

EAGAL

Sistemas Operativos Avanzados



Cuatrimestre: 1er cuatrimestre

Año: 2018

Integrantes:

- Calvo, Rodrigo
- Corleto, Lucia
- Garcia, Diego
- Martinez, Julian
- Nosedá, Maximiliano

# Índice

Introducción: -----	3
Materiales: -----	4
Sensores: -----	5
Problemas: -----	10
Aplicación Android: -----	12
Desarrollo:-----	12
Diagramas en bloque: -----	15
Bibliografía Utilizada:-----	17

## Introducción

A continuación, se describirán las funcionalidades y prestaciones propuestas por el producto Eagal. Su fin principal es proveer al usuario de un dispositivo el cual podrá utilizar como reloj despertador inteligente, siendo este configurable totalmente desde una aplicación para dispositivos móviles Android. A su vez, Eagal posee una serie de sensores conectados que suministrarán información relevante al usuario mediante su pantalla táctil o en la aplicación.

El producto está diseñado para conectarse a Internet mediante una conexión Wi-Fi, mediante la cual proveerá los datos pertinentes a la nube para ser consultados por la aplicación así como también por esta vía recibirá las instrucciones necesarias para su configuración.

El módulo principal es el de la alarma inteligente, que consta de un botón y un parlante exclusivos para este fin. El dispositivo interconectado con un pequeño parlante será el encargado de hacer sonar y notificar al usuario de que llegó la hora del día indicada. El horario en el que la alarma va a sonar y la canción predefinida se configuran desde la aplicación y se pueden ver tanto desde la misma como desde la pantalla.

Al estar totalmente interconectado, la aplicación permite apagar la alarma en el instante en que está sonando con solo agitar el celular, adicionalmente al botón ubicado en el dispositivo en el caso de estar cerca.

Eagal también tiene la capacidad mediante sus sensores, de medir la temperatura y humedad del ambiente donde se ubique, siendo esta una funcionalidad muy interesante para calefaccionar o refrigerar el lugar físico en el que se encuentra. Esta información puede ser vista tanto en la pantalla del dispositivo como a través de la aplicación, ya que la misma se actualiza cada un periodo de tiempo hacia la nube.

Adicionalmente a la temperatura ambiente, Eagal realiza consultas periódicas mediante internet a un servicio externo de clima al cual consulta según la ubicación provista por el GPS del celular, información obtenida mediante la aplicación que luego informa en la pantalla. De esta manera el usuario puede ver rápidamente al utilizar el producto, el pronóstico del clima en la ciudad en la que se encuentra.

Por último, el producto también cuenta con un sensor de luz el cual utiliza para regular automáticamente el brillo de la pantalla y así lograr un consumo adecuado de energía. También brinda la posibilidad de adecuarse al brillo captado por el sensor de luz que tenga el celular ya que suelen ser más precisos.

## Informe de desarrollo

En este informe se explicarán los materiales que fueron utilizados y se dará una explicación del desarrollo del sistema tanto a nivel sistema embebido como aplicación. También detallaremos el funcionamiento de los sensores elegidos para incluir en el dispositivo, así como algunos problemas implementando todo a nivel hardware.

Luego se pasa a describir el entorno elegido para el desarrollo del software como también la distribución lógica de los componentes y sus funciones.

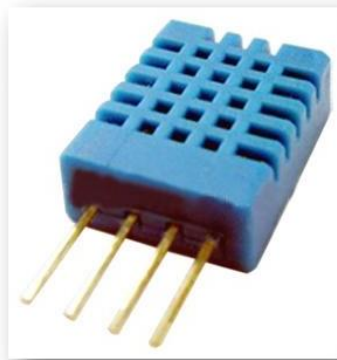
## Materiales

A continuación, se detallarán los materiales que se utilizaron para la realización del sistema junto con el uso de cada uno de estos y un apartado de los sensores utilizados con su respectivo funcionamiento.

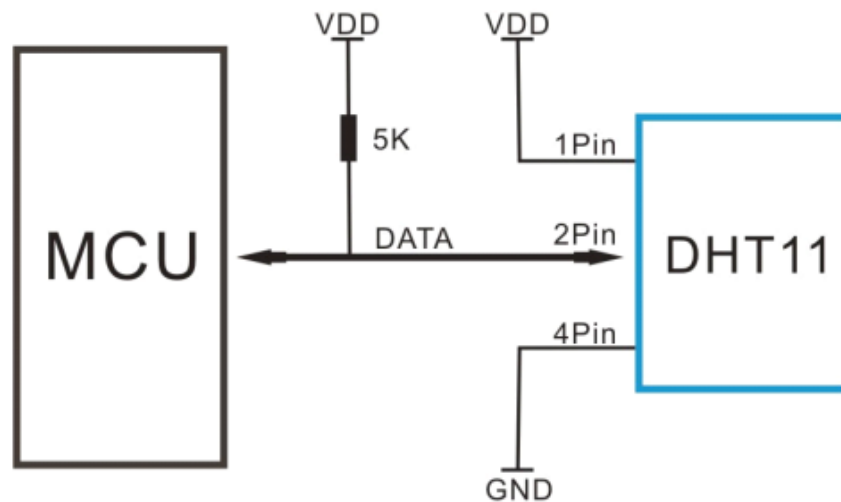
- Microcontrolador Particle Proton (P0): Sistema embebido utilizado para manejar la lógica de los sensores y actuadores.
  - Microcontrolador ARM Cortex M3.
  - Wi-Fi Module: Este módulo lo utilizamos para comunicar el embebido con la aplicación, por medio del servidor que proporciona el Particle Proton en la nube.
  - Fuente Alimentación: Photon se alimenta con un cable USB Micro B directamente en el conector VIN, el voltaje requerido debe estar entre 3,6 y 5.5VDC.
- Resistencia: 220 Ohm.
- Transistor: lo utilizamos para amplificar la salida del microcontrolador a 5V ya que la misma es de 3V.
- Módulo Botón KY-004: Es un módulo preparado con resistencia pull down para eliminar el ruido que puede generar falsos estados y un botón táctil. Se alimenta con 5V y su salida es digital, un 1 (al presionarlo deja pasar la corriente de entrada) o un 0 (cuando no está presionado).
- Pantalla Nextion Enhanced 3,5'': Es una pantalla TFT con un panel táctil. La misma posee un micro basado en ARM, una memoria flash dedicada al control de la pantalla y una memoria DRAM para gestionar el hardware y la actualización de la pantalla.
- Cables Jumper: Utilizados para realizar la conexión entre el embebido con los sensores y los actuadores.
- Protoboard: Sirve de soporte para el cableado entre los distintos dispositivos con el microcontrolador ya que posee conexiones internas siguiendo patrones para conectar componentes electrónicos (en nuestro caso los sensores y el buzzer) y cables para el armado de circuitos.
- Sensores: Detallados a continuación.

## Sensores

### DHT11



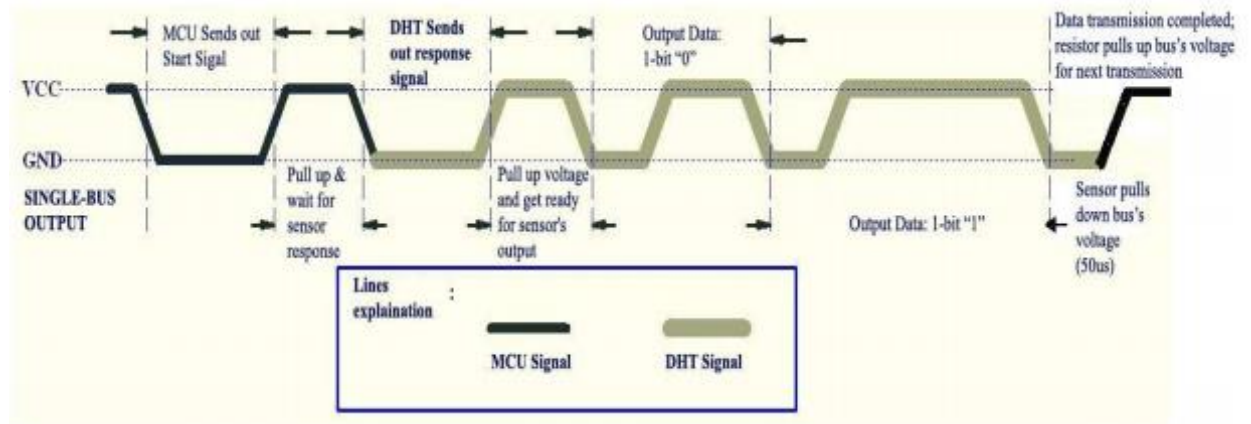
- Sensor de temperatura y humedad que utiliza el protocolo de comunicación Single-Wire (una línea).



- Una comunicación con el DHT11 lleva aproximadamente 4 milisegundos.
- Una respuesta contiene 40 bits, de las cuales se envía primero el bit más significativo, y los datos corresponden a:
  - 8 bits para la porción entera de la humedad relativa.
  - 8 bits para la porción decimal de la humedad relativa.
  - 8 bits para la porción entera de la temperatura.
  - 8 bits para la porción decimal de la temperatura.
  - 8 bits para comprobación de errores (check-sum)

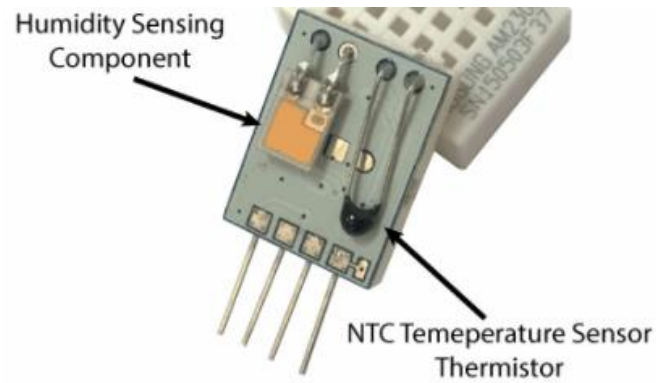
## Método de comunicación

- El bus de datos se encuentra siempre en estado alto para indicar que está libre.
- El MCU envía una señal de inicio cambiando el estado de alto a bajo. Luego, el estado se cambia nuevamente a alto, para que el DHT11 sepa que el bus de datos está libre. Inmediatamente, el DHT11 cambia el estado de alto a bajo para indicarle al MCU que va a transmitir. Luego levanta la señal nuevamente por unos microsegundos y comienza a transmitir los bits de los datos recolectados. Para cada transmisión de bit envía un pulso bajo de 50 microsegundos y luego un pulso alto. La duración del pulso determina si el dato es un cero o un uno. Para los ceros, se envía inmediatamente un pulso alto de 26 microsegundos. Para los unos, se envía inmediatamente un pulso alto de 70 microsegundos.
- Para finalizar la transmisión, el DHT11 cambia el estado a bajo durante 50 microsegundos y luego lo devuelve al estado alto para indicar que el bus está libre.

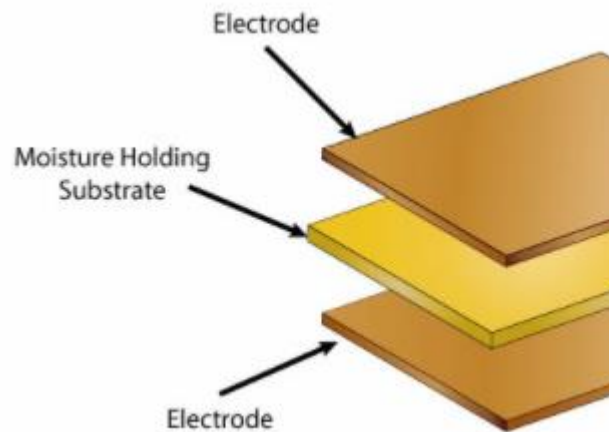


## Funcionamiento físico

El sensor DHT11 se compone internamente de un componente sensor de humedad y un componente sensor de temperatura (NTC, termistor), sumado al circuito integrado que permite la comunicación con el protocolo mencionado.



Para el sensor de humedad se utilizan dos placas paralelas denominadas electrodos, que contienen un material entre ellas que conserva la humedad. Por ello, se mide el cambio de la resistencia entre ambos electrodos a medida que el material interno absorbe más humedad.

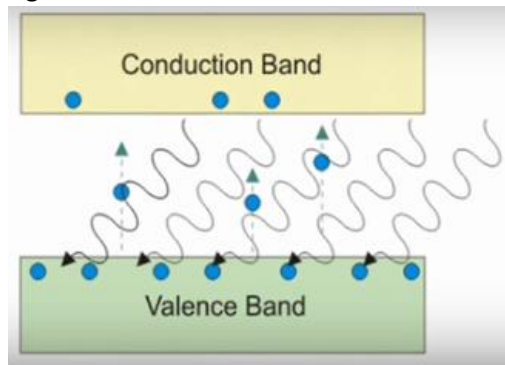


Para el sensor de temperatura, se utiliza un resistor variable que cambia su resistencia con la temperatura. Por ser un sensor NTC (Negative Temperature Coefficient), el sensor disminuye su resistencia a medida que aumenta la temperatura. Consisten en un material polímero o cerámico que varía en gran cantidad la resistencia a pequeños cambios de temperatura.

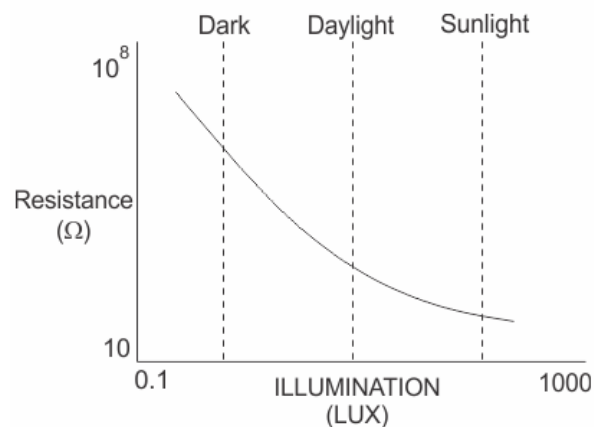
## LDR



- Sensor de luz que varía su resistencia según la incidencia de radiación electromagnética.
- Los LDR funcionan con el principio de fotoconductividad, que es un fenómeno óptico-eléctrico en el cual la conductividad de un material aumenta cuando la luz es absorbida por el mismo.
- El LDR está compuesto por una banda de valencia y una banda de conducción. Cuando los fotones ingresan a la banda de valencia, los electrones se excitan y desplazan a la banda de conducción. De esta forma, cuando muchos fotones ingresan a la banda de valencia aumentan los electrones en la banda de conducción, permitiendo grandes desplazamientos de carga.

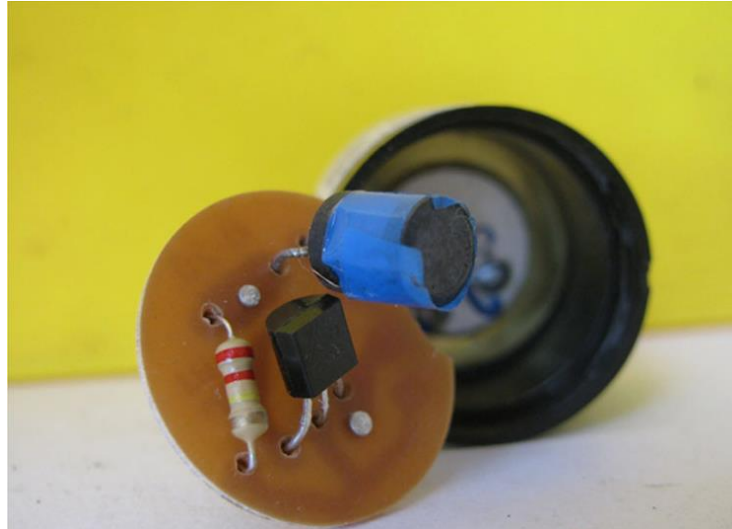


Los LDR entonces, reducen la resistencia a medida que ingresa más luz a la banda de valencia, y son fabricados con estructuras cristalinas fotoconductoras, como el sulfuro de cadmio y seleniuro de cadmio.





## Buzzer piezoeléctrico



- El buzzer piezoeléctrico contiene un resistor, un transistor y un inductor.
- El transistor y el resistor en combinación funcionan como un circuito oscilador y producen oscilaciones de baja amplitud con corriente directa. La magnitud de las oscilaciones se amplifica con el inductor.
- Luego se coloca encima un diafragma circular que es el que produce el sonido. El diafragma es metálico, de bronce o acero inoxidable, y se le incorpora un adhesivo conductor. La placa metálica entra en resonancia con la frecuencia del circuito.
- Cuando se aplica una tensión alternante al piezoeléctrico, hace que la placa metálica se expanda y se contraiga en dirección radial. Esto genera que el metal se doble en direcciones opuestas. Cuando la placa metálica se expande y contrae continuamente se producen ondas de sonido en el aire.

## Problemas

Aquí se detallarán los problemas que fueron encontrados a medida que se fue avanzando en el desarrollo del sistema y las soluciones a las que se llegaron.

### **Problemas con el hardware:**

- 1. Cambio de librería de módulo sensor DHT11:** La librería más utilizada del módulo DHT11 no realiza la comprobación de errores de lectura del mismo, y en nuestro primer uso no estábamos recibiendo datos. Para poder verificar por qué no se obtenía lectura del sensor tuvimos que utilizar otra librería que maneja todos los errores que informa el módulo sensor, como por ejemplo, si el sensor todavía se encuentra adquiriendo la información para transferir o si hubo un fallo al verificar los datos con el checksum. Gracias a esto pudimos comprobar que el sensor requería de un tiempo mínimo entre lecturas (2,5 segundos) y que debíamos hacerlas más espaciadas entre sí para comprobar su funcionamiento.
- 2. La pantalla no responde a los comandos:** La pantalla utilizada se comunica con el microcontrolador a través de la interfaz serial. En un principio la pantalla no respondía a los comandos enviados. Navegando nuevamente la documentación observamos que debían enviarse 3 caracteres especiales al final de cada comando para que la pantalla reconozca a los mismos.
- 3. Problemas de concurrencia con la interfaz serial:** El uso de la interfaz serial genera problemas en el caso de que ocurra una interrupción de nuestro botón o se atienda un pedido de internet, ya que podría no poder completarse un envío de comando a la pantalla. De esta manera, decidimos utilizar objetos Timer de la librería que provee el fabricante, que al ser llamados a ejecución se encolan en un stack y se ejecutan atómicamente uno detrás de otro en un núcleo del procesador aparte al de proceso principal. Esto nos permite también que la máquina de estados se ejecute constantemente, independientemente de si se está actualizando el pronóstico en la pantalla u otros datos.
- 4. El microcontrolador deja de responder:** En una versión el microcontrolador dejó de funcionar debido a que las canciones utilizadas para la alarma eran muy extensas y no cabían en la memoria del dispositivo. Tuvimos que realizar un procedimiento para poner el microcontrolador en modo seguro y volver a flashearlos, ya que en nuestro dispositivo el firmware se instala a través de WIFI.

## Problemas con el software:

5. **Instalación y configuración del ambiente de trabajo:** Trabajamos con Visual Studio Code y tuvimos que configurar el IDE para que utilice las herramientas de react-native y el SDK provisto por Android Studio. Estuvimos con problemas con las variables de entorno hasta que descubrimos como realizar la instalación: Instalamos Android Studio para tener el SDK de Android y ADB, por otro lado instalamos Node.JS, Python, jdk8 y react-native. Luego de instalar todo tuvimos que configurar desde el registro de Windows la variable de entorno de forma manual para que CMD y PowerShell soporte los comandos ADB y los de compilación y ejecución de react-native.
6. **Utilización de sensores:** React-native no provee soporte nativo para varios de los sensores de un dispositivo Android. Por ejemplo para poder utilizar el sensor de luz para variar el brillo de la pantalla de Egal tuvimos que instalar una extensión de React.
7. **Problemas con la actualización de parámetros en Egal:** Por motivos de diseño de la nube del embebido todos los post de datos son realizados en formato JSON con todos los nodos en valores string. Nos pasaba que si debíamos enviar un "1" o un "0" algunas veces lo enviábamos en forma string o entero sin criterio según como fue desarrollada esa parte de la app. Cuando detectamos que lo mejor era enviar todo en string hicimos la convención y todo dato enviado o recibido en JSON con la nube de Egal es en formato string.
8. **Valores del sensor de luz:** En Android los valores que puede tomar el sensor de luz oscilan entre 0 y 30000 pero por diseño en Egal aceptamos valores "porcentuales" que van desde 0% a 100%. Lo que hicimos para volverlos compatibles es calcular qué % de luz fue devuelta por el sensor de luz; es decir qué porcentaje del máximo de luz fue captado por el sensor para que el valor devuelto oscile entre 0 y 100.

## Aplicación Android

La aplicación para Android fue desarrollada utilizando el framework React Native el cual provee una plataforma que permite construir aplicaciones nativas utilizando código JavaScript y componentes React.

El fin de la aplicación es interconectarse con el dispositivo Egal para obtener la información de los sensores así como también ser la interfaz de configuración para que el usuario pueda interactuar de forma clara, fluida y amigable con el sistema embebido.

La manera de conectarse con Egal es mediante conexión a internet (ya sea Wi-Fi o redes móviles) y el consumo de servicios Web del tipo REST, provistos por Particle. La aplicación tiene un Token con el cual se identifica y accede a las distintas rutas abiertas a la consulta y envío de datos mediante solicitudes HTTP. La infraestructura de estos servicios se encuentra en servidores externos, siendo esta nube el nexo entre la aplicación y el sistema embebido.

## Desarrollo

A modo de desarrollo, se explicará el flujo normal para utilizar la aplicación mencionando los elementos más significativos del proyecto.

El software fue diseñado con una clara división en componentes, puestos en común en una misma Actividad los cuales se distribuyen de la siguiente manera:

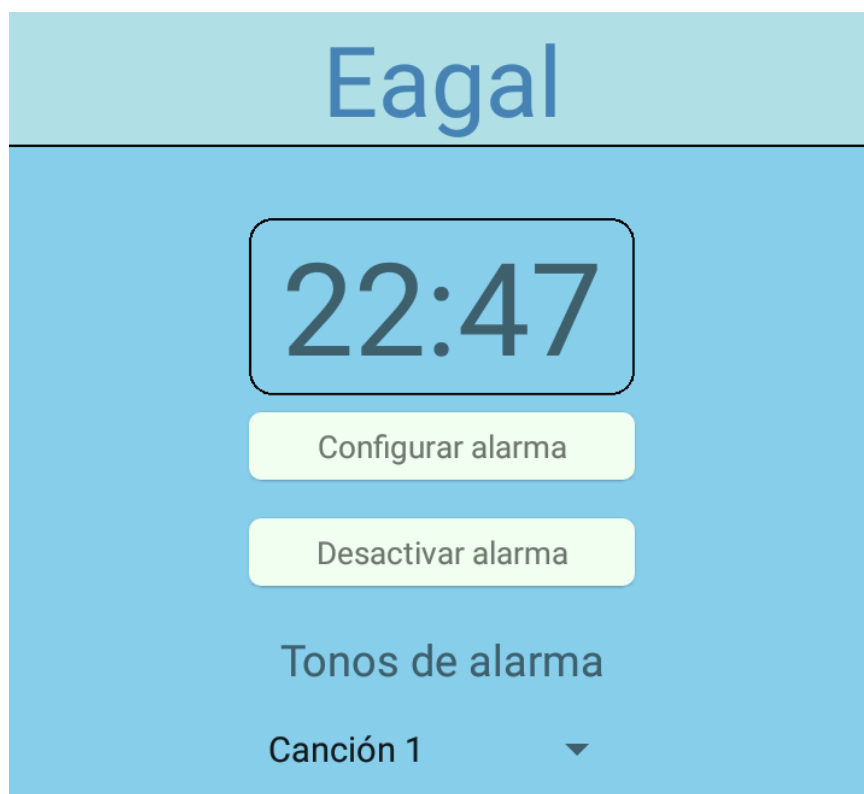
### **Alarma**

Este componente contiene en su definición las propiedades pertinentes para conocer y configurar la información relevante de la alarma, tanto el horario en el que está configurada en el dispositivo, como así los estados en los que la misma se encuentra (activada, desactivada, sonando, en espera, etc.)

También es el encargado de la comunicación con el acelerómetro, mediante el cual detecta un “shake” del dispositivo para apagar la alarma en el caso de que esté sonando, evento que genera que los números del reloj se conviertan en rojo.

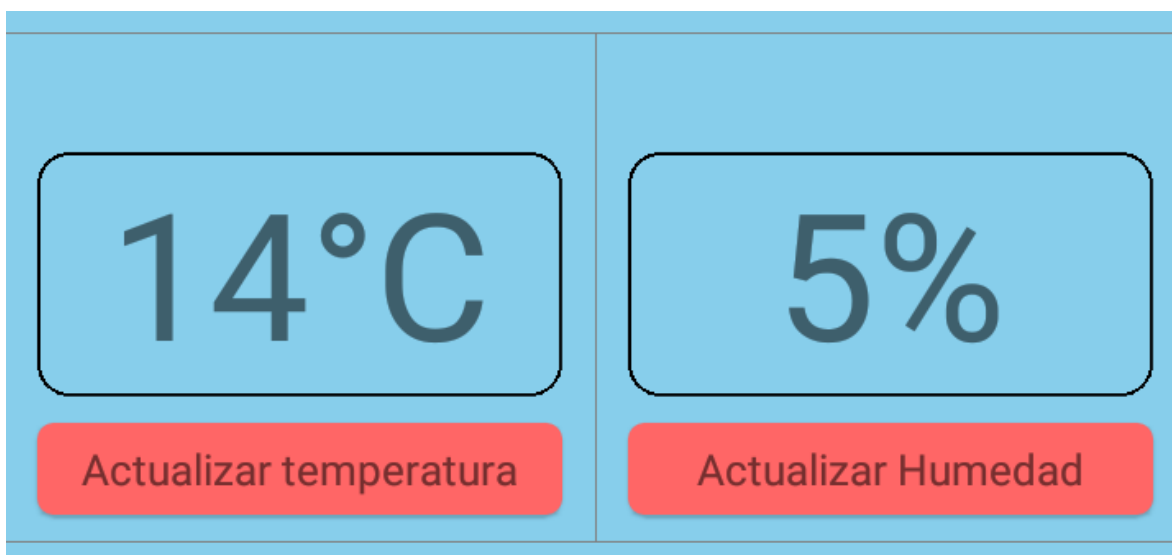
Consta de una serie de solicitudes ejecutadas asincrónicamente para ir actualizando los estados según lo reporte el dispositivo. A su vez, permite modificar la canción que será reproducida por la alarma en base a una lista predefinida.

En pantalla se muestra de la siguiente manera:



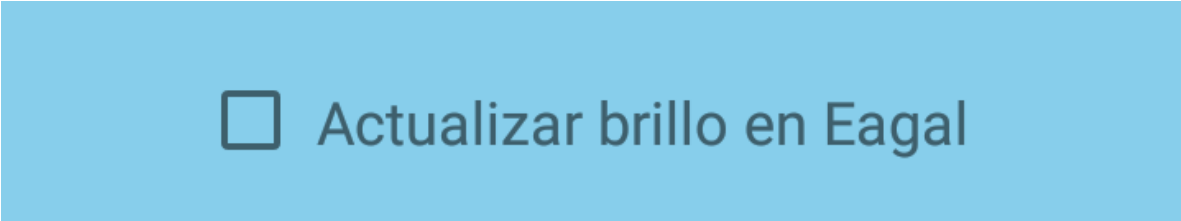
### Temperatura y humedad

Mediante este componente se presenta al usuario la posibilidad de ver en pantalla la información de los sensores de temperatura y de humedad, actualizándolos automáticamente cada cierto periodo de tiempo y brindando la opción de actualizarlos manualmente. Su visualización es la siguiente:



## Brillo

Es el componente encargado de obtener la información del sensor de luz y también tiene la posibilidad de activar la utilización de los valores obtenidos por dicho sensor para modificar el nivel de brillo de la pantalla de Eagal automáticamente. Utiliza una cuenta para traducir los niveles provistos por Android a lo compatible con el dispositivo. En pantalla se ve de la siguiente forma:

A screenshot of a mobile application interface. It features a light blue background. On the left, there is a white square toggle switch. To its right, the text "Actualizar brillo en Eagal" is displayed in a dark grey, sans-serif font.

## Ubicación

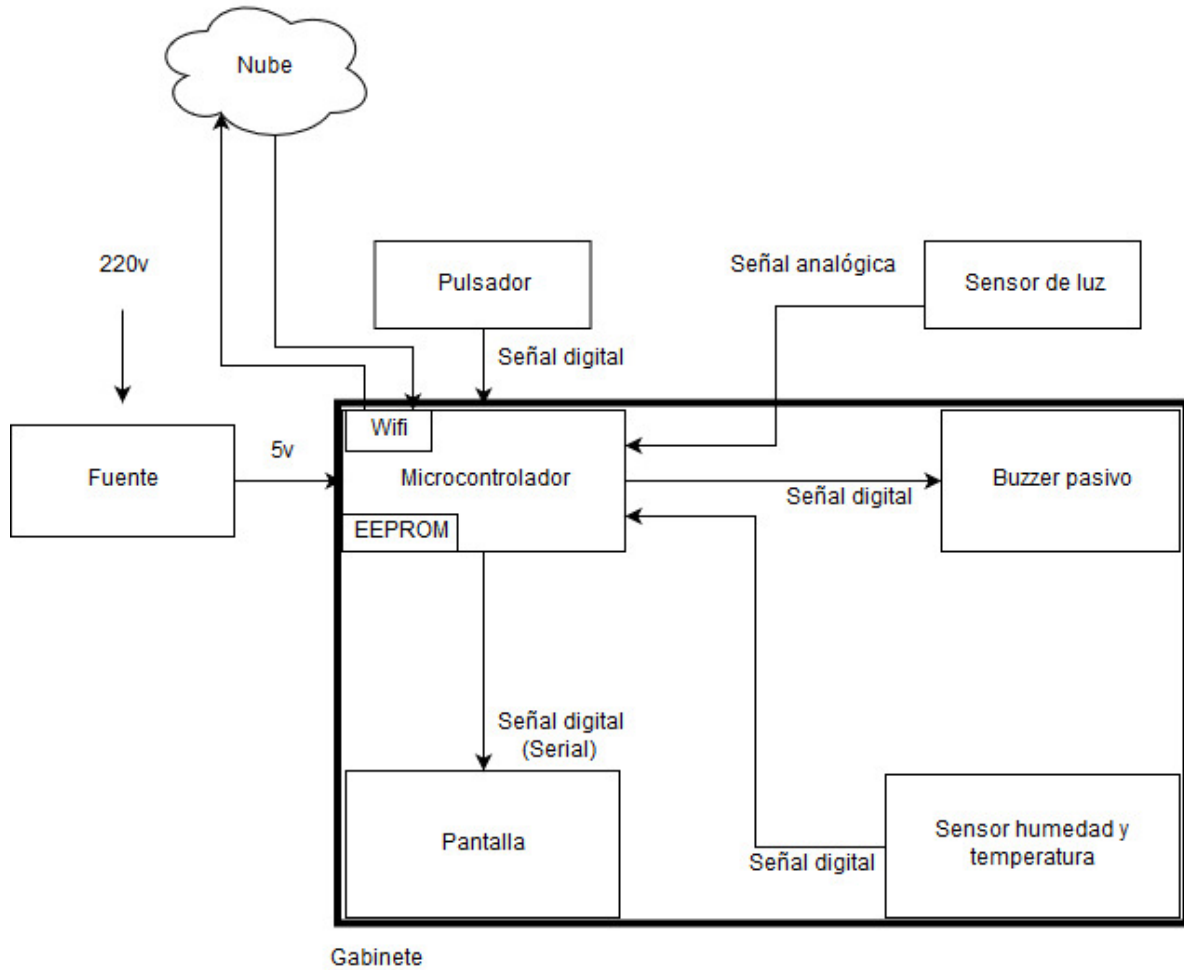
En este componente, se accede a la información del GPS del celular para su posterior informe a Eagal. Utilizando la geolocalización se obtienen coordenadas en latitud y longitud, por las cuales la aplicación contacta a un servicio Web de Google para obtener la dirección aproximada de dónde se encuentra el dispositivo en el momento. Posteriormente, se envían los datos a Eagal para que el dispositivo actualice el informe del clima con respecto a esta ubicación en pantalla.

En la aplicación se ve de la siguiente manera:

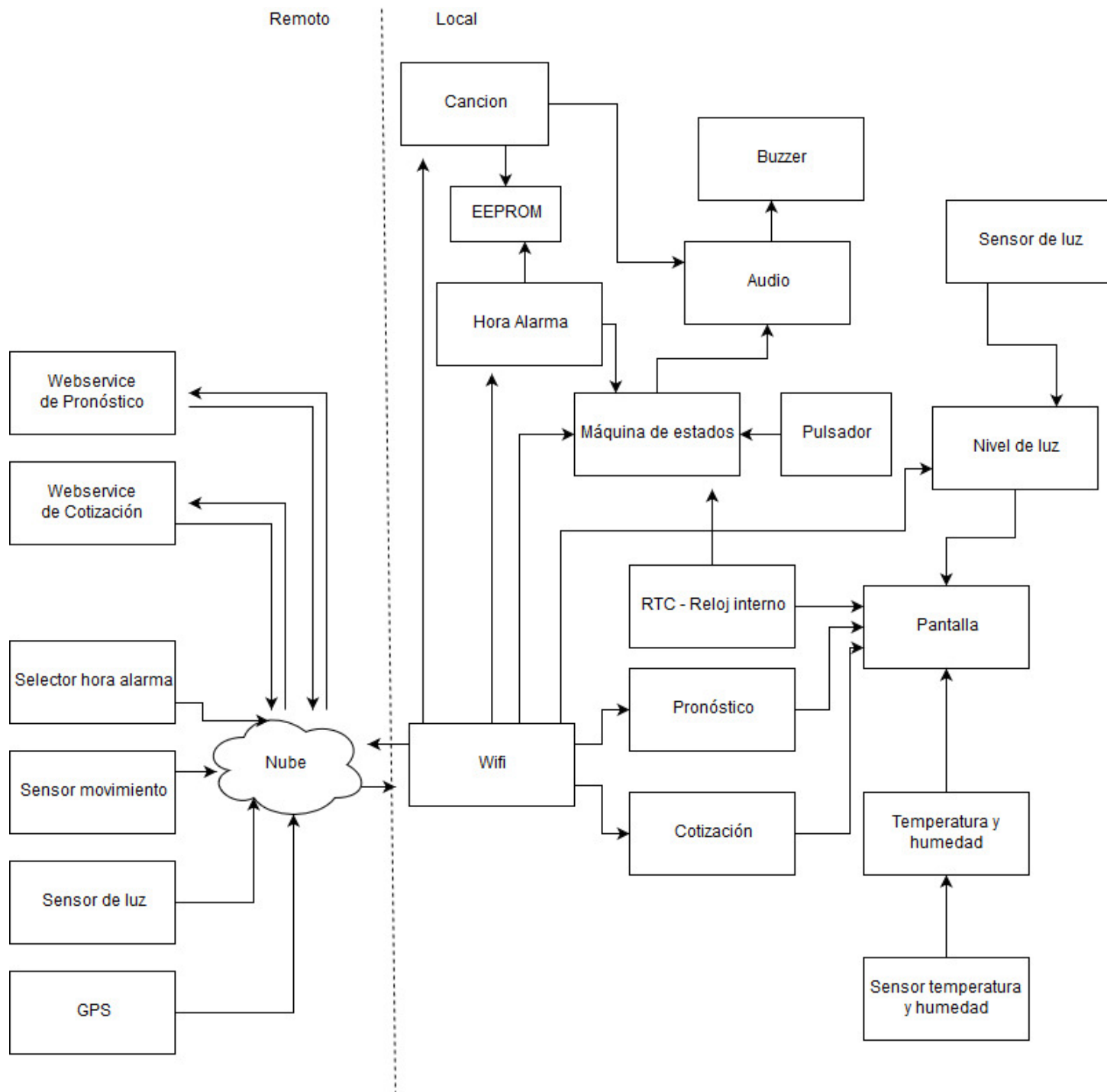
A screenshot of a mobile application interface. It has a solid blue background. At the top, the text "S.O.A - 2018" is centered in a white, sans-serif font. Below it, the address "San Francisco 1543, B1708BDS Morón, Buenos Aires, Argentina" is also centered in a white, sans-serif font.

## Diagramas en bloque

### Disposición física de los componentes

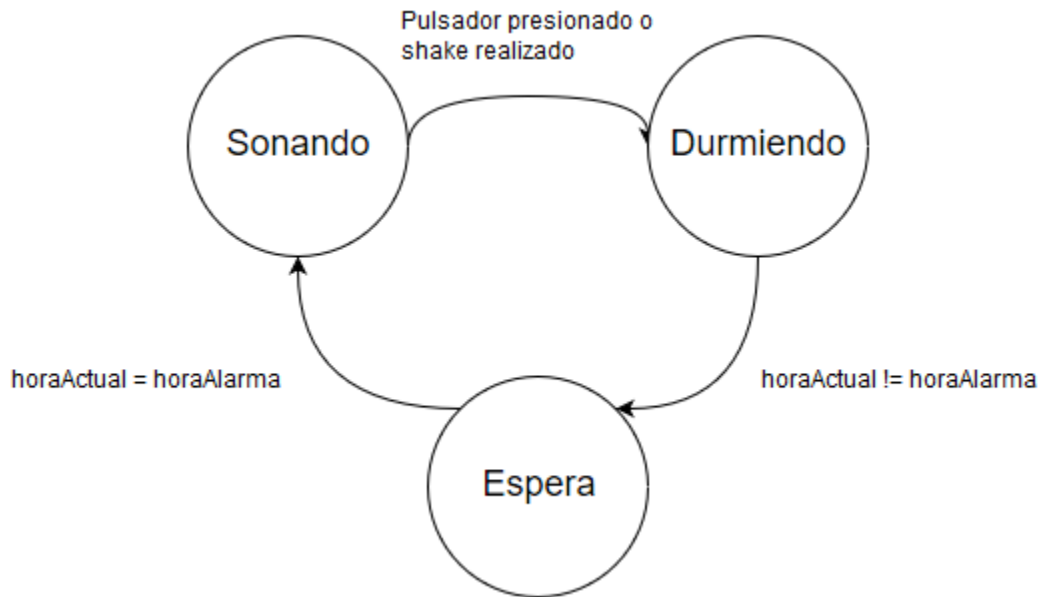


## Diagrama lógico





## Diagrama de estados



## Bibliografía Utilizada

- [https://docs.particle.io/datasheets/photon-\(wifi\)/photon-datasheet/](https://docs.particle.io/datasheets/photon-(wifi)/photon-datasheet/)
- <https://docs.particle.io/reference>
- <https://facebook.github.io/react-native/docs/getting-started.html>
- <https://nextion.itead.cc/resources>
- <https://reactjs.org/>
- <https://cdn-learn.adafruit.com/downloads/pdf/dht.pdf>
- <https://docs.particle.io/guide/getting-started/examples/photon/>
- <https://docs.particle.io/reference/firmware/photon/#tone->
- <https://docs.particle.io/reference/firmware/photon/#interrupts>
- <https://docs.particle.io/reference/firmware/photon/#time>
- <https://docs.particle.io/reference/firmware/photon/#software-timers>
- <http://diotlabs.daraghbyrne.me/6-controlling-outputs/piezo/>