A close up of text on a white background

Description automatically generated

**Introducción:**

En el presente proyecto se intentará tener un control y registro de las actividades de un robot autónomo de manera inalámbrica. Esto se logrará utilizando la base de datos de Firebase.

Se buscará establecer los conceptos básicos para el desarrollo de visión artificial aplicada a la agroindustria nacional. Siendo su principal objetivo el desarrollo de un sistema de visión artificial para la clasificación de hojas de soja según características externas.

La ventaja de los sistemas automatizados que se utilizan en la agroindustria, es que, además de tener una mayor productividad, menor costo y control de calidad mas efectivo, permite la inspección total del cultivo de un modo no destructivo consistente y objetivo. También tiene la capacidad de proporcionar un análisis de cada pixel de la superficie de la hoja, y la disponibilidad de generar rápidamente resultados del análisis y almacenar datos.

**Descripción general del proyecto:**

Nuestro proyecto esta constituido por 4 dispositivos principales, el cual colaboraran mutuamente para lograr sus objetivos.

1. Celular: el celular será el control inalámbrico, es decir que constara de un panel de control con el cual el usuario podrá interactuar. Además, utilizamos Firebase Authentication para diferenciar a todos los usuarios y sus dispositivos. Es decir, que para cada raspberry pi podrán acceder solo los propietarios de dicha raspberry.
2. Computadora: Nuestra computadora será el cerebro principal de reconocimiento de imágenes. Además, podrá administrar la base de datos de manera más optima.
3. Raspberry Pi: Nuestra Raspberry pi será la que le de las órdenes a nuestra placa de Arduino, además, tendrá el trabajo de recolectar fotos con su webcam.
4. Arduino Uno: La Arduino se encargará de los movimientos mecánico de nuestro robot autónomos, además deberá tener el control físico sobre la cámara que tomara las imágenes. Leyendo datos de sus sensores.

A Firebase lo podemos definir como una plataforma de desarrollo. Esta le pertenece a Google y utiliza la nube y un conjunto de herramientas para la creación y sincronización de proyectos. Esto facilita mucho la creación de apps de elevada calidad.

Firebase Realtime Database es la base de dato que utilizaremos en este proyecto, esta base de datos NoSQL (no utiliza SQL) se encuentra alojada en la nube, esto nos permite almacenar y sincronizar datos de usuarios y nuestro Bogie en tiempo real. ¿Como realiza esta sincronización? Bueno, esta sincronización se lleva a cabo con un archivo json, ¿y que es un archivo .json? Un archivo Jason es un formato de texto sencillo utilizado en la programación para intercambiar datos entre diferentes lenguajes.

Codigo JSON de nuestro Programa:

{

  "project\_info": {

    "project\_number": "238034566428",

    "firebase\_url": "https://sistemasbogie.firebaseio.com",

    "project\_id": "sistemasbogie",

    "storage\_bucket": "sistemasbogie.appspot.com"

  },

  "client": [

    {

      "client\_info": {

        "mobilesdk\_app\_id": "1:238034566428:android:e8500edde9724b34240ed9",

        "android\_client\_info": {

          "package\_name": "com.example.controlbogie"

        }

      },

      "oauth\_client": [

        {

          "client\_id": "238034566428-d6s6m7o40ijtrm74tao2rrp1f5ros4ld.apps.googleusercontent.com",

          "client\_type": 3

        }

      ],

      "api\_key": [

        {

          "current\_key": "AIzaSyCrRhi2PbwqbrL\_c\_T85jJuoK-4U5MPdlY"

        }

      ],

      "services": {

        "appinvite\_service": {

          "other\_platform\_oauth\_client": [

            {

              "client\_id": "238034566428-d6s6m7o40ijtrm74tao2rrp1f5ros4ld.apps.googleusercontent.com",

              "client\_type": 3

            }

          ]

        }

      }

    }

  ],

  "configuration\_version": "1"

}

Como Podemos ver poseemos varios identificadores únicos. Esto nos permite sincronizar y acceder desde cualquier dispositivo y lenguaje a nuestra base de datos.

Otra de las ventajas de esta Base de datos es que, no necesitamos servidores para nuestras aplicaciones. Lo que facilita muchísimo el trabajo y nos permite acceder incluso desde otras redes a nuestro proyecto y manejar nuestro robot Bogie desde cualquier lado.

En cuanto a la autentificación, bueno, nuestra idea es que cada usuario pueda manejar solo su Bogie ya que al probar este proyecto con más de un smartphone, todos podía controlar uno solo y esto podría generar conflictos. Por eso, implemente un login el cual el Usuario podrá registrarse y identificar su Bogie con un nro de serie y solo manejar ese bogie.

Android Studio:

Para empezar, creamos un proyecto, este proyecto debe sincronizarse debidamente con nuestra Firebase, para ello, entramos a Firebase y registramos una nueva aplicación. Una vez llenado todo los campos en la siguiente ventana nos aparecen pasos a seguir. Códigos que debemos pegar en nuestros Gradle Scripts, que es un sistema de automatización de construcción de código abierto que construye sobre los conceptos de apache (servidor web http de código abierto) e introduce un lenguaje especifico de dominio o DSL basado en Groovy (Lenguaje de programacion orientada a objetos implementados sobre la plataforma de Java), no XML (Lenguaje de marcado extensible).

Build.gradle (Module:app)

dependencies **{** implementation fileTree(dir: "libs", include: ["\*.jar"])  
 implementation 'com.google.firebase:firebase-analytics:17.4.4'  
 implementation 'androidx.appcompat:appcompat:1.1.0'  
 implementation 'androidx.constraintlayout:constraintlayout:1.1.3'  
 implementation 'com.google.firebase:firebase-core:17.4.4'  
 implementation 'com.google.firebase:firebase-auth:19.3.2'  
 implementation 'com.google.firebase:firebase-database:19.3.1'  
 implementation 'androidx.gridlayout:gridlayout:1.0.0'  
 testImplementation 'junit:junit:4.12'  
 androidTestImplementation 'androidx.test.ext:junit:1.1.1'  
 androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'  
  
**}**apply plugin: 'com.google.gms.google-services'

Esto es en Gradle Scripts, las implementaciones que hacemos, y las por defecto. Lo que hacemos es agregar las dependencias de la biblioteca de Android de Firebase authenticaction y realtime database, etc. al archivo grandle.

**Java**

Cuando creamos el proyecto nos da la opción del lenguaje, nos permite programar en Kotlin y Java. Seleccionamos Java y empezamos en el mainActivity, nombre por defecto que nos da nuestro app a la plantilla.

Para cada pantalla de aplicación estamos creando un XML con algunos layouts (implementación estática). En ejecución, cada vez que se lee el archivo XML para obtener todos los recursos que quiere mostrar la pantalla que diseñamos.

A screenshot of a cell phone

Description automatically generated![A screenshot of a computer

Description automatically generated]()![A picture containing grass, photo, car, truck

Description automatically generated]()

Ilustración MenuActivity

Ilustración RegActivity

Ilustración MainActivity

A close up of text on a white background

Description automatically generatedA picture containing clock, drawing

Description automatically generatedA screenshot of a cell phone

Description automatically generated

Ilustración GraficaActivity

Ilustración ControlActivity

Ilustración DatabaseActivity

MainActivity

En nuestro MainActivity mostraremos el login, en donde podremos ingresar nuestro usuario y contraseña y avanzar hacia la ventana del menú principal, o con el botón registrar, registrar un nuevo usuario.

public class MainActivity extends AppCompatActivity {  
 /\*  
 Declaramos Botones y EditText.  
 \*/  
 private Button bIniciar;  
 private Button bRegistrar;  
 private EditText tEmail;  
 private EditText tPass;  
 /\*  
 Variables para manipular los datos de los EditText como Strings.  
 \*/  
 private String email = "";  
 private String pass = "";  
 /\*  
 Instanciamos Firebase Authentication  
 \*/  
 private FirebaseAuth mAuth;  
 /\*  
 Metodo onCreate no solo es responsable de crear la actividad, sino que, tambien puede guardar el estado de la aplicacion utilizando Bundle  
 savedInstanceState. Básicamente, la clase Bundle se usa para almacenar los datos de la actividad siempre que ocurra la condición anterior  
 en la aplicación.  
 \*/  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 /\*  
 Metodos para mostrar el diseno en pantalla, donde R es recurso, layout diseno y activity\_reg es el archivo XML.  
 Metodo para mostrar el SplashScreen  
 \*/  
 setContentView(R.layout.*activity\_main*);  
 setTheme(R.style.*AppTheme*);  
 /\*  
 Inicializamos la intancia de FirebaseAuthentication.  
 \*/  
 mAuth = FirebaseAuth.*getInstance*();  
 /\*  
 Utilizamos el findViewById para buscar una vista determinada por su id dentro de un layout especifico habiendo especificado el tipo ej EditText.  
 \*/  
 tEmail = (EditText) findViewById(R.id.*emailreg*);  
 tPass = (EditText) findViewById(R.id.*password*);  
 bIniciar = (Button) findViewById(R.id.*botoniniciar*);  
 bRegistrar = (Button) findViewById(R.id.*botonregistrar*);  
 /\*  
 Boton Iniciar, declaramos el metodo onClick, el cual le diremos que funciones debera realizar al clickear este boton.  
 \*/  
 bIniciar.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 /\*  
 Obtenemos el contenido de los EditText y los pasamos a String para almacenarlos en nuestras variables anteriormente declaradas.  
 \*/  
 email = tEmail.getText().toString();  
 pass = tPass.getText().toString();  
 /\*  
 Preguntamos si los EditText no estan vacios. Si lo estan, mostramos mensaje para avisarle al usuario.  
 \*/  
 if(!email.isEmpty() && !pass.isEmpty()){  
 /\*  
 Llamamos al metodo loginUser()  
 \*/  
 loginUser();  
 }else{  
 /\*  
 Mensaje de error si no se completaron los campos.  
 \*/  
 Toast.*makeText*(MainActivity.this, "Porfavor, complete los campos y vuelva a intentarlo.", Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
  
 private void loginUser() {  
 /\*  
 Logueamos con nuestro Email y contrasena, si es que se encuentran registradas en Firebase Authentication.  
 \*/  
 mAuth.signInWithEmailAndPassword(email, pass).addOnCompleteListener(new OnCompleteListener<AuthResult>() {  
 @Override  
 public void onComplete(@NonNull Task<AuthResult> task) {  
 /\*  
 Si la verificacion es correcta llamamos al metodo starActivities.  
 \*/  
 if(task.isSuccessful()){  
 /\*  
 Lanzamos una nueva actividad, Intent describimos las actividades, primero en la que nos encontramos y luego a la cual queremos acceder.  
 Luego cerramos la actividad actual.  
 \*/  
 startActivities(new Intent[]{new Intent( MainActivity.this, CargandoActivity.class)});  
 finish();  
 }else{  
 /\*  
 Si no se encontro el Email y la contrasena o no hay acceso a internet, informamos al usuario que no se pudo iniciar sesion.  
 \*/  
 Toast.*makeText*(MainActivity.this, "No se pudo iniciar sesion.", Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
 });  
 }  
 });  
 /\*  
 Boton Registrar, declaramos el metodo onClick, el cual le diremos que funciones debera realizar al clickear este boton.  
 \*/  
 bRegistrar.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 /\*  
 Lanzamos una nueva actividad, Intent describimos las actividades, primero en la que nos encontramos y luego a la cual queremos acceder.  
 Luego cerramos la actividad actual.  
 \*/  
 startActivities(new Intent[]{new Intent( MainActivity.this, RegActivity.class)});  
 finish();  
 }  
 });  
  
  
 }  
}

RegActivity

Pantalla de registro, aquí encontramos varios campos para completar, Nombre de usuario, correo, contraseña, localidad,nro de serie y domicilio. Básicamente lo que hacemos aquí es registrar en Firebase Auth nuestro mail y contraseña, y los demás datos, van a parar a la base de datos.

public class RegActivity extends AppCompatActivity {  
 /\*  
 Declaramos Botones y EditText.  
 \*/  
 private Button bVolver;  
 private Button bSiguiente;  
 private EditText tNroSerie;  
 private EditText tLocalidad;  
 private EditText tDomicilio;  
 private EditText tUsuario;  
 private EditText tEmail;  
 private EditText tContrasena;  
 /\*  
 Variables para manipular los datos de los EditText como Strings.  
 \*/  
 private String srie = "";  
 private String loc = "";  
 private String dom = "";  
 private String usu = "";  
 private String ema = "";  
 private String con = "";  
 /\*  
 Instanciamos Firebase Authentication y Realtime Database  
 \*/  
 FirebaseAuth mAuth;  
 DatabaseReference mDatabase;  
 /\*  
 Metodo onCreate no solo es responsable de crear la actividad, sino que, tambien puede guardar el estado de la aplicacion utilizando Bundle  
 savedInstanceState. Básicamente, la clase Bundle se usa para almacenar los datos de la actividad siempre que ocurra la condición anterior  
 en la aplicación.  
 \*/  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 /\*  
 Metodos para mostrar el diseno en pantalla, donde R es recurso, layout diseno y activity\_reg es el archivo XML  
 \*/  
 setContentView(R.layout.*activity\_reg*);  
 /\*  
 Inicializamos la intancia de FirebaseAuthentication y FirebaseDatabase. En Realtime Database Firebase obtenemos la referencia del nodo principal.  
 \*/  
 mAuth = FirebaseAuth.*getInstance*();  
 mDatabase = FirebaseDatabase.*getInstance*().getReference();  
 /\*  
 Utilizamos el findViewById para buscar una vista determinada por su id dentro de un layout especifico habiendo especificado el tipo ej EditText.  
 \*/  
 tNroSerie = (EditText) findViewById(R.id.*nroSerie*);  
 tLocalidad = (EditText) findViewById(R.id.*Localidad*);  
 tDomicilio = (EditText) findViewById(R.id.*domicilio*);  
 tUsuario = (EditText) findViewById(R.id.*usuario*);  
 tEmail = (EditText) findViewById(R.id.*emailreg*);  
 tContrasena = (EditText) findViewById(R.id.*contrareg*);  
 bSiguiente = (Button) findViewById(R.id.*botonSiguente*);  
 bVolver = (Button) findViewById(R.id.*botonVolver*);  
  
 /\*  
 Boton Volver, declaramos el metodo onClick, el cual le diremos que funciones debera realizar al clickear este boton  
 \*/  
 bVolver.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 /\*  
 Lanzamos una nueva actividad, Intent describimos las actividades, primero en la que nos encontramos y luego a la cual queremos acceder.  
 Luego cerramos la actividad actual.  
 \*/  
 startActivities(new Intent[]{new Intent( RegActivity.this, MainActivity.class)});  
 finish();  
 }  
 });  
 /\*  
 Boton Siguiente, declaramos el metodo onClick, el cual le diremos que funciones debera realizar al clickear este boton  
 \*/  
 bSiguiente.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 /\*  
 Obtenemos el contenido de los EditText y los pasamos a String para almacenarlos en nuestras variables anteriormente declaradas.  
 \*/  
 srie = tNroSerie.getText().toString();  
 loc = tLocalidad.getText().toString();  
 dom = tDomicilio.getText().toString();  
 usu = tUsuario.getText().toString();  
 con = tContrasena.getText().toString();  
 ema = tEmail.getText().toString();  
 /\*  
 Preguntamos si los EditText no estan vacios. Si lo estan, mostramos mensaje para avisarle al usuario.  
 \*/  
 if (!dom.isEmpty() && !srie.isEmpty() && !loc.isEmpty() && !usu.isEmpty() && !con.isEmpty() && !ema.isEmpty()) {  
 /\*  
 Queremos que nuestra contrasena tenga al menos 6 caracteres, entonces preguntamos si la variable con posee 6 o mas caracteres.  
 \*/  
 if (con.length() >= 6) {  
 /\*  
 Llamamos al metodo registrarBogie()  
 \*/  
 registrarBogie();  
 } else {  
 /\*  
 Mensaje de error si la contrasena tiene menos de 6 caracteres.  
 \*/  
 Toast.*makeText*(RegActivity.this, "La Password debe tener al menos 6 caracteres.", Toast.*LENGTH\_SHORT*).show();  
 }  
 } else {  
 /\*  
 Mensaje si no se llenaron todos los campos.  
 \*/  
 Toast.*makeText*(RegActivity.this, "Porfavor, complete los campos y vuelva a intentarlo.", Toast.*LENGTH\_SHORT*).show();  
  
 }  
 }  
  
  
 private void registrarBogie() {  
 /\*  
 Creamos un usuario a partir de el Email y la contrasena.  
 \*/  
 mAuth.createUserWithEmailAndPassword(ema, con).addOnCompleteListener(new OnCompleteListener<AuthResult>() {  
 /\*  
 Este metodo nos da un resultado Booleano, es True si se completo el Registro correctamente.  
 \*/  
 @Override  
 public void onComplete(@NonNull Task<AuthResult> task) {  
 if (task.isSuccessful()) {  
 /\*  
 Necesitamos la id del usuario para utilizar como clave unica en nuestra base de datos.  
 \*/  
 String id = mAuth.getCurrentUser().getUid(); //Obtenemos la clave unica de usuario  
 /\*  
 Nos situamos en el nodo Id de nuestra DB e ingresamos el nombre de usuario.  
 \*/  
 mDatabase.child("Bogie").child(srie).child("Administradores").child(id).setValue(usu);  
 /\*  
 Generamos un HashMap, este HashMap almacenara la informacion del usuario y el numero de serie de su bogie.  
 \*/  
 Map<String, Object> map = new HashMap<>();  
 map.put("Usuario", usu);  
 map.put("E-Mail", ema);  
 map.put("Contraseña", con);  
 map.put("Localidad", loc);  
 map.put("Domicilio", dom);  
 map.put("Serie", srie);  
 /\*  
 Nos situamos en el nodo Id e ingresamos los datos del usuario, y generamos un metodo si se realizaron correctamente.  
 \*/  
 mDatabase.child("Identificaciones").child(id).setValue(map).addOnCompleteListener(new OnCompleteListener<Void>() {  
 @Override  
 public void onComplete(@NonNull Task<Void> task2) {  
 /\*  
 Si esto es verdadero, es decir, que se guardo correctamente en la Base de Dato, vamos a la pantalla CargandoActivity.  
 \*/  
 if (task2.isSuccessful()) {  
 /\*  
 Lanzamos una nueva actividad, Intent describimos las actividades, primero en la que nos encontramos y luego a la cual queremos acceder.  
 Luego cerramos la actividad actual.  
 \*/  
 startActivities(new Intent[]{new Intent( RegActivity.this, CargandoActivity.class)});  
 finish();  
 } else {  
 /\*  
 Mensaje de error, si no se logro completar en la Base de Datos.  
 \*/  
 Toast.*makeText*(RegActivity.this, "Error.", Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
 });  
 }else {  
 /\*  
 Mensaje de error, si no se logro completar en la Base de Datos.  
 \*/  
 Toast.*makeText*(RegActivity.this, "Error no se logro registrar el usuario.", Toast.*LENGTH\_SHORT*).show();  
 }  
  
 }  
 });  
 }  
 });  
  
 }  
}

MenuActivity

Aca encontramos varios botones, el de Encender envia un valor booleano a la base de dato, este es leído por la raspberry pi zero, si es true, el GPIO 17 se pone en alto dando la orden al Arduino para que comience a trabajar. El botón cerrar sesión, tal y como dice, nos desloguea y nos lleva a la pantalla del main, el control nos permite tomar el modo control manual de el robot, y por ultimo el botón, Realtime database nos muestra información de nuestra base de datos.

public class MenuActivity extends AppCompatActivity {  
 /\*  
 Declaramos Botones y EditText.  
 \*/  
 private Button bCerrarSesion ;  
 private Switch bApagarEncender ;  
 private Button bBaseDeDatos ;  
 private Button bControlManual ;  
 private Button bCopyright ;  
 private ImageView iOnOff;  
 /\*  
 Variables para manipular los datos de los EditText como Strings.  
 \*/  
 private String serie ="";  
 private String nombre ="";  
 /\*  
 Instanciamos Firebase Authentication y Realtime Database  
 \*/  
 FirebaseAuth mAuth;  
 DatabaseReference mDatabase;  
 /\*  
 Metodo onCreate no solo es responsable de crear la actividad, sino que, tambien puede guardar el estado de la aplicacion utilizando Bundle  
 savedInstanceState. Básicamente, la clase Bundle se usa para almacenar los datos de la actividad siempre que ocurra la condición anterior  
 en la aplicación.  
 \*/  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 /\*  
 Metodos para mostrar el diseno en pantalla, donde R es recurso, layout diseno y activity\_reg es el archivo XML  
 \*/  
 setContentView(R.layout.*activity\_menu*);  
 /\*  
 Inicializamos la intancia de FirebaseAuthentication y FirebaseDatabase. En Realtime Database Firebase obtenemos la referencia del nodo principal.  
 \*/  
 mAuth = FirebaseAuth.*getInstance*();  
 mDatabase = FirebaseDatabase.*getInstance*().getReference(); //Aqui nos referimos al nodo principal de nuestra base de datos  
 /\*  
 Utilizamos el findViewById para buscar una vista determinada por su id dentro de un layout especifico habiendo especificado el tipo ej EditText.  
 \*/  
 final TextView tUsuario = (TextView) findViewById(R.id.*tusuario*);  
 final TextView tSerie = (TextView) findViewById(R.id.*tSerie*);  
 bCerrarSesion = (Button) findViewById(R.id.*cerrarSesion*);  
 bApagarEncender = (Switch) findViewById(R.id.*swich*);  
 bBaseDeDatos = (Button) findViewById(R.id.*datosAlmacenados*);  
 bControlManual = (Button) findViewById(R.id.*cmanual*);  
 bCopyright = (Button) findViewById(R.id.*copyright*);  
 iOnOff = (ImageView) findViewById(R.id.*imagenfoco*);  
 /\*  
 Necesitamos la id del usuario para utilizar como clave unica en nuestra base de datos.  
 \*/  
 String id = mAuth.getCurrentUser().getUid();  
 /\*  
 Nos situamos en el nodo Id e ingresamos los datos del usuario, y generamos un metodo si se realizaron correctamente.  
 \*/  
 mDatabase.child("Identificaciones").child(id).addValueEventListener(new ValueEventListener() {  
 @Override  
 public void onDataChange(@NonNull DataSnapshot snapshot) {  
 /\*  
 Obtenemos el contenido del nodo al cual hicimos referencia en nuestra Base de Dato y obtenemos los valores que necesitamos.  
 \*/  
 nombre = snapshot.child("Usuario").getValue().toString();  
 serie = snapshot.child("Serie").getValue().toString();  
 /\*  
 Establecemos el texto escaneado en nuestro TextView.  
 \*/  
 tUsuario.setText("Bienvenido " + nombre);  
 tSerie.setText("Nro de Serie: " + serie);  
 }  
 @Override  
 public void onCancelled(@NonNull DatabaseError error) {  
 }  
 });  
 /\*  
 Boton Cerrar sesion, declaramos el metodo onClick, el cual le diremos que funciones debera realizar al clickear este boton  
 \*/  
 bCerrarSesion.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 /\*  
 Utilizamos la funcion singOut para desloguear.  
 \*/  
 mAuth.signOut();  
 /\*  
 Lanzamos una nueva actividad, Intent describimos las actividades, primero en la que nos encontramos y luego a la cual queremos acceder.  
 Luego cerramos la actividad actual.  
 \*/  
 startActivities(new Intent[]{new Intent(MenuActivity.this, MainActivity.class)});  
 finish();  
 }  
 });  
 /\*  
 Swich ApagarEncender, declaramos el metodo onClick, el cual le diremos que funciones debera realizar al clickear este boton  
 \*/  
 bApagarEncender.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {  
 @Override  
 public void onCheckedChanged(CompoundButton compoundButton, boolean b) {  
 /\*  
 Preguntamos si el TooglerButton fue pulsado.  
 Si es verdador creamos un HashMap para luego remplazar en nuestra base de dato lo que necesitemos.  
 En nuestro caso colocamos un true en encendico para darle la señal a nuestra RPI que ponga el GPIO en Out.  
 Si es falso hacemos lo mismo pero colocando el falso.  
 \*/  
 if(b){  
 /\*  
 Generamos un HashMap que nos indica que Encendido es igual a true.  
 \*/  
 Map<String, Object> map = new HashMap<>();  
 map.put("Encendido", b);  
 mDatabase.child("Bogie").child(serie).child("Automatico").updateChildren(map);  
 /\*  
 Indicamos la imagen que queremos mostrar.  
 \*/  
 iOnOff.setImageResource(R.drawable.*ecoluzon*);  
  
 }else{  
 /\*  
 Generamos un HashMap que nos indica que Encendido es igual a False.  
 \*/  
 Map<String, Object> map = new HashMap<>();  
 map.put("Encendido", b);  
 mDatabase.child("Bogie").child(serie).child("Automatico").updateChildren(map);  
 /\*  
 Indicamos la imagen que queremos mostrar.  
 \*/  
 iOnOff.setImageResource(R.drawable.*ecoluzoff*);  
 }  
 }  
 });  
 /\*  
 Boton Control Manual, declaramos el metodo onClick, el cual le diremos que funciones debera realizar al clickear este boton  
 \*/  
 bControlManual.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 /\*  
 Lanzamos una nueva actividad, Intent describimos las actividades, primero en la que nos encontramos y luego a la cual queremos acceder.  
 Luego cerramos la actividad actual.  
 \*/  
 startActivities(new Intent[]{new Intent( MenuActivity.this, ManualActivity.class)});  
 finish();  
 }  
 });  
 /\*  
 Boton Base de Datos, declaramos el metodo onClick, el cual le diremos que funciones debera realizar al clickear este boton  
 \*/  
 bBaseDeDatos.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 /\*  
 Lanzamos una nueva actividad, Intent describimos las actividades, primero en la que nos encontramos y luego a la cual queremos acceder.  
 Luego cerramos la actividad actual.  
 \*/  
 startActivities(new Intent[]{new Intent( MenuActivity.this, DatabaseActivity.class)});  
 finish();  
 }  
 });  
 /\*  
 Boton Copytight, declaramos el metodo onClick, el cual le diremos que funciones debera realizar al clickear este boton  
 \*/  
 bCopyright.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 /\*  
 Mostramos mensaje  
 \*/  
 Toast.*makeText*(MenuActivity.this, "UADER FCYT - 2020", Toast.*LENGTH\_SHORT*).show();  
 }  
 });  
 }  
}

ControlActivity

Acá quisimos simular la vista de un control al estilo joystick, y que a la izquierda se visualice las fotografías que tomaremos de la pi cámara.

public class ManualActivity extends AppCompatActivity {  
 /\*  
 Declaramos Botones y ImageView.  
 \*/  
 private Button bParar;  
 private Button bStop;  
 private Button bDerecha;  
 private Button bIzquierda;  
 private Button bAtras;  
 private Button bAdelante;  
 private ImageView iImagen;  
 /\*  
 Variables para manipular los datos de los EditText como Strings.  
 \*/  
 private String serie = "";  
 private String id = "";  
 /\*  
 Instanciamos Firebase Authentication, Realtime Database y Storage  
 \*/  
 FirebaseAuth mAuth;  
 DatabaseReference mDatabase;  
 StorageReference mStorage;  
 DatabaseReference first = FirebaseDatabase.*getInstance*().getReference().child("imagen");  
 /\*  
 Metodo onCreate no solo es responsable de crear la actividad, sino que, tambien puede guardar el estado de la aplicacion utilizando Bundle  
 savedInstanceState. Básicamente, la clase Bundle se usa para almacenar los datos de la actividad siempre que ocurra la condición anterior  
 en la aplicación.  
 \*/  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 /\*  
 Metodos para mostrar el diseno en pantalla, donde R es recurso, layout diseno y activity\_reg es el archivo XML  
 \*/  
 setContentView(R.layout.*activity\_manual*);  
 /\*  
 Inicializamos la intancia de FirebaseAuthentication y FirebaseDatabase. En Realtime Database Firebase obtenemos la referencia del nodo principal.  
 \*/  
 mAuth = FirebaseAuth.*getInstance*();  
 mDatabase = FirebaseDatabase.*getInstance*().getReference();  
 /\*  
 Utilizamos el findViewById para buscar una vista determinada por su id dentro de un layout especifico habiendo especificado el tipo ej EditText.  
 \*/  
 iImagen = (ImageView) findViewById(R.id.*imagen*);  
 bAdelante = (Button) findViewById(R.id.*adelante*);  
 bIzquierda = (Button) findViewById(R.id.*izquierda*);  
 bDerecha = (Button) findViewById(R.id.*derecha*);  
 bAtras = (Button) findViewById(R.id.*atras*);  
 bParar = (Button) findViewById(R.id.*parar*);  
 bStop = (Button) findViewById(R.id.*stop*);  
 /\*  
 Necesitamos la id del usuario para utilizar como clave unica en nuestra base de datos.  
 \*/  
 id = mAuth.getCurrentUser().getUid();  
 /\*  
 Nos situamos en el nodo Id e ingresamos los datos del usuario, y generamos un metodo si se realizaron correctamente.  
 \*/  
 mDatabase.child("Identificaciones").child(id).addValueEventListener(new ValueEventListener() {  
 @Override  
 public void onDataChange(@NonNull DataSnapshot snapshot) {  
 /\*  
 Obtenemos el contenido del nodo al cual hicimos referencia en nuestra Base de Dato y obtenemos los valores que necesitamos.  
 \*/  
 serie = snapshot.child("Serie").getValue().toString();  
 /\*  
 Generamos un HashMap, este HashMap almacenara la informacion de los movimientos del bogie.  
 \*/  
 Map<String, Object> map = new HashMap<>();  
 map.put("Avanzar", false);  
 map.put("Reverza", false);  
 map.put("Izquierda", false);  
 map.put("Derecha", false);  
 /\*  
 Nos situamos en el nodo Modo Manual e ingresamos los datos del HashMap.  
 \*/  
 mDatabase.child("Bogie").child(serie).child("Modo Manual").setValue(map);  
 /\*  
 Boton adelante, declaramos el metodo onClick, el cual le diremos que funciones debera realizar al clickear este boton.  
 \*/  
 bAdelante.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 /\*  
 Generamos un HashMap, este HashMap almacenara la informacion de los movimientos del bogie.  
 \*/  
 Map<String, Object> map = new HashMap<>();  
 map.put("Avanzar", true);  
 map.put("Reverza", false);  
 map.put("Izquierda", false);  
 map.put("Derecha", false);  
 /\*  
 Nos situamos en el nodo Modo Manual e ingresamos los datos del HashMap.  
 \*/  
 mDatabase.child("Bogie").child(serie).child("Modo Manual").setValue(map);  
 }  
 });  
 /\*  
 Boton Stop, declaramos el metodo onClick, el cual le diremos que funciones debera realizar al clickear este boton.  
 \*/  
 bStop.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 /\*  
 Generamos un HashMap, este HashMap almacenara la informacion de los movimientos del bogie.  
 \*/  
 Map<String, Object> map = new HashMap<>();  
 map.put("Avanzar", false);  
 map.put("Reverza", false);  
 map.put("Izquierda", false);  
 map.put("Derecha", false);  
 /\*  
 Nos situamos en el nodo Modo Manual e ingresamos los datos del HashMap.  
 \*/  
 mDatabase.child("Bogie").child(serie).child("Modo Manual").setValue(map);  
 }  
 });  
 /\*  
 Boton Atras, declaramos el metodo onClick, el cual le diremos que funciones debera realizar al clickear este boton.  
 \*/  
 bAtras.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 /\*  
 Generamos un HashMap, este HashMap almacenara la informacion de los movimientos del bogie.  
 \*/  
 Map<String, Boolean> map = new HashMap<>();  
 map.put("Avanzar", false);  
 map.put("Reverza", true);  
 map.put("Izquierda", false);  
 map.put("Derecha", false);  
 /\*  
 Nos situamos en el nodo Modo Manual e ingresamos los datos del HashMap.  
 \*/  
 mDatabase.child("Bogie").child(serie).child("Modo Manual").setValue(map);  
 }  
 });  
 /\*  
 Boton Derecha, declaramos el metodo onClick, el cual le diremos que funciones debera realizar al clickear este boton.  
 \*/  
 bDerecha.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 /\*  
 Generamos un HashMap, este HashMap almacenara la informacion de los movimientos del bogie.  
 \*/  
 Map<String, Boolean> map = new HashMap<>();  
 map.put("Avanzar", false);  
 map.put("Reverza", false);  
 map.put("Izquierda", false);  
 map.put("Derecha", true);  
 /\*  
 Nos situamos en el nodo Modo Manual e ingresamos los datos del HashMap.  
 \*/  
 mDatabase.child("Bogie").child(serie).child("Modo Manual").setValue(map);  
 }  
 });  
 /\*  
 Boton Izquierda, declaramos el metodo onClick, el cual le diremos que funciones debera realizar al clickear este boton.  
 \*/  
 bIzquierda.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 /\*  
 Generamos un HashMap, este HashMap almacenara la informacion de los movimientos del bogie.  
 \*/  
 Map<String, Boolean> map = new HashMap<>();  
 map.put("Avanzar", false);  
 map.put("Reverza", false);  
 map.put("Izquierda", true);  
 map.put("Derecha", false);  
 /\*  
 Nos situamos en el nodo Modo Manual e ingresamos los datos del HashMap.  
 \*/  
 mDatabase.child("Bogie").child(serie).child("Modo Manual").setValue(map);  
 }  
 });  
 /\*  
 Boton parar, declaramos el metodo onClick, el cual le diremos que funciones debera realizar al clickear este boton.  
 \*/  
 bParar.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 /\*  
 Lanzamos una nueva actividad, Intent describimos las actividades, primero en la que nos encontramos y luego a la cual queremos acceder.  
 Luego cerramos la actividad actual.  
 \*/  
 startActivities(new Intent[]{new Intent( ManualActivity.this, MenuActivity.class)});  
 finish();  
 }  
 });  
 }  
  
 @Override  
 public void onCancelled(@NonNull DatabaseError error) {  
  
 }  
 });  
  
 }  
 @Override  
 protected void onStart(){  
 super.onStart();  
 first.addValueEventListener(new ValueEventListener() {  
 @Override  
 public void onDataChange(@NonNull DataSnapshot snapshot) {  
 String link = snapshot.getValue(String.class);  
 Picasso.*get*().load(link).into(iImagen);  
 }  
 @Override  
 public void onCancelled(@NonNull DatabaseError error) {  
 }  
 });  
 }  
}

ControlActivity

Aca simplemente mostramos datos de la base de datos en textView y colocamos 2 botones para ver la grafica histórica de los datos obtenidos de los sensores.

public class DatabaseActivity extends AppCompatActivity {  
 /\*  
 Declaramos Botones y TextView.  
 \*/  
 private TextView tUsu;  
 private TextView tSer;  
 private TextView tLoc;  
 private TextView tAut;  
 private TextView tTem;  
 private TextView tHum;  
 private TextView tPre;  
 private Button bVol;  
 private Button bCopyright ;  
 private Button bGraficas;  
 /\*  
 Variables para manipular los datos de los EditText como Strings.  
 \*/  
 private String usu;  
 private String ser;  
 private String loc;  
 private String aut;  
 private String tem;  
 private String hum;  
 private String pre;  
 /\*  
 Instanciamos Firebase Authentication y Realtime Database  
 \*/  
 DatabaseReference mDatabase;  
 FirebaseAuth mAuth;  
 /\*  
 Metodo onCreate no solo es responsable de crear la actividad, sino que, tambien puede guardar el estado de la aplicacion utilizando Bundle  
 savedInstanceState. Básicamente, la clase Bundle se usa para almacenar los datos de la actividad siempre que ocurra la condición anterior  
 en la aplicación.  
 \*/  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 /\*  
 Metodos para mostrar el diseno en pantalla, donde R es recurso, layout diseno y activity\_reg es el archivo XML  
 \*/  
 setContentView(R.layout.*activity\_database*);  
 /\*  
 Inicializamos la intancia de FirebaseAuthentication y FirebaseDatabase. En Realtime Database Firebase obtenemos la referencia del nodo principal.  
 \*/  
 mAuth = FirebaseAuth.*getInstance*();  
 mDatabase = FirebaseDatabase.*getInstance*().getReference();  
 /\*  
 Utilizamos el findViewById para buscar una vista determinada por su id dentro de un layout especifico habiendo especificado el tipo ej EditText.  
 \*/  
 tUsu = (TextView) findViewById(R.id.*tUsua*);  
 tSer = (TextView) findViewById(R.id.*tSeri*);  
 tLoc = (TextView) findViewById(R.id.*tLoca*);  
 tAut = (TextView) findViewById(R.id.*tAuto*);  
 tTem = (TextView) findViewById(R.id.*tTemp*);  
 tHum = (TextView) findViewById(R.id.*tHume*);  
 tPre = (TextView) findViewById(R.id.*tPres*);  
 bGraficas = (Button) findViewById(R.id.*graficas*);  
 bVol = (Button) findViewById(R.id.*volver*);  
 bCopyright = (Button) findViewById(R.id.*copyright*);  
 /\*  
 Necesitamos la id del usuario para utilizar como clave unica en nuestra base de datos.  
 \*/  
 String id = mAuth.getCurrentUser().getUid();  
 /\*  
 Nos situamos en el nodo Id e ingresamos los datos del usuario, y generamos un metodo si se realizaron correctamente.  
 \*/  
 mDatabase.child("Identificaciones").child(id).addValueEventListener(new ValueEventListener() {  
 @Override  
 public void onDataChange(@NonNull DataSnapshot dataSnapshot) {  
 /\*  
 Obtenemos el contenido del nodo al cual hicimos referencia en nuestra Base de Dato y obtenemos los valores que necesitamos.  
 \*/  
 usu = dataSnapshot.child("Usuario").getValue().toString();  
 ser = dataSnapshot.child("Serie").getValue().toString();  
 loc = dataSnapshot.child("Localidad").getValue().toString();  
 /\*  
 Establecemos el texto escaneado en nuestro TextView.  
 \*/  
 tUsu.setText(usu);  
 tSer.setText(ser);  
 tLoc.setText(loc);  
 /\*  
 Nos situamos en el nodo Nro de serie de nuestro bogie para obtener los datos que necesitamos.  
 \*/  
 mDatabase.child("Bogie").child(ser).addValueEventListener(new ValueEventListener() {  
 @Override  
 public void onDataChange(@NonNull DataSnapshot dataSnapshot) {  
 /\*  
 Obtenemos el contenido del nodo al cual hicimos referencia en nuestra Base de Dato y obtenemos los valores que necesitamos.  
 \*/  
 aut = dataSnapshot.child("Automatico").child("Encendido").getValue().toString();  
 /\*  
 Establecemos el texto escaneado en nuestro TextView.  
 \*/  
 tAut.setText(aut);  
  
 }  
 @Override  
 public void onCancelled(@NonNull DatabaseError error) {  
 }  
 });  
 }  
 @Override  
 public void onCancelled(@NonNull DatabaseError error) {  
 }  
 });  
 /\*  
 Boton Registrar, declaramos el metodo onClick, el cual le diremos que funciones debera realizar al clickear este boton.  
 \*/  
 bVol.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 /\*  
 Lanzamos una nueva actividad, Intent describimos las actividades, primero en la que nos encontramos y luego a la cual queremos acceder.  
 Luego cerramos la actividad actual.  
 \*/  
 startActivities(new Intent[]{new Intent(DatabaseActivity.this, MenuActivity.class)});  
 finish();  
 }  
 });  
 /\*  
 Boton Copyright, declaramos el metodo onClick, el cual le diremos que funciones debera realizar al clickear este boton.  
 \*/  
 bCopyright.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 /\*  
 Mostramos mensaje  
 \*/  
 Toast.*makeText*(DatabaseActivity.this, "UADER FCYT - 2020", Toast.*LENGTH\_SHORT*).show();  
 }  
 });  
 /\*  
 Boton Graficas, declaramos el metodo onClick, el cual le diremos que funciones debera realizar al clickear este boton.  
 \*/  
 bGraficas.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 /\*  
 Lanzamos una nueva actividad, Intent describimos las actividades, primero en la que nos encontramos y luego a la cual queremos acceder.  
 Luego cerramos la actividad actual.  
 \*/  
 startActivities(new Intent[]{new Intent(DatabaseActivity.this, GraficaActivity.class)});  
 finish();  
 }  
 });  
  
  
 }  
}

GraficaActivity

Aca a con la librería LineChart generamos gráficos de los datos que obtuvimos.

package com.example.controlbogie;  
/\*  
 Autores: Roldan Federico, Marandino Juan, Schoulteis Joaquin.  
 Materia: Proyecto de Sistemas Digitales  
 Profesor: Gerard Guillermo.  
 Universidad: Universidad Autonoma de Entre Rios - UADER  
 Facultad: Facultad de Ciencia y Tecnologia - FCYT  
 Año: 2020  
\*/  
import androidx.appcompat.app.AppCompatActivity;  
  
import android.content.Intent;  
import android.graphics.Color;  
import android.os.Bundle;  
import android.provider.ContactsContract;  
import android.view.View;  
import android.widget.Button;  
  
import com.github.mikephil.charting.charts.LineChart;  
import com.github.mikephil.charting.data.Entry;  
import com.github.mikephil.charting.data.LineData;  
import com.github.mikephil.charting.data.LineDataSet;  
import com.github.mikephil.charting.interfaces.datasets.ILineDataSet;  
import com.google.firebase.database.DatabaseReference;  
  
import java.util.ArrayList;  
import java.util.Map;  
  
public class GraficaActivity extends AppCompatActivity {  
 /\*  
 Declaramos Botones y ImageView.  
 \*/  
 private Button bVolver;  
 /\*  
 Instanciamos LineChart  
 \*/  
 LineChart mpLineChart;  
 /\*  
 Metodo onCreate no solo es responsable de crear la actividad, sino que, tambien puede guardar el estado de la aplicacion utilizando Bundle  
 savedInstanceState. Básicamente, la clase Bundle se usa para almacenar los datos de la actividad siempre que ocurra la condición anterior  
 en la aplicación.  
 \*/  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 /\*  
 Metodos para mostrar el diseno en pantalla, donde R es recurso, layout diseno y activity\_reg es el archivo XML  
 \*/  
 setContentView(R.layout.*activity\_grafica*);  
 /\*  
 Utilizamos el findViewById para buscar una vista determinada por su id dentro de un layout especifico habiendo especificado el tipo ej EditText.  
 \*/  
 mpLineChart = findViewById(R.id.*linechart*);  
 bVolver = (Button) findViewById(R.id.*volver*);  
 /\*  
 Boton Volver, declaramos el metodo onClick, el cual le diremos que funciones debera realizar al clickear este boton.  
 \*/  
 bVolver.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 /\*  
 Lanzamos una nueva actividad, Intent describimos las actividades, primero en la que nos encontramos y luego a la cual queremos acceder.  
 Luego cerramos la actividad actual.  
 \*/  
 startActivities(new Intent[]{new Intent(GraficaActivity.this, DatabaseActivity.class)});  
 finish();  
 }  
 });  
  
 LineDataSet lineDataSet1 = new LineDataSet(dataValues1(),"Temperatura");  
 LineDataSet lineDataSet2 = new LineDataSet(dataValues2(),"Humedad");  
 LineDataSet lineDataSet3 = new LineDataSet(dataValues3(),"Presion");  
 lineDataSet1.setColor(Color.*rgb*(255, 87,34));  
 lineDataSet2.setColor(Color.*rgb*(3,169,244));  
 lineDataSet3.setColor(Color.*rgb*(139, 195,74));  
 ArrayList<ILineDataSet> dataSets = new ArrayList<>();  
 dataSets.add(lineDataSet1);  
 dataSets.add(lineDataSet2);  
 dataSets.add(lineDataSet3);  
 mpLineChart.setDrawBorders(true);  
 LineData data = new LineData(dataSets);  
 mpLineChart.setData(data);  
 mpLineChart.invalidate();  
  
  
 }  
 private ArrayList<Entry> dataValues1()  
 {  
 ArrayList<Entry> datosTemp = new ArrayList<Entry>();  
 datosTemp.add(new Entry(0,15));  
 datosTemp.add(new Entry(1,14));  
 datosTemp.add(new Entry(2,15));  
 datosTemp.add(new Entry(3,15));  
 datosTemp.add(new Entry(4,16));  
 datosTemp.add(new Entry(5,17));  
 datosTemp.add(new Entry(6,18));  
 datosTemp.add(new Entry(7,19));  
 datosTemp.add(new Entry(8,20));  
 datosTemp.add(new Entry(9,22));  
 datosTemp.add(new Entry(10,22));  
 datosTemp.add(new Entry(11,25));  
 datosTemp.add(new Entry(12,28));  
  
 return datosTemp;  
 }  
 private ArrayList<Entry> dataValues2()  
 {  
 ArrayList<Entry> datosHum = new ArrayList<Entry>();  
 datosHum.add(new Entry(0,90));  
 datosHum.add(new Entry(1,89));  
 datosHum.add(new Entry(2,93));  
 datosHum.add(new Entry(3,95));  
 datosHum.add(new Entry(4,90));  
 datosHum.add(new Entry(5,89));  
 datosHum.add(new Entry(6,93));  
 datosHum.add(new Entry(7,95));  
 datosHum.add(new Entry(8,90));  
 datosHum.add(new Entry(9,89));  
 datosHum.add(new Entry(10,93));  
 datosHum.add(new Entry(11,95));  
 datosHum.add(new Entry(12,90));  
  
 return datosHum;  
 }  
 private ArrayList<Entry> dataValues3()  
 {  
 ArrayList<Entry> datosPre = new ArrayList<Entry>();  
 datosPre.add(new Entry(0,95));  
 datosPre.add(new Entry(1,98));  
 datosPre.add(new Entry(2,101));  
 datosPre.add(new Entry(3,95));  
 datosPre.add(new Entry(4,99));  
 datosPre.add(new Entry(5,102));  
 datosPre.add(new Entry(6,96));  
 datosPre.add(new Entry(7,105));  
 datosPre.add(new Entry(8,102));  
 datosPre.add(new Entry(9,96));  
 datosPre.add(new Entry(10,105));  
 datosPre.add(new Entry(11,102));  
 datosPre.add(new Entry(12,96));  
  
 return datosPre;  
 }  
}

GraficaActivity

Este archivo se ejecuta en nuestra raspberry pi, se conecta con nuestra base de datos de firebase y obtiene los datos que nos interesan, en este caso, el true del Encender. Esto lo ejecutamos en un ciclo infinito while.

#!/usr/bin/python

# -\*- coding: utf-8 -\*-

from time import sleep                 #importamos del modulo time la funcion sleep

from picamera import PiCameraa         #importamos del modulo picamera la funcion Picameraa

import RPi.GPIO as GPIO                #importamos la lireria RPi.GPIO para manejar los GPIO desde Python

import pyrebase                        #importamos el contenedor de python para la API de firebase

#Autenticación basada en el usuario

config = {

    "apiKey" : "AIzaSyCrRhi2PbwqbrL\_c\_T85jJuoK-4U5MPdlY",

    "authDomain": "sistemashttps://sistemasbogie.firebaseio.combogie.firebaseapp.com",

    "databaseURL": "",

    "storageBucket":"sistemasbogie.appspot.com"

    }

firebase = pyrebase.initialize\_app(config)

#Configuramos y seteamos el pin GPIO17 como salida

GPIO.setmode(GPIO.BCM)

GPIO.setup(17,GPIO.OUT)

#Obtenemos una referencia al servicio de base de datos y al servicio de almacenamiento

db = firebase.database()

storage = firebase.storage()

#Lista de ID que podran manejar esta RaspberryPi

listaId = ["RxrTAzAD4abRWdL9D8Km3R6FvDz2", "Prueba"]

#Iniciamos siempre con el Bogie Encendido en False

data ={

            "Bogie":{

                "Encendido" : False

                },

    }

#Recorremos la lista de ID generadas por nuestra Database

for i in listaId:

  s = ""

  s = i

#Subimos los datos a nuestra base de datos

result = db.child("Usuario").child(s).update(data)

while True:

    #Preguntamos si algun usuario de esta raspberry a encendido el bogie

    mostrar = db.child("Usuario").child("RxrTAzAD4abRWdL9D8Km3R6FvDz2").child("Bogie").get()

    #Devuelve una lista de objetos en cada uno de los cuales se puede llamar y obtener los datos

    for raspi in mostrar.each():

        print(raspi.val())

        if (raspi.val()):

            GPIO.output(17,GPIO.HIGH)                       #si se cumple condicional,seteamos en HIGH pin GPIO17

            print('bogie encendido')

            with Picamara.PiCamara() as camara:             #Se construiye el objeto Picamara

                camara.start\_preview()                      #Este método inicia una vista previa de la cámara

                camara.resolution = 'VGA'                   #Establecemos la resolución con la que se capturarán las imágenes

                time.sleep(2)                               #Se suspende la ejecución durante un número dado de segundos

                camara.capture('foto.png', format='png', use\_video\_port=true)  #Capturamos una imagen, almacenándola en la salida.

                camara.stop\_preview()                       #Oculta la superposición de vista previa

                camara.close()                              #Finaliza el estado de la cámara

            #Creamos ruta de acceso a los datos con el servicio de almacenamiento y se toma la ruta de acceso al archivo local

            storage.child("Imagenes/2.png").put("foto.png")

            time.sleep(3)                                   #Se suspende la ejecución durante un número dado de segundos

        else:

            GPIO.output(17, GPIO.LOW)                       #si no se cumple condicional,seteamos en LOW pin GPIO17

            print('bogie apagado')

GPIO.clear()                                                #Se limpian los puertos utilizados