

# BOTBLOQ: Ecosistema integral para el diseño, fabricación y programación de robots DIY

---

Proyecto Financiado por el Centro de Desarrollo Tecnológico Industrial (CDTI)

EXPEDIENTE: IDI-20150289

Cofinanciado por el Fondo Europeo de Desarrollo Regional (FEDER) a través del Programa Operativo Plurirregional de Crecimiento Inteligente 2014-2020

*ACRÓNIMO DEL PROYECTO: BOTBLOQ*



**CDTI**

Centro para el  
Desarrollo  
Tecnológico  
Industrial



**UNIÓN EUROPEA**

**Fondo Europeo de  
Desarrollo Regional (FEDER)**

*Una manera de hacer Europa*

## ENTREGABLE E.5.2.1. Informe descriptivo de la herramienta desarrollada para kits genéricos de robótica

---

Programación nativa  
Frameworks móviles

### RESUMEN DEL DOCUMENTO

El objetivo de este documento es diseñar una aplicación que permita interactuar con kits genéricos de robótica, tanto para recibir datos de los mismos como para enviarle. Para ello se analizarán las distintas alternativas y finalmente se expondrá el resultado obtenido.





# Índice

<b>1.- INTRODUCCIÓN.....</b>	<b>5</b>
<b>2.- ESTADO DEL ARTE.....</b>	<b>6</b>
2.1. FRAMEWORKS MÓVILES .....	7
2.1.1. <i>Ionic v1</i> .....	7
<b>2.- DESARROLLO .....</b>	<b>9</b>
2.1.- <i>PLUGINS Y LIBRERÍAS</i> .....	9
2.1.1. <i>Bluetooth Serial Plugin for PhoneGap</i> .....	9
2.1.2. <i>SpeechRecognition Plugin</i> .....	10
2.1.3. <i>Twit</i> .....	10
2.1.4. <i>Native Audio Plugin</i> .....	11
2.1.5. <i>Flashlight plugin</i> .....	12
2.1.6. <i>Cordova Sensors Plugin</i> .....	12
2.2. ESTRUCTURA Y ARQUITECTURA DEL PROYECTO.....	16
2.3. DISEÑO .....	19
2.5. IMPLEMENTACIÓN .....	22
2.5.1. <i>Móvil recibe texto de la placa</i> .....	23
2.5.2. <i>Móvil emite sonido</i> .....	24
2.5.3. <i>Móvil envía comando a la placa</i> .....	25
2.5.4. <i>Placa interactúa con la linterna del móvil</i> .....	26
2.4.6. <i>Placa publica un tweet</i> .....	29
2.4.7. <i>Placa recibe datos de los sensores</i> .....	32
<b>3. MANUAL DE USUARIO .....</b>	<b>34</b>

## 1.- Introducción

En este documento se tratará el desarrollo de la herramienta destinada a la interacción con kits genéricos de robótica.

Inicialmente, se expondrán las posibilidades que existen para el desarrollo de aplicaciones móviles y se elegirá una teniendo en cuenta los requisitos.

Una vez elegido el *framework* más adecuado para la aplicación, se hablará de su desarrollo (arquitectura de la aplicación, diseño, plugins utilizados, control de la placa e implementación).

Finalmente, se presentará una guía de usuario en el que se resume el funcionamiento básico de la aplicación.

## 2.- Estado del arte

Debido al gran auge de los smartphones. en la actualidad existen numerosas tecnologías que nos permiten programar aplicaciones móviles. Se considera imprescindible realizar un análisis de todas las tecnologías disponibles dados los requisitos presentados anteriormente.

De modo general, para programar aplicaciones existen dos enfoques:

- **Programación nativa:** se trata de programar las aplicaciones específicamente para el S.O. sobre el que van a ser ejecutadas, en este caso Android o iOS.
- **Uso de frameworks móviles:** se trata de programar las aplicaciones móviles híbridas en un lenguaje de programación intermedio (HTML, Javascript, AngularJS, etc.) y que sea el framework el encargado de transformar ese lenguaje a nativo del S.O. (compilar la aplicación).

Estos enfoques poseen ventajas y desventajas, que se listan a continuación:

	Ventajas	Desventajas
<b>Programación nativa</b>	<ul style="list-style-type: none"><li>• Programación más específica de elementos hardware del dispositivo.</li><li>• Mayores opciones en la programación.</li></ul>	<ul style="list-style-type: none"><li>• Programación en exclusiva para ese S.O. (controlar versiones de Android).</li><li>• Curva de aprendizaje más lenta.</li><li>• No extensible a otro S.O. (sería desarrollo de 0 con nueva curva de aprendizaje).</li></ul>
<b>Frameworks móviles</b>	<ul style="list-style-type: none"><li>• Curva de aprendizaje rápida (aprovechar lenguajes de programación conocidos).</li><li>• 1 desarrollo, múltiples plataformas.</li><li>• Mantenibilidad (si se hace un cambio, sólo se hace en un código).</li></ul>	<ul style="list-style-type: none"><li>• Programación más limitada (uso de frameworks que no permiten acceder a todo el conjunto del dispositivo).</li></ul>

Analizando las ventajas y desventajas de cada una de las opciones, finalmente se decidió la utilización de *frameworks* móviles, ya que:

1. El equipo de desarrollo estaba familiarizado con tecnologías web y la curva de aprendizaje era muy rápida. Esto implicaba un desarrollo mucho más rápido.
2. Aunque el objetivo inicial era programar aplicaciones Android, dejaba la puerta abierta para la integración con iOS.
3. Los requisitos que imponía la aplicación no implicaba un uso avanzado de la parte hardware, por lo que con las librerías ofrecidas en los *frameworks* era suficiente.

## 2.1. Frameworks móviles

De cara a comenzar la programación, se realizó un análisis de los frameworks móviles disponibles y más utilizados. Para ello, se consultó información en el siguiente enlace: [www.hongkiat.com/blog/mobile-frameworks](http://www.hongkiat.com/blog/mobile-frameworks).

Finalmente, y debido a que la aplicación de Botbloq está desarrollada en AngularJS (<https://angularjs.org/>) empaquetada con Electron, se eligió desarrollar la aplicación mediante el *framework* **Ionic v1** (<https://ionicframework.com/>), ya que permite desarrollar aplicaciones móviles en AngularJS como se detallará a continuación.

Se descartó el uso de la versión 2 del *framework* debido a dos razones fundamentalmente:

- Ionic versión 2 está escrito en AngularJS 2, lenguaje que no es conocido para el equipo de desarrollo y que supone un cambio drástico con respecto a la versión anterior, por lo que implicaría invertir horas en aprendizaje y se retrasaría el desarrollo.
- Ionic versión 2 ha sido lanzado hace relativamente poco, por lo que existe menos comunidad y resolver cualquier problema supondría invertir más tiempo.

### 2.1.1. Ionic v1

Ionic es un *framework* de desarrollo de aplicaciones móviles destinado a construir aplicaciones móviles híbridas. Las aplicaciones híbridas son esencialmente pequeñas páginas web que se ejecutan en un navegador dentro de la aplicación y tienen acceso a la capa nativa de la plataforma.

Ionic se orquesta como un *framework front-end* que se encarga de la apariencia y de las interacciones con la interfaz. Sería comparable a un 'Bootstrap para nativo', pero tiene la posibilidad de interactuar con un amplio conjunto de componentes nativos del móvil.

Debido a que Ionic es un *framework* HTML5, necesita un *wrapper* nativo como Cordova o PhoneGap para que funcione como una aplicación nativa. En nuestro caso, elegimos **Apache Cordova** (<https://cordova.apache.org/>).

**Apache Cordova** es un *framework open-source* para desarrollo móvil. Permite utilizar tecnologías web estándar - HTML5, CSS3, Javascript para desarrollo multi-plataforma. Las aplicaciones se ejecutan en *wrappers* enfocados a cada plataforma, y acceden a sensores, datos, estado de la red, etc., mediante llamadas a una API conforme con los estándares.

En nuestro caso, utilizaremos los *plugins* que nos ofrece Apache Cordova. Estos proporcionan una interfaz de comunicación entre Cordova y los componentes nativos así como un enlace a las APIs estándares del dispositivo. En resumen, permiten invocar código nativo desde Javascript.

Apache Cordova mantiene una serie de *plugins* llamados **Core Plugins** (<https://cordova.apache.org/docs/en/latest/guide/support/index.html#core-plugin-apis>), que permiten acceder a componentes del dispositivo como batería, cámara, sensores, etc.

Además de estos *plugins*, existen *plugins* de terceros que proporcionan otros enlaces a funcionalidades del dispositivo, aunque estos no aseguran que estas funcionalidades estén disponibles en todas las plataformas.

En la parte de desarrollo se detallarán los *plugins* utilizados en la aplicación.



## 2.- Desarrollo

Como se comentó anteriormente, para el desarrollo de la aplicación móvil se utilizó el *framework* Ionic 1, basado en AngularJS. El proyecto es *open source* y está disponible para su descarga en <https://github.com/bq/bitbloq-communication-app>. Finalmente se decidió darle el nombre de Bitbloq Connect.

Para la implementación de las funcionalidades se han utilizado *plugins* de terceros, los cuáles se describirán a continuación.

### 2.1.- Plugins y librerías

#### 2.1.1. Bluetooth Serial Plugin for PhoneGap

Este *plugin* permite la comunicación serie mediante Bluetooth. Fue escrito para comunicar Android o iOS y una placa Arduino.

En el caso de Android se usa el Bluetooth clásico, no siendo el mismo caso en iOS, que utiliza Bluetooth de Baja Energía.

Este *plugin* está disponible en <https://github.com/don/BluetoothSerial>.

Ofrece múltiples métodos, de los cuáles destacaremos los utilizados en la aplicación para la comunicación Bluetooth:

- `bluetoothSerial.list(success, failure);`  
Lista los dispositivos de Bluetooth vinculados.
- `bluetoothSerial.connect(macAddress_or_uuid, connectSuccess, connectFailure)`  
Permite conectar a un dispositivo Bluetooth.
- `bluetoothSerial.disconnect([success], [failure]);`  
Permite desconectar la conexión actual de Bluetooth.
- `bluetoothSerial.write(data, success, failure);`  
Escribe datos en el puerto serie (este método se utilizará para enviar información a la placa).
- `bluetoothSerial.read(success, failure);`  
Lee datos del puerto serie (este método se utilizará para leer los comandos enviados por la placa).
- `bluetoothSerial.subscribe('\n', success, failure);`  
Se suscribe al puerto serie para ser notificado cuando se reciben datos. Mediante '\n' especificamos el delimitador de los datos. La función

'success' es llamada con los datos tan pronto como se lee el delimitador.

- `bluetoothSerial.unsubscribe();`  
Se desuscribe de cualquier suscripción.
- `bluetoothSerial.isConnected(success, failure);`  
Devuelve el estado de la conexión. Si el dispositivo está conectado por Bluetooth llama a la función `success`. En caso contrario, llama a `failure`.
- `bluetoothSerial.isEnabled(success, failure);`  
Devuelve si el Bluetooth está habilitado. Si el Bluetooth está habilitado llama a la función `success`. En caso contrario, llama a `failure`.
- `bluetoothSerial.enable(success, failure);`  
Habilita el Bluetooth en el dispositivo. Esta función provoca que aparezca una ventana emergente preguntando al usuario si quiere activar el Bluetooth.

### 2.1.2. SpeechRecognition Plugin

Este *plugin* se utiliza para convertir voz a texto, de modo que sea posible comunicarse con la aplicación mediante voz, ésta lo transforme a texto y lo envíe a la placa mediante Bluetooth.

Este *plugin* está disponible en

<https://github.com/macdonst/SpeechRecognitionPlugin>.

### 2.1.3. Twit

El objetivo del uso de esta librería es poder publicar mensajes en Twitter.

La principal dificultad que se encontró a la hora de la implementación de esta funcionalidad fue que la mayoría de los *plugins* no permitían la configuración de los credenciales sin que el usuario tuviera que introducir su usuario y contraseña de Twitter en el móvil. Esto era algo incómodo ya que no permitía la completa automatización del proceso.

Finalmente, y después de no encontrar *plugins* de Cordova que permitieran la implementación de esta funcionalidad, se utilizó el cliente de Twitter para Node llamado **Twit**.

Este cliente permite enviar mensajes a Twitter, previa configuración de sus credenciales (`consumer_key`, `consumer_secret`, `access_token`, `access_token_secret`), que se pueden obtener creando una nueva aplicación a través de <https://apps.twitter.com/>.

Debido a que la aplicación está desarrollada con Ionic, que utiliza AngularJS por debajo, fue necesario utilizar Browserify (<http://browserify.org/>) sobre Twit. Browserify nos permite utilizar módulos de NodeJS en el navegador. Para ello, se configuraron los métodos necesarios del cliente y luego se ejecutó el comando `browserify` pasándole el fichero JavaScript escrito en Node.

Una vez obtenido el fichero, se podrá acceder a él utilizando el elemento *window* del navegador.

Se utilizaron los siguientes métodos del cliente:

- `T.post('statuses/update', { status: 'hello world!' }, function(err, data, response) { console.log(data)})`  
Este método nos permiten publicar mensajes en un Twitter configurado previamente (mediante la variable T).

T es un nuevo objeto Twit que recibe los credenciales obtenidos al crear una aplicación.

```
var Twit = require('twit')
var T = new Twit({
  consumer_key:      '...',
  consumer_secret:    '...',
  access_token:       '...',
  access_token_secret: '...',
  timeout_ms:         60*1000,
})
```

#### 2.1.4. Native Audio Plugin

Este *plugin*, disponible en <https://github.com/floatinghotpot/cordova-plugin-nativeaudio>, se utiliza para reproducir archivos de audio en el móvil.

Los métodos más importantes de cara al desarrollo de la aplicación son:

- `preloadComplex: function ( id, assetPath, volume, voices, delay, successCallback, errorCallback)`  
Carga un fichero de audio en memoria con identificador `id` y ruta `assetPath`. Puede ser parado o puesto en bucle.
- `play: function (id, successCallback, errorCallback, completeCallback)`  
Reproduce un fichero de audio de identificador `id`.

- `stop: function (id, successCallback, errorCallback)`  
Para un fichero de audio de identificador `id`. Sólo funciona si el fichero se ha cargado utilizando la función `preloadComplex`.
- `unload: function (id, successCallback, errorCallback)`  
Elimina un fichero cargado de memoria con identificador `id`.

De modo general, necesitamos cargar un audio antes de poder reproducirlo o pararlo, pero una vez cargado se puede reproducir o parar sin tener que cargarlo de nuevo.

#### 2.1.5. Flashlight plugin

Este *plugin* permite interactuar con la linterna del dispositivo.

Posee métodos para comprobar si la linterna está disponible  
(`window.plugins.flashlight.available(availableFunction)`),  
para encenderla  
(`window.plugins.flashlight.switchon(successFunction, errorCallback, {intensity:0.3})`), apagarla  
(`window.plugins.flashlight.switchoff(successFunction, errorCallback)`) y hacerla parpadear  
(`window.plugins.flashlight.switchon(successFunction, errorCallback, {intensity:0.3})`).

#### 2.1.6. Cordova Sensors Plugin

Este *plugin*, disponible en <https://github.com/fabiorogerosj/cordova-plugin-sensors>, se utiliza para obtener los datos de los distintos sensores que componen el dispositivo.

La mayoría de dispositivos Android están compuestos por sensores que miden movimiento, orientación y varias condiciones ambientales. Estos sensores son capaces de proporcionar datos con gran precisión y exactitud, y son útiles de cara a monitorizar el movimiento tridimensional o la posición del dispositivo, así como para obtener los cambios ambientales cerca del mismo.

Los dispositivos Android poseen tres categorías de sensores:

- **Sensores de movimiento**

Estos sensores miden la fuerza de la aceleración y las fuerzas rotacionales a lo largo de los tres ejes. Esta categoría incluye acelerómetros, sensores de gravedad, giroscopios y sensores de vectores rotacionales.

- **Sensores ambientales**

Estos sensores miden varios parámetros ambientales, como la temperatura del aire y la presión, la iluminación y la humedad. Esta categoría incluye barómetros, fotómetros y termómetros.

- **Sensores de posición**

Estos sensores miden la posición física del dispositivo. Esta categoría incluye sensores de orientación y magnetómetros.

Android permite acceder a muchos tipos de sensores. Algunos de ellos son sensores **basados en hardware** y otros **basados en software**. Los sensores basados en hardware son componentes físicos del dispositivo. Los datos de los mismos se obtienen directamente midiendo propiedades ambientales específicas, como la aceleración, la fuerza del campo geomagnético o el cambio angular. Los sensores basados en software no son elementos físicos, aunque imitan a los sensores basados en hardware. Los sensores basados en software derivan sus datos de uno o más sensores basados en hardware.

En la siguiente tabla se observan los sensores, tanto hardware como software, que pueden formar parte de un dispositivo Android.

Sensor	Tipo	Descripción
<b>Acelerómetro</b>	Hardware	Mide la fuerza de la aceleración en $m/s^2$ que es aplicada al dispositivo en los tres ejes físicos (x,y,z), incluyendo la fuerza de la gravedad.
<b>Temperatura Ambiente</b>	Hardware	Mide la temperatura ambiente en grados Celsius ( $^{\circ}C$ ).
<b>Gravedad</b>	Software o Hardware	Mide la fuerza de la gravedad en $m/s^2$ que es aplicada al dispositivo en los tres ejes físicos (x,y,z).
<b>Giroscopio</b>	Hardware	Mide el ratio de rotación del dispositivo en rad/s alrededor de cada uno de los tres ejes físicos (x,y,z).
<b>Luz</b>	Hardware	Mide el nivel de luz ambiente (iluminación) en lx.
<b>Aceleración lineal</b>	Software o Hardware	Mide la fuerza de aceleración en $m/s^2$ que es aplicada a un dispositivo en los tres ejes físicos (x,y,z), excluyendo la fuerza de la gravedad.
<b>Campo magnético</b>	Hardware	Mide el campo geomagnético ambiente para los tres ejes físicos (x,y,z) en $\mu T$ .
<b>Orientación</b>	Software	Mide los grados de rotación que un dispositivo realiza alrededor de los tres ejes físicos (x,y,z).
<b>Presión</b>	Hardware	Mide la presión ambiental del aire en hPa o mbar.
<b>Proximidad</b>	Hardware	Mide la proximidad de un objeto en cm relativo a la pantalla de un dispositivo. Muchos dispositivos devuelven 0 si está muy próximo y 5 si no lo está.
<b>Humedad relativa</b>	Hardware	Mide la humedad ambiente relativa en porcentaje (%).
<b>Vector de rotación</b>	Hardware o Software	Mide la orientación de un dispositivo proporcionando los tres elementos del vector de rotación del dispositivo.
<b>Temperatura</b>	Hardware	Mide la temperatura del dispositivo en grados Celsius ( $^{\circ}C$ ).

El *plugin* permite acceder a los siguientes sensores:

- Proximidad (PROXIMITY).
- Acelerómetro (ACCELEROMETER).
- Gravedad (GRAVITY).
- Giroscopio (GYROSCOPE).
- Giroscopio sin calibrar (GYROSCOPE\_UNCALIBRATED).
- Aceleración lineal (LINEAR\_ACCELERATION).
- Vector de rotación (ROTATION\_VECTOR).
- Contador de pasos (STEP\_COUNTER).
- Vector de rotación de juego (GAME\_ROTATION\_VECTOR).
- Vector de rotación geomagnético (GEOMAGNETIC\_ROTATION\_VECTOR).
- Campo magnético (MAGNETIC\_FIELD).
- Campo magnético sin calibrar (MAGNETIC\_FIELD\_UNCALIBRATED).
- Orientación (ORIENTATION).
- Temperatura ambiente (AMBIENT\_TEMPERATURE).
- Luz ambiente (LIGHT).
- Presión (PRESSURE).
- Humedad relativa (RELATIVE\_HUMIDITY).
- Temperatura (TEMPERATURE).

De cara a realizar las pruebas, se utilizó un Aquaris E5 (<https://www.bq.com/es/aquaris-e5s>). Se realizaron pruebas con este móvil para comprobar qué sensores estaban disponibles y se obtuvieron los resultados que se recogen en la siguiente tabla.

Sensor	Accesible
PROXIMITY	SÍ
ACCELEROMETER	SÍ
GRAVITY	SÍ
GYROSCOPE	SÍ
GYROSCOPE_UNCALIBRATED	NO
LINEAR_ACCELERATION	SÍ
ROTATION_VECTOR	SÍ
STEP_COUNTER	NO
GAME_ROTATION_VECTOR	NO
GEOMAGNETIC_ROTATION_VECTOR	NO
MAGNETIC_FIELD	SÍ
MAGNETIC_FIELD_UNCALIBRATED	NO
ORIENTATION	SÍ
AMBIENT_TEMPERATURE	NO
LIGHT	SÍ
PRESSURE	NO
RELATIVE_HUMIDITY	NO
TEMPERATURE	NO

Teniendo en cuenta la anterior tabla, se implementó en la aplicación la lectura de los sensores que eran accesibles en el dispositivo seleccionado para las pruebas. El *plugin* permite acceder a los sensores anteriores mediante este código:

```
function onSuccess(values) {
  $scope.state = values[0];
};
document.addEventListener("deviceready", function () {
  sensors.enableSensor("TIPO DE SENSOR");
  $interval(function() {
    sensors.getState(onSuccess);
  }, 100);
}, false);
```

Como se puede observar, inicialmente hay que habilitar el sensor (sustituyendo TIPO DE SENSOR por el sensor correspondiente según la tabla anterior). Tras hacerlo, se pueden obtener los valores que proporciona. En el caso de que el sensor proporcione medidas en cada uno de los ejes, la respuesta será un *array* de 3 elementos, cada uno de los cuales corresponderá a uno de los ejes (x,y,z). En el caso de ser valores únicos (luminosidad, proximidad), el resultado se devolverá en el elemento 0 del *array*.

## 2.2. Estructura y arquitectura del proyecto

El proyecto está organizado en las siguientes carpetas:

- **bower\_components**  
En esta carpeta se encontrarán las librerías descargadas que se indican en el bower.json.
- **hooks**  
Los hooks de Cordova representan scripts especiales que pueden ser añadidos por cada aplicación. Permiten realizar actividades especiales cuando se ejecutan comandos de Cordova.  
A su vez, esta carpeta se subdivide en:
  - after\_platform\_add
  - after\_plugin\_add
  - after\_plugin\_rm
  - after\_prepare
  - before\_platform\_add  
En el caso del desarrollo de esta aplicación, no se han creado nuevos hooks. Por tanto, en la carpeta están los creados por defecto.



- **node\_modules**

En esta carpeta se encontrarán las librerías descargadas que se indican en el `package.json`.

- **platforms**

Esta carpeta se genera cuando se ejecuta el comando `ionic platform add android/ios`. Contiene un JSON con los datos de las plataformas sobre las que se empaquetará la aplicación (Android, iOS...). Una vez hemos creado la aplicación en AngularJS con Ionic, si ejecutamos `ionic build android/ios`, se crea una carpeta referente al sistema operativo correspondiente en el que se encuentran los archivos de la aplicación en lenguaje nativo de la plataforma. En esta carpeta se encuentra también el archivo de la aplicación empaquetada.

- **plugins**

En esta carpeta se encuentran los *plugins* de Cordova instalados.

- **res**

En esta carpeta se encuentran imágenes redimensionadas a los distintos tamaños necesarios.

- **resources**

En esta carpeta incluiremos el icono de la aplicación y la pantalla de carga de la aplicación y será Ionic (mediante el comando `ionic resources`) quien las adapte a todos los tamaños para que la aplicación pueda ser ejecutada en todos los dispositivos independiente del tamaño de la pantalla.

- **scss**

Hojas de estilos de los HTML que componen la aplicación.

- **www**

En esta carpeta se encuentran los archivos de la aplicación en AngularJS. En ella se encuentra el archivo `index.html`, en el que se importan todos los archivos necesarios para hacer funcionar la aplicación (JavaScripts, hojas de estilo) y se embebe la vista principal.

- **audio**  
En esta carpeta se encuentran los archivos mp3 de las pistas de audio que reproduce la aplicación cuando la placa le envía la orden.
- **css**  
Hojas de estilo de la aplicación.
- **img**  
Imágenes de la aplicación.
- **js**  
Módulos Javascript de la aplicación. Contiene el fichero app.js, dónde se definen las dependencias de la aplicación y las rutas de la aplicación con su *controller* asociado. Además, contiene las traducciones en castellano e inglés de los textos de la aplicación.

Contiene también el fichero ng-cordova.min.js, que contiene las librerías de Cordova minimizadas.

Además de los ficheros anteriores, la carpeta js contiene las siguientes subcarpetas:

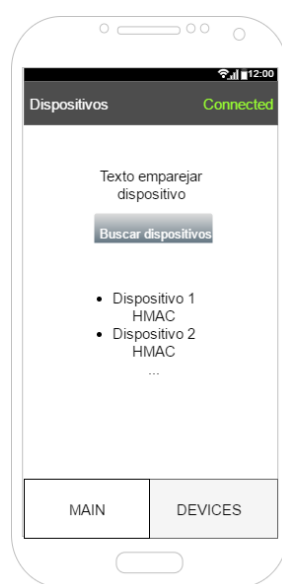
- **controllers**  
Controladores de la aplicación. En esta carpeta se encuentra el controlador del bluetooth (bluetooth.js), el de la pestaña de dispositivos – que permite elegir un dispositivo Bluetooth y conectarse a él (devices.js), el de la pantalla principal de la aplicación (main.js) y el encargado de la barra superior (menu.js).
- **services**  
Servicios de la aplicación. Incluye en servicio del Bluetooth (bluetooth.js) y un fichero de funciones comunes a varios controladores (common.js).
- **lib**  
En esta carpeta se encuentran librerías externas, de angular y de ionic y además una librería propia, tweet-ionic.js. Esta librería fue generada con browserify a partir de la librería Twit escrita en NodeJS.
- **templates**  
En esta carpeta se encuentran las vistas de la aplicación: bluetooth.html, browse.html y devices.html que controlar la conexión Bluetooth con el dispositivo, main.html que es la vista principal de la aplicación y menu.html que controla la vista donde se indexa tanto la vista principal como la de los dispositivos.

### 2.3. Diseño

El paradigma que se ha seguido a la hora de diseñar la aplicación es el de SPA (*Single Page Application*) de las páginas web tradicionales. En este caso, se ha querido que toda la aplicación esté en un 'panel de control' en el cual el usuario pueda observar qué órdenes le está enviando la placa de una sola vista.

A pesar de esto, y debido a que la aplicación tiene que comunicarse por Bluetooth con la placa, se hace imprescindible tener una pantalla en la cual aparezcan las placas y permita seleccionar a cuál se quiere conectar.

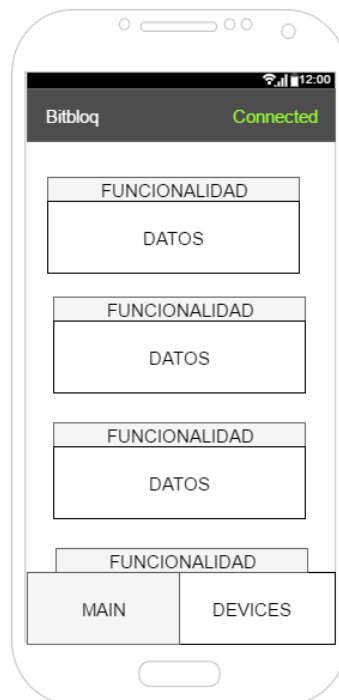
Por ello, la aplicación se ha diseñado según el *Wireframe* que se observa a continuación.



Como se puede observar, y debido a que el menú sólo tiene dos opciones (Principal y Dispositivos), se ha elegido un menú horizontal basado en pestañas. Además, de cara a conocer siempre el estado de la conexión, se ha colocado una barra horizontal en la parte posterior de la aplicación donde se informa del estado de la conexión Bluetooth (Conectado o Conexión perdida). En esta barra también se podrá reconectar en el caso de perder la conexión. La palabra Conectado irá en color verde, ya que este color significa que todo ha ido bien. En cambio, la palabra Conexión perdida irá en rojo, ya que implica un error o un funcionamiento que no es el esperado.

En la pantalla de Dispositivos aparece un botón que permite listar los dispositivos enlazados al móvil, y al pulsar en uno de ellos el teléfono se conecte a él.

La pantalla principal de la aplicación seguirá el diseño que se expone a continuación:



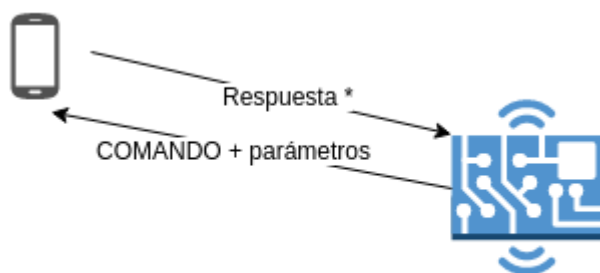
En la pantalla principal se agruparán las funcionalidades en cajas, de modo que cuando se reciba el comando asociado a una funcionalidad, esa caja se coloreará de verde para llamar la atención del usuario.

La aplicación actualmente está disponible en castellano y en inglés.

## 2.4. Control de la placa

En este apartado se explicará el modo en el que la placa se comunica con la placa y viceversa.

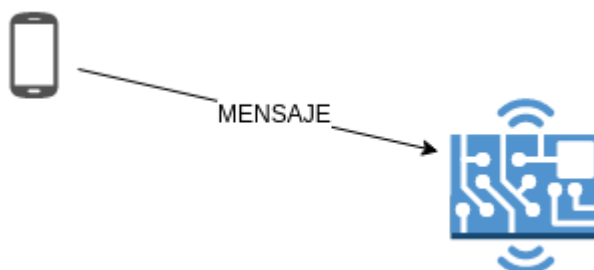
En la siguiente imagen se resumen el funcionamiento general de la aplicación.



\*La respuesta es opcional

Una vez se conecta el dispositivo móvil a la placa, ésta empieza a enviarle comandos al móvil incluyendo los datos necesarios. En el momento en el que los recibe, el móvil realiza la acción asociada al comando (por ejemplo, si la placa le envía "playSound-bongo", el móvil reproducirá el sonido bongo). Existen casos en el que el dispositivo enviará una respuesta a la placa (como por ejemplo, en el caso en el que la placa le pida leer el valor de un sensor), por lo que en ese caso sí habrá respuesta.

Existe una excepción a este funcionamiento general, que es el caso de "Enviar comandos a la placa", que se explicará a continuación en el apartado 2.5. de implementación. En este caso, el diagrama es el siguiente:



El móvil le envía un mensaje a la placa, la cual se encontrará esperando a recibir mensajes por Bluetooth. En este caso, la placa no responde al móvil con ningún mensaje (simplemente recoge el dato y realiza las acciones que se programen en la misma). Esta funcionalidad resulta muy útil para controlar la placa desde el dispositivo móvil (por ejemplo, controlar el encendido y apagado de un led por voz).

## 2.5. Implementación

En este apartado se detallará la implementación de la aplicación. Toda la lógica de la conexión por Bluetooth se encuentra en el servicio `bluetooth.js`, disponible en `www/js/services/bluetooth.js`.

Una vez el usuario pulsa el dispositivo al que se quiere conectar, se ejecuta la función `connectToDevice(device)`, pasándole como parámetro el dispositivo al que se quiere conectar. Esta función está disponible en el controlador de la vista `devices.html`, cuyo nombre es `DevicesCtrl` y está disponible en `www/js/controllers/devices.js`.

La implementación de esta función se especifica a continuación:

```
$scope.connectToDevice = function(device) {
  bluetooth.connect(device.address).then(
    function(response) {
      common.selectedDevice = device;
      console.log('full conected');
    }
  );
};
```

En el código anterior, se llama a la función `connect` del servicio `bluetooth`. Ésta utiliza el *plugin* de Bluetooth para conectar con la placa y se suscribe al mismo a la espera de datos, siendo el final de los mismos `"/n"`.

```
exports.connect = function(address) {
  var command;
  var credentials;
  var q = $q.defer();
  $ionicLoading.show({
    template: 'Conectando...'
  });
  $window.bluetoothSerial.connect(address, function() {
    common.isConnected = true;
    $window.bluetoothSerial.subscribe('\n', function(data) {
      switch (data.split(/-(.+)?/)[0]) {
        case 'playSound':
          $rootScope.$emit('bluetoothSerial:playSound',
data.split(/-(.+)?/)[1]);
          break;
        case 'write':
          $rootScope.$emit('bluetoothSerial:write',
data.split(/-(.+)?/)[1]);
          break;
```

```
    }  
    [...]  
    }, function(error) {  
        alert('Error al intentar recibir mensajes  
por Bluetooth');  
    });  
    $ionicLoading.hide();  
    q.resolve();  
    }, function(error) {  
        common.isConnected = false;  
        alert('No hemos podido conectar al  
dispositivo, verifica que está emparejado y  
encendido. ' + JSON.stringify(error));  
        $ionicLoading.hide();  
        q.reject(error);  
    });  
    return q.promise;  
};
```

Los mensajes de la placa vendrán en el formato COMANDO-DATOS. Una vez llega el mensaje, se divide por el carácter '-': el primer elemento será el comando y el segundo serán los datos.

Dependiendo del evento, se envía un evento en `rootScope` con la forma `'bluetoothSerial:COMANDO'`.

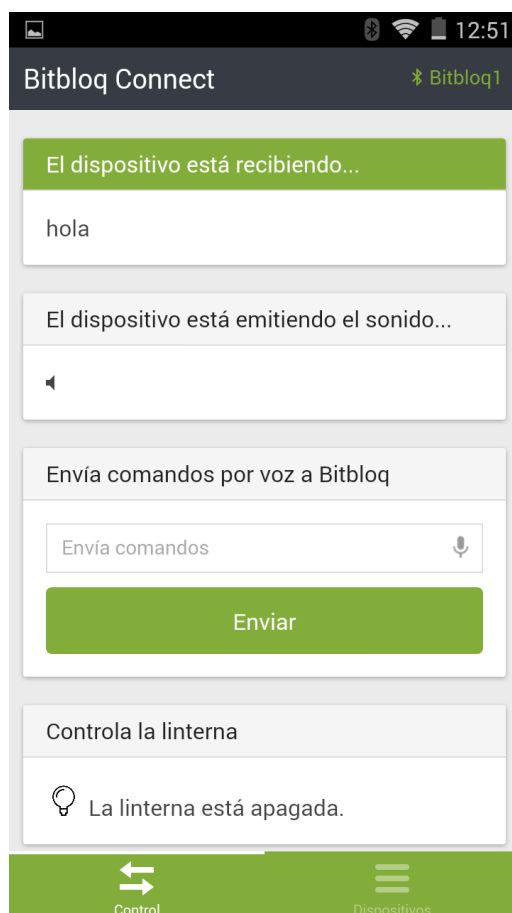
Este evento lo recibirá el controlador `MainCtrl`, disponible en `js/controllers/main.js`. En este controlador existirá un bloque de código como el siguiente por cada uno de los eventos.

```
$rootScope.$on('bluetoothSerial:turnoffFlashlight',  
function(evt) {  
    turnoffFlashlight();  
});
```

Una vez ha recibido el evento, ejecuta la función que se encuentra en su interior. Esta función depende del comando que se mande. A continuación, describiremos cada una de las funcionalidades de la aplicación.

### 2.5.1. Móvil recibe texto de la placa

En esta funcionalidad, la placa le envía un texto a la aplicación y ésta lo muestra por pantalla. El evento que escuchará será `bluetoothSerial:write` y actualizará el valor de la variable para mostrarlo en la pantalla. Por ejemplo, si desde la placa le enviamos el texto 'hola', el resultado será el mostrado a continuación.



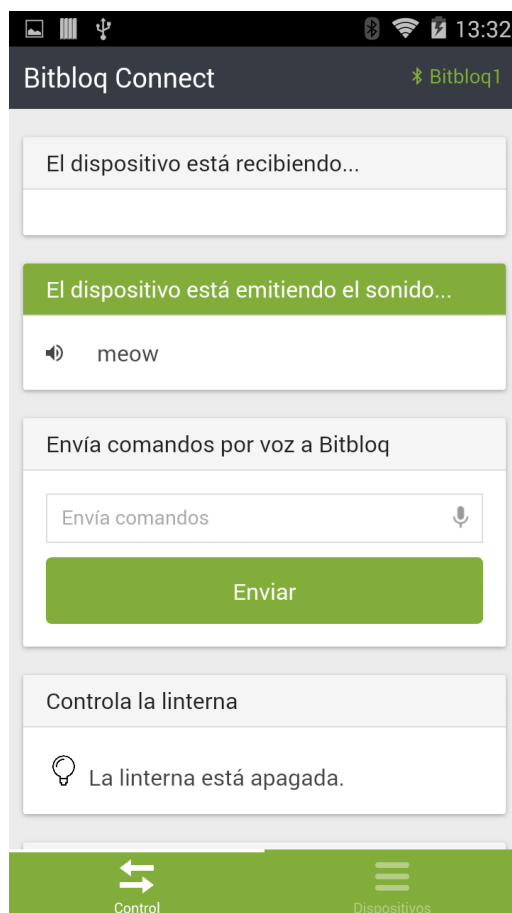
### 2.5.2. Móvil emite sonido

En esta funcionalidad, la placa le envía al móvil la orden de reproducir un sonido en particular y éste lo reproduce. Específicamente, le envía el nombre del sonido y es la aplicación la que busca entre los archivos de la carpeta audio el fichero con el nombre correspondiente. El evento que escuchará será `bluetoothSerial:playSound` y recibirá como datos el nombre del sonido a reproducir.

Se ejecutará la función `playSound`. Esta función reproducirá el sonido que se le pase como parámetro. En el caso de que ya haya un sonido reproduciendo, lo parará y ejecutará el nuevo. Como antes de reproducir el sonido es necesario cargarlo (con la función `preloadComplex`), se mantendrá en un *array* los sonidos que han sido ya cargados de cara a no volverlos a cargar en una nueva iteración.

De cara a conocer el sonido que se está reproduciendo, se mostrará su nombre en la pantalla principal de la aplicación, así como un icono de un altavoz emitiendo.

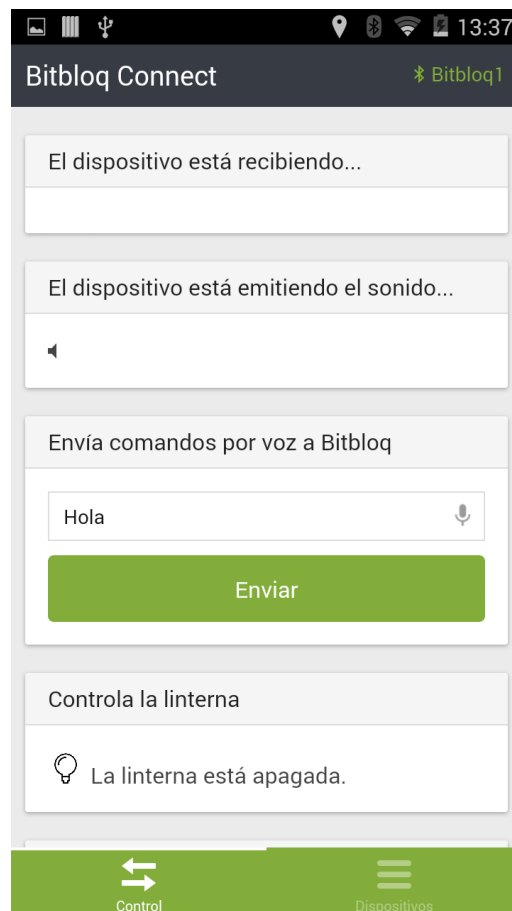




### 2.5.3. Móvil envía comando a la placa

En esta funcionalidad, el móvil le envía a la placa un comando (una cadena de texto, en general). Este comando puede ser introducido por texto o por voz (que posteriormente se convertirá en texto con el plugin correspondiente). Si el usuario desea introducirlo a voz, deberá pulsar el micrófono, decir una palabra y pulsar el botón Enviar.

De cara a conocer si es la palabra correcta, ésta aparecerá en la barra de texto. Para la conversión de voz a texto, se ha utilizado la librería SpeechRecognition.



En este caso no se sigue la lógica explicada en el principio de este apartado, ya que no es necesario que la placa le envíe el comando (es el móvil quién debe enviar los datos a la placa y ser ésta quién los escuche).

Por tanto, en este caso, se utilizará la función `write` del servicio `bluetooth`.

#### 2.5.4. Placa interactúa con la linterna del móvil

En esta funcionalidad, la placa interactúa con la linterna del móvil. Existen dos tipos de interacciones:

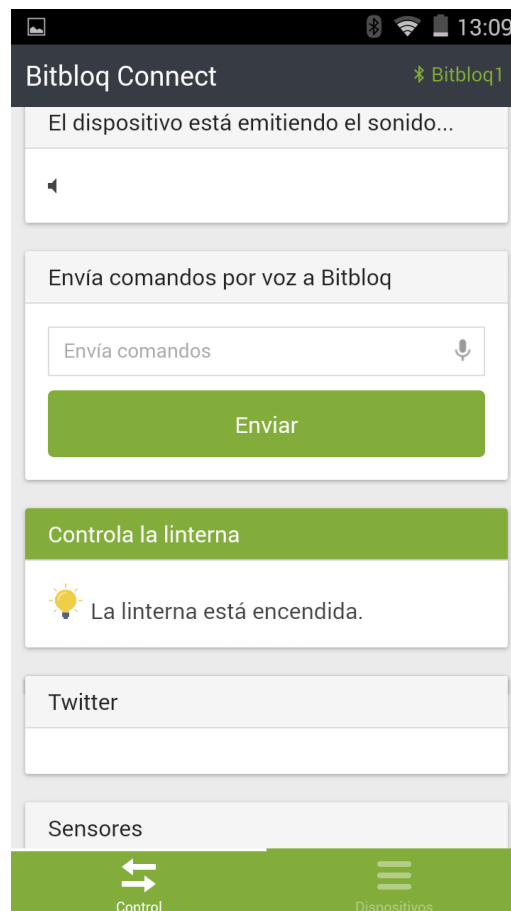
##### 2.5.4.1. Encender la linterna del móvil

Esta funcionalidad permite encender la linterna (el flash de la cámara) del dispositivo. El evento que escuchará será `bluetoothSerial:turnonFlashlight`.

Se ejecutará la función `turnonFlashlight`. Debido a que la linterna no puede conmutar de estado a un tiempo menor de 500 ms, se define la variable `flashTimer`, que no permite cambiar el estado de la linterna hasta que no pasen 500ms, independientemente de los comandos que le lleguen de la placa. Esto fue necesario debido a que si no se configuraba así la aplicación dejaba de funcionar.

Se utilizará el plugin `flashlight` que se comentó en el apartado anterior para interactuar con la linterna.

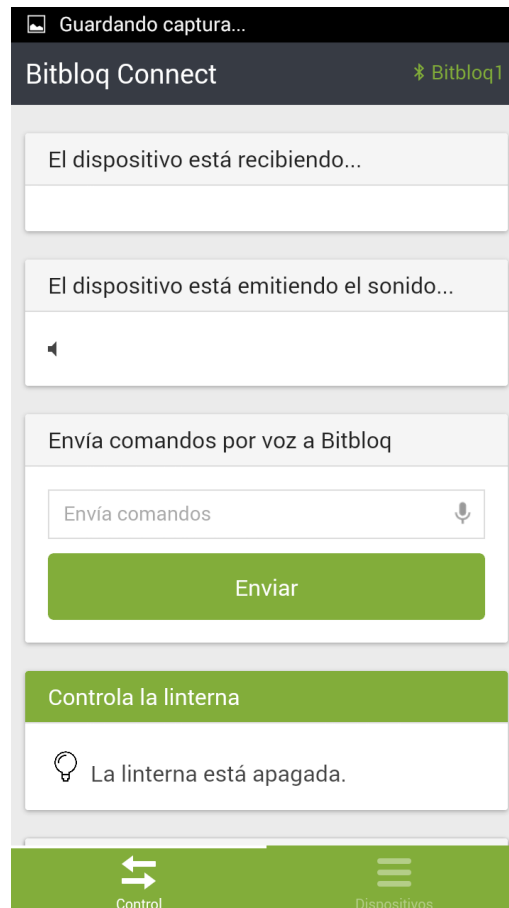
De cara a conocer el estado de la linterna, éste se mostrará en pantalla en su apartado correspondiente unido a un icono de una bombilla que encendida o apagada.



#### 2.5.4.2. Apagar la linterna del móvil

Esta función es similar al anterior, sólo que el evento que escuchará será `bluetoothSerial:turnoffFlashlight`. Su función es apagar la linterna.

En este caso, la función que se ejecuta es `turnoffFlashlight`, que hace la misma comprobación con respecto a la variable `flashTimer` y apaga la linterna.



#### 2.4.6. Placa publica un tweet

Esta funcionalidad permite publicar un mensaje en Twitter si previamente se han configurado sus credenciales.

Para esta funcionalidad utilizaremos la librería creada `tweet-ionic.js`, que posee dos funciones: `configTwitter`, que permite construir un objeto con las credenciales de una cuenta de Twitter, y `sendTwitter`, que recibe como parámetros el mensaje a obtener y ese objeto de credenciales previamente configurado con `configTwitter`.

Se subdivide en dos funcionalidades:

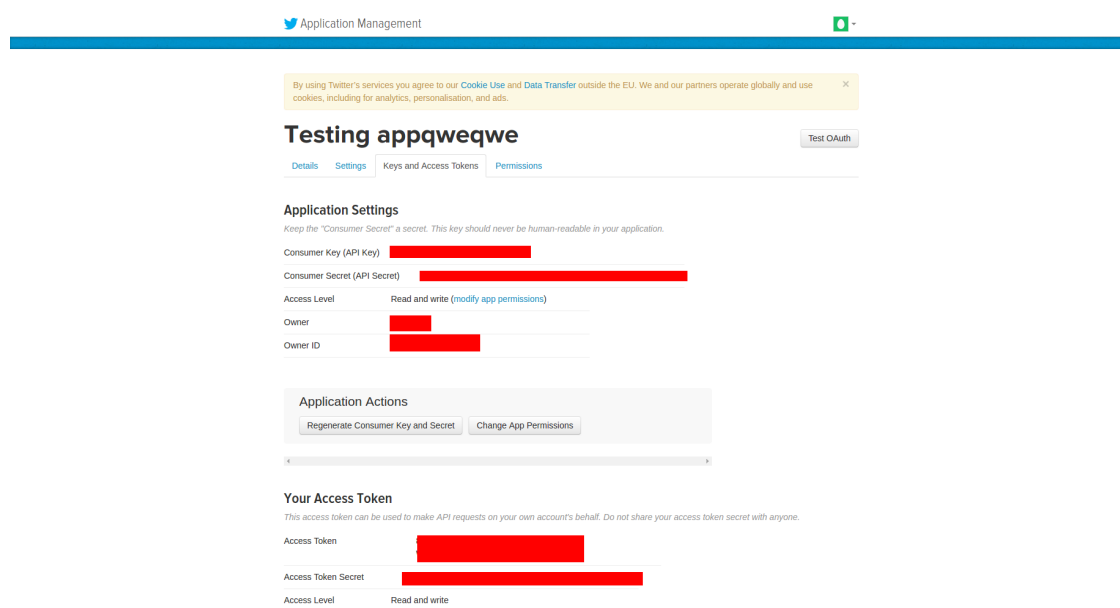
##### 2.4.6.1. Configurar cuenta de Twitter

El objetivo de esta funcionalidad es crear un objeto que será necesario para realizar peticiones de POST a la API de Twitter para publicar un mensaje.

El evento que escuchará será `bluetoothSerial:twitterConfig`, que recibirá como datos un array formado por los siguientes datos:

- API Key
- API Secret
- Consumer Key
- Consumer Secret

Estos datos se pueden obtener en la página <https://apps.twitter.com>, en la que se debe crear una nueva aplicación para interactuar con Twitter. Una vez se cree, se genera automáticamente una API Key y un API Secret. Para conseguir un Consumer Key y un Consumer Secret, se debe crear un token.



The screenshot shows the Twitter Application Management interface. At the top, there's a navigation bar with 'Application Management' and a 'Test OAuth' button. Below this is a yellow banner with a cookie notice. The main section is titled 'Testing appqweqwe' and has tabs for 'Details', 'Settings', 'Keys and Access Tokens', and 'Permissions'. The 'Settings' tab is active, showing 'Application Settings'. It includes fields for 'Consumer Key (API Key)', 'Consumer Secret (API Secret)', 'Access Level' (set to 'Read and write (modify app permissions)'), 'Owner', and 'Owner ID'. Below these are 'Application Actions' with buttons for 'Regenerate Consumer Key and Secret' and 'Change App Permissions'. The 'Your Access Token' section shows the 'Access Token', 'Access Token Secret', and 'Access Level' (set to 'Read and write').

#### 2.4.6.2. Publicar mensaje en Twitter configurado previamente

Esta funcionalidad permite publicar mensajes en Twitter siempre que haya sido configurado previamente.

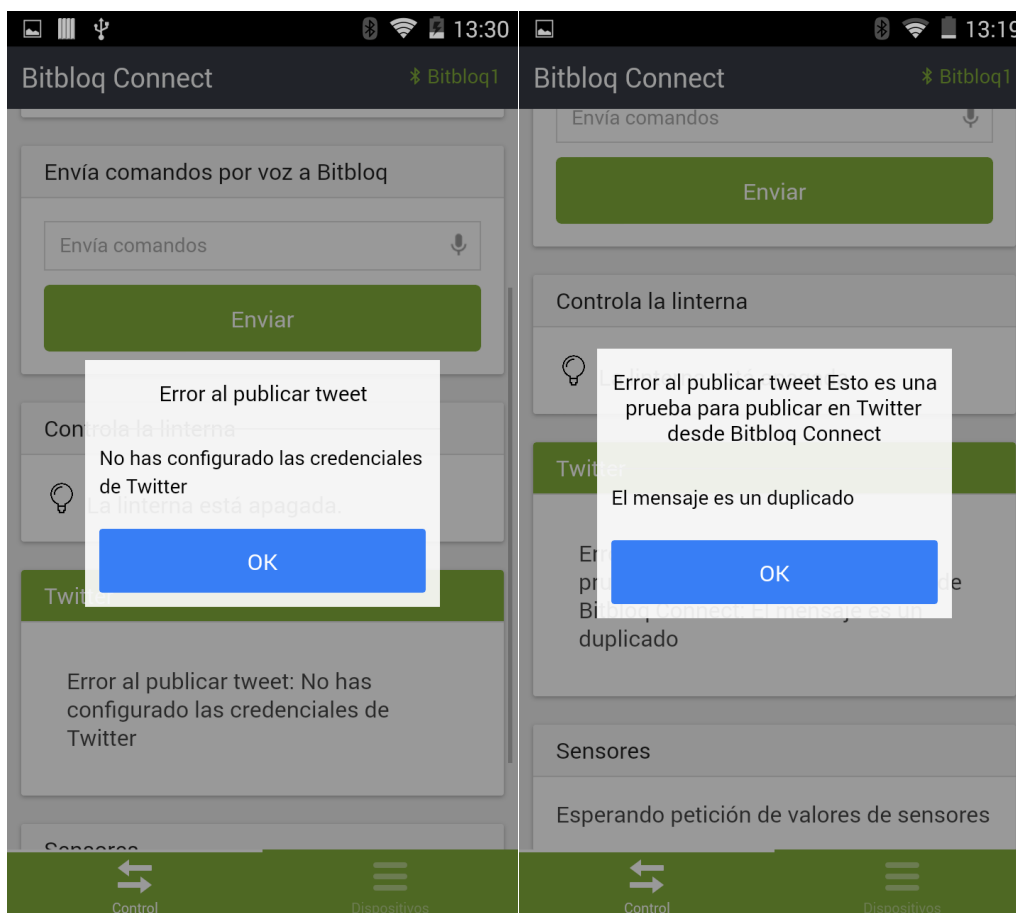
El evento que escuchará será `bluetoothSerial:twitterSend`, que recibirá como datos el mensaje a publicar.

Se ejecutará la función `sendTwitter`, que lo primero que comprobará es si se han definido credenciales. En caso de no haberlo hecho, se le mostrará al usuario un mensaje informándole. En caso de que sí las haya configurado, se llamará a la función de `sendTwitter` de la librería. Si el tweet ha sido correctamente publicado, se informará en la pantalla de la aplicación.



Se contemplan dos posibilidades en cuyo caso el mensaje no se publica:

1. **Las credenciales son erróneas:** en este caso, el usuario ha especificado mal las credenciales y es avisado tanto en la pantalla principal como mediante una ventana emergente.
2. **Ha publicado el mismo tweet muy poco tiempo:** de cara a evitar el spam, la política de Twitter no permite publicar el mismo mensaje inmediatamente. Por tanto, si esto ocurre, se le informará el usuario tanto en la pantalla principal como mediante una ventana emergente.



#### 2.4.7. Placa recibe datos de los sensores

Esta funcionalidad permite obtener los datos de los sensores tanto hardware como software del dispositivo móvil. Como se explicó en el apartado 2.1.6., para desarrollar la funcionalidad previamente se realizó un análisis de los sensores que estaban disponibles en el móvil de pruebas (Aquaris E5) y finalmente se desarrolló esta funcionalidad para los sensores que se indican en la siguiente tabla, en la que también aparece el evento que escuchan y de la función que se ejecuta.

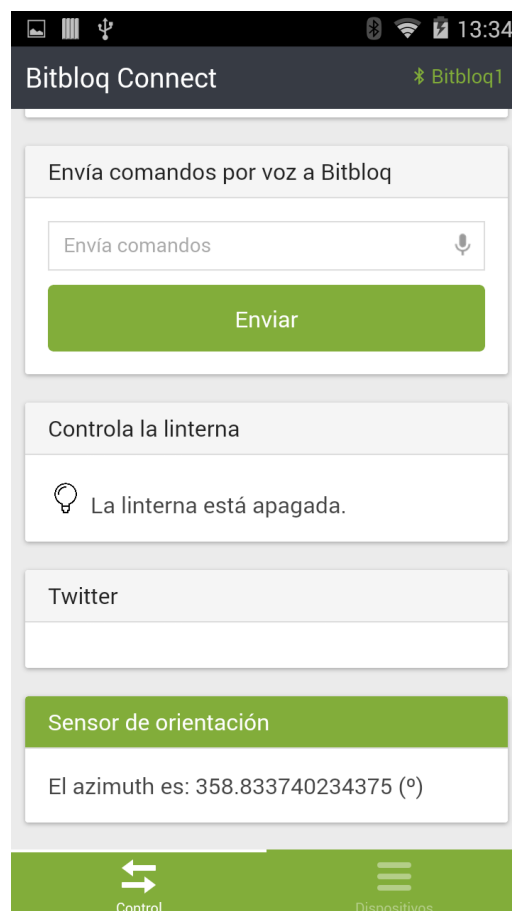
Sensor	Evento	Función
Acelerómetro (valores en eje x, y, z).	bluetoothSerial:readAccel	sendAccel
Aceleración lineal (valores en eje x, y, z).	bluetoothSerial:readLAccel	sendLAccel
Giroscopio (valores en eje x, y, z).	bluetoothSerial:readGyros	sendGyros



Sensor de campo magnético (valores en eje x, y, z).	bluetoothSerial:readMagnetic	sendMagnetic
Sensor de luz.	bluetoothSerial:readLight	sendLight
Aceleración de la gravedad (valores en eje x, y, z).	bluetoothSerial:readGravity	sendGravity
Sensor de proximidad.	bluetoothSerial:readProx	sendProx
Sensor de orientación (valores de azimuth, roll y pitch).	bluetoothSerial:readOrientation	sendOrientation

En el caso de que el sensor proporcione los valores en cada uno de los ejes, el evento recibirá como datos el eje en el que se desea mediar, que luego se le pasará a la función. En caso que sea una magnitud absoluta (como la luz), no recibirá datos. Cabe destacar el caso del sensor de proximidad, que recibirá como dato si se desea saber si está tapado o no tapado.

Todas las funciones asociadas devolverán el valor solicitado a la placa y lo escribirán en la pantalla principal de la aplicación, como se puede observar en la siguiente figura (en este caso, la placa solicita el valor del azimuth).



### 3. Manual de Usuario

En este apartado se define la interacción del usuario con la aplicación Bitbloq Connect. Está disponible para dispositivos Android en Play Store a través del siguiente enlace:

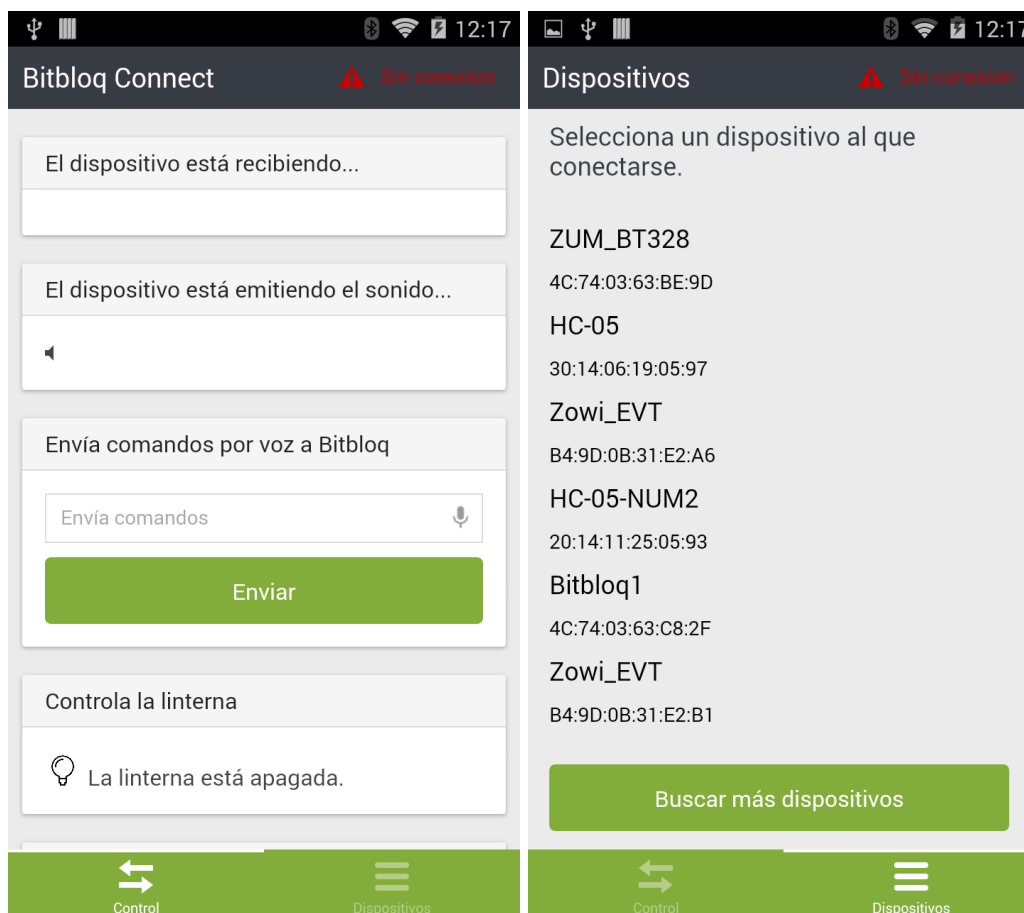
<https://play.google.com/store/apps/details?id=com.bq.bitbloq.connect>.



De modo general, el usuario deberá ser capaz de:

1. Conectar el móvil a un dispositivo emparejado mediante Bluetooth.
2. Recibir datos del dispositivo emparejado o enviarle datos al mismo.

Por ello, como se indicó en el apartado de Diseño, se decidió diseñar la aplicación con un menú de dos pestañas, una que fuera la aplicación en sí, y otra que fuera 'Dispositivos', dónde se pudiera conectar con el dispositivo (una placa, en nuestro caso).



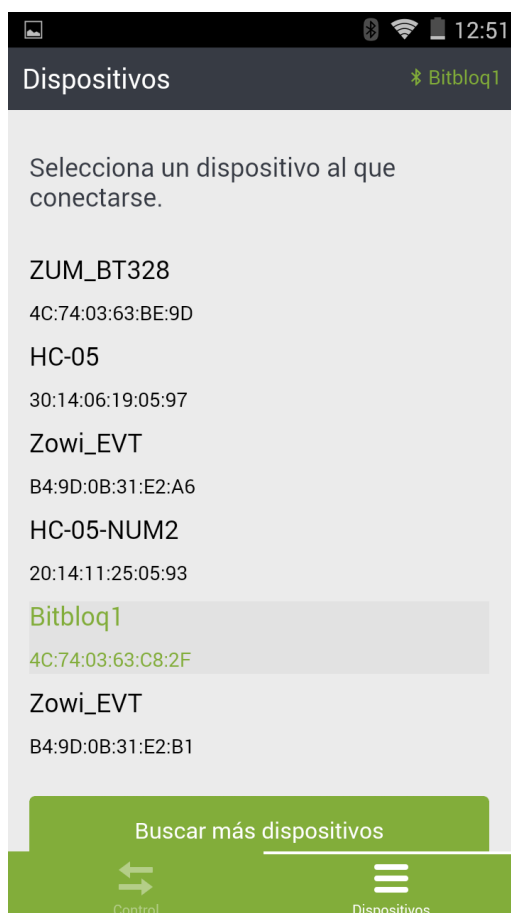
La interacción básica del usuario se resume en los siguientes pasos:

1. Inicialmente, en la pestaña Dispositivos se le muestra al usuario los dispositivos que tiene enlazados. El usuario deberá pulsar uno de ellos para conectarse. Como en el caso de este documento el dispositivo Bluetooth siempre será una placa, nos referiremos al mismo con este nombre. La placa debe estar programada previamente con Bitbloq (para ello, se debe consultar el documento E5.3.1). En el caso en el que la placa no aparezca en el menú de sincronizados, el usuario podrá pulsar el botón “Buscar más dispositivos”. Una vez hecho esto aparecerá la siguiente imagen.

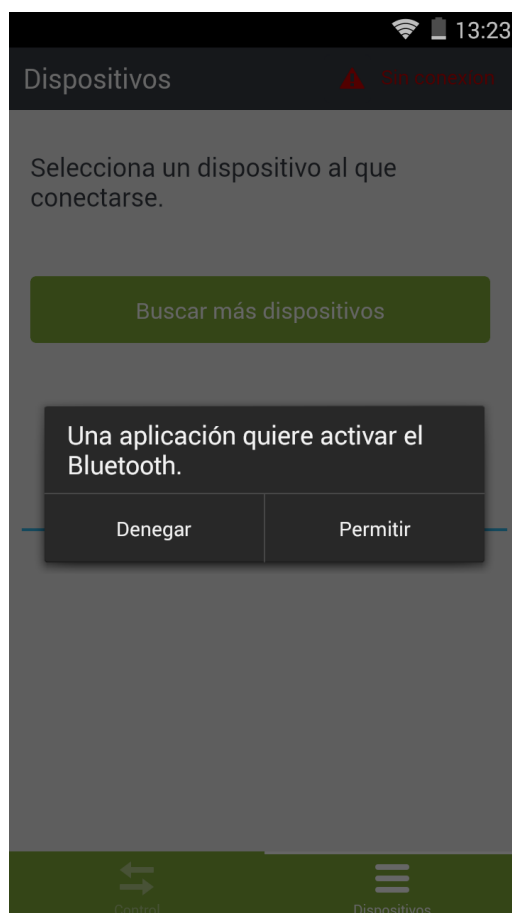


El usuario deberá pulsar el botón de Buscar dispositivos, seleccionar la placa y enlazarla al dispositivo móvil.

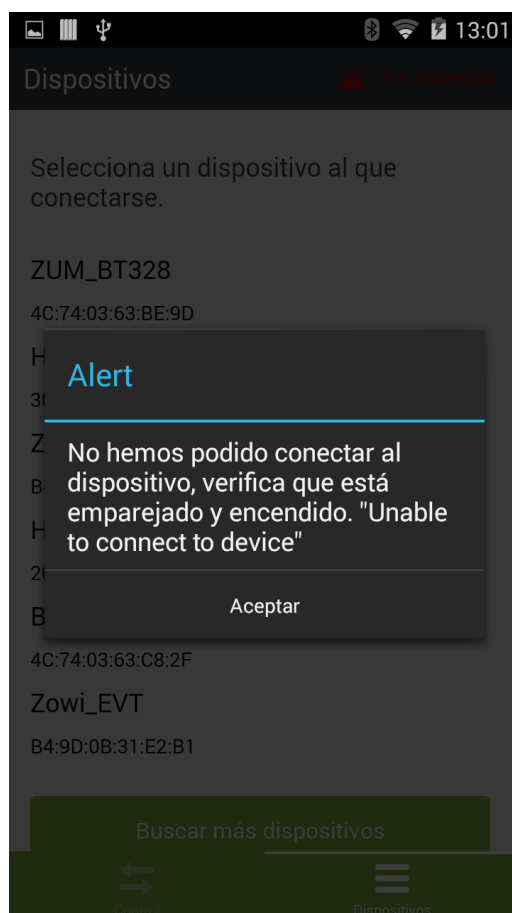
2. Una vez enlazada la placa, el usuario debe pulsar la tecla de retroceso del móvil, lo que hará que se vuelva a la pestaña Dispositivos. En este momento, la lista estará actualizada con el nuevo dispositivo (la placa, en este caso) enlazado. Para conectar con la placa, el usuario deberá pulsar sobre el dispositivo (en nuestro caso Bitbloq1).



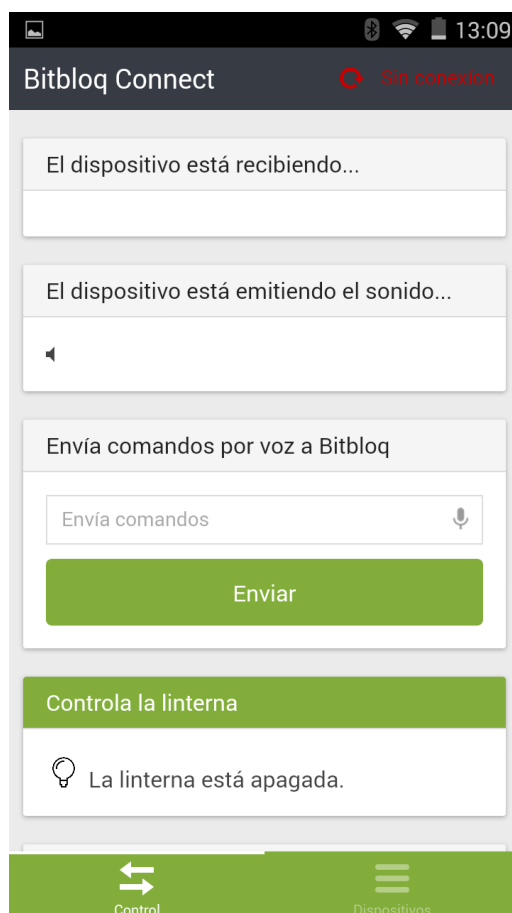
En el caso en el que el Bluetooth no esté activado, la aplicación mostrará un mensaje con un botón para activarlo.



Una vez conectado aparecerá un mensaje en la barra gris oscuro que indica el dispositivo con el que la aplicación está conectada. En el caso que se pierda la conexión Bluetooth, aparecerá un mensaje que indicará que no se ha podido conectar con el dispositivo.



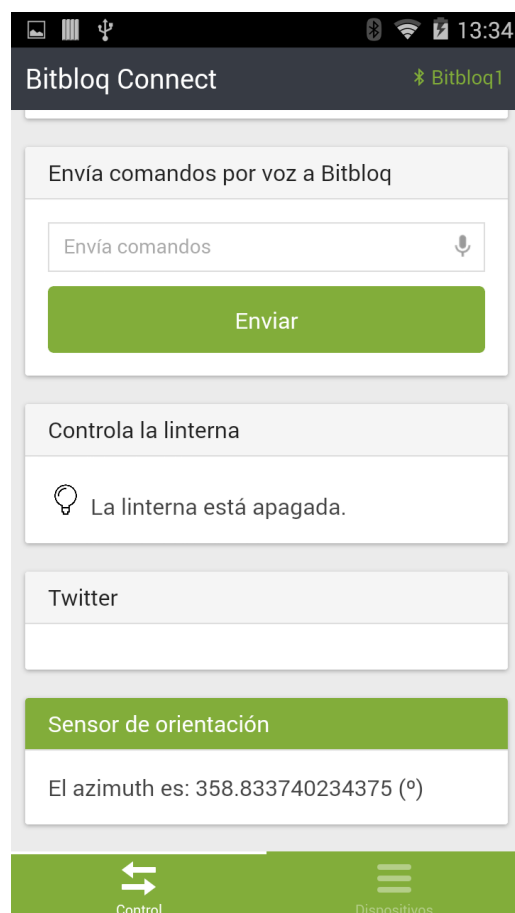
Además, el mensaje de la barra cambiará a Conexión perdida y al darle al símbolo de su izquierda se podrá volver a conectar (siempre que el Bluetooth de la placa esté disponible).



Una vez conectado el dispositivo, el usuario recibirá los comandos que envía la placa y generarán una acción concreta que se podrá observar en la pantalla Control de la aplicación.

Como se indicó en el apartado de diseño, el bloque de la pantalla principal que responda al comando enviado por la placa se coloreará en verde para indicar al usuario la interacción de la placa. Esto se puede observar en la siguiente imagen.





En este caso concreto, la placa estaba solicitando al móvil el valor del azimuth.