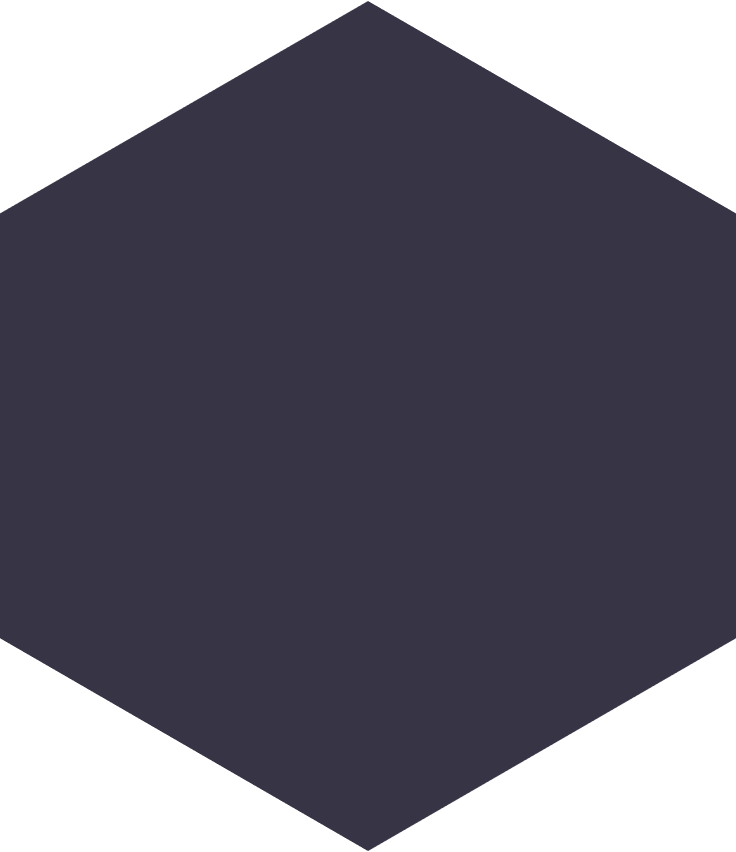
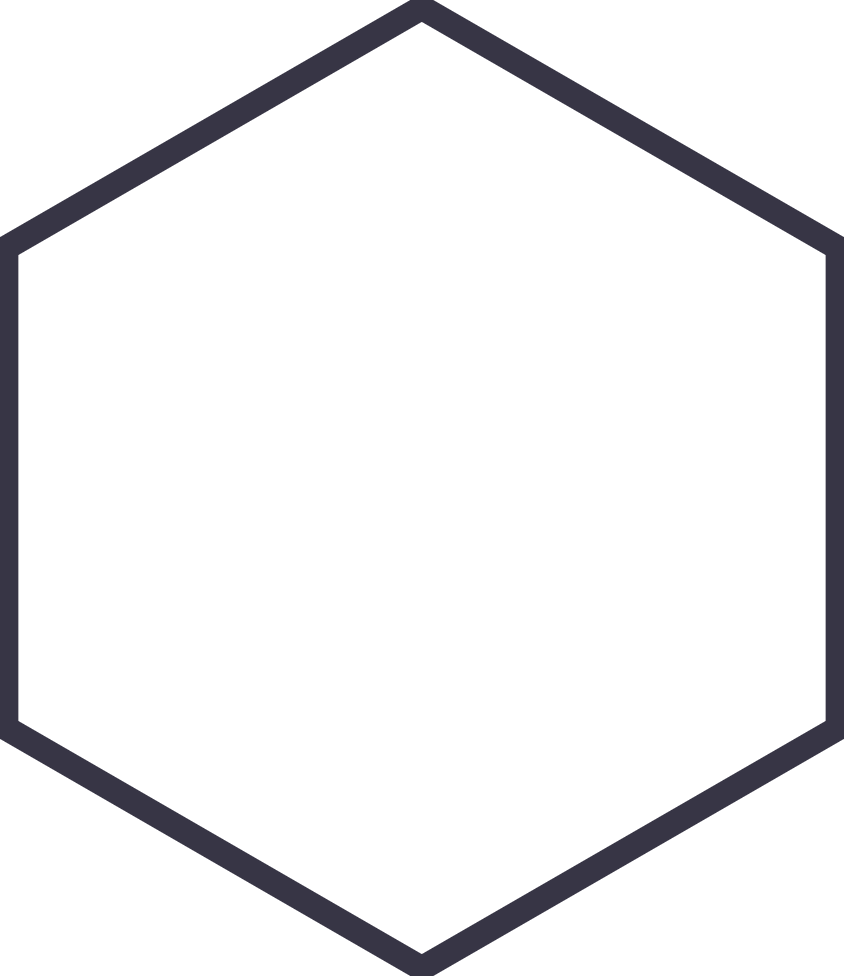
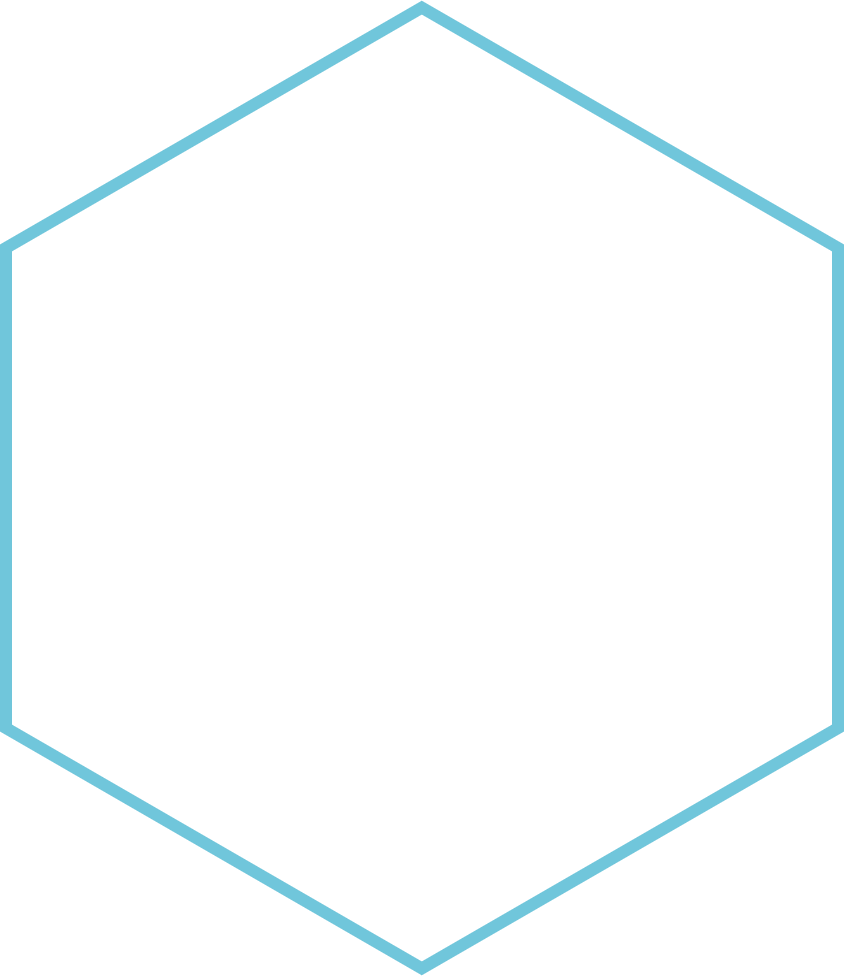


|  |
| --- |
| **MANUAL TECNICO** |
| AUTORES:  ELIZABETH MARTÍNEZ  JORGE PINARGOTE  KEVIN LÓPEZ |
|  |
|  |

**Versión 1**

12/09/2020

APLICACIONES MÓVILES Y SERVICIOS TELEMÁTICOS



# Introducción

El presente manual técnico tiene como finalidad describir el diseño del prototipo para la gestión de i-Tank en ambientes industriales, así como también la aplicación nativa de Android. La implementación de i-Tank se basa en teléfonos móviles ya que se ha comprobado que conforme avanza la tecnología estos dispositivos cada vez adquieren mejor funcionamiento y mayor adaptabilidad a las diferentes tecnologías. Por ello muchas personas han preferido este tipo de dispositivos ya que son más prácticos y fácil de llevar debido a su tamaño y peso.

La aplicación nativa de Android funciona como un método de consulta de la información por lado del servidor, así como también se toma en cuenta la utilización de la misma como responsabilidad de cada usuario. El desarrollo de este sistema se basa en un conjunto de Hardware y Software totalmente integrado con comunicaciones mediante redes LPWAN y acciones directas con el usuario.

Finalmente, i-Tank es un aplicativo diseñado para los restaurantes que poseen varios tanques de gas en el área de la cocina, los cuales se tienen que reponer al momento de que se agota el gas. Esto se debe a que los chefs no podrían conocer que tanque de gas está vacío hasta que prueban encender una hornilla sin éxito.

# Pre-requisitos

## Hardware

Velostat

Este material conductor también conocido como Linqstat es sensible a la presión ya que al aplicarle una fuerza o doblarlo su resistencia cambia. Material de embalaje hecho de una lámina polimérica (poli olefinas) impregnada con negro de carbón para que sea eléctricamente conductor.

Figura 1. Velostat

Arduino Mega

El Arduino Mega posee 54 pines digitales que funcionan como entrada/salida; 16 entradas análogas, un cristal oscilador de 16 MHz, una conexión USB, un botón de reset y una entrada para la alimentación de la placa.

La comunicación entre la computadora y Arduino se produce a través del Puerto Serie. Posee un convertidor usb-serie, por lo que sólo se necesita conectar el dispositivo a la computadora utilizando un cable USB como el que utilizan las impresoras.

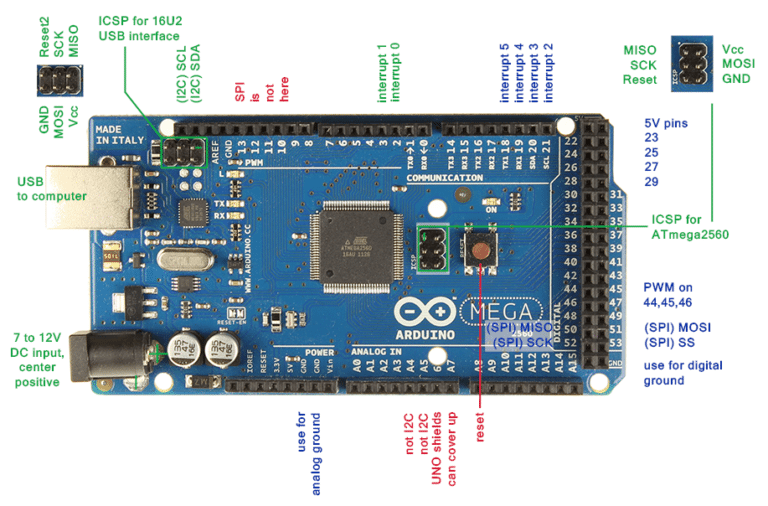
[](http://panamahitek.com/wp-content/uploads/2013/01/Mega2560_R3_Label-small-v2-2.jpg)

Figura 2. Arduino Mega

Sigfox Kit

Sigfox es una tecnología complementaria de bajo costo y poco consumo de energía, considerada como parte de una red LPWAN. Sigfox es compatible con Bluetooth, GPS 2G / 3G / 4G y Wifi. Al combinar otras soluciones de conectividad con Sigfox, los casos de negocios y la experiencia del usuario se pueden mejorar drásticamente.

La solución de conectividad única proporciona el dispositivo a la nube simplificar las comunicaciones, permitimos un inmejorable bajo consumo de energía pudiendo evitar la necesidad de reemplazar o recargar las baterías, ya que los dispositivos generarán energía por sí mismos.

Figura 3. Sigfox Radio communication Kit for Arduino

Elementos electrónicos

Dentro de esta categoría necesitaremos elementos básicos de electrónica como:

* Componentes Pasivos (Resistencias, Potenciómetros, Switch, Capacitor, Cables de conexión)
* Componentes activos (Fuente de alimentación para simular batería)

## Software

IDE: Android Developer Tools Plugin IDE

Requisitos Sistema Operativo (Android Studio)

|  |  |  |
| --- | --- | --- |
|  | Windows | OS X/macOS |
| OS version | Windows 10/8/7 (32- o 64-bit) | Mac OS X 10.10 (Yosemite) o superior, hasta 10.13 ( macOS High Sierra) |

Manejador de Base de Datos: MySQL, Django.

Lenguaje de Programación: XML, HTML, JavaScript y Java JDK 7.

Teléfono Celular: Sistema operativo Android 4.1 JellyBean en adelante.

Sigfox platform

En los proyectos que involucran el IoT, la conectividad al internet es la parte principal y actualmente Sigfox es el medio por el que se envía cualquier tipo de dato e información. Esta plataforma proporciona conectividad inalámbrica y fue creada para que funcione e interactúe con dispositivos de bajo consumo energético - tales como sensores que funcionan con pilas convencionales - con tasas de transferencias de datos de hasta 12 bytes y un máximo número de mensajes o peticiones de 140 diarias.

Spring

Spring facilita la integración con los servicios web alojados de Amazon. Ofrece una forma conveniente de interactuar con los servicios proporcionados por AWS utilizando expresiones idiomáticas y API conocidas de Spring, como la API de mensajería o almacenamiento en caché. Los desarrolladores pueden construir su aplicación alrededor de los servicios alojados sin tener que preocuparse por la infraestructura o el mantenimiento.

Django

Django es un framework de aplicaciones web gratuito y de código abierto (open source) escrito en Python. Un framework web es un conjunto de componentes que te ayudan a desarrollar sitios web más fácil y rápidamente.

# Diseño

## Hardware

El diseño final de este proyecto tuvo que pasar por diferentes fases, especialmente en la parte del sensor sensible a la presión(velostat).

## Prototipo 1

El primer diseño del prototipo se realizó de la siguiente forma:

**Materiales**

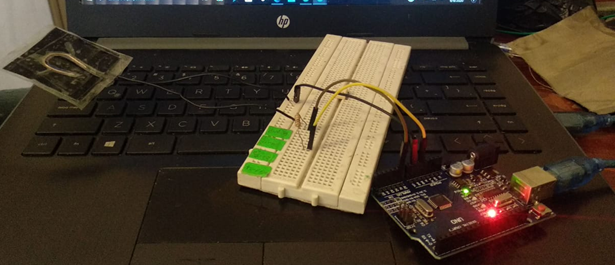
* Velostat tamaño 5x5cm
* Cinta de cobre adhesiva tamaño 6x6cm
* Cables de cobre

**Descripción de las capas a realizar:**

Se realiza una topología tipo sándwich como se observa en la ilustración 1, para armar las capas del sensor sensible a la presión, en este caso se realizó dos veces, uno para cada cable. Nota: el cable debe hacer contacto con el velostat, una ves realizado el contacto, se procede a colocar las capas de cinta de cobre.

*Ilustración 1 Topología sándwich.*

Una vez realizado los dos diseños, se procede a unirlos, y se le agrega una capa más de cinta para mantener unido al velostat, como se muestra en la ilustración 2.



|  |
| --- |
| **Ilustración 2: Velostat con divisor de voltaje** |

Tambien se observa en la ilustración 2, que un cable va al gnd y otro en serie con otra resistencia, para formar un divisor de voltaje, y adicionalmente se puntea el divisor de voltaje a una de las entradas analógicas del Arduino, para censar los datos.

## Prototipo 2

El segundo intento del prototipo se realizó con los siguientes materiales:

**Materiales**

* Velostat tamaño 5x15cm
* Cinta aislante
* Cable calibre 22
* Algodón
* Vaquelita 10x10 cm
* Algodón 9x9 cm
* Alfombra roja 20x20 cm
* Aluminio 5x15 cm

Esta vez se tiene mayor área de velostat, y lo que se procede hacer que la parte mas larga se la divide en 3 secciones, pero no se la recorta, si no que se la dobla , prácticamente se obtiene un velostat de 5x5cm, pero con 3 areas o secciones. En la parte central de las secciones se procede a colocar el calibre de cobre, se le agrega cinta para mantenerlo fijo. Despues se procede a cubrir el velostat, con papel aluminio, con la misma dimensión del velostat para mantenerlo lo mas liso o plano posible. El velostat cubierto de aluminio se procede agregarlo en la base de baquelita, solo se observa en la ilustración 3.



En la ilustración 3, también se observa una capa de algodón, esta se la coloca encima del velostat, y encima del algodón se coloca la otra lamina de vaquelita, con cinta se procede ajustar los extremos del velostat, para mantenerlo fijo, una vez realizado todo este proceso, se cubre el velostat con la alfombra, lo ideal es que el velostat quede cubierto para evitar que la frecuencia del ruido interfieran o afecten la sensibilidad del velostat.



En la ilustración 4, se observa cómo queda este diseño, y el circuito para leer los datos, es el mismo al primer diseño.

## Prototipo final

El prototipo final tomando en consideración las anteriores observaciones se optaron como primero poner una base de madera donde estaré apoyado y inmovilizado con cinta aislante a nuestro pequeño sensor.

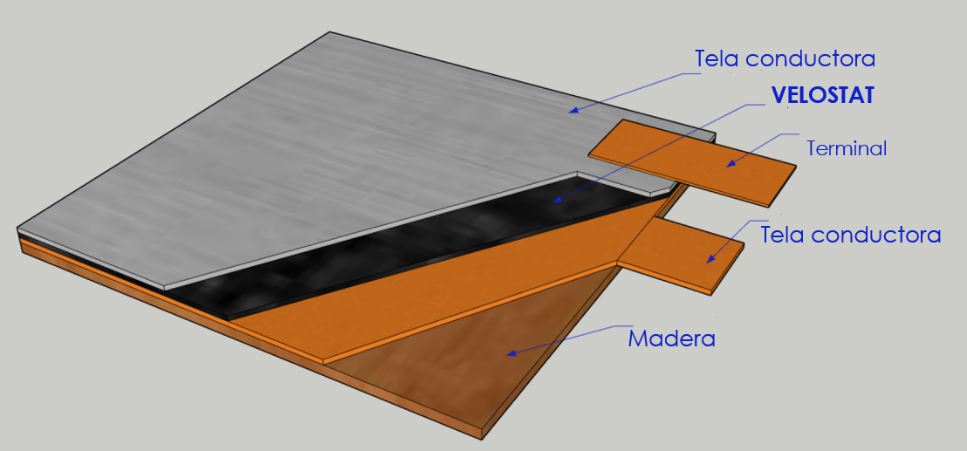
Los materiales usados en el siguiente prototipo exitoso para el proyecto fueron:

**Materiales**

* 1 Pedazo de Velostat tamaño 3x3cm
* 2 Pedazos de tela conductora 2x2cm
* Cinta aislante
* Cable calibre 24
* Base de madera 10x10cm (puede ser una base más pequeña)
* Recubierta de vidrio 10x10cm (puede ser una base más pequeña)

**Descripción de las capas a realizar:**

Teniendo en cuenta que la topología del sensor sigue siendo de sándwich, se puede observar que ahora la capa conductora será 2 trozos de tela conductora como se puede visualizar en la siguiente imagen.



*Ilustración 6 Capas del prototipo final.*

Los cables de calibre 24 que van entre las capas de tela conductora y el velostat vienen colocado de forma espiral tratando que cubran la mayor cantidad de superficie.

Una vez armado el sándwich se fija todo, poniendo trozos de cinta aislante alrededor del sensor, como muestra la siguiente imagen.

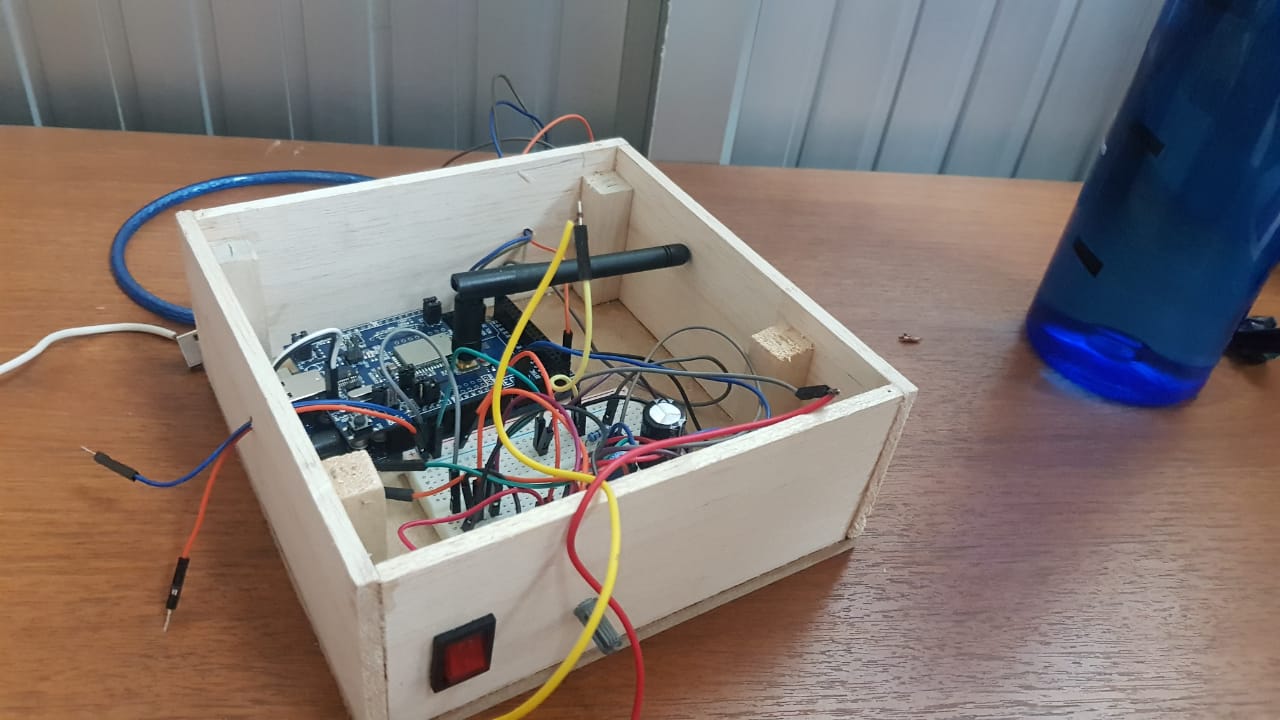


*Ilustración 7 El prototipo armado con sus respectivas capas.*

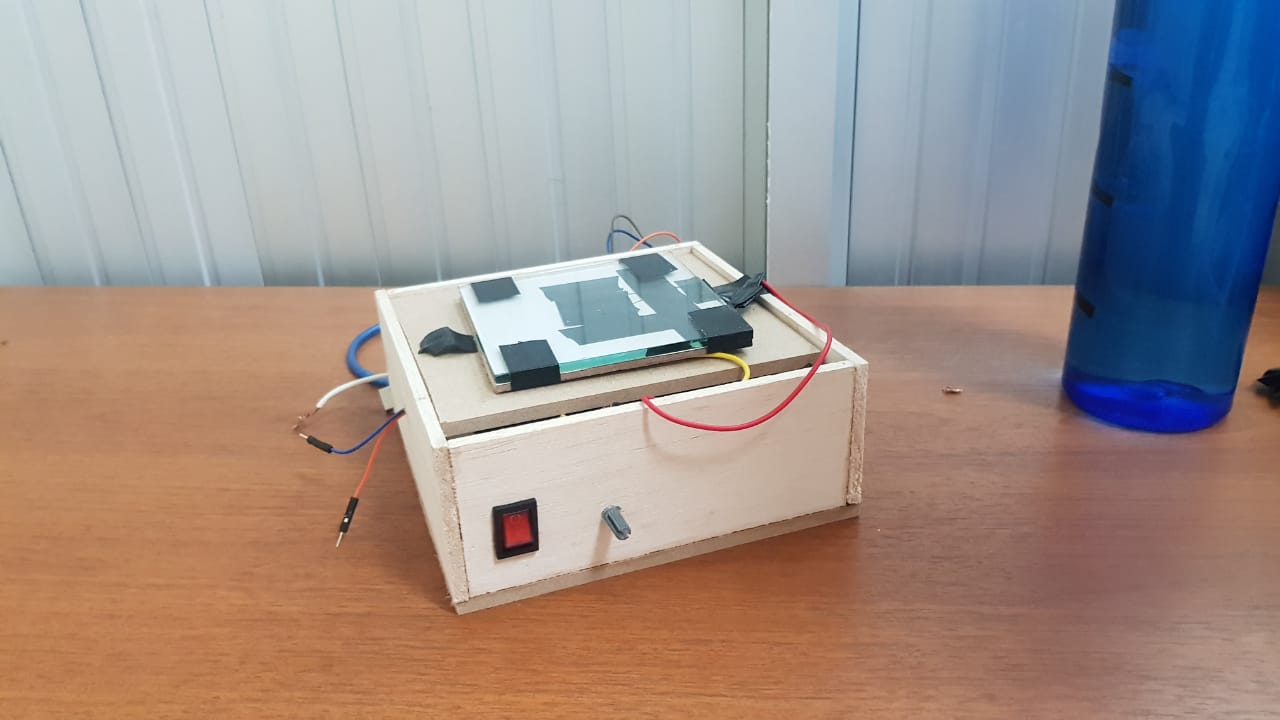
En la base de madera vendrá fijado el sándwich con trozos de cinta aislante y recubierto con el trozo de vidrio que también vendrá fijado con cinta.



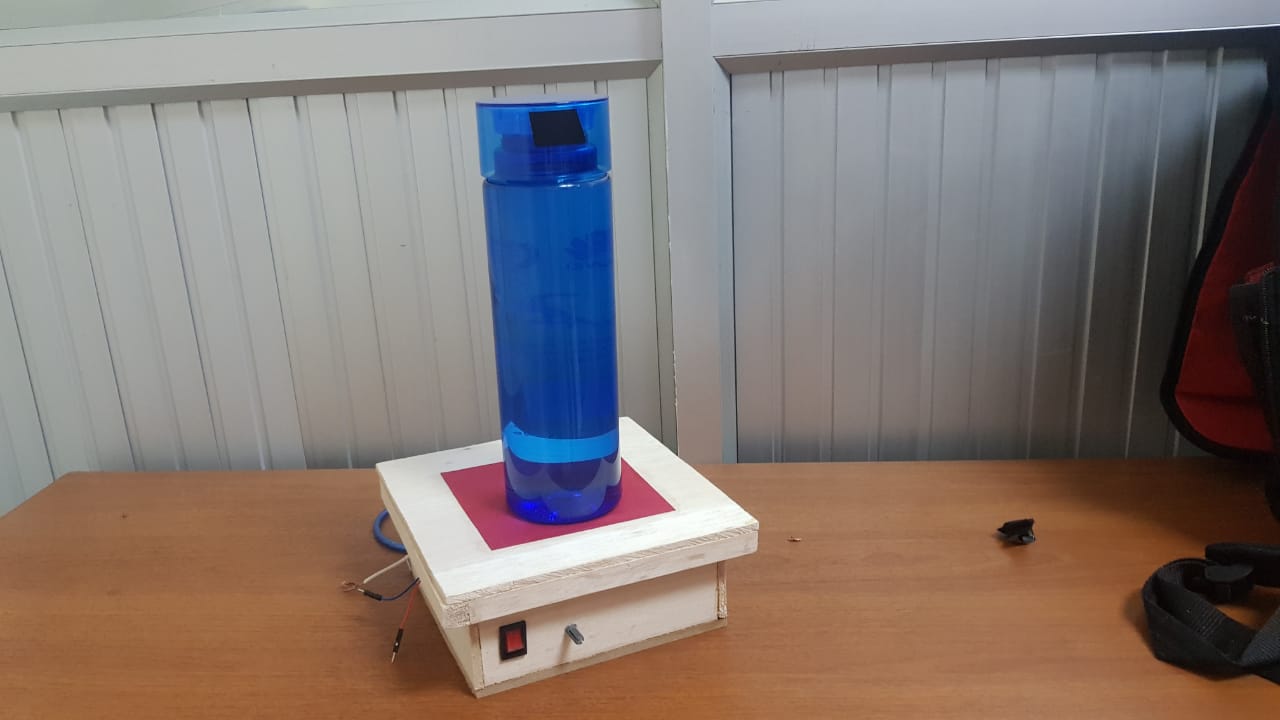
*Ilustración 8 El prototipo del sensor terminado con su base de madera y recubierta de vidrio.*



*Ilustración 8 Circuito en su respectiva caja.*

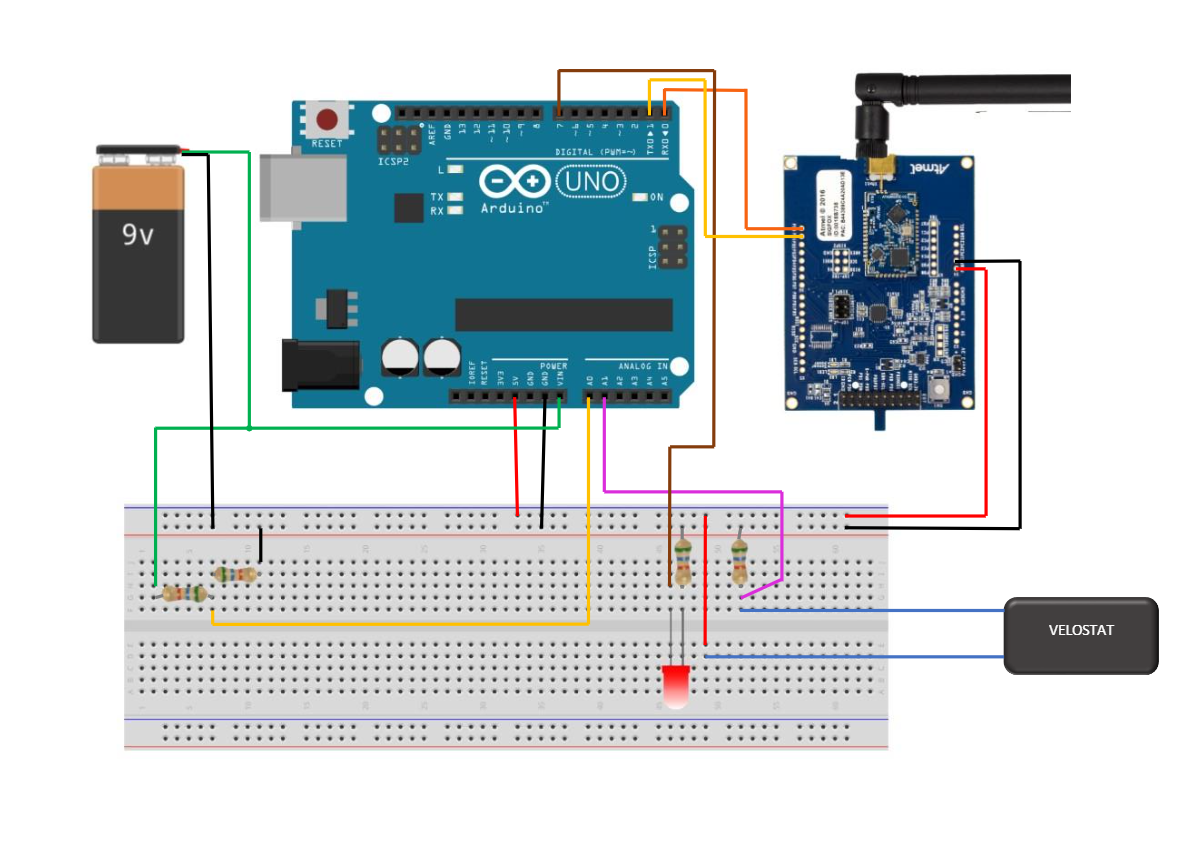


*Ilustración 8 Sensor puesto en la caja.*



*Ilustración 8 Contenedor del sensor cerrado y listo para el uso.*

## Esquemático de conexión Hardware



# Software

## Sigfox

Sigfox es una herramienta usada en el prototipo de nuestro proyecto como plataforma fundamental para la conexión entre nuestro dispositivo IoT y nuestra nube de datos. Este dispositivo como integración de Hardware y Software obtiene lecturas del sensor (Velostat) y las envía mediante un Callback hacia un servidor MySQL de Spring con la siguiente configuración.

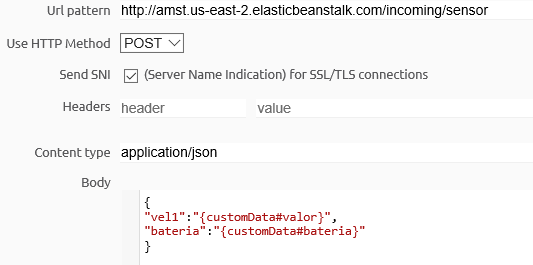


Figura. Configuracion del Backend de Sigfox para Callback de nuestro proyecto.

Obtiene lecturas del nivel de batería.

Módulo de Comunicaciones (THINXTRA).

Opción de Debug.

|  |
| --- |
| **// These constants won't change. They're used to give names to the pins used:**  **const int analogInPin = A8;**  **const int analogBateria = A9;// Analog input pin that the potentiometer is attached to**  **int sensorValue = 0;**  **int flag = 0;**  **int bateria = 0;**  **int copiaSensor = 0; // value read from the pot**  **void setup() {**  **Serial.begin(9600);**  **Serial1.begin(19200);**  **}**  **void loop() {**  **sensorValue = analogRead(analogInPin);**  **while(1){**  **copiaSensor = analogRead(analogInPin);**  **bateria = analogRead(analogBateria);**  **bateria= map(bateria,400,1023,0,100);**    **Serial1.print(sensorValue);**  **Serial1.print(",");**  **Serial1.print(copiaSensor);**  **Serial1.print(",");**  **Serial1.println(bateria);**  **if (copiaSensor%10==0){**  **break;**  **}else if (bateria < 50 ){**  **if (flag == 0){**  **flag = 1;**  **break;**  **}**  **}**  **delay(100);**  **}**  **sensorValue = analogRead(analogInPin);**  **bateria = analogRead(analogBateria);**  **bateria= map(bateria,400,1023,0,100);**  **sendMessage(sensorValue,bateria);**  **delay(5000);**  **}**  **void sendMessage(int sensor, char bateria){**  **Serial.println("AT$RC");**  **Serial.print("AT$SF=");**  **if(sensorValue<16)Serial.print("000");**  **else if(sensorValue<256)Serial.print("00");**  **else if(sensorValue<4096)Serial.print("0");**  **Serial.print(sensorValue,HEX);**  **if (bateria < 16)Serial.print("0");**  **Serial.println(bateria,HEX);**  **}**  **}** |

Spring

Maneja datos envíados desde el Callback de Sigfox.

Permite configuración de umbrales de decisión.

Envía peticiones para el servicio de notificaciones (FIREBASE).

Registra datos en API del curso (Django).

Registra datos en base de datos externa.

Django

Para el desarrollo de este proyecto se usa Django como API REST de conexión y como única interacción del usuario con la red. Se genera un registro sobre información del estado del tanque de gas con el siguiente formato.

|  |
| --- |
| {  "id": 906,  "fechaRegistro": "2019-08-13T09:29:41.862963-05:00",  "estado": "ME",  "tanque": 2  } |

Donde se encuentra los diferentes estados para el tanque de gas:

ES à Estable

ME à Medio

VA à Vacío

i-Tank

Inicio de session mediante usuarios registrados.

Muestra nivel de tanques registrados.

Muestra datos históricos de los tanques.

Muestra estadisticas del estado de los tanques.

Implementa servicio de notificaciones por FIREBASE.

Consume data del servidor de Django.

Firebase

Se utilizó la infraestructura de Firebase Google como informes de fallas y mensajería, de manera que puedas ser más eficiente y enfocarse en mantener actulizado a los usuarios con respecto al estado de sus tanques de gas incluso si no se consulta en el aplicativo móvil.

# Documentación API del servicio SPRING

Para evitar conflictos entre el callback generado por el backend de Sigfox y el API proporcionado para el desarrollo del proyecto en Django se procedió a crear un servicio intermedio al cual le lleguen todos los datos generados por el Callback y de acuerdo a parámetros de configuración establecidos se generen HTTP REQUEST de acuerdo a cada una de las funcionalidades del proyecto.

El servicio de SPRING se ejecuta en un contenedor de Amazon Web Services: ElasticBeanstalk, el cual permite hacer un deployment rápido de servicios basados en JAVA, .NET, Django, etc.

Este servicio al estarse ejecutando para un ambiente de pruebas de la aplicación aplicar para el servicio Free Tier de AWS, donde gratuitamente proporcionan servicios con un limite de uso tanto en peticiones HTTP, almacenamiento y consumo de recursos virtuales de maquinas EC2.



Ilustración Recursos consumidos durante 1 mes de prueba de aplicación

Al ser SPRING un framework para desarrollo rápido de aplicaciones que proporciona un ambiente web (TOMCAT) con conectores para bases de datos y otras opciones de *fast deployment* al momento de probar localmente todas las funcionalidades de los APIs construidos se genera el archivo *.jar* el cual será cargado en un ambiente virtual (llamado envioroment en AWS) con las configuraciones necesarias para que pueda correr autónomamente el servicio.

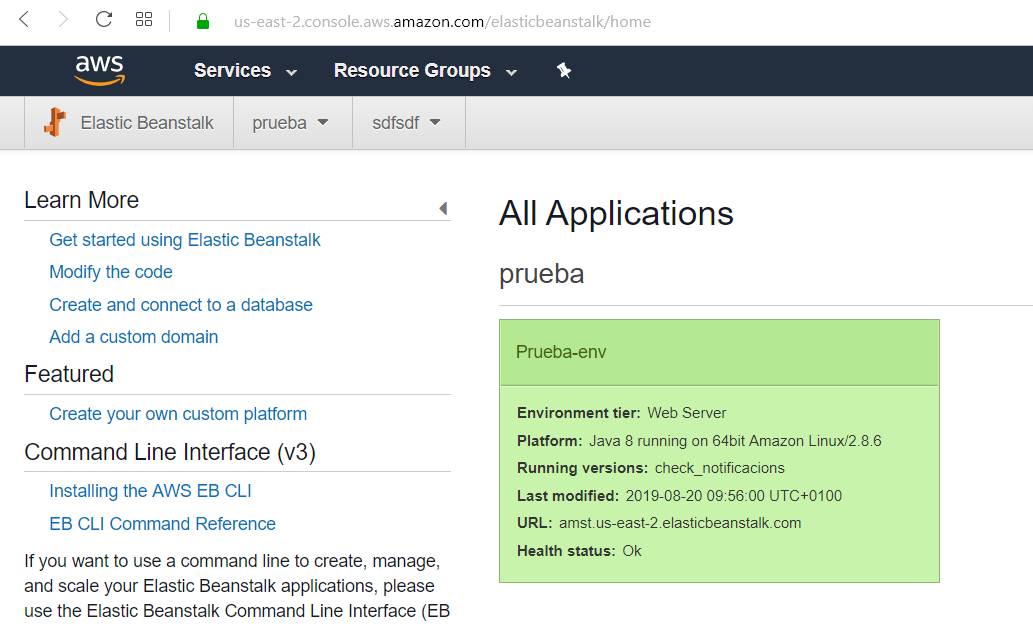


Ilustración Ambiente de Free Tier AWS

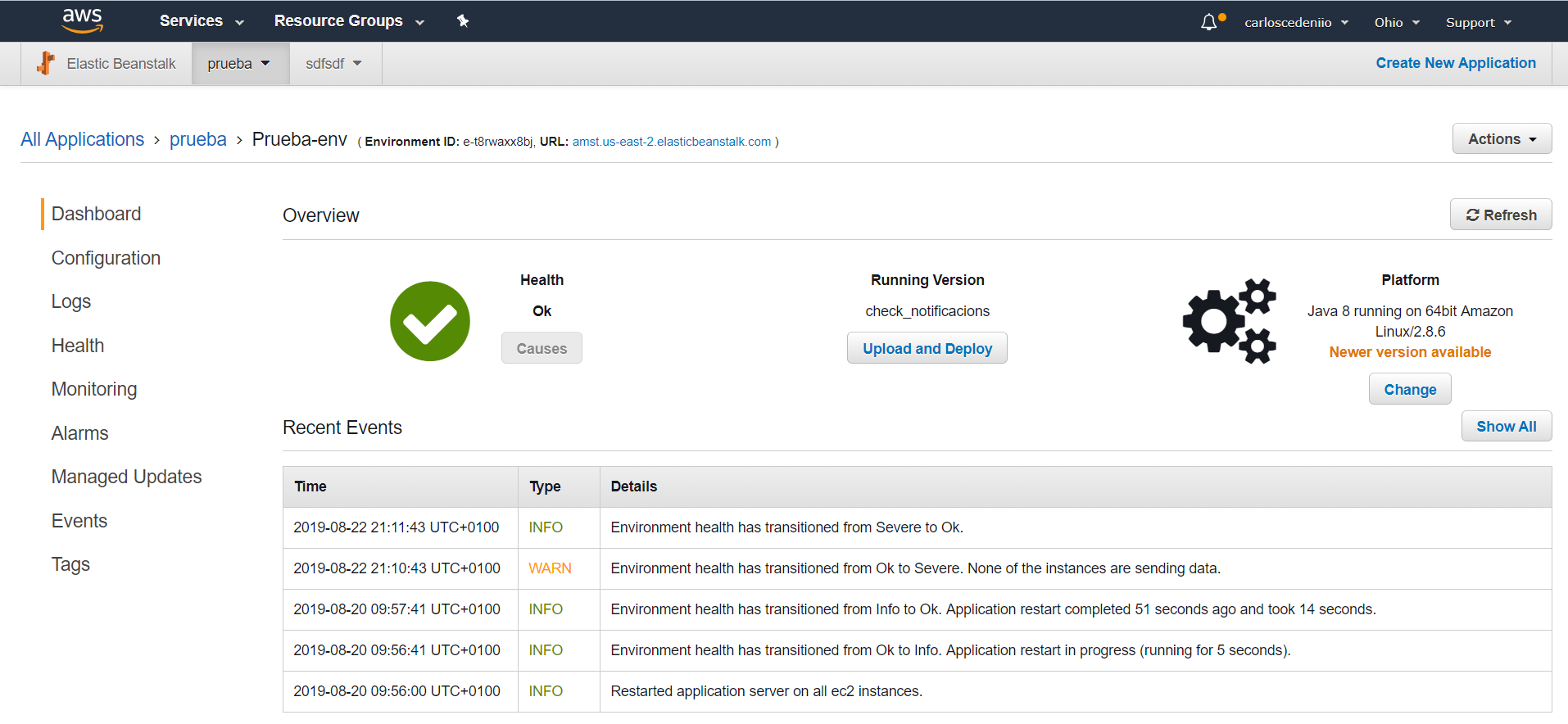


Ilustración Contenedor de JAVA para SPRING

A continuación se van a detallar los APIs proporcionados.

1. Registro de nueva data desde callback de Sigfox

|  |  |
| --- | --- |
| **URL:** http://amst.us-east-2.elasticbeanstalk.com/incoming/sensor  **Method:** POST  **Descripción:** Se requiere que en el body se ingrese un Json de tipo Sensor (descrito abajo). Crea un nuevo elemento Sensor y lo ingresa al repositorio remoto (MySql).  Ingresa el nivel presión del velostat en Django con su equivalencia. Ingresa el nivel de la batería en Django. Si el nivel de la batería recibida es menor que el umbral establecido previamente inicia un Request hacia el servicio de Firebase para enviar notificaciones automaticas. | |
|  | @PostMapping("/sensor")  public Sensor createNote(@Valid @RequestBody Sensor sensor) {  Sensor actual = noteRepository.save(sensor);  ingress(actual.getVel1());  ingressBatery(actual.getBateria());  if (Integer.parseInt(actual.getBateria())<= NoteController.bateria){  setNotification(actual.getBateria());  }  return actual;  } |
| E.g | {  "vel1":"180",  "bateria":"30"  }  #Donde vel1 es un entero.  #Donde batería es un entero |

1. Crear notificaciones de Firebase

|  |  |
| --- | --- |
| **URL:** <http://amst.us-east-2.elasticbeanstalk.com/incoming/sensor> (forwarded)  **Method:** N.A  **Descripción:** Es un método interno que se llama desde la API /incoming/sensor  Recibe el nivel de batería y crea un HTTP request hacia los servidores de Firebase, dirigiendo la notificación hacia /topics/all y que todos los dispositivos puedan recibir la notificación ya que siempre se consume desde ese tema principal. Se debe establecer en el Header la autorización que puede ser consultada como un parámetro del proyecto de Fireabase. | |
|  | **public void** setNotification(String bateria) {  **try** {   URL url = **new** URL(**"https://fcm.googleapis.com/fcm/send"**);  HttpURLConnection conn = (HttpURLConnection) url.openConnection();  conn.setDoOutput(**true**);  conn.setRequestMethod(**"POST"**);   conn.setRequestProperty(**"Content-Type"**, **"application/json"**);  conn.setRequestProperty(**"Authorization"**, **"key=AIzaSyCjogHet9iSGwjNRpJ5qh9AcT6LP8Ulgfg"**);  String option=**"/topics/all"**;  String input = **"{\n"** +  **" \"to\" : \"/topics/all\",\n"** +  **" \"collapse\_key\" : \"type\_a\",\n"** +  **" \"notification\" : {\n"** +  **" \"body\" : \"Batería actual: "**+bateria+**"\",\n"** +  **" \"title\": \"Alerta de bateria\"\n"** +  **" }\n"** +  **"}"**;   OutputStream os = conn.getOutputStream();  os.write(input.getBytes());  os.flush(); BufferedReader br = **new** BufferedReader(**new** InputStreamReader(  (conn.getInputStream())));   String output;  System.***out***.println(**"Output from Server .... \n"**);  **while** ((output = br.readLine()) != **null**) {  System.***out***.println(output);  }   conn.disconnect();   } **catch** (MalformedURLException e) {   e.printStackTrace();   } **catch** (IOException e) {   e.printStackTrace();   }   } |

1. Reconfiguración de parámetros (umbrales) para decisión de niveles de presión del Velostat y batería

|  |  |
| --- | --- |
| **URL:** http://amst.us-east-2.elasticbeanstalk.com/incoming/niveles  **Method:** POST  **Descripción:** Se requiere que en el body se ingrese un Json con los parámetros descritos abajo. Cada uno de estos parámetros son ingresados en las variables estáticas del servicio, las cuales luego serán consumidas por el API de ingreso de nueva data. Por defauil viene con los valores establecidos en el ejemplo  un Request hacia el servicio de Firebase para enviar notificaciones automaticas. | |
|  | @PostMapping(**"/niveles"**) **public void** actulizarNiveles(@RequestBody Map<String,Object> body) {  NoteController.*setNivelAlto*(Integer.*parseInt*(body.get(**"alto"**).toString()));  NoteController.*setNivelBajo*(Integer.*parseInt*(body.get(**"bajo"**).toString()));  NoteController.*setBateria*(Integer.*parseInt*(body.get(**"bateria"**).toString())); } |
| E.g | {  "alto":"221",  "bajo":"201",  "bateria":"60"  }  #Donde alto es un entero.  #Donde bajo es un entero  #Donde batería es un entero entre 0 y 100 |

Adicional a los APIs expuestos toda información procesada es guardada en un repositorio alterno con fines de backup y futuro uso para análisis de los datos. Esta BD es gratuita proporcionada por [*http://remotemysql.com*](http://remotemysql.com)

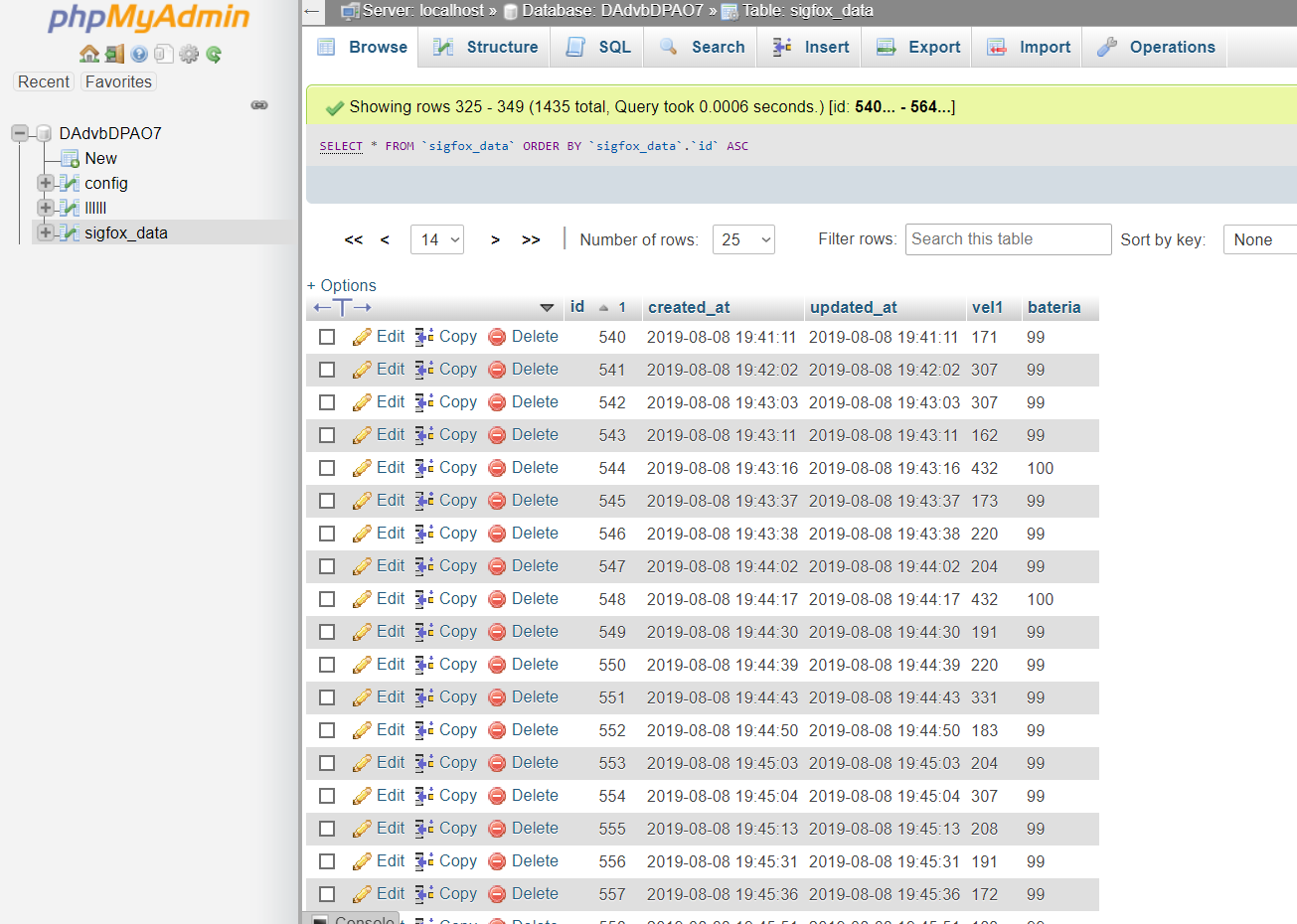
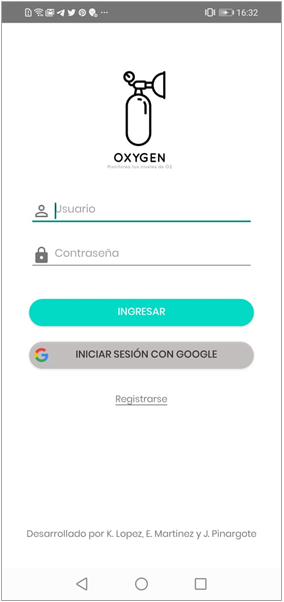


Ilustración Vista del phpMyAdmin de BD alterna

# Screenshoots de la aplicación



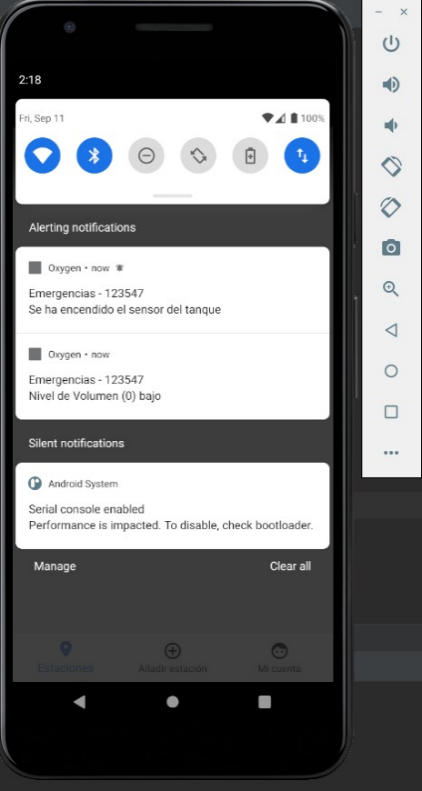
VENTANA 1. INICIO DE SESION

 VENTANA 2. LISTA DE TANQUES REGISTRADOS

VENTANA 3. INFORMACION DETALLADA DEL TANQUE





VENTANA 4. UBICACIÓN DEL TANQUE

VENTANA 5. NOTIFICACIONES

# Códigos en Android Studio

|  |  |
| --- | --- |
| **Nombre de archivo:** MainActivity.class  **Nombre de función:** isOnline()  **Descripción:** Esta función permite detectar si la aplicación tiene conexión de internet, retorna un booleano | |
|  | public boolean isOnLine(){  ConnectivityManager connectivityManager = (ConnectivityManager) getSystemService(Context.CONNECTIVITY\_SERVICE);  NetworkInfo actNetInfo = connectivityManager.getActiveNetworkInfo();  return (actNetInfo != null && actNetInfo.isConnected());  } |

|  |  |
| --- | --- |
| **Nombre de archivo:** Main.class  **Nombre de función:** iniciarsesion (View v)  **Descripción:** Esta función maneja el evento del botón cuando el usuario quiere iniciar sesión con Google, y antes de realizarse esta opción, se valida si el usuario tiene conexión de internet. | |
|  | public void iniciarsesion(View v){  if(isOnLine()){  Intent signInIntent = mGoogleSignInClient.getSignInIntent();  startActivityForResult(signInIntent,GOOGLE\_SIGN\_IN);  }else{  Toast.makeText(MainActivity.this, "No hay conexion de internet", Toast.LENGTH\_SHORT).show();  }  } |

|  |  |
| --- | --- |
| **Nombre de archivo:** MainActivity.class  **Nombre de función:** getUserInfo ()  **Descripción:** Esta función permite cargar al usuario que ah iniciado sesión en la aplicación, para aquello se realiza la consulta a la base de datos, por medio del id único del usuario, para una búsqueda rápida y eficiente | |
|  | private void getUserInfo(){  String id = mAuth.getCurrentUser().getUid().toString();  databaseReference.child(VariablesUnicas.USUARIO\_FI).child(id).addValueEventListener(new ValueEventListener() {  @Override  public void onDataChange(@NonNull DataSnapshot dataSnapshot) {  if(dataSnapshot.exists()){  usuario = dataSnapshot.getValue(Usuario.class);  }  }  @Override  public void onCancelled(@NonNull DatabaseError databaseError) {  }  });  } |
| **Nombre de archivo:** PrincipalActivity.class  **Nombre de función:** solicitarTanque ()  **Descripción:** Esta función permite obtener en tiempo real los tanques registrados por el usuario registrado en la aplicación. Los tanques encontrados, los retorna en un TreeSet<Tanque> | |
|  | public TreeSet<Tanque> solicitarTanque(){  final TreeSet<Tanque> tanquesEncontrados = new TreeSet<>();  tanquesEncontrados.clear();  Usuario u = MainActivity.getUsuario();  if(u != null){  databaseReferenceTanques.child(u.getIdUser()).addValueEventListener(new ValueEventListener() {  @Override  public void onDataChange(@NonNull DataSnapshot dataSnapshot) {  for (DataSnapshot data: dataSnapshot.getChildren()){  Tanque t = data.getValue(Tanque.class);  tanquesEncontrados.add(t);  }  }  @Override  public void onCancelled(@NonNull DatabaseError databaseError) {  }  });  }else{  u = RegistroActivity.getUsuario();  if(u!= null){  databaseReferenceTanques.child(u.getIdUser()).addValueEventListener(new ValueEventListener() {  @Override  public void onDataChange(@NonNull DataSnapshot dataSnapshot) {  for (DataSnapshot data: dataSnapshot.getChildren()){  Tanque t = data.getValue(Tanque.class);  tanquesEncontrados.add(t);  }  }  @Override  public void onCancelled(@NonNull DatabaseError databaseError) {  }  });  }  }  return tanquesEncontrados;  } |

|  |  |
| --- | --- |
| **Nombre de archivo:** PrincipalActivity.class  **Nombre de función:** solicitarUbicaciones ()  **Descripción:** Esta función permite obtener en tiempo real las ubicaciones encontradas por el usuario registrado en la aplicación. Las ubicaciones encontrados, los retorna en un TreeSet<Ubicacion> | |
|  | public TreeSet<Ubicacion> solicitarUbicaciones(){  final TreeSet<Ubicacion> ubicacionesEncontradas = new TreeSet<>();  //ubicacionesEncontradas.clear();  Usuario u = MainActivity.getUsuario();  if(u != null){  databaseReferenceUbicaciones.child(u.getIdUser()).addValueEventListener(new ValueEventListener() {  @Override  public void onDataChange(@NonNull DataSnapshot dataSnapshot) {  for (DataSnapshot d: dataSnapshot.getChildren()  ) {  Ubicacion u = d.getValue(Ubicacion.class);  ubicacionesEncontradas.add(u);  }  }  @Override  public void onCancelled(@NonNull DatabaseError databaseError) {  }  });  }else{  u = RegistroActivity.getUsuario();  if(u!= null){  databaseReferenceUbicaciones.child(u.getIdUser()).addValueEventListener(new ValueEventListener() {  @Override  public void onDataChange(@NonNull DataSnapshot dataSnapshot) {  for (DataSnapshot d: dataSnapshot.getChildren()  ) {  Ubicacion u = d.getValue(Ubicacion.class);  ubicacionesEncontradas.add(u);  }  }  @Override  public void onCancelled(@NonNull DatabaseError databaseError) {  }  });  }  }  return ubicacionesEncontradas;  } |

|  |  |
| --- | --- |
| **Nombre de archivo:** AgregaeEstacionFragment.class  **Nombre de función:** setOnClickListener (new View.OnClickListener(){})  **Descripción:** Este evento se ejecuta cada vez que el usuario de click en el botón de agregar estación, este evento también valida si todos los datos han sido ingresados correctamente, si todo es correcto se agrega el tanque y su ubicación en la | |
|  | btn\_agregar.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View view) {  txt\_Nombre = (EditText)v.findViewById(R.id.txt\_nombre\_estacion);  txt\_descripcion = (EditText)v.findViewById(R.id.txt\_descripcion);  txt\_codigo = (EditText)v.findViewById(R.id.txt\_codigo);  txt\_habitacion = (EditText)v.findViewById(R.id.txt\_habitacion);  txt\_piso = (EditText)v.findViewById(R.id.txt\_piso);  txt\_vol = (EditText)v.findViewById(R.id.txt\_volumen);  //String contenidoT = "Oxigeno";  String nombreT = txt\_Nombre.getText().toString();  String descripcion = txt\_descripcion.getText().toString();  String codigo = txt\_codigo.getText().toString();  String ubicacion = txt\_ubicacion.getText().toString();  String habitacion = txt\_habitacion.getText().toString();  int habitacionA = Integer.parseInt(habitacion);  int piso = Integer.parseInt(txt\_piso.getText().toString());  String volumenInicial = txt\_vol.getText().toString();  // Tanque estacion = new Tanque(nombreE,volumenInicial,"","",codigo,1,MainActivity.getUsuario().getIdUser());  if(txt\_Nombre.getText().toString().isEmpty() || txt\_codigo.getText().toString().isEmpty() || txt\_ubicacion.getText().toString().isEmpty() || txt\_habitacion.getText().toString().isEmpty() ||  txt\_piso.getText().toString().isEmpty() || txt\_vol.getText().toString().isEmpty() || txt\_descripcion.getText().toString().isEmpty()){  Toast.makeText(getActivity(), "Datos incompletos", Toast.LENGTH\_SHORT).show();  }  else{  Toast.makeText(getActivity(), "Estación añadida", Toast.LENGTH\_SHORT).show();  TreeSet<Integer> idUbicacionUser = PrincipalActivity.getIdUbicacionesUser();  // System.out.println("Numero de ubicaciones: " + idUbicacionUser.size());  if(idUbicacionUser.size()>0){  int valor = idUbicacionUser.size() + 1 ;  String valorA = String.valueOf(valor);  Tanque tanque = new Tanque(codigo,valor,nombreT,"0",volumenInicial,"0");  Usuario u = MainActivity.getUsuario();  Ubicacion ubicacion1 = new Ubicacion(ubicacion,habitacionA,valor,piso,latitud,longitud);  databaseReference.child(VariablesUnicas.TANQUES\_FI).child(u.getIdUser()).child(tanque.getIdModulo()).setValue(tanque);  databaseReference.child(VariablesUnicas.TANQUES\_USER).child(tanque.getIdModulo()).setValue(u.getIdUser());  databaseReference.child(VariablesUnicas.UBICACIONES\_FI).child(u.getIdUser()).child(String.valueOf(idUbicacionUser.size()+1)).setValue(ubicacion1);  }else{  Tanque tanque = new Tanque(codigo,1,nombreT,"100",volumenInicial,volumenInicial);  Usuario u = MainActivity.getUsuario();  Ubicacion ubicacion1 = new Ubicacion(ubicacion,habitacionA,1,piso,latitud,longitud);  databaseReference.child(VariablesUnicas.TANQUES\_FI).child(u.getIdUser()).child(tanque.getIdModulo()).setValue(tanque);  databaseReference.child(VariablesUnicas.TANQUES\_USER).child(tanque.getIdModulo()).setValue(u.getIdUser());  databaseReference.child(VariablesUnicas.UBICACIONES\_FI).child(u.getIdUser()).child("1").setValue(ubicacion1);  }  }  }  }); |

|  |  |
| --- | --- |
| **Nombre de archivo:** AgregarEstacionFragment.class  **Nombre de función:** obtenerUbicacion (Button btn, TextView tv)  **Descripción:** Esta función permite obtener la ubicación exacta del teléfono, se asume que cuando el usuario registre el tanque, lo realiza en un lugar estatico o fijo, como su hogar, en caso que sea un lugar publico, puede ser un centro de salud, por esa razón se obtiene la ubicación del dispositivo y se la almacena en la base de datos de firebase. | |
|  | public void obtenerUbicacion(Button btn, TextView textView){  btn.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View view) {  if (ActivityCompat.checkSelfPermission(getContext(), Manifest.permission.ACCESS\_FINE\_LOCATION) != PackageManager.PERMISSION\_GRANTED){  ActivityCompat.requestPermissions(getActivity(),  new String[]{Manifest.permission.ACCESS\_FINE\_LOCATION},44);  }  else {  fusedLocationProviderClient.getLastLocation().addOnCompleteListener(  new OnCompleteListener<Location>() {  @Override  public void onComplete(@NonNull Task<Location> task) {  Location location = task.getResult();  if (location != null) {  try {  //Intent i = new Intent(MainActivity.this, Ubicacion.class);  Geocoder geocoder = new Geocoder(getActivity(), Locale.getDefault());  List<Address> list = geocoder.getFromLocation(location.getLatitude(),  location.getLongitude(), 1);  String titulo = list.get(0).getLocality();  textView.setText(titulo);  latitud = list.get(0).getLatitude();  longitud = list.get(0).getLongitude();  coordenadas.add(latitud);  coordenadas.add(longitud);  } catch (IOException e) {  e.printStackTrace();  }  }  }  });  }  }  });  } |

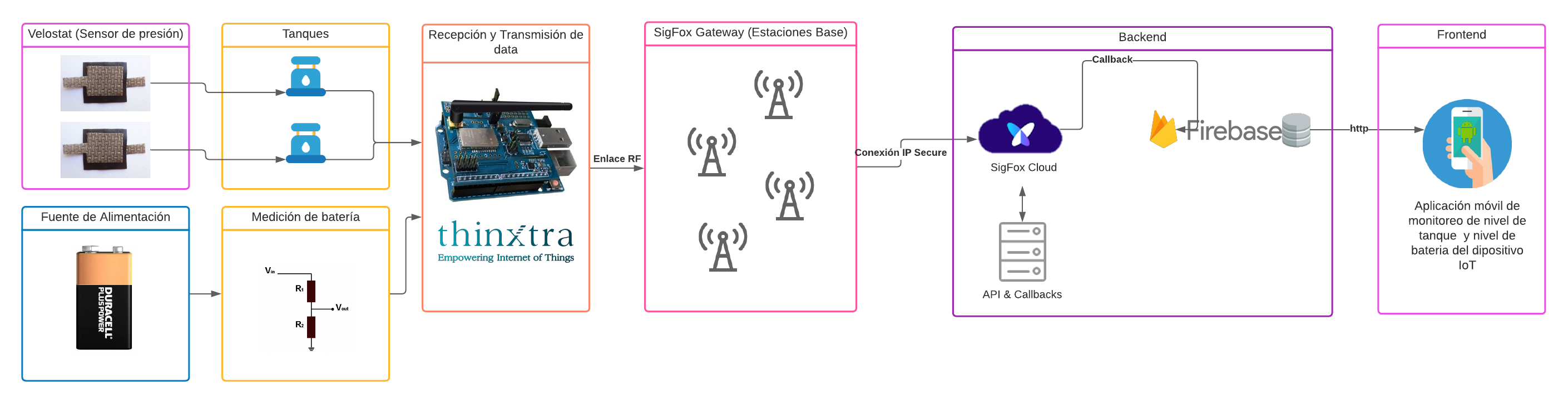
|  |  |
| --- | --- |
| **Nombre de archivo:** EstacionsFragment.class  **Nombre de función:** crearLayout ()  **Descripción:** Esta función crear el Layout de cada uno de los tanques que se encuentran registrados por el usuario, en caso que el usuario agregue otro tanque, esta función en tiempo real agregara el nuevo tanque en el Fragment de Estaciones. | |
|  | public RelativeLayout crearLayout(Tanque tanque, LayoutInflater inflater){  RelativeLayout relativeLayout;  String nombreModulo = tanque.getNombre();  String volumenActual = tanque.getVolumenMedido();  String volumenInicial = tanque.getVolumenInicial();  //añadir los layouts con la información de las vistas  int id = R.layout.layout\_estacion; //layout con datos de la estación  //Inicializacion relative layout  relativeLayout = (RelativeLayout)inflater.inflate(id,null,false);  //titulo  TextView titulo = (TextView)relativeLayout.findViewById(R.id.titulo\_estacion) ;  String titulo\_id = "Tanque: " + nombreModulo;  titulo.setText(titulo\_id);  //porcentaje velostat  TextView prctjEstacion = (TextView)relativeLayout.findViewById(R.id.porcentaje);  com.ramijemli.percentagechartview.PercentageChartView pcv = (com.ramijemli.percentagechartview.PercentageChartView)relativeLayout.findViewById(R.id.pcv\_oxigeno);  int porcentaje = 0;  //verificar si se encuentra o no el tanque  if(Integer.parseInt(volumenActual) == -1){  prctjEstacion.setText("0");  pcv.setProgress(0,true);  TextView descripcion = (TextView)relativeLayout.findViewById(R.id.descripcion\_estacion);  String aviso = "No se encuentra tanque";  descripcion.setText(aviso);  }else{  porcentaje = (int)((Integer.parseInt(volumenActual)\*100)/(Integer.parseInt(volumenInicial)));  String porcentaje\_str = porcentaje + "%";  prctjEstacion.setText(porcentaje\_str);  pcv = (com.ramijemli.percentagechartview.PercentageChartView)relativeLayout.findViewById(R.id.pcv\_oxigeno);  pcv.setProgress(porcentaje,true);  }  //mostrar alerta en caso de porcentaje menores  if(porcentaje < 25){  RelativeLayout fondo = (RelativeLayout)relativeLayout.findViewById(R.id.fondo);  fondo.setBackgroundResource(R.drawable.animation\_background);  AnimationDrawable animationDrawable= (AnimationDrawable)fondo.getBackground();  animationDrawable.setEnterFadeDuration(1000);  animationDrawable.setExitFadeDuration(1000);  animationDrawable.start();  }  return relativeLayout;  } |

|  |  |
| --- | --- |
| **Nombre de archivo:** InforTanque.class  **Nombre de función:** mostrarInformacionTanque ()  **Descripción:** Esta función muestra la información del tanque seleccionado por el usuario, se muestra a detalle, el porcentaje de batería de la estación, el porcentaje de volumen actual del tanque, un botón que permite ver la ubicación, etc. | |
|  | public void mostrarInformacionTanque(){  txt\_porcentajeBateria.setText(porcentajeBateria);  txt\_volumenInicial.setText(volumenInicial);  txt\_volumenActual.setText(volumenActual);  pcv.setProgress(porcentaje,true);  txt\_porcentaje.setText(porcentaje\_str);  txt\_pisoEstacion.setText(piso);  txt\_habitacionEstacion.setText(habitacion);  txt\_ubicacionEstacion.setText(ciudad);  btn\_verUbicacion.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View view) {  Intent i = new Intent(InforTanque.this, MapsActivity.class);  //ubicacion = (Ubicacion)i.getSerializableExtra("ubicacion");  i.putExtra("ciudad", ubicacion.getCiudad());  i.putExtra("latitud", ubicacion.getLatitud());  i.putExtra("longitud", ubicacion.getLongitud());  startActivity(i);  }  });  } |

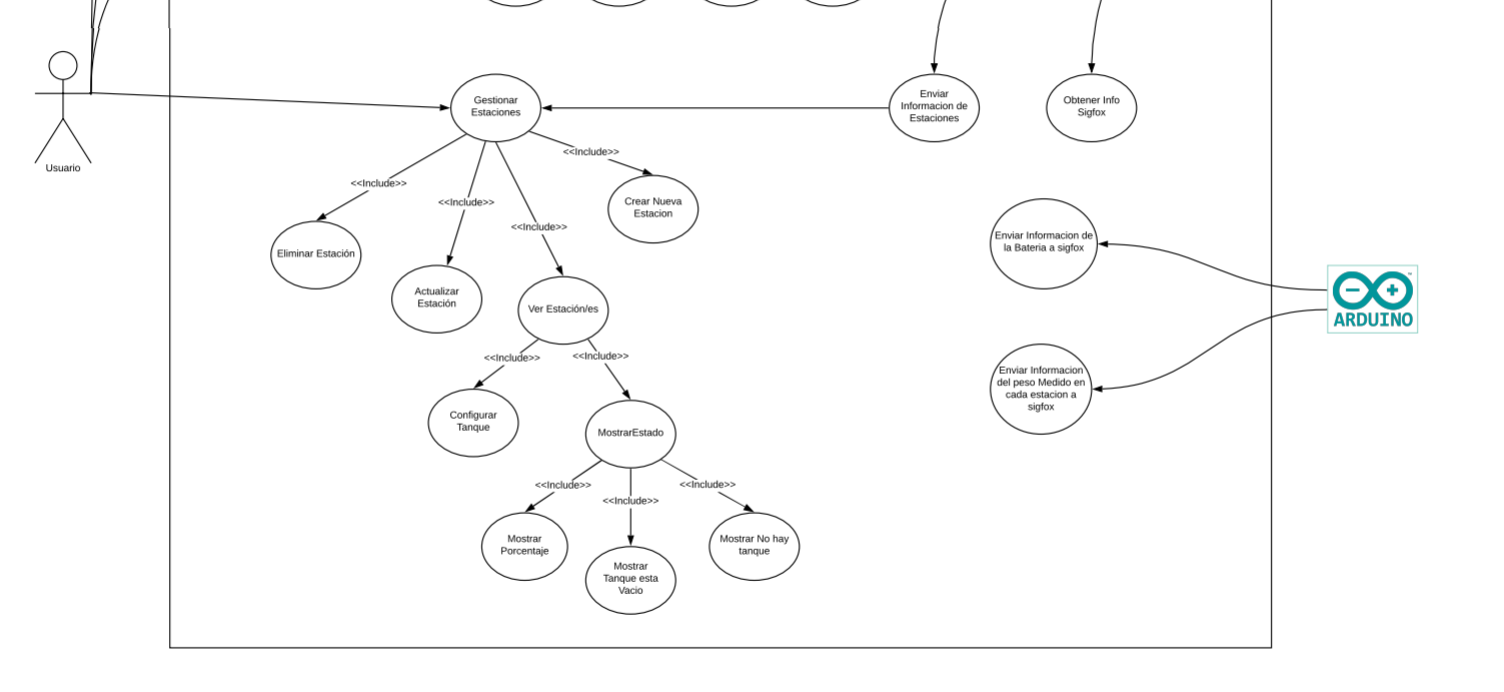
|  |  |
| --- | --- |
| **Nombre de archivo:** MapsActivity.class  **Nombre de función:** onMapReady(GoogleMap googleMap)  **Descripción:** Esta función permite ver la ubicación del tanque | |
|  | public void onMapReady(GoogleMap googleMap) {  Bundle bundle = getIntent().getExtras();  mMap = googleMap;  mMap.getUiSettings().setZoomControlsEnabled(true);  // Add a marker in Sydney and move the camera  LatLng sydney = new LatLng(bundle.getDouble("latitud"), bundle.getDouble("longitud"));  mMap.addMarker(new MarkerOptions().position(sydney).title(bundle.getString("ciudad")));  mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(sydney,10));  } |

|  |  |
| --- | --- |
| **Nombre de archivo:** ProfileFragmet.class  **Nombre de función:** onCreateView (LayoutInflater inflater, ViewGroup container)  **Descripción:** Esta función permite ver toda la información del usuario que se encuentra almacenada en la base de datos, es decir muestra su información mas relevante, como su nombre, correo, estaciones registradas, además muestra un botón que le permite cerrar la aplicacion. | |
|  | public View onCreateView(LayoutInflater inflater, ViewGroup container,  Bundle savedInstanceState) {  // Inflate the layout for this fragment  // FirebaseAuth mAuth = FirebaseAuth.getInstance();  View mainF = inflater.inflate(R.layout.fragment\_profile, container, false);  Button btn = mainF.findViewById(R.id.btn\_CerrarS);  TextView txt\_name = mainF.findViewById(R.id.txt\_user);  TextView txt\_email = mainF.findViewById(R.id.txt\_email);  ImageView imv\_photo = mainF.findViewById(R.id.imv\_foto);  TextView estacionesRegistrada = mainF.findViewById(R.id.estacionesRegistradas);  Usuario usuario = MainActivity.getUsuario();  if(usuario!= null){  TreeSet<Tanque> tanquesUsuario = PrincipalActivity.getTanques();  txt\_name.setText(usuario.getNombreUsuario());  txt\_email.setText(usuario.getCorreo());  String photo = usuario.getImagen();  Picasso.with(getActivity().getApplicationContext()).load(photo).into(imv\_photo);  estacionesRegistrada.setText(String.valueOf(tanquesUsuario.size()));  }else{  usuario = RegistroActivity.getUsuario();  TreeSet<Tanque> tanquesUsuario = PrincipalActivity.getTanques();  txt\_name.setText(usuario.getNombreUsuario());  txt\_email.setText(usuario.getCorreo());  String photo = usuario.getImagen();  Picasso.with(getActivity().getApplicationContext()).load(photo).into(imv\_photo);  estacionesRegistrada.setText(String.valueOf(tanquesUsuario.size()));  }  cerrarS(btn);  return mainF;  } |

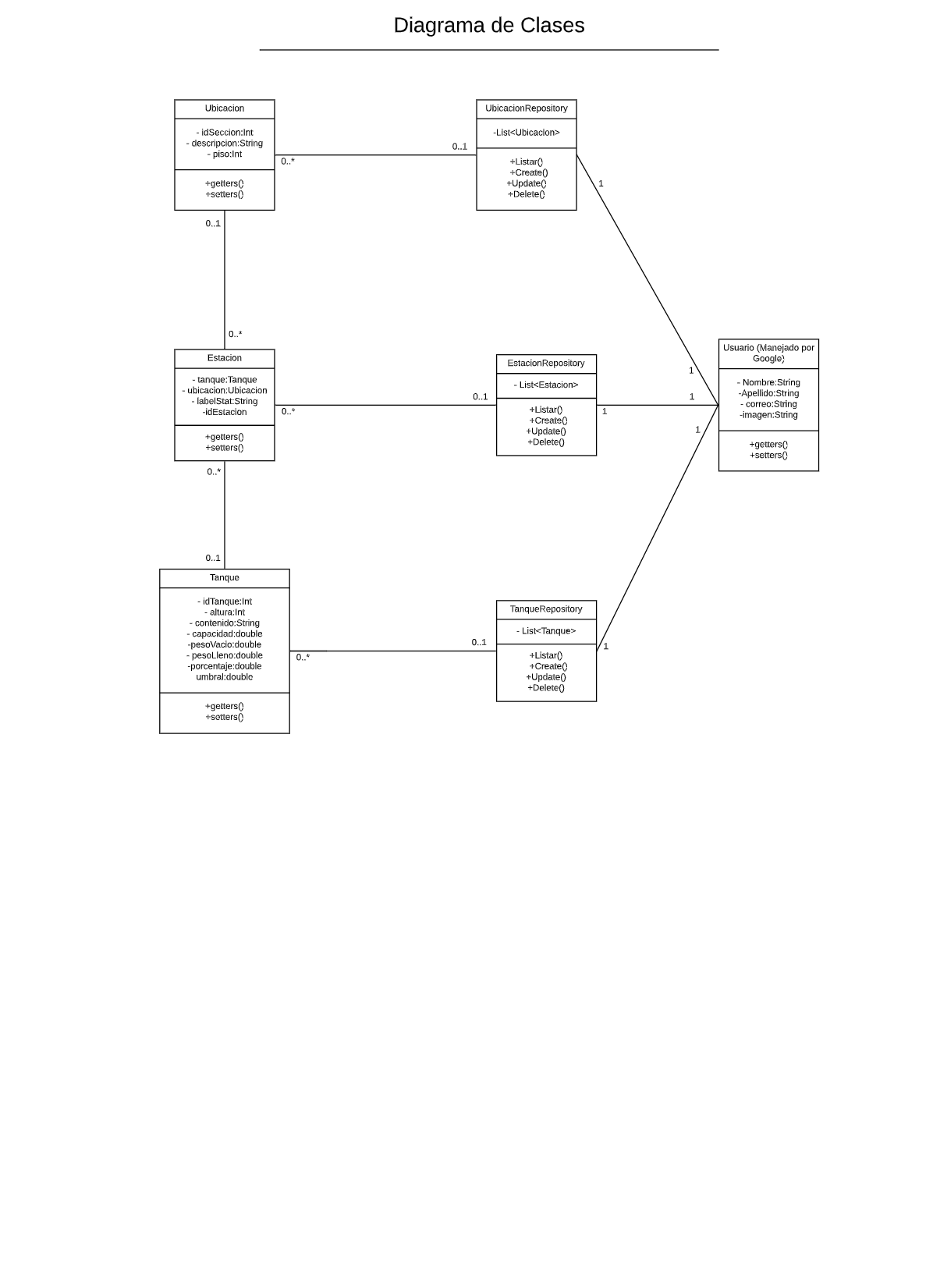
## Propuesta de Solución



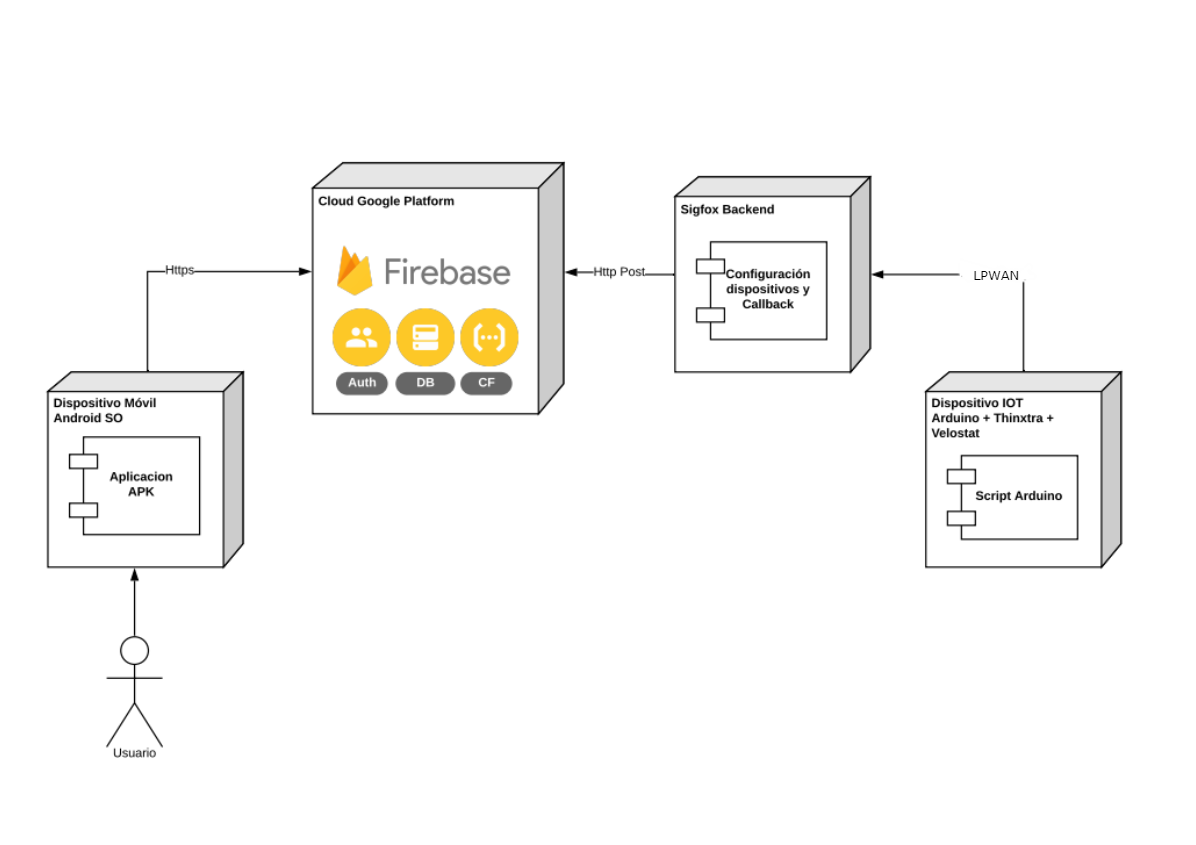
## Diagrama casos de uso



## Diagrama de Clases



## Diagrama de despliegue



## Limitaciones

La principal limitante al realizar este proyecto fue el sensor, es decir el velostat, debido a su característica de ser sensible a la presión, para mejorar el censo de los datos se debe diseñarlo de la manera mas optima, buscar los materiales idóneos que lo proteja a factores externos como el ruido, y además que las capas utilizadas para mejorar el censo de los datos, permiten que el velostat sea sensible y los datos censados se mantengan de manera constante, en este proyecto se buscó realizar una gráfica de presión vs voltaje del velostat, pero lastimosamente esta función obtenida tenia sus limitantes, ya que se utilizo algodón, y la función obtenida solo funcionaria de manera temporal. Se recomienda reemplazar el algodón por otro material como la esponja, pero de esas esponjas que se encuentran en equipos eléctricos, como las laptops, y las dimensiones de la esponja dependerá de las dimensiones del velostat.

Utilizar Sigfox para el envío de datos, Sigfox permite un máximo de 140 mensajes diarios, y para luego la información enviada por cada mensaje no debe sobrepasar los 12 bytes.

# Referencias

[1] StackOverflow. Como poner videos de youtube en aplicación Android. 2017. Obtenido en: <https://es.stackoverflow.com/questions/211495/como-poner-videos-de-youtube-en-aplicacion-android>

[2] "aplicación móvil con integración de base de datos externas y privadas usando rest api," espol, 2019.

[3] Sigfox, «aprendiendo arduino,» [en línea]. available: https://aprendiendoarduino.wordpress.com/tag/backend-sigfox/. [Último acceso: 2019].

[4] Spring Framework, 2019 Pivotal Software, Inc. available: https://spring.io/projects/spring-framework