1. **Crea un Proyecto en Firebase Console**:
   * Ve a [Firebase Console](https://console.firebase.google.com/" \t "_blank).
   * Haz clic en "Agregar proyecto" y sigue las instrucciones.
2. **Agrega tu Aplicación Android**:
   * En la sección de "Configuración del proyecto", selecciona "Agregar aplicación" y elige Android.
   * Introduce el nombre del paquete de tu aplicación (debe coincidir con el que usaste al crear el proyecto).
3. **Descarga google-services.json**:
   * Después de registrar tu aplicación, descarga el archivo google-services.json.
   * Coloca este archivo en la carpeta app/ de tu proyecto.
4. **Configura Gradle**:
   * En build.gradle (Project), agrega el plugin de Google Services:
5. Para que los SDK de Firebase puedan acceder a los valores de configuración de google-services.json, necesitas el complemento Gradle de los servicios de Google.

DSL de Kotlin (build.gradle.kts)

Groovy (build.gradle)

Agrega el complemento como una dependencia a tu archivo build.gradle.kts **de nivel de proyecto**:

Archivo de Gradle de nivel de raíz (nivel de proyecto) (<project>/build.gradle.kts):

plugins {

  // ...

  // Add the dependency for the Google services Gradle plugin

  id("com.google.gms.google-services") version "4.4.2" apply false

content\_copy

}

1. Luego, en el archivo build.gradle.kts del **módulo (nivel de la app)**, agrega el complemento google-services y cualquier SDK de Firebase que quieras usar en tu app:

Archivo de Gradle del módulo (nivel de app) (<project>/<app-module>/build.gradle.kts):

plugins {

  id("com.android.application")

content\_copy

  // Add the Google services Gradle plugin

  id("com.google.gms.google-services")

content\_copy

  ...

}

dependencies {

  // Import the Firebase BoM

  implementation(platform("com.google.firebase:firebase-bom:33.11.0"))

content\_copy

  // TODO: Add the dependencies for Firebase products you want to use

  // When using the BoM, don't specify versions in Firebase dependencies

  implementation("com.google.firebase:firebase-analytics")

content\_copy

  // Add the dependencies for any other desired Firebase products

  // https://firebase.google.com/docs/android/setup#available-libraries

}

Si usas la BoM de Firebase para Android, tu app siempre utilizará versiones compatibles de la biblioteca de Firebase. [Más información](https://firebase.google.com/docs/android/learn-more?hl=es&authuser=0#bom)

Después de agregar el complemento y los SDK deseados, sincroniza tu proyecto de Android con archivos de Gradle.

**3. Agrega tu API Key de Google Maps**

Para agregar la API Key de Google Maps:

1. **Obtén la API Key**:
   * Ve a [Google Cloud Console](https://console.cloud.google.com/).
   * Crea un nuevo proyecto o selecciona uno existente.
   * Activa las APIs necesarias (Maps SDK for Android).
   * Crea credenciales para obtener una API Key.
2. **Agrega la API Key a tu Proyecto**:
   * En AndroidManifest.xml, agrega la siguiente línea dentro de la etiqueta <application>:

xml

<meta-data

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

android:value="TU\_API\_KEY\_AQUI"/>

**4. Implementa las Reglas de Seguridad Necesarias en Firestore**

Para proteger tus datos en Firestore, implementa reglas de seguridad:

1. Ve a la consola de Firebase y selecciona Firestore.
2. Haz clic en "Reglas" y establece las reglas adecuadas según tus necesidades.

Ejemplo básico para permitir solo a los usuarios autenticados acceder a su propia información:

javascript

rules\_version = '2';

service cloud.firestore {

match /databases/{database}/documents {

match /usuarios/{userId} {

allow read, write: **if** request.auth != **null** && request.auth.uid == userId;

}

match /eventos/{eventId} {

allow read: **if** true; *// Todos pueden leer eventos*

allow create: **if** request.auth != **null**; *// Solo usuarios autenticados pueden crear eventos*

}

match /mascotas\_perdidas/{mascotaId} {

allow read: **if** true; *// Todos pueden leer mascotas perdidas*

allow create: **if** request.auth != **null**; *// Solo usuarios autenticados pueden crear mascotas perdidas*

}

}

}

**5. Personaliza los Estilos y Componentes Según tus Necesidades**

Puedes personalizar los estilos y componentes utilizando temas y estilos en Jetpack Compose:

1. **Crea un Archivo de Tema**:
   * Crea un archivo llamado Theme.kt donde definas tus colores, tipografías y estilos personalizados.

kotlin

**private** **val** DarkColorPalette = darkColors(

primary = Color(0xFFBB86FC),

primaryVariant = Color(0xFF3700B3),

secondary = Color(0xFF03DAC6)

)

**private** **val** LightColorPalette = lightColors(

primary = Color(0xFF6200EE),

primaryVariant = Color(0xFF3700B3),

secondary = Color(0xFF03DAC6)

)

@Composable

**fun** PetCommunityTheme(darkTheme: Boolean = isSystemInDarkTheme(), content: @Composable () -> Unit) {

**val** colors = **if** (darkTheme) {

DarkColorPalette

} **else** {

LightColorPalette

}

MaterialTheme(

colors = colors,

typography = Typography,

shapes = Shapes,

content = content

)

}

1. **Usa los Estilos Personalizados**:
   * Aplica los estilos definidos en tus componentes utilizando MaterialTheme.typography y MaterialTheme.colors.

rules\_version = '2';

service cloud.firestore {

match /databases/{database}/documents {

*// Reglas para la colección de usuarios*

match /usuarios/{userId} {

allow read, write: **if** request.auth != **null** && request.auth.uid == userId;

}

*// Reglas para la colección de mascotas*

match /mascotas/{mascotaId} {

allow read: **if** request.auth != **null** && get(/databases/$(database)/documents/usuarios/$(request.auth.uid)).data.mascotas.hasAny([mascotaId]);

allow create: **if** request.auth != **null** && request.resource.data.usuarioId == request.auth.uid;

allow update, **delete**: **if** request.auth != **null** && get(/databases/$(database)/documents/mascotas/$(mascotaId)).data.usuarioId == request.auth.uid;

}

*// Reglas para la colección de eventos*

match /eventos/{eventoId} {

allow read: **if** true;

allow create: **if** request.auth != **null**;

allow update, **delete**: **if** request.auth != **null** && get(/databases/$(database)/documents/eventos/$(eventoId)).data.creadorId == request.auth.uid;

}

*// Reglas para la colección de mascotas\_perdidas*

match /mascotas\_perdidas/{mascotaId} {

allow read: **if** true;

allow create: **if** request.auth != **null** && request.resource.data.usuarioId == request.auth.uid;

allow update, **delete**: **if** request.auth != **null** && get(/databases/$(database)/documents/mascotas\_perdidas/$(mascotaId)).data.usuarioId == request.auth.uid;

}

}

}

Estas reglas establecen lo siguiente:

* Los usuarios solo pueden leer y escribir sus propios datos en la colección usuarios.
* Solo el dueño de una mascota puede leer, actualizar o eliminar la información de su mascota.
* Cualquiera puede leer eventos, pero solo usuarios autenticados pueden crearlos. Solo el creador puede modificar o eliminar un evento.
* Cualquiera puede ver mascotas perdidas, pero solo usuarios autenticados pueden reportarlas. Solo el usuario que reportó puede actualizar o eliminar el reporte.

Siguiendo estas pautas, deberías tener una aplicación bien configurada y segura.

quí te explico **paso a paso** cómo mover tu API Key de Google Maps del AndroidManifest.xml a un lugar más seguro (local.properties), usando el sistema de **BuildConfig**:

**Paso 1: Agregar la clave a local.properties**

1. Abre el archivo local.properties (en la raíz de tu proyecto Android)
2. Agrega tu API Key (asegúrate de que no tenga comillas):

properties

Copy

MAPS\_API\_KEY=AIzaSyCvm\_psUmNFxWI1Qrepe6q1F9IawFMQ0Ak

**Paso 2: Configurar el archivo build.gradle (Module:app)**

1. Abre build.gradle.kts (del módulo app)
2. Agrega esto en el bloque android (antes de buildTypes):

kotlin

Copy

android {

// ... (configuraciones existentes)

defaultConfig {

// ... (otras configs)

// Crea una variable BuildConfig con la API Key

buildConfigField("String", "MAPS\_API\_KEY", "\"${properties["MAPS\_API\_KEY"]}\"")

}

}

1. **Para debug/QA seguro** (opcional pero recomendado):

kotlin

Copy

buildTypes {

debug {

// Oculta la key en logs

resValue("string", "maps\_api\_key", "${properties["MAPS\_API\_KEY"]}")

}

release {

// Usa la misma key o una diferente para producción

resValue("string", "maps\_api\_key", "${properties["MAPS\_API\_KEY"]}")

}

}

**Paso 3: Actualizar AndroidManifest.xml**

Reemplaza el meta-data con la referencia a la variable:

xml

Copy

<meta-data

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

android:value="@string/maps\_api\_key" />

Run HTML

**Paso 4: Verificar que local.properties esté en .gitignore**

Asegúrate de que tu archivo .gitignore (en la raíz del proyecto) incluya:

Copy

/local.properties

**Paso 5: Sincronizar y limpiar el proyecto**

1. Haz clic en **"Sync Project with Gradle Files"** (elefante + flecha en la barra superior).
2. Ejecuta **Build > Clean Project** y luego **Build > Rebuild Project**.

**¿Cómo usar la key en código Kotlin?**

Si necesitas acceder a la key programáticamente:

kotlin

Copy

val apiKey = BuildConfig.MAPS\_API\_KEY

**PARA ENVIAR MAILS:**

**🚀 PASOS PARA USAR EMAILJS DESDE ANDROID**

**✅ 1. Crear cuenta en EmailJS**

1. **Ve a** [**https://www.emailjs.com**](https://www.emailjs.com)**.**
2. **Crea una cuenta gratuita (incluye 200 correos/mes).**
3. **Verifica tu dirección de correo electrónico.**

**✅ 2. Conectar tu servicio de email**

1. **En el dashboard de EmailJS, ve a Email Services.**
2. **Haz clic en “Add new service”.**
3. **Elige tu proveedor de email (ej. Gmail).**
4. **Autoriza tu cuenta.**
5. **Te dará un Service ID (guárdalo, lo necesitarás).**

**✅ 3. Crear una plantilla de email**

1. **En el panel izquierdo, ve a Email Templates.**
2. **Crea una nueva plantilla.**
3. **Puedes usar variables como:**

**html**

**CopiarEditar**

**<p>Hola, ${user\_name}. ¡Bienvenido a nuestra app!</p>**

1. **Guarda la plantilla. Te dará un Template ID.**

**✅ 4. Obtener tu User ID o Public Key**

1. **Ve a Account > API Keys.**
2. **Copia tu Public Key (user ID).**

**✅ 5. Usar EmailJS en Android**

**✅ Opción B: Usar Volley (alternativa simple)**

**🔧 Paso 1: Agrega Volley al build.gradle**

**gradle**

**CopiarEditar**

**implementation 'com.android.volley:volley:1.2.1'**

**🔧 Paso 2: Envía el correo con JsonObjectRequest**

**kotlin**

**CopiarEditar**

**val url = "https://api.emailjs.com/api/v1.0/email/send"**

**val queue = Volley.newRequestQueue(this)**

**val params = JSONObject().apply {**

**put("service\_id", "YOUR\_SERVICE\_ID")**

**put("template\_id", "YOUR\_TEMPLATE\_ID")**

**put("user\_id", "YOUR\_PUBLIC\_KEY")**

**put("template\_params", JSONObject().apply {**

**put("user\_name", "Juan Pérez")**

**put("user\_email", "juan@ejemplo.com")**

**})**

**}**

**val request = object : JsonObjectRequest(**

**Method.POST, url, params,**

**{ response ->**

**Log.d("EmailJS", "Correo enviado: $response")**

**},**

**{ error ->**

**Log.e("EmailJS", "Error: ${error.message}")**

**}**

**) {**

**override fun getHeaders(): MutableMap<String, String> {**

**return mutableMapOf("Content-Type" to "application/json")**

**}**

**}**

**queue.add(request)**