

# **Gestión de Dispositivos IoT del Hogar**

**Marta Muñoz Barrios**

**16/03/2025**

**IES El Majuelo, Gines**

## Contenido

<b>Gestión de Dispositivos IoT del Hogar .....</b>	<b>1</b>
Requisitos de hardware y software.....	4
Requisitos de hardware.....	4
Requisitos de software .....	4
Guía de instalación del entorno de desarrollo .....	6
Instalación del entorno .....	6
Arquitectura de la solución .....	7
Arquitectura general .....	7
Arquitectura MVC.....	7
Diagrama de arquitectura .....	9
Puesta en marcha de la aplicación .....	11
Puesta en marcha del simulador de Estación Meteorológica DHT22 .....	11
Puesta en marcha de la aplicación móvil .....	12
Descripción de procesos de la aplicación.....	13
Comunicación con el dispositivo .....	13
Backend .....	13
Aplicación web .....	14
Aplicación móvil .....	15
Dificultades encontradas.....	17
Propuestas de mejora .....	18
Pruebas.....	19
Aplicación web .....	19
Aplicación móvil .....	27
Comunicación entre plataformas.....	33
Bibliografía .....	38

## Resumen

Aplicación web y móvil a través de la cual los usuarios pueden visualizar el estado de funcionamiento y gestionar los dispositivos IoT del hogar.

La aplicación permitirá apagar y encender los dispositivos y ver su estado (encendido/apagado) así como añadir nuevos dispositivos al hogar. Los dispositivos serán simulados para esta versión de la aplicación.

## Requisitos de hardware y software

### Requisitos de hardware

#### *Especificaciones mínimas*

- Procesador 2 núcleos a 2.0 GHz
  - o 2 núcleos para manejar tareas simultáneas, como el procesamiento de los datos de temperatura y humedad, la consulta de la base de datos y la gestión de peticiones del backend.
- 4 GB de RAM
  - o Para manejar la concurrencia y almacenamiento temporal de datos.
- Espacio en disco al menos 40 GB (HDD o SSD)
  - o Para guardar logs y archivos temporales.
- Conexión a Internet
  - o Fundamental para la comunicación de MQTT y la consulta de la API.

#### *Especificaciones recomendadas*

- Procesador 4 núcleos a 2.5GHz
- 8 GB de RAM
- 80 GB de almacenamiento (SSD)

### Requisitos de software

#### *Sistema Operativo*

- Windows 10.

#### *Herramientas*

- IDE/Editor de código:
  - o Visual Studio Code
  - o PowerShell
    - Para ejecutar scripts. Se ha utilizado la extensión de PowerShell de Visual Studio Code.
- Postman
  - o Facilita la creación de solicitudes HTTP y pruebas de endpoints de la API para asegurar su correcto funcionamiento.
- MQTT Broker
  - o Se usa para la transmisión de datos de temperatura y humedad entre dispositivos mediante el protocolo MQTT
- Navegador Web
  - o Necesario para acceder al frontend y visualizar la aplicación.
- Android Studio
  - o Para la programación de la aplicación móvil.

#### *Lenguajes y Frameworks:*

- Python 3
  - o Lenguaje de programación utilizado en el backend.
- Flask
  - o Framework para crear API RESTful
- Flask-CORS
  - o Extensión para habilitar el soporte de CORS (Cross-Origin Resource Sharing).
- Librería mysql-connector-python

- Conector de Python para interactuar con bases de datos MySQL.
- Node.js y NPM
  - Node.js es un entorno de ejecución de JavaScript y NPM es el gestor de paquetes. Utilizado en el frontend (como Express.js).
- Express.js
- http-proxy-middleware
  - Middleware para la configuración de proxies HTTP en Node.js. Ayuda a gestionar las conexiones entre el frontend y el backend.
- HTML, CSS
  - Lenguajes para el desarrollo del frontend.
- Bootstrap: Framework para el diseño responsivo del frontend.

### Base de Datos

- XAMPP – MySQL
  - Contiene instalación de MySQL.

### Aplicación Móvil

- Android Studio Ladybug Feature Drop | 2024.2.2 Patch 1
  - IDE oficial para el desarrollo de aplicaciones móviles Android.
- Kotlin
  - Lenguaje para el desarrollo de la aplicación Android.
- Jetpack Compose
  - Framework elegido para la UI en la aplicación móvil.

### Resumen de Paquetes/Librerías Requeridas:

<b>Backend</b>	Flask Flask-CORS mysql-connector-python paho-mqtt
<b>Frontend</b>	Express.js http-proxy-middleware Bootstrap
<b>Base de Datos</b>	MySQL (XAMPP recomendado para local)
<b>MQTT</b>	paho-mqtt
<b>Aplicación Móvil</b>	Android Studio Kotlin Jetpack Compose Retrofit Material UI

## Guía de instalación del entorno de desarrollo

### Instalación del entorno

- Instalación de Python.
  - I. Se descarga la última versión de Python de la página oficial. [Windows installer \(64-bit\)](#)
  - II. Se ejecuta el instalador y se marca la opción "Add Python to PATH"



- III. Se hace click en Install Now y se espera a que finalice.
  - IV. Por último, se verifica la instalación de Python en el terminal de Windows con el comando:  

```
python --version
```
- Instalación de Visual Studio Code y extensiones necesarias.
    - I. Descargar VS Code descargando el instalador para Windows de la página oficial.
    - II. Instalar VS Code ejecutando el instalador y siguiendo las instrucciones, marcar "Add to PATH"
  - Instalación de XAMPP.
    - I. Se descarga el instalador para Windows de XAMPP de la web oficial.
    - II. Se ejecuta el instalador descargado y se siguen las instrucciones. Es necesario marcar MySQL para este proyecto,
    - III. Una vez que MySQL esté en funcionamiento, se puede acceder a phpMyAdmin, que es una herramienta web para gestionar bases de datos MySQLe en <http://localhost/phpmyadmin>.
  - Instalación de NGROK
    - I. Descargar Ngrok del sitio web oficial e instalar para Windows.
    - II. Crear cuenta y autenticarse
    - III. Una vez registrado, ir a panel de control en Ngrok Dashboard y copiar el token de autenticación (se encuentra en la sección Auth).

## Arquitectura de la solución

### Arquitectura general

- Explicación del flujo de datos:
  1. El simulador genera datos de temperatura y humedad.
  2. Los datos se publican en un tópico específico de un bróker MQTT.
  3. El conector de la base de datos escucha el tópico y guarda la información en MySQL.
  4. El backend (Flask) ofrece una API para consultar los datos.
  5. El frontend (Bootstrap + HTML + JavaScript o Aplicación Android) solicita datos al backend mediante protocolo HTTP (REST API) a través del proxy web en Node.js.
  6. El proxy web gestiona el acceso entre el frontend y la API.

### Arquitectura MVC

#### 1. Backend (Flask):

- **/app** (directorio principal de la aplicación Flask)
  - **/models:** Contendrá los archivos relacionados con la base de datos y las interacciones con MySQL. El modelo debe encargarse de la lógica de acceso a datos.
    1. `__init__.py` indica al intérprete de Python que el directorio *package* contiene un módulo, y que debe tratarlo como tal (es decir, hacer que sea posible importar los archivos como parte del módulo).
    2. `bd_conf.py` contiene la lógica y configuración necesarias para la conexión con la base de datos.
    3. `mqtt_data_handler.py` gestiona la lógica de base de datos y procesamiento de mensajes procedentes del cliente MQTT.
    4. `devices_model.py` contiene las funciones que interactúan directamente con la base de datos para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre los dispositivos.
  - **/controllers:** Contendrá los controladores que manejarán las rutas y las solicitudes HTTP.
    1. `__init__.py` para que el directorio se interprete como módulo.
    2. `mqtt_client_handler.py` encapsula toda la lógica relacionada con el cliente MQTT: conexión, desconexión, suscripción y recepción de mensajes.
    3. `mqtt_controller.py` instancia el objeto `mqtt_client_handler` y realiza la conexión y mantenimiento del script en ejecución.

4. routes.py define las rutas (endpoints) de la API de la aplicación en Flask. Se encarga de manejar la lógica de las peticiones HTTP recibidas.

- requirements.txt contiene las librerías necesarias para la ejecución de las dependencias existentes en la aplicación Flask, la conexión con la base de datos y el manejo de la conexión MQTT.

## 2. Frontend (/mi-web):

- **/public:** Aquí se incluyen los archivos HTML, CSS y JavaScript que definen la interfaz al usuario.
  - /images contiene los recursos png que se han utilizado para el frontend.
  - index.html es la página principal de la aplicación web. Muestra la lista de dispositivos conectados.
  - add-device.html es la interfaz para agregar nuevos dispositivos.
  - add-device.js maneja la lógica del formulario en add-device.html. Captura los datos ingresados por el usuario, valida la información antes de enviarla y envía una petición al backend para registrar el nuevo dispositivo.
  - display-device.js se encarga de mostrar y actualizar la lista de dispositivos en index.html: obtiene la lista de dispositivos desde el backend (API), renderiza los dispositivos en la interfaz de usuario y permite realizar acciones como activar/desactivar dispositivos o eliminarlos.
  - style.css contiene los estilos visuales de la aplicación web.

## 3. Proxy (/proxy)

- server-proxy.js incluye la generación del servidor proxy configurado con Node.js y Express.

## 4. Simulador de dispositivo (/device\_simulator):

- simulation.py es un script que simula el funcionamiento de una estación meteorológica dht22 que realiza mediciones periódicas de temperatura y humedad y las envía a un servidor MQTT.

### *Aplicación móvil – Arquitectura MVVM*

La aplicación móvil está organizada de acuerdo con un patrón de arquitectura de software conocido como MVVM (Modelo-Vista-VistaModelo), que es común en aplicaciones móviles, especialmente en plataformas como Android, ya que ayuda a separar las responsabilidades de la UI, la lógica de negocio y la fuente de datos.

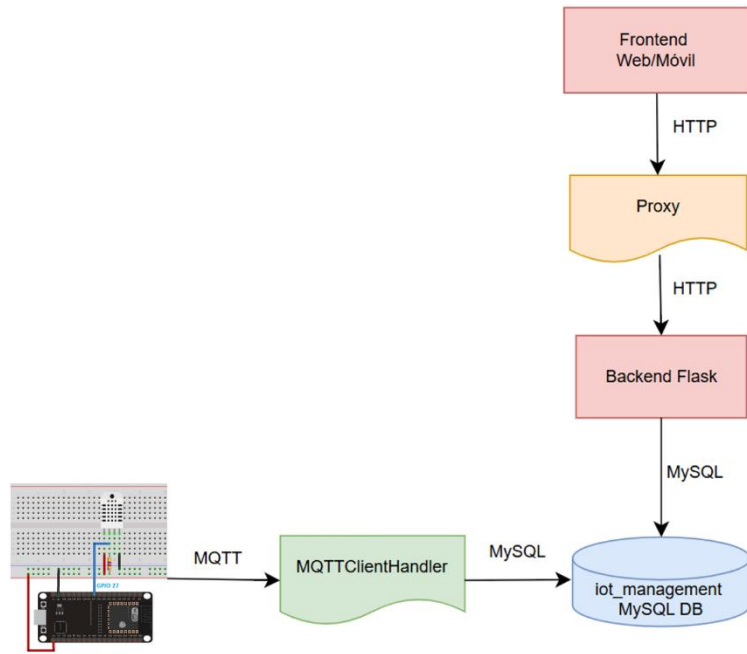
1. /data interacción con las fuentes de datos locales y remotas
  - /local



- Device.kt define el modelo de datos de un dispositivo
- DeviceDao.kt es la interfaz de acceso a datos (DAO, Data Access Object) que define las operaciones que se pueden realizar sobre la base de datos local, como insert(), getAll(), update(), etc.
- /remote contiene las clases que manejan la comunicación con el backend a través de la API. Aquí se gestionan las solicitudes HTTP y las respuestas.
  - ApiClient.kt configuración básica de la red, como la creación del cliente de Retrofit o la configuración de las cabeceras de la API.
  - ApiService.kt contiene los métodos de las solicitudes a la API, se definen los endpoints (por ejemplo, getDevices(), addDevice(), etc.) que se corresponden con las rutas del backend.
  - DeviceDto.kt define los objetos de transferencia de datos (DTOs) que se usan para transferir información entre la aplicación móvil y el backend. Contiene las propiedades de los dispositivos que se reciben o envían a la API.
- /repository maneja la lógica de acceso a datos
  - DeviceRepository.kt gestiona la comunicación entre la aplicación y las fuentes de datos (el API remoto).
- 2. /presentation lógica relacionada con la interfaz de usuario y la comunicación entre la vista (UI) y los datos
  - AddDeviceScreen.kt representa la pantalla para agregar un nuevo dispositivo. Contiene la lógica de la interfaz de usuario para obtener la entrada del usuario, como el nombre, tipo y estado del dispositivo, y enviarla al ViewModel para que se maneje la lógica de negocio.
  - DeviceListScreen.kt representa la pantalla donde se muestra una lista de dispositivos. Recibe la lista de dispositivos desde el **ViewModel** y los presenta al usuario.
  - DeviceViewModel.kt es el responsable de manejar la lógica de la presentación.
- 3. /theme contiene archivos relacionados con el tema y la apariencia de la aplicación, como estilos, colores y fuentes.
- 4. MainActivity.kt carga las vistas iniciales, como la pantalla de inicio.

### Diagrama de arquitectura

La arquitectura implementada en este proyecto es la siguiente:



## Puesta en marcha de la aplicación

Para la puesta en marcha en primer lugar, hay que ejecutar las siguientes aplicaciones:

- I. XAMPP.
  - a. Ir al gestor de aplicaciones de Windows y cerrar los procesos mysql.exe en ejecución
  - b. Ir a XAMPP y darle a “start” a los servicios de Apache y MySQL
- II. Visual Studio Code: abrir el directorio /TFG en el terminal y ejecutar el script start-app.ps1 que contiene las tareas de puesta en marcha de la aplicación.

**start-app.ps1: script** en PowerShell gestiona las tareas de inicio de forma automática:

1. Mata procesos en los puertos que se van a utilizar por los procesos principales de la aplicación.
  - Verifica si los puertos 3000 (proxy) y 1883 (MQTT) están ocupados.
  - Si hay procesos usándolos, los finaliza automáticamente.
  - Esto evita conflictos cuando se reinicia la aplicación.
2. Instala dependencias de Python definidas en requirements.txt
  - Verifica si pip está instalado.
  - Instala las dependencias definidas en requirements.txt.
3. Conexión a MySQL y creación de base de datos.
  - Comprueba si el servicio MySQL está en ejecución.
  - Si no lo está, intenta iniciarlo desde XAMPP.
  - Luego, prueba la conexión y crea la base de datos si no existe.
4. Iniciar el servidor de Flask
  - Ejecuta el script routes.py, que contiene la lógica del backend.
5. Instala dependencias del proxy y lo inicia
6. Iniciar el controlador de la conexión MQTT con la base de datos.
7. Abrir la interfaz web: Una vez que todo está en marcha, abre <http://localhost:3000> para interactuar con la aplicación.

Este script cubre la necesidad de disponer de múltiples procesos ejecutándose simultáneamente.

Únicamente se ha dejado un paso manual, con el objetivo de replicar un caso real de forma más precisa:

- Ejecución del simulador del dispositivo.

## Puesta en marcha del simulador de Estación Meteorológica DHT22

Para representar las mediciones de la Estación Meteorológica DHT22, se siguen los siguientes pasos:

1. Abrir el cmd e ir a la ruta donde se encuentra el archivo simulation.py  
`cd C:\Users\marmu\OneDrive\Documentos\TFG\device_simulator\`
2. Instalar las librerías necesarias para la ejecución del simulador:  
`install pip`  
`pip install paho-mqtt`
3. Ejecutar el script mediante el siguiente comando  
`python simulation.py`

Una vez puesto en marcha el simulador, comprobar en la terminal de Windows que se generan mensajes con el envío al bróker MQTT de las mediciones y se conecta con éxito.

### [Puesta en marcha de la aplicación móvil](#)

Se ha instalado la aplicación en un dispositivo móvil real para la presentación. También se puede poner en marcha mediante el simulador de dispositivos de Android Studio, ejecutando este entorno de desarrollo y haciendo clic en Run 'app'.

Para la ejecución de la apk en un dispositivo móvil real, es necesaria la instalación de ngrok para publicar el servidor que está corriendo en local a una URL estática. Pasos para la configuración:

`ngrok http 5000`

`ngrok authtoken <tu_token_de_autenticación>`

Obtener URL pública: Forwarding      `http://abc123.ngrok.io -> http://localhost:5000`

## Descripción de procesos de la aplicación

### Comunicación con el dispositivo

El broker MQTT, al ser un broker gratuito, limita el tiempo de conexión o la cantidad de mensajes enviados en un período corto.

El cliente MQTT se desconectaba de forma recurrente con el código 7. Este código indica que la conexión con el broker se perdió inesperadamente.

Para evitar estos problemas, en primer lugar, reemplazamos `loop_forever()` por `loop_start()`, ya que este segundo se ejecuta en un hilo separado, permitiendo que el programa siga corriendo sin bloqueos.

Adicionalmente, se controla manualmente el bucle principal, lo que garantiza que, si se desconecta, se realiza un reintento de conexión al bróker.

### Backend

#### *Descripción de la API*

La API Rest se estructura en 3 partes: la definición de los endpoints, la gestión del acceso a la base de datos, y las funciones de modelo que realizan operaciones CRUD sobre la BD.

Se trata de una API REST (Representational State Transfer) por las siguientes razones fundamentales que se alinean con los principios del estilo arquitectónico REST:

- Uso de métodos HTTP estándar: Utiliza los métodos HTTP (GET, POST, PUT, DELETE) para realizar operaciones CRUD sobre los dispositivos.
- Recursos identificados por URLs: Los dispositivos son identificados por URLs únicas.
- Sin estado (Stateless): Cada solicitud es independiente, lo que significa que no se mantiene información del cliente entre solicitudes.
- Respuestas en formato JSON: La API devuelve respuestas en formato JSON, que es un estándar común para representar los recursos.

La API cuenta con los siguientes endpoints:

- Obtener todos los dispositivos
  - Ruta: `/devices`
  - Método HTTP: GET
  - Descripción: Obtiene todos los dispositivos de la base de datos, con la opción de ordenarlos por un campo específico (`sort_by`) y en un orden determinado (`order`).
- Añadir un nuevo dispositivo
  - Ruta: `/add_device`
  - Método HTTP: POST
  - Descripción: Añade un nuevo dispositivo a la base de datos. El cuerpo de la solicitud debe contener los datos del dispositivo, como `name`, `type`, `status`, y opcionalmente `battery_level`.
- Actualizar un dispositivo existente

- Ruta: /update\_device/<int:device\_id>
- Método HTTP: PUT
- Descripción: Actualiza los datos de un dispositivo existente en la base de datos. Se requiere el ID del dispositivo (device\_id) en la URL, y el cuerpo de la solicitud debe contener los nuevos valores para los campos name, type, y status.
- Cambiar el estado de un dispositivo
  - Ruta: /toggle\_device/<int:device\_id>
  - Método HTTP: POST
  - Descripción: Cambia el estado de un dispositivo a ON o OFF. Se requiere el ID del dispositivo (device\_id) en la URL y el cuerpo de la solicitud debe contener el nuevo estado en formato JSON.
- Eliminar un dispositivo por ID
  - Ruta: /delete\_device/<int:device\_id>
  - Método HTTP: DELETE
  - Descripción: Elimina un dispositivo de la base de datos, identificado por su ID (device\_id).

#### *Proxy web (proxy.js, con Express.js y http-proxy-middleware)*

El proxy se encarga de redirigir las solicitudes del frontend a la API. Usar esta capa tiene los siguientes beneficios:

- Evitar CORS: Esto soluciona problemas de CORS, que ocurrirían al hacer peticiones directamente entre dominios/puertos distintos.
- Lógica de la API "oculta" para el frontend a través del proxy.
- Configuración sencilla: Con Node.js y Express, configurar un proxy es relativamente sencillo y no requiere configuraciones complicadas.

El archivo server-proxy configura un servidor que escucha en el puerto 3000 y redirige las solicitudes a la API que está corriendo en el puerto 5000.

Node.js es el entorno de ejecución que permite correr el código JavaScript fuera del navegador. Sobre este entorno se realizan los siguientes procesos:

- Se utiliza Express para crear un servidor web sencillo. Node.js ejecuta Express para manejar las solicitudes HTTP que llegan al servidor.
- El http-proxy-middleware permite redirigir las solicitudes HTTP desde el proxy hacia el servidor Flask.

#### *Aplicación web*

##### **1. Pantalla principal**

La pantalla principal de la aplicación web permite gestionar una lista dinámica de dispositivos IoT, interactuando con una API para obtener, actualizar, eliminar y mostrar dispositivos en un formato adecuado. Contiene los siguientes elementos:

- El encabezado es un header con la clase container-fluid, que asegura que el encabezado ocupe todo el ancho de la pantalla.
- Se incluye una barra de navegación de Bootstrap (navbar), que se adapta a pantallas grandes y pequeñas y permite navegar a la otra funcionalidad disponible (Añadir dispositivo).
- La aplicación mediante la lógica del frontend programada en JavaScript, carga los dispositivos de la API y los muestra en la página en formato de tarjetas. Los dispositivos se ordenan según el criterio seleccionado por el usuario (ID, nombre, tipo, estado), y se organiza la presentación de cada uno en tarjetas de Bootstrap.

## 2. Pantalla de Añadir Dispositivo

Esta pantalla contiene la funcionalidad de agregar nuevos dispositivos IoT a través de un formulario en una interfaz web. Al enviarse el formulario, los datos del dispositivo se envían a la API de Flask para ser procesados y almacenados.

Los usuarios reciben un mensaje inmediato sobre el éxito o fracaso de la operación sin necesidad de navegar fuera de la página o realizar pasos adicionales.

El nivel de batería se genera de forma aleatoria, lo que simula una variable dinámica en el dispositivo.

### Aplicación móvil

A continuación, se explican los procesos más importantes de la aplicación móvil:

#### 1. Comunicación con el API REST a través de Retrofit.

El ApiClient actúa como un punto único de acceso para las solicitudes de API a través del ApiService. La instancia de ApiService se inicializa de manera perezosa (by lazy), lo que significa que solo se crea cuando se necesita por primera vez.

En el caso de Android, el servidor debe accederse desde <http://10.0.2.2:5000/>

Se crea una instancia de ApiService utilizando Retrofit. El GsonConverterFactory convierte las respuestas JSON de la API en objetos de Kotlin.

La clase ApiService define las Rutas de la API que la aplicación móvil utilizará para interactuar con el backend. Aquí se especifican las solicitudes HTTP (GET, POST, DELETE, etc.) y cómo deben ser tratadas.

#### 2. Obtención de datos del API REST

Esta comunicación se realiza en la clase DeviceRepository, que sigue el patrón Repositorio, que sirve de intermediario entre el API remoto y la capa de presentación.

Esta clase hace uso de corutinas para realizar las operaciones de red de manera asíncrona, evitando bloquear el hilo principal de la aplicación (UI).

Realiza las siguientes funciones:

- Obtener dispositivos (fetchDevices): Realiza una llamada asíncrona al backend para obtener la lista de dispositivos y pasarla a un callback.
- Eliminar dispositivo (deleteDevice): Permite eliminar un dispositivo enviando una solicitud al servidor y devuelve un valor booleano que indica el éxito o fracaso.

- Cambiar el estado de un dispositivo (toggleDevice): Permite cambiar el estado de un dispositivo ("encender" o "apagar") mediante una llamada al API.
- Agregar un dispositivo (addDevice): Permite agregar un nuevo dispositivo enviando sus datos al backend.

### 3. Pantalla de listado de dispositivos

La capa de presentación de la aplicación se ha realizado usando el framework Jetpack Compose. Frente a la utilización de XML supone una eficiencia a la hora de integrar en un mismo código el manejo del UI y la gestión de la navegación.

La pantalla de listado de dispositivos, obtiene los dispositivos desde DeviceViewModel usando collectAsState(), que permite observar los cambios en tiempo real.

Implementa un Scaffold con una TopAppBar que contiene:

- Un botón Agregar Dispositivo (navigate("addDeviceScreen")).
- Un botón Refrescar (viewModel.fetchDevices()).

Muestra un mensaje y permite refrescar si no hay dispositivos disponibles.

El listado de dispositivos se realiza haciendo uso de LazyColumn, componente para mostrar grandes volúmenes de datos. LazyColumn se usa para mostrar la lista de dispositivos obtenidos desde el ViewModel.

Cada uno de los dispositivos, se muestran en Cards que organizan la información del dispositivo. La vista de elemento de dispositivo implementa botones para:

- Cambiar Estado: Modifica currentStatus y llama a onToggleStatus().
- Eliminar Dispositivo: Muestra un AlertDialog de confirmación antes de eliminarlo.

### 4. Pantalla de creación de nuevo dispositivo

Esta pantalla (AddDeviceScreen) presenta un formulario que incluye varios campos de entrada, como el nombre del dispositivo, el tipo de dispositivo, el estado del dispositivo (encendido o apagado), y la validación de los datos antes de enviar el formulario. La pantalla también maneja la interacción con el viewModel para agregar el dispositivo a la base de datos o API.

La pantalla para la creación de nuevos dispositivos contiene los siguientes elementos:

- La barra de navegación contiene el título "Añadir Dispositivo" y un botón para volver a la pantalla anterior.
- Las variables de estado se utilizan para almacenar y actualizar valores en la UI, como la información ingresada en los formularios.
- Proceso de envío del formulario: Si el formulario es válido, se crea un objeto Device y se llama a la función addDevice del viewModel para agregar el dispositivo.
- El selector de tipo de dispositivo es un DropdownMenu que permite al usuario seleccionar el tipo de dispositivo de una lista de opciones. El menú se activa al hacer clic en el ícono de flecha hacia abajo.
- El selector de estado permite al usuario cambiar el estado del dispositivo entre "ON" y "OFF" utilizando botones. El color de los botones también cambia según el estado.



## Dificultades encontradas

1. Dificultades a la hora de separar las funcionalidades según el patrón de arquitectura MVC. Ya que inicialmente, realicé el código de forma monolítica y tuve que volver a realizar las mismas funcionalidades, pero con división de responsabilidades, lo que me generó muchos errores.
2. La simulación del dispositivo y la comunicación a través de un bróker MQTT fue un desafío ya que el bróker dispone de muchas limitaciones y se desconecta de forma periódica para evitar la sobrecarga, y realiza bloqueos por IP. Para esto tuve que implementar un mecanismo de reintentos en el controlador de la conexión.
3. La gestión de la UI en Compose, aunque más sencilla que en XML, supuso dificultades en la gestión dinámica del cambio de propiedades de los dispositivos.

## Propuestas de mejora

El presente proyecto dispone de un gran potencial de mejora, con el objetivo de realizar una plataforma realmente útil para la gestión de dispositivos IoT. Las siguientes son algunas de las principales ideas que priorizaría a la hora de continuar con este proyecto:

- Conexión con dispositivos reales y otros protocolos de conexión (zigBee, LoraWan). De forma que la solución sea más flexible a la hora de integrarse con multitud de fabricantes.
- Mejorar la lógica de conexión con el dispositivo simulado, generando un `device_id` desde el dispositivo para poder conectarlo con la base de datos de una forma adecuada y similar al trabajo con dispositivos reales.
- Ampliación de las opciones de las aplicaciones web y móvil, como una función de editar dispositivo que permita modificar los valores seleccionados para un dispositivo existente.
- Diseño de la UI de la aplicación móvil con una paleta de colores, selección de fuentes, estilos, etc.
- La comunicación entre los distintos módulos de python, ya que el intérprete no disponía de las herramientas para entender que el código que estaba importando era un módulo.

## Pruebas

### Aplicación web

1. **Ordenar la pantalla de dispositivos por diferentes criterios**

## Dispositivos IoT

Añadir Dispositivo +

ID  Ascendente  Ordenar

### Lista de Dispositivos



#### Termómetro entrada

Tipo: Sensor Temperatura

Estado: off

Temperatura: 29.1°C

Batería: 42%

Presión: null hPa

Humedad: 51.23%

Last update: Mon, 17 Mar  
2025 11:27:42 GMT%

Encender Eliminar



#### Barometro Jardín

Tipo: Sensor de Presión

Estado: on

Temperatura: null°C

Batería: 0%

Presión: null hPa

Humedad: null%

Last update: Wed, 12 Mar  
2025 16:36:56 GMT%

Apagar Eliminar



#### Luz Pasillo

Tipo: Iluminación

Estado: on

Temperatura: null°C

Batería: 58%

Presión: null hPa

Humedad: null%

Last update: Mon, 17 Mar  
2025 11:15:40 GMT%

Apagar Eliminar



#### Sensor de presion de la terraza

Tipo: Sensor de Presión

Estado: off

Temperatura: null°C

Batería: 13%

Presión: null hPa

Humedad: null%

Last update: Mon, 17 Mar  
2025 10:50:18 GMT%

Encender Eliminar



#### Estacion del hall

Tipo: Sensor de temperatur

Estado: on

Temperatura: null°C

Batería: 100%

Presión: null hPa

Humedad: null%

Last update: Mon, 17 Mar  
2025 11:48:09 GMT%

Apagar Eliminar



#### Estacion del hall

Tipo: Sensor de temperatur

Estado: ON

Temperatura: null°C

Batería: 100%

Presión: null hPa

Humedad: null%

Last update: Mon, 17 Mar  
2025 11:47:57 GMT%

Encender Eliminar



Tipo


▼

Descendente

▼

Ordenar

### Lista de Dispositivos



**Termómetro entrada**

Tipo: Sensor Temperatura

Estado: off

Temperatura: 29.11°C


Batería: 42%

Presión: null hPa

Humedad: 51.23%

Last update: Mon, 17 Mar 2025 11:27:42 GMT%

Encender Eliminar



**Estacion del hall**

Tipo: Sensor de temperatur

Estado: on

Temperatura: null°C


Batería: 100%

Presión: null hPa

Humedad: null%

Last update: Mon, 17 Mar 2025 11:48:09 GMT%

Apagar Eliminar



**Estacion del hall**

Tipo: Sensor de temperatur

Estado: ON

Temperatura: null°C


Batería: 100%

Presión: null hPa

Humedad: null%

Last update: Mon, 17 Mar 2025 11:47:57 GMT%

Encender Eliminar



**Dispositivo de prueba**

Tipo: Sensor de temperatur

Estado: ON

Temperatura: null°C


Batería: 100%

Presión: null hPa

Humedad: null%

Last update: Mon, 17 Mar 2025 11:51:16 GMT%

Encender Eliminar



**Barometro Jardín**

Tipo: Sensor de Presión

Estado: on

Temperatura: null°C


Batería: 0%

Presión: null hPa

Humedad: null%

Last update: Wed, 12 Mar 2025 16:36:56 GMT%

Apagar Eliminar



**Sensor de presion de la terraza**

Tipo: Sensor de Presión

Estado: off

Temperatura: null°C

Batería: 13%

Presión: null hPa


Humedad: null%

Last update: Mon, 17 Mar 2025 10:50:18 GMT%

Encender Eliminar


Estado
Descendente
Ordenar

### Lista de Dispositivos




**Barometro Jardín**  
Tipo: Sensor de Presión  
Estado: on  
Temperatura: null°C  
Batería: 0%  
Presión: null hPa  
Humedad: null%  
Last update: Wed, 12 Mar 2025 16:36:56 GMT%

Apagar
Eliminar




**Luz Pasillo**  
Tipo: Iluminación  
Estado: on  
Temperatura: null°C  
Batería: 58%  
Presión: null hPa  
Humedad: null%  
Last update: Mon, 17 Mar 2025 11:15:40 GMT%

Apagar
Eliminar




**Estacion del hall**  
Tipo: Sensor de temperatur  
Estado: on  
Temperatura: null°C  
Batería: 100%  
Presión: null hPa  
Humedad: null%  
Last update: Mon, 17 Mar 2025 11:48:09 GMT%

Apagar
Eliminar




**Estacion del hall**  
Tipo: Sensor de temperatur  
Estado: ON  
Temperatura: null°C  
Batería: 100%  
Presión: null hPa  
Humedad: null%  
Last update: Mon, 17 Mar 2025 11:47:57 GMT%

Encender
Eliminar



**Dispositivo de prueba**  
Tipo: Sensor de temperatur  
Estado: ON  
Temperatura: null°C  
Batería: 100%  
Presión: null hPa  
Humedad: null%  
Last update: Mon, 17 Mar 2025 11:51:16 GMT%


Encender
Eliminar



**Termómetro entrada**  
Tipo: Sensor Temperatura  
Estado: off  
Temperatura: 29.11°C  
Batería: 42%  
Presión: null hPa  
Humedad: 51.23%  
Last update: Mon, 17 Mar 2025 11:27:42 GMT%

Encender
Eliminar

- Comprobar que se actualiza la información de un dispositivo al recibir mensaje del simulador de estación meteorológica



**Termómetro entrada**  
Tipo: Sensor Temperatura  
Estado: off  
Temperatura: 29.11°C  
Batería: 42%  
Presión: null hPa  
Humedad: 51.23%  
Last update: Mon, 17 Mar 2025 11:27:42 GMT%

Encender
Eliminar

```
Actualizado:  
Enviando al tópico MQTT wokwi-weather: {"temp": 28.31, "humidity": 48.79}  
Mensaje enviado correctamente  
Mensaje 2 publicado exitosamente
```



### Termómetro entrada

**Tipo:** Sensor Temperatura

**Estado:** off

**Temperatura:** 28.31°C

**Batería:** 42%

**Presión:** null hPa

**Humedad:** 48.79%

**Last update:** Mon, 17 Mar  
2025 15:02:30 GMT%

EncenderEliminar

### 3. Crear nuevo dispositivo y comprobarlo en el listado

## Añadir Dispositivo

**Nombre del Dispositivo:**

Sensor presión jardín

**Tipo de Dispositivo:**

Iluminación

Sensor de temperatura

Sensor de movimiento

Sensor de presión

**Estado:**

☒

Agregar Dispositivo

Volver al listado



Dispositivo agregado con éxito

Volver al listado



### Sensor presión jardín

**Tipo:** Sensor de Presión

**Estado:** on

**Temperatura:** null°C

**Batería:** 23%

**Presión:** null hPa

**Humedad:** null%

**Last update:** Mon, 17 Mar  
2025 15:04:25 GMT%

Apagar

Eliminar

#### 4. Borrar dispositivo

localhost:3000 dice

¿Seguro que deseas eliminar este dispositivo?

Aceptar

Cancelar

Apagar

Eliminar

Encender



**Dispositivo de prueba**

Tipo: Sensor de temperatur

Estado: ON

Temperatura: null°C

Batería: 100%

Presión: null hPa

Humedad: null%

Last update: Mon, 17 Mar 2025 11:51:16 GMT%

Encender

Eliminar



**Sensor presión jardín**

Tipo: Sensor de Presión

Estado: on

Temperatura: null°C

Batería: 23%

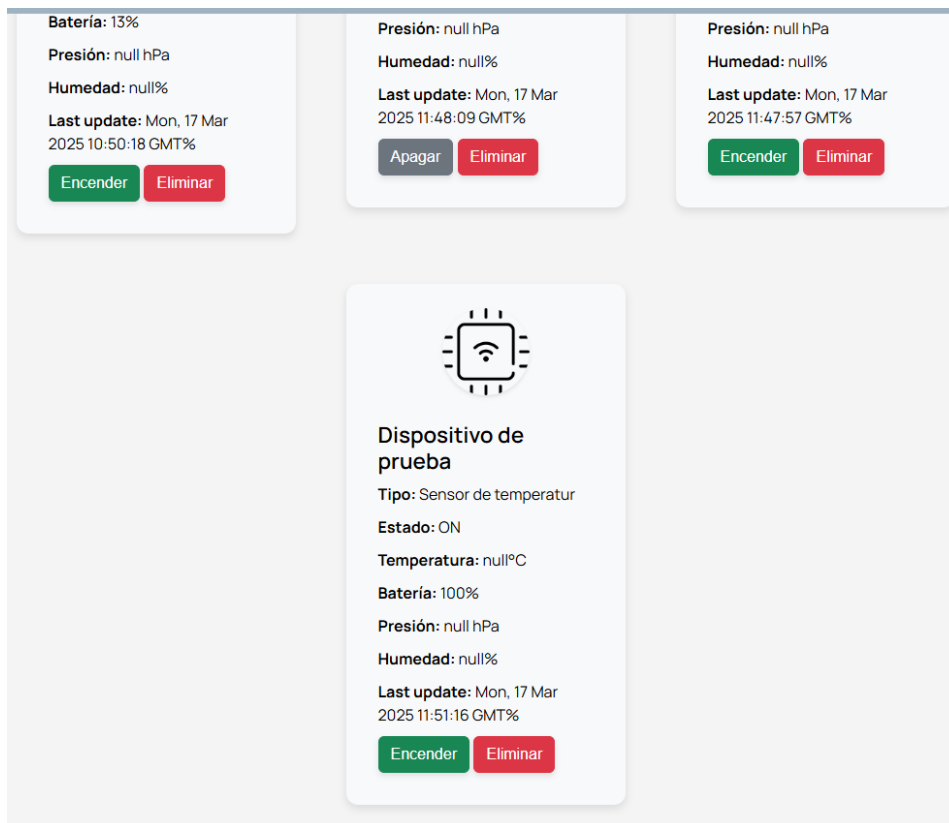
Presión: null hPa

Humedad: null%

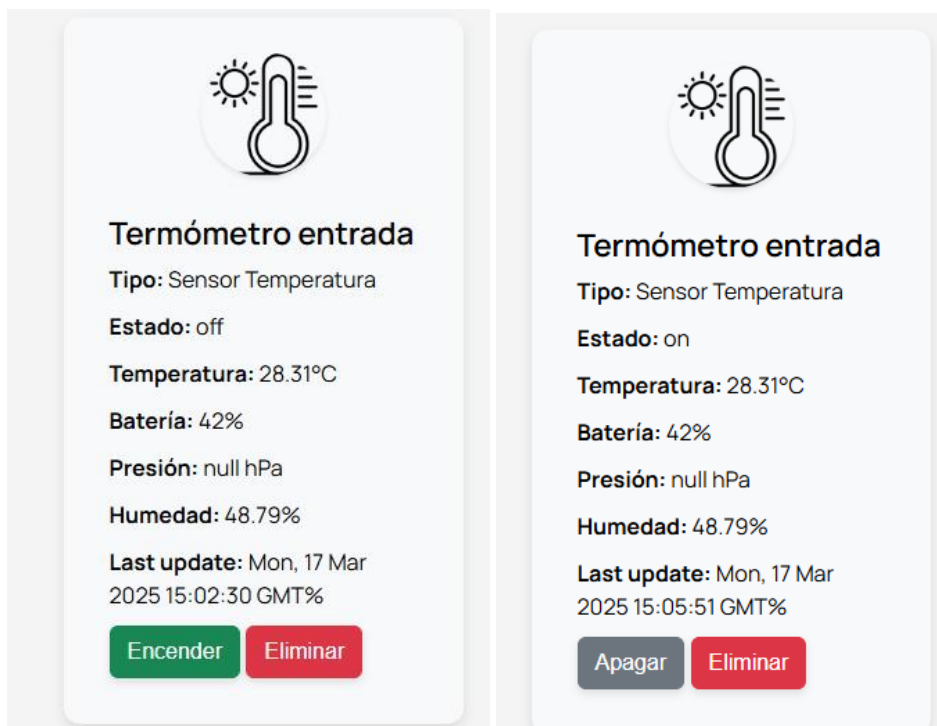
Last update: Mon, 17 Mar 2025 15:04:25 GMT%

Apagar

Eliminar



## 5. Cambiar estado del dispositivo



## Aplicación móvil

1. Comprobar que se actualiza la información de un dispositivo al recibir mensaje del simulador de estación meteorológica

Nombre: Termómetro entrada  
ID: 545033 - Sensor Temperatura  
Estado: on  
Batería: 42%  
Temperatura: 28.31°C  
Humedad: 48.79%  
Presión: N/A hPa  
Último movimiento: N/A  
Última actualización: Mon, 17 Mar 2025  
15:05:51 GMT

Cambiar Estado

Eliminar

```
Enviando al tópico MQTT wokwi-weather: {"temp": 20.16, "humidity": 58.3}  
Mensaje enviado correctamente  
Mensaje 3 publicado exitosamente  
Conectado exitosamente al broker MQTT  
Publicando datos en el tópico wokwi-weather
```

Nombre: Termómetro entrada  
ID: 545033 - Sensor Temperatura  
Estado: on  
Batería: 42%  
Temperatura: 20.16°C  
Humedad: 58.3%  
Presión: N/A hPa  
Último movimiento: N/A  
Última actualización: Mon, 17 Mar 2025  
15:16:29 GMT

Cambiar Estado

Eliminar

## 2. Crear nuevo dispositivo y comprobarlo en el listado

Nombre del dispositivo

Tipo de dispositivo  
Sensor temperatura

Estado: ON OFF

Cancelar Guardar

← Añadir Dispositivo

Nombre del dispositivo  
Dispositivo prueba app movil

Tipo de dispositivo  
Sensor humedad

Estado: ON OFF

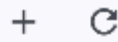
Cancelar Guardar

Nombre: Dispositivo prueba app movil  
ID: 545045 - Sensor humedad  
Estado: ON  
Batería: 100%  
Temperatura: N/A°C  
Humedad: N/A%  
Presión: N/A hPa  
Último movimiento: N/A  
Última actualización: Mon, 17 Mar 2025  
15:17:54 GMT

Cambiar Estado Eliminar

### **3. Borrar dispositivo**

## Lista de Dispositivos



Nombre: Dispositivo de prueba  
ID: 545043 - Sensor de temperatur  
Estado: ON  
Batería: 100%  
Temperatura: N/A°C  
Humedad: N/A%  
Presión: N/A hPa  
Último movimiento: N/A  
Últi

11: **Confirmar eliminación**



¿Realmente desea borrar el dispositivo?

Cancelar

Sí, eliminar

Noi  
ID: 545043 - Sensor humedad  
Estado: ON  
Batería: 100%  
Temperatura: N/A°C  
Humedad: N/A%  
Presión: N/A hPa  
Último movimiento: N/A  
Última actualización: Mon, 17 Mar 2025  
15:17:54 GMT

Cambiar Estado

Eliminar

Lista de Dispositivos

+

↺

Nombre: Estacion del hall

ID: 545042 - Sensor de temperatur

Estado: ON

Batería: 100%

Temperatura: N/A°C

Humedad: N/A%

Presión: N/A hPa

Último movimiento: N/A

Última actualización: Mon, 17 Mar 2025 11:47:57 GMT

Cambiar Estado

Eliminar

Nombre: Dispositivo de prueba

ID: 545043 - Sensor de temperatur

Estado: ON

Batería: 100%

Temperatura: N/A°C

Humedad: N/A%

Presión: N/A hPa

Último movimiento: N/A

Última actualización: Mon, 17 Mar 2025 11:51:16 GMT

Cambiar Estado

Eliminar

#### 4. Cambiar estado del dispositivo



Nombre: Termómetro entrada  
ID: 545033 - Sensor Temperatura  
Estado: on  
Batería: 42%  
Temperatura: 20.16°C  
Humedad: 58.3%  
Presión: N/A hPa  
Último movimiento: N/A  
Última actualización: Mon, 17 Mar 2025  
15:16:29 GMT

Cambiar Estado

Eliminar

Nombre: Termómetro entrada  
ID: 545033 - Sensor Temperatura  
Estado: off  
Batería: 42%  
Temperatura: 20.16°C  
Humedad: 58.3%  
Presión: N/A hPa  
Último movimiento: N/A  
Última actualización: Mon, 17 Mar 2025  
15:18:50 GMT

Cambiar Estado

Eliminar

### Comunicación entre plataformas

1. **Cambiar de estado un dispositivo desde la app móvil y visualizar en la web**



## Termómetro entrada

**Tipo:** Sensor Temperatura

**Estado:** on

**Temperatura:** 28.31°C

**Batería:** 42%

**Presión:** null hPa

**Humedad:** 48.79%

**Last update:** Mon, 17 Mar  
2025 15:05:51 GMT%

Apagar

Eliminar

Nombre: Termómetro entrada

ID: 545033 - Sensor Temperatura

Estado: off

Batería: 42%

Temperatura: 20.16°C

Humedad: 58.3%

Presión: N/A hPa

Último movimiento: N/A

Última actualización: Mon, 17 Mar 2025  
15:18:50 GMT

Cambiar Estado

Eliminar



## Termómetro entrada

**Tipo:** Sensor Temperatura

**Estado:** off

**Temperatura:** 20.16°C

**Batería:** 42%

**Presión:** null hPa

**Humedad:** 58.30%

**Last update:** Mon, 17 Mar  
2025 15:18:50 GMT%

Encender

Eliminar

2. Cambiar de estado un dispositivo desde la web y visualizar en app móvil

Nombre: Barometro Jardín  
ID: 545034 - Sensor de Presión  
Estado: on  
Batería: 0%  
Temperatura: N/A°C  
Humedad: N/A%  
Presión: N/A hPa  
Último movimiento: N/A  
Última actualización: Wed, 12 Mar 2025  
16:36:56 GMT

Cambiar Estado

Eliminar



## Barometro Jardín

**Tipo:** Sensor de Presión

**Estado:** off

**Temperatura:** null°C

**Batería:** 0%

**Presión:** null hPa

**Humedad:** null%

**Last update:** Mon, 17 Mar  
2025 15:19:54 GMT%

Encender

Eliminar

Nombre: Barometro Jardín  
ID: 545034 - Sensor de Presión  
Estado: off  
Batería: 0%  
Temperatura: N/A°C  
Humedad: N/A%  
Presión: N/A hPa  
Último movimiento: N/A  
Última actualización: Mon, 17 Mar 2025  
15:19:54 GMT

Cambiar Estado

Eliminar

## Bibliografía

- <https://bitybyte.github.io/Organizando-codigo-Python/>
- <https://www.ibm.com/es-es/topics/rest-apis>
- <https://stackoverflow.com/questions/338768/python-error-importerror-no-module-named>
- <https://mailchimp.com/es/resources/rest-api/>
- <https://stackoverflow.com/questions/56580182/how-to-load-xml-in-to-the-draw-io-diagram>
- <https://stackoverflow.com/questions/51969662/run-multiple-python-scripts-at-the-same-time>
- <https://wokwi.com/>