Manual Técnico de Aether

UNIVERSIDAD
NACIONAL
DE COLOMBIA
SEDE BOGOTÁ

Integrantes:

Juan Pablo Avendaño Acero

Diego Jose Navarro Lopez

Entregado a:
Nestor German Bolivar Pulgarin

2023 - 2

Índice

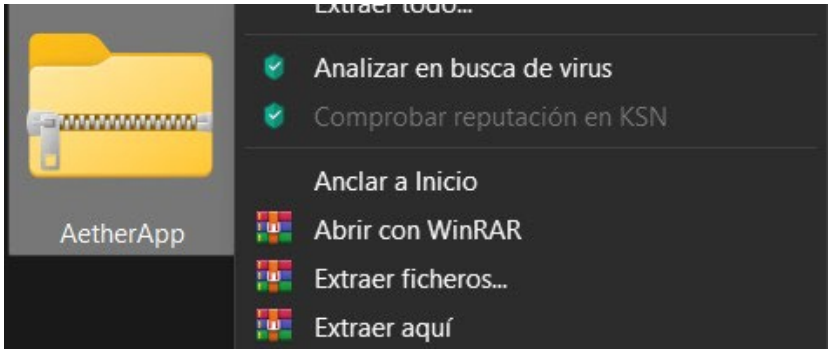
Manual Técnico

Instalación

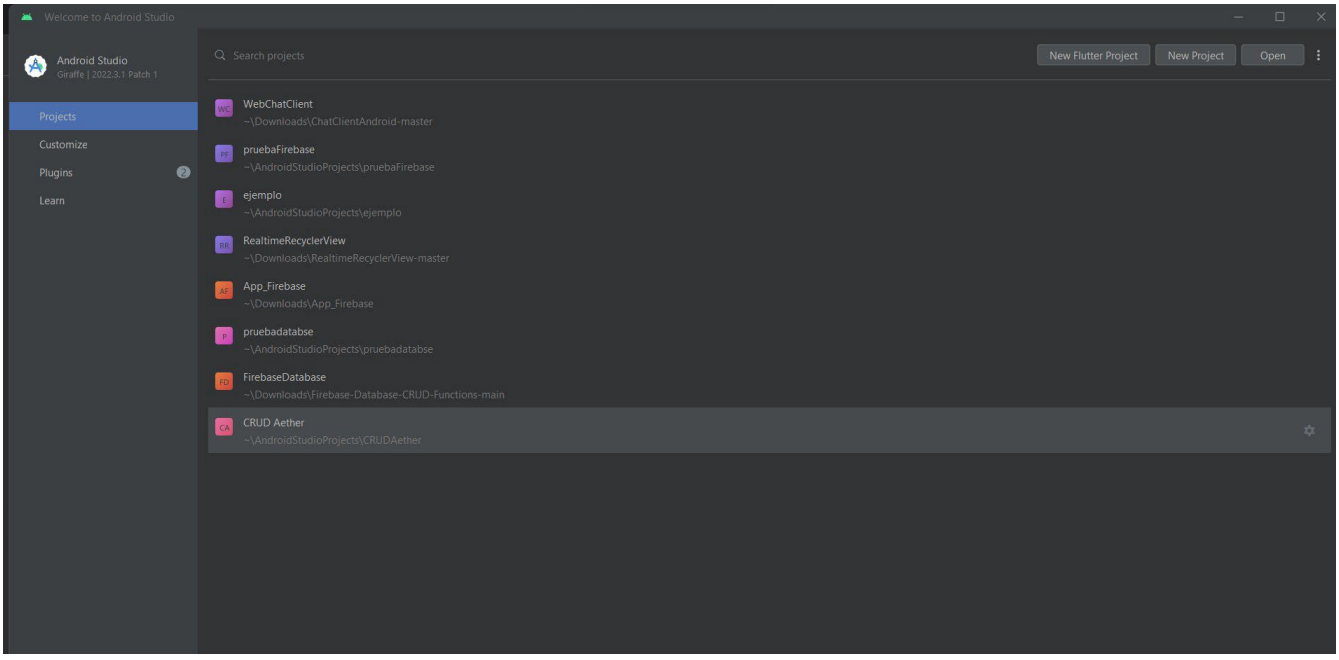
Para poder usar la aplicación de Aether es necesario instalar Android Studio en un PC y descargar el archivo AetherApp.zip, descomprimirlo y posteriormente abrirlo en el IDE:

Link repositorio del archivo: https://github.com/Jp101466/Aether-files

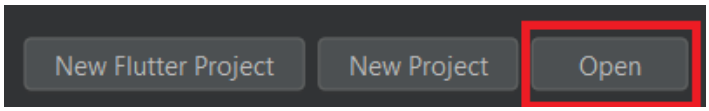
1. Descargar, guardar y descomprimir el archivo.

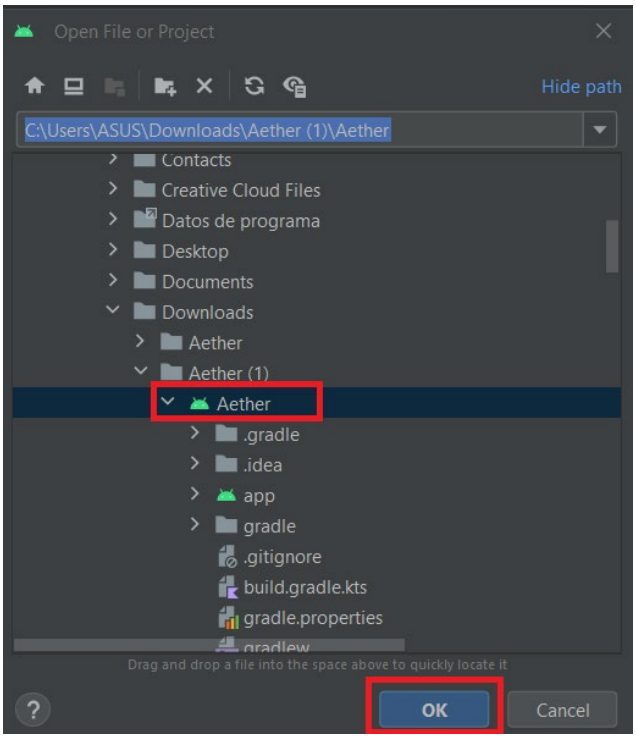


2. Abrir Android Studio.



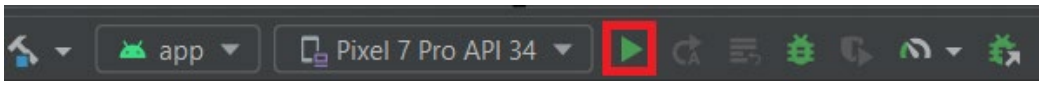
3. Click en "Open", buscar el archivo y abrir.



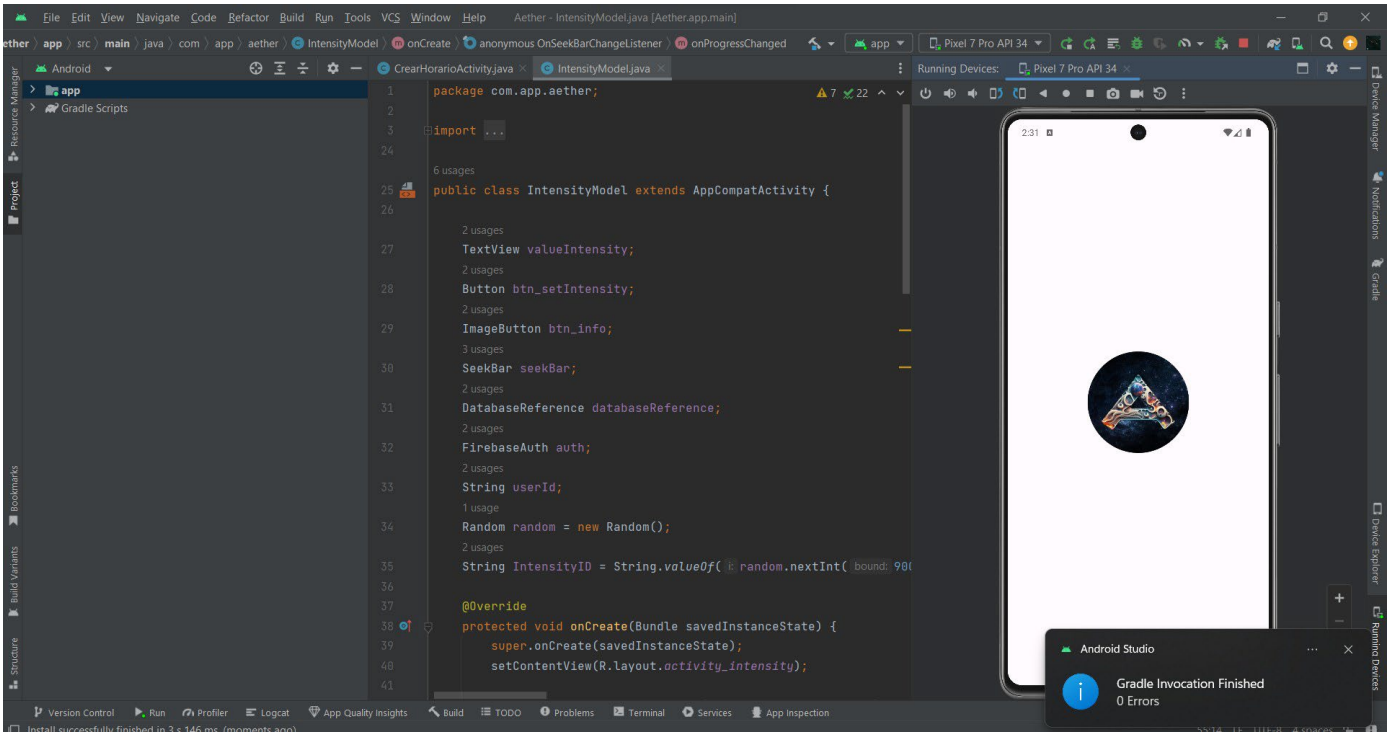


Posterior a ello, dentro del programa damos al botón de "play" y así se ejecutará la aplicación dentro de un emulador (también es posible conectar el dispositivo móvil vía USB al PC y el programa se ejecutará en el dispositivo), también es necesario tener el wifi activado y conceder los permisos de notificaciones a la aplicación para su correcto funcionamiento).

4. Ejecución de la aplicación:



Vista del código y el emulador:



Estructura en Android Studio

Dentro del IDE encontraremos que la aplicación (en la vista Android) está compuesta por la carpeta principal "app" donde dentro de ella se hallan las carpetas "manifests", "java", "res" y "Gradle Scripts", dentro de estas carpetas se encuentran más subcarpetas que contienen archivos y código cuan importantes como necesarios, que definen el funcionamiento y comportamiento de la aplicación.

Manifests

Esta carpeta contiene un archivo llamado "AndroidManifest.xml" en este archivo encontramos diversos elementos como "<uses-permission>", este se utiliza para declarar los permisos que la aplicación necesita para acceder a ciertas funcionalidades del sistema operativo Android. En este caso para notificaciones (POST_NOTIFICATIONS) y acceso a internet (INTERNET). Configuración de la aplicación, el elemento <application> contiene configuraciones generales como:

android:allowBackup: Indica si la aplicación permite o no realizar copias de seguridad, android:icon y android:roundIcon: Especifican los íconos de la aplicación. android:label: Define el nombre de la aplicación. android:theme: Establece el tema visual de la aplicación. Actividades, El archivo AndroidManifest.xml enumera todas las actividades (pantallas o componentes de la interfaz de usuario) que componen la aplicación. Cada interfaz "<activity>" tiene un nombre (android:name) que especifica la clase de Java asociada con esa actividad.

Estructura de AndroidManifest.xml:

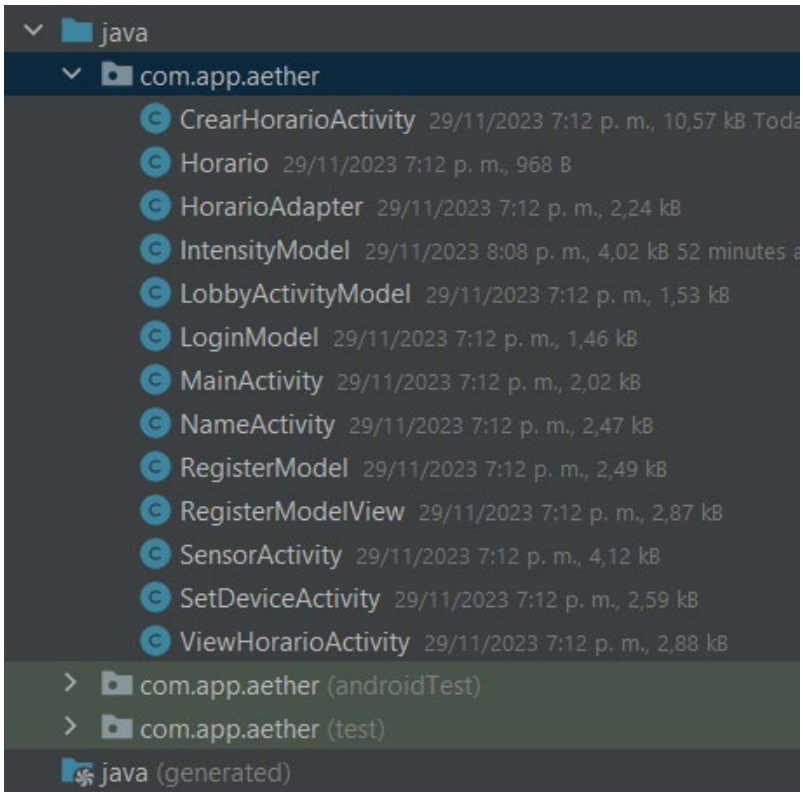
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Aether"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Aether"
        tools:targetApi="31">
        <activity
            android:name=".IntensityModel"
            android:exported="false" />
        <activity
            android:name=".SensorActivity"
            android:exported="false" />
        <activity
            android:name=".NameActivity"
            android:exported="false" />
        <activity
            android:name=".SetDeviceActivity"
```

Carpeta java.

Dentro de esta carpeta encontramos 3 subcarpetas, "com.app.aether", "com.app.aether (android test)" y "com.app.aether (test)", siendo la más importante la primera, en ella encontramos apectos como el almacenamiento de Código Fuente en Java, esta carpeta es el lugar donde se almacena el código fuente de la aplicación escrita en el lenguaje de

programación Java. Este es el código que define la lógica de la aplicación, incluyendo la implementación de actividades y cualquier otra clase Java necesaria para el funcionamiento de la aplicación por otra parte tenemos la organización por Paquetes, si así se desea, podemos clasificar las clases en paquetes. Estos paquetes ayudan a estructurar y organizar el código de manera lógica y modular.

Estructura de la carpeta:



Carpeta res.

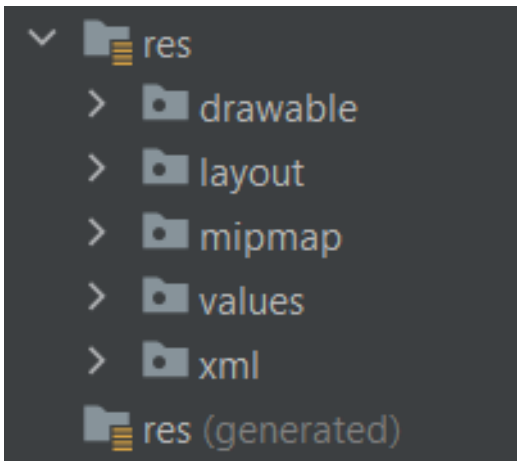
La carpeta "res" en Android Studio juega un papel crucial en la organización de recursos de una aplicación Android, como abreviatura de "resources" esta carpeta almacena diversos tipos de recursos que son utilizados por la aplicación, como gráficos, diseños, valores de cadena, estilos y otros elementos. Detalladamente tenemos las siguientes 5 subcarpetas:

- drawable: Contiene archivos de imágenes y gráficos utilizados en la interfaz

de usuario.

- layout: Almacena archivos XML que definen la estructura y diseño de las pantallas de la aplicación.
- values: Contiene archivos XML con valores, como cadenas de texto, dimensiones, colores y estilos.
- xml: Guarda archivos XML que definen animaciones.
- mipmap: Contiene íconos de la aplicación en diferentes densidades de píxeles.

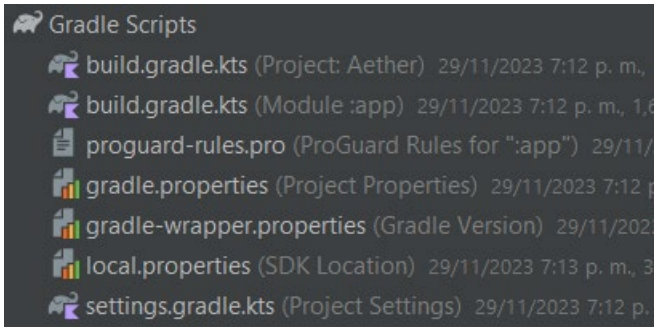
Vista de la estructura de la carpeta:



Carpeta Gradle Scripts:

Dentro de esta carpeta se encuentran varios archivos, siendo los más importantes los “build.gradle.kts”, ya que estos archivos definen la configuración global del proyecto, como la configuración del repositorio de Maven, versiones de plugins, dependencias y otras configuraciones a nivel de proyecto.

Vista de la estructura de la carpeta:



Archivo JSON

Es un archivo de configuración que proporciona información clave a la aplicación para que pueda conectarse y comunicarse con los servicios de Firebase. Este archivo es esencial para la inicialización y la autenticación de la aplicación con Firebase, los elementos clave presentes en el archivo JSON de Firebase son:

- apiKey: La clave API es utilizada para autenticar solicitudes a los servicios de Firebase, como Firebase Authentication y Firebase Realtime Database.
- authDomain: Indica el dominio de autenticación que Firebase utilizará para autenticar usuarios.
- projectId: Identifica el proyecto de Firebase asociado con la aplicación. Cada proyecto de Firebase tiene un ID único.
- storageBucket: Especifica el cubo de almacenamiento de Firebase que se utilizará para almacenar y recuperar archivos desde Firebase Storage.
- messagingSenderId: Se utiliza para la configuración de Firebase Cloud Messaging (FCM) y está asociado con el ID del proyecto.
- appId: Es el ID de la aplicación específica y es necesario para la inicialización de Firebase en la aplicación.
- measurementId (opcional): Se utiliza para la integración con Google

Analytics para Firebase. Esta clave es opcional y se puede omitir si se está utilizando Google Analytics.

Estructura del archivo JSON:

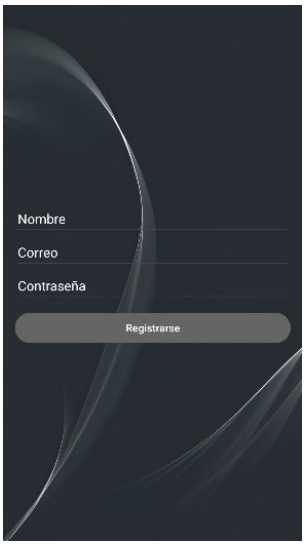
FrontEnd

Los apartados que componen al frontend son los archivos que residen en la carpeta res, cuyas funciones fueron explicadas anteriormente, sin

embargo, el apartado más importante que permite desarrollar la experiencia de usuario es la subcarpeta Layout, donde junto con sus respectivas clases, complementan el apartado visual y estético de la aplicación, dentro de Layout encontramos una serie de xml los cuales contienen la interfaz gráfica para cada apartado de la aplicación:

- activity_main.xml, es la primera ventana establecida para la aplicación, donde se le permite al usuario registrarse o iniciar sesión en la aplicación para acceder a sus servicios.

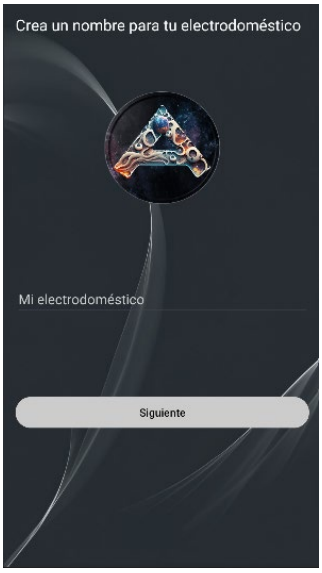
- activity_register.xml, en esta ventana el usuario puede crear su cuenta para usar la aplicación:



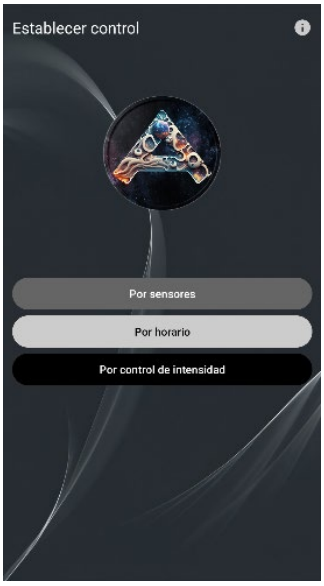
- activity_lobby.xml: En esta actividad el usuario puede ver las

opciones que le brinda la aplicación, agregar un electrodoméstico, ver sus horarios creados o cerrar sesión.

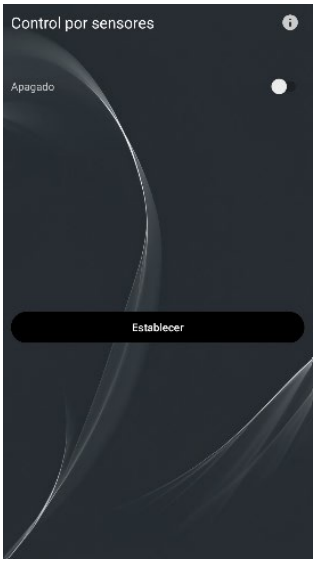
- activity_name: Si el usuario decide agregar un electrodoméstico, será dirigido a esta ventana, en la cual se le pide únicamente ingresar el nombre con el que quiere identificar a su electrodoméstico.



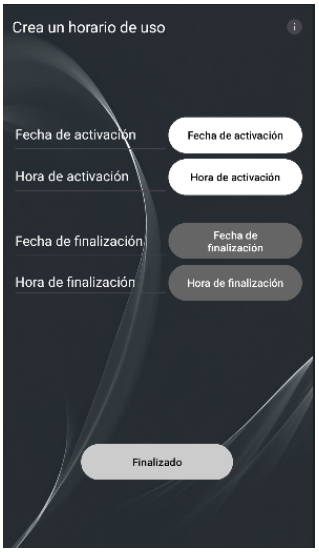
- activity_set_device: Después el usuario será dirigido a esta ventana, donde podrá seleccionar la opción más conveniente que le parezca para controlar su electrodoméstico.



- activity_sensor: Si el usuario escoge el control por sensores, será llevado a esta ventana, donde puede encender o apagar el electrodoméstico de forma manual por medio de la aplicación.

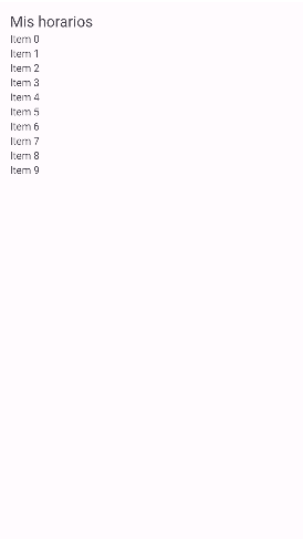


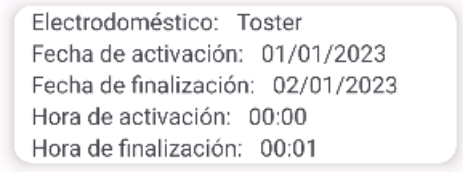
- activity_crear_horario: Si el usuario escoge el control por horarios, será llevado a esta ventana, donde puede encender o apagar el electrodoméstico de forma automática por medio de la aplicación.



- activity_intensity: Si el usuario escoge el control por intensidad, será llevado a esta ventana, donde puede controlar el flujo de corriente (apto solo para la iluminación).

- activity_view_horario e item: Si el usuario escoge en activity_lobby ver sus horarios, será llevado a esta ventana, donde puede los horarios creados para cada electrodoméstico, por su parte, item es el formato que adapta activity_view_horario para mostrar los horarios.





BackEnd

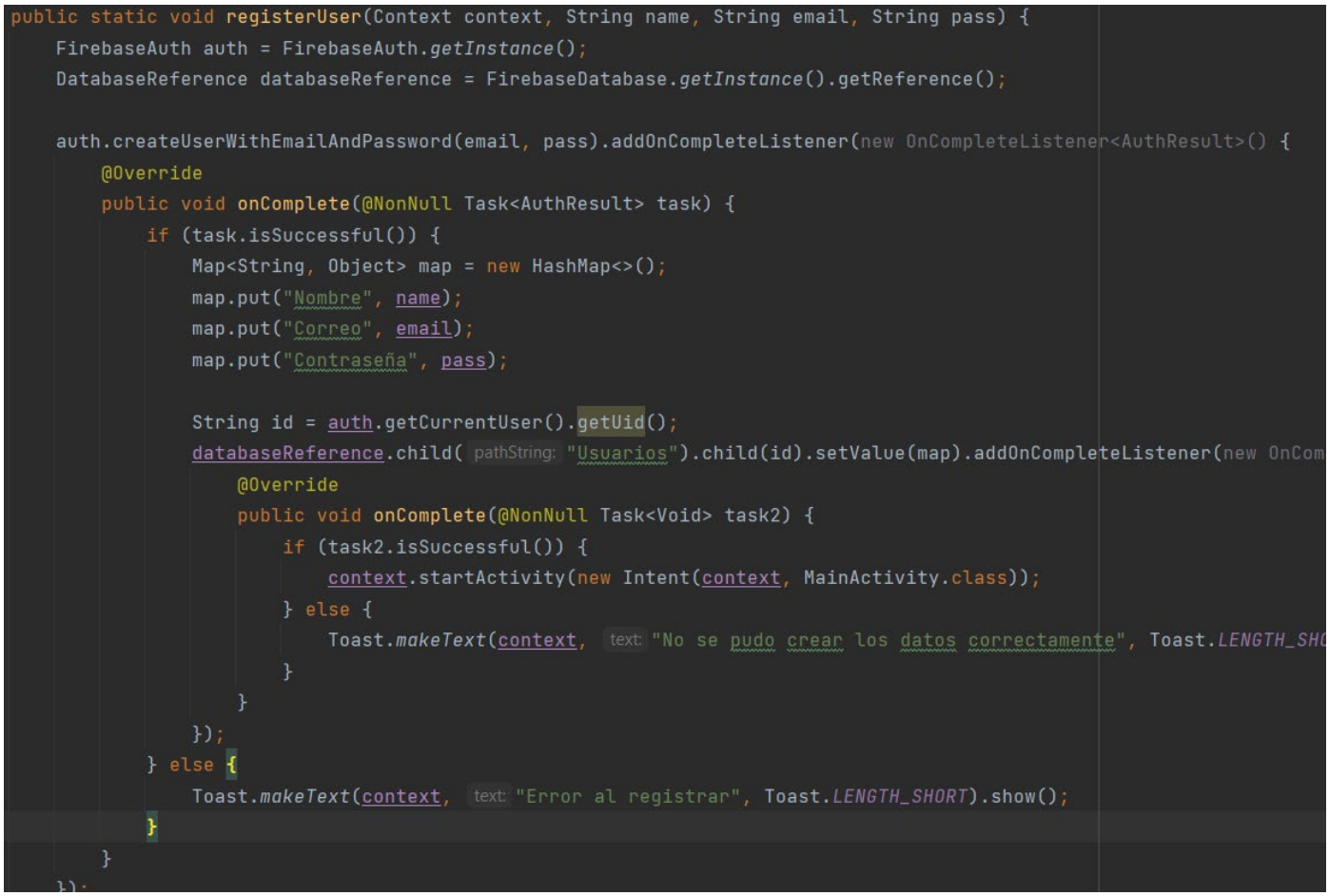
Para el manejo de datos del usuario se creó una conexión en Firebase para establecer una base de datos y así mantener la información.

Registro de usuario:

Para ello se creó el método registerUser dentro de la clase RegisterModel, este se encarga de registrar un nuevo usuario en Firebase Authentication y almacenar información adicional sobre ese usuario en la base de datos de Firebase.

Primero se utiliza el método createUserWithEmailAndPassword de FirebaseAuth para crear un nuevo usuario con el correo electrónico y la contraseña proporcionados luego, se verifica si la creación del usuario fue exitosa. Si es exitosa, se procede a almacenar información adicional en la base de datos. despues se crea un mapa (map) que contiene la información adicional que se desea almacenar en la base

de datos como el nombre, correo electrónico y contraseña del usuario, finalmente se verifica si el almacenamiento de datos en la base de datos fue exitoso. Si es así, se inicia una nueva actividad (MainActivity). Si no es exitoso, se muestra un mensaje de error.



```
public static void registerUser(Context context, String name, String email, String pass) {
    FirebaseAuth auth = FirebaseAuth.getInstance();
    DatabaseReference databaseReference = FirebaseDatabase.getInstance().getReference();

    auth.createUserWithEmailAndPassword(email, pass).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
        @Override
        public void onComplete(@NonNull Task<AuthResult> task) {
            if (task.isSuccessful()) {
                Map<String, Object> map = new HashMap<>();
                map.put("Nombre", name);
                map.put("Correo", email);
                map.put("Contraseña", pass);

                String id = auth.getCurrentUser().getUid();
                databaseReference.child( pathString: "Usuarios").child(id).setValue(map).addOnCompleteListener(new OnCom
                    @Override
                    public void onComplete(@NonNull Task<Void> task2) {
                        if (task2.isSuccessful()) {
                            context.startActivity(new Intent(context, MainActivity.class));
                        } else {
                            Toast.makeText(context,  text: "No se pudo crear los datos correctamente", Toast.LENGTH_SHO
                        }
                    }
                });
            } else {
                Toast.makeText(context,  text: "Error al registrar", Toast.LENGTH_SHORT).show();
            }
        }
    });
```

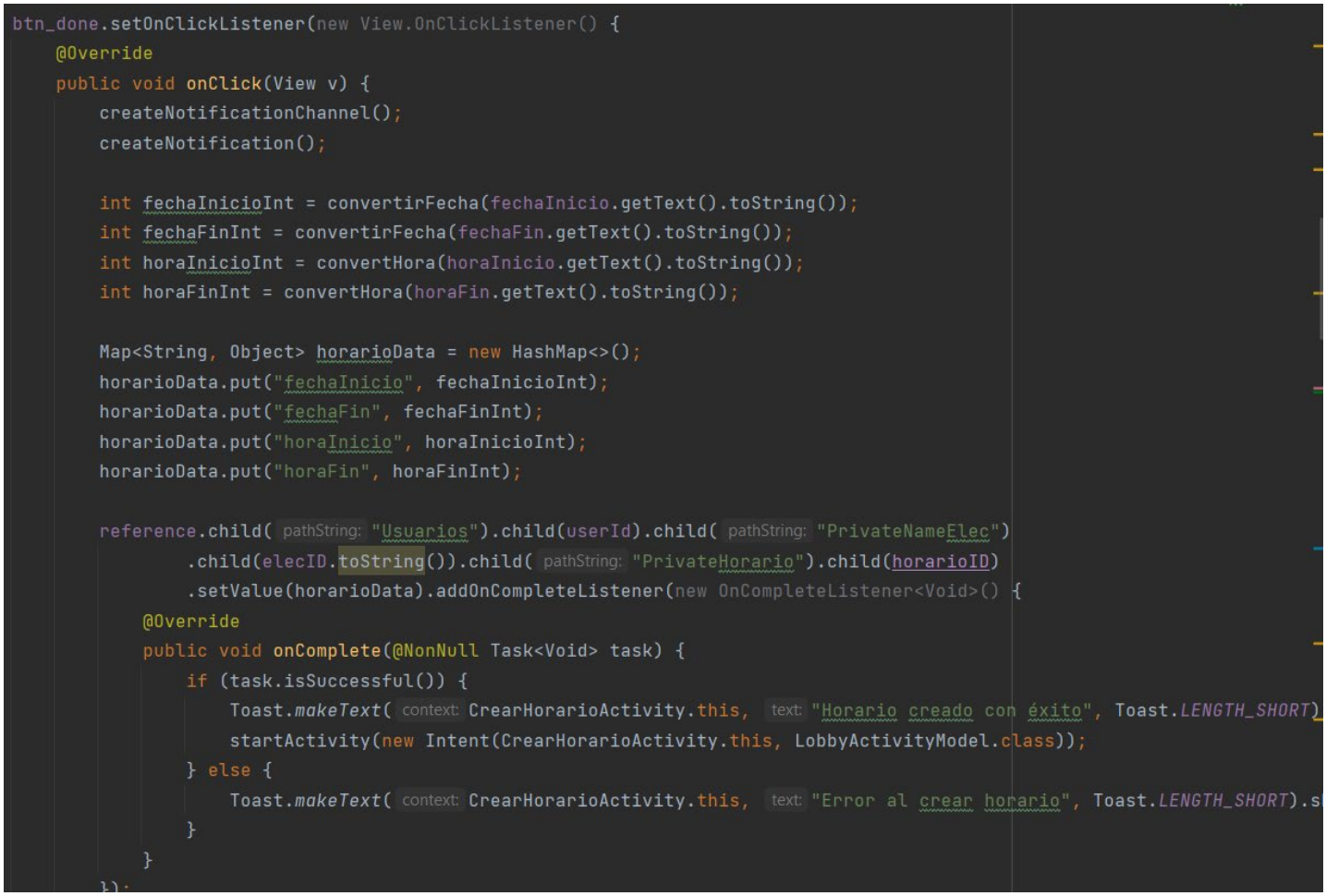
Para la autenticación de usuarios tenemos el método loginUser de la clase LoginModel se utiliza el método signInWithEmailAndPassword de FirebaseAuth para intentar iniciar sesión con el correo electrónico y la contraseña proporcionados, luego se utiliza un OnCompleteListener para verificar si el inicio de sesión fue exitoso (task.isSuccessful()). En caso de éxito, se inicia una nueva actividad (LobbyActivityModel) y se muestra un mensaje de éxito, finalmente Se utiliza un OnFailureListener para manejar los casos en que el inicio de sesión no es exitoso. En este caso, se muestra un mensaje indicando que el usuario o la contraseña son incorrectos.

```java
public static void loginUser(Context context, String email, String pass) {
    FirebaseAuth mAuth = FirebaseAuth.getInstance();

    mAuth.signInWithEmailAndPassword(email, pass).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
        @Override
        public void onComplete(@NonNull Task<AuthResult> task) {
            if (task.isSuccessful()) {
                // Ingreso exitoso, redirigir a la actividad deseada
                Intent intent = new Intent(context, LobbyActivityModel.class);
                context.startActivity(intent);
                Toast.makeText(context,  text: "¡Ingreso exitoso!", Toast.LENGTH_SHORT).show();
            }
        }
    }).addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Toast.makeText(context,  text: "Usuario o contraseña incorrectos", Toast.LENGTH_SHORT).show();
        }
    });
}
```

Para la creación del electrodoméstico configuramos el evento del botón “btn_savename”, cuando este es presionado, se ejecuta el código dentro del método onClick(View v) del OnClickListener, en ello, Se crea un HashMap llamado nameInfo que contendrá la información del nombre a ser guardado. En este caso, el nombre se extrae de un objeto EditText llamado nombreElec. Después se utiliza un objeto de referencia (reference) a la base de datos de Firebase para almacenar la información del nombre en una definida dentro de una colección, finalmente se utiliza un OnCompleteListener para verificar si la operación de almacenamiento en la base de datos fue exitosa. Si es así, se muestra un mensaje de éxito y se inicia una nueva actividad (SetDeviceActivity), si no, se muestra un mensaje de error.

```java
btn_savename.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        Map<String, Object> nameInfo = new HashMap<>();
        nameInfo.put("Nombre", nombreElec.getText().toString());

        reference.child( pathString: "Usuarios").child(userId).child( pathString: "PrivateNameElec")
                .child(elecID).setValue(nameInfo).addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                if (task.isSuccessful()) {
                    Toast.makeText( context: NameActivity.this,  text: "Nombre guardado con éxito", Toast.LENGTH_SHORT).show();
                    startActivity(new Intent(NameActivity.this, SetDeviceActivity.class));
                } else {
                    Toast.makeText( context: NameActivity.this,  text: "Error al guardar nombre", Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
});
```

Para guardar la información de horarios se maneja el evento de clic en el botón “btn_done”. Cuando el botón es presionado, se ejecuta el código dentro del método onClick(View v) del OnClickListener, en ello, Se llama al método createNotificationChannel(). Este método se encarga de crear un canal de notificación para crear y mostrar una notificación en el sistema de los datos que fueron creados, luego se convierten las fechas y horas ingresadas en EditText a valores enteros utilizando los métodos convertirFecha() y convertHora(), luego se crea un HashMap llamado horarioData que contiene la información del horario, incluyendo las fechas y horas convertidas a enteros, finalmente se utiliza un objeto de referencia (reference) a la base de datos de Firebase para almacenar la información del horario en una ubicación específica.



```
btn_done.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        createNotificationChannel();
        createNotification();

        int fechaInicioInt = convertirFecha(fechaInicio.getText().toString());
        int fechaFinInt = convertirFecha(fechaFin.getText().toString());
        int horaInicioInt = convertHora(horaInicio.getText().toString());
        int horaFinInt = convertHora(horaFin.getText().toString());

        Map<String, Object> horarioData = new HashMap<>();
        horarioData.put("fechaInicio", fechaInicioInt);
        horarioData.put("fechaFin", fechaFinInt);
        horarioData.put("horaInicio", horaInicioInt);
        horarioData.put("horaFin", horaFinInt);

        reference.child( pathString: "Usuarios").child(userId).child( pathString: "PrivateNameElec")
                .child(elecID.toString()).child( pathString: "PrivateHorario").child(horarioID)
                .setValue(horarioData).addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                if (task.isSuccessful()) {
                    Toast.makeText( context: CrearHorarioActivity.this,  text: "Horario creado con éxito", Toast.LENGTH_SHORT)
                    startActivity(new Intent(CrearHorarioActivity.this, LobbyActivityModel.class));
                } else {
                    Toast.makeText( context: CrearHorarioActivity.this,  text: "Error al crear horario", Toast.LENGTH_SHORT).s
                }
            }
        });
```

Para que el usuario vea sus horarios creados, se debe rescatar la información subida a Firebase para mostrarla en pantalla, para ello en la clase viewHorarioActivity se define un recyclerView, un componente de interfaz de usuario utilizado para mostrar una lista de elementos con la posibilidad de desplazarse, DatabaseReference que apunta a la ubicación en la base de datos de Firebase donde se almacenan los horarios ("PrivateHorario"), myAdapter: Es una instancia de HorarioAdapter, un adaptador personalizado que se utiliza para manejar la presentación de los datos en el RecyclerView y list, una lista (ArrayList) que almacenará objetos de tipo Horario, habiendo definido lo anterior se inicializa la lista y el adaptador y junto a ello se utiliza un ValueEventListener para escuchar los cambios en la ubicación de la base de datos (PrivateHorario). Cuando hay cambios, como la adición de nuevos horarios, en ello, el método onDataChange se activa, dentro de este se itera a través de los hijos de la instantánea (snapshot) para obtener cada objeto Horario almacenado en la base de datos. Estos objetos se agregan a la lista, finalmente dentro de onCancelled, se manejan errores específicos de la base de datos, como la denegación de permisos, problemas de red

y otros errores generales.

```
public class ViewHorarioActivity extends AppCompatActivity {

    4 usages
    RecyclerView recyclerView;
    2 usages
    DatabaseReference database;
    3 usages
    HorarioAdapter myAdapter;
    3 usages
    ArrayList<Horario> list;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_view_horario);

        recyclerView = findViewById(R.id.horariosList);
        database = FirebaseDatabase.getInstance().getReference( path: "PrivateHorario");
        recyclerView.setHasFixedSize(true);
        recyclerView.setLayoutManager(new LinearLayoutManager( context: this));

        list = new ArrayList<>();
        myAdapter = new HorarioAdapter( context: this,list);
        recyclerView.setAdapter(myAdapter);

        database.addValueEventListener(new ValueEventListener() {
```

```
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {

                for (DataSnapshot dataSnapshot : snapshot.getChildren()){

                    Horario horario = dataSnapshot.getValue(Horario.class);
                    list.add(horario);
                }
                myAdapter.notifyDataSetChanged();
            }
            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                if (error.getCode() == DatabaseError.PERMISSION_DENIED) {
                    Toast.makeText( context: ViewHorarioActivity.this,  text: "No tienes permiso para acceder a los datos", Toast.LENGT
                } else if (error.getCode() == DatabaseError.NETWORK_ERROR) {
                    Toast.makeText( context: ViewHorarioActivity.this,  text: "Error de red, por favor, comprueba tu conexión", Toast.L
                } else {
                    Toast.makeText( context: ViewHorarioActivity.this,  text: "Se produjo un error al acceder a los datos", Toast.LENGT
                }
            }
        });
    }
}
```

Si el usuario establece un control por sensores, el botón “btn_setSensor” entra en acción, cuando este es presionado, se ejecuta el código dentro del método onClick(View v) del OnClickListener, dentro de este se utiliza un objeto de referencia (databaseReference) a la base de datos de Firebase para almacenar un cambio en la base de datos, el valor almacenado es switchState, que representa el estado actual del interruptor, finalmente se utiliza un OnCompleteListener para verificar si la operación de almacenamiento en la base de datos fue exitosa. Si es así, se muestra un mensaje de éxito y se inicia una nueva actividad (LobbyActivityModel). Si no, se muestra un mensaje de error.

```
btn_setSensor.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        databaseReference.child( pathString: "Usuarios").child(userId).child( pathString: "PrivateNameElec")
                .child(String.valueOf(switchID)).child( pathString: "PrivateSensor").child(String.valueOf(switchID))
                .setValue(switchState).addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                if(task.isSuccessful()) {
                    Toast.makeText( context: SensorActivity.this,  text: "Cambio establecido", Toast.LENGTH_SHORT).show();
                    startActivity(new Intent(SensorActivity.this, LobbyActivityModel.class));
                } else {
                    Toast.makeText( context: SensorActivity.this,  text: "Error al establecer el cambio", Toast.LENGTH_SHORT).sho
                }
            }
        });
    }
});
```

Si el usuario desea establecer un control por intensidad, el botón “btn_setIntensity” entra en acción, cuando este es presionado, se ejecuta el código dentro del método onClick(View v) del OnClickListener, dentro de este obtiene el valor actual de la intensidad desde un SeekBar (barra de potencia), luego se utiliza un objeto de referencia (databaseReference) a la base de datos de Firebase para almacenar el valor de la intensidad en la base de datos, finalmente se utiliza un OnCompleteListener para verificar si la operación de almacenamiento en la base de datos fue exitosa. Si es así, se muestra un mensaje de éxito y se inicia una nueva actividad (LobbyActivityModel). Si no, se muestra un mensaje de error.

```
btn_setIntensity.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        int intensityValue = seekBar.getProgress();

        databaseReference.child( pathString: "Usuarios").child(userId).child( pathString: "PrivateNameElec") DatabaseReference
                .child(IntensityID).child( pathString: "PrivateIntensity").child(IntensityID).setValue(intensityValue) Task<V
                .addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                if(task.isSuccessful()) {
                    Toast.makeText( context: IntensityModel.this,  text: "Intensidad establecida", Toast.LENGTH_SHORT).show();
                    startActivity(new Intent(IntensityModel.this, LobbyActivityModel.class));
                }else {
                    Toast.makeText( context: IntensityModel.this,  text: "Error al establecer intensidad", Toast.LENGTH_SHORT)
                }
            }
        });
    }
});
```

Puente Firebase - Arduino

Para establecer la conexión entre la aplicación y el electrodoméstico, el ESP32 que está vinculado al timer que a su vez está integrado en el electrodoméstico, se configura mediante wifi para que pueda acceder a la base de datos para leer y efectuar los cambios que guardó el usuario, para ello primero se definen las credenciales de Wi-Fi (WIFI_SSID y WIFI_PASSWORD) y las credenciales de Firebase (API_KEY, DATABASE_URL, USER_EMAIL, y USER_PASSWORD), luego se realiza la inicialización de Firebase y se configura el tiempo mediante la librería ESP32Time y se definen objetos relacionados con Firebase, como FirebaseData, FirebaseAuth, y FirebaseConfig, después se crean instancias de sistemas de iluminación (bath, room, living, light, bathFire, roomFire, livingFire, lightFire) utilizando una clase llamada LightSystem. Luego de ello se da paso a la configuración de hardware. Para ello se configuran los pines del ESP32 para su uso, tanto salidas (OUTPUT) como entradas (INPUT) y junto a ello los modos de iluminación para cada habitación utilizando métodos de la clase LightSystem.

Finalmente se configura la función principal del código , en ello se realiza la conexión a Wi-Fi y se espera hasta que la conexión sea exitosa para luego obtener la hora y fecha actuales de Firebase para almacenarlas en variables locales, después

se configuran los parámetros de Firebase, como la clave API, las credenciales de usuario y la URL de la base de datos así como las funciones de devolución de llamada para manejar el estado del token y se inicia la conexión a Firebase , de allí se obtienen datos específicos de la base de datos, como el estado de los sensores, la activación de horarios y otros datos relacionados con la iluminación, finalmente se implementa la lógica para controlar los sistemas de iluminación (activación de sensores, horarios, configuración de intensidad) según los datos obtenidos de Firebase.

```cpp
// Inclusión de Librerías Necesarias
#include "LightSystem.h"
#include <ESP32Time.h>
#include <Arduino.h>
#include <WiFi.h>
#include <FirebaseESP32.h>

#include <addons/TokenHelper.h>
#include <addons/RTDBHelper.h>

// Definición de Credenciales de Wi-Fi y Firebase
#define WIFI_SSID "Redmi Note 11"
#define WIFI_PASSWORD "1234abcd"
#define API_KEY "AIzaSyC3p_QBCbZko4fcUhT68zmeeUsbL-Af7fg"
#define DATABASE_URL "aether-f4235-default-rtdb.firebaseio.com"
#define USER_EMAIL "diegojnavarrol@gmail.com"
#define USER_PASSWORD "12345abcd"

// Inicialización de la Clase para Manejo del Tiempo en ESP32
ESP32Time rtc;

// Parámetros para Configuración del Tiempo
const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = -5*3600;
const int dayLightOffset_sec = 0;

// Configuración de Firebase
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;

// Variables para el Control del Tiempo de Envío de Datos
unsigned long sendDataPrevMillis = 0;

// Contador
unsigned long count = 0;
```

```cpp
// Declaración de Objetos de Sistemas de Iluminación
LightSystem bath, room, living, light;
LightSystemFire bathFire, roomFire, livingFire, lightFire;

void setup() {

// Inicialización de la Comunicación Serial
  Serial.begin(115200);

// Conexión a Wi-Fi
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to Wi-Fi");
  while (WiFi.status() != WL_CONNECTED)
  {
    Serial.print(".");
    delay(300);
  }
  Serial.println();
  Serial.print("Connected with IP: ");
  Serial.println(WiFi.localIP());
  Serial.println();

// Configuración del Tiempo y Obtención de la Hora Actual
  configTime(gmtOffset_sec, dayLightOffset_sec, ntpServer);
  struct tm timeinfo;
  if(getLocalTime(&timeinfo)){
    rtc.setTimeStruct(timeinfo);
  }

  Serial.printf("Firebase Client v%s\n\n", FIREBASE_CLIENT_VERSION);

  config.api_key = API_KEY;

  auth.user.email = USER_EMAIL;
  auth.user.password = USER_PASSWORD;
```

```
  config.database_url = DATABASE_URL;

  config.token_status_callback = tokenStatusCallback;

  Firebase.reconnectNetwork(true);

  fbdo.setBSSLBufferSize(4096, 1024);

  Firebase.begin(&config, &auth);

  Firebase.setDoubleDigits(5);
//Declaración de pines a usar
  pinMode(2,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(22,OUTPUT);
  pinMode(23,OUTPUT);
  pinMode(34, INPUT);

//Configuración de la iluminación en los disitintos puntos de la casa
  bath.Mode(5, 14, 33);
  room.Mode(18, 27, 33);
  living.Mode(19, 26, 33);
  light.Mode(21, 25, 33);

  bathFire.Mode(5, 14);
  roomFire.Mode(18, 27);
  livingFire.Mode(19, 26);
  lightFire.Mode(21, 25);

}
void loop() {

  // Encendido de Luces de Prueba
  digitalWrite(2, HIGH);
  digitalWrite(4, HIGH);
  digitalWrite(22, HIGH);
  digitalWrite(23, HIGH);

  // Lectura del Valor Analógico en el Pin 33
  Serial.println(analogRead(33));

  // Obtención de Componentes de la Fecha y Hora Actual
  int year = rtc.getYear();
  int month = rtc.getMonth();
  int day = rtc.getDay();
  int hour = rtc.getHour(true);
  int minute = rtc.getMinute();

  // Creación de Números para Representar la Hora y Fecha Actual
  int actualtime = minute + (hour)*pow(10,2);
  int actualdate = year + (month)*pow(10,4) + (day)*pow(10,6);
  double actual = actualtime*pow(10,6) + actualdate/pow(10,2);

  // Espera de 1 segundo
  delay(1000);
```

```
if (Firebase.ready() && (millis() - sendDataPrevMillis > 15000 || sendDataPrevMillis == 0))
{
  sendDataPrevMillis = millis();

  // Obtención de Datos de Firebase
  bool actsen;
  Serial.printf("Get bool ref... %s\n", Firebase.getBool(fbdo, F("/Usuarios/TTvUfct2vrUP9SwavNEqSbdkL1n2/PrivateNameElec/PrivateSensor/active")
  , &actsen) ? actsen ? "true" : "false" : fbdo.errorReason().c_str());

  bool actsche;
  Serial.printf("Get bool ref... %s\n", Firebase.getBool(fbdo, F("/Usuarios/TTvUfct2vrUP9SwavNEqSbdkL1n2/PrivateNameElec/PrivateHorario/active")
  , &actsen) ? actsen ? "true" : "false" : fbdo.errorReason().c_str());

  int fechaInicio = 0;
  Serial.printf("Get int ref... %s\n", Firebase.getInt(fbdo, F("/Usuarios/TTvUfct2vrUP9SwavNEqSbdkL1n2/PrivateNameElec/PrivateHorario/fechaInicio")
  , &fechaInicio) ? String(fechaInicio).c_str() : fbdo.errorReason().c_str());

  int horaInicio = 0;
  Serial.printf("Get int ref... %s\n", Firebase.getInt(fbdo, F("/Usuarios/TTvUfct2vrUP9SwavNEqSbdkL1n2/PrivateNameElec/PrivateHorario/horaInicio")
  , &horaInicio) ? String(horaInicio).c_str() : fbdo.errorReason().c_str());

  int fechaFin = 0;
  Serial.printf("Get int ref... %s\n", Firebase.getInt(fbdo, F("/Usuarios/TTvUfct2vrUP9SwavNEqSbdkL1n2/PrivateNameElec/PrivateHorario/fechaFin")
  , &fechaFin) ? String(fechaFin).c_str() : fbdo.errorReason().c_str());

  int horaFin = 0;
  Serial.printf("Get int ref... %s\n", Firebase.getInt(fbdo, F("/Usuarios/TTvUfct2vrUP9SwavNEqSbdkL1n2/PrivateNameElec/PrivateHorario/horaInicio")
  , &horaFin) ? String(horaFin).c_str() : fbdo.errorReason().c_str());

  int faclum = 0;
  Serial.printf("Get int ref... %s\n", Firebase.getInt(fbdo, F("/Usuarios/TTvUfct2vrUP9SwavNEqSbdkL1n2/PrivateNameElec/PrivateSensor/intense")
  , &faclum) ? String(faclum).c_str() : fbdo.errorReason().c_str());

  double inicio = fechaInicio/pow(10,2) + horaInicio*pow(10,6);
  double fin = fechaFin/pow(10,2)+ horaFin*pow(10,6);
    // Control de Iluminación basado en Condiciones de Firebase
    if(actsen == true){

        bathFire.Luminity(5, 14, faclum);
        roomFire.Luminity(18, 27, faclum);
        livingFire.Luminity(19, 26, faclum);
        lightFire.Luminity(21, 25, faclum);

    } else if(actsche == true){

      if((inicio < actual)&(fin > actual)){

        bath.Luminity(5, 14, 33);
        room.Luminity(18, 27, 33);
        living.Luminity(19, 26, 33);
        light.Luminity(21, 25, 33);

      }

    } else {

      bath.Luminity(5, 14, 33);
      room.Luminity(18, 27, 33);
      living.Luminity(19, 26, 33);
      light.Luminity(21, 25, 33);

    }
  }
}

```

LightSystem.h:

Esta clase está definida para manejar sistemas de iluminación basados en la lectura de sensores de luz en placas ESP32.

Atributos:

led, read, y factor son variables de instancia que representan los pines del LED, el sensor de luz y un factor de referencia

respectivamente.

Métodos:

Mode(int led, int read, int factor): Este método configura los modos de los pines según los parámetros dados.

Luminity(int led, int read, int factor): Este método lee los valores analógicos del sensor de luz (read) y del factor (factor). Si el valor del sensor es menor o igual al valor del factor, se enciende el LED; de lo contrario, se apaga.

```
#include "HardwareSerial.h"
#include "esp32-hal-adc.h"

// Definición de la clase LightSystem
class LightSystem {
  private:
    int led;
    int read;
    int factor;

  public:
    // Método para configurar los pines y la comunicación serial
    void Mode(int led, int read, int factor) {
      pinMode(led, OUTPUT);
      pinMode(read, INPUT);
      pinMode(factor, INPUT);
      Serial.begin(115200);
    }

    // Método para controlar la iluminación basada en la lectura del sensor
    void Luminity(int led, int read, int factor) {
      if (analogRead(read) <= analogRead(factor)) {
        digitalWrite(led, HIGH);
      } else {
        digitalWrite(led, LOW);
      }
    }
};
```

```
// Definición de la clase LightSystemFire
class LightSystemFire {
  private:
    int led;
    int read;
    int factor;

  public:
    // Método para configurar los pines y la comunicación serial
    void Mode(int led, int read) {
      pinMode(led, OUTPUT);
      pinMode(read, INPUT);
      Serial.begin(115200);
    }

    // Método para controlar la iluminación basada en la lectura del sensor con un factor específico
    void Luminity(int led, int read, int factor) {
      // Multiplica el factor por un valor específico (4095/100)
      if (analogRead(read) <= factor * 4095 / 100) {
        digitalWrite(led, HIGH);
      } else {
        digitalWrite(led, LOW);
      }
    }
};
```

FirebaseESP32.h:

Esta librería facilita la interacción con la base de datos en tiempo real de Firebase de Google, así como con otras características de Firebase, el almacenamiento en la nube y las funciones en la misma, al incluir FirebaseESP32.h se incorporan funcionalidades que permiten realizar operaciones como leer y escribir datos en la base de datos en tiempo real de Firebase, autenticar usuarios, enviar y recibir notificaciones en tiempo real, entre otras cosas. Para usar esta librería, se necesita configurar adecuadamente las credenciales de Firebase, como la clave de API, la URL de la base de datos y, en caso de ser necesario, las credenciales de usuario para la autenticación.