**Manual Técnico**

GRUPO #6

Tema: Clasificador de objetos mediante peso

Profesora:

* Msig. Adriana Collaguazo Jaramillo

Integrantes:

* Nécker Iván Espinosa Feijoo
* Sophia Denisse Gómez Quimí
* Steeven Joshua Rodríguez Alvarado
* Jonathan Oswaldo Yagual Peredo

**Descripción del proyecto**

El clasificador de objetos deberá identificar que objeto está encima de la plataforma, conociendo su peso. Para ello se requiere aplicar machine learning. Esta información debe ser registrada y enviada a la nube.

**Descripción del problema**

La banaera EcIOT busca automatizar la distribución de cajas para ubicarlas en el camión respectivo. Actualmente tienen varios productos y cajas con diferentes tamaños (con variaciones de ± 5g). Las cajas de diferentes tamaños se mueven automáticamente en una cinta transportadora y no se dispone de suficiente tiempo para pesarlas y clasificarlas al mismo tiempo.

**Historias de usuarios**

* **Historia de usuario 1: Consulta**

Como usuario puedo clasificar los objetos por su peso respectivo para agilizar el ordenamiento de utensilios.

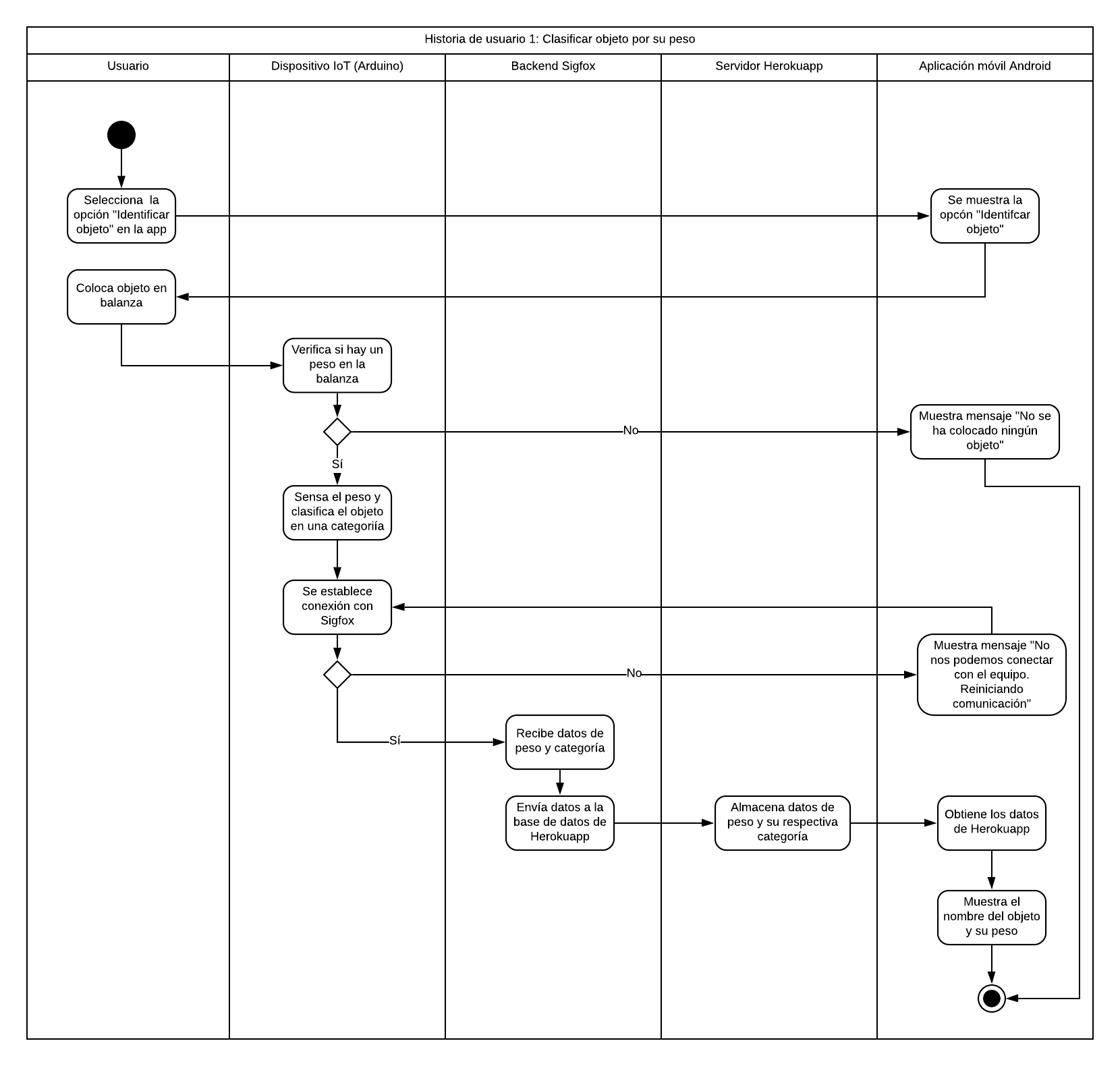


Figura 1 Historia de usuario 1

* **Historia de usuario 2: Reporte**

Como usuario puedo entrenar mi clasificador para mejorar el porcentaje de aciertos actual de mi dispositivo.

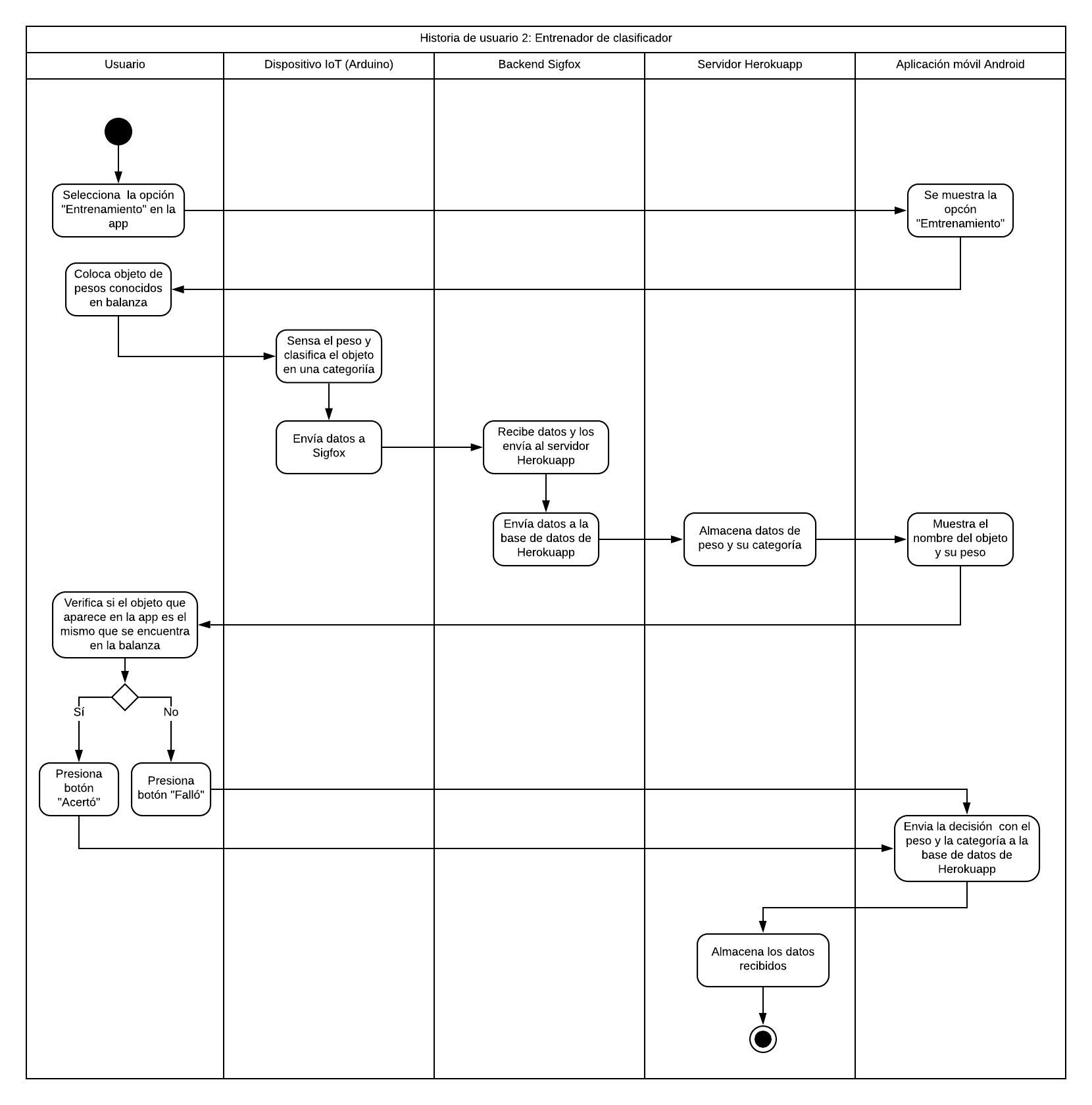


Figura 2 Historia de usuario 2

* **Historia de usuario 3: Loop**

Como usuario puedo requiero que la pesa clasificadora este constantemente registrando el nivel de batería (cada 10 minutos) para saber cuándo debería cambiarla.

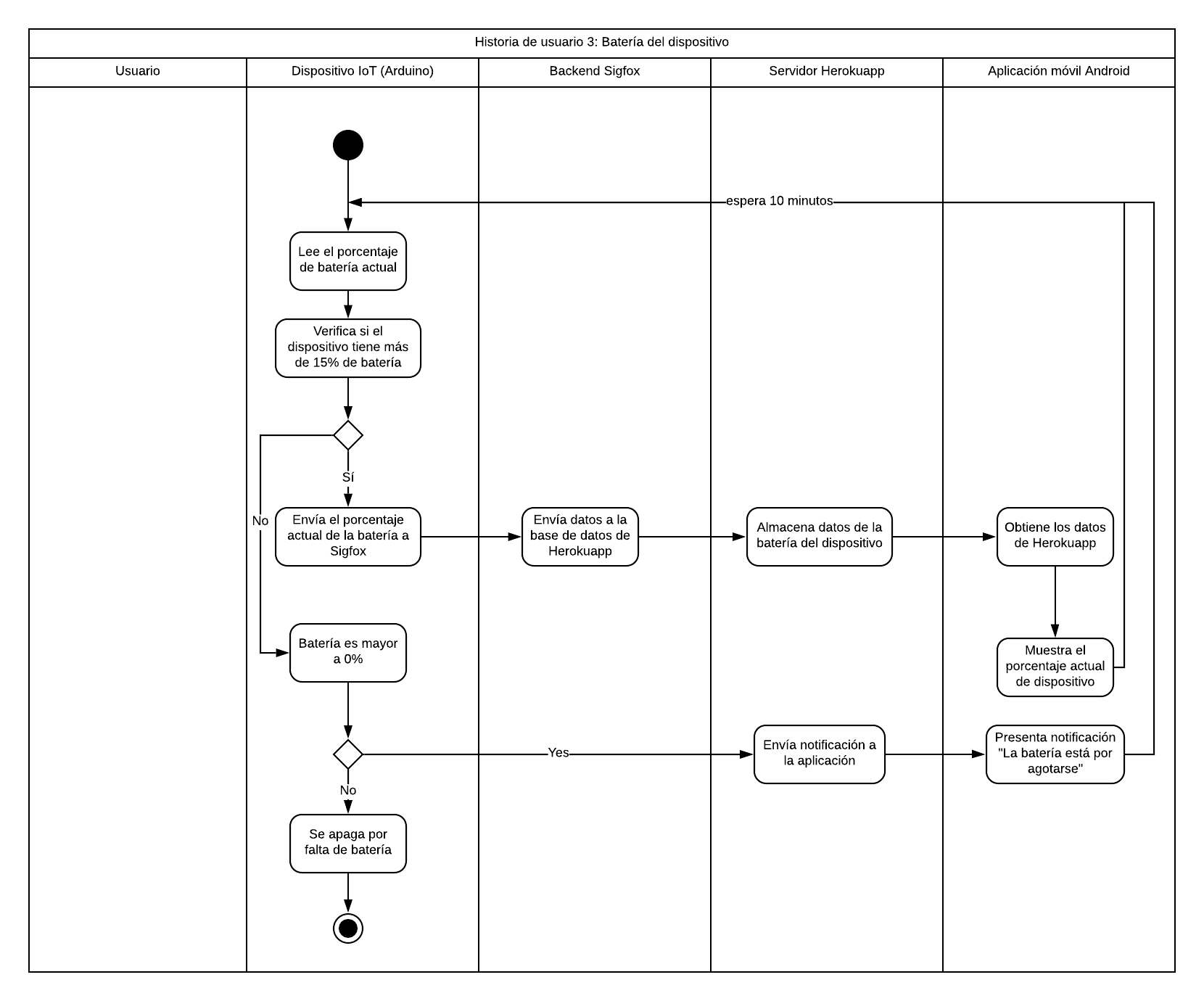


Figura 3 Historia de usuario 3

* **Historia de usuario 4: Start up**

Como usuario puedo requiero encender mi dispositivo IoT e iniciar comunicaciones para para comenzar a utilizar el sistema clasificador.

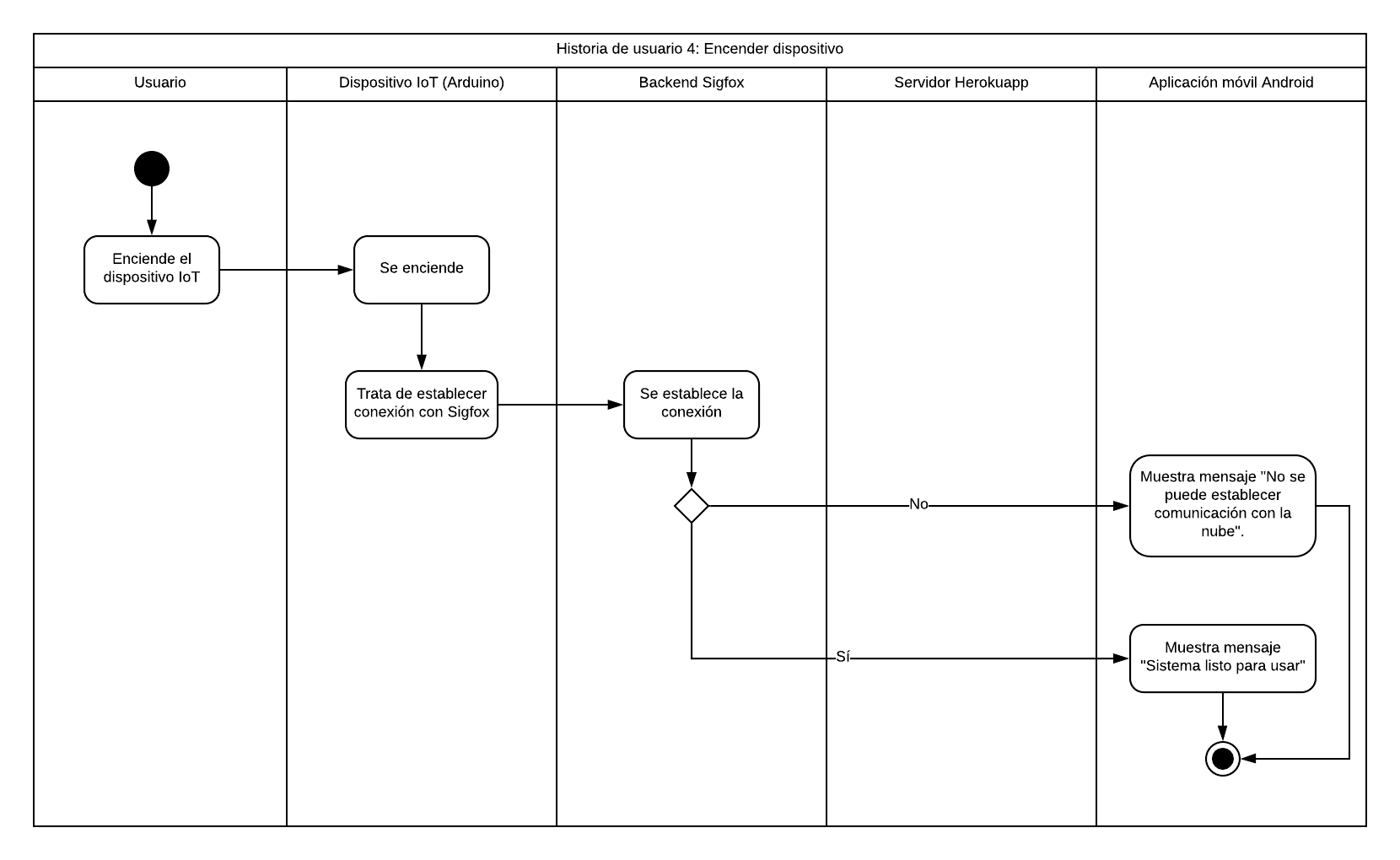


Figura 4 Historia de usuario 4

* **Historia de usuario 5: Shutdown**

Como usuario puedo requiero apagar mi dispositivo cuando termine de usarlo para poder ahorrar energía de la batería.

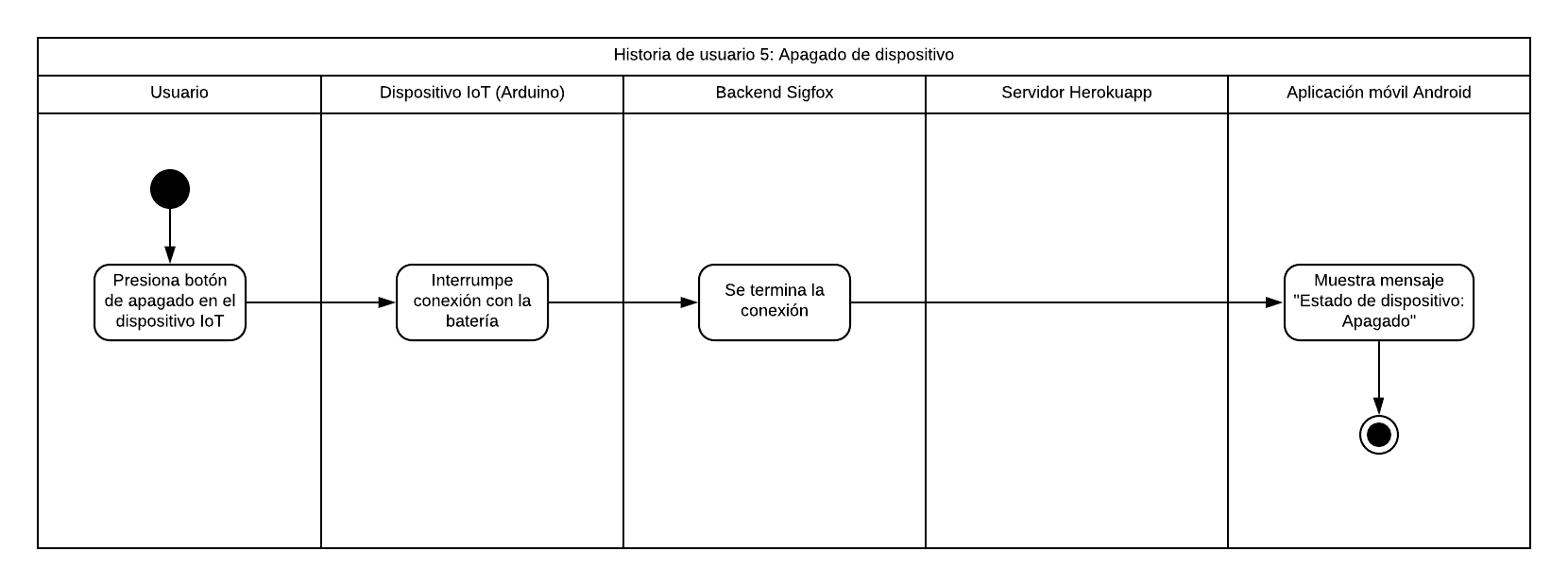


Figura 5 Historia de usuario 5

**Diagrama de despliegue**

El diagrama de despliegue describe la topología del sistema, la estructura de los elementos de hardware y el software que ejecuta cada uno de ellos.

En este caso los dispositivos que interactúan son:

* + El dispositivo Android que usa la librería Volley para conectarse con el servidor.
  + El servidor que contiene la aplicación web en Python/Django y la base de datos PostgreSQL.
  + El sevidor Sigfox que contiene los módulos encargados de recibir los datos sensados en el dispositivo IoT y enviarlos a almacenar a la base de datos.
  + El dispositiov IoT que se integra con el dispositivo Thinktra para la comunicación con el servidor de Sigfox, y además se encarga de la clasificación de los pesos.

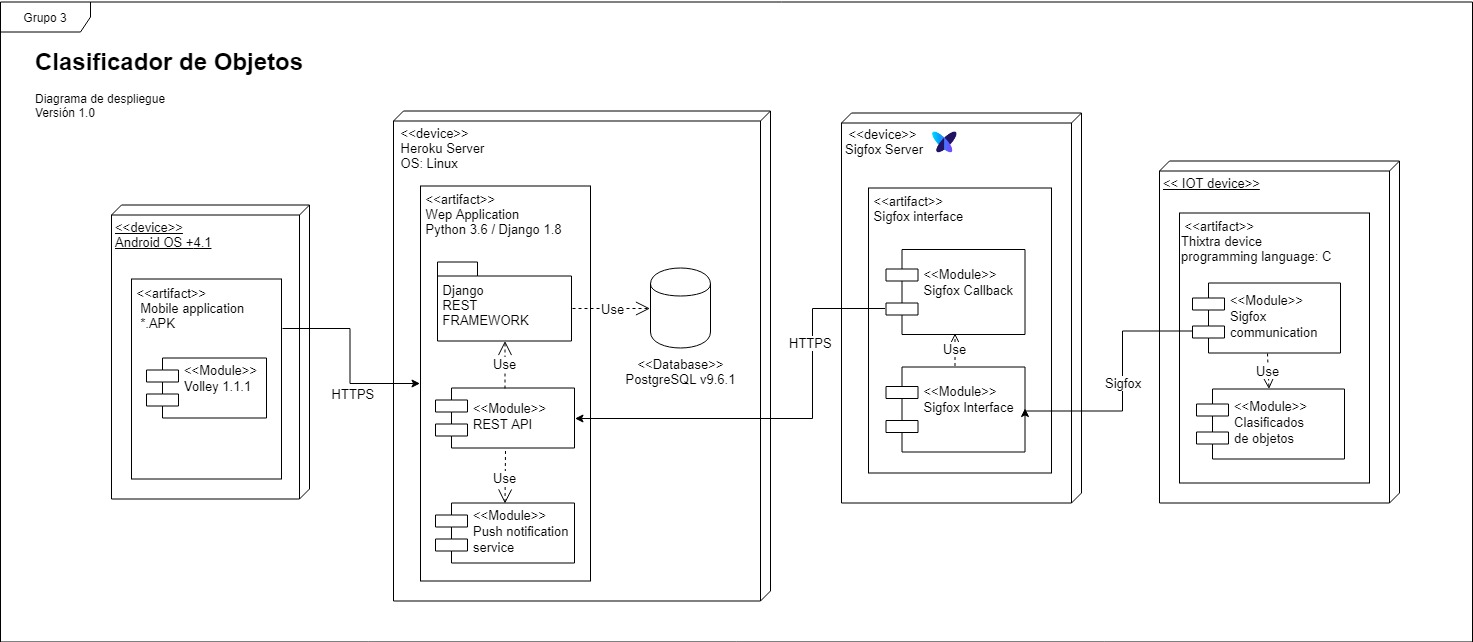


Figura 6 Diagrama de despliegue

**Diagrama de casos de uso**

El diagrama de casos de uso es utilizado para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios u otros sistemas.

En este caso, el actor principal es el usuario que va a usar el sistema clasificador de objeto, este podrá realizar varias actividades que se encuentran especificadas como una opción del menú dentro de la aplicación móvil.

El dispositivo IoT es otro actor dentro del sistema. Este es el encargado de sensar el peso y enviar los datos a sigfox cumple otras funciones que se muestran en la Figura 7.

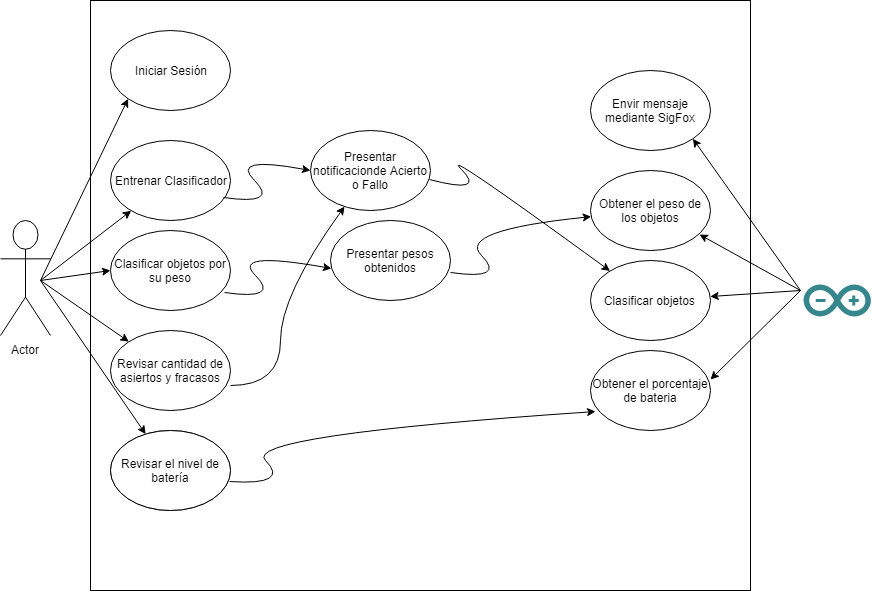


Figura 7 Diagrama de casos de uso

**Diagrama de entidad – relación**

El diagrama entidad-relación es una herramienta que se usa para el modelado de datos. En este se pueden ver las relaciones que existen en las tablas.

En la base de Clasificador de objetos se tienen 5 tablas que es donde se almacena el usuario que tiene acceso a la aplicación, el nivel de batería del dispositivo Iot, las categorías y e l peso de los objetos sensados.

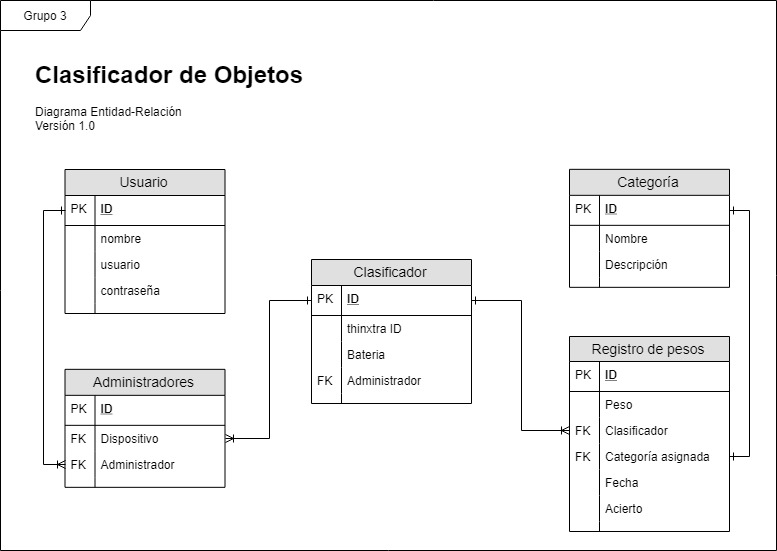


Figura 8 Diagrama Entidad Relación

**Diagrama de clases**

Este diagrama describa la estructura del sistema mostrando sus clases, los atributos y métodos que contienen; además, de la relación entre los objetos, como se puede observar en la Figura 9.

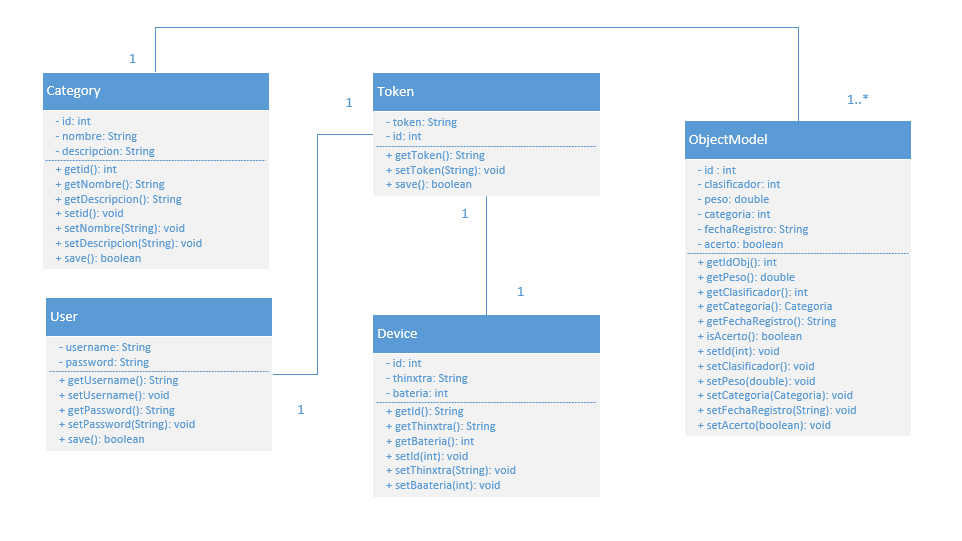


Figura 9 Diagrama de clases

**Diagrama de circuito**

El circuito muestra al arduino conectado con el dispositivo Thinxtra que se encarga de enviar lo que se obtiene del sensor al backend de Sigfox para ser almacenados en la base de datos de Herokuapp y luego poder revisarlos en el aplicativo móvil Android.

Por otro lado el arduino se encuentra alimentado a una batería de 9v, en el pin Vin que soporta de 6v a 12v. El nivel de la batería debe ser visualizado dentro de la aplicación móvil, por lo que este nivel debe ingresar en un pin de entrada analógico del arduino, pero debido a que estos pines soportan un máximo de 5v, se realiza un divisor de voltaje con resistencias de igual valor, por lo que se tienen 4.5v que pasan por un seguidor de voltaje para que no afecte el circuito interno del arduino con las resistencias del divisor. Así, en el pin A3 se tiene 4.5v cuando la batería tiene un 100% de carga es decir tiene 9v, así se reduce el voltaje en el pin A3 proporcionalmente al nivel de la batería.

A su vez al arduino va conectado a la celda de carga HX711 que conectado al sensor de peso de 5kg, verifican la deformación del sensor de peso y después de ser calibrado se obtiene el peso del objeto en el sensor.

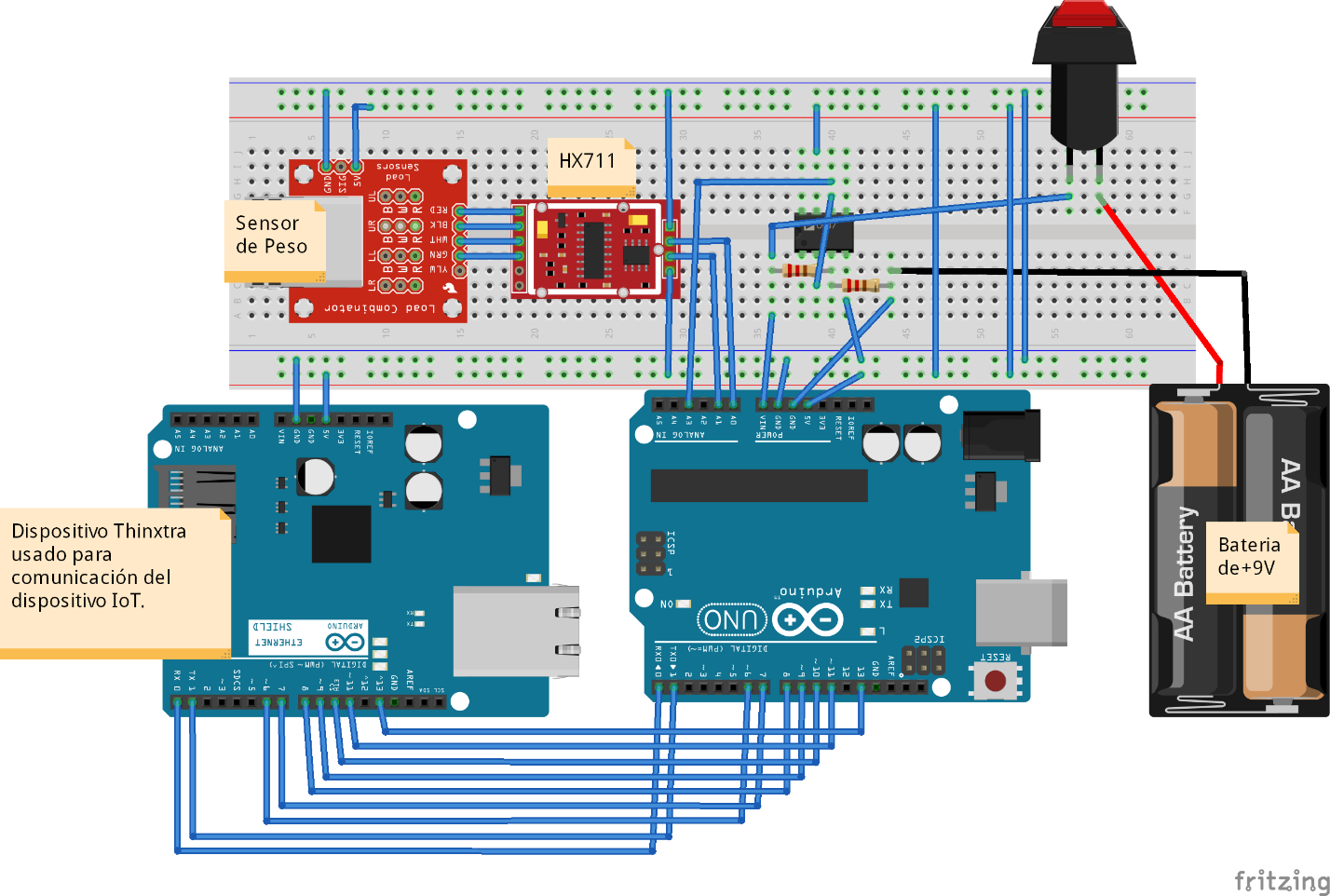


Figura 10 Diagrama de circuito

**Estructura del sistema (código fuente)**

* **Clase Categoría**

Este objeto categoría contiene los atributos id, nombre y descripción que es donde se especifican las categorías que tenemos para el sistema de sensado. Cuenta con sus funciones get y set respectivamente para obtener los atributos del objeto.

**public class** Category **extends** RealmObject{  
 @PrimaryKey  
 @SerializedName(**"id"**)  
 @Expose  
 **private int id**;  
 @SerializedName(**"nombre"**)  
 @Expose  
 **private** String **nombre**;  
 @SerializedName(**"descripcion"**)  
 @Expose  
 **private** String **descripcion**;  
  
 **public int** getId() {  
 **return id**;  
 }  
  
 **public void** setId(**int** id) {  
 **this**.**id** = id;  
 }  
  
 **public** String getNombre() {  
 **return nombre**;  
 }  
  
 **public void** setNombre(String nombre) {  
 **this**.**nombre** = nombre;  
 }  
  
 **public** String getDescripcion() {  
 **return descripcion**;  
 }  
  
 **public void** setDescripcion(String descripcion) {  
 **this**.**descripcion** = descripcion;  
 }  
  
 **public boolean** save() {  
 **boolean** isSuccess = **true**;  
 Realm realm = Realm.*getDefaultInstance*();  
 **try** {  
 realm.beginTransaction();  
 realm.copyToRealmOrUpdate(**this**);  
 realm.commitTransaction();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 isSuccess = **false**;  
 } **finally** {  
 realm.close();  
 }  
  
 **return** isSuccess;  
 }  
  
  
}

* **Clase Object Model (objeto sensado)**

Este objeto Object Model se refiere a los objetos que se sensan. Esta clase tiene los atributos del registro de peso y el id de clasificador se refiere a la categoría a la que pertenece el objeto. Cuenta con sus funciones get y set respectivamente para obtener los atributos del objeto.

**public class** ObjectModel **extends** RealmObject {  
 @PrimaryKey  
 @SerializedName(**"id"**)  
 @Expose(serialize = **false**)  
 **private int id**;  
 @SerializedName(**"clasificador"**)  
 @Expose  
 **private int clasificador**;  
 @SerializedName(**"categoria"**)  
 @Expose  
 **private int categoria**;  
 @SerializedName(**"fecha\_registro"**)  
 @Expose  
 **private** String **fechaRegistro**;  
 @SerializedName(**"peso"**)  
 @Expose  
 **private double peso**;  
 @SerializedName(**"acerto"**)  
 @Expose  
 **private boolean acerto**;  
  
 **public int** getId() {  
 **return id**;  
 }  
  
 **public void** setId(**int** id) {  
 **this**.**id** = id;  
 }  
  
 **public int** getClasificador() {  
 **return clasificador**;  
 }  
  
 **public void** setClasificador(**int** clasificador) {  
 **this**.**clasificador** = clasificador;  
 }  
  
 **public int** getCategoria() {  
 **return categoria**;  
 }  
  
 **public void** setCategoria(**int** categoria) {  
 **this**.**categoria** = categoria;  
 }  
  
 **public** String getFechaRegistro() {  
 **return fechaRegistro**;  
 }  
  
 **public void** setFechaRegistro(String fechaRegistro) {  
 **this**.**fechaRegistro** = fechaRegistro;  
 }  
  
 **public double** getPeso() {  
 **return peso**;  
 }  
  
 **public void** setPeso(**double** peso) {  
 **this**.**peso** = peso;  
 }  
  
 **public boolean** isAcerto() {  
 **return acerto**;  
 }  
  
 **public void** setAcerto(**boolean** acerto) {  
 **this**.**acerto** = acerto;  
 }  
}

* **Clase User**

Este objeto User contiene los atributos username y password. Este objeto se inicializa con el usuario y contraseña que inicia sesión. Cuenta con sus funciones get y set respectivamente para obtener los atributos del objeto.

**public class** User **extends** RealmObject {  
  
 **private** String **username**;  
  
 **private** String **password**;  
  
  
 **public** User() {  
 }  
  
 **public** User(String username, String password) {  
 **this**.**username** = username;  
 **this**.**password** = password;  
 }  
  
 **public** String getUsername() {  
 **return username**;  
 }  
  
 **public void** setUsername(String username) {  
 **this**.**username** = username;  
 }  
  
 **public** String getPassword() {  
 **return password**;  
 }  
  
 **public void** setPassword(String password) {  
 **this**.**password** = password;  
 }  
  
 **public boolean** save() {  
 **boolean** isSuccess = **true**;  
 Realm realm = Realm.*getDefaultInstance*();  
 **try** {  
 realm.beginTransaction();  
 realm.copyToRealmOrUpdate(**this**);  
 realm.commitTransaction();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 isSuccess = **false**;  
 } **finally** {  
 realm.close();  
 }  
  
 **return** isSuccess;  
 }  
}

* **Clase Token**

Este objeto Token contiene los atributos id y token. Este objeto es el token con que se puede realizar los requerimientos http al servidor de Herokuapp.

**public class** Token **extends** RealmObject {  
 @PrimaryKey  
 **private int id**;  
  
 @SerializedName(**"token"**)  
 @Expose  
 **private** String **token**;  
 **public** Token(){  
 **this**.**id** = **new** Date().hashCode();  
 }  
  
 **public** Token(String token) {  
 **this**.**id** = **new** Date().hashCode();  
 **this**.**token** = token;  
 }  
 **public** String getToken() {  
 **return token**;  
 }  
  
 **public void** setToken(String token) {  
 **this**.**token** = token;  
 }  
  
 **public boolean** save() {  
 **boolean** isSuccess = **true**;  
 Realm realm = Realm.*getDefaultInstance*();  
 **try** {  
 realm.beginTransaction();  
 realm.copyToRealmOrUpdate(**this**);  
 realm.commitTransaction();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 isSuccess = **false**;  
 } **finally** {  
 realm.close();  
 }  
  
 **return** isSuccess;  
 }  
  
}

* **Clase Device**

Este objeto Device contiene los datos del dispositivo con el nivel de batería que tiene el arduino. Cuenta con sus funciones get y set respectivamente para obtener los atributos del objeto.

**public class** Device **extends** RealmObject {  
 @PrimaryKey  
 @SerializedName(**"id"**)  
 @Expose  
 **private int id**;  
 @SerializedName(**"thinxtra"**)  
 @Expose  
 **private** String **thinxtra**;  
 @SerializedName(**"bateria"**)  
 @Expose  
 **private int bateria**;  
  
 **public int** getId() {  
 **return id**;  
 }  
  
 **public void** setId(**int** id) {  
 **this**.**id** = id;  
 }  
  
 **public** String getThinxtra() {  
 **return thinxtra**;  
 }  
  
 **public void** setThinxtra(String thinxtra) {  
 **this**.**thinxtra** = thinxtra;  
 }  
  
 **public int** getBateria() {  
 **return bateria**;  
 }  
  
 **public void** setBateria(**int** bateria) {  
 **this**.**bateria** = bateria;  
 }  
  
}

**Recursos de hardware/software para usuario final de proyecto**

* **Hardware:**
  + Dispositivo móvil Android

|  |  |
| --- | --- |
| Dispositivo | Precio |
| Dispositivo Arduino | $ 10.50 |
| Kit Thinxtra DEV (incluido envío) | $ 143.87 |
| Sensores de peso | $ 7 |
| Conversores de peso HX711 | $ 3.95 |
| Total | $ 165.32 |

* **Software:**
  + Sistema operativo Android v.8 o superior
  + Base de datos Postgre V9.6.1

**Recursos de hardware/software para desarrollo**

* **Hardware:**
  + Dispositivo Arduino
  + Kit Thinktra DEV
  + Sensores de peso
  + Conversores de peso HX711
  + Computadora con 8Gb de ram (mínimo 2Gb) y espacio libre de 4 Gb.
* **Software:**
  + Android Studio
  + Java 8
  + Base de datos Postgre V9.6.1

**Fotografías del prototipo**

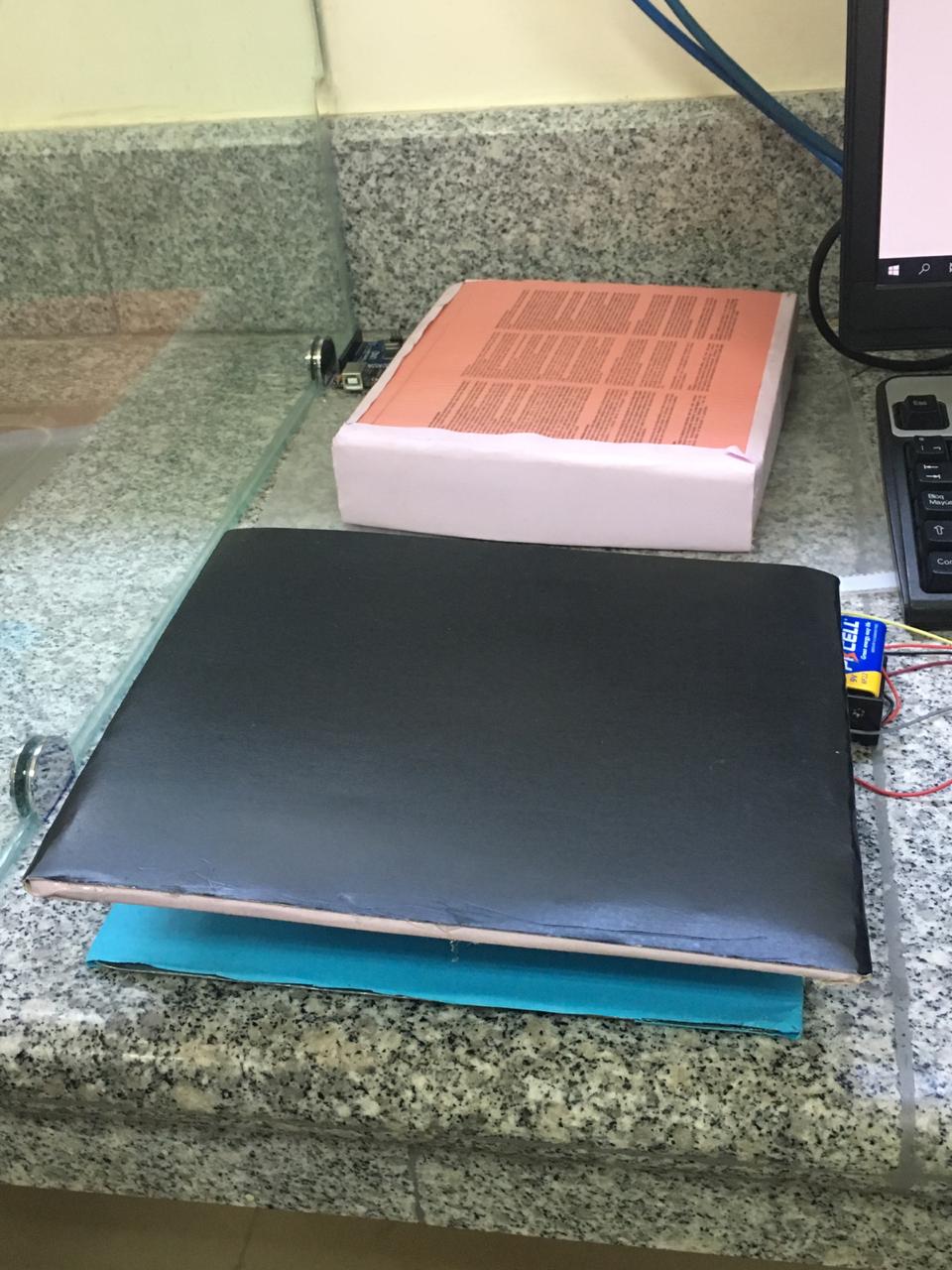


Figura 11 Prototipo final



Figura 12 Prototipo final con el interruptor de encendido



Figura 13 Sensor ubicado en el interior de la balanza