Manual Backend - SmartHome

UNIVERSIDAD NACIONAL DE COLOMBIA

Integrantes:

Cristian Camilo Castillo Meneses

Juan Felipe Velasquez

Luis Alvarado Olivos

Universidad Nacional de Colombia

Programación Orientada a Objetos

Docente: Néstor German Bolivar Pulgarin

2022

Indice

Resumen/Abstract

Hola no hay resumen pq aún no está terminado el texto

Palabras clave: primera línea con sangría, escribe, tu texto, aquí

Introducción

Proposito del manual

Este manual es una guía completa diseñada para ofrecer a los desarrolladores una comprensión profunda del funcionamiento de la aplicación SmartHome desde una perspectiva técnica y de desarrollo. Su objetivo es brindar un conocimiento exhaustivo de la arquitectura, los componentes y las funcionalidades clave de la aplicación.

Dentro del manual, encontrará una serie de secciones detalladas que cubren: un análisis de la estructura interna y cómo se conectan sus diferentes módulos; una revisión de los elementos fundamentales que conforman la aplicación, desde la interfaz de usuario hasta las integraciones con servicios externos; una descripción de las características principales que ofrece SmartHome y cómo están implementadas; y, para finalizar, una explicación del hardware desarrollado para demostrar las capacidades de la aplicación en un entorno real.

El manual está escrito en un lenguaje claro y accesible, acompañado de diagramas ilustrativos y ejemplos de código prácticos, lo que facilita su comprensión y aplicación en el desarrollo.

Descripción del proyecto

La aplicación SmartHome es una aplicación para dispositivos móviles que ofrece a los usuarios la posibilidad de controlar de manera eficiente y segura los dispositivos en su hogar mediante una interfaz intuitiva. En esta, los usuarios pueden registrarse y acceder de forma segura, garantizando la protección de sus datos personales. Una vez autenticados, pueden explorar el menú principal de la aplicación para gestionar todos los aspectos de su hogar inteligente, desde la comodidad de su dispositivo móvil.

Así mismo, con la aplicación los usuarios pueden monitorear y controlar el estado de los dispositivos conectados en diferentes habitaciones de su hogar. La aplicación permite seleccionar una habitación específica para verificar qué dispositivos están activos y realizar ajustes en su configuración cuando sea necesario. Esta funcionalidad asegura que la información sobre el estado del hogar esté siempre actualizada, proporcionando un control total sobre el entorno doméstico y permitiendo que los usuarios mantengan su hogar en óptimas condiciones.

Además, la aplicación envía notificaciones para alertar a los usuarios cuando algún dispositivo ha permanecido encendido durante un período prolongado sin uso. Esto no solo ayuda a evitar el desperdicio de energía, sino que también contribuye a la seguridad del hogar. Los usuarios pueden personalizar estas alertas según sus necesidades, asegurándose de estar siempre informados sobre lo que sucede en su hogar. En conjunto, estas características hacen de SmartHome una herramienta esencial para cualquier hogar inteligente, ofreciendo un control constante y seguro sobre los dispositivos conectados.

Tecnologias Utilizadas

El backend de la aplicación se ha desarrollado utilizando tecnologías clave que aseguran un rendimiento robusto y eficiente. El lenguaje de programación elegido para este proyecto es Java, conocido por su estabilidad y versatilidad en el desarrollo de aplicaciones a gran escala. Java permite manejar múltiples procesos de manera eficiente, lo que es crucial para una aplicación que necesita gestionar y responder a varias interacciones de usuarios simultáneamente. Además, su orientación a objetos facilita la creación de un código modular y reutilizable, lo que contribuye a la mantenibilidad y escalabilidad del proyecto.

La aplicación también se basa en Firebase para gestionar la autenticación de usuarios y almacenar la información en tiempo real. Firebase es una plataforma de desarrollo de aplicaciones proporcionada por Google, que ofrece una base de datos en la nube con sincronización en tiempo real, lo que significa que cualquier cambio en los datos se refleja instantáneamente en todos los dispositivos conectados. Esta característica es fundamental para SmartHome, ya que asegura que los usuarios siempre vean la información más reciente sobre el estado de sus dispositivos domésticos. Además, Firebase proporciona herramientas de autenticación seguras, permitiendo que los usuarios se registren e inicien sesión utilizando métodos como el correo electrónico, Google, entre otros, sin comprometer la seguridad de sus datos.

Para el entorno de desarrollo, se utilizó Android Studio, el IDE oficial para el desarrollo de aplicaciones Android. Android Studio proporciona un entorno de trabajo optimizado para escribir código en Java y permite una integración perfecta con Firebase. Además, ofrece herramientas avanzadas para el diseño de interfaces de usuario, la depuración y el monitoreo del rendimiento de la aplicación. Con su editor de código inteligente y su capacidad para emular diferentes dispositivos Android, Android Studio facilita el desarrollo de aplicaciones eficientes y visualmente atractivas, garantizando una excelente experiencia de usuario en SmartHome.

Configuración del entorno de desarrollo

Requisitos del sistema

Para utilizar la aplicación SmartHome, es importante asegurarse de cumplir con ciertos requisitos del sistema. En primer lugar, el sistema operativo debe ser compatible con Android, preferiblemente en su versión 5.0 o superior para asegurar la mejor experiencia de usuario y

compatibilidad con todas las funcionalidades de la aplicación. En cuanto al desarrollo, se necesita tener instalada la versión más reciente de Java, al menos Java 8, ya que es la base sobre la cual está construido el backend de la aplicación. Además, Android Studio debe estar instalado en su versión más actualizada para garantizar que todas las herramientas y funciones necesarias estén disponibles. Finalmente, es esencial que Firebase esté correctamente configurado, ya que se utiliza para la autenticación y el manejo de la base de datos en tiempo real.

Instalación de dependencias

Una vez asegurados los requisitos del sistema, el siguiente paso es la instalación de las dependencias necesarias. Para ello, se deben ejecutar comandos específicos dentro de Android Studio. Al abrir el proyecto, Android Studio gestionará automáticamente la instalación de las librerías requeridas a través de Gradle, que es la herramienta de automatización de compilaciones integrada en el IDE. Es recomendable verificar que todas las dependencias estén correctamente instaladas y que no haya errores en el proceso de sincronización de Gradle, ya que esto podría afectar el funcionamiento del proyecto.

Configuración de la base de datos

La configuración de la base de datos es otro paso fundamental. En este caso, Firebase se utiliza como la solución de base de datos en tiempo real. Para comenzar, es necesario crear un proyecto en Firebase y conectar la aplicación SmartHome a este proyecto. Esto implica configurar las variables de entorno en el archivo `google-services.json`, que debe ser descargado desde la consola de Firebase y colocado en el directorio adecuado del proyecto Android. Además, es importante asegurarse de que las reglas de seguridad de Firebase estén configuradas correctamente para permitir el acceso adecuado a los datos. Una vez completados

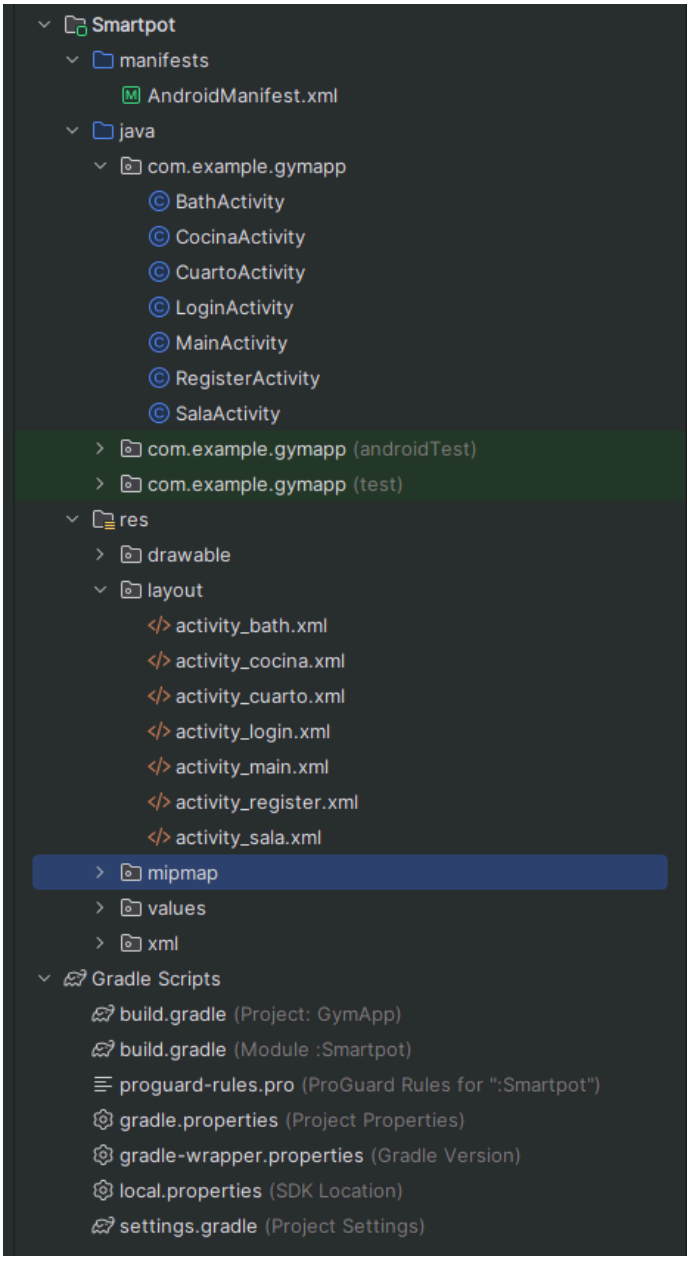
estos pasos, cualquier cambio en la estructura de la base de datos se reflejará automáticamente sin necesidad de migraciones tradicionales, gracias a las características de Firebase.

Ejecución de la app

Finalmente, para ejecutar el servidor de desarrollo y comenzar a trabajar con la aplicación, se debe iniciar Android Studio y cargar el proyecto de SmartHome. Una vez que el proyecto esté completamente cargado, se puede iniciar la aplicación en un emulador de Android o en un dispositivo físico conectado, utilizando el botón "Run" en Android Studio. Esto iniciará el servidor local y desplegará la aplicación, la cual puede ser accesible a través de la interfaz de usuario del emulador o del dispositivo físico. Al hacer esto, se podrá visualizar y probar la aplicación en un entorno de desarrollo que simula el comportamiento real de SmartHome.

Estructura del proyecto

La aplicación está organizada de una manera que facilita tanto su mantenimiento como la incorporación de nuevas funcionalidades. A continuación, se proporciona una visión general de su estructura. Esta estructura incluye directorios esenciales como el manifiesto de la aplicación, el código fuente Java, los recursos visuales y los scripts de Gradle. Cada uno de estos elementos cumple un papel importante en el funcionamiento integral del proyecto:



Archivo `AndroidManifest.xml`

El archivo `AndroidManifest.xml` es un componente esencial de cualquier aplicación Android, ya que define la estructura básica y las configuraciones necesarias para el funcionamiento de la aplicación. A continuación, se detalla la estructura y los elementos clave presentes en el archivo `AndroidManifest.xml` de la aplicación SmartHome.

Declaración de Permisos

Los permisos son declaraciones que informan al sistema operativo sobre las funcionalidades que la aplicación necesita utilizar. En el archivo `AndroidManifest.xml`, se han declarado los siguientes permisos:

- *android.permission.INTERNET:* Permite que la aplicación acceda a Internet. Este permiso es crucial para que SmartHome pueda comunicarse con servicios externos, como APIs o Firebase, y para la transferencia de datos en tiempo real.
- *android.permission.POST_NOTIFICATIONS*: Autoriza a la aplicación a enviar notificaciones push a los usuarios. Esto es útil para alertar a los usuarios sobre eventos importantes, como cambios en el estado de los dispositivos conectados.
- *android.permission.VIBRATE:* Permite que la aplicación utilice la función de vibración del dispositivo. Este permiso se emplea en conjunto con las notificaciones para proporcionar alertas más efectivas y personalizadas.

```
4    <uses-permission android:name="android.permission.INTERNET" />
5    <uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
6    <uses-permission android:name="android.permission.VIBRATE" />
```

Elemento <application>

El bloque <application> contiene configuraciones globales que afectan a toda la aplicación. A continuación, se describen los atributos y subcomponentes principales definidos dentro de este elemento:

- *`android:allowBackup="true":* Permite que la aplicación realice copias de seguridad automáticas de sus datos.
- *android:dataExtractionRules="@xml/data_extraction_rules" y android:fullBackupContent="@xml/backup_rules":* Especifican las reglas para la

extracción y el contenido de las copias de seguridad, definiendo qué datos se incluyen o excluyen.

- *android:icon="@mipmap/smth":* Define el ícono de la aplicación que se muestra en el dispositivo del usuario.
- *android:label="@string/SmartPot":* Establece el nombre visible de la aplicación, en este caso, "SmartPot".
- *android:supportsRtl="true":* Indica que la aplicación soporta layouts de derecha a izquierda, lo cual es importante para idiomas que se leen de esta manera.
- *android:theme="@style/Theme.GymApp":* Define el tema visual de la aplicación, incluyendo colores, fuentes y estilos generales.
- *tools:targetApi="31":* Especifica que la aplicación está dirigida a la API de Android 31 (Android 12).

```
9     <application
10        android:allowBackup="true"
11        android:dataExtractionRules="@xml/data_extraction_rules"
12        android:fullBackupContent="@xml/backup_rules"
13        android:icon="@mipmap/smth"
14        android:label=""
15        android:supportsRtl="true"
16        android:theme="@style/Theme.GymApp"
17        tools:targetApi="31">
```

Declaración de Actividades

Las actividades representan las diferentes pantallas o interfaces de usuario de la aplicación. Cada actividad se declara dentro del elemento <application>, especificando su nombre de clase y otras configuraciones relevantes. A continuación, se describen las actividades definidas:

- *LoginActivity:*

- *android:name=".LoginActivity":* Especifica la clase que implementa esta actividad.
- *android:exported="true":* Indica que esta actividad puede ser iniciada por otras aplicaciones externas.
- *<intent-filter>:* Define que esta actividad es el punto de entrada principal de la aplicación. Los filtros de intención `MAIN` y `LAUNCHER` hacen que `LoginActivity` se inicie cuando el usuario abre la aplicación desde el menú principal del dispositivo.

```
18        <activity
19            android:name=".LoginActivity"
20            android:exported="true">
21            <intent-filter>
22                <action android:name="android.intent.action.MAIN" />
23
24                <category android:name="android.intent.category.LAUNCHER" />
25            </intent-filter>
26        </activity>
```

- Otras Actividades:
  - *RegisterActivity, MainActivity, CuartoActivity, BathActivity, SalaActivity, CocinaActivity:* Estas actividades gestionan diferentes funcionalidades y secciones de la aplicación SmartHome. Por ejemplo, `RegisterActivity` maneja el registro de nuevos usuarios, mientras que `CuartoActivity`, `BathActivity`, `SalaActivity` y `CocinaActivity` están dedicadas al control de dispositivos en habitaciones específicas de la casa inteligente.
  - *android:exported="true":* Solo algunas actividades, como `CuartoActivity`, están marcadas como exportadas, lo que significa que pueden ser accesibles desde otras aplicaciones si es necesario. Las demás actividades no lo están, restringiendo su acceso exclusivamente dentro de la aplicación.

```
27          <activity android:name=".RegisterActivity" />
28          <activity android:name=".MainActivity" />
29          <activity
30              android:name=".CuartoActivity"
31              android:exported="true" />
32          <activity android:name=".BathActivity" />
33          <activity android:name=".SalaActivity" />
34          <activity android:name=".CocinaActivity" />
35
36
37      </application>
38
```

Flujo de la Aplicación

El flujo de navegación dentro de la aplicación está determinado por la configuración de las actividades en el `AndroidManifest.xml`. Al iniciar la aplicación, el sistema operativo Android busca la actividad con el filtro de intención `MAIN` y `LAUNCHER`, que en este caso es `LoginActivity`. Esto establece que la primera pantalla que verá el usuario será la pantalla de inicio de sesión.

Desde `LoginActivity`, una vez que el usuario se autentica correctamente, la aplicación puede navegar a `MainActivity`, que actúa como el punto central para acceder a las diferentes secciones de la aplicación, como el registro de nuevos usuarios (`RegisterActivity`) o el control de dispositivos en distintas habitaciones (`CuartoActivity`, `BathActivity`, etc.).

Este diseño asegura que solo los usuarios autenticados puedan acceder a las funcionalidades principales de la aplicación, manteniendo la seguridad e integridad de los datos personales y del control de dispositivos inteligentes.

Directorio *java/com.example.gymapp/*

Este es el núcleo de la aplicación, donde se encuentra el código fuente en Java. En este proyecto, cada clase representa una Actividad específica en la aplicación móvil:

- *LoginActivity*: Maneja la lógica de autenticación de usuarios.
- *RegisterActivity*: Proporciona una interfaz para el registro de nuevos usuarios.
- *MainActivity*: Actúa como el punto de control principal, desde donde los usuarios pueden acceder a diferentes funcionalidades de la aplicación.
- Actividades específicas como *CuartoActivity*, *CocinaActivity*, *SalaActivity*, *BathActivity* gestionan la interacción con dispositivos inteligentes en distintas habitaciones del hogar.

Cada una de estas actividades está vinculada a un archivo XML correspondiente en la carpeta de layouts, donde se define el diseño visual que verá el usuario.

LoginActivity

La clase *LoginActivity* es una de las actividades más importantes dentro de la aplicación SmartHome, ya que se encarga del proceso de autenticación de usuarios. Esta actividad permite que los usuarios registrados inicien sesión utilizando su correo electrónico y contraseña. Además, proporciona un enlace para que los nuevos usuarios puedan registrarse en la aplicación. A continuación, se explican los componentes clave y el funcionamiento de la actividad.

Componentes principales

- EditText:
    - *editTextEmail*: Campo de texto donde el usuario introduce su dirección de correo electrónico.
    - *editTextPassword:* Campo de texto donde el usuario introduce su contraseña.
- Button:

- *buttonLogin:* Botón que, al ser presionado, inicia el proceso de autenticación.

- *TextView:*
  - *textViewRegisterLink:* Texto interactivo que, al ser presionado, redirige a la pantalla de registro de nuevos usuarios (actividad *RegisterActivity*).
- *FirebaseAuth:*
  - *mAuth*: Es una instancia de FirebaseAuth que permite interactuar con Firebase para autenticar a los usuarios utilizando sus credenciales (correo electrónico y contraseña).

Flujo de funcionamiento

1. Inicialización de los Componentes:

- En el método *onCreate()*, se vinculan los campos de texto, el botón y el enlace de registro con sus respectivas vistas en el archivo XML de la interfaz de usuario (*activity_login.xml*).

- Se instancia el objeto *FirebaseAuth*, que maneja las operaciones de autenticación.

```
36          editTextEmail = findViewById(R.id.editTextEmail);
37          editTextPassword = findViewById(R.id.editTextPassword);
38          buttonLogin = findViewById(R.id.buttonLogin);
39          textViewRegisterLink = findViewById(R.id.textViewRegisterLink);
40
41          mAuth = FirebaseAuth.getInstance();
```

2. Proceso de Autenticación:

- Cuando el usuario hace clic en el botón *Login*, se recuperan las entradas del correo electrónico y la contraseña. Luego, se utiliza *signInWithEmailAndPassword()* de FirebaseAuth para autenticar al usuario con esas credenciales.

- Si el inicio de sesión es exitoso, se redirige al usuario a la *MainActivity*, que actúa como el menú principal de la aplicación.

- En caso de un error (por ejemplo, si las credenciales son incorrectas), se puede mostrar un mensaje de error al usuario.

```
buttonLogin.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String email = editTextEmail.getText().toString().trim();
        String password = editTextPassword.getText().toString().trim();

        mAuth.signInWithEmailAndPassword(email, password)
                .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
                    @Override
                    public void onComplete(@NonNull Task<AuthResult> task) {
                        if (task.isSuccessful()) {
                            startActivity(new Intent( packageContext: LoginActivity.this, MainActivity.class));
                            finish();
                        } else {
                        }
                    }
                });
    }
});
```

3. Enlace de Registro:

- Si el usuario no tiene una cuenta, puede hacer clic en el *textViewRegisterLink* que lo llevará a la actividad de registro. Esto se gestiona utilizando un *Intent* que inicia la actividad *RegisterActivity*.

```
textViewRegisterLink.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        startActivity(new Intent( packageContext: LoginActivity.this, RegisterActivity.class));
    }
});
```

RegisterActivity

La clase *RegisterActivity* gestiona el proceso de registro de nuevos usuarios en la aplicación SmartHome. Al igual que en la *LoginActivity*, Firebase Authentication se utiliza para manejar la creación de usuarios, lo que garantiza que el registro se realice de manera segura y eficiente. A continuación, se detalla el funcionamiento de esta actividad:

Componentes principales

- EditText:
    - *editTextName:* Campo de texto donde el usuario introduce su nombre completo.
    - *editTextEmail*: Campo de texto donde el usuario introduce su dirección de correo electrónico.
    - *editTextPassword:* Campo de texto donde el usuario introduce su contraseña.
- Button:
    - *buttonRegister*: Botón que, al ser presionado, inicia el proceso de registro del usuario.
- FirebaseAuth:
    - *mAuth*: Es una instancia de FirebaseAuth que se utiliza para crear un nuevo usuario utilizando el correo electrónico y la contraseña que proporciona el usuario.

Flujo de funcionamiento

1. Inicialización de los Componentes:

En el método *onCreate()*, se vinculan los campos de texto y el botón de registro con sus respectivas vistas definidas en el archivo XML (*activity_register.xml*). Además, se inicializa la instancia de *FirebaseAuth*, que es la que se encargará de manejar el registro del usuario.

```
28          editTextName = findViewById(R.id.editTextName);
29          editTextEmail = findViewById(R.id.editTextEmail);
30          editTextPassword = findViewById(R.id.editTextPassword);
31          buttonRegister = findViewById(R.id.buttonRegister);
32
33          mAuth = FirebaseAuth.getInstance();
```

2. Proceso de Registro:

- Cuando el usuario ingresa su información y presiona el botón de registro, la aplicación obtiene los valores introducidos en los campos de texto (nombre, correo electrónico y contraseña). Luego, se utiliza el método *createUserWithEmailAndPassword()* de Firebase para crear una nueva cuenta de usuario.

- Si el registro es exitoso, el usuario es redirigido automáticamente a la *MainActivity*, que es el punto central de la aplicación una vez que el usuario está autenticado.

- En caso de un error (por ejemplo, si el correo ya está registrado o la contraseña no cumple con los requisitos mínimos), se puede mostrar un mensaje de error al usuario.

```
40          buttonRegister.setOnClickListener(new View.OnClickListener() {
41              @Override
42              public void onClick(View v) {
43                  String name = editTextName.getText().toString().trim();
44                  String email = editTextEmail.getText().toString().trim();
45                  String password = editTextPassword.getText().toString().trim();
46
47                  mAuth.createUserWithEmailAndPassword(email, password)
48                          .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
49                              @Override
50                              public void onComplete(@NonNull Task<AuthResult> task) {
51                                  if (task.isSuccessful()) {
52                                      startActivity(new Intent( packageContext: RegisterActivity.this, MainActivity.class));
53                                      finish();
54                                  } else {
55                                  }
56                              }
57                          });
58              }
59          });
60      }
```

3. Gestión de Errores:

- El bloque *onComplete()* maneja tanto el éxito como el fracaso del proceso de registro. En caso de éxito, se redirige al usuario a la pantalla principal (*MainActivity*), mientras que los errores se pueden gestionar mostrando mensajes relevantes (por ejemplo, si el correo electrónico ya está en uso o si la contraseña es demasiado corta).

MainActivity

La clase *MainActivity* es el punto central desde donde el usuario navega hacia diferentes partes de la aplicación para gestionar y controlar los dispositivos en las distintas áreas de su hogar inteligente. Esta clase actúa como el "panel de control" desde el cual se accede a otras actividades que representan habitaciones o áreas específicas (como el cuarto, la sala, el baño, y la cocina). A continuación, se describe el flujo y los componentes de la actividad.

Componentes principales

- Buttons:
    - *buttonUpperBody:* Botón que, al ser presionado, redirige a la actividad *CuartoActivity*, donde el usuario puede controlar los dispositivos del cuarto.
    - *Button1, button2, button3:* Estos botones redirigen a otras actividades que representan diferentes habitaciones de la casa:
        - *button1*: Redirige a *SalaActivity.*
        - *button2:* Redirige a *BathActivity.*
        - *button3:* Redirige a *CocinaActivity.*
- TextView:
    - *textViewUsername:* Un componente de texto que puede mostrar el nombre de usuario. Aunque no está configurado en el código actual, podría utilizarse para personalizar la experiencia del usuario mostrando su nombre después de iniciar sesión.
- FirebaseAuth:
    - *mAuth*: Es una instancia de FirebaseAuth que se utiliza para manejar la autenticación del usuario. Aunque no se hace explícito en este fragmento de

código, se puede utilizar para verificar si el usuario está autenticado y redirigirlo si no lo está.

Flujo de funcionamiento

1. Inicialización de los componentes:

En el método *onCreate()*, se inicializan los botones y el componente de texto vinculándolos con sus respectivas vistas definidas en el archivo XML (*activity_main.xml*). También se inicializa *FirebaseAuth*, aunque en este fragmento no se le da uso, podría ser utilizado para funciones de autenticación como cerrar sesión o verificar el estado del usuario.

```
23          mAuth = FirebaseAuth.getInstance();
24
25          buttonUpperBody = findViewById(R.id.buttonUpperBody);
26          button1 = findViewById(R.id.button1);
27          button2 = findViewById(R.id.button2);
28          button3 = findViewById(R.id.button3);
```

2. Navegación a otras actividades:

Los botones en la pantalla principal permiten al usuario navegar hacia diferentes actividades, cada una representando una habitación o área del hogar. Cada botón tiene un *OnClickListener*, que responde al clic del usuario y redirige a la actividad correspondiente mediante un *Intent*.

- Cuarto: El botón *buttonUpperBody* lleva a *CuartoActivity*, que gestiona el control de los dispositivos del cuarto:

```
33          buttonUpperBody.setOnClickListener(new View.OnClickListener() {
34              @Override
35              public void onClick(View v) {
36                  Intent intent = new Intent( packageContext: MainActivity.this, CuartoActivity.class);
37                  startActivity(intent);
38              }
39          });
```

- Sala: El botón *button1* lleva a *SalaActivity*:

```
43          button1.setOnClickListener(new View.OnClickListener() {
44              @Override
45              public void onClick(View v) {
46                  Intent intent = new Intent( packageContext: MainActivity.this, SalaActivity.class);
47                  startActivity(intent);
48              }
49          });
```

- Baño: El botón *button2* lleva a *BathActivity*:

```
51          button2.setOnClickListener(new View.OnClickListener() {
52              @Override
53              public void onClick(View v) {
54                  Intent intent = new Intent( packageContext: MainActivity.this, BathActivity.class);
55                  startActivity(intent);
56              }
57          });
```

- Cocina: El botón *button3* lleva a *CocinaActivity*:

```
59          button3.setOnClickListener(new View.OnClickListener() {
60              @Override
61              public void onClick(View v) {
62                  Intent intent = new Intent( packageContext: MainActivity.this, CocinaActivity.class);
63                  startActivity(intent);
64              }
65          });
```

SalaActivity

La clase *SalaActivity* es responsable de gestionar y controlar los dispositivos conectados en la sala de la aplicación. En esta actividad, el usuario puede interactuar con varios dispositivos en tiempo real, tales como las luces, el balcón y la puerta, utilizando interruptores de tipo *ToggleButton*. La clase también está integrada con Firebase Realtime Database, lo que permite que los estados de los dispositivos se sincronicen en tiempo real y se almacenen de forma remota. A continuación, se explica su funcionamiento:

Componentes principales

- Buttons:
    - *buttonBack4*: Botón que permite al usuario regresar a la actividad anterior.

- TextViews:
    - *ts_texto, tgB_txt, tp_txt:* Estas vistas de texto se utilizan para mostrar el estado de los dispositivos en la interfaz de usuario.
- ToggleButtons:
    - *toggleButton7*: Controla el estado de la luz de la sala.
    - *toggleButton8*: Controla el estado del balcón (abierto/cerrado).
    - *toggleButton9*: Controla el estado de la puerta (abierta/cerrada).
- ImageViews:
    - *focoApagado1*: Imagen que representa el estado de la luz (encendida o apagada).
    - *balconCerrado*: Imagen que representa el estado del balcón.
    - *puertaCerrada*: Imagen que representa el estado de la puerta.
- Firebase Realtime Database:
    - *mDatabase*: Es una instancia de Firebase Realtime Database que se utiliza para leer y actualizar los estados de los dispositivos en la sala. Cada cambio en los *ToggleButtons* se refleja inmediatamente en Firebase, y cualquier actualización en Firebase se sincroniza en tiempo real con la interfaz de usuario.

Flujo de funcionamiento

1. Inicialización de los componentes:

En el método *onCreate()*, se inicializan los botones, los textos y las imágenes que representan el estado de los dispositivos, vinculándolos con sus respectivas vistas definidas en el archivo XML (*activity_sala.xml*). Además, se inicializa la referencia a Firebase para interactuar con la base de datos en tiempo real.

```
        ts_texto = (TextView) findViewById(R.id.ts_texto);
        tgB_txt = (TextView) findViewById(R.id.tgB_txt);
        tp_txt = (TextView) findViewById(R.id.tp_txt);
        mDatabase = FirebaseDatabase.getInstance().getReference();
        toggleButton7 = findViewById(R.id.toggleButton7);
        toggleButton8 = findViewById(R.id.toggleButton8);
        toggleButton9 = findViewById(R.id.toggleButton9);
        focoApagado1 = findViewById(R.id.FocoApagadoSala);
        balconCerrado = findViewById(R.id.BalconCerradoSala);
        puertaCerrada = findViewById(R.id.PuertaCerradaSala);
```

2. Control de la luz de la sala:

- *toggleButton7* gestiona el estado de la luz de la sala. Cuando el usuario cambia el estado del botón, la imagen asociada de la luz cambia entre encendida y apagada.
- Se utiliza Firebase para mantener el estado de la luz sincronizado en tiempo real. Si se cambia el estado de la luz desde otro dispositivo, Firebase actualiza automáticamente el estado en la aplicación.

```
        toggleButton7.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (hg==0) {
                    focoApagado1.setImageResource(R.drawable.foco_apagado);
                } else {
                    focoApagado1.setImageResource(R.drawable.foco_encendido);
                }

            }
        });



        mDatabase.child( pathString: "sala").child( pathString: "LuzSala").addValueEventListener(new ValueEventListener() {
            @Override  2 usages
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                if (dataSnapshot.exists()) {
                    // Obtener y actualizar el valor de 'ha'
                    hg = Integer.parseInt(dataSnapshot.getValue().toString());

                    // Actualizar el TextView y el estado del ToggleButton
                    ts_texto.setText(String.valueOf(hg));
                    toggleButton7.setChecked(hg == 1);
                }
            }
```

```
        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            // Manejo de errores
        }
    });

    // Listener para manejar los cambios de estado del ToggleButton
    toggleButton7.setOnCheckedChangeListener((buttonView, isChecked) -> {
        // Actualizar la variable 'ha' según el estado del ToggleButton
        hg = isChecked ? 1 : 0;

        // Actualizar el valor de "luz cuarto" en Firebase
        mDatabase.child( pathString: "sala").child( pathString: "LuzSala").setValue(hg);
    });
```

3. Control del balcón:

- *toggleButton8* gestiona el estado del balcón, cambiando entre abierto y cerrado. El estado del balcón se sincroniza con Firebase de manera similar a la luz.

```
toggleButton8.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (hh==0) {
            balconCerrado.setImageResource(R.drawable.balcon_cerrado);
        } else {
            balconCerrado.setImageResource(R.drawable.balcon_abierto);
        }

    }
});

mDatabase.child( pathString: "sala").child( pathString: "BalconSala").addValueEventListener(new ValueEventListener() {
    @Override  2 usages
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        if (dataSnapshot.exists()) {
            // Obtener y actualizar el valor de 'ha'
            hh = Integer.parseInt(dataSnapshot.getValue().toString());

            // Actualizar el TextView y el estado del ToggleButton
            tgB_txt.setText(String.valueOf(hh));
            toggleButton8.setChecked(hh == 1);
        }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {
        // Manejo de errores
    }
});

// Listener para manejar los cambios de estado del ToggleButton
toggleButton8.setOnCheckedChangeListener((buttonView, isChecked) -> {
    // Actualizar la variable 'ha' según el estado del ToggleButton
    hh = isChecked ? 1 : 0;

    // Actualizar el valor de "luz cuarto" en Firebase
    mDatabase.child( pathString: "sala").child( pathString: "BalconSala").setValue(hh);
});
```

4. Control de la puerta:

- *toggleButton9* gestiona el estado de la puerta de la sala, cambiando entre abierta y cerrada, y sincroniza su estado con Firebase.

```
        toggleButton9.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (hi==0) {
                    puertaCerrada.setImageResource(R.drawable.closed_door);
                } else {
                    puertaCerrada.setImageResource(R.drawable.open_door);
                }

            }
        });

        mDatabase.child( pathString: "sala").child( pathString: "puerta").addValueEventListener(new ValueEventListener() {
            @Override  2 usages
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                if (dataSnapshot.exists()) {
                    // Obtener y actualizar el valor de 'ha'
                    hi = Integer.parseInt(dataSnapshot.getValue().toString());

                    // Actualizar el TextView y el estado del ToggleButton
                    tp_txt.setText(String.valueOf(hi));
                    toggleButton9.setChecked(hi == 1);
                }
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                // Manejo de errores
            }
        });
        // Listener para manejar los cambios de estado del ToggleButton
        toggleButton9.setOnCheckedChangeListener((buttonView, isChecked) -> {
            // Actualizar la variable 'ha' según el estado del ToggleButton
            hi = isChecked ? 1 : 0;

            // Actualizar el valor de "luz cuarto" en Firebase
            mDatabase.child( pathString: "sala").child( pathString: "puerta").setValue(hi);
        });
    }
}
```

BathActivity

La clase *BathActivity* permite al usuario gestionar y controlar los dispositivos conectados en el baño. Los dispositivos que se controlan desde esta actividad incluyen la luz y el grifo. Al igual que en otras actividades, la *BathActivity* está integrada con Firebase Realtime Database para la

sincronización en tiempo real del estado de los dispositivos, permitiendo a los usuarios controlar los dispositivos desde la aplicación móvil y ver los cambios reflejados instantáneamente.

Componentes principales

- Button:
    - *buttonBack3*: Botón que permite regresar a la actividad anterior cerrando la actual.
- TextViews:
    - *tb_texto*: Muestra el estado actual de la luz del baño.
    - *tgr_txt*: Muestra el estado del grifo del baño.
- ToggleButtons:
    - *toggleButton5*: Controla el estado de la luz del baño (encendida o apagada).
    - *toggleButton6*: Controla el estado del grifo del baño (abierto o cerrado).
- ImageViews:
    - *luz*: Imagen que cambia dependiendo de si la luz está encendida o apagada.
    - *grifo*: Imagen que representa si el grifo está abierto o cerrado.

Flujo de funcionamiento

1. Inicialización de componentes:

En el método *onCreate()*, se inicializan los botones, las vistas de texto, las imágenes y los interruptores (toggle buttons), vinculándolos con sus respectivas vistas en el archivo XML (*activity_bath.xml*). También se inicializa la referencia a Firebase para interactuar con la base de datos en tiempo real.

```
48          buttonBack3 = findViewById(R.id.buttonBack3);
49          tb_texto = (TextView) findViewById(R.id.tb_texto);
50          tgr_txt = (TextView) findViewById(R.id.tgr_txt);
51          mDatabase = FirebaseDatabase.getInstance().getReference();
52          toggleButton5 = findViewById(R.id.toggleButton5);
53          toggleButton6 = findViewById(R.id.toggleButton6);
54          luz = findViewById(R.id.imageView2);
55          grifo = findViewById(R.id.imageView4);
```

2. Control de la luz del baño:

- *toggleButton5* gestiona el estado de la luz del baño. Cuando el usuario cambia el estado del botón, la imagen de la luz cambia entre encendida y apagada.
- Firebase se utiliza para mantener el estado de la luz sincronizado en tiempo real con la base de datos, de modo que cualquier cambio en la luz se refleja instantáneamente en la aplicación.

```
67          toggleButton5.setOnClickListener(new View.OnClickListener() {
68              @Override
69              public void onClick(View v) {
70                  if (he==0) {
71                      luz.setImageResource(R.drawable.foco_apagado);
72                  } else {
73                      luz.setImageResource(R.drawable.foco_encendido);
74                  }
75
76              }
77          });
78
79          mDatabase.child( pathString: "bath").child( pathString: "LuzBath").addValueEventListener(new ValueEventListener() {
80              @Override  2 usages
81              public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
82                  if (dataSnapshot.exists()) {
83                      // Obtener y actualizar el valor de 'ha'
84                      he = Integer.parseInt(dataSnapshot.getValue().toString());
85
86                      // Actualizar el TextView y el estado del ToggleButton
87                      tb_texto.setText(String.valueOf(he));
88                      toggleButton5.setChecked(he == 1);
89                  }
90              }
91
92              @Override
93              public void onCancelled(@NonNull DatabaseError error) {
94                  // Manejo de errores
95              }
96          });
97
98          // Listener para manejar los cambios de estado del ToggleButton
99          toggleButton5.setOnCheckedChangeListener((buttonView, isChecked) -> {
100             // Actualizar la variable 'ha' según el estado del ToggleButton
101             he = isChecked ? 1 : 0;
102
103             // Actualizar el valor de "luz cuarto" en Firebase
104             mDatabase.child( pathString: "bath").child( pathString: "LuzBath").setValue(he);
105         });
```

3. Control del grifo:

- *toggleButton6* controla el estado del grifo del baño. La imagen del grifo cambia para indicar si está abierto o cerrado según el estado del botón.
- El estado del grifo también se mantiene sincronizado con Firebase, lo que permite reflejar cualquier cambio de estado en tiempo real.

```
        toggleButtonó.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (hf==0) {
                    grifo.setImageResource(R.drawable.grifo_cerrado);
                } else {
                    grifo.setImageResource(R.drawable.grifo_abierto);
                }

            }
        });

        mDatabase.child( pathString: "bath").child( pathString: "GrifoBath").addValueEventListener(new ValueEventListener() {
            @Override  2 usages
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                if (dataSnapshot.exists()) {
                    // Obtener y actualizar el valor de 'ha'
                    hf = Integer.parseInt(dataSnapshot.getValue().toString());

                    // Actualizar el TextView y el estado del ToggleButton
                    tgr_txt.setText(String.valueOf(hf));
                    toggleButtonó.setChecked(hf == 1);
                }
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                // Manejo de errores
            }
        });

        // Listener para manejar los cambios de estado del ToggleButton
        toggleButtonó.setOnCheckedChangeListener((buttonView, isChecked) -> {
            // Actualizar la variable 'ha' según el estado del ToggleButton
            hf = isChecked ? 1 : 0;

            // Actualizar el valor de "luz cuarto" en Firebase
            mDatabase.child( pathString: "bath").child( pathString: "GrifoBath").setValue(hf);
        });
```

CocinaActivity

La clase *CocinaActivity* permite al usuario gestionar y controlar los dispositivos conectados en la cocina. Estos dispositivos incluyen la luz y el suministro de gas. Al igual que en otras actividades de la aplicación, *CocinaActivity* está integrada con Firebase Realtime Database para la sincronización en tiempo real del estado de los dispositivos, lo que permite a los usuarios controlar la cocina de manera eficiente y ver los cambios reflejados al instante en la interfaz de la aplicación.

Componentes principales

- Button:
    - *buttonBack*: Botón que permite regresar a la actividad anterior cerrando la actividad actual.
- TextViews:
    - *tc_texto*: Muestra el estado actual de la luz de la cocina.
    - tg_txt: Muestra el estado actual del suministro de gas.
- *ToggleButtons:*
    - *toggleButton3*: Controla el estado de la luz de la cocina (encendida o apagada).
    - *toggleButton4*: Controla el estado del suministro de gas (abierto o cerrado).
- ImageViews:
    - *luz*: Imagen que representa el estado de la luz (encendida o apagada).
    - *gas*: Imagen que representa el estado del suministro de gas (abierto o cerrado).

Flujo de funcionamiento

1. Inicialización de los componentes:

En el método *onCreate()*, se inicializan los botones, las vistas de texto, las imágenes y los interruptores (toggle buttons), vinculándolos con sus respectivas vistas en el archivo XML (*activity_cocina.xml*). También se inicializa la referencia a Firebase para interactuar con la base de datos en tiempo real.

```
        tc_texto = (TextView) findViewById(R.id.tc_texto);
        tg_txt = (TextView) findViewById(R.id.tg_txt);
        mDatabase = FirebaseDatabase.getInstance().getReference();
        toggleButton3 = findViewById(R.id.toggleButton3);
        toggleButton4 = findViewById(R.id.toggleButton4);
        luz = findViewById(R.id.imageView5);
        gas = findViewById(R.id.imageView6);
```

2. Control de la luz de la cocina:

- *toggleButton3* gestiona el estado de la luz de la cocina. Cuando el usuario cambia el estado del botón, la imagen de la luz cambia entre encendida y apagada.
- Firebase se utiliza para mantener el estado de la luz sincronizado en tiempo real con la base de datos. Si se cambia el estado desde otro dispositivo, Firebase actualizará el estado automáticamente en la interfaz de la aplicación.

```
        toggleButton3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (hc==0) {
                    luz.setImageResource(R.drawable.foco_apagado);
                } else {
                    luz.setImageResource(R.drawable.foco_encendido);
                }

            }
        });

        mDatabase.child( pathString: "cocina").child( pathString: "LuzCocina").addValueEventListener(new ValueEventListener() {
            @Override  2 usages
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                if (dataSnapshot.exists()) {
                    // Obtener y actualizar el valor de 'ha'
                    hc = Integer.parseInt(dataSnapshot.getValue().toString());

                    // Actualizar el TextView y el estado del ToggleButton
                    tc_texto.setText(String.valueOf(hc));
                    toggleButton3.setChecked(hc == 1);
                }
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                // Manejo de errores
            }
        });

        // Listener para manejar los cambios de estado del ToggleButton
        toggleButton3.setOnCheckedChangeListener((buttonView, isChecked) -> {
            // Actualizar la variable 'ha' según el estado del ToggleButton
            hc = isChecked ? 1 : 0;

            // Actualizar el valor de "luz cuarto" en Firebase
            mDatabase.child( pathString: "cocina").child( pathString: "LuzCocina").setValue(hc);
        });
```

3. Control del suministro de gas:

- *toggleButton4* gestiona el estado del suministro de gas de la cocina. Cuando el usuario cambia el estado del botón, la imagen cambia para indicar si el gas está abierto o cerrado.
- El estado del gas también se sincroniza en tiempo real con Firebase, de modo que cualquier cambio se actualiza automáticamente en la aplicación.

```
104         toggleButton4.setOnClickListener(new View.OnClickListener() {
105             @Override
106             public void onClick(View v) {
107                 if (hd==0) {
108                     gas.setImageResource(R.drawable.gas_cerrado);
109                 } else {
110                     gas.setImageResource(R.drawable.gas_encendido);
111                 }
112 
113             }
114         });
115 
116         mDatabase.child( pathString: "cocina").child( pathString: "GasCocina").addValueEventListener(new ValueEventListener() {
117             @Override  2 usages
118             public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
119                 if (dataSnapshot.exists()) {
120                     // Obtener y actualizar el valor de 'ha'
121                     hd = Integer.parseInt(dataSnapshot.getValue().toString());
122 
123                     // Actualizar el TextView y el estado del ToggleButton
124                     tg_txt.setText(String.valueOf(hd));
125                     toggleButton4.setChecked(hd == 1);
126                 }
127             }
128 
129             @Override
130             public void onCancelled(@NonNull DatabaseError error) {
131                 // Manejo de errores
132             }
133         });
134 
135         // Listener para manejar los cambios de estado del ToggleButton
136         toggleButton4.setOnCheckedChangeListener((buttonView, isChecked) -> {
137             // Actualizar la variable 'ha' según el estado del ToggleButton
138             hd = isChecked ? 1 : 0;
139 
140             // Actualizar el valor de "luz cuarto" en Firebase
141             mDatabase.child( pathString: "cocina").child( pathString: "GasCocina").setValue(hd);
142         });
143     }
```

CuartoActivity

La clase *CuartoActivity* permite a los usuarios gestionar los dispositivos conectados en el dormitorio de su hogar inteligente. Esta actividad incluye la capacidad de controlar el estado de la luz y la ventana de la habitación, sincronizando estos dispositivos con Firebase Realtime Database en tiempo real. Además, la clase tiene implementada una función para enviar notificaciones a los usuarios en caso de cambios importantes.

Componentes principales

- Button:
    - *buttonBack*: Botón que cierra la actividad y regresa a la pantalla anterior.
- TextViews:
    - *th_texto*: Muestra el estado de la luz del cuarto.
    - *ts_txt*: Muestra el estado de la ventana del cuarto.
- ToggleButtons:
    - *toggleButton1*: Controla el estado de la luz del dormitorio (encendida o apagada).
    - *toggleButton2*: Controla el estado de la ventana del dormitorio (abierta o cerrada).
- ImageViews:
    - *focoApagado*: Imagen que cambia dependiendo del estado de la luz.
    - *windowclosed*: Imagen que representa el estado de la ventana (abierta o cerrada).
- Firebase Realtime Database:
    - *mDatabase*: Es una instancia de Firebase Realtime Database que se utiliza para sincronizar el estado de los dispositivos en tiempo real.

Flujo de funcionamiento

1. Inicialización de componentes:

En el método *onCreate()*, se inicializan los botones, las vistas de texto y las imágenes de la interfaz de usuario, junto con los *ToggleButtons* que permiten el control de los dispositivos. También se establece la referencia a Firebase para interactuar con la base de datos en tiempo real.

```
57          th_texto = (TextView) findViewById(R.id.th_texto);
58          ts_txt = (TextView) findViewById(R.id.ts_txt);
59          mDatabase = FirebaseDatabase.getInstance().getReference();
60          toggleButton1 = findViewById(R.id.toggleButton1);
61          toggleButton2 = findViewById(R.id.toggleButton2);
62          focoApagado = findViewById(R.id.FocoApagado);
63          windowclosed = findViewById(R.id.WindowClosed);
```

2. Control de la luz del cuarto:

- *toggleButton1* se utiliza para gestionar el estado de la luz del cuarto. Al cambiar el estado del botón, la imagen de la luz se actualiza entre "apagada" y "encendida".
- *Firebase* se utiliza para sincronizar el estado de la luz con la base de datos en tiempo real, de modo que los cambios se reflejen en todos los dispositivos conectados.

```
        toggleButton1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (ha==0) {
                    focoApagado.setImageResource(R.drawable.foco_apagado);
                } else {
                    focoApagado.setImageResource(R.drawable.foco_encendido);
                }

            }
        });

        mDatabase.child( pathString: "dormitorio").child( pathString: "LuzCuarto").addValueEventListener(new ValueEventListener() {
            @Override  2 usages
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                if (dataSnapshot.exists()) {
                    // Obtener y actualizar el valor de 'ha'
                    ha = Integer.parseInt(dataSnapshot.getValue().toString());

                    // Actualizar el TextView y el estado del ToggleButton
                    th_texto.setText(String.valueOf(ha));
                    toggleButton1.setChecked(ha == 1);
                }
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                // Manejo de errores
            }
        });

        // Listener para manejar los cambios de estado del ToggleButton
        toggleButton1.setOnCheckedChangeListener((buttonView, isChecked) -> {
            // Actualizar la variable 'ha' según el estado del ToggleButton
            ha = isChecked ? 1 : 0;

            // Actualizar el valor de "luz cuarto" en Firebase
            mDatabase.child( pathString: "dormitorio").child( pathString: "LuzCuarto").setValue(ha);
        });
```

3. Control de la ventana del cuarto:

- *toggleButton2* controla el estado de la ventana del dormitorio, alternando entre "cerrada" y "abierta" visualmente mediante un cambio en la imagen.
- Al igual que con la luz, el estado de la ventana se sincroniza en tiempo real con Firebase para garantizar que los cambios se reflejen en todos los dispositivos.

```
        toggleButton2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (hb==0) {
                    windowclosed.setImageResource(R.drawable.closed_window);
                } else {
                    windowclosed.setImageResource(R.drawable.open_window);
                }

            }
        });

        mDatabase.child( pathString: "dormitorio").child( pathString: "VentanaCuarto").addValueEventListener(new ValueEventListener() {
            @Override  2 usages
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                if (dataSnapshot.exists()) {
                    // Obtener y actualizar el valor de 'ha'
                    hb = Integer.parseInt(dataSnapshot.getValue().toString());

                    // Actualizar el TextView y el estado del ToggleButton
                    ts_txt.setText(String.valueOf(hb));
                    toggleButton2.setChecked(hb == 1);
                }
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                // Manejo de errores
            }
        });

        // Listener para manejar los cambios de estado del ToggleButton
        toggleButton2.setOnCheckedChangeListener((buttonView, isChecked) -> {
            // Actualizar la variable 'ha' según el estado del ToggleButton
            hb = isChecked ? 1 : 0;

            // Actualizar el valor de "luz cuarto" en Firebase
            mDatabase.child( pathString: "dormitorio").child( pathString: "VentanaCuarto").setValue(hb);
        });
```

4. Notificaciones:

La clase también incluye un método llamado *sendNotification()* que permite enviar notificaciones al usuario cuando ocurren eventos importantes, como cambios en el estado de los dispositivos. Las notificaciones son compatibles con versiones de Android 8.0 (Oreo) en adelante, mediante la creación de un *NotificationChannel* si es necesario

```
    private void sendNotification(String title, String message) {  no usages
        NotificationManager notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

        // Verifica si se necesita crear un canal de notificación para versiones de Android 8.0 y superiores
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            NotificationChannel channel = new NotificationChannel( id: "default_channel_id",  name: "Default Channel", NotificationManager.IMPORTANCE_DEFAULT);
            notificationManager.createNotificationChannel(channel);
        }

        // Crea una notificación
        NotificationCompat.Builder builder = new NotificationCompat.Builder( context: this,  channelId: "default_channel_id")
                .setSmallIcon(R.drawable.smth)
                .setContentTitle(title)
                .setContentText(message)
                .setPriority(NotificationCompat.PRIORITY_DEFAULT);

        // Muestra la notificación
        notificationManager.notify( id: 1, builder.build());
    }

}
```

Directorio *res*

Este directorio contiene todos los recursos utilizados por la aplicación, tales como imágenes, layouts, valores y otros elementos gráficos:

- ***drawable/:*** Almacena los recursos gráficos como íconos e imágenes.
- ***layout/*:** Contiene los archivos XML que definen el diseño visual de cada actividad, como *activity_login.xml, activity_cocina.xml, activity_main.xml,* entre otros. Estos archivos son esenciales para construir las interfaces de usuario, asegurando que cada pantalla tenga la disposición correcta de botones, textos y otros elementos visuales.

Scripts de Gradle

Los scripts de **Gradle** son fundamentales para gestionar las dependencias y la configuración del proyecto. Estos archivos controlan cómo se compila y ejecuta la aplicación, permitiendo la integración con bibliotecas externas como Firebase, y la correcta sincronización de las herramientas necesarias.

- **build.gradle** (módulo y proyecto): Define las dependencias del proyecto, incluyendo librerías externas como Firebase, y otros ajustes específicos del entorno de desarrollo.

Base de Datos - Firebase Realtime Database

La base de datos es uno de los componentes fundamentales del backend de la aplicación SmartHome. En este caso, la aplicación utiliza Firebase Realtime Database como la principal herramienta para almacenar y gestionar los datos en tiempo real. Firebase permite sincronizar los datos entre dispositivos de los usuarios y mantener actualizada la información del estado de los dispositivos inteligentes en una estructura jerárquica.

Estructura General de la Base de Datos

La base de datos está organizada jerárquicamente bajo un nodo principal llamado *Casa*, el cual contiene subnodos que representan las diferentes habitaciones del hogar inteligente. Cada habitación tiene a su vez subnodos que corresponden a los dispositivos conectados en esa habitación, tales como luces, grifos y ventanas. Estos dispositivos almacenan un valor binario (0 o 1) que indica si están apagados/cerrados o encendidos/abiertos.

Ejemplo de la estructura:

- Baño: Contiene los dispositivos *grifo2* y *luz4*.

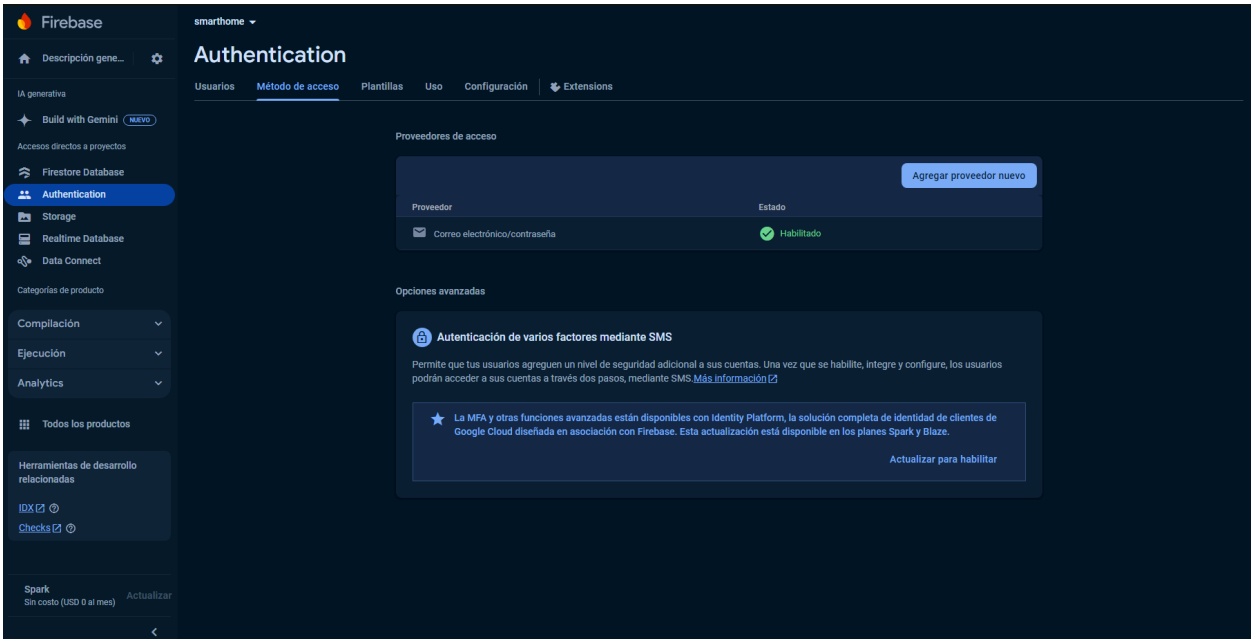
- Cocina: Contiene los dispositivos *grifo1* y *luz3*.
- Cuarto: Tiene los dispositivos *luz2* y *ventana1*.
- Sala: Controla *luz1* y *puerta*.

Cada uno de estos dispositivos tiene un valor binario que cambia en función de su estado actual (0 para apagado/cerrado y 1 para encendido/abierto). Esta estructura permite controlar de manera individual los dispositivos conectados en cada habitación.

Autenticación y Seguridad

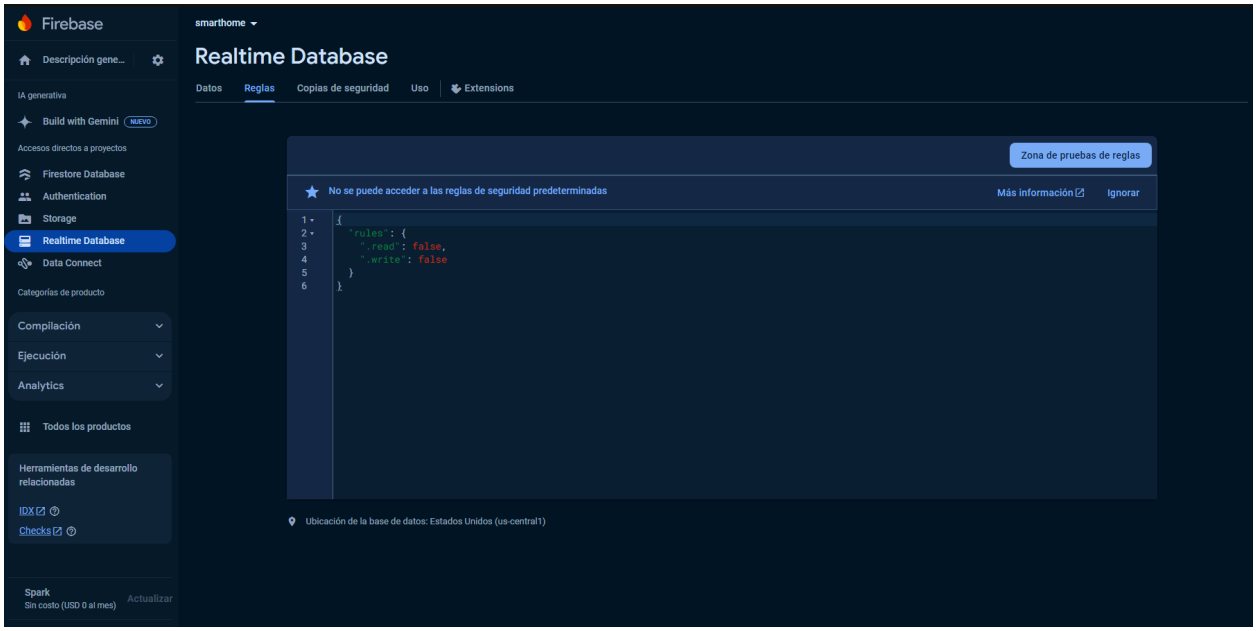
El sistema de autenticación en Firebase garantiza que solo los usuarios autenticados puedan interactuar con la base de datos. En este caso, Firebase Authentication ha sido configurado para utilizar correo electrónico y contraseña como método de acceso. Esto asegura que cada usuario tenga una cuenta única y autenticada antes de interactuar con los dispositivos de su hogar inteligente.

En la imagen mostrada, podemos ver que se ha habilitado el método de autenticación por correo electrónico y contraseña. Cada usuario necesita autenticar su identidad para poder acceder al estado de sus dispositivos o cambiar su configuración.



Reglas de Seguridad en la Base de Datos

En la configuración actual, las reglas de seguridad para la base de datos están configuradas para que no se permitan lecturas ni escrituras desde la base de datos por ningún usuario no autenticado:



Estas reglas básicas bloquean el acceso a la base de datos tanto para lectura como para escritura, lo cual protege la información y evita que usuarios no autorizados accedan a los datos. Sin embargo, en una implementación final, se recomienda configurar las reglas para que solo los usuarios autenticados puedan leer y escribir en los datos relacionados con sus dispositivos.

Sincronización en Tiempo Real

Uno de los principales beneficios de utilizar Firebase Realtime Database es la capacidad de sincronizar los datos en tiempo real. Esto significa que cada vez que un dispositivo cambia de estado (por ejemplo, cuando se enciende una luz o se abre una ventana), el cambio se refleja instantáneamente en la base de datos, y cualquier usuario conectado recibe la actualización de

inmediato. Esta característica es especialmente útil en aplicaciones como SmartHome, donde es necesario monitorear y controlar dispositivos en tiempo real.

Interacción con el Backend

El backend de la aplicación SmartHome se encarga de gestionar la interacción entre los dispositivos controlados desde la interfaz de usuario y la base de datos en Firebase. Cada vez que el usuario realiza una acción (por ejemplo, encender una luz), el backend escribe el nuevo estado en Firebase, lo que a su vez actualiza la interfaz de usuario de todos los dispositivos conectados.

Los dispositivos en la base de datos se gestionan a través de *ToggleButtons* en la interfaz de usuario, que están vinculados a los datos de Firebase. Por ejemplo, cuando un usuario enciende o apaga una luz, el backend envía el nuevo estado al nodo correspondiente en la base de datos, actualizando el valor del dispositivo en tiempo real.

Conclusiones

El desarrollo del backend para la aplicación SmartHome ha sido un proceso integral que combina tecnologías robustas y bien establecidas para asegurar un sistema eficiente, seguro y escalable. A lo largo de este manual, se ha detallado cada componente esencial, desde la estructura de la base de datos hasta la implementación de la autenticación y el control en tiempo real de los dispositivos.

Utilizando Java como lenguaje base y Firebase como servicio backend, SmartHome garantiza que los usuarios puedan interactuar de forma intuitiva y segura con los dispositivos conectados de su hogar. Las funcionalidades de sincronización en tiempo real, autenticación segura y la integración con servicios en la nube han sido implementadas para proporcionar una experiencia de usuario fluida y confiable.

La organización modular del código y la clara separación de responsabilidades dentro de la arquitectura aseguran que el sistema sea mantenible y escalable. Esto permite que la aplicación no solo cumpla con los requisitos actuales, sino que también esté preparada para futuras expansiones y actualizaciones.

Referencias

Firebase. (n.d.). *Firebase documentation*. Google. https://firebase.google.com/docs

Oracle. (n.d.). *The Java tutorials*. Oracle. https://docs.oracle.com/javase/tutorial/

Android Studio. (n.d.). *Developer guide*. Android Developers. https://developer.android.com/studio