内容

[1 開発環境 3](#_Toc38908260)

[1.1 アンドロイドスタジオ（Android Studio 3.61.1） 3](#_Toc38908261)

[1.2 Google Cloud Pllatform 3](#_Toc38908262)

[2 アンドロイドAPPの作成 3](#_Toc38908263)

[2.1 「Bottom Navigation Activity」でプロジェクトを作成します。 3](#_Toc38908264)

[3 認証 4](#_Toc38908265)

[3.1 Firebase を Android プロジェクトに追加します。 4](#_Toc38908266)

[3.2 FirebaseUI の依存関係を追加 4](#_Toc38908267)

[3.3 Firebaseログイン方法 4](#_Toc38908268)

[4 Googleマップの表示 5](#_Toc38908269)

[4.1 APIキーを取得します。 5](#_Toc38908270)

[4.2 アプリにGoogleマップを追加します。 5](#_Toc38908271)

[5 位置情報追跡 6](#_Toc38908272)

[5.1 位置パーミッションをアプリに追加します。 6](#_Toc38908273)

[5.2 MyLocationManagerについて 6](#_Toc38908274)

[5.3 位置情報追跡を開始します。 6](#_Toc38908275)

[5.5 位置情報追跡を終了します。 7](#_Toc38908276)

[6 ジオコーディングについて 7](#_Toc38908277)

[7 位置情報の追跡 7](#_Toc38908278)

[7.1 Google Cloud Firestore APIを有効します。 7](#_Toc38908279)

[7.2 FireBaseに接続します。 8](#_Toc38908280)

[7.3 アプリにCloud Firestoreを追加します。 8](#_Toc38908281)

[8 位置情報をGoogle Firestoreに保存します。 8](#_Toc38908282)

[8.1 Cloud Firestoreを初期化します。 8](#_Toc38908283)

[8.2 Cloud Firestoreにデータを追加します。 8](#_Toc38908284)

[9 感染地域情報作成 8](#_Toc38908285)

[9.1 初期化します。 8](#_Toc38908286)

[9.2 感染者の位置履歴を即時感染エリアに追加します。 8](#_Toc38908287)

[9.3 感染者の位置履歴を即時感染エリアから削除します。 9](#_Toc38908288)

[9.4 即時感染エリア作成時、感染密度を更新します。 10](#_Toc38908289)

[9.5 即時感染エリア削除時、感染密度を更新します。 10](#_Toc38908290)

[9.6 感染報告された場合、位置履歴を感染エリアに更新します。 10](#_Toc38908291)

[10 自分の感染危険度について 13](#_Toc38908292)

[11 セキュリティ 13](#_Toc38908293)

[11.1 認証 13](#_Toc38908294)

[11.2 承認 13](#_Toc38908295)

[12 感染エリアの生成について 14](#_Toc38908296)

[13 自分の感染危険度について 14](#_Toc38908297)

# 開発環境

## アンドロイドスタジオ（Android Studio 3.61.1）

下記のURLを参照し、インストールします。

<https://developer.android.com/studio/install?authuser=1&hl=ja>

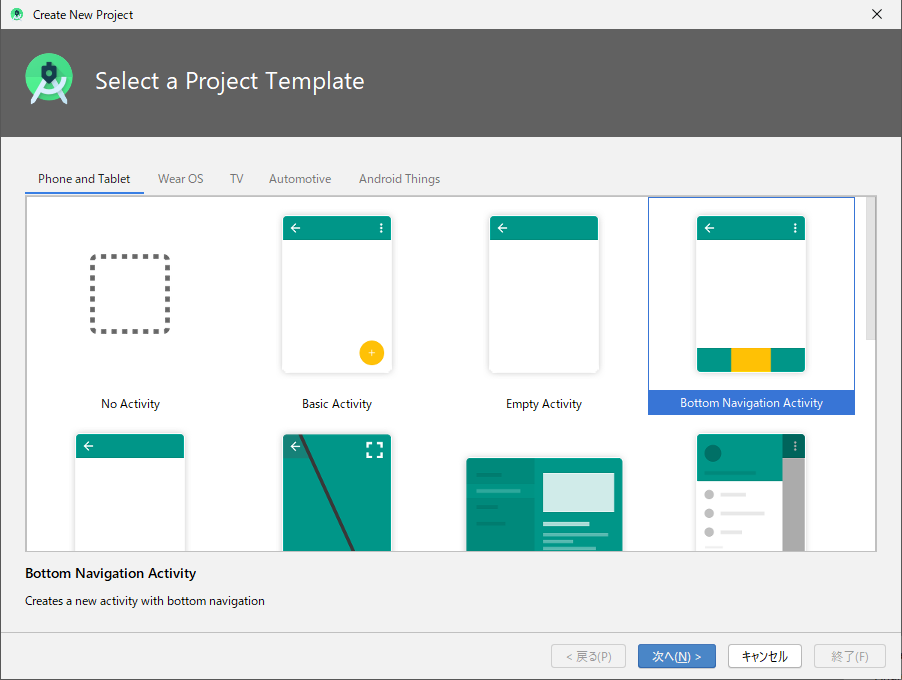
## Google Cloud Pllatform

下記のURLを参照し、アカウントを作成します。

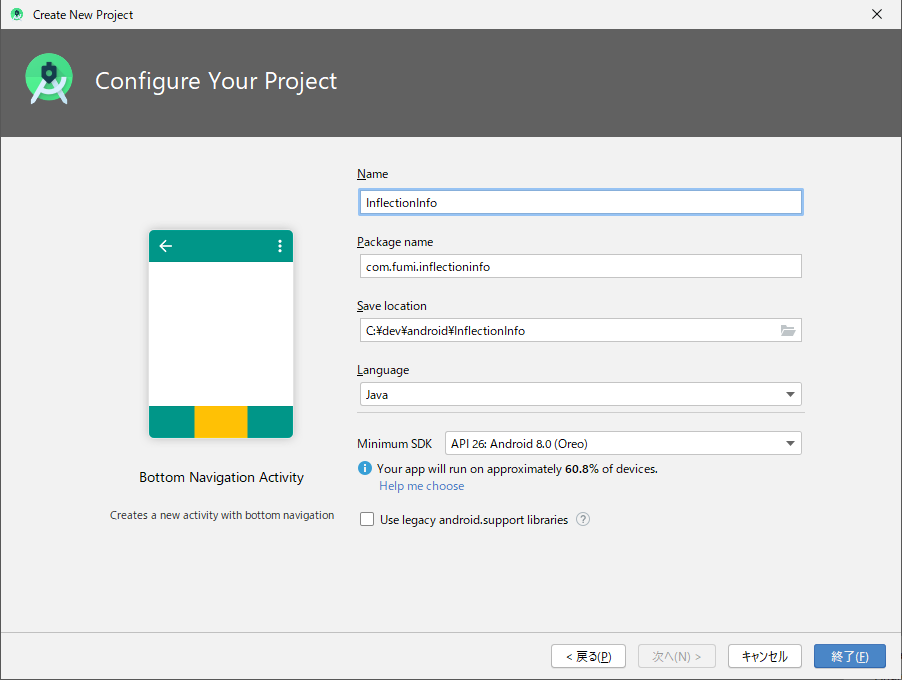
<https://cloud.google.com/docs/?hl=ja>

# アンドロイドAPPの作成

## 「Bottom Navigation Activity」でプロジェクトを作成します。



* 1. プロジェクト名などを入力します。



* 1. 位置情報追跡サービスを追加します。
  2. TODO

# 認証

## Firebase を Android プロジェクトに追加します。

## FirebaseUI の依存関係を追加

アプリレベルの build.gradle ファイルに FirebaseUI の依存関係を追加します。

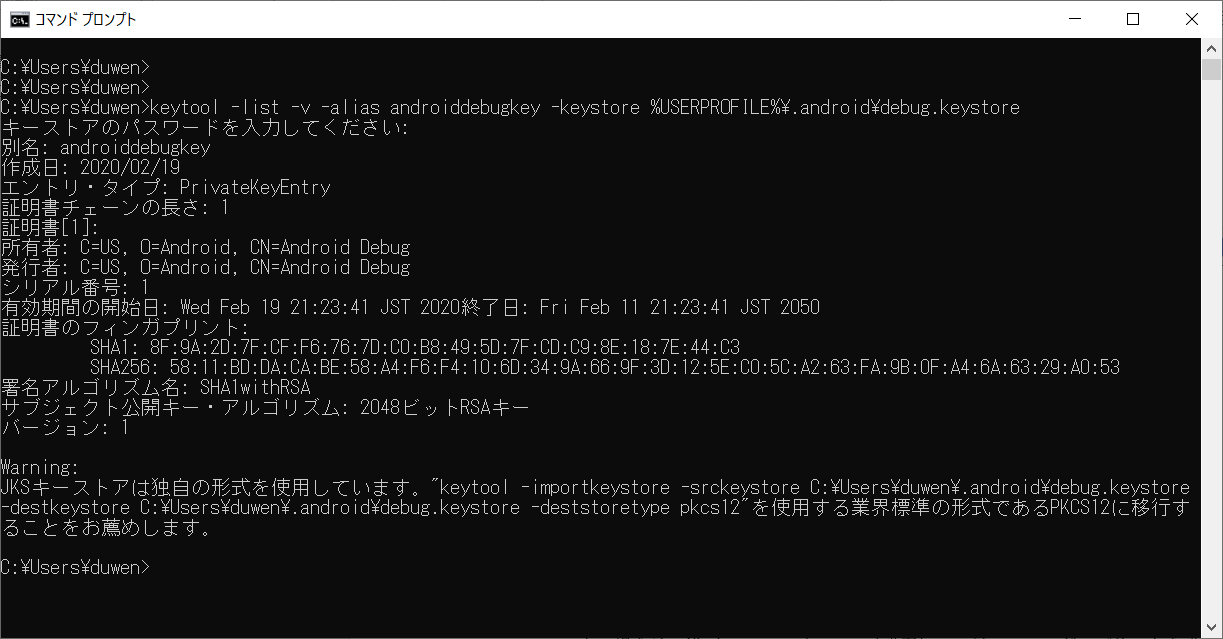
implementation 'com.firebaseui:firebase-ui-auth:4.3.1'

## Firebaseログイン方法

Firebase コンソールで [Authentication] セクションを開き、サポートするログイン方法を有効にします。

Google ログインをサポートし、アプリの SHA-1 フィンガープリントを指定していない場合は、Firebase コンソールの [設定] ページでフィンガープリントを設定します。

keytool -list -v -alias androiddebugkey -keystore %USERPROFILE%\.android\debug.keystore



# Googleマップの表示

## APIキーを取得します。

* + 1. Google Cloud Platform Consoleにログインします。
    2. メニュー「APIとサービス」の「ダッシュボード」をクリックします。
    3. メニュー「新庄情報」をクリックします。
    4. 画面上にある「+認証情報を作成」をクリックし、「APIキー」を選択します。
    5. 画面「APIキーを作成しました」のボタン「キーを制限」をクリックします。
    6. アプリケーションの制限には「Androidアプリ」を選択します。
    7. Android アプリに使用を限定を追加し、パッケージ名およびSHA-1証明書のフィガープリントを入力します。
    8. 画面「認証情報」から作成したAPIキーをコピーします。

## アプリにGoogleマップを追加します。

* 1. マニフェストファイルにAPIキーを設定します。
     + - 1. <meta-data

android:name="com.google.android.geo.API\_KEY"

android:value="@string/google\_maps\_key" />

* 1. 画面Layoutに追加します。

<fragment android:id="@+id/map"

android:name="com.google.android.gms.maps.SupportMapFragment"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".MainActivity" />

* 1. ソースコードに追加します。
     1. インタフェース「OnMapReadyCallback」を追加します。

public class DashboardFragment extends Fragment

implements OnMapReadyCallback {

* + 1. メソッド「onMapReady(GoogleMap googleMap)」を追加します。

public void onMapReady(GoogleMap googleMap) {

mMap= googleMap;

}

* + 1. メソッド「onCreateView」にListenerを登録します。

SupportMapFragment mapFragment =

(SupportMapFragment) getChildFragmentManager()

.findFragmentById(R.id.mapFragment);

mapFragment.getMapAsync(this);

# 位置情報追跡

## 位置パーミッションをアプリに追加します。

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

## MyLocationManagerについて

## 位置情報追跡を開始します。

locationManager = new MyLocationManager(getApplicationContext(), this);

locationManager.startLocationUpdates();

* 1. 位置情報追跡をします。

@Override

public void onLocationResult(LocationResult locationResult) {

if (locationResult == null) {

return;

}

for (Location location : locationResult.getLocations()) {

// …

}

}

## 位置情報追跡を終了します。

if (locationManager != null) {

locationManager.stopLocationUpdates();

}

# ジオコーディングについて

ジオコーディングとは地名や住所に地理座標を与えることであり、典型的には経緯度がそれにあたります。

様々な地理サービスやGPSで位置情報を示す値として使われています。

この経緯度とは別のジオコーディングの仕組みがOpen Location Code（OLC）です。

これは以下のような表記によって地球上の位置を表すことができます。

8Q9WCVW2+JFJ

このコードは以下のようにそれぞれの桁に意味があり、地球上を格子状に段階的に分割していき、それぞれの格子にコードを割り当てることで任意の地点を一意のコードで表現するようになっています。

これによるとコードの各桁は以下のような意味を持ちます。

8Q : 2200Km四方の範囲指定

9W : 110Km四方の範囲指定

CV : 5.5Km四方の範囲指定

W2 : 275m四方の範囲指定

JF : 14m四方の範囲指定

J : 3.5m四方の範囲指定

# 位置情報の追跡

位置情報をgoogle firestoreに保存します。

## Google Cloud Firestore APIを有効します。

Android Studioのメニュー「ツール」―＞「Firebase」をクリックします。

## FireBaseに接続します。

## アプリにCloud Firestoreを追加します。

implementation 'com.google.firebase:firebase-firestore:21.4.2'

## 位置情報をGoogle Firestoreに保存します。

## Cloud Firestoreを初期化します。

FirebaseAuth mFirebaseFirestore = FirebaseFirestore.getInstance();

## Cloud Firestoreにデータを追加します。

activityInfo.put("timestamp", timestamp);

GeoPoint gp =

new GeoPoint(location.getLatitude(), location.getLongitude());

activityInfo.put("location", gp);

activityInfo.put("locationcode", locCode);

// Add a new document with a generated ID;

mFirebaseFirestore.collection("users")

.document(currentUser.getEmail())

.collection("trace-infos")

.document(Constants.DATE\_FORMAT\_4\_NAME.format(

timestamp.toDate()))

.set(activityInfo)

……

# 感染地域情報作成

サーバー側機能のCloud Functions を使用します。

## 初期化します。

const functions = require('firebase-functions');

const admin = require('firebase-admin');

admin.initializeApp();

## 感染者の位置履歴を即時感染エリアに追加します。

exports.userTraceListener = functions

.region('asia-northeast1')

.firestore

.document('users/{userid}/trace-infos/{traceid}')

.onCreate(async (snap, context) => {

const statusRef = admin.firestore().collection('users')

.doc(context.params.userid)

.collection('infos').doc("status");

const status = await statusRef.get();

if (status.data().infection\_flag === 1) {

const inflectionRef = admin.firestore().collection('corona-infos')

.doc(snap.data()

.locationcode)

.collection('users')

.doc(context.params.userid);

return inflectionRef.set({

timestamp: snap.data().timestamp,

userid: context.params.userid,

});

}

return null;

});

## 感染者の位置履歴を即時感染エリアから削除します。

exports.userTraceListener = functions

.region('asia-northeast1')

.firestore

.document('users/{userid}/trace-infos/{traceid}')

.onDelete(async (snap, context) => {

const inflectionRef = admin.firestore().collection('corona-infos')

.doc(snap.data().locationcode)

.collection('users')

.doc(context.params.userid);

return inflectionRef.delete({

timestamp: snap.data().timestamp,

userid: context.params.userid,

});

});

## 即時感染エリア作成時、感染密度を更新します。

exports.addInflectionListener = functions

.region('asia-northeast1')

.firestore

.document('corona-infos/{locationcode}/users/{userid}')

.onCreate( async (snap, context) => {

const inflectionRef = admin.firestore().collection('corona-infos')

.doc(context.params.locationcode);

return inflectionRef.set({

density: admin.firestore.FieldValue.increment(1)},

{merge: true});

});

## 即時感染エリア削除時、感染密度を更新します。

exports.delInflectionListener = functions

.region('asia-northeast1')

.firestore

.document('corona-infos/{locationcode}/users/{userid}')

.onDelete( async (snap, context) => {

const inflectionRef = admin.firestore().collection('corona-infos')

.doc(context.params.locationcode);

return inflectionRef.set({

density: admin.firestore.FieldValue.increment(-1)},

{merge: true});

});

## 感染報告された場合、位置履歴を感染エリアに更新します。

exports.userInflectionReportListener = functions

.region('asia-northeast1')

.firestore

.document('users/{userid}/infos/status')

.onUpdate(async (change, context) => {

const coronaInfosRef =

admin.firestore().collection('corona-infos').doc('info');

const coronaInfosInfos = await coronaInfosRef.get();

if (change.before.data().infection\_flag === 0) {

if (change.after.data().infection\_flag === 1) {

console.info("create inflection infomation",

"==>", context.params.userid);

const traceInfosRef = admin.firestore().collection('users')

.doc(context.params.userid)

.collection('trace-infos');

const traceInfos = await traceInfosRef.get();

//console.log("traceInfos", '=>', traceInfos);

let batch = admin.firestore().batch();

traceInfos.docs.forEach(doc => {

//console.log(doc.id, '=>', doc.data());

const inflectionRef = admin.firestore()

.collection('corona-infos')

.doc(doc.data().locationcode)

.collection('users')

.doc(context.params.userid);

batch.set(inflectionRef, {

timestamp: doc.data().timestamp,

userid: context.params.userid,

});

});

await(batch.commit());

}

}

else if (change.before.data().infection\_flag === 1) {

if (change.after.data().infection\_flag === 0) {

console.info("delete inflection infomation",

"==>", context.params.userid);

await deleteUserInflectInfos(

admin.firestore(), context.params.userid, 10000);

}

}

const coronaInfoRef =

admin.firestore().collection('corona-infos').doc("info");

return coronaInfoRef.set(

{timestamp: admin.firestore.FieldValue.serverTimestamp()},

{merge: true});

});

function deleteUserInflectInfos(db, userid, batchSize) {

let collectionRef = db.collectionGroup("users");

//let query = collectionRef.orderBy(admin.firestore.FieldPath.documentId())

// .limit(batchSize)

// .where(userid, '==', userid);

let query = collectionRef.orderBy(admin.firestore.FieldPath.documentId())

.limit(batchSize);

return new Promise((resolve, reject) => {

deleteInflectionUser(db, query, userid, batchSize, resolve, reject);

});

}

function deleteInflectionUser(db, query, userid, batchSize, resolve, reject) {

query.get()

.then((snapshot) => {

// When there are no documents left, we are done

if (snapshot.size == 0) {

return 0;

}

// Delete documents in a batch

let batch = db.batch();

snapshot.docs.forEach((doc) => {

if (doc.data().userid === userid) {

batch.delete(doc.ref);

}

});

return batch.commit().then(() => {

return snapshot.size;

});

}).then((numDeleted) => {

if (numDeleted === 0) {

resolve();

return;

}

// Recurse on the next process tick, to avoid

// exploding the stack.

process.nextTick(() => {

deleteInflectionUser(db, query, userid, batchSize, resolve, reject);

});

})

.catch(reject);

}

# 自分の感染危険度について

# セキュリティ

## 認証

## 承認

# 感染エリアの生成について

# 自分の感染危険度について