

|  |
| --- |
| **Manual Técnico** |
| Manual Técnico del sistema integrado “i-Tank” |
| Se describe el diseño del prototipo para la gestión de i-Tank en ambientes industriales, así como también la aplicación nativa de Android. |
|  |



**Integrantes:**

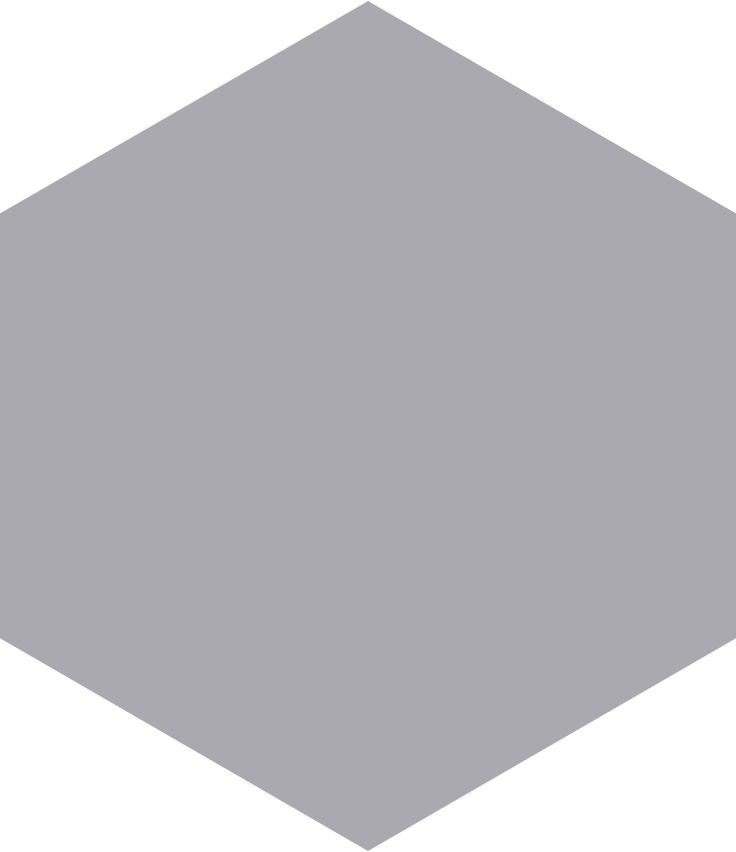
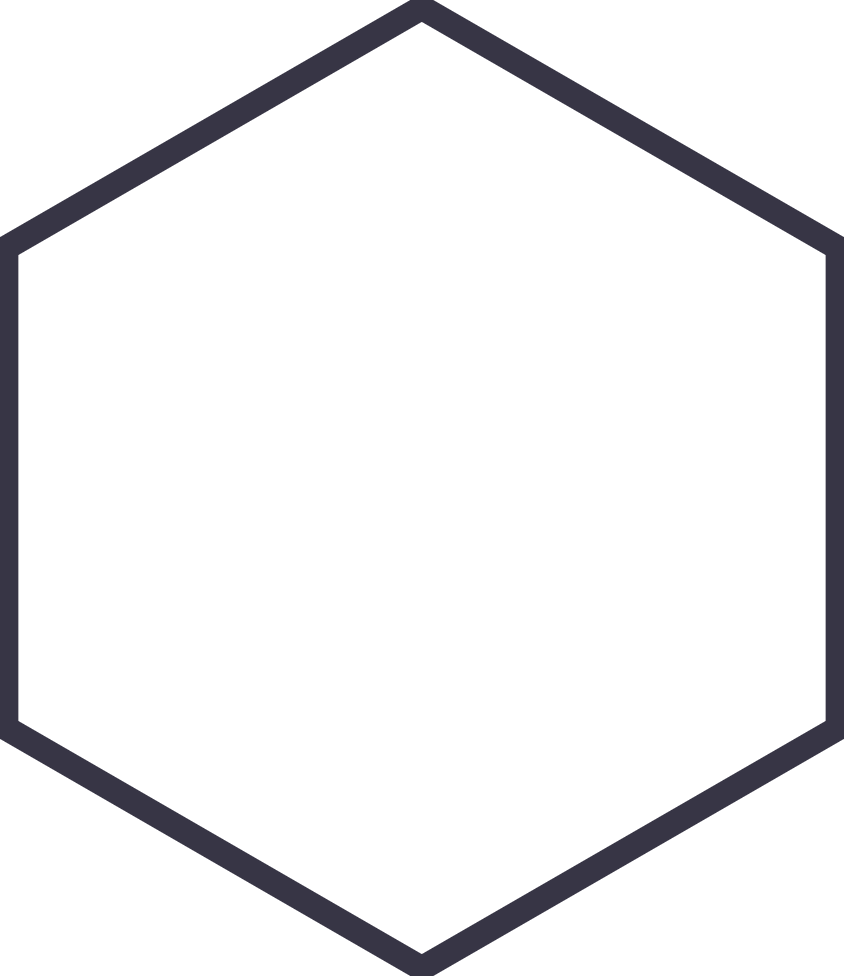
Cabezas Ana

Cedeño Carlos

Mera Renan

**24/08/2019 Versión 1**

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.

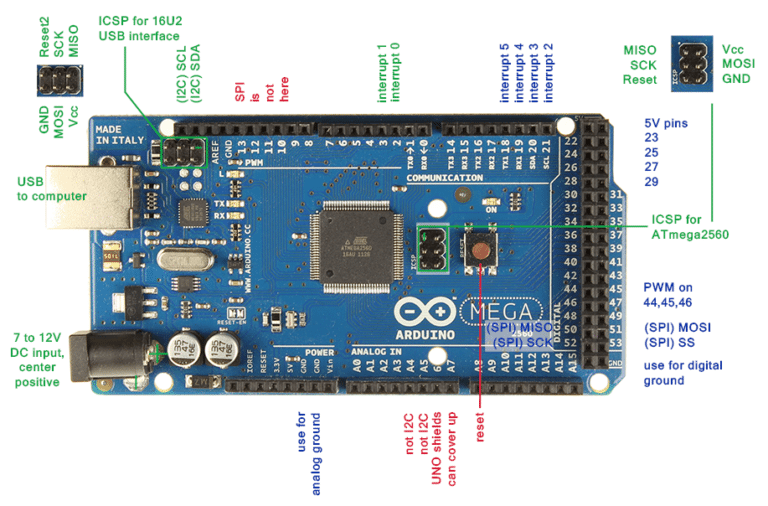
[](http://panamahitek.com/wp-content/uploads/2013/01/Mega2560_R3_Label-small-v2-2.jpg)

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

Para el diseño de Hardware de nuestro prototipo se procedió a realizar diferentes modelos de configuración del Velostat, de los cuales se describirán los mas significativos y el prototipo final del sensor realizado.

## Prototipo 1

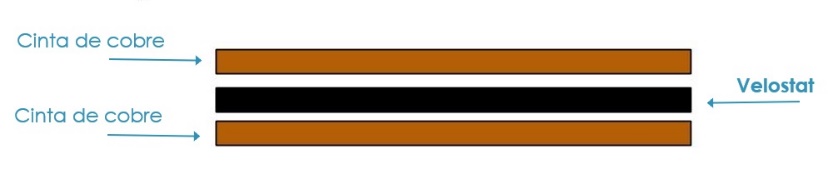
El primer intento del prototipo se realizó con los siguientes materiales:

**Materiales**

* Velostat tamaño 4x4cm
* Cinta de cobre adhesiva tamaño 3x3cm
* Jumpers

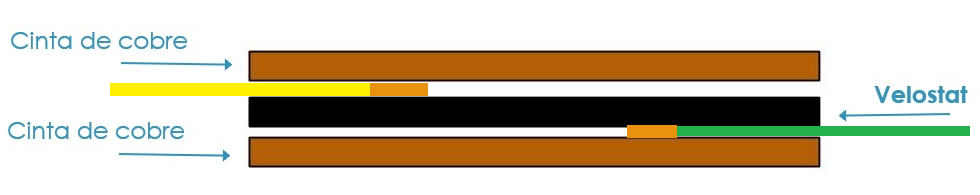
**Descripción de las capas a realizar:**

Se realiza una topología tipo sándwich como muestra la imagen, para armar las capas del sensor.

**

*Ilustración 1 Topología sándwich.*

Ahora que se tiene mentalizado la forma del sensor, se tiene que poner dos jumpers que serán los terminales el cual uno ira a GND y el otro al divisor de voltaje. En la Siguiente imagen mostrara como deben ir los cables entre las capas.



*Ilustración 2 Posición de los cables entre las capas.*

Se pudo observar de este prototipo que el tamaño del velostat influye en la construcción del sensor ya que a más tamaño menos estable y que los jumpers no brindaban el suficiente contacto con la superficie de la cinta de cobre.

## Prototipo 2

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

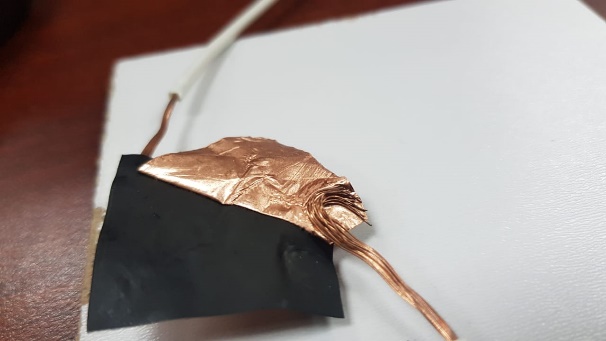
**Materiales**

* 2 Velostat tamaño 3x3cm
* Cinta de cobre tamaño 2x2cm
* Cable calibre 20

**Descripción de las capas a realizar:**

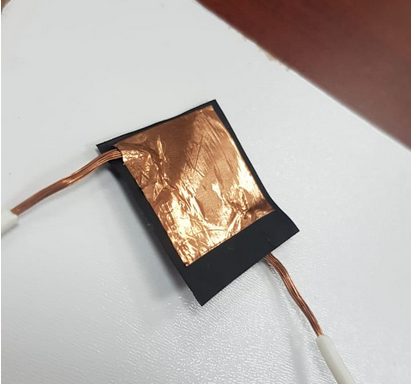
Se realiza una topología tipo sándwich como el prototipo 1 con la diferencia que el velostat ahora es doble capa.

Ahora que se tiene mentalizado la forma del sensor, se tiene que poner dos cables que serán los terminales el cual uno ira a GND y el otro al divisor de voltaje. En la Siguiente imagen mostrara como deben ir los cables entre las capas, además de observar la posición también se les dio una forma de espiral a los cables tratando de recubrir la mayor superficie.



*Ilustración* *4 Posición de los cables entre las capas.*

Para este segundo prototipo se supone que la posición de los cables influye en la estabilidad del sensor por lo tanto ahora vamos a posicionar los cables de la siguiente manera como muestra la imagen.



*Ilustración 5 Doble capa de Velostat y la posición particular del cable.*

## Prototipo 3

El prototipo numero 3 se diferencia por la particularidad que al prototipo numero 2 se lo recubrió completamente de cinta adhesiva con el propósito de dar más estabilidad al velostat, pero resultó lo contrario dado que aumento el ruido en la señal y se notó que la deformación del velostat demoraba mucho tiempo en poder regresar al estado original.

Por lo tanto, lo que se pudo conocer haciendo este tercer prototipo fue que el sensor necesitaba tener una superficie rígida y plana para que su estabilidad no sea afectada.

## 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.

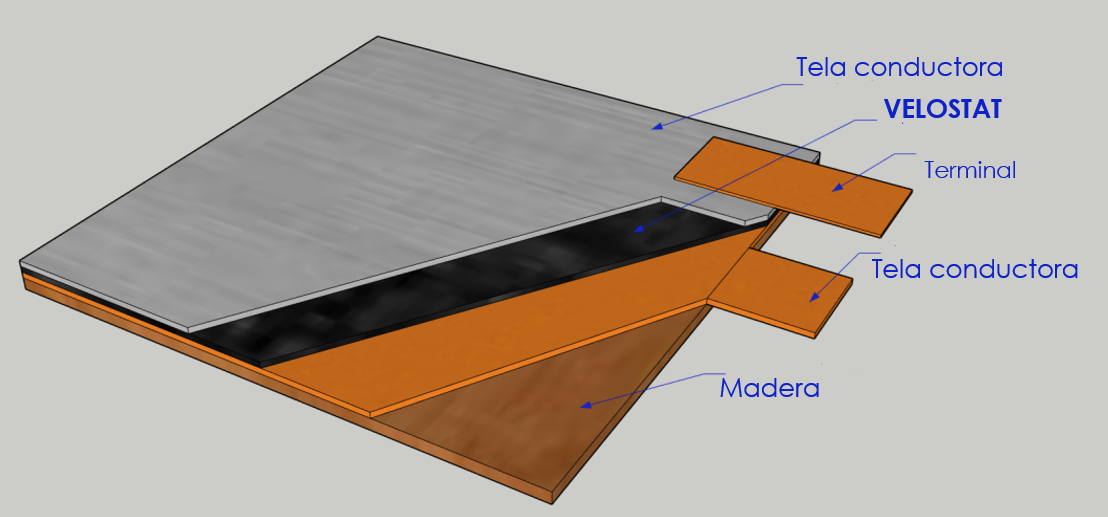
Los materiales usados en el siguiente prototipo exitoso para el proyecto fueron:

**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.

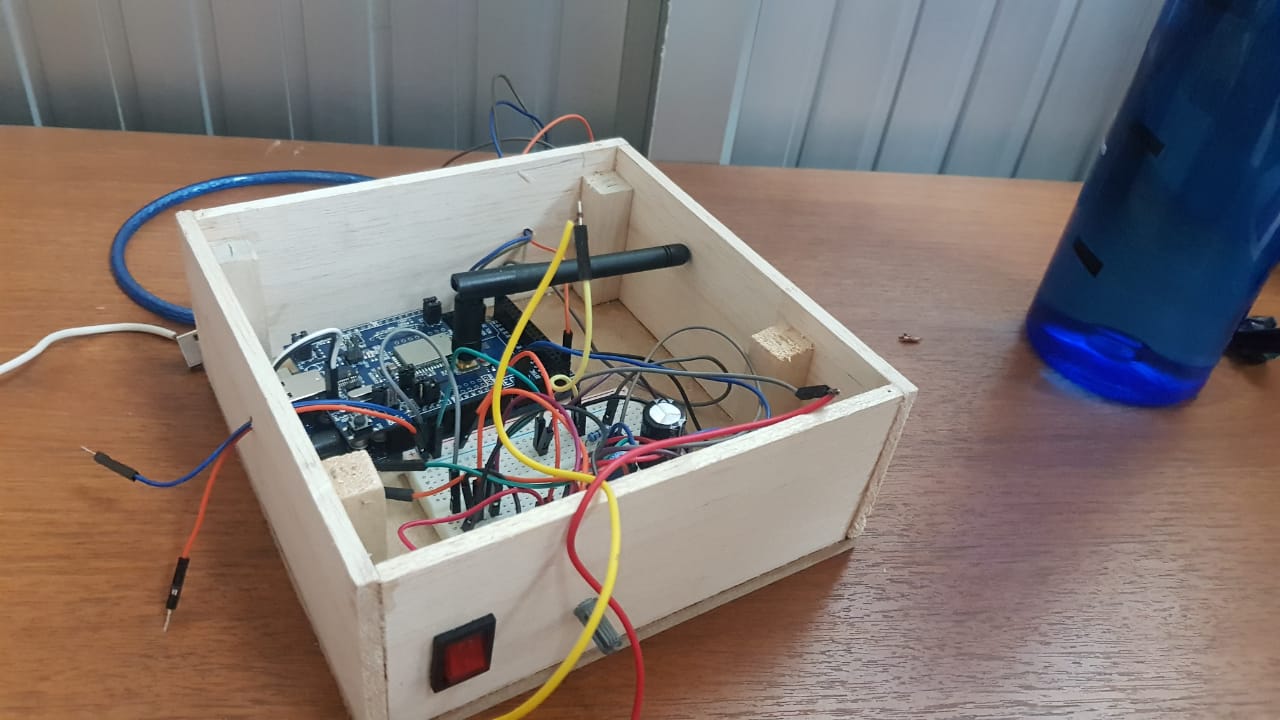


*Ilustración 7 El prototipo armado con sus respectivas capas.*

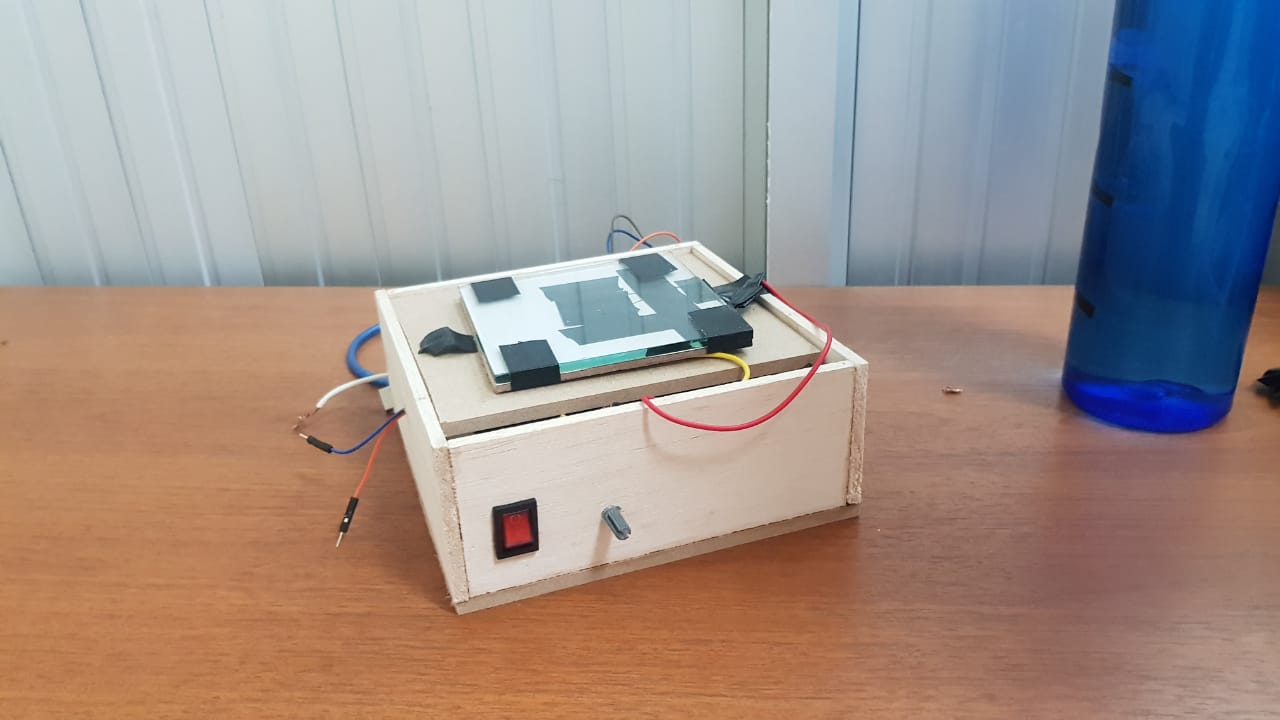
En la base de madera vendrá fijado el sándwich con trozos de cinta aislante y recubierto con el trozo de vidrio que también vendrá fijado con cinta.



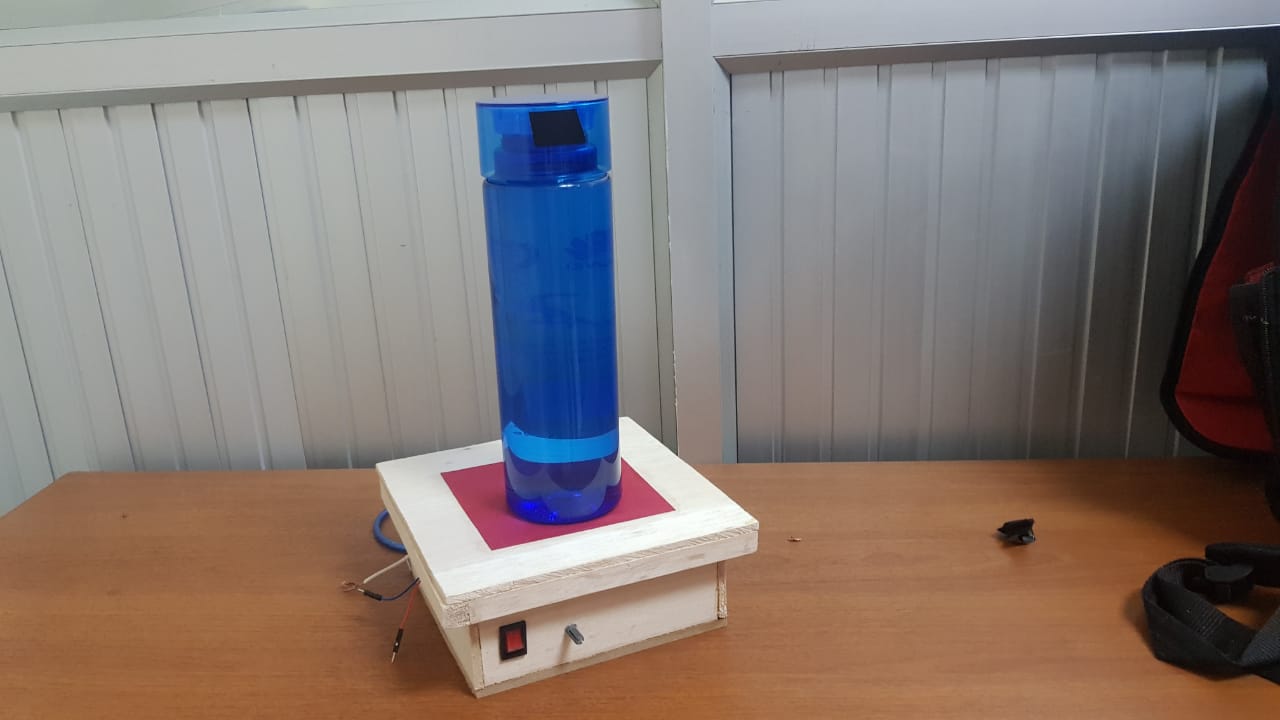
*Ilustración 8 El prototipo del sensor terminado con su base de madera y recubierta de vidrio.*



*Ilustración 8 Circuito en su respectiva caja.*

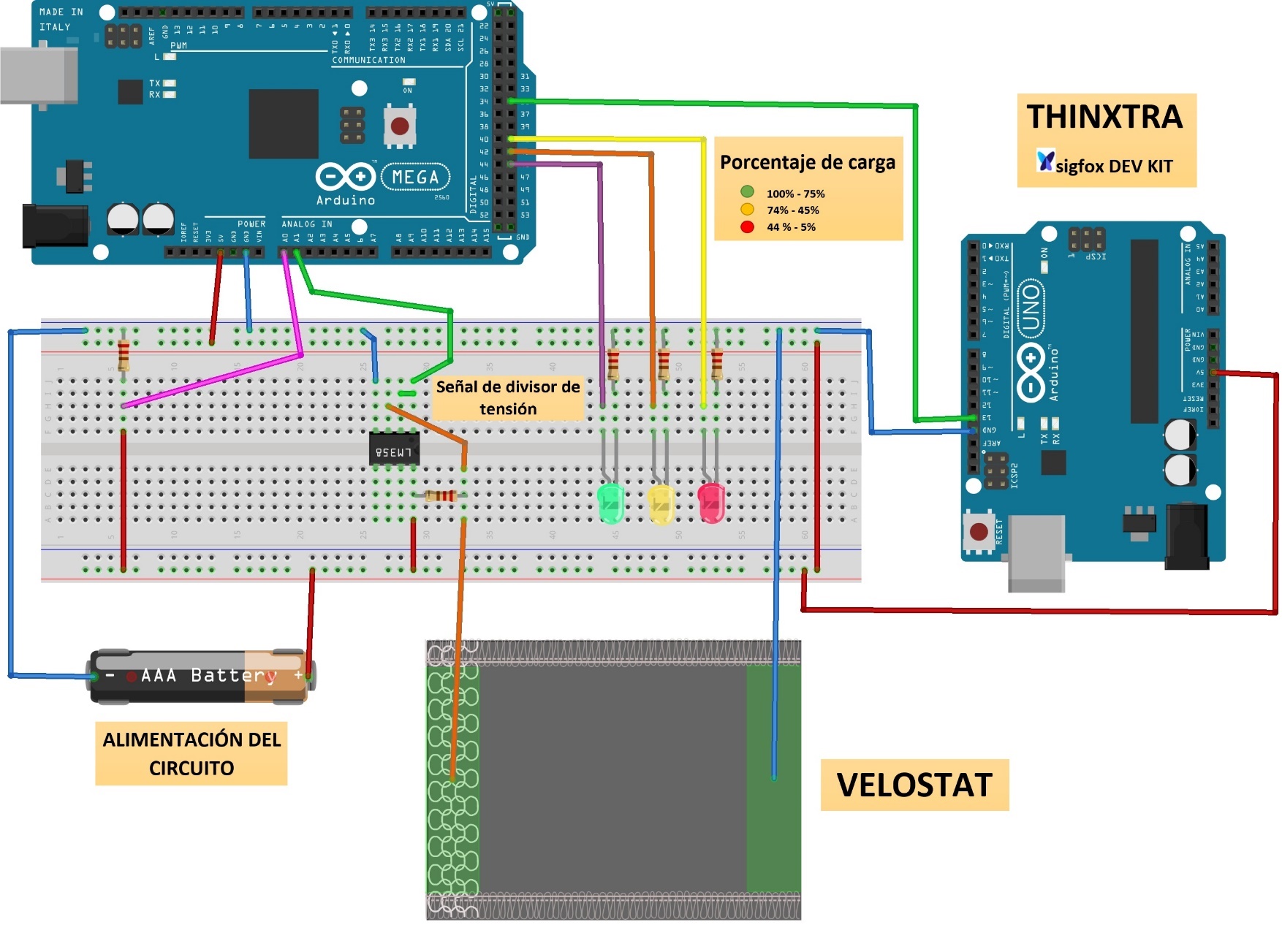


*Ilustración 8 Sensor puesto en la caja.*



*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.

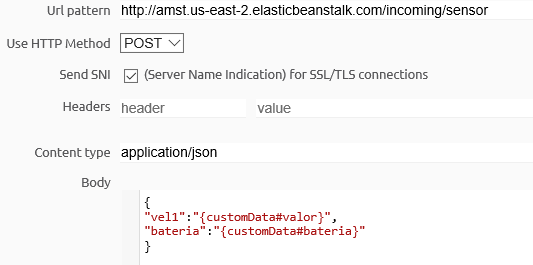


Figura. Configuracion del Backend de Sigfox para Callback de nuestro proyecto.

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 🡪 Estable

ME 🡪 Medio

VA 🡪 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.

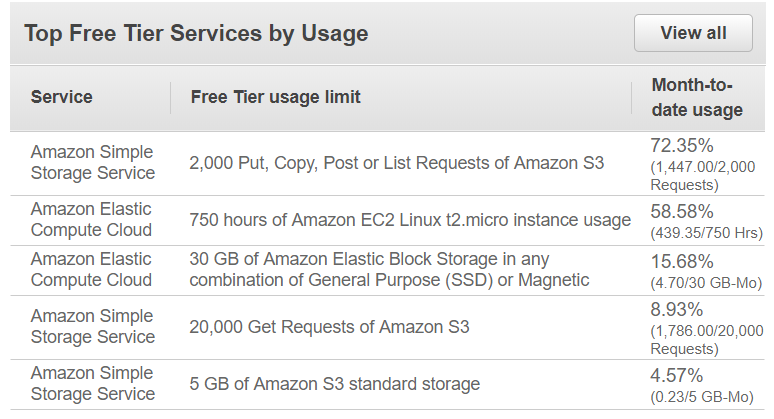


Ilustración 1 Recursos consumidos durante 1 mes de prueba de aplicación

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.

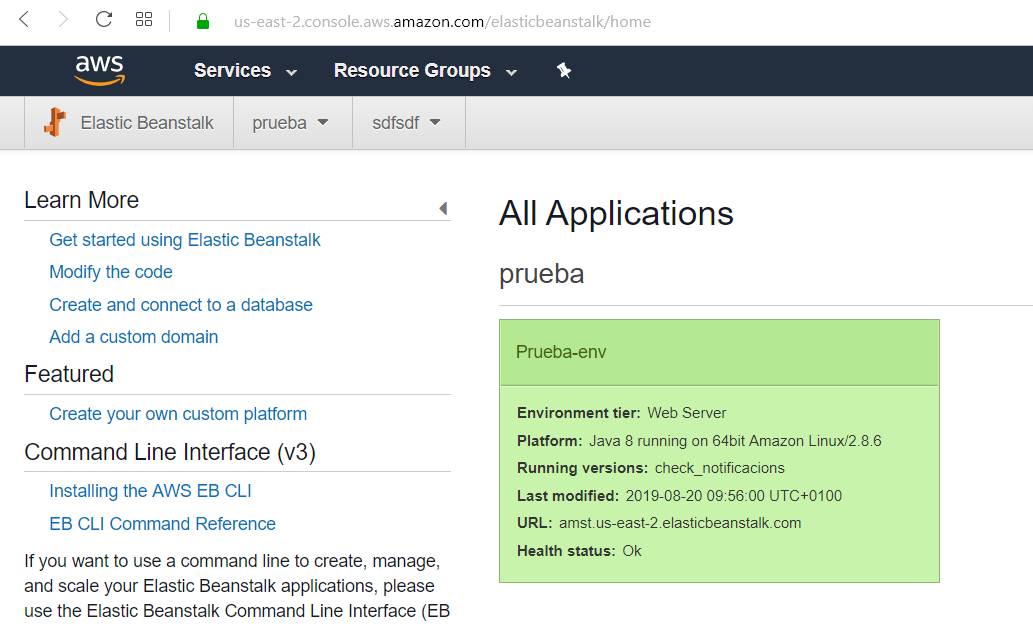


Ilustración 2 Ambiente de Free Tier AWS

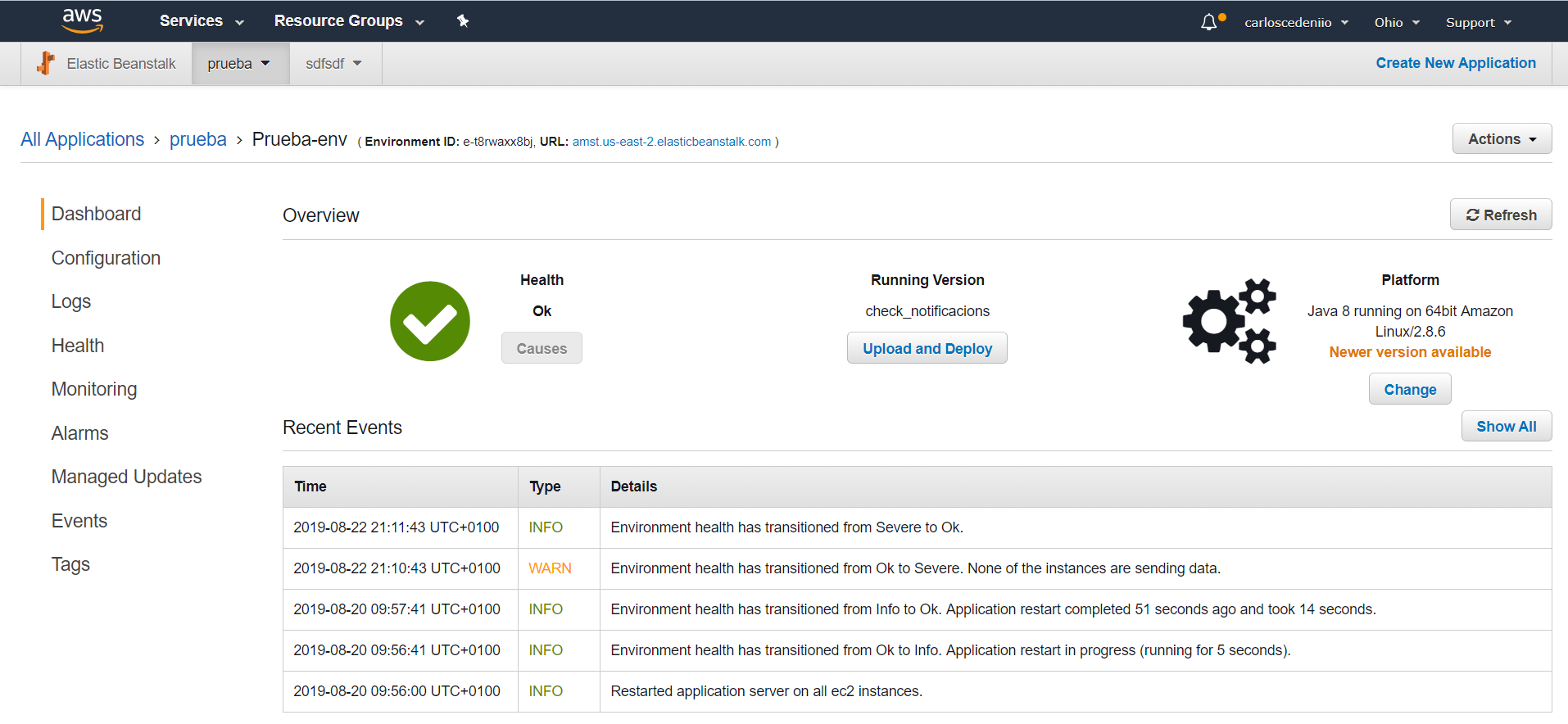


Ilustración 3 Contenedor de JAVA para SPRING

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)

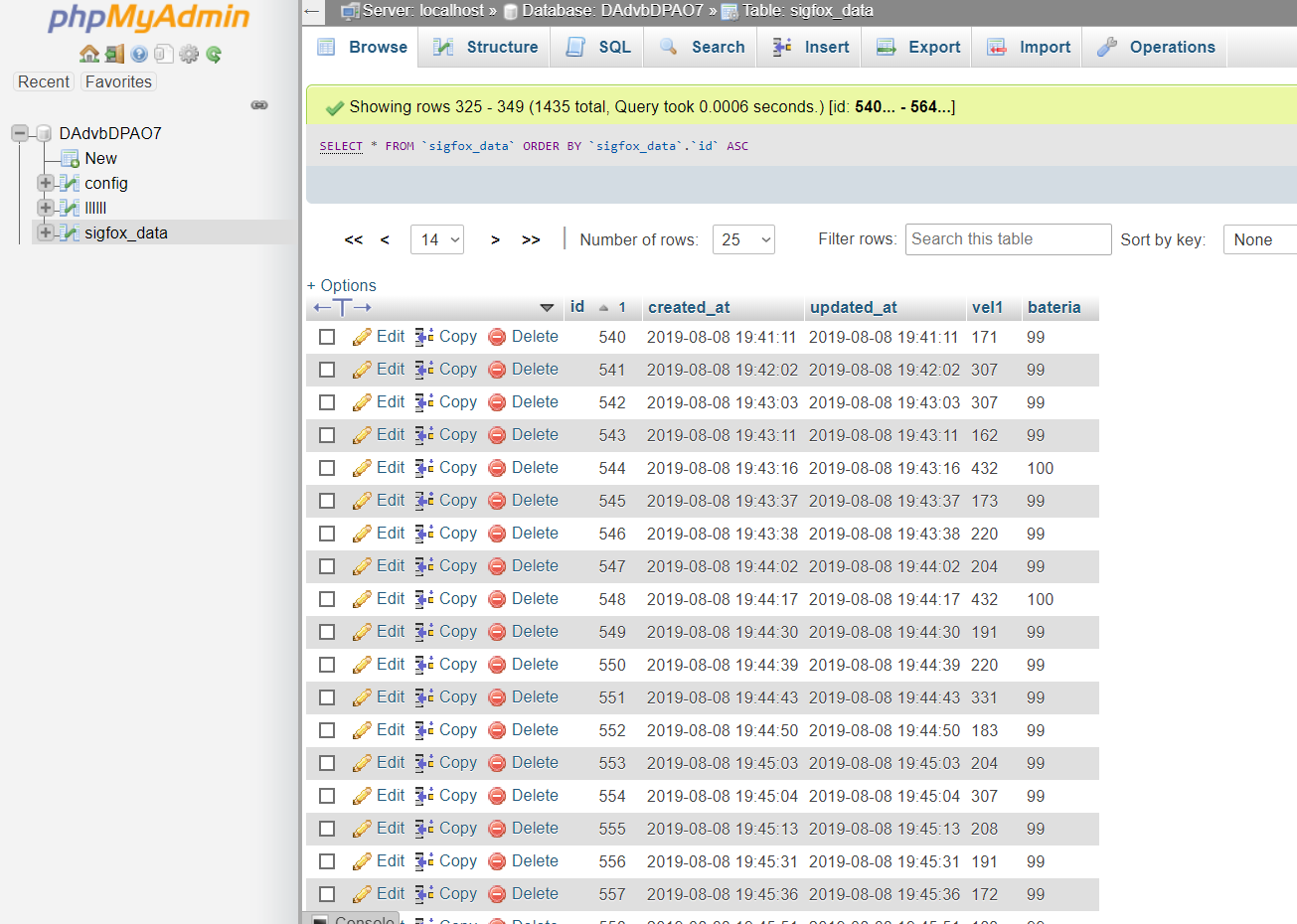
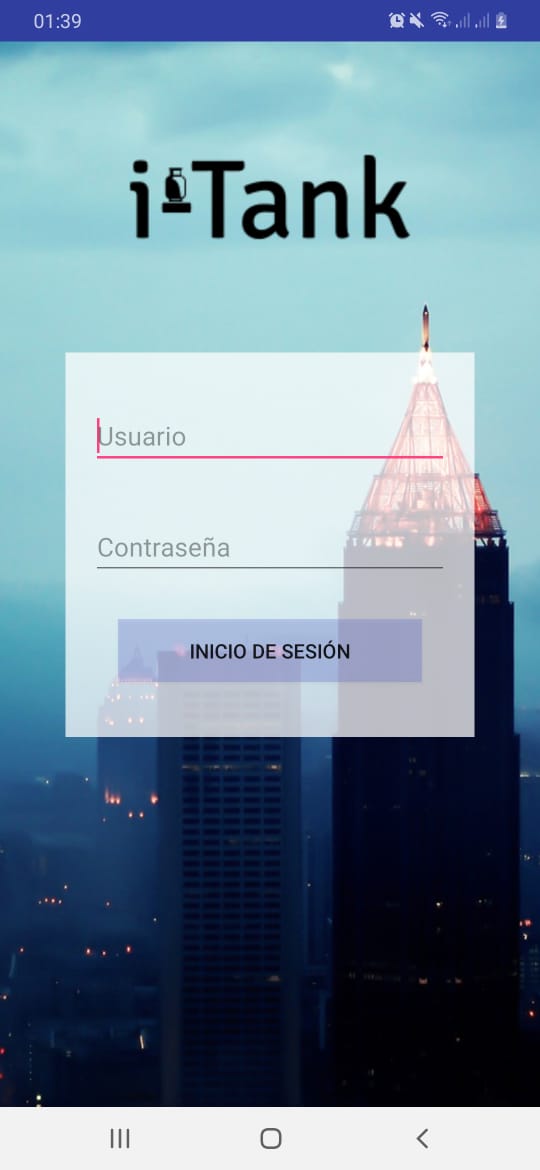
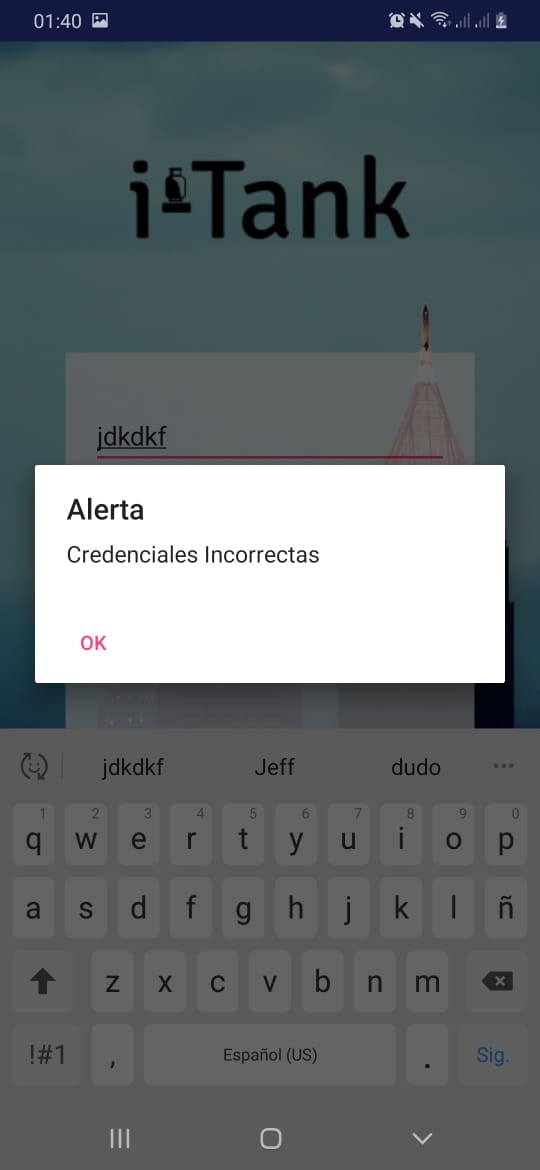


Ilustración 4 Vista del phpMyAdmin de BD alterna

# Screenshoots de la aplicación



VENTANA 1. INICIO DE SESION

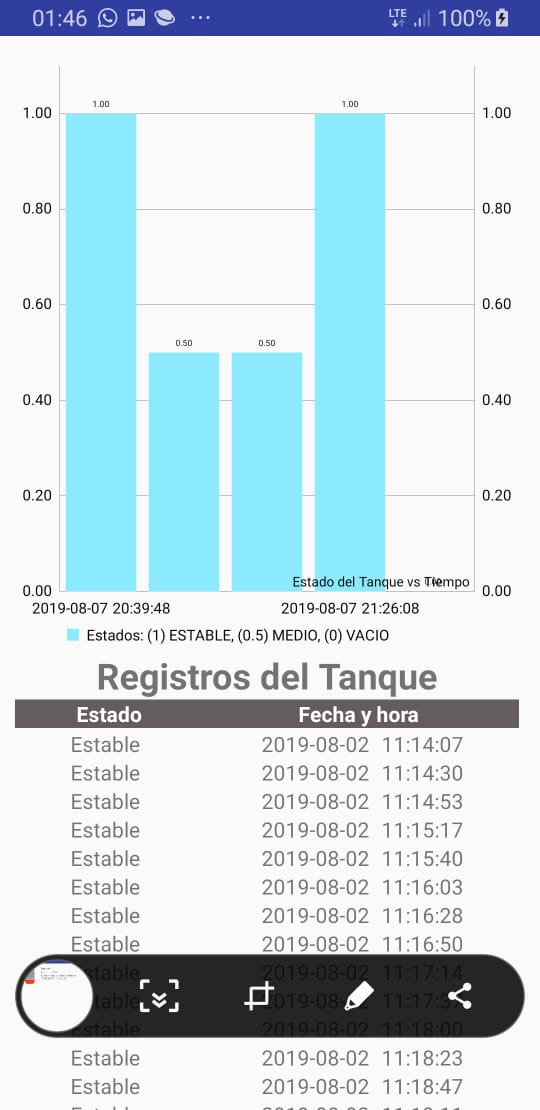


ALERTA. VERIFICACION DE CREDENCIALES

VENTANA 2. MENU PRINCIPAL DEL APLICATIVO



VENTANA 3. LISTA DE TANQUES REGISTRADOS (VISUALIZACION DEL USUARIO)



VENTANA 4. PRESENTACION DE UN REGITRO HISTORICO A TRAVES DEL TIEMPO MEDIANTE GRAFICA DE BARRAS

# Códigos en Android Studio

|  |  |
| --- | --- |
| **Nombre de archivo:** menú.class  **Nombre de función:** initializeComponents ()  **Descripción:** Esta función permite presentar en tiempo real, la información de la batería en nuestro dispositivo IoT mediante 2 barras (circular y rectangular) ubicadas en la ventana principal del aplicativo (menú). | |
|  | **public void** initializeComponents(){  **final** ProgressBar bateria = (ProgressBar) findViewById(R.id.***progressBar2***);  **final** ProgressBar bateria\_circulo = (ProgressBar) findViewById(R.id.***progressBar3***);  **final** TextView porcentaje = findViewById(R.id.***porcentaje\_bateria***);  String url1 = **" https://amstdb.herokuapp.com/db/dispositivo/7"**;   JsonObjectRequest request = **new** JsonObjectRequest(   Request.Method.***GET***, url1, **null**,   **new** Response.Listener<JSONObject>() {   @Override  **public void** onResponse(JSONObject response) {   **try** {  String nivel\_bateria = response.getString(**"bateria"**);  bateria\_circulo.setProgress(Integer.*parseInt*(nivel\_bateria));  bateria.setProgress(Integer.*parseInt*(nivel\_bateria));  porcentaje.setText(nivel\_bateria+**"%"**);   } **catch** (Exception e) {  e.printStackTrace();  }   }  }, **new** Response.ErrorListener() {  @Override  **public void** onErrorResponse(VolleyError error) {  }  }) {  @Override  **public** Map<String, String> getHeaders() **throws** AuthFailureError {  Map<String, String> params = **new** HashMap<String, String>();  params.put(**"Authorization"**, **"JWT "** + **token**);  **return** params;  }  };  **mQueue**.add(request); **final** Handler handler = **new** Handler();  **final** Runnable runnable = **new** Runnable() {  @Override  **public void** run() {  initializeComponents();  }  };  handler.postDelayed(runnable, 3000);} |

|  |  |
| --- | --- |
| **Nombre de archivo:** EstadoTanque.class  **Nombre de función:** Obtener\_estado\_de\_tanques ()  **Descripción:** Esta función presenta todos los tanques registrados por un usuario con una breve descripción del estado actual para cada uno. En esta ventana se podrá seleccionar cualquiera de estos tanques y accionara una nueva ventana con registros históricos a través del tiempo del tanque escogido. | |
|  | **private void** Obtener\_estado\_de\_tanques() {   String url1 = **" https://amstdb.herokuapp.com/db/registroEstadoTanque"**;   JsonArrayRequest request = **new** JsonArrayRequest(   Request.Method.***GET***, url1, **null**,   **new** Response.Listener<JSONArray>() {   @Override  **public void** onResponse(JSONArray response) {   **try** {  *//Se crea lista de tanques definidos como Categorias* **final** ArrayList<Category> lista\_tanques = **new** ArrayList<>();  ArrayList<String> tanques = ResponseUtils.*obtenerListaTanques*(response);  *//Se obtiene informacion de todos los tanques pero solo se usa la ultima actualizacion que se realizo  //para poder presentar al usuario mediante el metodo obtenerUltimoRegistro* **for** (String id\_tanque : tanques) {  JSONObject x1 = (JSONObject) ResponseUtils.*obtenerUltimoRegistro*(id\_tanque,response);  *//Se crea el tanque con una figura representativa de su estado y se lo añade a la lista* Drawable imagen = (Drawable) *obtener\_imagen\_estado\_del\_tanque*(x1.getString(**"estado"**),**actividad**);  Category elemento\_tanque = **new** Category(x1.getString(**"estado"**),  x1.getString(**"fechaRegistro"**),  x1.getString(**"tanque"**), imagen);  lista\_tanques.add(elemento\_tanque);*//tanque añadido* }   *//Se presenta la informacion de los tanques en un ListView* ListView lv = (ListView) findViewById(R.id.***listView***);  AdapterItem adapter = **new** AdapterItem(**actividad**, lista\_tanques);  lv.setAdapter(adapter);  lv.setOnItemClickListener(**new** AdapterView.OnItemClickListener() {  @Override  **public void** onItemClick(AdapterView<?> parent, View view, **int** position, **long** id) {  *//Si el tanque es clickeado en la lista entonces se abre una ventana que  //muestra el registro historico de ese tanque* Intent tanque = **new** Intent(getBaseContext(),registroHistorico.**class**);  *//Se añade el ID del tanque clickeado a la otra ventana para saber sobre que tanque  //hay que presentar la informacion* tanque.putExtra(**"id\_tanque"**,lista\_tanques.get(position).getTitle());  tanque.putExtra(**"token"**,**token**);  startActivity(tanque);  }  });   } **catch** (Exception e) {  e.printStackTrace();  }   }  }, **new** Response.ErrorListener() {  @Override  **public void** onErrorResponse(VolleyError error) {  }  }) {  @Override  **public** Map<String, String> getHeaders() **throws** AuthFailureError {  Map<String, String> params = **new** HashMap<String, String>();  params.put(**"Authorization"**, **"JWT "** + **token**);  **return** params;  }  };  **mQueue**.add(request);   **final** Handler handler = **new** Handler();  **final** Runnable runnable = **new** Runnable() {  @Override  **public void** run() {  Obtener\_estado\_de\_tanques();  }  };  handler.postDelayed(runnable, 3000); } |

|  |  |
| --- | --- |
| **Nombre de archivo:** registroHistorico.class  **Nombre de función:** presentar\_grafico ()  **Descripción:** Esta función obtiene los 5 ultimos registros y recolecta esta información en forma numérica para presentar un gráfico de barras sobre los estados más actuales de nuestro tanque. | |
|  | **public void** presentar\_grafico(JSONArray response) {   ArrayList<BarEntry> entradas = **new** ArrayList<>();  ArrayList<String> etiquetas = **new** ArrayList<>();  **int** contador = 0;  **try** {  *//Se obtiene todos los registros de un tanque en especifico. Esto se realiza gracias al ID  //guardado como parametro principal de esta ventana* JSONArray respuesta = (JSONArray) ResponseUtils.*obtenerRegistrosTanque*(**id\_tanque**, response);  **int** len = respuesta.length();  *//Presenta los 5 ultimos datos mas actuales en la grafica* **for** (**int** i = len - 5; i < len; i++) {  JSONObject p = (JSONObject) respuesta.get(i);  String temp1 = (String) p.get(**"estado"**).toString();  String[] fecha = p.get(**"fechaRegistro"**).toString().split(**"T"**);  String temp2 = (String) fecha[0]+**"\n"**+fecha[1].split(**"\\."**)[0];  *//Se presenta la informacion del estado de tanque en forma numerica  /\*  1 --- Representa estado lleno  0.5 --- Representa estaco medio  0 --- Representa estado vacio  \*/* **if** (temp1.equals(**"ES"**)) {  entradas.add(**new** BarEntry(1, contador));  etiquetas.add(temp2);  contador++;  }**else if**(temp1.equals(**"ME"**)){  entradas.add(**new** BarEntry(0.5f, contador));  etiquetas.add(temp2);  contador++;  }**else** {  entradas.add(**new** BarEntry(0, contador));  etiquetas.add(temp2);  contador++;  }  }  iniciargrafico();*//Se prepara el grafico para poder instanciarlo* llenargrafico(entradas,etiquetas);*//Se llena grafico con informacion relacionada* } **catch** (JSONException e) {  e.printStackTrace();  } } |
| **Nombre de archivo:** registroHistorico.class  **Nombre de función:** llenar\_grafico ()  **Descripción:** Esta función permite presentar en tiempo real, los datos obtenidos mediante un gráfico de barras sobre los estados más actuales de nuestro tanque. Se cargan estos datos mediante el comando graficoBarras.setData(datos); | |
|  | **private void** llenargrafico(ArrayList<BarEntry> entradas,ArrayList<String> etiquetas) {  BarChart graficoBarras = findViewById(R.id.***barChart***);  BarDataSet dataset;   **if** ( graficoBarras.getData() == **null**) {  graficoBarras.animateY(1500);  }  */\*Se instancia el grafico con los datos enviados a traves de las listas  Entradas --- Cantidades numericas del estado del tanque 1/0.5/0  Etiquetas --- Informacion de fecha y hora para ese estado  \*\*\* Las etiquetas se colocan en la parte inferior de cada barra en el grafico para mayor  \*\*\* interactividad con el usuario  \*/* dataset = **new** BarDataSet(entradas, **"Estados: (1) ESTABLE, (0.5) MEDIO, (0) VACIO"**);  BarData datos = **new** BarData(etiquetas, dataset);  graficoBarras.setData(datos);  graficoBarras.invalidate();   *//Se mantiene actualizada la informacion cada 3 segundos una vez que los datos en el grafico han sido presentados* **final** Handler handler = **new** Handler();  **final** Runnable runnable = **new** Runnable() {  @Override  **public void** run() {  Obtener\_estado\_de\_tanque();  }  };  handler.postDelayed(runnable, 3000); } |

|  |  |
| --- | --- |
| **Nombre de archivo:** registroHistorico.class  **Nombre de función:** presentar\_datos ()  **Descripción:** Esta función permite presentar los registros históricos obtenidos mediante la lectura en la base de datos pero de forma escrita. De este modo se expone al usuario el estado del tanque en una fecha y hora indicada. | |
|  | **public void** presentar\_datos(JSONArray response){  LinearLayout nuevoRegistro;  TextView fechaRegistro;  TextView valorRegistro;  **contenedorTemperaturas** = findViewById(R.id.***cont\_temperaturas***);  *//Se configura los parametros de un LinearLayout para agregar informacion redactada* LinearLayout.LayoutParams parametrosLayout = **new** LinearLayout.LayoutParams(  LinearLayout.LayoutParams.***WRAP\_CONTENT***,  LinearLayout.LayoutParams.***WRAP\_CONTENT***, (**float**) 1);  **try** {  JSONArray respuesta = (JSONArray) ResponseUtils.*obtenerRegistrosTanque*(**id\_tanque**, response);  **int** len =respuesta.length();   **for** (**int** i = 1; i <= len; i++) {  JSONObject registroTemp = (JSONObject) respuesta.get(len-i);  String registroId = registroTemp.getString(**"id"**);  */\*Se consulta si el ID del registro ya ha esta presentada en la ventana  Informacion no presentada?  TRUE---Se crean TextViews para presentar estado y fecha de registro y luego se los agrega al  LinearLayout principal de la ventana  FALSE---NOTHING---Ya estan presentados  \*/* **if**( !**this**.**Dic\_Estados**.containsKey(registroId) && !**this**.**Dic\_fechas**.containsKey(registroId) ){  nuevoRegistro = **new** LinearLayout(**this**);  nuevoRegistro.setOrientation(LinearLayout.***HORIZONTAL***);  valorRegistro = **new** TextView(**this**);  valorRegistro.setLayoutParams(parametrosLayout);  String estado = registroTemp.getString(**"estado"**);  String imprime\_estado = **""**;   **if** (estado.equals(**"ES"**)){  imprime\_estado = **"Estable"**;  }**else if**(estado.equals(**"ME"**)){  imprime\_estado = **" Medio"**;  }**else** imprime\_estado = **" Vacio"**;   valorRegistro.setText(imprime\_estado+**"\t\t"**);  valorRegistro.setGravity(Gravity.***CENTER***);   **if** (estado.equals(**"VA"**)){  valorRegistro.setTextColor(Color.***RED***);  }**else if**(estado.equals(**"ME"**)){  valorRegistro.setTextColor(Color.*parseColor*(**"#FF8E96C8"**));  }   nuevoRegistro.addView(valorRegistro);  fechaRegistro = **new** TextView(**this**);  fechaRegistro.setLayoutParams(parametrosLayout);  String[] fecha = registroTemp.getString(**"fechaRegistro"**).split(**"T"**);  String cadenafechayhora = (String) fecha[0]+**"\t\t"**+fecha[1].split(**"\\."**)[0];  fechaRegistro.setText(cadenafechayhora);  fechaRegistro.setGravity(Gravity.***CENTER***);  nuevoRegistro.addView(fechaRegistro);   **if** (**primera\_instancia**){  **contenedorTemperaturas**.addView(nuevoRegistro);  }**else contenedorTemperaturas**.addView(nuevoRegistro,0);   **this**.**Dic\_fechas**.put(registroId, fechaRegistro);  **this**.**Dic\_Estados**.put(registroId, valorRegistro);  }  }   **primera\_instancia**=**false**;   } **catch** (JSONException e) {  e.printStackTrace();  } } |

|  |  |
| --- | --- |
| **Nombre de archivo:** Estadisticas.class  **Nombre de función:** Obtener\_estado\_de\_tanques ()  **Descripción:** Esta función permite presentar en tiempo real, la información estadística de los estados para la cantidad de tanques estables (llenos), medios y vacíos que estén registrados. De este modo, se presenta un porcentaje de tanques llenos y un porcentaje de tanques vacíos que se exponen al usuario mediante un diagrama de pastel. | |
|  | **public void** Obtener\_estado\_de\_tanques() {  String url1 = **" https://amstdb.herokuapp.com/db/registroEstadoTanque"**;  **final** JSONArray[] responseR = **new** JSONArray[1];   JsonArrayRequest request = **new** JsonArrayRequest(   Request.Method.***GET***, url1, **null**,   **new** Response.Listener<JSONArray>() {   @Override  **public void** onResponse(JSONArray response) {   **try** {  **int** uno = ResponseUtils.*obtenerEstadisticas*(response)[0];  **int** dos = ResponseUtils.*obtenerEstadisticas*(response)[1];  PieChart pieChart;  pieChart = (PieChart) findViewById(R.id.***pieChart***);  **if**(pieChart.getData()==**null**){  pieChart.animateXY(1500, 1500);  }  */\*definimos algunos atributos\*/* pieChart.setHoleRadius(35f);  */\*creamos una lista para los valores Y\*/* ArrayList<Entry> valsY = **new** ArrayList<Entry>();  valsY.add(**new** Entry((uno\*100)/(uno+dos),0));  valsY.add(**new** Entry((dos\*100)/(uno+dos),1));  */\*creamos una lista para los valores X\*/* ArrayList<String> valsX = **new** ArrayList<String>();  valsX.add(**"% Estables"**);  valsX.add(**"% Vacios"**);  */\*creamos una lista de colores\*/* ArrayList<Integer> colors = **new** ArrayList<Integer>();  colors.add(Color.*parseColor*(**"#FFA0B1A6"**));  colors.add(Color.*parseColor*(**"#FFA8857E"**));  */\*seteamos los valores de Y y los colores\*/* PieDataSet set1 = **new** PieDataSet(valsY,**""**);  set1.setColors(colors);  */\*seteamos los valores de X\*/* PieData data = **new** PieData(valsX, set1);  data.setValueTextSize(16f);  data.setValueTextColor(Color.***WHITE***);  pieChart.setData(data);  ArrayList<String> tanques = ResponseUtils.*obtenerListaTanques*(response);  pieChart.setDescription(**"Las estadisticas se obtuvieron de los "**+tanques.size()+**" tanques registrados"**);  } **catch** (Exception e) {  e.printStackTrace();  }   }  }, **new** Response.ErrorListener() {  @Override  **public void** onErrorResponse(VolleyError error) {  }  }){  @Override  **public** Map<String, String> getHeaders() **throws** AuthFailureError  {  Map<String, String> params = **new** HashMap<String, String>();  params.put(**"Authorization"**, **"JWT "** + *token*);  **return** params;  }  };  *mQueue*.add(request);   *//Se actualiza el grafico cada 3 segundos* **final** Handler handler = **new** Handler();  **final** Runnable runnable = **new** Runnable() {  @Override  **public void** run() {  Obtener\_estado\_de\_tanques();  }  };  handler.postDelayed(runnable, 3000); } |

|  |  |
| --- | --- |
| **Nombre de archivo:** ResponseUtils.class  **Nombre de función:** obtenerListaTanques ()  **Descripción:** Esta función permite obtener una lista con todos los tanques registrados de manera única (sin repetirse). | |
|  | **public static** ArrayList<String> obtenerListaTanques(JSONArray response) **throws** JSONException {  ArrayList<String> tanques = **new** ArrayList<>();  **for** (**int** i = 0; i < response.length(); i++) {  JSONObject p = (JSONObject) response.get(i);  String temp = (String) p.get(**"tanque"**).toString();  **if** (!tanques.contains(temp)) {  tanques.add(temp);  }  }  **return** tanques; } |

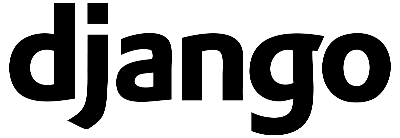
|  |  |
| --- | --- |
| **Nombre de archivo:** ResponseUtils.class  **Nombre de función:** obtenerRegistrosTanque ()  **Descripción:** Esta función permite obtener un arreglo de respuestas JSON con todos los registros de un tanque en específico y de manera ordenada por la clave “fecha\_de\_registro”. | |
|  | **public static** JSONArray obtenerRegistrosTanque(String id, JSONArray response) **throws** JSONException {  JSONArray registro = **new** JSONArray();  **for** (**int** i = 0; i < response.length(); i++) {  JSONObject p = (JSONObject) response.get(i);  String temp = (String) p.get(**"tanque"**).toString();  **if** (id.equals(temp)) {  registro.put((JSONObject) response.get(i));  }  }   *//AQUI comienza el proceso de ordenado de la informacion para facilidad de recursos* JSONArray sortedJsonArray = **new** JSONArray();  List<JSONObject> jsonValues = **new** ArrayList<JSONObject>();   **for** (**int** i = 0; i < registro.length(); i++) {  jsonValues.add(registro.getJSONObject(i));  }   Collections.*sort*( jsonValues, **new** Comparator<JSONObject>() {  *//You can change "Name" with "ID" if you want to sort by ID* **private static final** String ***KEY\_NAME*** = **"fechaRegistro"**;   @Override  **public int** compare(JSONObject a, JSONObject b) {  String valA = **new** String();  String valB = **new** String();   **try** {  valA = (String) a.get(***KEY\_NAME***);  valB = (String) b.get(***KEY\_NAME***);  }  **catch** (JSONException e) { }   **return** valA.compareTo(valB);  *//if you want to change the sort order, simply use the following:  //return -valA.compareTo(valB);* }  });   **for** (**int** i = 0; i < registro.length(); i++) {  sortedJsonArray.put(jsonValues.get(i));  }   **return** sortedJsonArray; } |

|  |  |
| --- | --- |
| **Nombre de archivo:** ResponseUtils.class  **Nombre de función:** obtenerEstadisticas ()  **Descripción:** Esta función obtiene la cantidad de tanques estables o medios y la cantidad de vacíos para presentar al usuario una breve estadística de los estados de sus tanques. | |
|  | **public static** Integer[] obtenerEstadisticas(JSONArray response) **throws** JSONException {  ArrayList<String> tanques = ResponseUtils.*obtenerListaTanques*(response);  ArrayList<Integer> estadisticas = **new** ArrayList<>();  estadisticas.add(0);  estadisticas.add(0);  **for** (String x : tanques) {  JSONObject x1 = (JSONObject) ResponseUtils.*obtenerUltimoRegistro*(x, response);  **if** (x1.getString(**"estado"**).equals(**"ES"**) || x1.getString(**"estado"**).equals(**"ME"**)) {  estadisticas.set(0, estadisticas.get(0) + 1);  } **else** {  estadisticas.set(1, estadisticas.get(1) + 1);  }  }  Integer[] datos = {estadisticas.get(0), estadisticas.get(1)};  **return** datos; } |

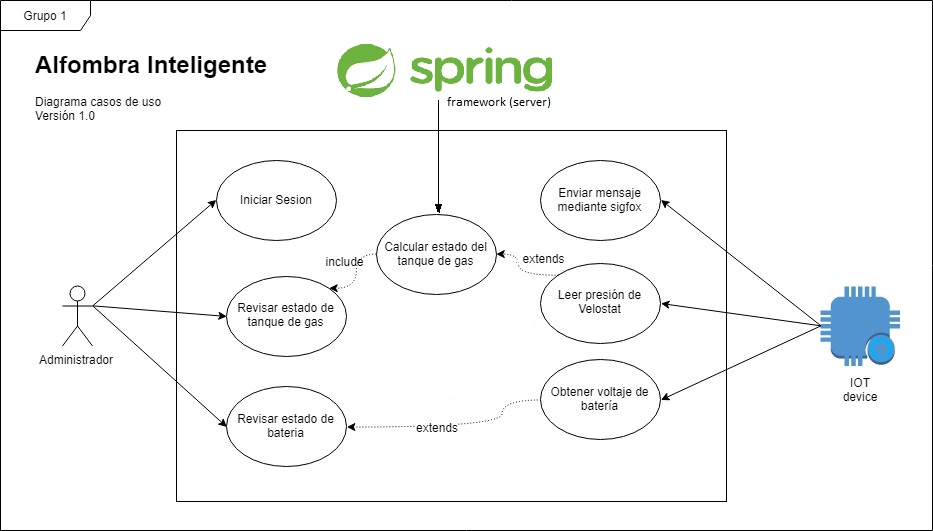
|  |  |
| --- | --- |
| **Nombre de archivo:** ResponseUtils.class  **Nombre de función:** obtener\_imagen\_estado\_del\_tanque ()  **Descripción:** Esta función retorna un componente de imagen de acuerdo al estado del tanque. De esto se presenta al usuario una imagen apropiada que describa el estado actual de cada tanque en una forma sencilla y fácil de distinguir. | |
|  | **public static** Drawable obtener\_imagen\_estado\_del\_tanque(String porcentaje\_tanque, Activity actividad) {  Resources res = actividad.getResources();  **if** (porcentaje\_tanque.equals(**"ES"**)) {  **return** res.getDrawable(R.drawable.***tanque\_full***);  } **else if** (porcentaje\_tanque.equals(**"ME"**)) {  **return** res.getDrawable(R.drawable.***tanque\_medio***);  } **else return** res.getDrawable(R.drawable.***tanque\_low***); } |

|  |  |
| --- | --- |
| **Nombre de archivo:** MyFirebaseInstanceService.class  **Nombre de función:** obtener\_imagen\_estado\_del\_tanque ()  **Descripción:** Esta función presenta una notificación externa al aplicativo sobre el estado de batería critico en nuestro dispositivo IoT mediante la consola de FireBase de Google para comentar al usuario registrado sobre el cambio de batería pertinente para la correcta funcionalidad del aplicativo. | |
|  | **private void** showNotification(String title, String body){  NotificationManager notificationManager =  (NotificationManager) getSystemService(Context.***NOTIFICATION\_SERVICE***);  String NOTIFICATION\_CHANNEL\_ID = **"com.amst.firebasenotify.test"**;  **if**(Build.VERSION.***SDK\_INT*** >= Build.VERSION\_CODES.***O***){  NotificationChannel notificationChannel =  **new** NotificationChannel(NOTIFICATION\_CHANNEL\_ID, **"Notification"**,  NotificationManager.***IMPORTANCE\_DEFAULT***);  notificationChannel.setDescription(**"EDMT Channel"**);  notificationChannel.enableLights(**true**);  notificationChannel.setLightColor(Color.***BLUE***);  notificationChannel.setVibrationPattern(**new long**[]{0, 1000, 500, 1000});  notificationManager.createNotificationChannel(notificationChannel);  }  NotificationCompat.Builder notificationBuilder =  **new** NotificationCompat.Builder(**this**, NOTIFICATION\_CHANNEL\_ID);  notificationBuilder.setAutoCancel(**true**)  .setDefaults(Notification.***DEFAULT\_ALL***)  .setWhen(System.*currentTimeMillis*())  .setContentTitle(title)  .setContentText(body)  .setContentInfo(**"info"**);  notificationManager.notify(  **new** Random().nextInt(), notificationBuilder.build()); } |

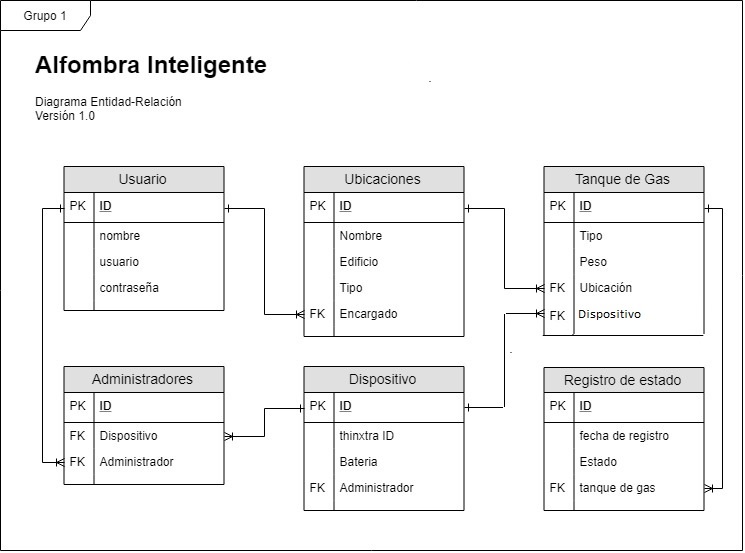
## Propuesta de Solución



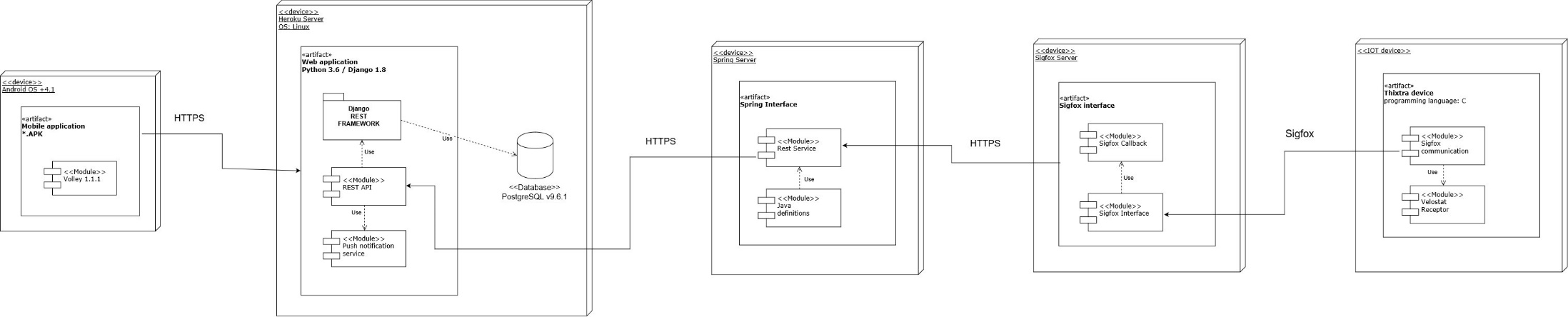
## Diagrama casos de uso



## Diagrama Entidad-Relación



## Diagrama de despliegue

**

## Limitaciones

Para realizar este proyecto se debe tener en cuenta el diseño del sensor con el Velostat debido a su característica de inestabilidad al momento de ejercerse presión sobre el mismo. Haciendo que este material sea muy variable al momento de presentar los resultados.

El uso de la plataforma Sigfox para el envío de datos e información mediante Internet es limitado ya que nos permite un máximo de 140 mensajes diarios lo que se verá afectado a la cantidad de aplicaciones que se pueden usar mediante esta plataforma en nuestros aplicativos móviles.

Uso de lenguaje de programación orientado a objetos JAVA para la realización del programa y de ciertas librerías que permitan realizar las presentaciones de gráficos y multimedia.

# 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

**GITs adicionales**

[**https://desarrolloweb.com/manuales/manual-de-git.html**](https://desarrolloweb.com/manuales/manual-de-git.html)

[**https://github.com/johnspice/jplot-android**](https://github.com/johnspice/jplot-android)

[**https://github.com/Jorgesys/Android-Youtube-API**](https://github.com/Jorgesys/Android-Youtube-API)