



**FIC – UDC – 1º cuatrimestre 17/18**

**Proyecto Programación de Sistemas  
EasyArduino!**

<https://github.com/joseluis9595/easy-arduino>

Jose Luis Álvarez Quiroga – jose.luis.alvarezq@udc.es  
Matías Cubilla Currás – matias.cubillac@udc.es

# ÍNDICE

<b>Introducción</b>	<b>2</b>
<b>Funcionamiento</b>	<b>3</b>
<b>Diseño</b>	<b>4</b>
Arquitectura	4
Componentes	5
Diseño Gráfico	6
➤ Icono de la aplicación	6
➤ Tutorial	6
➤ Views personalizadas	7
➤ Animaciones y transiciones	8
➤ Otros	9
Reparto de trabajo	9
<b>Pruebas de funcionamiento</b>	<b>10</b>
Comunicación Android-Arduino	10
Gestión de paneles y controladores	11
Ajustes de usuario	12
<b>Aspectos a destacar</b>	<b>13</b>
<b>Trabajo futuro</b>	<b>13</b>
<b>Problemas conocidos</b>	<b>14</b>
<b>Referencias</b>	<b>14</b>

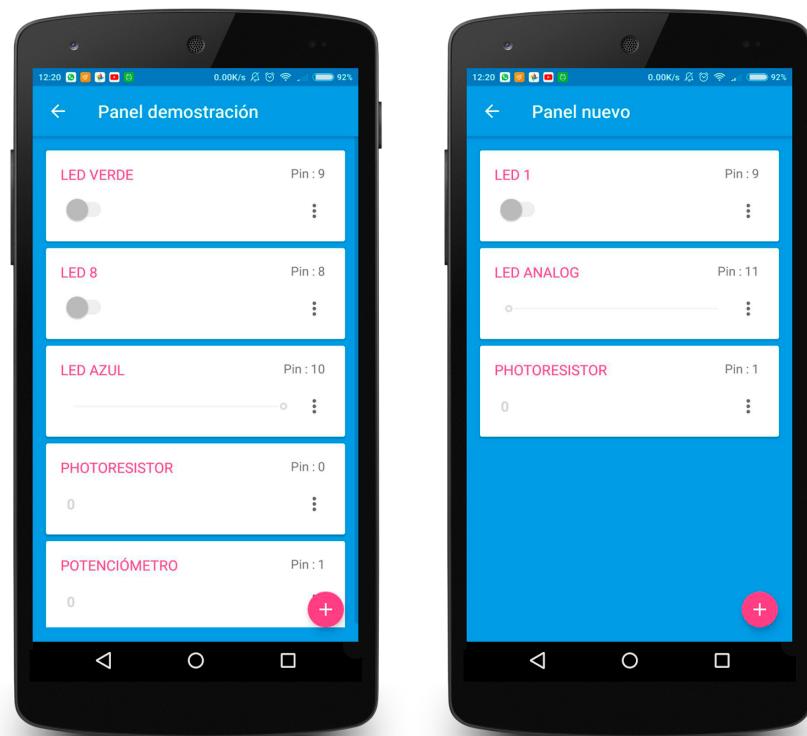
# Introducción

“Easy Arduino” es una aplicación enfocada a simplificar la interacción del usuario con un circuito conectado a una placa Arduino.

Ya que la aplicación se centra en la facilidad de uso, se ha desarrollado teniendo en cuenta que la interfaz de usuario ha de ser simple e intuitiva, y que la interacción del usuario debe ser lo más directa posible, evitando tener un número elevado de actividades o pasos intermedios para llegar a donde el usuario necesita llegar.

“Easy Arduino” trabaja mediante paneles y controladores, los cuales uno puede crear, modificar o eliminar en cualquier momento. Cada panel es la representación dentro de la aplicación de un “circuito”, y cómo es de esperarse, dentro de cada panel, dispondremos que un controlador por cada componente del circuito. Dichos controladores se dividirán principalmente en dos tipos, de lectura o de escritura, según reciban o manden datos. El manejo de estos controladores se ha implementado de tal manera que la interacción sea muy intuitiva y fácil.

Aquí tenemos dos ejemplos de dos paneles distintos, el de la izquierda contiene 5 controladores. Los dos primeros son de escritura digital, y se manejan mediante un switch. El tercero es un controlador de escritura analógica, y permite enviar datos deslizando el slider. Los dos últimos, son de lectura, y no permiten ninguna interacción por parte del usuario, solamente mostrarán los datos leídos del Arduino para cada uno de los pines indicados.



También, a la hora de acercar “Easy Arduino” a distintas personas del mundo, se ha desarrollado de tal manera que se pueda usar tanto en inglés como en español.

# Funcionamiento

Para utilizar nuestra aplicación son necesarios una serie de componentes: dispositivo Android, dispositivo Arduino y cable USB que permita conexión entre ambos (*es posible que se necesiten adaptadores para este fin*).

Una vez disponemos de estos componentes, necesitaremos subir el código .ino<sup>1</sup>, disponible [aquí](#) a nuestro Arduino. Tras esto, ya podemos conectar el Arduino a nuestro dispositivo Android y comenzar a usar la aplicación.

El funcionamiento de la aplicación se basa en el paso de mensajes entre Arduino y nuestro dispositivo Android, de momento mediante tecnología USB<sup>2</sup>. Dichos mensajes, serán interpretados en Arduino, que actuará conforme a los datos recibidos.

Explicaremos la estructura de dichos mensajes mediante una serie de ejemplos.

Mensaje enviado desde Android a Arduino, para encender un LED digital en el puerto 8. El controlador que envía el mensaje tiene el id 3.

\*3-2-D-W-008-0001

*	<b>Primer carácter</b> de la instrucción
3	<b>Id del controlador</b> que envía el mensaje
2	Entero que indica el <b>tipo de controlador</b> que manda el mensaje <sup>3</sup>
D	Indica si se trata de un valor <b>A</b> (nalógico) o <b>D</b> (igital).
W	Nos indica si se trata de un mensaje de lectura ( <b>R</b> ead) o de escritura ( <b>W</b> rite)
008	<b>Pin del Arduino</b> al que hace referencia el mensaje
0001	<b>Datos</b> enviados en el mensaje

Un controlador de lectura, con id 5, envía una petición de lectura del valor de un fotoresistor en el pin analógico 0. El mensaje enviado es el siguiente:

\*5-6-A-R-000-0000 *(El campo de datos en este caso no es necesario para realizar la petición, por tanto se pone a 0)*

La respuesta del Arduino a dicho mensaje (*suponiendo que el valor del fotoresistor en este momento es de 145*) será la siguiente:

\*5-6-A-R-000-0145

Los controladores de lectura, envían un mensaje cada segundo solicitando una lectura de datos de un sensor. Dando por hecho que puede haber varios controladores de lectura funcionando a la vez, y que además el usuario puede pulsar algún controlador de escritura, existe la posibilidad de que haya problemas de concurrencia a la hora de mandar mensajes. Para solucionar este problema, los controladores mandan su mensaje a un buffer, donde se almacenan, y un thread que corre en segundo plano se encarga de enviar cada uno de los mensajes en orden y procesar la respuesta antes de mandar el siguiente.

---

<sup>1</sup> Para subir código .ino a nuestro Arduino, seguir este tutorial: <https://www.arduino.cc/en/Main/Howto>

<sup>2</sup> Para llevar a cabo la comunicación serie mediante USB, se ha hecho uso de la librería [UsbSerial](#), disponible en GitHub, creada por el usuario felHR85.

<sup>3</sup> Dichos enteros están guardados como constantes en la clase de gestión de comunicación Arduino.

## Diseño

# Arquitectura

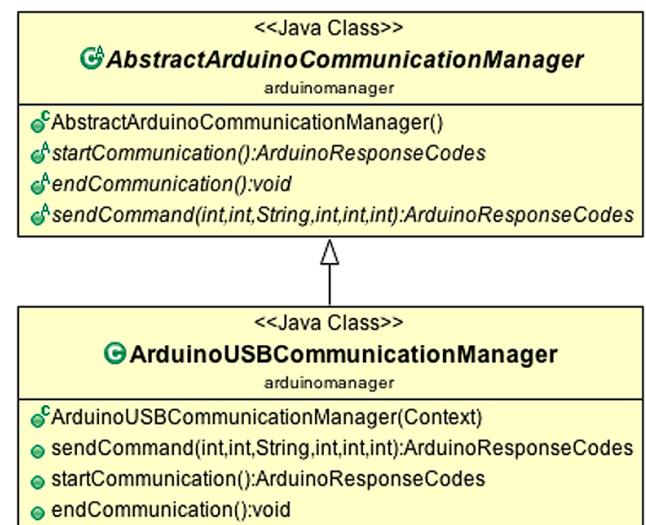
A la hora de diseñar la arquitectura del proyecto, se ha tomado en consideración, por una parte, la escalabilidad del código, y por otra, la posibilidad de un cambio de tecnología para la comunicación mediante Android y Arduino. Para poder cumplir con estas decisiones, la estructura del código está dividida en varios paquetes.

# ArduinoManager

En este paquete van incluidas todas las clases que gestionan la lógica de la conexión entre Android y Arduino.

- AbstractArduinoCommunicationManager : clase abstracta donde se definen las funciones necesarias para la capa de conexión con Arduino. Ahora mismo, la comunicación se lleva a cabo mediante USB, pero modificar la tecnología implicaría sencillamente crear una nueva implementación de esta clase.
  - ArduinoUSBCommunicationManager : implementación de la clase anterior. Se encarga de gestionar la comunicación con Arduino mediante USB.
  - ArduinoSerialConnectionListener : Interfaz que se usa a modo de listener.
  - ArduinoResponseCodes : Códigos de respuesta de Arduino.

```
classDiagram
    class AbstractArduinoCommunicationManager {
        <<Java Class>>
        <<arduinomanager>>
        <<AbstractArduinoCommunicationManager>>
        <<startCommunication()>>:ArduinoResponseCodes
        <<endCommunication()>>:void
        <<sendCommand(int,int,String,int,int,int)>>:ArduinoResponseCodes
    }
    class ArduinoUSBCommunicationManager {
        <<Java Class>>
        <<arduinomanager>>
        <<ArduinoUSBCommunicationManager>>
        <<ArduinoUSBCommunicationManager(Context)>>
        <<sendCommand(int,int,String,int,int,int)>>:ArduinoResponseCodes
        <<startCommunication()>>:ArduinoResponseCodes
        <<endCommunication()>>:void
    }
    AbstractArduinoCommunicationManager <|-- ArduinoUSBCommunicationManager
```



## Storage

Aquí se incluyen todas las clases que se encargan del almacenamiento de datos, como MySQLite o SharedPreferences.

Util

Paquete que contiene funciones y variables útiles, usadas a lo largo de todo el proyecto.

## View

Las clases que se encargan de la vista de la aplicación. Aquí se encuentran todas las views personalizadas, las activities, las vistas de los controladores... etc.

## Componentes

Componentes estudiados en clase que han sido utilizados para llevar a cabo el proyecto:

- **Activities** : Hay un total de 5 actividades en el proyecto.
  - **AboutActivity** : Actividad “Acerca de”. Contiene información de la aplicación, como el nombre de los desarrolladores, datos de contacto, y versión.
  - **PanelActivity** : Representa el contenido de un panel. Dentro se incluyen todos los controladores de dicho panel.
  - **MainActivity** : Actividad principal. Contiene una lista con todos los paneles creados por el usuario
  - **SettingsActivity** : Aquí se encuentran los ajustes que el usuario puede modificar.
  - **TutorialActivity** : Pequeño tutorial explicativo que se muestra cuando la aplicación se abre por primera vez (*también accesible desde el menú*).
- **Intents** : Comunicación entre activities mediante uso de intents.
- **Threads** : Se hace uso de threads corriendo en segundo plano en varias ocasiones. Por ejemplo, en la clase que gestiona la comunicación de Android con Arduino, disponemos de un thread, con un buffer de mensajes, que envía dichos mensajes de uno en uno de forma ordenada en segundo plano.
- **Interfaces** : Para implementar una comunicación entre distintas clases de una forma “limpia” se ha hecho uso de interfaces a modo de listeners en varias ocasiones.
- **Broadcast Receivers** : Para gestionar eventos como la conexión y desconexión de dispositivos USB.
- **Bases de datos MySQLite** : En ella se almacenan los datos de los paneles creados, así como los controladores y sus estados.
- **SharedPreferences** : Se han creado dos archivos de SharedPreferences, uno de ellos para almacenar variables usadas por la aplicación (como la comprobación de si es la primera vez que se abre la aplicación, o si el tutorial ya ha sido visto) y otro con preferencias del usuario (como por ejemplo, el tema escogido)
- **ViewPager** : Usado para crear el tutorial que se muestra por primera vez al usuario.
- **Menús contextuales** : Utilizados para mostrar las opciones que el usuario tiene para interactuar con los paneles y controladores, como editar y eliminar.

Componentes no estudiados en clase que han sido utilizados para llevar a cabo el proyecto

- **Animaciones** : Animaciones personalizadas en una gran cantidad de elementos de la interfaz de usuario.
- **VectorDrawable** : Se hace uso de VectorDrawable para todos los iconos usados en nuestra aplicación
- **AnimatedVectorDrawable** : Se trata Vector Drawables con una animación incluida en el código xml. Se ha usado con animación de path-morphing en el último ícono del tutorial.
- **Activity-alias** : Disponemos de dos activity-alias para nuestra aplicación, cada uno con un ícono diferente. De esta manera, permitimos al usuario modificar el ícono desde la actividad de ajustes.
- **GridLayout, CardView y otros layouts** : Se han utilizado layouts complejos adaptados a las necesidades de nuestra aplicación.
- **PreferenceScreen** : Layout que permite al usuario cambiar una serie de ajustes y preferencias de nuestra aplicación.
- **Styles.xml** : Uso de styles.xml para formatear la gran mayoría de los elementos del layout.
- **Modificaciones del build.gradle** : Se explicará más en detalle en el apartado de *Aspectos a destacar*.
- **Código .ino** : La programación de Arduino no es material estudiado en la materia.

## Diseño Gráfico

A continuación se explicarán las características del diseño gráfico de la aplicación más destacables.

### ➤ Ícono de la aplicación

El ícono de la aplicación se ha diseñado en Photoshop. Se han creado dos versiones del mismo, versión clara y versión oscura, permitiendo al usuario seleccionar la que mejor se adapte a sus gustos o necesidades.

### ➤ Tutorial

Se ha incluido una actividad a modo de tutorial para la primera apertura de la aplicación. En esta actividad, se hace uso de un ViewPager personalizado, con una serie de pantallas indicando los pasos necesarios para crear un primer circuito Arduino y el panel y controlador correspondiente para poder interactuar con él.

Esta actividad tiene un diseño muy cuidado, con una serie de animaciones personalizadas que se mencionan más adelante en el apartado de *Animaciones*, así como capturas de pantalla ayudando al usuario a seguir cada una de las indicaciones.

## > Views personalizadas

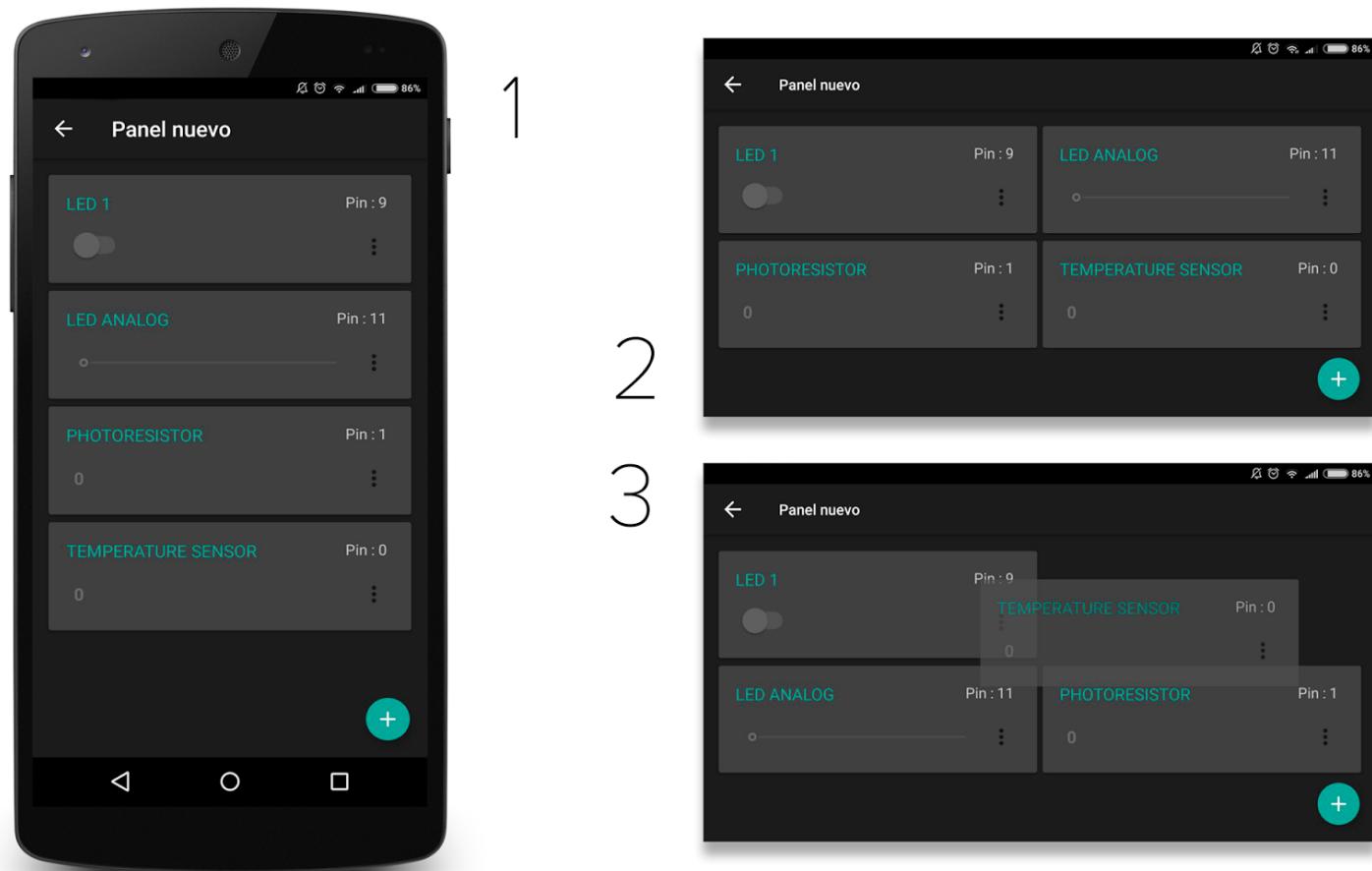
Se han creado una serie de elementos de UI personalizados, explicaremos los que más complejidad presentan:

### Drag&Drop Grid layout<sup>4</sup>

Para el layout de cada uno de los paneles se ha creado una estructura personalizada. Se trata de un grid layout modificado para permitir que el usuario pueda mantener pulsado un elemento, y moverlo de posición arrastrando el dedo por la pantalla. Dichas posiciones se almacenan en la base de datos para la próxima vez que el usuario abra la aplicación.

Cada elemento de dicho grid estará compuesto por una vista de controlador, que irá empaquetada dentro de un CardView (*layout incluido en la librería support de Android*).

Consta de dos archivos xml con disposiciones diferentes. El archivo .xml normal, presenta el grid layout con una sola columna por cada una de las filas (*imagen1*). El archivo .xml que se usa cuando la pantalla del dispositivo está en modo “landscape” le añade una segunda columna al grid layout por cada una de las filas (*imagen2*). Se puede comprobar el funcionamiento del Drag&Drop en la *imagen3*.



<sup>4</sup> Podemos encontrar ejemplos de GridLayout más sencillos pero con funcionamiento similar en el apartado de *Referencias* al final de este documento.

## **Custom Alert Dialog**

AlertDialogs personalizados para crear paneles o controladores. Permiten una implementación sencilla y rápida. Están anclados a la parte inferior de la pantalla, disponen de un diseño plano y sencillo.

Se ha diseñado de tal manera que el propio AlertDialog se encarga de gestionar sus propias animaciones y ensombrecimiento del fondo, sin necesidad de código adicional por parte del programador.

Solo están compuestos de título y dos botones. Es tarea del programador añadirle una vista a través de la función `setView(View view)`.

## **ControllerView**

Cada uno de los elementos del GridLayout mencionado anteriormente. Se trata de una clase abstracta con distintas implementaciones. Cada una de las implementaciones representa a un tipo de controlador distinto. Nos encontramos con 3 tipos de implementaciones:

- ControllerAnalogWriteView : Utilizado para los controladores de escritura de datos de tipo analógico, como servos, LEDs analógicos.. etc. El método de interacción con el usuario es un slider.
- ControllerDigitalWriteView : Utilizado para los controladores de escritura de datos de tipo digital, como por ejemplo LEDs simples. El método de interacción con el usuario es un switch.
- ControllerReadView : Utilizado para los controladores de lectura de datos, como sensores, potenciómetros.. etc. Dispone de un texto simple donde se mostrarán los datos leídos. Este tipo de controlador en concreto, tendrá un thread de fondo que mandará peticiones lectura cada segundo.

## **> Animaciones y transiciones**

### **Transiciones personalizadas entre actividades**

Al abrir y cerrar un panel, se utiliza una animación de SharedElements, usando las funciones proporcionadas por Android.

### **Animaciones del tutorial**

Para el ViewPager del tutorial, se han creado una serie de animaciones que permiten una mayor fluidez y mejor experiencia de usuario.

Dichas animaciones se han conseguido modificando el valor de *alpha* de cada uno de los elementos del tutorial a medida que se desliza desde una pantalla a otra. De la misma forma, se modifica el valor de traslación en el eje x de los elementos de la pantalla, pero con velocidades diferentes para conseguir un efecto “parallax”.

También se modifica el color de la barra inferior progresivamente, y se ocultan o muestran dinámicamente los botones de “siguiente”, “anterior” o “finalizar”.

### **Animaciones de elementos de la interfaz de usuario**

Se han incluido también animaciones en distintos elementos de la interfaz de usuario, como Floating Action Buttons.

## > Otros

### Ajustes para cambiar de tema

Desde el apartado de ajustes, se permite la opción de cambiar el tema de la aplicación. Se han incluido 4 temas creados manualmente de tal forma que la visualización de todos los elementos del UI sea correcto con cualquiera de ellos.

### Ajustes para cambiar icono la aplicación

Se le permite al usuario la opción de modificar el color del ícono de la aplicación que se mostrará en el launcher. Esto se ha conseguido haciendo uso de activity-alias e incluyendo dos íconos en la carpeta de resources de la aplicación, uno claro y otro oscuro.

### VectorDrawable para ahorrar espacio

Para disminuir el tamaño del .apk, se ha hecho uso de VectorDrawables<sup>5</sup> para los íconos, en formato .xml, en vez de íconos en formato .png.

Para conseguir esto, hay páginas que permiten descargar una serie de íconos en formato png o svg, como por ejemplo <https://material.io/icons/>.

Sin embargo, para nuestra aplicación hemos hecho uso de íconos que no fue posible encontrar en internet disponibles para descarga, por ello, se diseñaron en Photoshop y se exportaron en formato svg.

Una vez tenemos el archivo svg, en Android Studio le indicamos que queremos crear un nuevo Vector Drawable, y él se encargará de generar el archivo xml necesario.

## Reparto de trabajo

Trabajo llevado a cabo por cada uno de los componentes del proyecto.

<b>Matías Cubilla Currás</b> Planificación Documentación Funcionalidad básica de las views Almacenamiento en base de datos MySQLite. Corrección de errores Creación de diversos circuitos de Arduino de prueba Presentación	<b>Jose Luis Álvarez Quiroga</b> Comunicación Arduino-Android Vistas personalizadas (Drag&Drop GridLayout, CustomAlertDialog... etc) Animaciones personalizadas Diseño de la interfaz gráfica Diseño del logo e íconos de la aplicación Actividad de ajustes de usuario y tutorial Corrección de errores Código Arduino Creación de los circuitos personalizados para la presentación y la corrección de la práctica
---	---

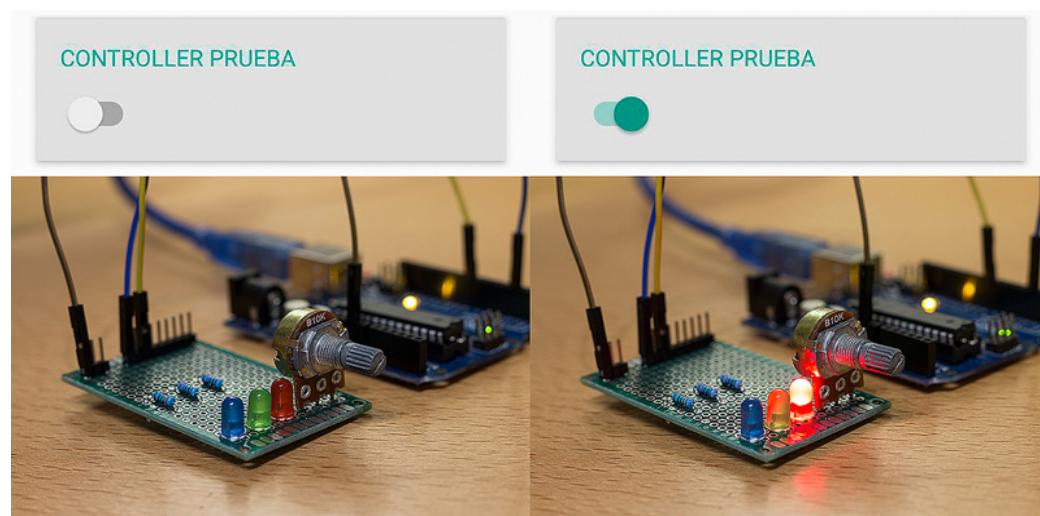
<sup>5</sup> También se ha hecho uso de AnimatedVectorDrawables, pero el procedimiento es notablemente más complejo. Se hablará más extensamente de ello en el apartado de *Aspectos a destacar*. Más info: <https://developer.android.com/reference/android/graphics/drawable/AnimatedVectorDrawable.html>

# Pruebas de funcionamiento

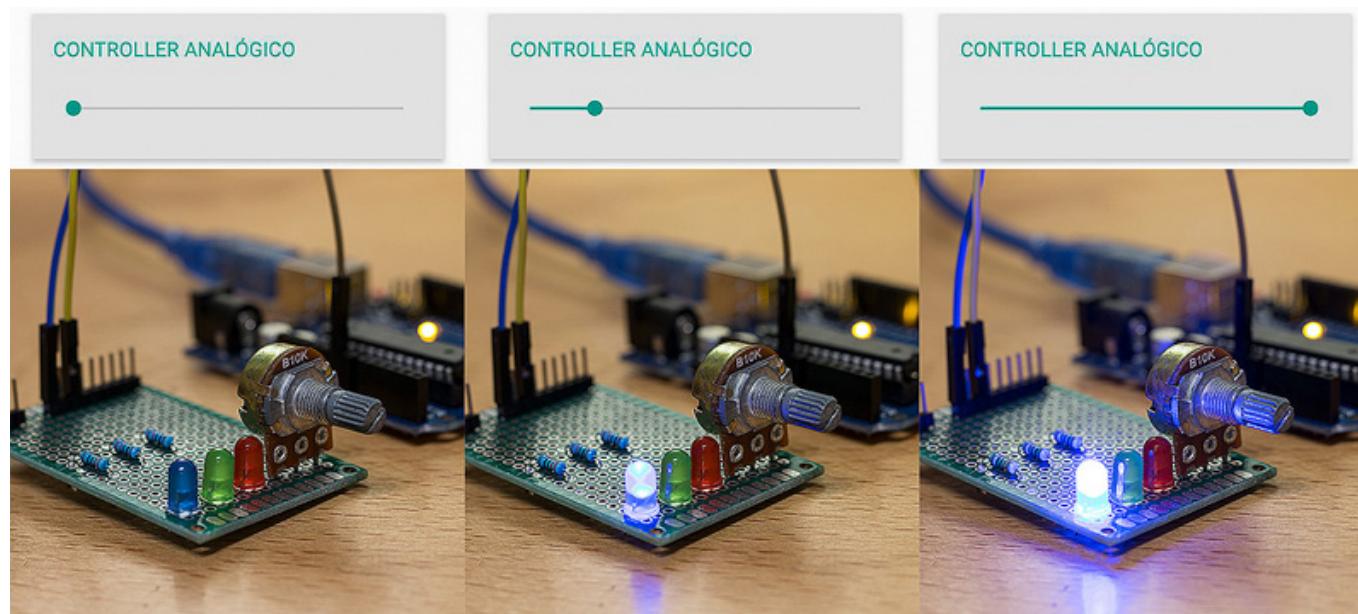
## Comunicación Android-Arduino

Las pruebas realizadas en este apartado irán enfocadas al correcto funcionamiento de la comunicación entre Android y Arduino, verificando que nuestro Arduino actúa de manera esperada en respuesta a los controladores disponibles en nuestra aplicación.

A continuación se muestra un ejemplo de un controlador de escritura digital, con un LED rojo conectado en el puerto 9 del Arduino. Al pulsar el switch, el LED se enciende correctamente.

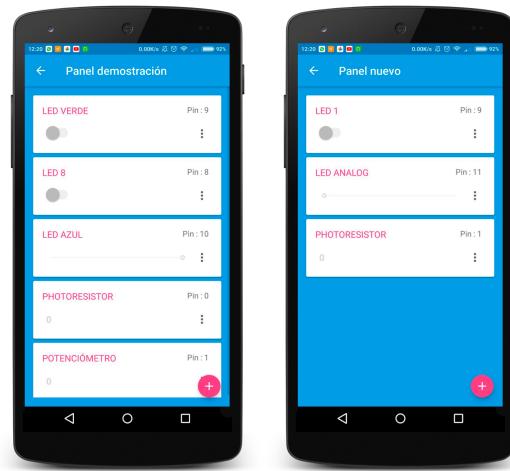


En este ejemplo probaremos un controlador de escritura analógica. Con el slider situado en la posición 0, el LED de la imagen está completamente apagado. A medida que deslizamos el slider hacia la derecha, la intensidad del LED irá aumentando progresivamente.



## Gestión de paneles y controladores

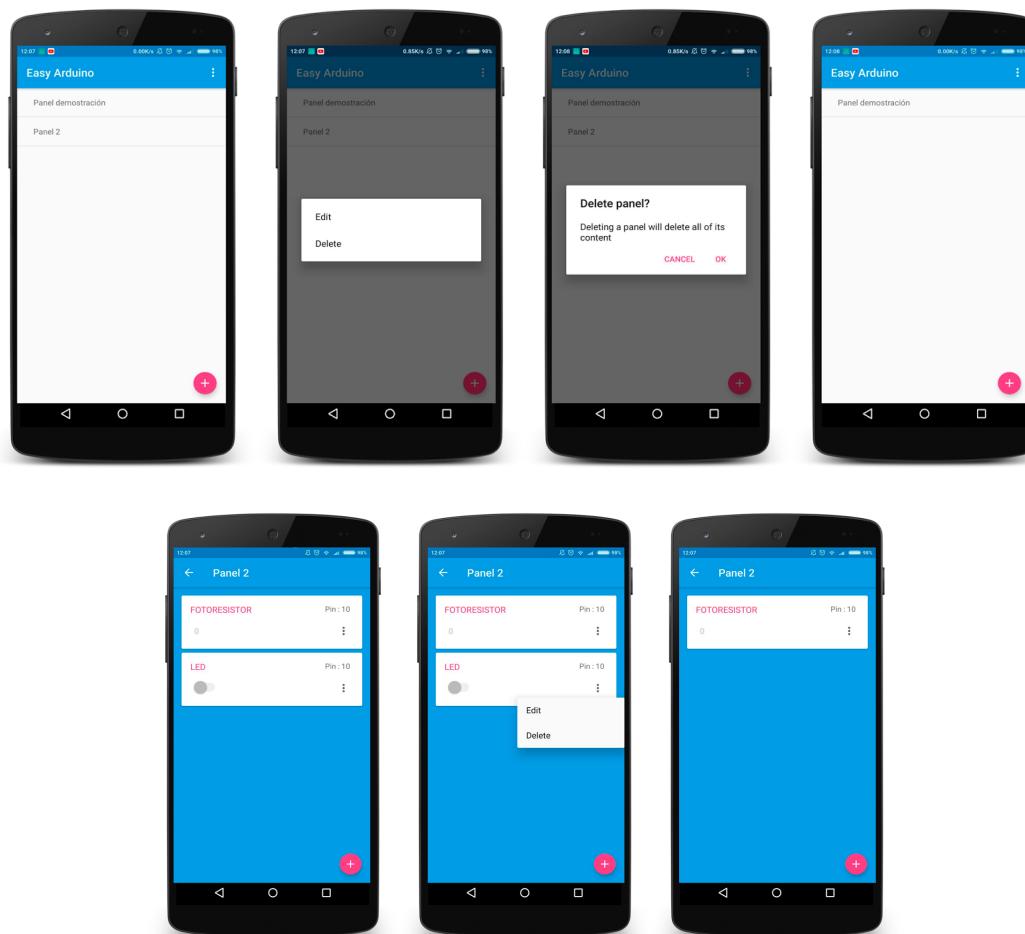
Primero creamos una serie de paneles y controladores de prueba para verificar que el funcionamiento es correcto.



Seguidamente, probamos a eliminar un controlador, observando que se elimine correctamente tanto de la base de datos como de la vista de la aplicación.

Haremos lo mismo para los paneles, pero en este caso, también han de borrarse de la base de datos los controladores que dicho panel tenga asociados.

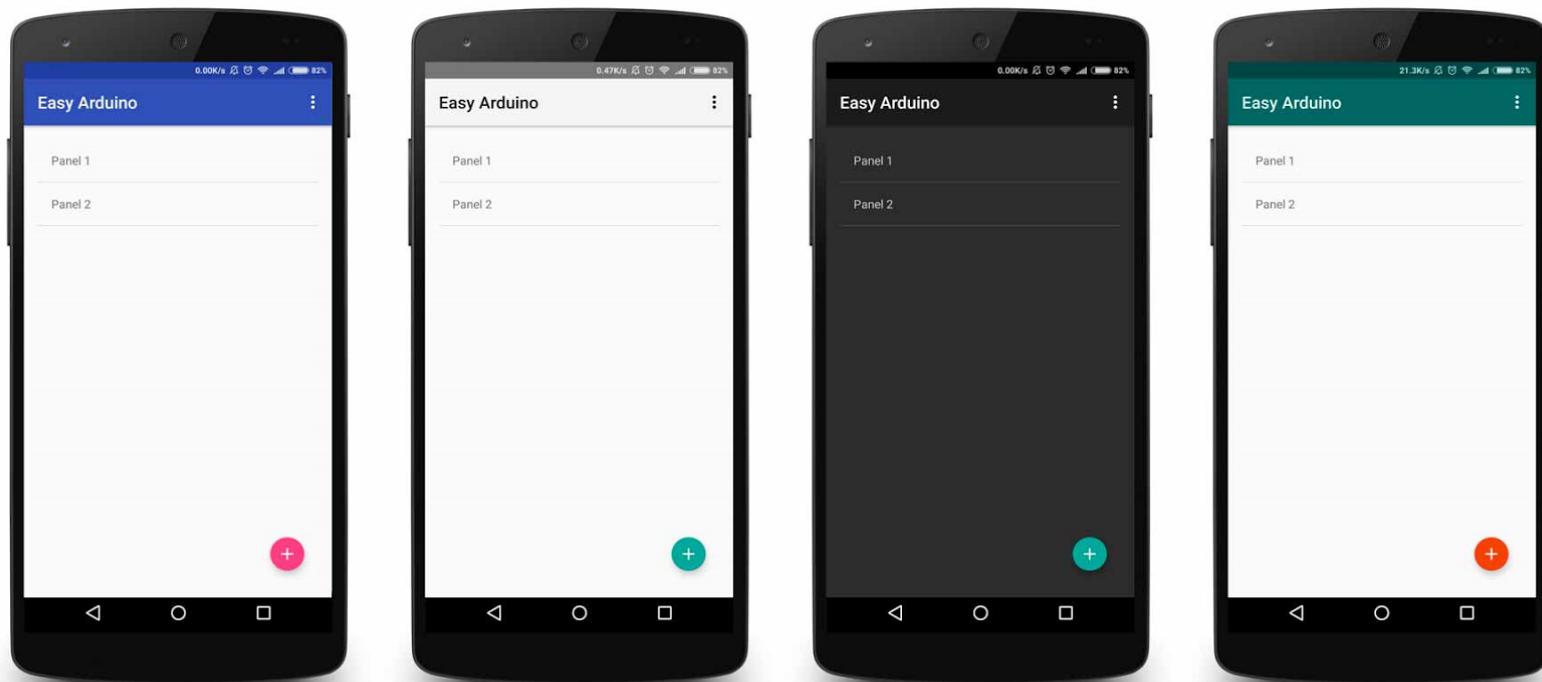
Se muestran a continuación unas capturas de pantalla mostrando el procedimiento para borrar tanto paneles como controladores mediante sendos menús contextuales:



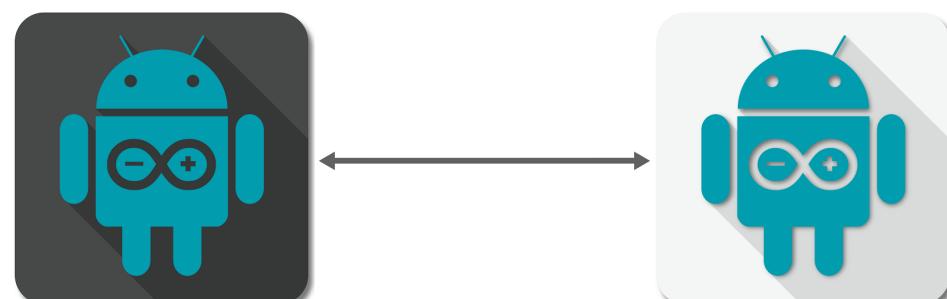
Por último, comprobamos que al cerrar la aplicación, así como al cerrar un panel, o bloquear el teléfono, los estados de los controladores, así como su posición, se mantienen al volver a abrirlos.

## Ajustes de usuario

Comprobamos que se aplica correctamente el tema de la aplicación desde el apartado de ajustes.



También comprobamos que el usuario puede cambiar el icono de la aplicación desde la actividad de ajustes.



# Aspectos a destacar

## Modificaciones build.gradle

Se ha modificado el build.gradle del proyecto, añadiendo cuatro variables numéricas, que indican el número de versión “major”, “minor”, “patch” y “build”. De esta manera se consigue mantener un numerado de versiones de manera más cómoda y visual.

Se ha modificado también el nombre con el que se genera el apk automáticamente para que incluya el nombre de la aplicación, así como el nombre de la versión en la que se encuentra. El formato elegido es el siguiente : “EasyArduino-vx.x.x-release.apk”.

Para reducir el tamaño del apk, se ha hecho uso de las herramientas proporcionadas por Android, para ofuscar código, y evitar incluir recursos innecesarios en el archivo apk final.

## Archivo Changelog

A lo largo de todo el desarrollo del proyecto, con cada nueva versión del archivo .apk, se ha ido añadiendo un pequeño resumen de funcionalidades en un archivo changelog.txt, incluido en la carpeta /res/raw del código Android.

## VectorDrawable y AnimatedVectorDrawable

Se ha hecho uso de VectorDrawables para todos los iconos de la aplicación. Esto ha supuesto un mayor grado de complejidad frente a usar iconos en formato png, pero de esta manera se consigue reducir notablemente el tamaño del archivo .apk, así como conseguir una resolución óptima para cualquier tipo de pantalla.

También se ha hecho uso de un AnimatedVectorDrawable, en el último botón del tutorial en la barra inferior. Ha llevado una gran carga de trabajo para un simple ícono, pero permite conseguir un efecto visualmente agradable de una manera muy elegante y óptima en cuanto a rendimiento.

# Trabajo futuro

- Cambiar la tecnología usada para llevar a cabo la comunicación. En vez de USB, usar por ejemplo Bluetooth.
- Ampliar la lista de componentes electrónicos disponibles a la hora de crear controladores.
- Permitir al usuario, en los controladores de lectura, modificar el intervalo con el que se envían peticiones de lectura al Arduino (actualmente es 1 segundo).
- Implementar la opción de permitir al usuario añadir autenticación a los paneles.
- Modificar el código del Arduino, para permitir que nuestra app reconozca mediante un id de qué Arduino en concreto se trata, y abrir automáticamente el panel con el que haya sido asociado previamente.
- Añadir soporte para otros tamaños de pantallas y para otros idiomas.

## Problemas conocidos

- A la hora de cambiar el icono de la aplicación desde el apartado de ajustes, tarda unos cuantos segundos en surtir efecto. No es algo que podamos controlar ya que es el launcher el que lo gestiona.
- A pesar de se indica que se pueden crear controladores de servo motores, y sensores de temperatura y de humedad, dicha funcionalidad aún no está implementada en la parte de Arduino por falta de tiempo. Se pueden crear dichos controladores igualmente, haciéndolo de forma manual con el tipo de controlador “de propósito general”.

## Referencias

**Librería que permite la comunicación serie mediante USB**

<https://github.com/felHR85/UsbSerial>

**Iconos basados en material design**

<https://material.io/icons/>

**Implementaciones simples de Grid layouts con Drag&Drop**

<https://github.com/patrick-iv/DragNDropApp>

<https://stackoverflow.com/questions/7146639/android-gridview-reorder-elements-via-drag-and-drop>

**Animated Vector Drawables**

<https://developer.android.com/reference/android/graphics/drawable/AnimatedVectorDrawable.html>

**ViewPager animations**

<https://medium.com/@BashaChris/the-android-viewpager-has-become-a-fairly-popular-component-among-android-apps-its-simple-6bca403b16d4>