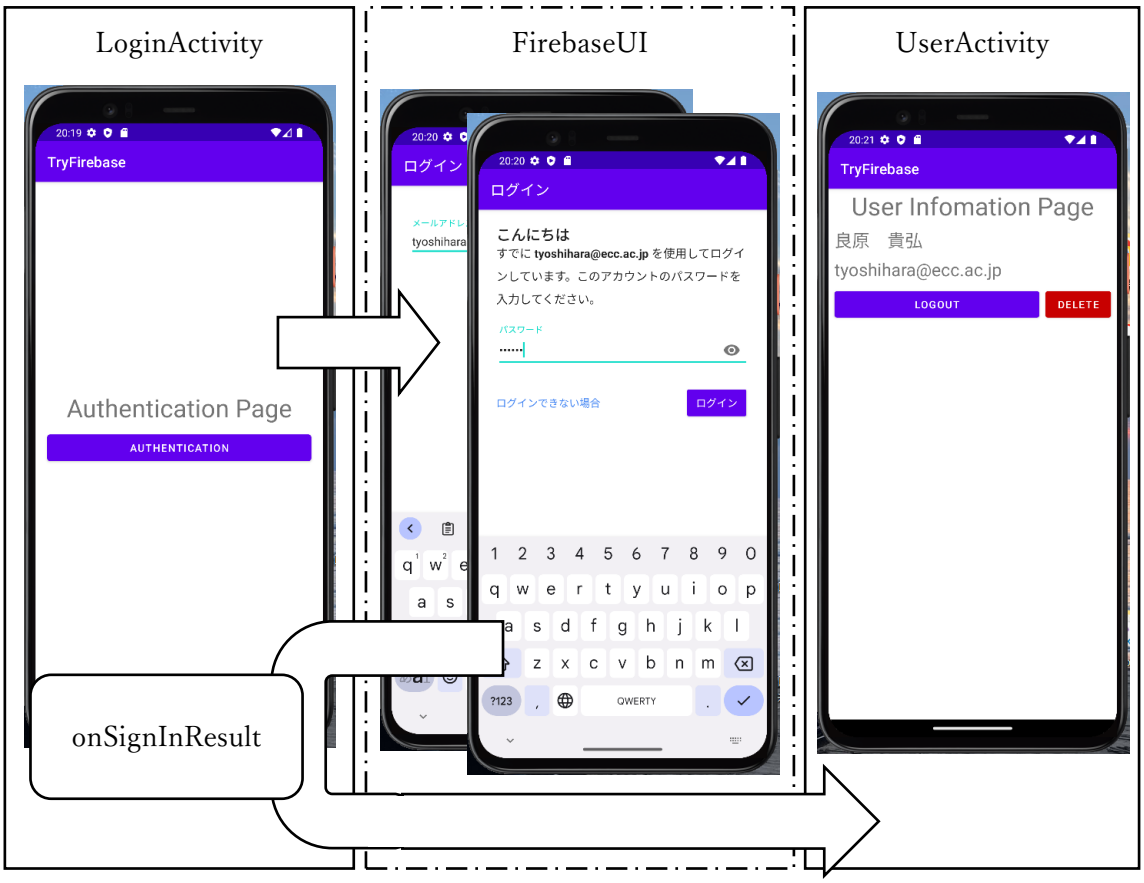
４．ランチャー設定の onSignInResult メソッドのコメントを外して有効化する

```
    // this.onSignInResult(res)
    ↓
    this.onSignInResult(res)
```

５．UserActivity の onCreate メソッド内で前画面から送られたパラメータを
画面に表示する。

```
// 前画面からユーザ情報を受け取る
val myIntent = intent;
val user = myIntent.getStringExtra("userName")
val email = myIntent.getStringExtra("email")

val userTv = findViewById<TextView>(R.id.userTextView)
val emailTv = findViewById<TextView>(R.id.emailTextView)
userTv.text = user
emailTv.text = email
```



フェデレーションプロバイダーによる認証

先ほどのハンズオンで、電子メール/パスワードによる認証を実装することが出来ました。
ここからさらに、フェデレーションプロバイダーによる認証の実装を学習します。フェデレーションプロバイダーによる認証は、プロバイダーごとに設定が必要となります。
今回は、最も設定が安易な Google アカウントによる認証を行います。

Google 認証では、フィンガープリントと呼ばれる電子証明書の発行が必要で、
デバック用なら Android Studio のコマンドプロンプトから発行することが出来ます。

まずは、フィンガープリントの発行を行い、そのフィンガープリントを Firebase プロジェクトに登録するところから進めていきます。

ハンズオン　Google アカウント認証を実装する

1．Android Studio のコンソール画面からフィンガープリント発行コマンドを実行する

keytool -list -v -alias androiddebugkey -keystore C:¥Users¥ユーザ名¥.android¥debug.keystore

```
\TryFirebase>
\TryFirebase> keytool -list -v -alias androiddebugkey -keystore C:\Users\70795\.android\debug.keystore
```

※コマンド実行後のパスワードは「android」です。

Java の Path が通っていない場合はそのままで使えない為

cd で C:¥Users¥ユーザ名¥.jdks¥openjdk-21.0.1¥bin

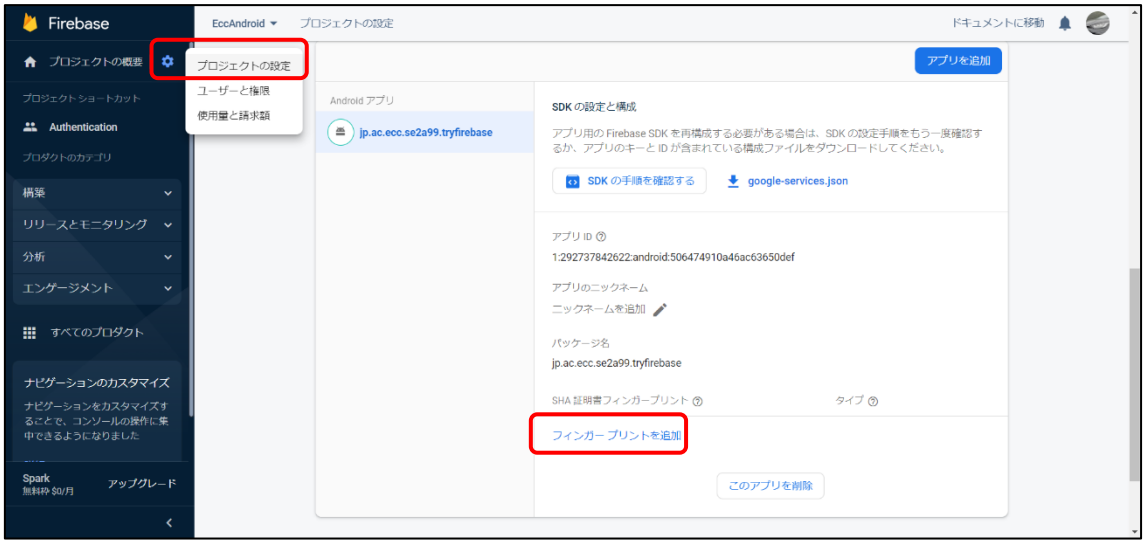
辺りに移動してから keytool コマンドを実行してください

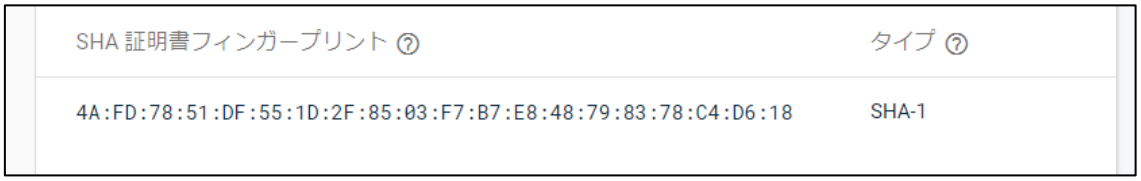
2．発効された SHA1 のハッシュキーをコピーする

```
シリアル番号： 1
有効期間の開始日： Mon Sep 26 15:29:40 JST 2022終了日： Wed Sep 18 15:29:40 JST 2052
証明書のフィンガプリント：
         SHA1: 4A:FD:78:51:DF:55:1D:2F:85:03:F7:B7:E8:48:79:83:78:C4:D6:18
         SHA256: 72:21:DE:9B:7C:AE:10:02:E9:E4:B2:86:F8:45:68:48:E9:61:43:73:2C:79:FE:10:BB:9C:3C:91:A2:5E:3E:E9
署名アルゴリズム名： SHA1withRSA (弱)
サブジェクト公開キー・アルゴリズム： 2048ビット RSAキー
バージョン： 1
```

※各自、発行されたものをコピーすること！！

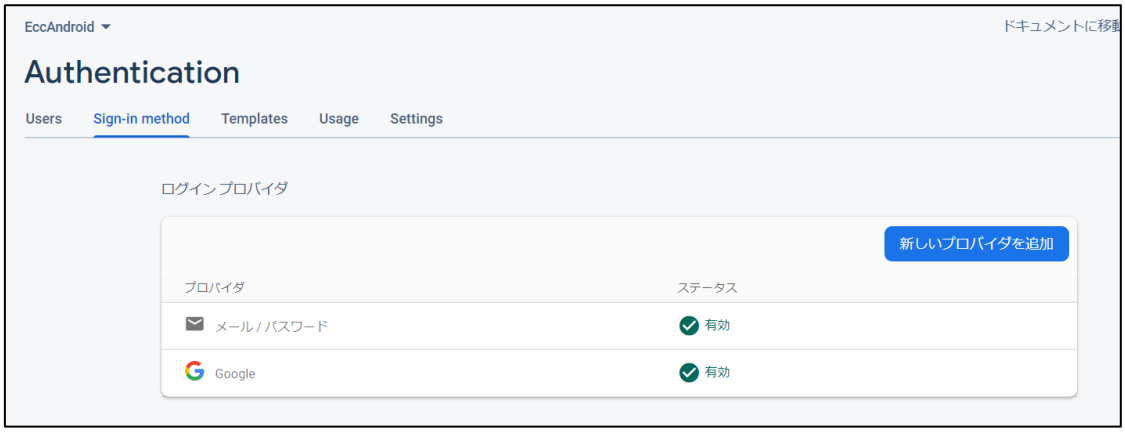
3．Firebase コンソール画面のプロジェクト設定から、フィンガープリントの追加を行う。





4．Authentication から Google 認証のログインプロバイダーを追加する

※プロジェクトのサポートメールは、自身の Gmail アドレスを設定してください。



5．認証プロバイダーの設定に Google 認証を追加する。

```
// 認証プロバイダー設定
val providers = arrayListOf(
    AuthUI.IdpConfig.GoogleBuilder().build(), // Google 認証追加
    AuthUI.IdpConfig.EmailBuilder().build()
)
```

6．FirebaseUI のデザインのカスタマイズを行う

```
// サインインインテントをランチャーにセットして FirebaseUI を起動
val signInIntent = AuthUI.getInstance()
    .createSignInIntentBuilder()
    .setAvailableProviders(providers)
    .setLogo(R.mipmap.ic_launcher)       // UI にアイコンを表示
    .setTheme(R.style.Theme_TryFirebase)// テーマカラーをアプリに合わせる
    .build()
signInLauncher.launch(signInIntent)
```