



CICLO FORMATIVO DE GRADO SUPERIOR  
DESARROLLO DE APLICACIONES MULTIPLATAFORMA

PROYECTO FIN DE CICLO

---

**DRYICE**

**Autores**  
**CURSO 2020-21**

**TÍTULO :** DryIce

**AUTORES:** PABLO DÍAZ SANZ

SERGIU DANIEL TURDASAN

ALBERTO GARCÍA LÓPEZ

**TUTOR DEL PROYECTO:** Ernesto Ramiro Córdoba

**FECHA DE LECTURA:** ... Junio de 2021

En Madrid

Ernesto Ramiro Córdoba  
Tutor del PFC

## Resumen de la aplicación

Presentamos DRY ICE: Esta aplicación se basa en control ambiental de interiores. Es una medida que creemos necesaria ya que ocupamos el mayor tiempo de nuestra vida en interiores y eso ha hecho que cada vez, existan interiores mejor acondicionados y más confortables pero, la reciente llegada del virus COVID-19 ha hecho que tengamos que poner más énfasis y ser más minuciosos a la hora de saber el ambiente que tenemos ya sea en nuestro hogar, oficina, en nuestros restaurantes habituales y demás espacio social de interior.



En la lucha contra el COVID-19, está demostrado que la ventilación de espacios interiores es de vital importancia pero sabemos que no es posible tener un espacio ventilando el 100% del tiempo ya que hay causas que no podemos controlar, como por ejemplo situaciones meteorológicas adversas y esto hace que no siempre podamos tener un espacio ventilado. Pero lo que si podemos tener controlado es el nivel de CO<sub>2</sub>, humedad y temperatura de nuestro espacio. Controlando la calidad del aire y sabiendo los niveles recomendados de dióxido de carbono en nuestro ambiente, podremos controlar la ventilación de nuestro espacio.

DRY ICE es una aplicación sencilla con una interfaz muy intuitiva, pensada y desarrollada para aquellos que muestren interés en analizar la calidad del aire que les rodea.

Consiste en unos diagramas que recogen la información en tiempo real y muestra una serie de gráficas fácilmente entendibles que nos van a ayudar en cada momento a conocer la calidad del aire del lugar donde tengamos instalado el dispositivo.

Toda la información se recoge a través de un sensor y una Raspberry para que, a través de un código de programación toda esa información se inserte en una base de datos. Luego, con más código de programación transformamos esa información y la añadimos a nuestras gráficas, las cuales van a traducir esos datos en diagramas entendibles, sencillos y cómodos.

## Resumen de la aplicación ( Inglés )

This application is based on indoor environmental control. It is a measure that we believe is necessary since we spend most of our lives indoors.

The recent arrival of the COVID-19 virus has made us have to put more emphasis on knowing the environment we have, whether in our home, office, in our usual restaurants and other indoor social spaces.

Because of COVID-19, we know that we should have controlled the level of CO2, humidity and temperature of our space. Controlling air quality and knowing the recommended levels of carbon dioxide in our environment, we can control the ventilation of our space.

DRY ICE is a simple application with a very intuitive interface, designed and developed for everyone.

It consists of diagrams that collect information in real time and shows a series of easily understandable graphs that will help us at all times to know the air quality of the place where we have the device installed.

All the information is collected through a sensor and a Raspberry so that, through a programming code, all that information is inserted into a database. Then, with more programming code, we transform that information and add it to our graphs, which will translate that data into understandable, simple and comfortable diagrams for the user.

## AGRADECIMIENTOS

Todos nuestros agradecimientos van dedicados a todos y cada uno de los profesores que hemos tenido durante el ciclo formativo.

Gracias a todos y cada uno por vuestra paciencia, esmero y dedicación.

Gracias por habernos enseñado todo lo que sabemos y la base de nuestro futuro.

Todos ellos son:

- Raúl Rodríguez
- Ernesto Ramiro
- Pedro Camacho
- Juan Jose Rodriguez
- Felix de Pablo
- Jose Antonio Perez
- Teresa
- Luis Sanz



Esta obra se distribuye bajo una licencia Creative Commons.

Se permite la copia, distribución, uso y comunicación de la obra si se respetan las siguientes condiciones:

- Se debe reconocer explícitamente la autoría de la obra incluyendo esta nota y su enlace.
- La copia será literal y completa
- No se podrá hacer uso de los derechos permitidos con fines comerciales, salvo permiso expreso de los autores.

El texto precedente no es la licencia completa sino una nota orientativa de la licencia original completa(jurídicamente válida) que puede encontrarse en:

<http://creativecommons.org/licenses/by-nc-nd/3.0/deed.es>

## Tabla de contenido

Resumen de la aplicación.....	2
Resumen de la aplicación ( Inglés ) .....	3
AGRADECIMIENTOS.....	4
Introducción de la aplicación.....	7
Arquitectura de la aplicación .....	8
Ventana bienvenida ( SplashScreen ).....	8
Ventana acceso ( login ).....	9
-REGISTRO: Parte 1 (Datos personales) .....	10
-REGISTRO: Parte 2 (Configuración RPI) .....	11
Componente superior (ToolBar) .....	14
Perfil .....	14
Componente inferior (BottomNavigationBar) .....	15
Mapa .....	15
Caso de uso.....	16
Raspberry PI + Sensor .....	17
Módulo de sensores SCD30 .....	17
¿Cómo hacemos funcional la RPI + Sensor?.....	18
Paletas y tipografía .....	20
Paleta de colores .....	20
Tipografía .....	22
Logotipo.....	23
Gráficas.....	24
Gráfica de líneas (LineChart).....	24
Gráficas (Librerías y lenguaje) .....	25
BarChart:.....	25
Conclusión .....	34
BIBLIOGRAFÍA Y WEBGRAFÍA .....	35

# Introducción de la aplicación

El proposito de este proyecto de fin de ciclo es crear una aplicación que cubra una necesidad. Es por ello, que, tras haber planteado crear una aplicación híbrida con React Native y que esa propuesta no haya llegado a mas ya que, tras muchos intentos, teníamos muchos errores que no sabíamos solucionar.

Gracias a una idea de nuestro tutor Ernesto, pudimos reconducir el proyecto y llevarlo al terreno de IOT (Internet of Things), en el cual, mediante un dispositivo (Raspberry o Arduino) y un sensor (Sensirion SCD30), obtendríamos datos del CO2, temperatura y humedad del aire.

Esta idea la utilizamos para obtener esos datos y mostrarlos en una aplicación de Android, y que esos datos se mostrasen en tiempo real. También, queríamos que esa aplicación mostrase las demás raspberries que pudiesen estar registradas y ver los datos que tuviesen.

Por lo tanto, para continuar con esta idea, iniciamos una tarea de investigación. Finalmente, compramos el sensor y nos dispusimos a trabajar con ello.

Inicialmente, optamos por obtener datos con una Arduino y el sensor. Las librerías de Arduino tenían ejemplos, pero como necesitábamos la librería de Firebase, y no conseguíamos que nos subiese los datos, pasamos a probar con una Raspberry.

En la Raspberry, creamos un programa de python para ejecutarlo y así descubrimos que con las librerías necesarias instaladas conseguimos los datos a nuestro gusto. Tras ello, nos marcamos los siguientes objetivos:

- Crear dos programas de Python, uno para el registro y otro para guardar los datos y recoger esos datos desde el sensor.
- Crear una caja en la cual guardar la Raspberry y el sensor.
- Crear una base de datos a tiempo real como es Firebase (Realtime Database), en la cual también utilizaremos los servicios de guardar imágenes y de Autenticacion para la aplicación.
- Crear una aplicación la cual muestre esos datos de Firebase

En la aplicación mostrar un mapa que muestre los distintos dispositivos registrados y sus datos.

Para concluir la introducción, pasaremos a explicar el desarrollo de la aplicacion



# Arquitectura de la aplicación

## Ventana bienvenida ( SplashScreen )

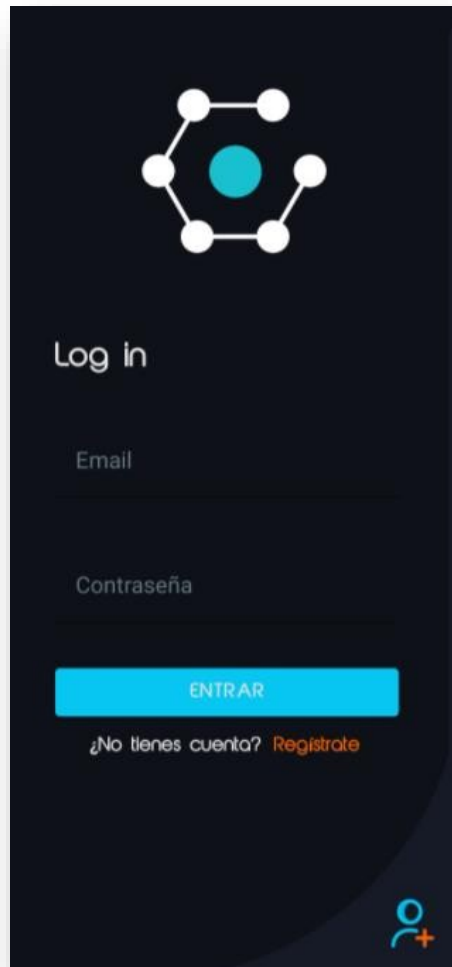
La primera pantalla que vamos a ver cuando arranquemos la aplicación es el Splash Screen. Una animación sencilla con el logotipo y el nombre de la aplicación y un slogan motivador con el objetivo de transmitir al usuario la esencia y propósito de DRYICE:



Queríamos una ventana de inicio breve ya que, un Splash demasiado largo puede resultar aburrido para el usuario.

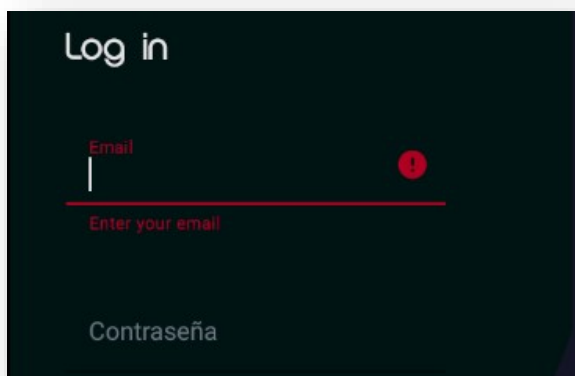
## Ventana acceso ( login )

Cuando este finaliza, encontramos la pantalla de Login la cual es completamente funcional y sirve para acceder a la aplicación mediante los credenciales personales creados en la sección "registro" que explicaremos más adelante.

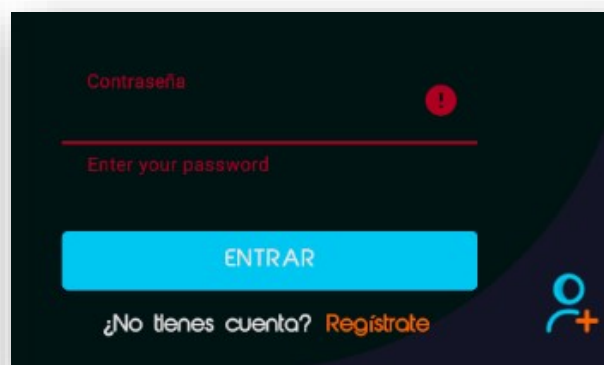


The screenshot shows a dark-themed login interface. At the top is a logo consisting of a teal circle surrounded by six white dots connected by lines. Below the logo is the text "Log in". There are two input fields: "Email" and "Contraseña". Below these fields is a teal button labeled "ENTRAR". Under the button is a link that says "¿No tienes cuenta? [Regístrate](#)". In the bottom right corner, there is a small icon of a person with a plus sign.

Todos los usuarios de DryIce deberán darse de alta a través de la ventana de registro, de no ser así no podrán acceder a la aplicación ya que hemos implementado un riguroso control de credenciales que tiene conexión directa con nuestra base de datos.



This screenshot shows the login screen with an error in the email field. The "Email" label and the "Enter your email" placeholder are in red. A red error icon (an exclamation mark inside a circle) is visible to the right of the input field. The "Contraseña" field and the "ENTRAR" button are still visible below.



This screenshot shows the login screen with an error in the password field. The "Contraseña" label and the "Enter your password" placeholder are in red. A red error icon (an exclamation mark inside a circle) is visible to the right of the input field. The "ENTRAR" button and the registration link are still visible below.

El apartado de registro se compone de dos ventanas principalmente que se transmite a un registro en dos pasos.


El primero de ellos consiste en una ventana en la que tendremos que introducir nuestros datos personales y de la cuenta.

Por ultimo el segundo paso consistirá en introducir los datos de la raspberry que se desea utilizar como plataforma de medida con sensor integrado para poder posteriormente geolocalizarnos y tomar mediciones mas precisas en el mapa.

## -REGISTRO: Parte 1 (Datos personales)

En la parte superior de la ventana deberemos pinchar en la imagen circular que nos redirigirá a la galleria de nuestro dispositivo con el objetivo de seleccionar una foto para nuestro perfil ( si es la primera vez que lo hacemos nos pedirá el permiso para el acceso a nuestra galleria ).

Después lo único que tendremos que hacer será rellenar los campos con información válida y posteriormente continuar con el siguiente paso:



Seleccione foto de perfil


Nombre completo

Email

Contraseña

Repetir contraseña

SIGUIENTE PASO



Seleccione foto de perfil

Nombre completo  
EquipoDryice

Email  
EquipoDryice

Contraseña  
.....

Repetir contraseña  
.....

SIGUIENTE PASO

## -REGISTRO: Parte 2 (Configuración RPi)

La segunda y última parte del registro consiste en la configuración de el dispositivo RPi junto con la activación de la ubicación.

Tendremos que introducir un ID RPi junto a una contraseña, ambos campos deben ser válidos, de no ser así la aplicación mandará un aviso de error que informará al usuario sobre la validez de la información introducida.

Al pulsar en "ACTIVAR Y FINALIZAR" la aplicación realizará tres tareas simultaneas que requieren de de ser 100% exitosas para finalizar el registro. Primero se comprobará la validez de la información introducida, si esta es válida pasará al siguiente paso que será pedirnos activar la localización de nuestro dispositivo ( para poder tomar las coordenadas exactas de nuestro dispositivo ), finalmente si todo lo mencionado anteriormente resulta ser exitoso entonces la aplicación almacenará todos los datos de la "Parte 1" y la " Parte 2" en la base de datos firebase asociados a un unico perfil que no se podrá duplicar.



Configuración RPi

ID de RPi

Contraseña RPi

\* Es necesario activar la localización para poder geolocalizar tu RPi. No olvides estar cerca de ella en el momento de la activación para una mayor precisión.

ACTIVAR Y FINALIZAR



Configuración RPi

ID de RPi  
RaspberryDAM

Contraseña RPi  
Rasppass

\* Es necesario activar la localización para poder geolocalizar tu RPi. No olvides estar cerca de ella en el momento de la activación para una mayor precisión.

ACTIVAR Y FINALIZAR

Una vez que hayamos accedido a la aplicación mediante el login lo primero que encontramos es la ventana principal que esta compuesta por 2 "Fragments" a los cuales se puede acceder con un sencillo movimiento de scroll hacia arriba o hacia abajo. Ambas ventanas pertenecen a una misma clase de java "ActivityGraphics" pero estan programadas de forma independiente dentro de las clases de los fragmentos "PageFragment1" y "PageFragment2"

## "PageFragment1"

Esta ventana que estamos viendo es la más importante y la que mayor refleja el proposito de DryIce.

Tenemos delante una ventana con un diseño simple, poco cargado, con elementos de gran tamaño cuya finalidad es mostrar la información buscada por el usuario de una forma directa y precisa.

Podemos distinguir en primer plano el elemento protagonista de la pantalla que es un "Progress Bar" que muestra la cantidad de CO2 existente en el aire. El circulo se rellena y va cambiando de color entre 3 diferentes según la cantidad de co2 medido en ppm, comienza desde el 0 que corresponderia al azul claro (sin riesgo ), pasaria por el azul mas oscuro donde el riesgo aumentaria (riesgo) y terminaría en el naranja mas intenso donde el riesgo sería muy alto (alto riesgo ), la cifra tope es 1850 ppm CO2.

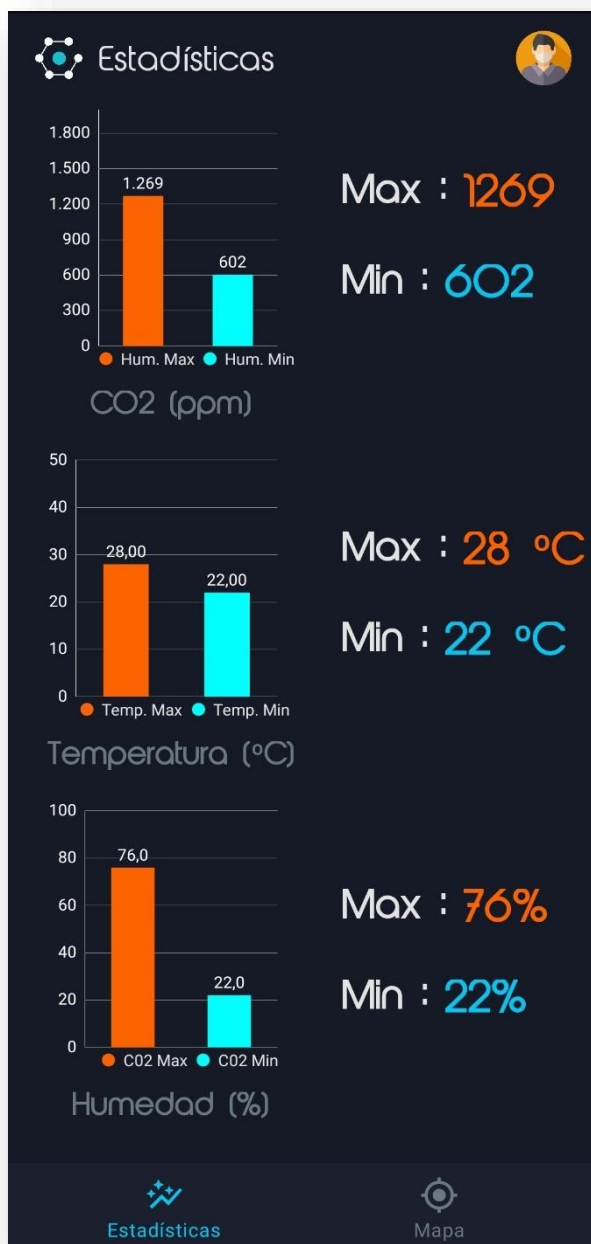
Un poco mas abajo podemos distinguir un "Cardview" de gran tamaño en el que se refleja la Temperatura y la Humedad con sus respectivas medidas.

Todas las mediciones que se muestran son a tiempo real actualizables cada 4s.



## “PageFragment2”

Para acceder a la siguiente pantalla “PageFragment2” basta con hacer un desplazamiento suave con el dedo hacia arriba y para volver a “PageFragment1” habría que hacer el movimiento opuesto.



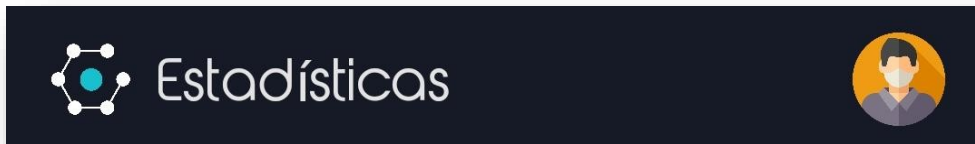
Lo primero que apreciamos es que estamos ante una ventana con información mas detallada y efecto así es. La anterior ventana tenia la finalidad de mostrar la información a tiempo real a diferencia de esta cuya finalidad es mostrar las unidades máximas y mínimas de cada una de las magnitudes que se miden en un periodo de tiempo determinado (1h).

Al representar una grafica de barras como la nuestra el usuario puede ver, analizar e interpretar la cantidd exacta y precisa entre el valor máximo y minimo de tal forma que no solo reciba información mediante valores sino que también de forma visual viendo la diferencia de tamaños entre ambas barras.

Tambien ayudamos a la percepción visual gracias a los colores empleados, el azul claro asociado a las minimas y el naranja intenso a las máximas, tanto el color de las barras como el del texto que muestra los valores.

## Componente superior (ToolBar)

Una vez hecho el acceso a dentro de la aplicación podremos encontrar en la parte superior una barra cuya finalidad es informar de la sección en la que estamos ( Estadísticas o Mapa ) y dar acceso directo al perfil del usuario mediante la imagen de nuestro avatar ubicada en la parte superior derecha.



## Perfil

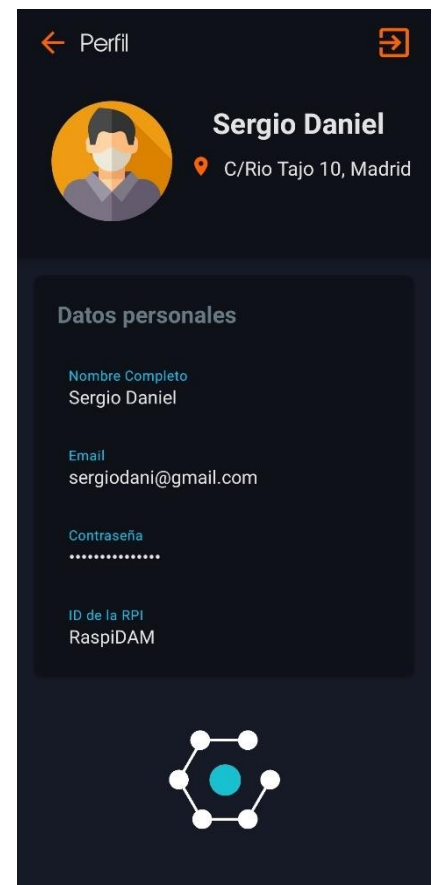
Al acceder al perfil se nos abrirá una nueva ventana mediante una animación lateral.

Esta es la ventana de Perfil, una ventana sencilla que muestra información sobre el usuario y su cuenta. En la parte superior se muestra su imagen de perfil junto a su nombre y la ubicación en la que se encuentra.

En la parte inferior se muestran sus datos personales como su nombre completo, su email y su contraseña. (En un futuro próximo se implementará la opción de editar dichos datos, de momento solo es una ventana informativa )

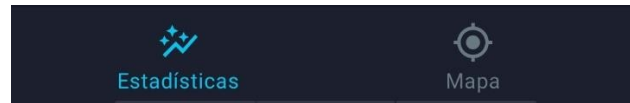
En la barra superior el usuario encontrará dos botones, en la izquierda una flecha para retroceder a la ventana anterior y en la derecha un icono que nos permite cerrar sesión y Volver directamente a la ventana de "login".

Por último en la parte mas inferior nos encontramos con un logo de DryIce de un tamaño "grande" que recordará al usuario nuestro logo de marca de una forma visual.



## Componente inferior (BottomNavigationBar)

En cuanto a la navegación dentro de la app hemos hecho uso de una barra inferior( BottomNavigationBar ) que nos permitirá el acceso a las diferentes ventanas dentro de la aplicación. El color azul accentúa la ventana en la que nos encontramos mientras que el gris apagado es utilizado para el resto de los iconos querepresentan las ventanas a las que podemos acceder.



## Mapa

Esta es la ventana de “Mapa”, corresponde a lo que es la segunda funcionalidad importante de DryIce.

La ventana es muy simple, se compone de un mapa (google Maps ) que ocupa la totalidad de la pantalla que va ser el encargado de mostrarnos la ubicación de nuestro dispositivo en tiempo real con un marcador que al pulsarlo nos mostrará los datos que recoge nuestra RPi.

En la parte inferior derecha hemos diseñado un componente “botón flotante” que al pulsarlo realizará el proceso de identificar nuestras coordenadas y actualizar la la ubicación en la base de datos.

Para el correcto y coherente funcionamiento de esta implementación es necesario estar lo mas próximo posible de nuestra RPi ya que la app detecta la ubicación de nuestro dispositivo no de la RPi.

En un future trataremos de actualizar y mejorar esta funcionalidad

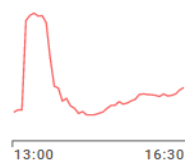




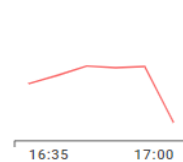
## Caso de uso

DRYICE	Control ambiental	
Versión	1.0 (01/06/2021)	
Precondición	El usuario ha instalado el sensor para posteriormente, descargar la app de Google Store, instalarla en su dispositivo y registrarse.	
Descripción	El sistema deberá comportarse como los puntos indicados a continuación:	
Secuencia normal	Paso	Acción
	1	El usuario instala la placa Raspberry + Sensor CO2 en un lugar específico de interior.
	2	Descarga e instala la aplicación DRY ICE
	3	Completa el registro en la aplicación
	4	Observa las gráficas y ve la calidad ambiental del espacio de interior.
	5	El usuario decide tomar medidas ya que ha entendido perfectamente la información de las gráficas.
Postcondición	El usuario decide compartir la aplicación a través de sus RRSS para dar mayor difusión	
Excepciones	Paso	Acción
	1	El sensor no recoge bien la información.
		No se puede completar el caso de uso.
		Se tramita la devolución del sensor por otro nuevo.
	2	El usuario no completa el registro
		El usuario se registra en la aplicación
		Continúa el caso de uso
	3	La aplicación DRY ICE no se encuentra disponible
		No se puede completar el caso de uso.
		El usuario puede ver la información del aire a través de la BBDD.
		El usuario finaliza conociendo el estado del aire en su espacio.

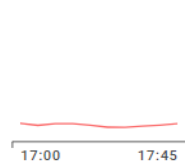
**Casa**



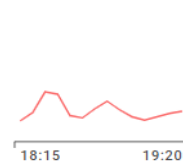
**Coche**



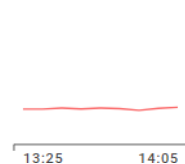
**Universidad**



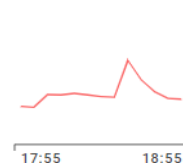
**Metro**



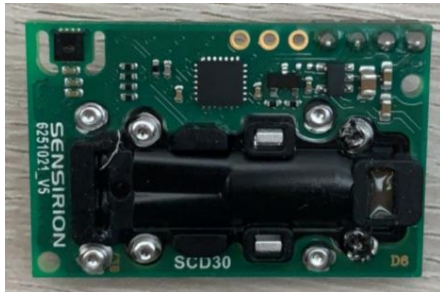
**Oficina**



**Médico**



## Raspberry PI + Sensor



### Módulo de sensores SCD30

El módulo de sensores SCD30 de Sensirion utiliza la tecnología de sensores de CO<sub>2</sub> NDIR para detectar CO<sub>2</sub> y cuenta con un sensor de temperatura y humedad integrado. La temperatura y la humedad ambiental se pueden medir mediante el control y compensación de las fuentes externas de calor sin necesidad de utilizar otros componentes. La altura reducida del módulo permite una integración sencilla en diferentes aplicaciones. El SCD30 ofrece detección de dos canales para obtener una estabilidad superior y una precisión de  $\pm 30 \text{ ppm} + 3 \%$

<https://www.mouser.es/>



La **Raspberry Pi** es una serie de ordenadores de placa reducida, ordenadores de placa única u ordenadores de placa simple (SBC) de bajo coste desarrollado en el Reino Unido por la Raspberry Pi Foundation, con el objetivo de poner en manos de las personas de todo el mundo el poder de la informática y la creación digital.<sup>3</sup> Si bien el modelo original buscaba la promoción de la enseñanza de informática en las escuelas,<sup>4</sup> este acabó siendo más popular de lo que se esperaba<sup>7</sup>, hasta incluso vendiéndose fuera del mercado objetivo para usos como robótica

[https://es.wikipedia.org/wiki/Raspberry\\_Pi](https://es.wikipedia.org/wiki/Raspberry_Pi)

## ¿Cómo hacemos funcional la RPI + Sensor?

En la RPI2 vamos a conectar un sensor medidor de CO2, temperatura y humedad para así vamos a leer los datos y subirlos a la RealtimeDatabase de Firebase para posteriormente leerlos en nuestra aplicación DRY ICE

Para ello, necesitaremos lo siguiente:

- Raspberry Pi 3B+ o Raspberry Pi 4B
- Sensor SENSIRION SCD30
- 4 Cables de Pines Hembra-Hembra
- Soldador y conectores para soldar los pines del sensor
- Perifericos para manejar Raspberry Pi

Primero de todo, descargaremos el sistema operativo Raspbian y lo instalaremos en la RPi. Para ello, hemos seguido el siguiente tutorial: <https://ubunlog.com/como-instalar-raspbian-en-una-raspberry-pi-4-y-disfrutar-de-un-centro-multimedia-y-mucho-mas/>

Tras ello, soldaremos los conectores en el sensor. Será necesario soldar los siguientes pines:

- VIN
- GND
- LX/SCL
- RX/SDA

Tras haberlos soldado, los conectaremos a los pines de la RPi con los cables Hembra-Hembra de esta manera: (En la RPi, tiene forma de L las conexiones. Si no sabes donde conectarlo, busca Raspberry GPIO y aparecerán varias imagenes en las que fijarte dónde conectarlo)

- VIN --> 3V3 power
- GND --> Ground
- LX/SCL --> GPIO 3 (SCL)
- RX/SDA --> GPIO 2 (SDA)

Cuando hayamos conectado los pines, abriremos una terminal y escribiremos lo siguiente, linea a linea.

- `sudo raspi-config` (Te saldrá una interfaz de configuraciones, pincha en "3 Interface Options" - "P5 I2C" - "Yes" - "Ok")
- `sudo reboot`

Cuando haya reiniciado la RPi, volvemos a abrir un terminal y hacemos lo siguiente:

- `ls /dev/i2c*` (Para comprobar que está activado el I2C, para los pines)
- `sudo pip3 install adafruit_blinka` (Para instalar los paquetes de la librería de Adafruit)
- `sudo pip3 install adafruit-circuitpython-scd30` (Para instalar la librería específica del sensor)
- `sudo pip3 install pyrebase` (Para instalar la librería de Python de Firebase)

Cuando ha salido todo bien, continuamos con los siguientes comandos:

- `mkdir dryice-workspace` (Para guardar los datos del repositorio que vamos a clonar, elegir donde guardar, no tiene por qué ser en la raíz de la RPi, puede ser en el lugar que queráis)
- `cd dryice-workspace/` (Para meternos en la carpeta)
- `git clone https://github.com/2DAMUE/pfcjun21-dryice-rpi.git` (Clonamos el repositorio de Git)
- `python3 RPiRegister.py` (Para registrar nuestra RPi para subir datos, puedes registrarte más de una vez)
- `python3 SCD30RPi.py` (Para comenzar a leer datos del sensor)

Es importante que antes de ejecutar el programa `SCD30RPi.py`, ejecutemos el programa `RPiRegister.py`, porque sino no empezará a leer datos del sensor si no te identificas.

**Programa SCD30RPi.py** (programa realiza una recogida de datos de un sensor SCD30, medidor de CO<sub>2</sub>, temperatura y humedad)

<https://github.com/2DAMUE/pfcjun21-dryice-rpi/blob/main/SCD30RPi.py>

**Programa RPiRegister.py** (El programa hace un registro de la raspberry para saber a qué usuario hay que darle los datos que obtengamos del programa `SCD30RPi.py`)

A partir de ahí la aplicación ya es completamente funcional.

# Paletas y tipografía

## Paleta de colores

La paleta de colores que hemos utilizado para la aplicación consta de cinco colores base mas otros dos que hemos utilizado para dar color a los textos y a las letras.

Hemos optado en este caso por un tipo de aplicación con fondos oscuros y sobre los cuales unos textos de color blanco y gris claro. De esta forma se conseguirá un aspecto mas moderno y unos textos mas legibles.

Los colores oscuros que hemos utilizado para los fondos principals y para los fondos de los elementos como "cardview" y "LinearLayouts" partiendo desde un azul muy oscuro, prácticamente negro, y aumentando la suavidad del color progresivamente para elementos cada vez mas superficiales, simulando distintos tipos de profundidad.

Por otro lado tenemos los colores de acento, un azul claro pero intenso y un naranja bastante intenso.

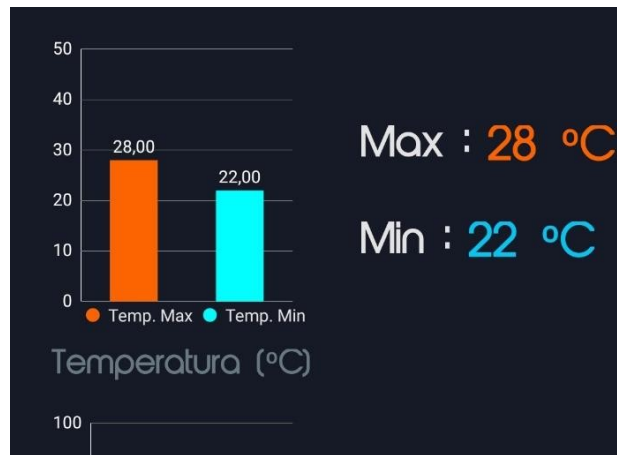
Ambos colores tienen muchísimo protagonismo dentro de la aplicación ya que los utilizamos para múltiples elementos como pueden ser las gráficas, textos, valores... ,hemos aprovechado el efecto visual que estos colores tienen para expresar valores máximos y mínimos ya que el naranja inspira algo saturado algo alterado y el azul es un color frío que expresa menos intensidad.

También hemos utilizado los colores accentuados para acciones dentro de la aplicación tales como la navegación, flecha de retroceder, cerrar sesión..

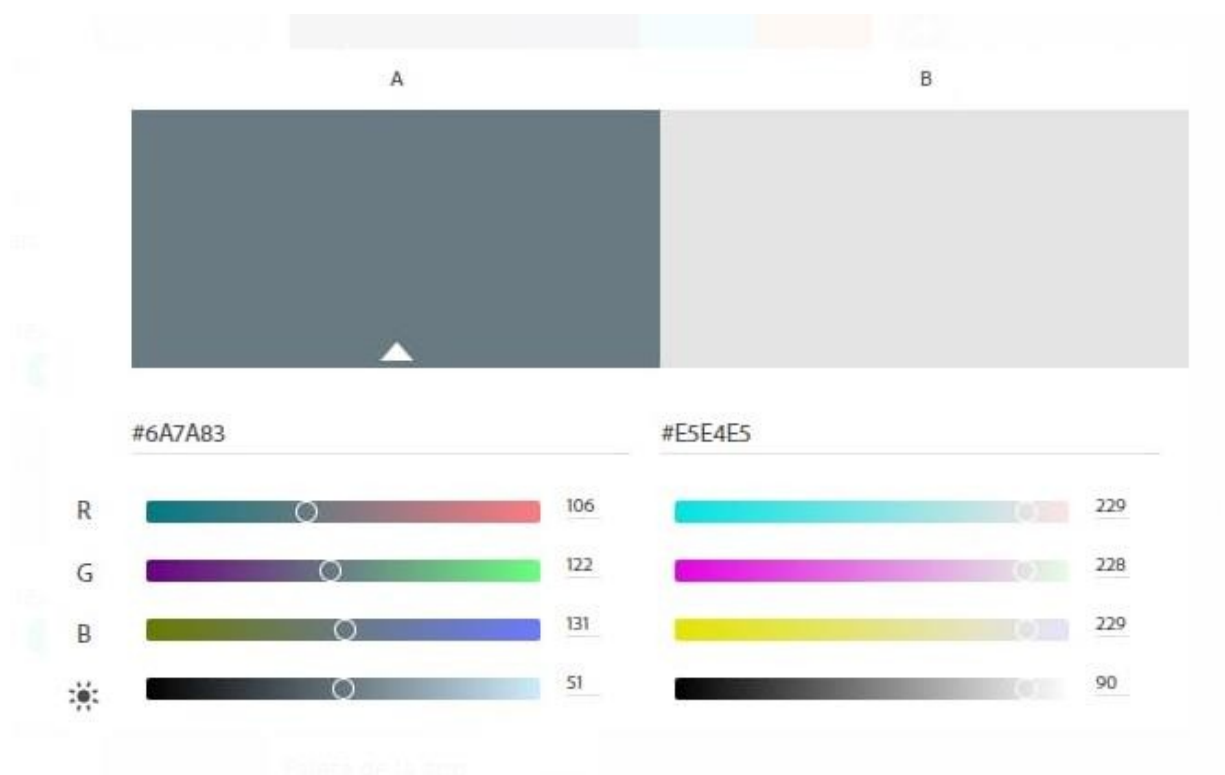


Pensamos que haber utilizado esta paleta de colores ha sido todo un acierto pese a que crear una aplicación entera en un tema oscuro conlleva sus riesgos y una dificultad añadida.

El resultado es muy bueno y permite al usuario enfocar su atención en los elementos más importantes, por otro lado los píxeles más oscuros permiten ahorrar batería del dispositivo y sobretodo cargar mucho menos la vista del usuario.



En la imagen superior Podemos apreciar como gracias al fondo oscuro las graficas y los textos cobran mucho protagonismo que se traduce a un mayor enfoque del usuario a los elementos que tiene la ventana.



Esta imagen superior representa los colores utilizados para los textos planos de la aplicación cuya función es de título y de descripción.

Hemos escogido estos colores porque nuestro fondo es oscuro por lo tanto el color de letra debía ser claro para que los textos sean perfectamente legibles.

## Tipografía

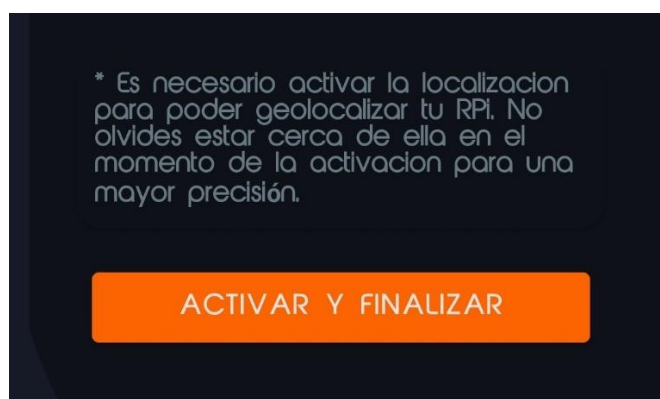
La tipografía es A beat By Kay: <https://www.dafont.com/es/abeatbykai.font>



La tipografía que hemos escogido se llama “AbeatbyKay”, en un principio esta tipografía la escogimos únicamente para el logotipo pero decidimos probar como quedaria si la implementabamos en toda la aplicación.

La tipografía resulta encajar con nuestros objetivos de diseño y nos gusto bastante.

Es una tipografía moderna, con bordes redondeados que da carácter unico a DryIce. Es una apuesta atrevida para sin duda diferente a la de la mayoría de apps.



## Logotipo

El diseño del logotipo es una mezcla de líneas y círculos, queriendo llegar a la mezcla entre figuras rectas y circulares para representar la composición del CO<sub>2</sub>, que es la unión entre moléculas de carbono con las de oxígeno. Los círculos de nuestro logo representan las moléculas y las líneas rectas su unión.

La traducción al español de "Dry Ice" es "Hielo Seco" que es el nombre que recibe el dióxido de carbono en estado sólido.

Recibe este nombre, porque, pese a parecerse al hielo normal o a la nieve por su aspecto y temperatura, cuando se sublima no deja residuo de humedad porque su base no es agua y su estado natural es gaseoso. Incluso a temperaturas ambientales bajas, tiene una temperatura de sublimación de  $-78,5\text{ }^{\circ}\text{C}$  (a una atmósfera de presión).





## Gráficas

La aplicación tiene 7 gráficas para que el usuario tenga la mayor información posible de la manera más ordenada y limpia.

Gráfica de barras (BarChart)



## Gráfica de líneas (LineChart)

Gráfica de pastel (PieChart)

Al tratar con información sensible como es la cantidad de CO2 es un área, para este diagrama que además, es el primero que aparece dentro del Fragment, hemos querido destacar el CO2 sobre la temperatura y humedad, avisando también del riesgo del área.



# Gráficas (Librerías y lenguaje)

## BarChart:

Hemos utilizado la librería MPChart para Android Studio:

<https://github.com/PhilJay/MPAndroidChart>

Es una librería muy extensa en la cual hay varios tipos de gráficas y ya es el propio programador el que elige cual implementar y adaptarla a su programa.

### Gradle Setup

```
repositories {  
    maven { url 'https://jitpack.io' }  
}  
  
dependencies {  
    implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'  
}
```

### Maven Setup

```
<!-- <repositories> section of pom.xml -->  
<repository>  
    <id>jitpack.io</id>  
    <url>https://jitpack.io</url>  
</repository>  
  
<!-- <dependencies> section of pom.xml -->  
<dependency>  
    <groupId>com.github.PhilJay</groupId>  
    <artifactId>MPAndroidChart</artifactId>  
    <version>v3.1.0</version>  
</dependency>
```

Al crear las dependencias y los repositorios, Android Studio detecta la librería y puedes empezar a crear las gráficas

## JAVA:

```
<com.github.mikephil.charting.charts.BarChart
    android:id="@+id/barChart3"
    android:layout_width="match_parent"
    android:layout_height="180dp"
    android:layout_alignParentTop="true"
    android:layout_marginTop="10dp"
    android:layout_marginLeft="20dp"/>
```



En la vista de diseño, no se muestra la gráfica ya que no hay datos para la gráfica disponibles. Para hacerla funcionar y que tenga un significado, es necesario emplear código de programación:

```
import com.github.mikephil.charting.charts.BarChart;
import com.github.mikephil.charting.charts.Chart;
import com.github.mikephil.charting.components.Legend;
import com.github.mikephil.charting.components.LegendEntry;
import com.github.mikephil.charting.components.XAxis;
import com.github.mikephil.charting.components.YAxis;
import com.github.mikephil.charting.data.BarData;
import com.github.mikephil.charting.data.BarDataSet;
import com.github.mikephil.charting.data.BarEntry;
import com.github.mikephil.charting.data.DataSet;
import com.github.mikephil.charting.formatter.IndexAxisValueFormatter;
```

Al importar la librería entera, ya puedes crear objetos Chart, darles una funcionalidad y hacerlas dinámicas a tu manera.

Creamos los objetos BarChart para el diagrama de barras.

```
private BarChart barChart;
private BarChart barChart2;
private BarChart barChart3;
```

Les localizamos por ID dentro de una vista (ya que estamos extendiendo de Fragment) y ya podemos empezar a utilizar métodos para darles funcionalidad.

```
barChart = (BarChart) rootView.findViewById(R.id.barChart);
barChart2 = (BarChart) rootView.findViewById(R.id.barChart2);
barChart3 = (BarChart) rootView.findViewById(R.id.barChart3);
```

Ahora estamos viendo el diagrama de barras y el código es un diferente al del resto de diagramas. Vamos a desgranar el código y a mostrarlo.

Después de los métodos listados a continuación, todo va a empezar a tener forma y el resultado va a ser el siguiente:

```
private Chart getChart(Chart chart, String description,
    int background, int animateY) {
    chart.getDescription().setText(description);
    chart.getDescription().setTextSize(15);
    chart.setBackgroundColor(background);
    chart.animateY(animateY);
    legend(chart);
    return chart;
}
```

```
private void legend(Chart chart) {
    Legend legend = chart.getLegend();
    legend.setForm(Legend.LegendForm.CIRCLE);
    legend.setTextColor(getResources()
        .getColor(R.color.claroLetras));

    legend.setHorizontalAlignment();
    ArrayList<LegendEntry> entries = new ArrayList<>();
    for (int i = 0; i < datosRasp.length; i++) {
        LegendEntry entry = new LegendEntry();
        entry.formColor = datosColores[i];
        entry.label = datos[i];
        entries.add(entry);
    }
    legend.setCustom(entries);
}
```

Este primer método devuelve un objeto del tipo Chart en el cual previamente, estamos añadiendo la gráfica una descripción, un texto, un fondo y una animación. En este caso, al mostrarse por primera vez la gráfica, las barras se elevan hasta llegar al parámetro de temperatura (número de tipo Double).

Este método crea la legenda (gracias a la librería MP Chart). Aquí simplemente creamos la leyenda, y asignamos colores que los tenemos creados dentro de una lista. Por lo que, por cada dato que tenemos en un ArrayList, tenemos un color en otro ArrayList.

Podemos jugar con formas, colores, tamaños etcétera:



Toda gráfica tiene un punto X y un punto Y (en este caso no necesitamos ningún punto Z) por lo que, hay que escribir los métodos que identifiquen ambos puntos y muestren la información de dichos puntos así como la apariencia de los puntos X e Y de la leyenda previamente creada:

```

private void axisX(XAxis axis) {
    axis.setGranularityEnabled(true);
    axis.setPosition(XAxis.XAxisPosition.BOTTOM);
    axis.setValueFormatter(new IndexAxisValueFormatter(datos));
    axis.setEnabled(false);
}

private void axisLeft(YAxis axis) {
    axis.setSpaceTop(30);
    axis.setAxisMinimum(0);
}

private void axisRight(YAxis axis) {
    axis.setEnabled(false);
}

```

Importante: Cuando utilizamos el método `setValueFormatter` y creamos el objeto "IndexAxisValueFormatter" el argumento que recibe el objeto viene de un Array de Strings creado previamente, donde ponemos los datos de la información del punto X.

```

private String[] datos = new String[]{"C02 Max", "C02 Min"};
private String[] datos2 = new String[]{"Temp. Max", "Temp. Min"};
private String[] datos3 = new String[]{"Hum. Max", "Hum. Min"};

```

Hasta aquí tenemos los objetos `BarChart` creados, la leyenda con los puntos X e Y, pero **NO** tenemos creada la gráfica. Hay que tener claro el orden y la estructura del código ya que es sencillo confundirse ya que hay varios pasos a seguir:

- 1- Obtenemos el objeto `Chart` con sus atributos (descripción, tamaño de texto, `BGColor` y animación)
- 2- Creamos la leyenda y sus características (formato, forma, colores, información etcétera).
- 3- Identificamos y damos formato a los puntos X e Y del diagrama (posición, formato).
- 4- Y por último, con los tres puntos anteriores programados, creamos el diagrama con los objetos: `Charts & legends`

```

private void createCharts() {
    barChart = (BarChart) getChart(barChart, "", Color.GRAY, Color.rgb(76, 80, 118), 3000);
    //barChart.setDrawGridBackground(false);
    //barChart.setDrawBarShadow(false);

    barChart.setData(getBarData());
    barChart.invalidate();
    axisX(barChart.getXAxis());
    axisLeft(barChart.getAxisLeft());
    axisRight(barChart.getAxisRight());

    barChart.getAxisLeft().setAxisMaximum(100);
    barChart.getAxisLeft().setAxisMinimum(0);
    barChart.setBackgroundColor(getResources().getColor(R.color.oscuroPrimary));
    barChart.getAxisLeft().setTextColor(getResources().getColor(R.color.white));
    barChart.getAxisLeft().setAxisLineColor(getResources().getColor(R.color.white));
}

```

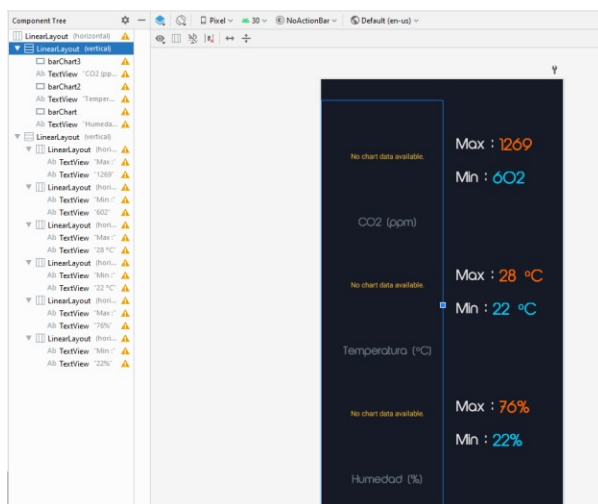
```
private DataSet getData(DataSet dataSet) {
    dataSet.setColors(datosColores);
    dataSet.setValueTextColor(Color.WHITE);
    dataSet.setValueTextSize(10);
    return dataSet;
}

private BarData getBarData() {
    BarDataSet barDataSet = (BarDataSet) getData(new BarDataSet(getBarEntries(), ""));
    barDataSet.setBarShadowColor(Color.DKGRAY);
    BarData barData = new BarData(barDataSet);
    barData.setBarWidth(0.45f);
    return barData;
}
```

Estos métodos tienen que replicarse tantas veces como diagramas quieras. En nuestro caso hemos necesitado 3 ya que queremos tener un diagrama para CO2, Temperatura y Humedad. El resultado final es el siguiente:

## XML:

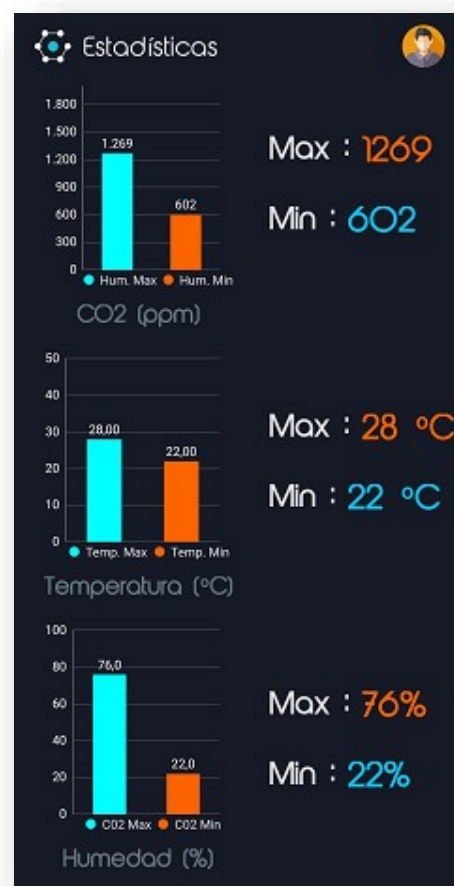
Para este Fragment, hemos utilizado un Linear Layout horizontal y vertical:



De esta manera, era más sencillo ordenar tanto los Charts como los textos.

## Otra forma de mostrar información:

Hemos basado la aplicación en una obtención de datos y estos estar representados de manera gráfica. Si bien casi la totalidad de la información representada nace de una librería (MP Chart) y objetos de tipo Chart, hemos querido utilizar otro tipo de objeto para mostrar y resaltar la información del CO2.

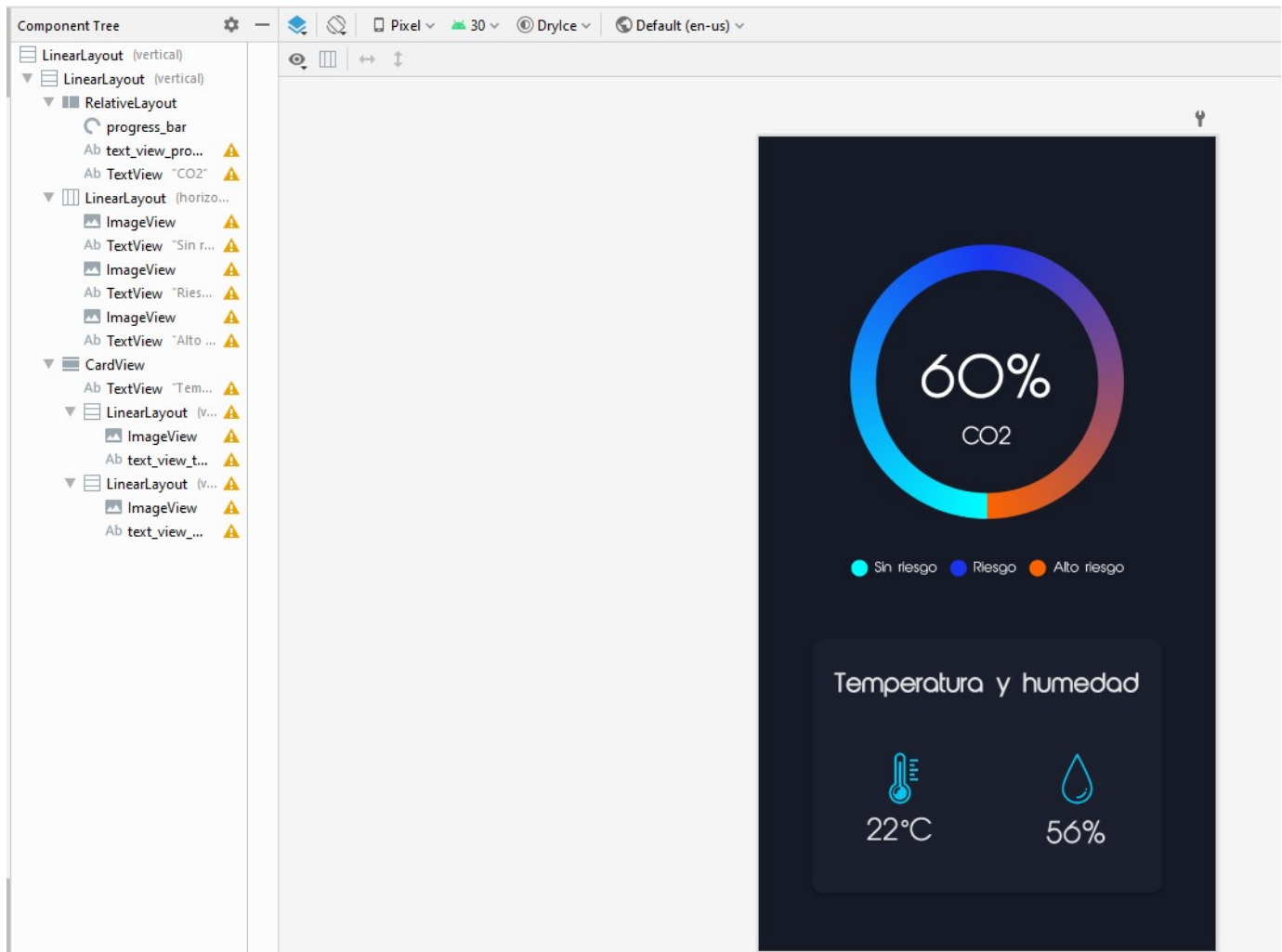


Para ello hemos utilizado el objeto Progress Bar, el cual nos permite tener algo parecido a un diagrama de pastel, pero más sencillo y práctico para desempeñar su función, que es mostrarnos un dato crucial:



XML:

Así se muestra en la vista de diseño, dicha Progress Bar:



De la misma manera que con las gráficas y su librería, este objeto requiere de lógica de programación para hacerlo funcional y de la misma manera que hemos hecho con el objeto BarChart, vamos a desgranar el código utilizado:

## JAVA:

Como en cualquier lenguaje orientado a objetos, lo primero es crear el objeto.

```
private TextView text_view_progress;  
private TextView text_view_temp;  
private TextView text_view_hum;  
private ProgressBar progress_bar;
```

Ahora, utilizamos una serie de métodos que van a hacer que el Progress Bar “cobre vida”:

```
private void updateProgressBar() {  
    new CountDownTimer(2000, 40) {  
        int progreso = 50; // Variable que va a ir aumentando del progreso  
  
        @Override  
        public void onTick(long millisUntilFinished) {  
            if (progreso <= co2) {  
                progress_bar.setProgress(progreso);  
  
                text_view_progress.setText(String.valueOf(progreso));  
                if (co2 <= 617)  
                    text_view_progress.setTextColor(Color.rgb(0, 255, 255));  
                else if (co2 <= 1234)  
                    text_view_progress.setTextColor(Color.rgb(26, 56, 241));  
                else  
                    text_view_progress.setTextColor(Color.rgb(252, 100, 2));  
                progreso += (50);  
            } else {  
                progress_bar.setProgress(co2);  
                text_view_progress.setText(String.valueOf(co2));  
            }  
        }  
  
        @Override  
        public void onFinish() {  
            progress_bar.setProgress(co2);  
        }  
    }.start();  
}
```

Lo que hace que nuestra Progress Bar se muestre de un color diferente dependiendo de las mediciones del sensor y esto tan importante, tiene sentido gracias al código mostrado arriba.

Tal y como hemos comentado, queríamos un objeto simple pero muy conciso a la hora de mostrar la gráfica de CO<sub>2</sub>. Recordad que los datos se actualizan en tiempo real gracias al sensor + Firebase.

Para la Progress Bar, no hay que importar ninguna librería externa ya que pertenece a las librerías internas de Java.



## LineChart:

Importando de la misma librería que BarChart (MP Charts) hemos utilizado esta gráfica para tener quizás, un dato menos representativo, pero igualmente válido y quizás dependiendo del usuario prefiera un tipo de dato u otro.

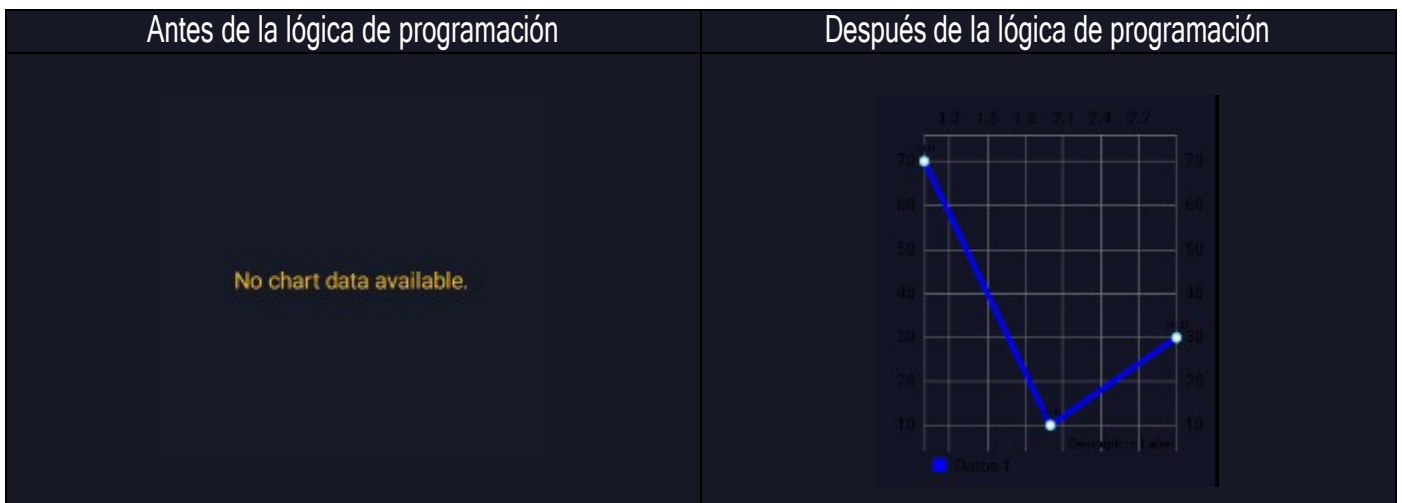
La implementación es similar al diagrama de barras. Así se muestra la vista de diseño:

## Java

```
<com.github.mikephil.charting.charts.LineChart
    android:id="@+id/lineChart2"
    android:layout_width="192dp"
    android:layout_height="220dp"
    android:layout_above="@+id/lineChart3"
    android:layout_alignParentEnd="true"
    android:layout_marginEnd="1dp"
    android:layout_marginBottom="39dp">
```

No chart data available.

Al igual que en los anteriores diagramas, podemos crear el objeto en la vista de diseño pero hasta que no desarrollamos la lógica, no tenemos el objeto disponible:



Para el Desarrollo de esta gráfica, no hemos necesitado crear métodos para los puntos X e Y. Es un código más simple ya que la propia gráfica es más simple.

Algunas de las líneas más importantes son:

```
private LineChart mChart;  
mChart = (LineChart) rootView.findViewById(R.id.LineChart);
```

Creación y encontrar el objeto por ID.

Ahora, hay que darle propiedades a dicha gráfica, por lo que utilizamos los siguientes métodos de la librería MP Chart:


```
mChart.setDragEnabled(true);  
mChart.setScaleEnabled(true);  
mChart2.setDragEnabled(true);
```

Pero ¿Cómo se mueven dichas líneas por la gráfica? Bien, existe un objeto llamado LineDataSet que, creando dicho objeto con unos argumentos tales como unos números enteros que representan mínimo y máximo y una etiqueta con un formato String, representa automáticamente la leyenda y la línea de la gráfica. Por eso dijimos anteriormente que esta gráfica, al ser más sencilla no requiere de tantos métodos, y esta pensada para registrar mínimos y máximos históricos.

```
LineDataSet set1 = new LineDataSet(yValores, "Datos CO2");  
LineDataSet set2 = new LineDataSet(yValores2, "Datos Temperatura");  
LineDataSet set3 = new LineDataSet(yValores3, "Datos Humedad");
```

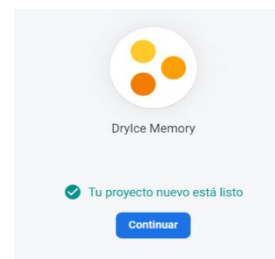
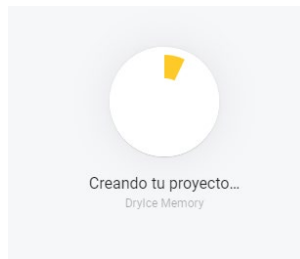
# Firebase

## Enlazar aplicación y base de datos.


Nada de lo mostrado hasta ahora tendría sentido si no tuviésemos implementada una conexión a una base de datos. Necesitábamos una base de datos que volcara información y la recogiera en tiempo real. Para ello hemos utilizado "Realtime Database. Firebase". 

Firebase es una base de datos muy gráfica y la conectividad a con nuestro proyecto es muy intuitiva. Siguiendo los siguientes pasos, crearemos nuestro proyecto:

- 1- Acceder a la URL: <https://firebase.google.com/>
- 2- Agregamos un nuevo proyecto y seguimos las indicaciones de google:



- 3- Ya tenemos en la nube de Firebase. Pero ¿Cómo lo implementamos?
- 4- Nuestro nuevo proyecto, va a tener un identificador que lo va a hacer único. Este va a ser el enlace de la base de datos recientemente creada, con nuestro proyecto:

 <https://dryice-memory-default-rtdb.firebaseio.com/>

- 5- Antes de utilizar el identificador, debemos de implementar las dependencias de Firebase:

```
implementation 'com.google.firebase:firebase-database:19.6.0'
implementation 'com.google.firebase:firebase-auth:20.0.3'
implementation 'com.firebaseui:firebase-ui-storage:6.4.0'
implementation 'com.google.firebase:firebase-storage:19.2.1'
```
- 6- Android Studio tiene su propia interfaz para enlazar la aplicación con nuestra base de datos. Siguiendo los pasos que indican la interfaz, enlazamos aplicación y base de datos sin ningún problema.



**Your Android Studio project is connected  
to your Firebase Android app**

You can now use Firebase in your project! Go back to Android Studio to start using one of the  
Firebase SDKs.

## Primeros pasos y subida de datos a Firebase

### Java

Cuando tenemos enlazada aplicación y base de datos, hay que empezar a crear métodos y lógica de programación para hacer todo más funcional. Firebase es una API que se integra perfectamente con JAVA, pero para ello, hay que crear objetos como si de una librería misma del lenguaje se tratase:

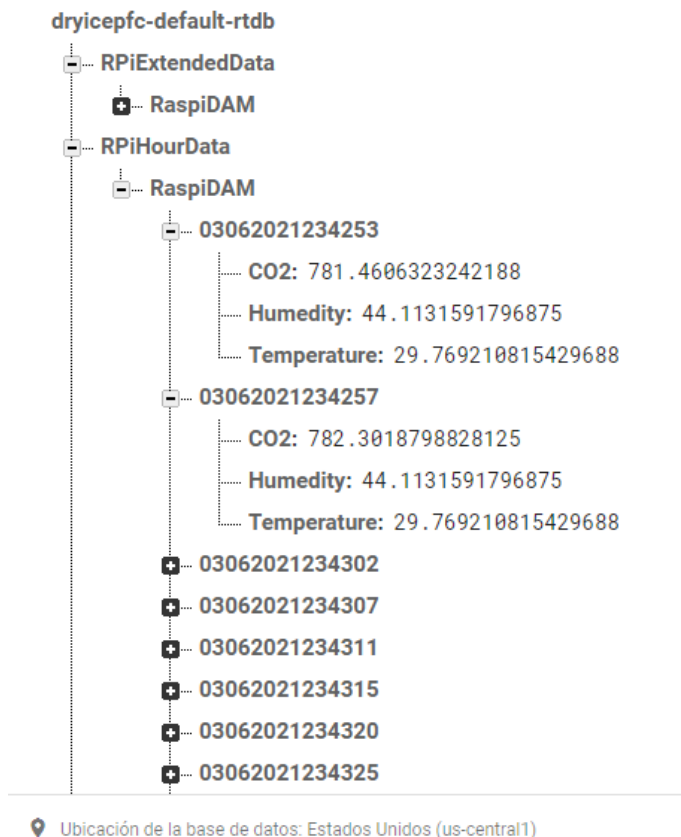
```
private static FirebaseDatabase myDatabase = FirebaseDatabase.getInstance();  
private static DatabaseReference myDatabaseReference;
```

No estamos trabajando con una base de datos relacional como por ejemplo, MySQL. Aquí la información se almacena en formato JSON y debemos de entender dicha estructura para poder subir la información y posteriormente descargarla para crear objetos funcionales y con datos en tiempo real.



Formato de datos en JSON

No estamos trabajando con una base de datos relacional como por ejemplo, MySQL. Aquí la información se almacena en formato JSON y debemos de entender dicha estructura para poder subir la información y posteriormente



Ubicación de la base de datos: Estados Unidos (us-central1)

```

public class CollectRPiHourData {
    private static FirebaseDatabase myDatabase =
        FirebaseDatabase.getInstance();
    private static DatabaseReference
        myDatabaseReference;

    public static void takeData(Communication
        communication, String idRPi){
        myDatabaseReference =
            myDatabase.getReference("RPiHourData").child(idRPi);

        List<RTExtendedData> rtExtendedDataList = new
            ArrayList<>();
        myDatabaseReference.addValueEventListener(new
            ValueEventListener() {

```

Nota: Los datos vienen del programa en Python y el sensor de CO2. Con los datos recogidos y el código JAVA, conseguimos tener los datos en firebase.

En estas dos imágenes vemos, a la izquierda cómo se visualizan los datos almacenados en Firebase, a la derecha la lógica de programación que hace que sea posible.

Cuando utilizamos el método `.getReference`, el parámetro de dicho método va a ser el nombre de nuestro campo en a base de datos. Dicho campo tendrá más campos en su interior, para ello utilizamos `.child`

En resumen, una referencia es la ubicación específica en la base de datos. Puede hacer referencia a la ubicación raíz o secundaria en su base de datos. *Ejemplo*

```

firebase.database().ref() O firebase.database().ref("child/path").

```

## Recuperación de datos con Firebase

### Java

Cuando tenemos enlazada aplicación y base de datos, hay que empezar a crear métodos y lógica de programación para así nutrir de datos las gráficas, responsables de mostrar la información que necesitamos.

```
@Override
public void onDataChange(@NonNull DataSnapshot snapshot) {
    Iterable<DataSnapshot> data = snapshot.getChildren();
    for (DataSnapshot d: data) {
        RPiUser idRPi = d.getValue(RPiUser.class);
        idsRPis.add(idRPi);
    }
    communication.sendDataRPiUsers(idsRPis);
}
```

Al utilizar el método: `myDatabaseReference.addValueEventListener` automáticamente se nos generan los métodos Override `onDataChange` y `onCancelled`. Es en el método `onDataChange` donde nos enfocamos para crear la lógica de programación y así recuperar los datos.

Concretando un poco más el código del método `onDataChange`, la línea más importante es la siguiente:  
`RPiUser idRPi = d.getValue(RPiUser.class);`

En la cual, con el método `getValue` obtenemos el valor de `idRPi` con los atributos de la clase `RpiUser`.

De esta forma, los datos subidos a firebase:



Se muestran en nuestra aplicación en forma de Market y del perfil relleno del usuario.

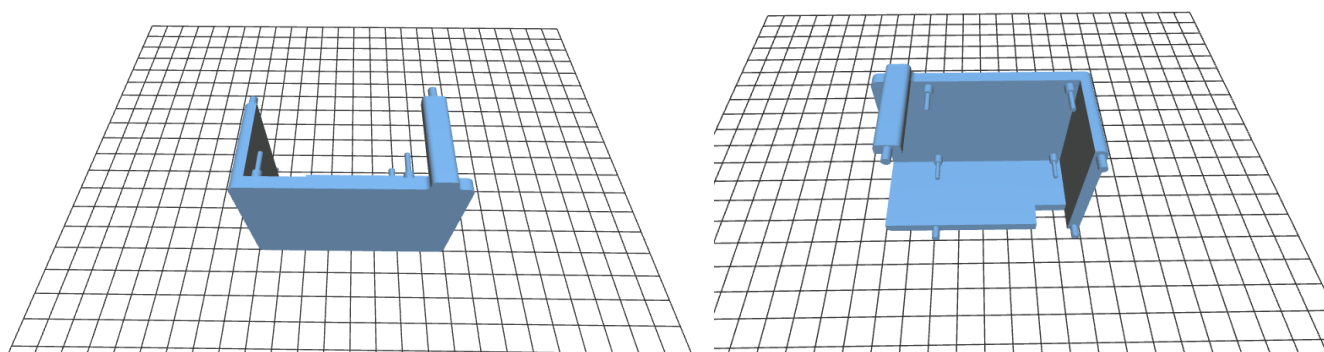


## Carcasa e impresión 3D

### Diseño

Hemos querido profesionalizar el sistema IoT con una carcasa donde todos los materiales tienen una mayor protección garantizar el cuidado del aparato y a su vez, mejorar la estética.

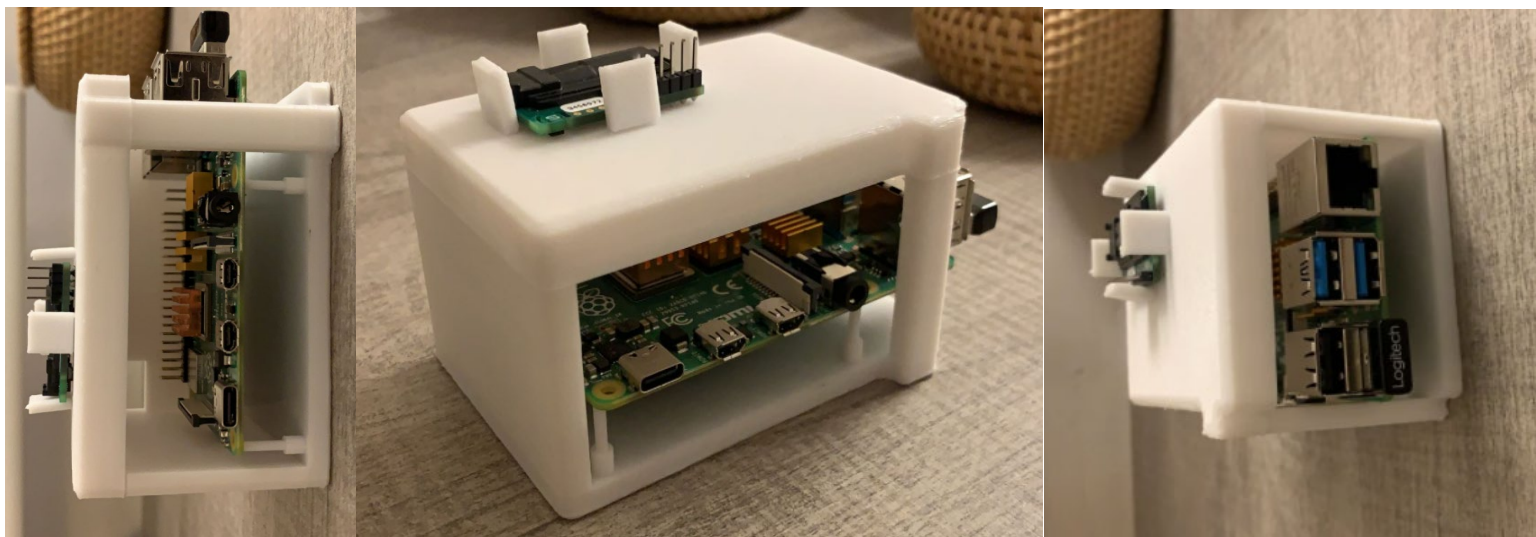
En primer lugar, se han realizado las medidas tanto de la RPI como del sensor, para así después realizar un diseño 3D sobre el cual, se va a construir la carcasa.



**Nota: Visualización de todo el diseño de la carcasa en 3D:**

<https://github.com/2DAMUE/pfcjun21-dryice/tree/master/3D%20Box>

## Resultado final



El resultado final es un sistema IoT completo, con una carcasa impresa en 3D integrada con una aplicación totalmente funcional.



## Conclusión

Para concluir la memoria, vamos a exponer los distintos problemas que nos han ido surgiendo y como finalmente los hemos solucionado.

Para empezar, la idea principal era hacer una aplicación híbrida con React Native. La aplicación iba a ir sobre información de coches.

Tras aprender esa tecnología, nos fue complicado hacer una aplicación porque teníamos una gran cantidad de errores que no sabíamos solucionar.

Tras ello, y como hemos comentado previamente, nos decantamos por hacer una aplicación para android con lenguaje java, que estuviese unida a obtener datos de un sensor conectado a una Raspberry. Es por ello por lo que comenzamos a elaborar el proyecto iot.

La primera idea era obtener los datos con el sensor a través de una arduino. Tras intentarlo, no conseguimos subir datos a firebase y cambiamos a la opción de conectar el sensor a la Raspberry. Con la Raspberry todo funcionaba bien, por lo que hicimos dos scripts, uno para registrar la Raspberry y otro para hacer login y pasar los datos con ese id del sensor y cada 4 segundos recoger datos.

Tras ello, firebase estaba organizado de tal forma que tenía Usuarios de la app, usuarios de la Raspberry, Datos simples de la Raspberry y Datos de Máximas y Mínimas de la Raspberry.

Las ventanas de la aplicación tienen un diseño muy preciso. Está diseñada para que el usuario identifique los datos de la mejor manera posible.

La aplicación tiene una pantalla de gráficos, que tiene un par de fragments implementados, en el que el primero muestra los datos a tiempo real y el segundo, muestra los datos de máximas y mínimas de cada uno.

La aplicación también tiene un mapa. Ese mapa tiene un botón en el que te ubicas y te aparecen las distintas ubicaciones de las raspberries y puedes ver los datos que tienen en el momento en el que le das.

Sin registrarse, entras en la app y te aparece un mapa en el que ves los datos de las raspberries de otros, por lo que cualquier usuario puede acceder a la app.

Finalmente, esta es la aplicación que hemos creado. Esta podría ser la primera versión que se podría implementar para todos los públicos.

El modelo de negocio que tendría este proyecto de iot sería muy grande.

## BIBLIOGRAFÍA Y WEBGRAFÍA

adafruit-circuitpython-scd30. (s/f). Recuperado el 25 de mayo de 2021, de Pypi.org website: <https://pypi.org/project/adafruit-circuitpython-scd30/>

Adobe Colors. (s/f). Recuperado el 22 de mayo de 2021, de Adobe.com website: <https://color.adobe.com/es/create/color-wheel>

Android Studio Documentation. (s/f). Recuperado el 23 de mayo de 2021, de Android.com website: <https://developer.android.com/docs>

Childs-Maidment, J. (s/f). Pyrebase.

Flaticon, la mayor base de datos de iconos vectoriales gratis. (s/f). Recuperado el 27 de mayo de 2021, de Flaticon.es website: <https://www.flaticon.es/>

Free Lottie animation files, tools & plugins - LottieFiles. (s/f). Recuperado el 1 de junio de 2021, de Lottiefiles.com website: <https://lottiefiles.com/>

Gay, W. W. (2014). GPIO. En Mastering the Raspberry Pi (pp. 117–155). Berkeley, CA: Apress.

Millington, S. (2018, septiembre 23). Hashing a Password in Java. Recuperado el 2 de junio de 2021, de Baeldung.com website: <https://www.baeldung.com/java-password-hashing>

PyPI · The Python Package Index. (s/f). Recuperado el 28 de mayo de 2021, de Pypi.org website: <http://pypi.org>

Stack Overflow en español. (s/f). Recuperado el 20 de mayo de 2021, de Stackoverflow.com website: <https://es.stackoverflow.com/>

YouTube. (s/f). Recuperado el 20 de mayo de 2021, de Youtube.com website: <https://www.youtube.com/>