Contenido

[MÉTODO PARA EDITAR REGISTROS MERCADOS 38](#_Toc188784996)

[MÉTODO PARA BUSCAR Y EDITAR MERCADOS REGISTRADOS 41](#_Toc188784997)

**INICIO SESION**

Claro, voy a explicarte el código línea por línea. Este código es parte de una actividad en una aplicación Android que maneja el inicio de sesión de un usuario utilizando Firebase Authentication.

**1. Declaración de variables**

EditText correo, pass;

FirebaseAuth auth;

* **correo y pass**: Son variables para referirse a los campos de texto donde el usuario va a ingresar su correo y su contraseña.
* **auth**: Es una instancia de FirebaseAuth, la cual se usa para autenticar a los usuarios con Firebase (en este caso, para iniciar sesión).

**2. Método onCreate**

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

EdgeToEdge.enable(this);

setContentView(R.layout.activity\_inicio\_sesion);

* **onCreate**: Este método es llamado cuando la actividad es creada. Es donde generalmente se configura la interfaz de usuario y las variables.
* **EdgeToEdge.enable(this)**: Esta línea permite que la interfaz de usuario ocupe todo el espacio de la pantalla, incluyendo las áreas de las "system bars" (barra de estado, barra de navegación).
* **setContentView(R.layout.activity\_inicio\_sesion)**: Se establece el layout de la actividad (activity\_inicio\_sesion), que define cómo se verá la pantalla.

**3. Ajustes de márgenes para las "system bars"**

ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {

Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());

v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);

return insets;

});

* Aquí, se ajustan los márgenes de la interfaz para asegurarse de que el contenido no se superponga con las barras del sistema (como la barra de estado o la barra de navegación). Esto es importante especialmente en dispositivos con pantallas más grandes o "notches" (muescas) en la pantalla.

**4. Inicialización de las vistas**

correo = findViewById(R.id.correoInicio);

pass = findViewById(R.id.passInicio);

auth = FirebaseAuth.getInstance();

* **correo = findViewById(R.id.correoInicio)**: Obtiene una referencia al campo de texto para el correo electrónico.
* **pass = findViewById(R.id.passInicio)**: Obtiene una referencia al campo de texto para la contraseña.
* **auth = FirebaseAuth.getInstance()**: Inicializa la instancia de FirebaseAuth que se utilizará para gestionar el inicio de sesión.

**5. Método iniciarSesion**

public void iniciarSesion(View view){

String correoUsuario = correo.getText().toString().trim();

String passUsuario = pass.getText().toString().trim();

if(correoUsuario.isEmpty() || passUsuario.isEmpty()){

Toast.makeText(this, "Ingrese los datos", Toast.LENGTH\_SHORT).show();

}else{

inicioUsuario(correoUsuario, passUsuario);

}

}

* **iniciarSesion**: Este es el método que se ejecuta cuando el usuario hace clic en el botón de inicio de sesión.
  + **correoUsuario y passUsuario**: Obtiene los valores que el usuario ha introducido en los campos de correo y contraseña.
  + **Comprobación de campos vacíos**: Si alguno de los campos está vacío, muestra un Toast notificando que debe ingresar los datos.
  + **Llamada a inicioUsuario**: Si ambos campos están completos, se llama al método inicioUsuario para proceder con la autenticación.

**6. Método inicioUsuario**

private void inicioUsuario(String correoUsuario, String passUsuario) {

auth.signInWithEmailAndPassword(correoUsuario, passUsuario).addOnCompleteListener(new OnCompleteListener<AuthResult>() {

@Override

public void onComplete(@NonNull Task<AuthResult> task) {

if(task.isSuccessful()){

finish();

startActivity(new Intent(InicioSesion.this, MainActivity.class));

Toast.makeText(InicioSesion.this, "Bienvenido/a", Toast.LENGTH\_SHORT).show();

}else{

Toast.makeText(InicioSesion.this, "Error", Toast.LENGTH\_SHORT).show();

}

}

}).addOnFailureListener(new OnFailureListener() {

@Override

public void onFailure(@NonNull Exception e) {

Toast.makeText(InicioSesion.this, "Error al iniciar sesion", Toast.LENGTH\_SHORT).show();

}

});

}

* **inicioUsuario**: Este método realiza el intento de inicio de sesión con los datos proporcionados (correo y contraseña).
  + **auth.signInWithEmailAndPassword(correoUsuario, passUsuario)**: Llama al método de Firebase para intentar autenticar al usuario con su correo y contraseña.
  + **addOnCompleteListener**: Se ejecuta cuando la operación de inicio de sesión ha terminado. Si la autenticación es exitosa:
    - **finish()**: Cierra la actividad de inicio de sesión.
    - **startActivity(new Intent(...))**: Inicia la actividad principal (MainActivity), redirigiendo al usuario a la pantalla principal de la app.
    - **Toast.makeText(...)**: Muestra un mensaje de bienvenida.
  + Si la autenticación falla, muestra un mensaje de error.
  + **addOnFailureListener**: En caso de que la operación de inicio de sesión falle por alguna razón, se muestra un mensaje de error adicional.

**7. Método registrarse**

public void registrarse(View view){

startActivity(new Intent(InicioSesion.this, Registro.class));

}

* **registrarse**: Este método se ejecuta cuando el usuario hace clic en el botón para registrarse. Redirige al usuario a la actividad de registro (Registro.class), donde podrá crear una cuenta si aún no tiene una.

**Resumen:**

Este código es para una pantalla de inicio de sesión en una aplicación Android que usa Firebase Authentication. Los pasos son:

1. El usuario ingresa su correo y contraseña.
2. Si los campos no están vacíos, se intenta iniciar sesión con Firebase.
3. Si el inicio de sesión es exitoso, se redirige al usuario a la pantalla principal de la aplicación (MainActivity).
4. Si el inicio de sesión falla, se muestra un mensaje de error.
5. Si el usuario no tiene cuenta, puede ir a la pantalla de registro.

Además, se realizan ajustes para asegurar que la interfaz no se solape con las barras del sistema en dispositivos con pantallas grandes o con notch.

**REGISTRO USUARIO**

Este código es para una actividad de registro en una aplicación Android que utiliza Firebase para registrar a un nuevo usuario. Te explicaré línea por línea qué hace cada parte del código.

**1. Declaración de variables**

EditText nombre, correo, pass;

FirebaseFirestore firestore;

FirebaseAuth auth;

* **nombre, correo, pass**: Son variables para almacenar las referencias a los campos de texto donde el usuario ingresará su nombre, correo electrónico y contraseña, respectivamente.
* **firestore**: Es una instancia de FirebaseFirestore, que se utilizará para almacenar los datos del usuario en la base de datos de Firebase.
* **auth**: Es una instancia de FirebaseAuth, que se usa para manejar la autenticación del usuario (en este caso, el registro).

**2. Método onCreate**

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

EdgeToEdge.enable(this);

setContentView(R.layout.activity\_registro);

* **onCreate**: Este método es el que se ejecuta cuando la actividad se crea.
* **EdgeToEdge.enable(this)**: Permite que la interfaz de usuario ocupe todo el espacio de la pantalla, incluyendo áreas como las barras de sistema (barra de estado, barra de navegación) si se está utilizando el diseño "edge-to-edge".
* **setContentView(R.layout.activity\_registro)**: Establece el archivo de diseño (XML) que se usará para la actividad, en este caso activity\_registro.

**3. Configuración de los márgenes para los "system bars"**

ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {

Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());

v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);

return insets;

});

* Aquí, se ajusta el diseño para que no se solapen con las barras del sistema (barra de estado y barra de navegación).
* ViewCompat.setOnApplyWindowInsetsListener: Se usa para escuchar los cambios en los márgenes de la ventana y ajustar el contenido de la interfaz de usuario.

**4. Inicialización de las vistas**

nombre = findViewById(R.id.nombre);

correo = findViewById(R.id.correo);

pass = findViewById(R.id.pass);

firestore = FirebaseFirestore.getInstance();

auth = FirebaseAuth.getInstance();

* Aquí se obtienen las referencias a los elementos de la interfaz (campo de texto para nombre, correo, y contraseña) utilizando findViewById. También se inicializan las instancias de FirebaseFirestore y FirebaseAuth.

**5. Método registrarme**

public void registrarme(View view) {

String nombreUsuario = nombre.getText().toString().trim();

String correoUsuario = correo.getText().toString().trim();

String passUsuario = pass.getText().toString().trim();

if (nombreUsuario.isEmpty() || correoUsuario.isEmpty() || passUsuario.isEmpty()) {

Toast.makeText(this, "Complete todos los campos", Toast.LENGTH\_SHORT).show();

} else {

registroUsuario(nombreUsuario, correoUsuario, passUsuario);

}

}

* **registrarme**: Este es el método que se ejecuta cuando el usuario hace clic en el botón de registro. Toma los valores de los campos de texto y los verifica:
  + Si alguno de los campos está vacío, muestra un mensaje (Toast) pidiendo que complete todos los campos.
  + Si todos los campos están completos, llama al método registroUsuario.

**6. Método registroUsuario**

private void registroUsuario(String nombreUsuario, String correoUsuario, String passUsuario) {

auth.createUserWithEmailAndPassword(correoUsuario, passUsuario).addOnCompleteListener(new OnCompleteListener<AuthResult>() {

@Override

public void onComplete(@NonNull Task<AuthResult> task) {

String id = auth.getCurrentUser().getUid();

Map<String, Object> map = new HashMap<>();

map.put("id", id);

map.put("nombre", nombreUsuario);

map.put("correo", correoUsuario);

map.put("pass", passUsuario);

firestore.collection("usuario").document(id).set(map).addOnSuccessListener(new OnSuccessListener<Void>() {

@Override

public void onSuccess(Void unused) {

finish();

startActivity(new Intent(Registro.this, MainActivity.class));

Toast.makeText(Registro.this, "Usuario registrado", Toast.LENGTH\_SHORT).show();

}

}).addOnFailureListener(new OnFailureListener() {

@Override

public void onFailure(@NonNull Exception e) {

Toast.makeText(Registro.this, "Error al guardar", Toast.LENGTH\_SHORT).show();

}

});

}

}).addOnFailureListener(new OnFailureListener() {

@Override

public void onFailure(@NonNull Exception e) {

Toast.makeText(Registro.this, "Error al registrar", Toast.LENGTH\_SHORT).show();

}

});

}

* **registroUsuario**: Este método es el que se encarga de registrar al usuario en Firebase.
  + Primero, crea una cuenta de usuario con el correo y la contraseña proporcionados mediante auth.createUserWithEmailAndPassword().
  + Si la creación del usuario es exitosa:
    - Obtiene el ID del usuario (auth.getCurrentUser().getUid()).
    - Crea un Map con los datos del usuario (ID, nombre, correo, y contraseña).
    - Guarda esos datos en una colección de Firestore llamada usuario utilizando firestore.collection("usuario").document(id).set(map).
    - Si la escritura en Firestore es exitosa, muestra un mensaje de éxito y redirige al usuario a la pantalla principal (MainActivity).
    - Si ocurre un error en cualquiera de las operaciones, se muestra un mensaje de error usando Toast.

**Resumen**

Este código permite que un usuario se registre en la aplicación mediante su correo electrónico y una contraseña. Los datos del usuario se almacenan en Firebase Authentication para la autenticación y en Firebase Firestore para la base de datos. Además, maneja correctamente los errores y muestra mensajes de retroalimentación al usuario.

¡Claro! Para crear un método listarUsuarios que liste todos los usuarios registrados en Firestore, primero necesitamos asegurarnos de que tienes una colección en Firestore donde se almacenan los usuarios. En el código que proporcionaste, los usuarios se almacenan en la colección "usuario".

Aquí te dejo un ejemplo de cómo podrías implementar el método listarUsuarios en la clase Registro, utilizando Firestore para obtener todos los documentos de la colección usuario y luego mostrarlos, por ejemplo, en un Toast o en un Log para propósitos de depuración:

**Código para el método listarUsuarios:**

private void listarUsuarios() {

firestore.collection("usuario") // Accedemos a la colección "usuario"

.get() // Obtiene todos los documentos de la colección

.addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {

@Override

public void onComplete(@NonNull Task<QuerySnapshot> task) {

if (task.isSuccessful()) {

QuerySnapshot querySnapshot = task.getResult(); // Obtenemos el resultado

StringBuilder usuariosList = new StringBuilder();

// Recorremos todos los documentos (usuarios) en la colección

for (DocumentSnapshot document : querySnapshot.getDocuments()) {

// Extraemos los datos de cada usuario (puedes agregar más campos si es necesario)

String id = document.getString("id");

String nombre = document.getString("nombre");

String correo = document.getString("correo");

// Añadimos la información de cada usuario a un StringBuilder

usuariosList.append("ID: ").append(id)

.append(", Nombre: ").append(nombre)

.append(", Correo: ").append(correo)

.append("\n");

}

// Mostramos los usuarios listados en un Toast o en Log (para depuración)

if (usuariosList.length() > 0) {

Toast.makeText(Registro.this, usuariosList.toString(), Toast.LENGTH\_LONG).show();

} else {

Toast.makeText(Registro.this, "No hay usuarios registrados.", Toast.LENGTH\_SHORT).show();

}

} else {

Toast.makeText(Registro.this, "Error al obtener los usuarios.", Toast.LENGTH\_SHORT).show();

}

}

});

}

**Explicación del código:**

1. **Acceso a la colección "usuario"**:
   * Usamos firestore.collection("usuario") para acceder a la colección de usuarios en Firestore.
2. **Obtención de los documentos**:
   * Usamos el método .get() para obtener todos los documentos de la colección. Esto devuelve un QuerySnapshot, que contiene todos los documentos encontrados.
3. **Procesamiento de los documentos**:
   * Si la operación es exitosa (task.isSuccessful()), recorremos los documentos obtenidos con querySnapshot.getDocuments().
   * Para cada documento, usamos document.getString("campo") para obtener los valores de los campos almacenados en Firestore (en este caso, id, nombre y correo).
4. **Mostrar los usuarios**:
   * La información de todos los usuarios se concatena en un StringBuilder, que luego se muestra en un Toast. Si no se encuentran usuarios, se muestra un mensaje indicando que no hay usuarios registrados.
5. **Manejo de errores**:
   * Si la operación falla, mostramos un mensaje de error.

**¿Cómo llamar al método listarUsuarios?**

Puedes llamar a este método en cualquier parte de la actividad, por ejemplo, después de un botón de "Ver usuarios", o llamarlo directamente desde el método onCreate para que se ejecute cuando la actividad se cargue.

Ejemplo de cómo llamar a listarUsuarios:

// Llamada al método desde algún botón, por ejemplo:

Button btnVerUsuarios = findViewById(R.id.btnVerUsuarios);

btnVerUsuarios.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

listarUsuarios();

}

});

**Consideraciones:**

* Este ejemplo muestra la lista de usuarios en un Toast. Si necesitas mostrar los usuarios en una interfaz de usuario más avanzada (por ejemplo, en una lista), podrías usar un RecyclerView para mostrar los datos de manera más estructurada.
* Asegúrate de manejar correctamente los permisos de acceso a Firestore si estás usando reglas de seguridad.

Este es un enfoque básico para listar los usuarios registrados en Firestore. ¡Puedes modificarlo según lo que necesites!

MAIN

¡Claro! Aquí tienes una explicación línea por línea de este código, que corresponde a una actividad principal (MainActivity) en una aplicación Android que maneja datos de mercados y permite iniciar sesión, cerrar sesión e insertar información en Firestore.

**1. Declaración de variables**

FirebaseAuth auth;

EditText id, nombre, ubicacion, inicio, fin;

FirebaseFirestore firestore;

* **auth**: Instancia de FirebaseAuth utilizada para gestionar la autenticación de usuarios (en este caso, para cerrar sesión).
* **id, nombre, ubicacion, inicio, fin**: Son referencias a campos de texto (EditText) donde el usuario ingresará la información del mercado.
* **firestore**: Instancia de FirebaseFirestore, que se utiliza para interactuar con la base de datos de Firestore.

**2. Método onCreate**

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

EdgeToEdge.enable(this);

setContentView(R.layout.activity\_main);

ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {

Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());

v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);

return insets;

});

* **onCreate**: Este es el método principal que se ejecuta cuando la actividad es creada. Aquí se inicializan las vistas y se configuran elementos de la interfaz.
* **EdgeToEdge.enable(this)**: Permite que el contenido de la aplicación se extienda por toda la pantalla, incluso debajo de las "system bars" (barra de estado y barra de navegación).
* **setContentView(R.layout.activity\_main)**: Establece el layout de la actividad, que define el diseño visual de la pantalla de la actividad principal.
* **ViewCompat.setOnApplyWindowInsetsListener(...)**: Ajusta el padding (relleno) del contenido para evitar que se solape con las barras del sistema (como la barra de estado o la barra de navegación).

**3. Inicialización de las vistas**

auth = FirebaseAuth.getInstance();

firestore = FirebaseFirestore.getInstance();

id = findViewById(R.id.idMercado);

nombre = findViewById(R.id.nombreMercado);

ubicacion = findViewById(R.id.ubiMercado);

inicio = findViewById(R.id.inicioMercado);

fin = findViewById(R.id.finMercado);

* **auth = FirebaseAuth.getInstance()**: Inicializa la instancia de FirebaseAuth, lo que permite manejar la autenticación del usuario.
* **firestore = FirebaseFirestore.getInstance()**: Inicializa la instancia de FirebaseFirestore, lo que permite interactuar con la base de datos de Firestore.
* **Inicialización de las vistas**: Cada campo de texto (EditText) se inicializa con findViewById, para asociarlo con su respectivo componente en el XML.

**4. Método cerrarSesion**

public void cerrarSesion(View view) {

auth.signOut();

finish();

startActivity(new Intent(this, InicioSesion.class));

}

* **cerrarSesion**: Este método se ejecuta cuando el usuario quiere cerrar sesión.
  + **auth.signOut()**: Cierra la sesión del usuario actual utilizando Firebase Authentication.
  + **finish()**: Finaliza la actividad actual (en este caso, MainActivity).
  + **startActivity(new Intent(this, InicioSesion.class))**: Redirige al usuario a la actividad de inicio de sesión (InicioSesion), lo que permite que el usuario inicie sesión nuevamente si lo desea.

**5. Método insertarMercado**

public void insertarMercado(View view) {

String idMercado = id.getText().toString().trim();

String nombreMercado = nombre.getText().toString().trim();

String ubicacionMercado = ubicacion.getText().toString().trim();

String inicioMercado = inicio.getText().toString().trim();

String finMercado = fin.getText().toString().trim();

agregarMercado(idMercado, nombreMercado, ubicacionMercado, inicioMercado, finMercado);

}

* **insertarMercado**: Este método se ejecuta cuando el usuario hace clic en el botón para insertar la información del mercado en Firestore.
  + **Obtención de datos**: Los valores de los campos EditText se obtienen con getText().toString().trim(), lo que asegura que se eliminen los espacios en blanco al inicio y al final.
  + **Llamada a agregarMercado**: Después de obtener los datos, se pasa esa información al método agregarMercado para agregarla a Firestore.

**6. Método agregarMercado**

private void agregarMercado(String idMercado, String nombreMercado, String ubicacionMercado, String inicioMercado, String finMercado) {

Map<String, Object> map = new HashMap<>();

map.put("id", idMercado);

map.put("nombre", nombreMercado);

map.put("ubicacion", ubicacionMercado);

map.put("inicio", inicioMercado);

map.put("fin", finMercado);

firestore.collection("mercado").add(map).addOnCompleteListener(new OnCompleteListener<DocumentReference>() {

@Override

public void onComplete(@NonNull Task<DocumentReference> task) {

Toast.makeText(MainActivity.this, "Mercado Registrado", Toast.LENGTH\_SHORT).show();

}

}).addOnFailureListener(new OnFailureListener() {

@Override

public void onFailure(@NonNull Exception e) {

Toast.makeText(MainActivity.this, "Error al insertar", Toast.LENGTH\_SHORT).show();

}

});

}

* **agregarMercado**: Este método agrega la información del mercado a Firestore.
  + **Map<String, Object> map = new HashMap<>()**: Crea un mapa que contiene los datos del mercado, como el id, nombre, ubicacion, inicio, y fin.
  + **firestore.collection("mercado").add(map)**: Agrega los datos del mercado en la colección "mercado" en Firestore. Este es un proceso asincrónico.
  + **addOnCompleteListener**: Se ejecuta cuando la operación de agregar el mercado se completa. Si la inserción es exitosa, muestra un Toast que dice "Mercado Registrado".
  + **addOnFailureListener**: Si la operación falla por alguna razón, muestra un Toast con el mensaje "Error al insertar".

**Resumen general del flujo de la aplicación:**

1. **Pantalla Principal (MainActivity)**:
   * El usuario puede insertar datos sobre un mercado (como id, nombre, ubicación, horarios de apertura y cierre).
   * Al hacer clic en el botón de "Insertar Mercado", los datos se recogen de los campos de texto y se almacenan en Firestore.
   * También puede cerrar sesión, lo que terminará su sesión actual y lo llevará a la pantalla de inicio de sesión.
2. **Firebase Authentication y Firestore**:
   * **Autenticación**: Se utiliza para gestionar la sesión del usuario.
   * **Firestore**: Se utiliza para almacenar los datos de los mercados.

Este código gestiona la sesión de un usuario, permite almacenar información sobre mercados en Firestore y manejar algunas interacciones básicas con la interfaz de usuario.

OBTENER UN REGISTRO

Para obtener un registro de Firebase en una aplicación Android escrita en Java, necesitas interactuar con Firebase Firestore, que es la base de datos en tiempo real de Firebase. A continuación te explicaré los pasos para obtener un documento (registro) de una colección en Firestore.

**Pasos para obtener un registro de Firestore en Android con Java**

**1. Agregar dependencias en build.gradle (nivel de proyecto y de módulo)**

Primero, asegúrate de tener configurada tu aplicación para usar Firebase. En el archivo build.gradle (nivel de módulo) de tu aplicación, agrega las dependencias necesarias para Firestore:

dependencies {

// Firebase Authentication (si lo usas)

implementation 'com.google.firebase:firebase-auth:21.0.1'

// Firebase Firestore

implementation 'com.google.firebase:firebase-firestore:24.0.0'

// Firebase BOM (es recomendable usar el BOM de Firebase para asegurar que todas las bibliotecas sean compatibles entre sí)

implementation platform('com.google.firebase:firebase-bom:31.0.0')

}

Recuerda sincronizar el proyecto después de agregar las dependencias.

**2. Configurar Firebase en tu proyecto**

Asegúrate de que has seguido los pasos para agregar Firebase a tu proyecto. Si no lo has hecho aún, sigue estos pasos:

1. Ve a la [Consola de Firebase](https://console.firebase.google.com/).
2. Crea un proyecto de Firebase (si aún no tienes uno).
3. Agrega tu aplicación Android en la consola de Firebase, descargando el archivo google-services.json y colocándolo en el directorio app/ de tu proyecto Android.
4. Activa Firestore en la consola de Firebase.

**3. Inicializar Firestore en tu actividad**

En el código de tu actividad o fragmento, necesitas inicializar Firebase y Firestore:

import com.google.firebase.firestore.FirebaseFirestore;

import com.google.firebase.firestore.DocumentSnapshot;

import com.google.firebase.firestore.QueryDocumentSnapshot;

import com.google.firebase.firestore.QuerySnapshot;

public class MainActivity extends AppCompatActivity {

// Instancia de Firestore

FirebaseFirestore firestore;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

// Inicializar Firestore

firestore = FirebaseFirestore.getInstance();

// Llamar a la función que obtendrá un registro

obtenerRegistro();

}

private void obtenerRegistro() {

// Referencia a la colección "usuarios"

firestore.collection("usuarios").document("id\_del\_usuario") // Asegúrate de usar el ID correcto del documento

.get() // Obtiene un documento

.addOnSuccessListener(new OnSuccessListener<DocumentSnapshot>() {

@Override

public void onSuccess(DocumentSnapshot documentSnapshot) {

if (documentSnapshot.exists()) {

// Si el documento existe, extraemos los datos

String nombre = documentSnapshot.getString("nombre");

String correo = documentSnapshot.getString("correo");

// Puedes hacer lo que necesites con estos datos, como mostrarlos en la interfaz

Toast.makeText(MainActivity.this, "Nombre: " + nombre + ", Correo: " + correo, Toast.LENGTH\_SHORT).show();

} else {

Toast.makeText(MainActivity.this, "No existe el documento", Toast.LENGTH\_SHORT).show();

}

}

})

.addOnFailureListener(new OnFailureListener() {

@Override

public void onFailure(@NonNull Exception e) {

// En caso de error

Toast.makeText(MainActivity.this, "Error al obtener el registro: " + e.getMessage(), Toast.LENGTH\_SHORT).show();

}

});

}

}

**Explicación del código**

1. **Inicialización de Firestore:**
2. firestore = FirebaseFirestore.getInstance();

Esta línea inicializa la instancia de Firestore. Es la manera de conectarse con la base de datos.

1. **Referencia a un documento específico:**
2. firestore.collection("usuarios").document("id\_del\_usuario")

Aquí se hace referencia a la colección "usuarios" y se obtiene un documento específico utilizando el método .document("id\_del\_usuario"). El ID del documento es único para cada registro y debe coincidir con el ID de un documento en tu colección.

1. **Obtener el documento con .get():**
2. .get()

El método .get() obtiene un solo documento de Firestore. Si el documento existe, puedes acceder a sus campos.

1. **Manejo de éxito con addOnSuccessListener:**
2. .addOnSuccessListener(new OnSuccessListener<DocumentSnapshot>() {
3. @Override
4. public void onSuccess(DocumentSnapshot documentSnapshot) {
5. if (documentSnapshot.exists()) {
6. // Extraemos los campos del documento
7. String nombre = documentSnapshot.getString("nombre");
8. String correo = documentSnapshot.getString("correo");
9. // Mostrar los datos
10. Toast.makeText(MainActivity.this, "Nombre: " + nombre + ", Correo: " + correo, Toast.LENGTH\_SHORT).show();
11. }
12. }
13. })

Si la operación de obtener el documento es exitosa, se ejecuta el código dentro de onSuccess. Aquí, se verifica si el documento existe con .exists(). Luego se obtienen los valores de los campos ("nombre", "correo") con getString().

1. **Manejo de error con addOnFailureListener:**
2. .addOnFailureListener(new OnFailureListener() {
3. @Override
4. public void onFailure(@NonNull Exception e) {
5. Toast.makeText(MainActivity.this, "Error al obtener el registro: " + e.getMessage(), Toast.LENGTH\_SHORT).show();
6. }
7. });

Si algo sale mal (por ejemplo, si no hay conexión a la red o el documento no se encuentra), se maneja el error en el bloque onFailure.

**Conclusión**

Este código muestra cómo obtener un registro de Firestore en una aplicación Android usando Java. Al hacer una consulta a Firestore, puedes obtener un documento específico de una colección y luego manejar los datos que contiene. Es importante manejar los casos en que el documento no existe o ocurre un error durante la consulta.

Si necesitas obtener varios registros, puedes usar get() en una colección completa o usar una consulta (Query) con condiciones específicas. Si tienes alguna otra pregunta o necesitas ejemplos adicionales, no dudes en preguntar.

VER REGISTROS

Para obtener todos los registros de una colección en Firebase Firestore y mostrarlos en un ListView en una aplicación Android usando Java, debes usar una consulta (Query) para recuperar todos los documentos de la colección y luego adaptar los datos obtenidos para que se muestren en el ListView.

El proceso generalmente implica los siguientes pasos:

1. **Obtener todos los documentos de la colección.**
2. **Almacenar los datos obtenidos en una lista de objetos.**
3. **Adaptar los datos a un ListView usando un ArrayAdapter o un CustomAdapter.**

**Pasos para obtener todos los registros y mostrarlos en un ListView:**

**1. Preparar la interfaz de usuario (XML)**

Primero, debes tener un ListView en el layout XML de tu actividad. Aquí hay un ejemplo de cómo hacerlo:

<ListView

android:id="@+id/listView"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"/>

**2. Crear un modelo de datos (opcional pero recomendado)**

Es recomendable crear un modelo de datos (una clase Java) que represente la estructura de los documentos que quieres mostrar. Por ejemplo, si tienes una colección de mercados, podrías tener una clase Mercado:

public class Mercado {

private String id;

private String nombre;

private String ubicacion;

private String inicio;

private String fin;

public Mercado(String id, String nombre, String ubicacion, String inicio, String fin) {

this.id = id;

this.nombre = nombre;

this.ubicacion = ubicacion;

this.inicio = inicio;

this.fin = fin;

}

// Getters y setters

public String getId() {

return id;

}

public String getNombre() {

return nombre;

}

public String getUbicacion() {

return ubicacion;

}

public String getInicio() {

return inicio;

}

public String getFin() {

return fin;

}

}

**3. Recuperar los datos de Firestore**

Ahora, en tu actividad, puedes usar la instancia de FirebaseFirestore para obtener todos los documentos de una colección y luego agregar esos documentos a una lista. Posteriormente, usas un ArrayAdapter o CustomAdapter para mostrar los datos en el ListView.

Aquí tienes el código que muestra cómo hacerlo:

import android.os.Bundle;

import android.view.View;

import android.widget.ArrayAdapter;

import android.widget.ListView;

import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

import com.google.android.gms.tasks.OnCompleteListener;

import com.google.android.gms.tasks.Task;

import com.google.firebase.firestore.FirebaseFirestore;

import com.google.firebase.firestore.QueryDocumentSnapshot;

import com.google.firebase.firestore.QuerySnapshot;

import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {

private ListView listView;

private ArrayList<Mercado> mercadosList;

private ArrayAdapter<Mercado> adapter;

private FirebaseFirestore firestore;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

// Inicializar el ListView y Firestore

listView = findViewById(R.id.listView);

firestore = FirebaseFirestore.getInstance();

// Crear lista vacía para almacenar los mercados

mercadosList = new ArrayList<>();

// Crear un adaptador

adapter = new ArrayAdapter<>(this, android.R.layout.simple\_list\_item\_1, mercadosList);

listView.setAdapter(adapter);

// Obtener los registros de Firestore

obtenerRegistros();

}

private void obtenerRegistros() {

firestore.collection("mercado")

.get() // Obtiene todos los documentos en la colección "mercado"

.addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {

@Override

public void onComplete(Task<QuerySnapshot> task) {

if (task.isSuccessful()) {

for (QueryDocumentSnapshot document : task.getResult()) {

// Obtener los campos de cada documento

String id = document.getString("id");

String nombre = document.getString("nombre");

String ubicacion = document.getString("ubicacion");

String inicio = document.getString("inicio");

String fin = document.getString("fin");

// Crear un objeto Mercado y añadirlo a la lista

Mercado mercado = new Mercado(id, nombre, ubicacion, inicio, fin);

mercadosList.add(mercado);

}

// Notificar al adaptador para que actualice el ListView

adapter.notifyDataSetChanged();

} else {

Toast.makeText(MainActivity.this, "Error al obtener los registros.", Toast.LENGTH\_SHORT).show();

}

}

});

}

}

**Explicación del código:**

1. **Inicialización de variables:**
2. listView = findViewById(R.id.listView);
3. firestore = FirebaseFirestore.getInstance();
4. mercadosList = new ArrayList<>();
5. adapter = new ArrayAdapter<>(this, android.R.layout.simple\_list\_item\_1, mercadosList);
6. listView.setAdapter(adapter);
   * **listView**: Es la referencia al ListView que definiste en el XML.
   * **firestore**: Inicializa Firebase Firestore.
   * **mercadosList**: Es una lista que almacenará todos los objetos de tipo Mercado.
   * **adapter**: Es un ArrayAdapter que se encargará de adaptar los datos (en este caso, los objetos Mercado) al ListView. Aquí se usa el layout android.R.layout.simple\_list\_item\_1, que es un layout predeterminado para mostrar texto en cada fila.
7. **Método obtenerRegistros:**
8. firestore.collection("mercado")
9. .get()
10. .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
11. @Override
12. public void onComplete(Task<QuerySnapshot> task) {
13. if (task.isSuccessful()) {
14. for (QueryDocumentSnapshot document : task.getResult()) {
15. // Obtener datos de cada documento
16. String id = document.getString("id");
17. String nombre = document.getString("nombre");
18. String ubicacion = document.getString("ubicacion");
19. String inicio = document.getString("inicio");
20. String fin = document.getString("fin");
21. // Crear objeto Mercado y añadirlo a la lista
22. Mercado mercado = new Mercado(id, nombre, ubicacion, inicio, fin);
23. mercadosList.add(mercado);
24. }
25. adapter.notifyDataSetChanged();
26. } else {
27. Toast.makeText(MainActivity.this, "Error al obtener los registros.", Toast.LENGTH\_SHORT).show();
28. }
29. }
30. });
    * Este código obtiene todos los documentos de la colección mercado con get(). Cuando la consulta es exitosa, se itera sobre los documentos recibidos (QueryDocumentSnapshot).
    * Para cada documento, se extraen los campos (id, nombre, ubicacion, etc.) y se crea un objeto Mercado con esos valores.
    * El objeto Mercado se agrega a la lista mercadosList.
    * Finalmente, se notifica al adaptador (adapter.notifyDataSetChanged()) para que actualice el ListView con los nuevos datos.
31. **Mostrar los datos en el ListView:** El ArrayAdapter se encarga de convertir los objetos Mercado en una representación visual para el ListView. Por defecto, si usas un ArrayAdapter con simple\_list\_item\_1, solo se mostrará el método toString() del objeto Mercado. Si deseas personalizar la forma en que se muestran los elementos, puedes crear un **CustomAdapter**.

**Personalización (opcional):**

Si quieres personalizar la apariencia de cada fila en el ListView (por ejemplo, mostrar el nombre del mercado en negrita y la ubicación en otro color), puedes crear un **CustomAdapter** que extienda BaseAdapter o ArrayAdapter.

**Conclusión:**

Este enfoque permite recuperar todos los registros de una colección en Firebase Firestore y mostrarlos en un ListView en Android. Puedes personalizar tanto el modelo de datos como la forma de mostrar los datos, dependiendo de tus necesidades.

**LISTAR USUARIOS MÉTODO**

Si prefieres obtener todos los registros de Firestore y mostrarlos en un ListView utilizando un **método**, sin la necesidad de crear una clase adaptadora personalizada, puedes hacerlo de la siguiente manera.

**Pasos:**

1. **Recuperar los datos de Firestore usando un método.**
2. **Almacenar los datos obtenidos en una lista.**
3. **Usar un ArrayAdapter para adaptar esos datos al ListView.**

**Código:**

import android.os.Bundle;

import android.view.View;

import android.widget.ArrayAdapter;

import android.widget.ListView;

import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

import com.google.android.gms.tasks.OnCompleteListener;

import com.google.android.gms.tasks.Task;

import com.google.firebase.firestore.FirebaseFirestore;

import com.google.firebase.firestore.QueryDocumentSnapshot;

import com.google.firebase.firestore.QuerySnapshot;

import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {

private ListView listView;

private ArrayList<String> mercadosList; // Lista para almacenar los datos como cadenas de texto

private ArrayAdapter<String> adapter;

private FirebaseFirestore firestore;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

// Inicializar el ListView y Firestore

listView = findViewById(R.id.listView);

firestore = FirebaseFirestore.getInstance();

// Crear lista vacía para almacenar los mercados como cadenas de texto

mercadosList = new ArrayList<>();

// Crear un adaptador para el ListView

adapter = new ArrayAdapter<>(this, android.R.layout.simple\_list\_item\_1, mercadosList);

listView.setAdapter(adapter);

// Llamar al método para obtener los registros

obtenerRegistros();

}

// Método para obtener los registros de Firestore y mostrarlos en el ListView

private void obtenerRegistros() {

firestore.collection("mercado")

.get() // Obtener todos los documentos de la colección "mercado"

.addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {

@Override

public void onComplete(Task<QuerySnapshot> task) {

if (task.isSuccessful()) {

for (QueryDocumentSnapshot document : task.getResult()) {

// Obtener los datos de cada documento

String id = document.getString("id");

String nombre = document.getString("nombre");

String ubicacion = document.getString("ubicacion");

String inicio = document.getString("inicio");

String fin = document.getString("fin");

// Crear una cadena que contenga los datos del mercado

String mercadoInfo = "ID: " + id + "\n" +

"Nombre: " + nombre + "\n" +

"Ubicación: " + ubicacion + "\n" +

"Horario: " + inicio + " - " + fin;

// Añadir la información a la lista

mercadosList.add(mercadoInfo);

}

// Notificar al adaptador que los datos han cambiado

adapter.notifyDataSetChanged();

} else {

Toast.makeText(MainActivity.this, "Error al obtener los registros.", Toast.LENGTH\_SHORT).show();

}

}

});

}

}

**Explicación:**

1. **Recuperación de datos:**
   * Usamos firestore.collection("mercado").get() para obtener todos los documentos de la colección mercado.
   * En el método onComplete(), recorremos los documentos y extraemos los campos de cada uno.
2. **Almacenamiento en una lista:**
   * En lugar de crear un modelo de datos, directamente construimos una cadena de texto (String) con la información del mercado y la añadimos a una lista de cadenas (mercadosList).
3. **Adaptador para el ListView:**
   * Usamos un ArrayAdapter para adaptar la lista de cadenas al ListView. El ArrayAdapter se actualiza con los nuevos datos llamando a adapter.notifyDataSetChanged().
4. **Mostrar la información:**
   * La cadena que añadimos al ArrayList tiene el formato:
   * ID: <id>
   * Nombre: <nombre>
   * Ubicación: <ubicacion>
   * Horario: <inicio> - <fin>
   * Esto se mostrará en cada fila del ListView.

**¿Por qué no necesitas una clase personalizada?**

Este código no usa una clase de modelo personalizada como Mercado. En su lugar, directamente se extraen los datos de Firestore, se combinan en una cadena y se muestran en el ListView. Esto es más simple, pero tiene la limitación de que no puedes hacer fácilmente personalizaciones complejas en la visualización de los elementos, como agregar imágenes o mostrar los datos en un formato más elaborado.

**Conclusión:**

Usar un método para obtener los registros y mostrarlos en un ListView es una opción sencilla y directa, pero solo es adecuada cuando los datos que quieres mostrar son simples y no necesitas personalizar demasiado la apariencia del ListView. Si más adelante necesitas una interfaz más compleja, te recomendaría usar un adaptador personalizado.

**CORREGIDO CODIGO BUSCAR USUARIO**

El problema principal está en esta línea del código:

firestore.collection("usuario").document("nombre").get()

Estás intentando acceder al documento con la ID "nombre" en la colección usuario. Sin embargo, en tu base de datos, no parece haber un documento con esa ID. En su lugar, parece que deseas buscar un documento en el que el campo nombre tenga un valor específico.

Para buscar un documento por un campo (como nombre), debes usar una consulta en lugar de acceder directamente a un documento por su ID. Aquí tienes cómo puedes corregir el método obtenerUsuario para que realice una consulta:

**Código corregido:**

public void obtenerUsuario(View view) {

String nombreBuscado = id.getText().toString().trim(); // Obtén el nombre que deseas buscar desde el EditText

// Realiza una consulta para buscar el documento donde el campo "nombre" coincida

firestore.collection("usuario")

.whereEqualTo("nombre", nombreBuscado)

.get()

.addOnSuccessListener(queryDocumentSnapshots -> {

if (!queryDocumentSnapshots.isEmpty()) {

// Obtén el primer documento que coincida

DocumentSnapshot documentSnapshot = queryDocumentSnapshots.getDocuments().get(0);

String nombreUsuario = documentSnapshot.getString("nombre");

String correoUsuario = documentSnapshot.getString("correo");

nombre.setText(nombreUsuario);

correo.setText(correoUsuario);

} else {

Toast.makeText(BuscarUsuario.this, "No existe un usuario con ese nombre", Toast.LENGTH\_SHORT).show();

}

})

.addOnFailureListener(e -> Toast.makeText(BuscarUsuario.this, "Error al obtener registro: " + e.getMessage(), Toast.LENGTH\_SHORT).show());

}

**Cambios realizados:**

1. **Uso de whereEqualTo**: Esto permite buscar documentos donde el campo nombre tenga un valor igual al que el usuario ingresó.
2. **Uso de get() para obtener una colección de documentos**: Como whereEqualTo puede devolver múltiples resultados, usamos get() para obtener una lista de coincidencias.
3. **Verificación de resultados vacíos**: Si no se encuentra ningún documento, se muestra un mensaje adecuado.

**Notas adicionales:**

* Asegúrate de que el EditText para el nombre del usuario (id) tenga el texto que deseas buscar antes de ejecutar la consulta.
* Si es posible que haya más de un documento con el mismo nombre, deberías decidir cómo manejar esos casos, ya que este código solo toma el primer resultado encontrado.

Con esta corrección, deberías poder buscar usuarios en tu base de datos Firebase Firestore correctamente.

**DIFERENCIAS ENTRE MÉTODOS PUBLICOS Y PRIVADOS DEL MAIN**

package com.daw.iniciosesion;  
  
import android.content.Intent;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.EditText;  
import android.widget.Toast;  
  
import androidx.activity.EdgeToEdge;  
import androidx.annotation.NonNull;  
import androidx.appcompat.app.AppCompatActivity;  
import androidx.core.graphics.Insets;  
import androidx.core.view.ViewCompat;  
import androidx.core.view.WindowInsetsCompat;  
  
import com.google.android.gms.tasks.OnCompleteListener;  
import com.google.android.gms.tasks.OnFailureListener;  
import com.google.android.gms.tasks.Task;  
import com.google.firebase.auth.FirebaseAuth;  
import com.google.firebase.firestore.DocumentReference;  
import com.google.firebase.firestore.FirebaseFirestore;  
import com.google.firebase.ktx.Firebase;  
  
import java.util.HashMap;  
import java.util.Map;  
  
public class MainActivity extends AppCompatActivity {  
 FirebaseAuth auth;  
 EditText id, nombre, ubicacion, inicio, fin;  
 FirebaseFirestore firestore;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 EdgeToEdge.*enable*(this);  
 setContentView(R.layout.*activity\_main*);  
 ViewCompat.*setOnApplyWindowInsetsListener*(findViewById(R.id.*main*), (v, insets) -> {  
 Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.*systemBars*());  
 v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);  
 return insets;  
 });  
  
 auth = FirebaseAuth.*getInstance*(); //le estamos dando un valor al atributo  
 firestore = FirebaseFirestore.*getInstance*();  
 id = findViewById(R.id.*idMercado*);  
 nombre = findViewById(R.id.*nombreMercado*);  
 ubicacion = findViewById(R.id.*ubiMercado*);  
 inicio = findViewById(R.id.*inicioMercado*);  
 fin = findViewById(R.id.*finMercado*);  
 }  
 public void cerrarSesion (View view) {  
 auth.signOut(); // Con esto hemos cerrado la sesión. Pero nos faltaría redirigir a inicio.  
 finish();  
 startActivity(new Intent(this, InicioSesion.class));  
 }  
  
 public void insertarMercado (View view) {  
 String idMercado = id.getText().toString().trim();  
 String nombreMercado = nombre.getText().toString().trim();  
 String ubicacionMercado = ubicacion.getText().toString().trim();  
 String inicioMercado = inicio.getText().toString().trim();  
 String finMercado = fin.getText().toString().trim();  
 //nos saltamos la comprobación de si los campos están vacios  
 agregarMercado(idMercado, nombreMercado, ubicacionMercado, inicioMercado, finMercado);  
 }  
  
 private void agregarMercado(String idMercado, String nombreMercado, String ubicacionMercado, String inicioMercado, String finMercado) {  
 Map<String, Object> map = new HashMap<>();  
 map.put("id", idMercado);  
 map.put("nombre", nombreMercado);  
 map.put("ubicacion", ubicacionMercado);  
 map.put("inicio", inicioMercado);  
 map.put("fin", finMercado);  
  
 firestore.collection("mercado").add(map).addOnCompleteListener(new OnCompleteListener<DocumentReference>() {  
 @Override  
 public void onComplete(@NonNull Task<DocumentReference> task) {  
 Toast.*makeText*(MainActivity.this, "Mercado Registrado", Toast.*LENGTH\_SHORT*).show();  
 }  
 }).addOnFailureListener(new OnFailureListener() {  
 @Override  
 public void onFailure(@NonNull Exception e) {  
 Toast.*makeText*(MainActivity.this, "Error al insertar", Toast.*LENGTH\_SHORT*).show();  
 }  
 });  
 }  
 public void buscarMercado (View view) {  
 startActivity(new Intent(this, BuscarMercado.class));  
 }  
  
}

Vamos a repasar los métodos que tienes en tu clase MainActivity y ver si es necesario declararlos de forma diferente en base a su uso. El objetivo principal es seguir las buenas prácticas de visibilidad y encapsulación.

**1. onCreate(Bundle savedInstanceState)**

Este es el método principal del ciclo de vida de una actividad en Android. Siempre debe ser **protected** porque es sobrescrito desde la clase AppCompatActivity. No se debe cambiar a private ni public, ya que su visibilidad está definida por el ciclo de vida de la actividad. **Está bien como está.**

**2. cerrarSesion(View view)**

Este método se encarga de cerrar sesión y redirigir a la pantalla de inicio de sesión. Lo estás declarando como **public**, lo cual es adecuado porque puede ser invocado desde una vista (probablemente desde el archivo XML de la interfaz de usuario como un OnClickListener), lo cual requiere que el método sea público. **Está bien como está.**

**3. insertarMercado(View view)**

Este método obtiene los valores de los campos del formulario y luego llama a agregarMercado. Lo estás declarando como **public**, pero en realidad, no es necesario que sea público si solo lo invocas dentro de la misma clase, como en este caso desde el método cerrarSesion. Podrías hacerlo **private** si no planeas invocar este método desde fuera de la clase.

**Cambiar a private:**

private void insertarMercado(View view) {

String idMercado = id.getText().toString().trim();

String nombreMercado = nombre.getText().toString().trim();

String ubicacionMercado = ubicacion.getText().toString().trim();

String inicioMercado = inicio.getText().toString().trim();

String finMercado = fin.getText().toString().trim();

agregarMercado(idMercado, nombreMercado, ubicacionMercado, inicioMercado, finMercado);

}

**4. agregarMercado(String idMercado, String nombreMercado, String ubicacionMercado, String inicioMercado, String finMercado)**

Este método realiza la operación de agregar un mercado a Firestore. Como ya mencionamos, debe ser **private**, ya que es un detalle de implementación interna de la clase y solo se llama dentro de insertarMercado. **Está bien como está.**

**5. buscarMercado(View view)**

Este método lanza la actividad BuscarMercado al hacer clic en un botón. Similar al método cerrarSesion, debe ser **public** ya que probablemente sea invocado desde la vista, es decir, desde el archivo de interfaz. **Está bien como está.**

**Recomendación general sobre visibilidad:**

* **public**: Se usa para métodos que necesitan ser accesibles fuera de la clase. Por ejemplo, los métodos llamados desde el XML (como los de los OnClickListeners en botones) deben ser públicos para que el sistema Android pueda invocar esos métodos.
* **private**: Se utiliza para métodos que solo se usan dentro de la clase. Es una buena práctica encapsular la lógica interna y no exponerla a otras clases innecesariamente.
* **protected**: Usado para métodos que pueden ser accesibles en clases hijas, pero no fuera del paquete. No es necesario en este caso porque no estás utilizando herencia.

**Resumen:**

* Los métodos cerrarSesion y buscarMercado deberían permanecer como **public** porque son llamados desde la interfaz (XML).
* El método insertarMercado podría ser **private** si solo se invoca dentro de la misma clase.
* El método agregarMercado debería ser **private**, ya que es un método auxiliar que solo se usa dentro de la clase.

# MÉTODO PARA EDITAR REGISTROS MERCADOS

Claro, te puedo ayudar a crear un método para editar y actualizar los valores de los registros de los mercados en Firestore. Para hacer esto, debemos realizar lo siguiente:

1. **Recuperar el documento de Firestore que deseas actualizar**.
2. **Modificar los campos del documento** con los nuevos valores.
3. **Guardar esos cambios** de nuevo en Firestore.

Para hacerlo, utilizaremos el método update() de Firestore, que nos permite actualizar campos específicos de un documento sin necesidad de sobrescribir todo el documento.

Aquí te dejo un ejemplo de cómo podrías implementar el método para editar un mercado en tu clase MainActivity:

**1. Método para Editar un Mercado**

Vamos a agregar un nuevo método llamado editarMercado que recibirá el idMercado (el identificador único del mercado) y los nuevos valores para los campos que deseas actualizar. Este método buscará el documento con ese idMercado y actualizará los campos correspondientes.

**Implementación:**

private void editarMercado(String idMercado, String nuevoNombre, String nuevaUbicacion, String nuevoInicio, String nuevoFin) {

// Referencia al documento del mercado en la colección "mercado"

DocumentReference mercadoRef = firestore.collection("mercado").document(idMercado);

// Crear un mapa con los nuevos valores a actualizar

Map<String, Object> map = new HashMap<>();

map.put("nombre", nuevoNombre);

map.put("ubicacion", nuevaUbicacion);

map.put("inicio", nuevoInicio);

map.put("fin", nuevoFin);

// Actualizar el documento en Firestore

mercadoRef.update(map)

.addOnCompleteListener(new OnCompleteListener<Void>() {

@Override

public void onComplete(@NonNull Task<Void> task) {

if (task.isSuccessful()) {

Toast.makeText(MainActivity.this, "Mercado actualizado correctamente", Toast.LENGTH\_SHORT).show();

} else {

Toast.makeText(MainActivity.this, "Error al actualizar el mercado", Toast.LENGTH\_SHORT).show();

}

}

})

.addOnFailureListener(new OnFailureListener() {

@Override

public void onFailure(@NonNull Exception e) {

Toast.makeText(MainActivity.this, "Error: " + e.getMessage(), Toast.LENGTH\_SHORT).show();

}

});

}

**Explicación:**

* **Referencia al documento**: Utilizamos firestore.collection("mercado").document(idMercado) para obtener la referencia al documento específico del mercado que queremos actualizar. Aquí, idMercado es el identificador único del mercado que quieres editar.
* **Mapa de actualización**: Creamos un HashMap con los campos que queremos actualizar (nombre, ubicacion, inicio, fin). En este mapa, las claves corresponden a los nombres de los campos en el documento de Firestore y los valores son los nuevos valores que el usuario ha introducido.
* **Actualizar documento**: Usamos update() para actualizar el documento en Firestore. Si el documento no existe, Firestore no lo creará, solo actualizará los campos especificados.
* **Manejo de éxito y error**: Al usar addOnCompleteListener y addOnFailureListener, podemos manejar los casos de éxito y error para informar al usuario si la actualización fue exitosa o si ocurrió un error.

**2. Cómo Llamar a este Método**

Ahora, necesitarías invocar este método desde un lugar adecuado en tu aplicación. Por ejemplo, si tienes un formulario de edición donde el usuario puede cambiar los datos de un mercado, podrías hacer algo como esto:

// Llamamos al método para editar un mercado

String idMercado = "12345"; // Este valor lo obtendrías de alguna parte, como una lista o un formulario

String nuevoNombre = "Nuevo Mercado";

String nuevaUbicacion = "Nueva Ubicación";

String nuevoInicio = "08:00";

String nuevoFin = "18:00";

// Llamamos al método editarMercado

editarMercado(idMercado, nuevoNombre, nuevaUbicacion, nuevoInicio, nuevoFin);

**3. Usando el Método en un Formulario de Edición**

Si tienes un formulario donde el usuario introduce nuevos valores, puedes obtener esos valores de los campos de texto (EditText) y pasarlos al método editarMercado. Por ejemplo:

// Suponiendo que tienes un formulario con campos EditText

String idMercado = id.getText().toString().trim();

String nuevoNombre = nombre.getText().toString().trim();

String nuevaUbicacion = ubicacion.getText().toString().trim();

String nuevoInicio = inicio.getText().toString().trim();

String nuevoFin = fin.getText().toString().trim();

// Llamar al método para editar el mercado

editarMercado(idMercado, nuevoNombre, nuevaUbicacion, nuevoInicio, nuevoFin);

**Consideraciones:**

* **Validación de datos**: Asegúrate de validar los datos antes de pasarlos al método editarMercado (por ejemplo, verificar que los campos no estén vacíos).
* **Manejo de errores**: Puedes mejorar el manejo de errores en el método para manejar casos como campos vacíos, problemas de conexión a la red, etc.

**Resumen:**

* El método editarMercado permite actualizar los campos de un mercado en Firestore.
* Utiliza la referencia al documento de Firestore y el método update() para realizar las actualizaciones de forma eficiente.
* Puedes llamar a este método desde un formulario de edición donde el usuario pueda cambiar los valores de los mercados.

¿Te gustaría que profundicemos más en alguna parte o necesitas alguna otra funcionalidad?

# MÉTODO PARA BUSCAR Y EDITAR MERCADOS REGISTRADOS

¡Entiendo! Si el objetivo es **buscar primero el mercado por su ID** antes de realizar la edición, podemos hacer lo siguiente:

1. **Buscar el documento** que queremos editar utilizando el idMercado (el identificador único del mercado).
2. **Recuperar los valores actuales** del mercado.
3. **Mostrar esos valores** en un formulario para que el usuario pueda editarlos.
4. **Actualizar el documento** con los nuevos valores introducidos por el usuario.

Para lograr esto, necesitamos realizar dos pasos principales:

* Primero **consultamos** el documento en Firestore.
* Luego **actualizamos** los campos que el usuario haya editado.

Voy a ajustar el código para reflejar estos pasos:

**Paso 1: Buscar el Mercado por su ID**

Primero, creamos un método para buscar el mercado usando su idMercado y obtener los datos actuales:

private void buscarYEditarMercado(String idMercado) {

// Referencia al documento del mercado en la colección "mercado"

DocumentReference mercadoRef = firestore.collection("mercado").document(idMercado);

// Buscar el documento en Firestore

mercadoRef.get().addOnCompleteListener(new OnCompleteListener<DocumentSnapshot>() {

@Override

public void onComplete(@NonNull Task<DocumentSnapshot> task) {

if (task.isSuccessful()) {

DocumentSnapshot document = task.getResult();

if (document.exists()) {

// Obtener los datos actuales del mercado

String nombre = document.getString("nombre");

String ubicacion = document.getString("ubicacion");

String inicio = document.getString("inicio");

String fin = document.getString("fin");

// Llenar los campos de edición con los datos actuales

nombreEditText.setText(nombre);

ubicacionEditText.setText(ubicacion);

inicioEditText.setText(inicio);

finEditText.setText(fin);

// Ahora puedes mostrar el formulario de edición al usuario

// Aquí podrías habilitar los campos EditText para que el usuario los edite

} else {

Toast.makeText(MainActivity.this, "No se encontró el mercado con el ID proporcionado", Toast.LENGTH\_SHORT).show();

}

} else {

Toast.makeText(MainActivity.this, "Error al obtener el mercado", Toast.LENGTH\_SHORT).show();

}

}

});

}

**Explicación:**

* **get()**: Usamos este método para obtener el documento con el idMercado especificado.
* **document.exists()**: Comprobamos si el documento con ese ID existe en Firestore.
* Si el documento existe, recuperamos los valores de los campos y los mostramos en los EditText del formulario para que el usuario pueda editarlos.
* Si no existe el documento o hay un error, mostramos un mensaje de error con Toast.

**Paso 2: Actualizar los Datos**

Luego, cuando el usuario haya editado los campos y haya presionado el botón de guardar, puedes llamar a editarMercado (que ya definimos antes) para actualizar los datos en Firestore:

private void editarMercado(String idMercado, String nuevoNombre, String nuevaUbicacion, String nuevoInicio, String nuevoFin) {

// Referencia al documento del mercado en la colección "mercado"

DocumentReference mercadoRef = firestore.collection("mercado").document(idMercado);

// Crear un mapa con los nuevos valores a actualizar

Map<String, Object> map = new HashMap<>();

map.put("nombre", nuevoNombre);

map.put("ubicacion", nuevaUbicacion);

map.put("inicio", nuevoInicio);

map.put("fin", nuevoFin);

// Actualizar el documento en Firestore

mercadoRef.update(map)

.addOnCompleteListener(new OnCompleteListener<Void>() {

@Override

public void onComplete(@NonNull Task<Void> task) {

if (task.isSuccessful()) {

Toast.makeText(MainActivity.this, "Mercado actualizado correctamente", Toast.LENGTH\_SHORT).show();

} else {

Toast.makeText(MainActivity.this, "Error al actualizar el mercado", Toast.LENGTH\_SHORT).show();

}

}

})

.addOnFailureListener(new OnFailureListener() {

@Override

public void onFailure(@NonNull Exception e) {

Toast.makeText(MainActivity.this, "Error: " + e.getMessage(), Toast.LENGTH\_SHORT).show();

}

});

}

**Paso 3: Integrar Todo Junto**

Finalmente, necesitas llamar a ambos métodos de manera apropiada. Primero, busca el mercado con su ID, muestra los datos en los campos de edición, luego, cuando el usuario haga clic en "Guardar", se actualizan esos datos en Firestore:

// Llamamos al método para buscar el mercado por su ID y editarlo

String idMercado = "12345"; // Este valor puede ser el ID que el usuario selecciona

buscarYEditarMercado(idMercado);

// Luego, cuando el usuario haga clic en el botón "Guardar":

String nuevoNombre = nombreEditText.getText().toString().trim();

String nuevaUbicacion = ubicacionEditText.getText().toString().trim();

String nuevoInicio = inicioEditText.getText().toString().trim();

String nuevoFin = finEditText.getText().toString().trim();

// Llamamos al método para actualizar el mercado con los nuevos valores

editarMercado(idMercado, nuevoNombre, nuevaUbicacion, nuevoInicio, nuevoFin);

**Consideraciones:**

* **Validación de datos**: Asegúrate de validar los campos antes de enviar la actualización. Por ejemplo, verifica que no estén vacíos.
* **Flujo de la aplicación**: El método buscarYEditarMercado debe ser invocado cuando se desee cargar un mercado para editarlo, y luego, cuando el usuario guarde los cambios, el método editarMercado debe ser llamado para actualizar los valores.
* **Interfaz de usuario**: Asegúrate de que la interfaz esté diseñada para permitir al usuario editar los campos, y que los valores actuales se muestren correctamente.

**Resumen:**

* **Buscar** el mercado usando su ID.
* **Mostrar los datos** actuales del mercado en un formulario de edición.
* **Actualizar** esos datos con los valores nuevos introducidos por el usuario.

¿Te gustaría que profundicemos más en la parte de la interfaz o en alguna otra parte del código?

# ACTUALIZAR MERCADO POR CAMPO ID

El problema con tu método de actualizar actualizarMercado es que estás utilizando el método document(idMercado) que busca directamente por el **ID del documento en Firestore**, pero lo que necesitas es buscar por el campo id que es diferente del ID del documento.

Para solucionar esto, primero debes buscar el documento que tenga el campo id igual al valor proporcionado, obtener su ID de documento y luego actualizarlo. A continuación, te muestro cómo puedes modificar el método actualizarMercado para que funcione correctamente:

**Código actualizado para actualizarMercado**

private void actualizarMercado(String idMercado, String nombreMercado, String ubicacionMercado, String inicioMercado, String finMercado) {

firestore.collection("mercado").whereEqualTo("id", idMercado).get()

.addOnSuccessListener(queryDocumentSnapshots -> {

if (!queryDocumentSnapshots.isEmpty()) {

// Obtén el primer documento que coincida

DocumentSnapshot documentSnapshot = queryDocumentSnapshots.getDocuments().get(0);

String docId = documentSnapshot.getId(); // ID del documento en Firestore

// Crea el mapa con los datos a actualizar

Map<String, Object> map = new HashMap<>();

map.put("id", idMercado);

map.put("nombre", nombreMercado);

map.put("ubicacion", ubicacionMercado);

map.put("inicio", inicioMercado);

map.put("fin", finMercado);

// Actualiza el documento utilizando su ID

firestore.collection("mercado").document(docId).update(map)

.addOnCompleteListener(task -> {

if (task.isSuccessful()) {

Toast.makeText(BuscarMercado.this, "Mercado Actualizado", Toast.LENGTH\_SHORT).show();

} else {

Toast.makeText(BuscarMercado.this, "Error al actualizar el mercado", Toast.LENGTH\_SHORT).show();

}

})

.addOnFailureListener(e -> {

Toast.makeText(BuscarMercado.this, "Error al actualizar: " + e.getMessage(), Toast.LENGTH\_LONG).show();

});

} else {

Toast.makeText(BuscarMercado.this, "No se encontró el mercado con el ID especificado", Toast.LENGTH\_SHORT).show();

}

})

.addOnFailureListener(e -> {

Toast.makeText(BuscarMercado.this, "Error al buscar el mercado: " + e.getMessage(), Toast.LENGTH\_LONG).show();

});

}

**Cambios realizados**

1. **Búsqueda por campo id:** Usamos .whereEqualTo("id", idMercado) para encontrar el documento basado en el campo id.
2. **Obtener ID del documento:** Una vez encontrada la coincidencia, se obtiene el ID real del documento con documentSnapshot.getId().
3. **Actualizar el documento:** Con el ID del documento obtenido, se llama a .document(docId).update(map).

**Flujo del método actualizado**

1. Busca en la colección mercado un documento que tenga un campo id con el valor proporcionado.
2. Si existe, obtiene el ID del documento en Firestore.
3. Usa el ID del documento para actualizar los campos especificados.
4. Maneja errores de búsqueda y actualización.

Con estos cambios, ahora podrás actualizar correctamente los registros basándote en el campo id en lugar del ID del documento.

# BORRAR MERCADO

Sí, es posible crear un método que elimine un mercado en función de su campo id. Esto se logra utilizando un enfoque similar al método de actualización: primero buscas el documento en la colección mercado cuyo campo id coincida con el valor proporcionado, obtienes el ID del documento, y luego lo eliminas.

Aquí tienes el método eliminarMercado implementado:

**Método para eliminar un mercado**

public void eliminarMercado(View view) {

String idMercado = mdo\_id.getText().toString().trim();

if (idMercado.isEmpty()) {

Toast.makeText(this, "Ingrese el ID del mercado a eliminar", Toast.LENGTH\_SHORT).show();

return;

}

firestore.collection("mercado").whereEqualTo("id", idMercado).get()

.addOnSuccessListener(queryDocumentSnapshots -> {

if (!queryDocumentSnapshots.isEmpty()) {

// Obtén el primer documento que coincida

DocumentSnapshot documentSnapshot = queryDocumentSnapshots.getDocuments().get(0);

String docId = documentSnapshot.getId(); // ID del documento en Firestore

// Elimina el documento utilizando su ID

firestore.collection("mercado").document(docId).delete()

.addOnCompleteListener(task -> {

if (task.isSuccessful()) {

Toast.makeText(BuscarMercado.this, "Mercado eliminado correctamente", Toast.LENGTH\_SHORT).show();

limpiarEntrada(view); // Limpia los campos después de la eliminación

} else {

Toast.makeText(BuscarMercado.this, "Error al eliminar el mercado", Toast.LENGTH\_SHORT).show();

}

})

.addOnFailureListener(e -> {

Toast.makeText(BuscarMercado.this, "Error al eliminar: " + e.getMessage(), Toast.LENGTH\_LONG).show();

});

} else {

Toast.makeText(this, "No se encontró el mercado con el ID especificado", Toast.LENGTH\_SHORT).show();

}

})

.addOnFailureListener(e -> {

Toast.makeText(BuscarMercado.this, "Error al buscar el mercado: " + e.getMessage(), Toast.LENGTH\_LONG).show();

});

}

**Explicación del código**

1. **Validación del campo idMercado:** Se verifica que el usuario haya ingresado un ID antes de intentar realizar la operación.
2. **Búsqueda por campo id:** Se utiliza whereEqualTo("id", idMercado) para encontrar el documento en Firestore.
3. **Obtención del ID del documento:** Una vez encontrado el documento, se obtiene su ID con documentSnapshot.getId().
4. **Eliminación del documento:** Usando el ID del documento, se llama al método delete() de Firestore.
5. **Mensajes de confirmación o error:** Se notifica al usuario si la operación fue exitosa o si hubo errores.

**Uso del método**

* Puedes llamar a este método desde un botón en tu interfaz gráfica. Por ejemplo, en el XML del diseño puedes asignar el método al botón:
* <Button
* android:id="@+id/btn\_eliminar\_mercado"
* android:layout\_width="wrap\_content"
* android:layout\_height="wrap\_content"
* android:text="Eliminar Mercado"
* android:onClick="eliminarMercado" />

**Flujo del método**

1. El usuario ingresa el id del mercado que desea eliminar.
2. El método busca el documento con ese id en la colección mercado.
3. Si encuentra el documento, lo elimina.
4. Si no lo encuentra o hay algún error, muestra un mensaje informativo.

Con este método, podrás eliminar mercados basándote en el valor del campo id almacenado en la colección mercado.