

Trabajo Fin de Grado en Ingeniería de las Tecnologías de Telecomunicación

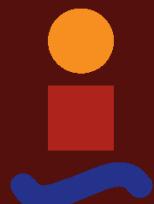
Control domótico de un sistema de riego, mediante
OpenHAB y Fiware, para el ahorro del consumo de
agua y energía

Autor: José Antonio Pozo Prior

Tutor: María Teresa Ariza Gómez

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2025



Trabajo Fin de Grado
en Ingeniería de las Tecnologías de Telecomunicación

Control domótico de un sistema de riego, mediante OpenHAB y Fiware, para el ahorro del consumo de agua y energía

Autor:

José Antonio Pozo Prior

Tutor:

María Teresa Ariza Gómez

Profesor titular

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2025

Trabajo Fin de Grado: Control domótico de un sistema de riego, mediante OpenHAB y Fiware, para el ahorro del consumo de agua y energía

Autor: José Antonio Pozo Prior

Tutor: María Teresa Ariza Gómez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2025

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

En primer lugar, me gustaría agradecer a mis padres y a mi hermana por todo el apoyo que he recibido de su parte durante mi carrera universitaria, que tanto en los buenos como en los malos momentos me han escuchado y dado su consejo.

También me gustaría agradecer a mi tutora del proyecto, María Teresa, por la ayuda prestada y por la cercanía que me ha demostrado a lo largo de su desarrollo.

Por último, deseo recordar a todos los miembros de mi familia que nos han dejado y agradecerles por los momentos que he vivido junto a ellos.

José Antonio Pozo Prior

Sevilla, 2025

Resumen

Los invernaderos actuales buscan un equilibrio entre productividad y eficiencia energética. Para lograrlo, este proyecto utiliza el Internet de las Cosas (IoT), un concepto que permite conectar objetos cotidianos a Internet para monitorizarlos y controlarlos de forma remota, para optimizar ese balance mediante una aplicación Android desarrollada para controlar distintos elementos (como sistemas de riego y sensores) y consultar en tiempo real los precios horarios de la electricidad. Esta información permite automatizar decisiones como, por ejemplo, activar equipos en horas de menor coste.

Entre las tecnologías usadas en el proyecto destacan OpenHAB, núcleo para la integración de dispositivos IoT y Fiware Orion responsable del intercambio de mensajes entre los elementos del sistema mediante suscripciones que permiten que cuando se actualicen los valores de un elemento los componentes del sistema reciban esta nueva información.

Por último, cabe destacar que se ha usado una placa ESP32 programada mediante el IDE de Arduino para la gestión de los distintos elementos del invernadero

Abstract

Today's greenhouses strive for a balance between productivity and energy efficiency. To achieve this, this project uses the Internet of Things (IoT), a concept that allows everyday objects to be connected to the internet for remote monitoring and control. This project optimizes this balance through an Android app developed to control various elements (such as irrigation systems and sensors) and view real-time hourly electricity prices. This information allows for automated decisions such as activating equipment during off-peak hours.

Among the technologies used in the project are OpenHAB, a core for integrating IoT devices, and Fiware Orion, responsible for exchanging messages between system elements through subscriptions. This allows system components to receive this new information when the values of an element are updated.

Finally, it is worth noting that an ESP32 board programmed using the Arduino IDE was used to manage the various greenhouse elements.

Índice

Agradecimientos	ix
Resumen	xi
Abstract	xiii
Índice	xiv
Índice de Tablas	xvii
Índice de Figuras	xviii
Notación	xxii
1 Introducción	1
1.1 <i>Motivación</i>	1
1.2 <i>Objetivos Generales</i>	1
1.2.1 Objetivos Específicos	2
1.3 <i>Funcionalidad</i>	3
1.4 <i>Esquema de la Arquitectura</i>	3
1.5 <i>Antecedentes</i>	4
1.6 <i>Estructura de la memoria</i>	4
2 Recursos utilizados	7
2.1 <i>Recursos Hardware</i>	7
2.1.1 Ordenador portátil	7
2.1.2 Placa ESP32	8
2.1.3 Punto de Acceso WIFI	9
2.1.4 Teléfono Móvil	9
2.1.5 Placa Protoboard	9
2.1.6 Módulo DTH11	10
2.1.7 Módulo MH del sensor de luz	10
2.1.8 Módulo de la humedad del suelo (HW-80 + HW-103)	11
2.1.9 Módulo de nivel del agua	12
2.1.10 Módulo del relé	12

2.1.11	Porta pilas 4xAA	13
2.1.12	Pilas	13
2.1.13	Motor de agua	13
2.1.14	Leds, cables y resistencias	14
2.1.15	Multímetro	14
2.1.16	Soldador Eléctrico	14
2.2	<i>Recursos Software</i>	15
2.2.1	OpenHAB	15
2.2.2	Fiware Orion Context Broker	15
2.2.3	MongoDB	15
2.2.4	Docker	16
2.2.5	Máquina Virtual/Linux	16
2.2.6	Spring	16
2.2.7	Android Studio	17
2.2.8	PostgreSQL	17
2.2.9	Arduino	17
2.2.10	Windows 11 Pro	17
2.3	<i>Tecnologías usadas</i>	18
2.3.1	C++	18
2.3.2	JSON	18
2.3.3	REST	18
2.3.4	SQL	19
2.3.5	HTTP	19
2.3.6	Java	19
3	Dispositivo y Context Broker	21
3.1	<i>Dispositivo de Arduino</i>	21
3.1.1	Placa ESP32	21
3.1.2	Sensores y actuadores	22
3.1.3	Programa del dispositivo	22
3.2	<i>Context Broker</i>	27
3.2.1	Entidades	28
3.2.2	Suscripciones	29
4	Servidor Web y Base de Datos	33
4.1	<i>Clases</i>	33
4.1.1	Principal	33
4.1.2	Configuración	33
4.1.3	Modelos	33
4.1.4	Repositorios	36
4.1.5	Servicios	37
4.1.6	Controladores	40
4.2	<i>Ficheros de configuración</i>	43
4.3	<i>BBDD</i>	43
4.3.1	Tabla usuarios	43
4.3.2	Tabla límites	43
4.3.3	Tabla límites_horas_riego	44
4.3.4	Tabla lecturas_sensores	44
4.3.5	Tabla estado_motor	44
4.3.6	Tabla eventos_motor	45
5	Interfaz de usuario y funcionalidad	47
5.1	<i>Aplicación Android</i>	47
5.1.1	Inicio de sesión	48
5.1.2	Registro	49
5.1.3	Consultar valores actuales	50

5.1.4	Consultar registros de los sensores	51
5.1.5	Consultar registro del motor	52
5.1.6	Consultar perfil	53
5.1.7	Establecer límites	54
5.1.8	Consultar precio de la luz	55
5.1.9	Consultar pronóstico del tiempo	56
5.1.10	Controlar actuadores	57
5.1.11	Ficheros de configuración	58
5.2	<i>Aplicación Openhab</i>	58
5.3	<i>Resumen</i>	62
6	Conclusiones y líneas futuras	63
Anexo A: Instalación de Fiware Orion usando Docker		64
Anexo B: Configuración del ESP32		66
Anexo C: Configuración de la base de datos		69
Anexo D: Construcción del sistema		72
Referencias		75

ÍNDICE DE TABLAS

Tabla 2-1. Características del Lenovo Legion 5 Pro	7
Tabla 2-2. Características del ESP32	8
Tabla 2-3. Características del módulo DTH11	10
Tabla 2-4. Características del módulo MH del sensor de luz	10
Tabla 2-5. Características del módulo de la humedad del suelo	11
Tabla 2-6. Características del módulo de nivel de agua	12
Tabla 2-7. Características del módulo del relé	12
Tabla 2-8. Características del motor de agua	13
Tabla 4-1. Peticiones respondidas por EstadoMotorController	40
Tabla 4-2. Peticiones respondidas por LimitesController	40
Tabla 4-3. Peticiones respondidas por LecturasSensoresController	41
Tabla 4-4. Peticiones respondidas por UsuariosController	41
Tabla 5-1. Ítems de los sensores	59
Tabla 5-2. Ítems de los límites	59
Tabla 5-3. Ítems de los actuadores	60

ÍNDICE DE FIGURAS

Figura 1-1. Esquema de la arquitectura del Sistema	3
Figura 2-1. Lenovo Legion Pro 5	8
Figura 2-2. Placa ESP32	8
Figura 2-3. Punto de Acceso Numcom	9
Figura 2-4. Redmi Note 13	9
Figura 2-5. Placa Protoboard	9
Figura 2-6. Módulo DTH11	10
Figura 2-7. Módulo MH del sensor de luz	11
Figura 2-8. Módulo de la humedad del suelo	11
Figura 2-9. Módulo de nivel de agua	12
Figura 2-10. Módulo del relé	12
Figura 2-11. Porta pilas 4xAA	13
Figura 2-12. Pilas AA	13
Figura 2-13. Motor de agua	13
Figura 2-14. Resistencias, leds y cables	14
Figura 2-15. Multímetro	14
Figura 2-16. Soldador Eléctrico	14
Figura 2-17. OpenHAB	15
Figura 2-18. Fiware Orion	15
Figura 2-19. MongoDB	15
Figura 2-20. Docker	16
Figura 2-21. Máquina Virtual y Ubuntu	16
Figura 2-22. Spring	16
Figura 2-23. Android Studio	17
Figura 2-24. PostgreSQL	17
Figura 2-25. Arduino	17
Figura 2-26. Windows 11 Pro	17
Figura 2-27. C++	18
Figura 2-28. JSON	18
Figura 2-29. REST	18
Figura 2-30. SQL	19
Figura 2-31. HTTP	19
Figura 2-32. Java	19
Figura 3-1. Pines ESP32	22

Figura 3-2. Configuración del Wifi	22
Figura 3-3. Enviar los datos de los sensores	23
Figura 3-4. Función para obtener los límites	25
Figura 3-5. Función de notificación del estado del aire acondicionado	26
Figura 3-6. Bloque de depuración	27
Figura 3-7. Entidades Fiware	29
Figura 3-8. Atributo de una entidad Fiware	29
Figura 3-9. Ejemplo de suscripción Fiware	30
Figura 4-1. Clase tipo modelo (EstadoMotor)	35
Figura 4-2. Clase tipo repositorio (EventosMotorRepository)	37
Figura 4-3. Clases de interfaz e implementacion (EstadoMotorService y EstadoMotorServiceImpl)	39
Figura 4-4. Documentación de Swagger	42
Figura 4-5. Tabla usuarios	43
Figura 4-6. Tabla limites	44
Figura 4-7. Tabla limites_horas_riegos	44
Figura 4-8. Tabla lecturas_sensores	44
Figura 4-9. Tabla estado_motor	45
Figura 4-10. Tabla eventos_motor	45
Figura 4-11. Diagrama ER	45
Figura 5-1. Diagrama de casos de uso	47
Figura 5-2. Inicio de sesión	48
Figura 5-3. Barra de aplicación	49
Figura 5-4. Registro	50
Figura 5-5. Consultar valores actuales	51
Figura 5-6. Consultar registro de los sensores	52
Figura 5-7. Consultar registro del motor	53
Figura 5-8. Consultar perfil	54
Figura 5-9. Consultar límites	55
Figura 5-10. Consultar precio de la luz	56
Figura 5-11. Consultar pronóstico del tiempo	57
Figura 5-12. Controlar actuadores	58
Figura 5-13. Librerías del fichero build.gradle	58
Figura 5-14. Ítems de OpenHAB	61
Figura 5-15. Reglas de OpenHAB	62
Figura 0-1. Descargar Docker	64
Figura 0-2. Contenedor apagado	65
Figura 0-3. Contenedor encendido	65
Figura 0-1. Descarga de la versión de Arduino	66
Figura 0-2. Configuración de la Url de Arduino	66

Figura 0-3. Gestor de placas	67
Figura 0-4. Montaje del sistema	67
Figura 0-5. Esquema del sistema	68
Figura 0-1. Nuevo proyecto Android Studio.	72
Figura 0-2. Inicio del Servidor Rest	73
Figura 0-3. Instalar OpenHAB desde la página oficial	73
Figura 0-4 Iniciar OpenHab	74

Notación

AgIoT	Agricultural Internet of Things
OpenHAB	Open Home Automation Bus
JSON	JavaScript Object Notation
IoT	Internet of Things
SQL	Structured Query Language
REST	Representational State Transfer
BBDD	Base de Datos
PC	Personal computer
USB	Universal Serial Bus
UART	Universal Asynchronous Receiver/Transmitter
POJO	Plain Old Java Objects
JPA	Java Persistence API
API	Application Programming Interfaces
FK	Foreign Key
JDBC	Java Database Connectivity
UTF-8	8-bit Unicode Transformation Format
UTC	Coordinated Universal Time
DAL	Data Access Layer
REE	Red Eléctrica Española

1 INTRODUCCIÓN

Antes de ser un dragón, hay que sufrir como una hormiga.

- Proverbio chino-

1.1 Motivación

Cuando mencionamos Internet de las Cosas (IoT) nos estamos refiriendo a una red de elementos físicos que están integrados con sensores, software y conectividad de red, de forma que pueden recopilar y compartir datos [1]. Los elementos pueden ir desde bombillas a robots lo que permite una gran diversidad de casos para los que se puede usar el IoT [2]. Esta diversidad ha provocado que el IoT se divida en distintas ramas, en las que destacan:

- IoT Industrial (IIoT): Aplicado a maquinaria y procesos automatizados.
- IoT de Consumo: Aplicado en la domótica y dispositivos inteligentes para el hogar.
- IoT en Agricultura (AgIoT): Enfocado en monitoreo ambiental, riego inteligente y gestión automatizada de cultivos. Una rama muy importante que para 2026, se espera que el mercado global de IoT agrícola supere los \$ 18.1 mil millones [3]

Este proyecto se centra en el **AgIoT**, específicamente en la automatización de invernaderos enfocado en el ahorro energético, donde sensores y actuadores trabajan en conjunto para mantener las condiciones óptimas de crecimiento sin intervención humana constante y reducir el consumo de energía al máximo posible. Además, para mejorar el beneficio de las empresas dedicadas a la agricultura este proyecto también permite obtener el precio de la energía de manera sencilla para que se pueda decidir cuándo entrarán en uso los actuadores.

1.2 Objetivos Generales

Este trabajo tiene como objetivos:

- Analizar el funcionamiento de OpenHAB.
- Usar el context broker Fiware Orion para conectar las tecnologías usadas en el proyecto.
- Mostrar el clima actual.
- Mostrar el precio de la luz actual.
- Instalar sensores y actuadores con Arduino y comunicar al broker Fiware.
- Crear un servidor REST.
- Crear un cliente móvil.
- Conectar la base de datos con el servidor REST.

1.2.1 Objetivos Específicos

- ❖ Leer los datos procedentes de los sensores y enviar las órdenes a los actuadores de forma automatizada. Para ello se leerán los datos de los sensores del Arduino, se compararán con los límites impuestos por el usuario y se enviarán órdenes que controlarán los actuadores. Se usan los siguientes elementos:
 - Sensores
 - Sensor DTH11: para medir la temperatura y la humedad del ambiente.
 - Sensor de luminosidad: para medir la luz ambiental.
 - Sensor de nivel resistivo: para medir el nivel del agua.
 - Sensor de humedad del suelo: para medir la humedad en la tierra donde están las plantas.
 - Actuadores
 - Motor de agua: Es el motor de agua usado para la extracción del agua del depósito.
 - Dos Leds: se utilizarán dos Leds, un led que funciona como una fuente de luz para el sensor de luminosidad simulando un foco y otro led simulando un aire acondicionado.
- ❖ Configurar la placa ESP32: instalar y configurar la placa ESP32 que controlará los actuadores y leerá los sensores antes mencionados dependiendo de los parámetros configurados por el usuario.
- ❖ Conectar la placa ESP32 y Fiware Orion: se configurará la placa ESP32 para que pueda recibir y enviar información a Fiware Orion.
- ❖ Conectar OpenHAB y Fiware Orion: se configurará OpenHAB para poder intercambiar información, ya sea los valores leídos por los sensores o los límites impuestos por el usuario, con Fiware Orion.
- ❖ Instalar y configurar OpenHAB: se instalará el programa para la gestión automatizada del invernadero, configurando entre otras cosas reglas personalizadas.
- ❖ Crear y configurar un servidor REST: creado con SPRING, el servidor REST se encargará de:
 - El registro de los usuarios: para ello guardará en la base de datos PostgreSQL la información de usuario que reciba del cliente.
 - La autenticación de los usuarios: para ello se comprobará en la base de datos que exista un usuario y contraseña que coincida con los introducidos por el cliente.
 - Consulta de las peticiones del cliente: será el servidor REST el que consulte todas las peticiones recibidas del cliente.
 - Actualizar la BBDD: será el servidor REST quien actualice la información de la base de datos recibida desde otros elementos del sistema.
- ❖ Conectar el servidor REST y Fiware Orion: se configurará el servidor REST para que pueda intercambiar información con Fiware Orion.
- ❖ Crear y configurar la aplicación móvil para el usuario: creada mediante Android Studio, la aplicación se comunica mediante la API de OpenWeather, con el servicio OpenWeather, y mediante la API de la red eléctrica española, con la REE, para obtener la previsión del tiempo y los precios por hora de la energía eléctrica respectivamente. Además, se conecta con el servidor REST y permite configurar los límites, visualizar los valores de los sensores, junto con sus registros y accionar los actuadores.
- ❖ Configurar Fiware Orion: para ello se ha usado un contenedor de DOCKER con la configuración de Fiware Orion para poder replicarlo y escalar en caso de que sea necesario.
- ❖ Conectar todos los elementos que hablan con Fiware Orion entre sí: para ello se ha hecho uso de la capacidad de Fiware Orion para mandar notificaciones a los elementos con los que está conectado.

1.3 Funcionalidad

De cara al usuario final, el Sistema diseñado ofrece las siguientes funcionalidades:

- ✓ Acceso a la aplicación para usuarios registrados.
- ✓ Registro para nuevos usuarios.
- ✓ Distinción de niveles de permisos de los usuarios.
- ✓ Consulta del clima.
- ✓ Consulta de los precios de la energía eléctrica para el día de la consulta.
- ✓ Consulta de los valores actuales de los sensores del invernadero.
- ✓ Consulta de los registros con el volumen de agua consumida durante el riego y la duración de este.
- ✓ Consulta de los registros de los valores de los sensores del invernadero.
- ✓ Gestión del estado del motor del agua.
- ✓ Gestión del estado de la fuente de luz.
- ✓ Gestión del estado del aire acondicionado.
- ✓ Consulta y modificación de los límites de la temperatura ambiental, la humedad ambiental, el nivel de luminosidad del invernadero, la humedad terrestre, las horas de riego permitidas, el nivel de agua mínimo para permitir el riego y el volumen de agua máximo gastado por sesión de riego.

1.4 Esquema de la Arquitectura

En la siguiente ilustración se muestra el diagrama completo del sistema del proyecto.

Como se puede observar, la gestión del invernadero por parte del cliente se puede hacer mediante dos elementos OpenHAB y la aplicación Android llamada “ETSI Natura”.

El servidor REST maneja todas las peticiones de la app y es el único que hace transacciones con la BBDD SQL. Dependiendo de la solicitud que reciba guardará la información en la base de datos y/o actualizará la nueva información en Fiware mediante el JSON correspondiente.

Fiware se comunica mediante suscripciones y notificaciones con OpenHAB, Arduino y el servidor REST para que todos tengan la misma información de todos los elementos del sistema (límites, valores de los sensores, estado de los actuadores). Esta información viaja en el formato JSON.

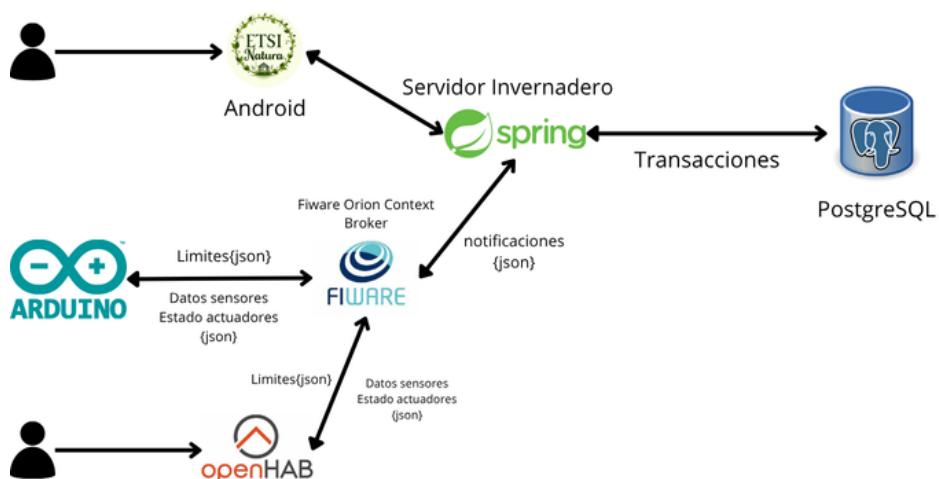


Figura 1-1. Esquema de la arquitectura del Sistema

1.5 Antecedentes

El proyecto realizado en 2023 por Jesús Vinuesa Serrano para su Trabajo de Fin de Grado sirvió de idea para este proyecto. En él se eligieron los sensores DTH11, el sensor de nivel de agua y el sensor de humedad del suelo, los actuadores: el motor de agua, dos leds y una bombilla Xiaomi y se integraron con un ESP32 para conformar un sistema IoT para la gestión de un invernadero. Esto se logró usando MQTT y Kafka, además el sistema permitía los comandos por voz.

En este proyecto se ha añadido el sensor de luminosidad, se ha quitado la bombilla Xiaomi. Además, se han cambiado las tecnologías MQTT y Kafka por el context broker Fiware Orion usando la característica de suscripciones/notificaciones. Por último, aunque ahora no reconoce comandos por voz la aplicación Android permite obtener la previsión climática del día y el precio de la energía eléctrica por hora.

1.6 Estructura de la memoria

Para facilitar la lectura del documento en este apartado se presenta un breve resumen de cada capítulo:

1. **Introducción:** Presentación del problema y la solución adoptada.
2. **Recursos utilizados:** Recursos software y hardware empleados para alojar la aplicación Android, OpenHAB, el servidor de base de datos y el context broker, así como para integrar los sensores y su comunicación con el resto del Sistema. Además, de las principales tecnologías empleadas para desarrollar el Proyecto.
3. **Dispositivo y Context Broker:** Análisis y características del dispositivo diseñado y del Context Broker Fiware Orion.
4. **Servidor Web y Base de Datos:** Estructura de la aplicación web, y análisis de la implementación y de la tecnología Spring utilizadas en el Proyecto.
5. **Interfaz de usuario y funcionalidad:** Descripción de las funcionalidades que ofrece la aplicación Android y OpenHAB de cara al usuario.
6. **Conclusiones y líneas futuras:** Ideas y características que podrían implementarse para mejorar el Sistema en próximas versiones de este.

Además de los nueve capítulos expuestos, se proporciona una serie de anexos que servirán como manual para reproducir el Sistema paso a paso:

Anexo A: Instalación de Fiware Orion utilizando Docker.

Anexo B: Configuración y conexión de sensores con ESP32.

Anexo C: Configuración del servidor de Base de Datos PostgreSQL.

Anexo D: Construcción del sistema.

2 RECURSOS UTILIZADOS

Cualquier tecnología suficientemente avanzada es indistinguible de la magia.

- Arthur C. Clarke -

En este capítulo se detallarán los recursos software y hardware que se han usado para la elaboración de este proyecto. Además, se detallarán las tecnologías usadas para la elaboración de este.

2.1 Recursos Hardware

2.1.1 Ordenador portátil

Se ha utilizado un ordenador portátil para crear y ejecutar todos los programas y servidores usados en el trabajo. En específico el ordenador ha sido un Lenovo Legion Pro 5 16ARX8. Con las siguientes características:

Componente	Características
Procesador	AMD Ryzen 7 7745HX
Almacenamiento	476 GB
RAM instalada	16 GB
Sistema Operativo	Windows 11 Pro
Tarjeta Gráfica	AMD Radeon (TM) 610M y NVIDIA GeForce RTX 4070

Tabla 2-1. Características del Lenovo Legion 5 Pro



Figura 2-1. Lenovo Legion Pro 5

2.1.2 Placa ESP32

Es la placa que maneja los sensores y actuadores, además, es compatible con Arduino IDE. Algunas de sus características técnicas son:

Categoría	Detalles
Microcontrolador	Xtensa® dual-core 32-bit LX6 (240 MHz)
Memoria Flash	4 MB
RAM	512 KB
Conectividad	Wi-Fi 802.11 b/g/n (2.4 GHz) + Bluetooth 4.2 (BLE y Classic)
Interfaces I/O	34 GPIO (12 bits ADC, 2 DAC, UART, SPI, I2C, PWM, etc.)
Protocolos Compatibles	MQTT, HTTP, TCP/IP, LoRa (con módulos externos)
Consumo	Modo sleep: ~5 µA / Activo: ~160 mA (Wi-Fi/BT en uso)
Tensión de Alimentación	3.3V (rango recomendado: 2.2V - 3.6V)
Rango de Temperatura	-40°C a +125°C (industrial)
Dimensiones	25 mm x 18 mm (ESP32-WROOM-32)
Cable de Alimentación	Tipo C

Tabla 2-2. Características del ESP32



Figura 2-2. Placa ESP32

2.1.3 Punto de Acceso WIFI

Punto de acceso WIFI que permite el establecimiento de la red local y de esta forma una comunicación entre los elementos software del proyecto. Específicamente ha sido el rúter nucom wireless 300mbps



Figura 2-3. Punto de Acceso Numcom

2.1.4 Teléfono Móvil

Es el dispositivo Android usado para la ejecución de la aplicación Android que permite al usuario la gestión y el control del invernadero. En este caso se ha usado el dispositivo simulado de Android Studio Pixel 4 XL API 35 y un dispositivo físico Android: Redmi Note 13.



Figura 2-4. Redmi Note 13

2.1.5 Placa Protoboard

La placa protoboard (o placa de pruebas) es un dispositivo en electrónica que permite montar y probar circuitos temporales sin necesidad de soldar. Está compuesta por una matriz de orificios interconectados eléctricamente mediante columnas, buses y filas, facilitando el rápido montaje de componentes como resistencias, leds y cables. En este proyecto se ha usado para unir los elementos del hardware con el ESP32.

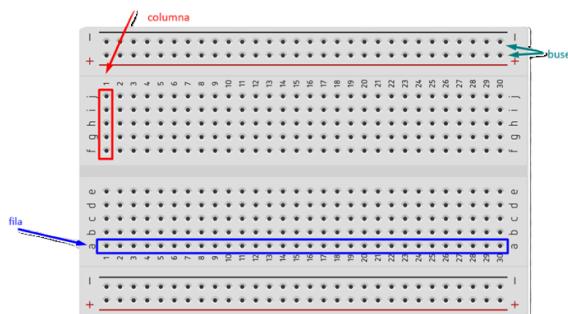


Figura 2-5. Placa Protoboard

2.1.6 Módulo DTH11

Contiene el sensor DTH11 que permite medir la humedad ambiental y temperatura con una precisión básica ($\pm 2^{\circ}\text{C}$ y $\pm 5\%$ HR). Utiliza un protocolo de comunicación serial de un solo cable y es ideal para proyectos IoT y domóticos donde no se requiera alta exactitud.

Parámetro	Especificación
Rango de Temperatura	0°C a 50°C ($\pm 2^{\circ}\text{C}$)
Rango de Humedad	20% a 80% HR ($\pm 5\%$)
Alimentación	3V a 5.5V DC
Salida	Señal digital
Uso típico	Invernaderos, estaciones meteorológicas, domótica

Tabla 2-3. Características del módulo DTH11

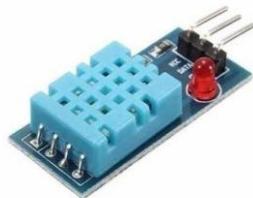


Figura 2-6. Módulo DTH11

2.1.7 Módulo MH del sensor de luz

Contiene un fotorresistor LDR (Light Dependent Resistor) que mide la intensidad lumínica ambiental. Su resistencia varía inversamente con la luz: disminuye en ambientes brillantes y aumenta en la oscuridad. Se usa comúnmente en sistemas automáticos por su bajo costo y fácil integración con microcontroladores.

Parámetro	Especificación
Rango de detección	0 a 100,000 lux
Voltaje operativo	3.3V a 5V DC
Salida	Analógica (ADC) o digital
Respuesta espectral	~550 nm (Similar al ojo humano)
Aplicaciones	Control de luces, alarmas, agricultura

Tabla 2-4. Características del módulo MH del sensor de luz



Figura 2-7. Módulo MH del sensor de luz

2.1.8 Módulo de la humedad del suelo (HW-80 + HW-103)

Está formado por el sensor HW-80 que mide la humedad del suelo mediante dos electrodos que detectan cambios en la conductividad eléctrica según el agua presente. Además, del convertidor HW-103 que transforma esta señal analógica en un valor digital (0-1023 ADC) para su lectura en los microcontroladores.

Parámetro	Especificación
Voltaje operativo	3.3V - 5V DC
Salida	Analógica (HW-80) / Digital (HW-103, ADC)
Rango de detección	0% (seco) - 100% (húmedo)
Corrosión	Electrodos no protegidos (vida útil limitada)
Aplicaciones	Riego automático, invernaderos, jardinería

Tabla 2-5. Características del módulo de la humedad del suelo

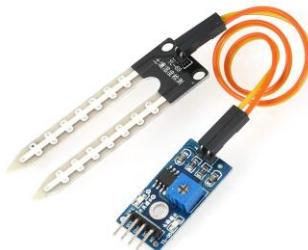


Figura 2-8. Módulo de la humedad del suelo

2.1.9 Módulo de nivel del agua

Está formado por un sensor de nivel de agua resistivo que detecta la presencia y el nivel del agua mediante unos electrodos expuestos, que miden los cambios en la conductividad del medio. Al entrar en contacto con el agua, se cierra el circuito entre los electrodos y se genera una señal analógica que proporciona el nivel. Como tiene los electrodos expuestos es muy susceptible a la corrosión.

Parámetro	Especificación
Voltaje operativo	3.3V - 5V DC
Salida	Señal analógica (0-1023 en ADC de Arduino)
Detección	Por contacto directo (electrodos sumergidos)
Material	Electrodos metálicos sin protección (cobre/acero)
Aplicaciones	Tanques, sistemas de riego, alarmas de inundación

Tabla 2-6. Características del módulo de nivel de agua



Figura 2-9. Módulo de nivel de agua

2.1.10 Módulo del relé

Es un interruptor electromecánico controlado por señales de bajo voltaje. Activa o desactiva circuitos mediante un electroimán que mueve un contacto interno.

Parámetro	Especificación
Voltaje de control	3.3V - 5V DC
Carga máxima	10A 250V AC / 10A 30V DC
Aplicaciones	Domótica, control de motores, sistemas IoT

Tabla 2-7. Características del módulo del relé



Figura 2-10. Módulo del relé

2.1.11 Porta pilas 4xAA

Porta pilas para 4 pilas AA que proporcionará el voltaje para el funcionamiento del motor de agua.



Figura 2-11. Porta pilas 4xAA

2.1.12 Pilas

Son la fuente de alimentación del motor de agua.



Figura 2-12. Pilas AA

2.1.13 Motor de agua

Es el encargado de suministrar el agua a las plantas del invernadero. El motor usado en este proyecto bombea agua a un flujo de 25 ml por segundo.

Parámetro	Especificación
Voltaje	DC3-5 V
Corriente	100-200 mA
Elevación	0,3-0,8 m
Flujo	1,2-1,6 l/min
Aplicaciones	Riego automático, fuentes decorativas

Tabla 2-8. Características del motor de agua



Figura 2-13. Motor de agua

2.1.14 Leds, cables y resistencias

Los leds son componentes que se iluminan cuando pasa corriente eléctrica en polaridad directa, es decir, desde el ánodo a + al cátodo a -.

Los cables conectan los leds a la fuente de alimentación, transmitiendo la corriente necesaria. Para evitar daños, se usan resistencias en serie con el led, con un valor calculado según la ley de Ohm, en este proyecto son de 220Ω pues la fuente ESP32 da 3,7V de alimentación.

Se han usado 3 tipos de cables para la realización de este proyecto, cables macho-hembra, cables macho-macho y cables hembra-hembra, que han permitido alargar las distancias entre los elementos.



Figura 2-14. Resistencias, leds y cables

2.1.15 Multímetro

Es un instrumento portátil que mide magnitudes eléctricas activas, como la tensión (V) y la corriente (A), y pasivas, como la resistencia (Ω). Funciona mediante puntas de prueba que se conectan al circuito.



Figura 2-15. Multímetro

2.1.16 Soldador Eléctrico

Es una herramienta que genera calor en su punta (entre 200°C y 450°C) al pasar corriente a través de una resistencia interna, fundiendo el estaño para unir componentes electrónicos. El calor se controla mediante una ruleta que indica la temperatura que va a alcanzar.



Figura 2-16. Soldador Eléctrico

2.2 Recursos Software

Se entiende como recursos software a los programas o componentes listos para usar que han facilitado el desarrollo, la gestión y la ejecución del proyecto.

2.2.1 OpenHAB

OpenHAB permite integrar los elementos IoT, los sensores, los actuadores y los límites establecidos por el usuario en una interfaz unificada. Además, permite personalizar reglas para la gestión de los elementos y es compatible con múltiples protocolos.



Figura 2-17. OpenHAB

2.2.2 Fiware Orion Context Broker

Es el agente que sirve de intermediario entre los elementos del sistema y que mediante notificaciones y suscripciones notifica siempre los últimos valores de las entidades al resto de componentes mediante mensajes JSON.



Figura 2-18. Fiware Orion

2.2.3 MongoDB

Es la base de datos que permite la persistencia de las últimas entidades que Fiware ha recibido y las mantiene para que cuando se reinicie el sistema Fiware mantenga las últimas entidades. Esto permite que el resto de los elementos puedan obtener de Fiware las últimas versiones de las entidades.



Figura 2-19. MongoDB

2.2.4 Docker

Ha servido para alojar a Fiware Orion junto a MongoDB mediante un contenedor de software que tiene los dos programas. Para ver su instalación diríjase al Anexo A: Instalación de Fiware Orion usando Docker.

Un contenedor es una unidad estándar de software que empaqueta el código y todas sus dependencias para que la aplicación se ejecute de forma rápida y fiable de un entorno informático a otro [4].

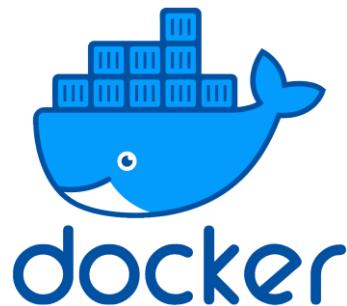


Figura 2-20. Docker

2.2.5 Máquina Virtual/Linux

Se ha usado la máquina virtual VMware con el OS Ubuntu para ejecutar sobre Linux el servidor de Spring y la base de datos SQL, esto permite el funcionamiento de gran parte del proyecto.



Figura 2-21. Máquina Virtual y Ubuntu

2.2.6 Spring

Es un framework Java para el desarrollo de aplicaciones, es de código Abierto y se ha usado para la creación del servidor REST.



Figura 2-22. Spring

2.2.7 Android Studio

Es el entorno de desarrollo oficial para aplicaciones Android y es el que se ha usado para la creación de la app de control del invernadero



Figura 2-23. Android Studio

2.2.8 PostgreSQL

Es una base de datos SQL que permite consultas complejas y es compatible con JSON por eso se ha usado para almacenar los valores que nos interesan de las entidades que monitorizamos.



Figura 2-24. PostgreSQL

2.2.9 Arduino

Es un IDE basado en C++ y se usa para la programación de microcontroladores. Concretamente, en este proyecto se ha usado el microcontrolador ESP32 y nos ha permitido programar las funciones que realizará el ESP32.



Figura 2-25. Arduino

2.2.10 Windows 11 Pro

Es el sistema operativo base donde se ha desarrollado el proyecto, pues en él se han instalado Docker, OpenHAB, Android Studio...



Figura 2-26. Windows 11 Pro

2.3 Tecnologías usadas

Se entiende como tecnologías a los lenguajes y a los protocolos que se han usado para la realización de este proyecto.

2.3.1 C++

Es un lenguaje de programación para desarrollo de sistemas. En este caso se ha usado el lenguaje de Arduino que está basado en este lenguaje.



Figura 2-27. C++

2.3.2 JSON

La tecnología JavaScript Object Notation es un formato ligero para el intercambio de datos, es la tecnología que han usado los recursos software para intercambiar las entidades entre sí.



Figura 2-28. JSON

2.3.3 REST

Es un estilo arquitectónico para sistemas distribuidos que utiliza los verbos HTTP (GET, POST, PUT, DELETE, etc.) como operaciones estándar. En el proyecto, las API siguen este paradigma mediante la identificación única de recursos a través de URIs (/invernadero/sensores, /invernadero/actuadores) y el intercambio de entidades en formato JSON



Figura 2-29. REST

2.3.4 SQL

El Structured Query Language es el lenguaje estándar para gestionar bases de datos relacionales. Este lenguaje permite CRUD (Create, Read, Update, Delete) sobre los datos. En el proyecto, PostgreSQL utiliza SQL para todas las operaciones de persistencia: almacenamiento, consulta, modificación y gestión de históricos de datos.



Figura 2-30. SQL

2.3.5 HTTP

El HyperText Transfer Protocol es un protocolo de la capa de aplicación que opera sobre TCP/IP para la comunicación entre clientes y servidores. Se ha utilizado para la comunicación entre la aplicación Android y el servidor REST.



Figura 2-31. HTTP

2.3.6 Java

Java es un lenguaje de programación orientado a objetos, multiplataforma y tipado estático que se ha empleado en este proyecto en dos componentes clave.

Como lenguaje principal para el desarrollo nativo de la aplicación Android, aprovechando su integración con el SDK de Android.

Como base para implementar el servidor REST mediante Spring Boot.



Figura 2-32. Java

3 DISPOSITIVO Y CONTEXT BROKER

¿Tanta tecnología para qué? Para mentes tan vacías.

-Pablo Hasél-

Tal y como se ha detallado en el apartado 1.4 Esquema de la arquitectura el sistema consta de 6 elementos, el dispositivo Arduino, el context broker, la aplicación Android, la aplicación OpenHAB, el servidor REST y la BBDD SQL.

En este capítulo vamos a profundizar en el funcionamiento del dispositivo de Arduino y el context broker.

3.1 Dispositivo de Arduino

El dispositivo de Arduino está compuesto por los siguientes elementos: una placa ESP32, varios sensores, un motor y dos leds. En las siguientes subsecciones se detallarán qué es cada componente y para qué se usan.

3.1.1 Placa ESP32

La placa ESP32 es un microprocesador cuyas características están definidas en el Capítulo 2, y es el elemento elegido como procesador lógico. Esto implica que es el encargado de leer y monitorizar los sensores, además de controlar los actuadores. Para lograr todo esto, se ejecutarán los códigos descritos en las siguientes subsecciones. Para conocer cómo se ha configurado, consultar el Anexo B: Configuración del ESP32.

El ESP32 tiene 38 pines, de los cuales 34 son de entrada/salida (llamados pines GPIO). Estos pines funcionan a 3.3 V, por lo que aplicar más de 5 V podría dañarlos.

Dispone de un conector USB-C que mediante comunicación serie se conecta al PC. Para usar la placa es necesario instalar el controlador del ESP32, el CH340 que se instalará automáticamente al conectar el USB al PC.

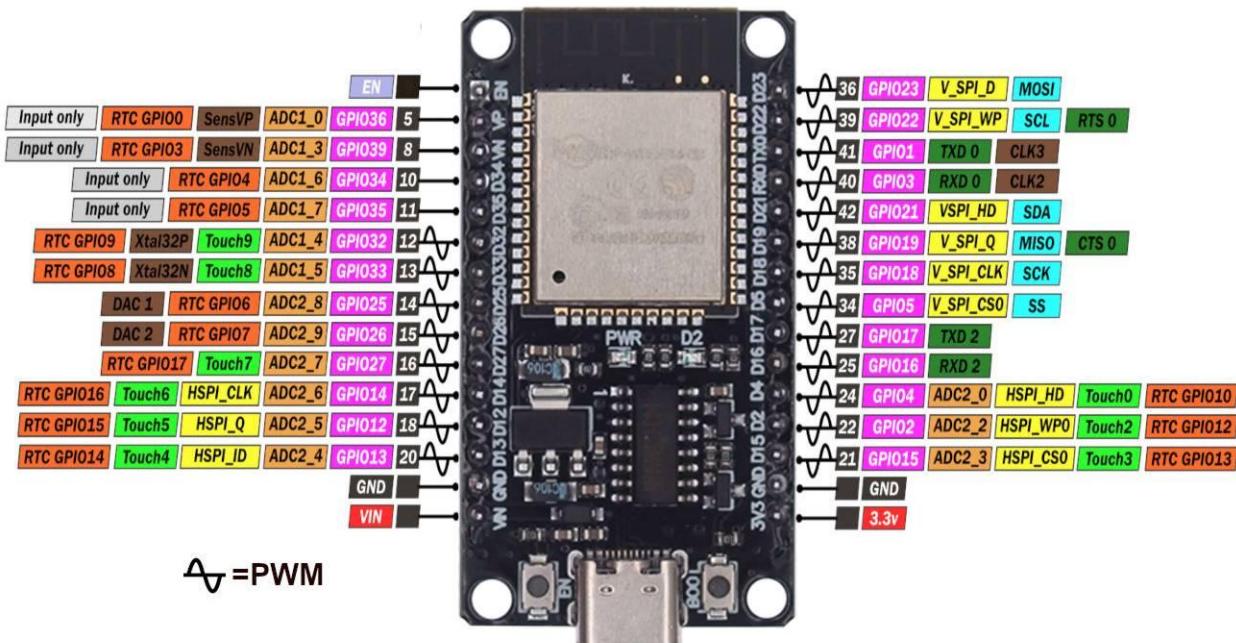


Figura 3-1. Pines ESP32

3.1.2 Sensores y actuadores

Los sensores y actuadores del sistema son los nombrados y con las características especificadas en el Capítulo 2. Son los elementos que nos permiten conocer el estado del invernadero y cómo reaccionar dependiendo de lo que estamos leyendo. Para que los datos leídos sean comprensibles se ha creado un programa que muestra los datos en porcentaje, se explicará en detalle más adelante. Para ver el montaje de los elementos ir al Anexo B.

3.1.3 Programa del dispositivo

En este subapartado vamos a analizar el programa que ejecuta el dispositivo para controlar todos los elementos hardware.

3.1.3.1 Bloque de configuración

Al iniciar el ESP32, se ejecuta el siguiente flujo de configuración:

1. Conexión WiFi: El dispositivo intenta conectarse a una red WiFi cuyo nombre (SSID) y contraseña deben estar predefinidos en el código. Si la conexión es exitosa, se asigna una IP fija (configurada manualmente) para evitar cambios aleatorios. Si la IP está ocupada, el sistema notifica el conflicto. Para ello se ha utilizado la librería: <WiFi>.

```

IPAddress local_IP(192, 168, 1, 100);
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 255, 0);
IPAddress primaryDNS(8, 8, 8, 8);
IPAddress secondaryDNS(8, 8, 4, 4);

WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    LOG_INFO("Conectando a WiFi...");
}
LOG_INFO("Conectado a WiFi. IP: " + WiFi.localIP().toString());
if (!WiFi.config(local_IP, gateway, subnet, primaryDNS, secondaryDNS)) {
    LOG_ERROR("Error al configurar IP estatica");
}

```

Figura 3-2. Configuración del Wifi

2. Sincronización horaria: Se configura la hora del sistema usando un servidor NTP (Network Time Protocol), que por defecto proporciona la hora en UTC+0. Se ajusta la zona horaria local para garantizar que las comparaciones de tiempo (ej: horarios de riego) sean precisas. Para ello se ha utilizado la librería: <NTPClient>.
3. Inicialización de los sensores y actuadores: Para el sensor DTH11 se necesita una librería específica la <DHT> que permite obtener directamente la humedad en porcentaje y la temperatura. Para el resto de los sensores y actuadores se inician los pines correspondientes.
4. Obtención de valores iniciales: El ESP32 solicita al Context Broker (vía HTTP GET) los últimos valores almacenados de las entidades (límites, estados de actuadores, etc.), recibiendo un JSON que se procesa y guarda en memoria.
5. Inicialización de un servidor de notificaciones: Se inicia un servidor web local en la ruta IP: puerto/update para recibir notificaciones del Context Broker cuando algún valor cambie. Para ello se ha utilizado la librería: <WebServer>.

3.1.3.2 Bloque de operación

En este bloque tenemos las funciones que hace de manera repetitiva mientras el ESP32 esté encendido. Lo primero que hacemos es activar la entrada del Serial para poder recibir comandos para la depuración, activar el manejador del servidor para las notificaciones y actualizamos la hora para cuando la queramos usar.

Lo siguiente es comprobar si el control manual del motor está activado, es decir, si una variable bandera está activa significa que hemos recibido una orden del usuario y eso tiene preferencia a lo automático y durante el tiempo que indique esta variable del actuador estará en el estado que indique el usuario y no lo que diga el control automático. Cuando supere el tiempo máximo permitido, se desactiva y se registra un log indicando que el control manual ha expirado.

Por último, lo que se va a hacer es leer los sensores, pero solo cada 5 segundos para reducir el consumo. Para lograr esto lo que hay que hacer es leer el valor analógico de los sensores y convertirlos en porcentaje mediante las operaciones correspondientes, se tomará el valor máximo que se lee y se convertirá en el 100% relativo. El siguiente paso es comprobar los valores de los sensores y si ha variado un porcentaje definido por el programador y si es así se mandará al context broker, mediante el código de la figura.

```
HTTPClient http;
http.begin(fiware_sensor_url);
http.addHeader("Content-Type", "application/json");

String payload = "{";
payload += "\"nivel\": {"value": " + String(nivel_value) + "},";
payload += "\"luz\": {"value": " + String(light_value) + "},";
payload += "\"humedad_suelo\": {"value": " + String(humedad_value) + "},";
payload += "\"humedad_ambiental\": {"value": " + String(humi) + "},";
payload += "\"temperatura_C\": {"value": " + String(temperature_C) + "}";
payload += "}";

int httpResponseCode = http.POST(payload);
```

Figura 3-3. Enviar los datos de los sensores

3.1.3.3 Bloque del control automático

En este bloque se va a explicar cómo se han hecho las funciones del control automático.

En primer lugar, tenemos las comprobaciones para determinar el estado que le corresponde al motor.

1. Lo primero que se va a comprobar es si el nivel del agua que detecta el sensor es mayor que el nivel mínimo establecido por el usuario. En caso contrario, se apagará el motor, se indicará el motivo en un log y se notificará al context broker.
2. Después se realiza la comprobación horaria, es decir, se verifica que la hora actual esté entre las horas configuradas por el usuario para el riego. Si no se cumple esta condición, se repite el proceso de error de la comprobación anterior.

3. A continuación, se comprueba si está activa la bandera de control manual. Si está activa, el proceso termina aquí; de lo contrario, se pasa al último paso.
4. Por último, se determina el estado que debería tener el motor (encendido si la humedad es menor que el mínimo configurado, o apagado en caso contrario) y se compara con el estado actual. Si no coinciden, se cambia el estado y se notifica al context broker.

En segundo lugar, tenemos las comprobaciones para determinar el estado del aire acondicionado.

1. Primero, el código verifica si el control manual del aire está desactivado.
2. Si se cumple esta condición, se evalúan las condiciones ambientales: el aire se encenderá si la humedad está fuera de los límites configurados (por debajo del mínimo o por encima del máximo) o si la temperatura supera los umbrales establecidos. Si el estado actual del pin del aire no coincide con el requerido, se actualiza el pin, se notifica al context broker y se registra el cambio en los logs.
3. Si el control manual supera el tiempo permitido, se desactiva automáticamente y se registra un log de que ha expirado.

Por último, tenemos las comprobaciones para determinar el estado de las luces.

1. Primero se verifica si el control manual de las luces está desactivado.
2. Si es así, se determina si las luces deben encenderse comparando los valores del sensor de luminosidad con los límites configurados. Si el estado actual del pin es distinto del requerido, se actualiza el pin, se notifica al context broker y se registra el evento en los logs.
3. Si el control manual excede el tiempo máximo permitido, se desactiva y se registra un log indicando que ha expirado.

3.1.3.4 Bloque para obtener del context broker

Este bloque está formado por las 4 funciones que obtienen los datos del context broker, es decir, obtener: los límites, y el estado del motor, las luces y el aire acondicionado. Las 3 últimas funciones son iguales, excepto el endpoint al que hacen la solicitud y el pin en el que guardan los datos. Por otro lado, la función de los límites, aunque muy parecida tiene diferencias más significativas, así que voy a explicar la función para obtener los límites y el motor en detalle.

La función para obtener el estado del motor consulta el estado actual del motor en el context broker para sincronizar el dispositivo físico con el estado registrado en el servidor.

1. Primero se verifica la conexión Wi-Fi, si no se cumple, se detiene. Si hay conexión, se realiza una petición HTTP GET al endpoint configurado.
2. Si la respuesta es exitosa, procesa el payload del JSON recibido: se extrae el valor del atributo estado y, si difiere del estado actual del motor, actualiza el pin físico, se activa el indicador de control manual, se registra el tiempo de inicio y se guarda el nuevo estado. Todo esto se registra en los logs.
3. Si hay errores, se notifica en los logs.
4. Finalmente, cierra la conexión HTTP.

La función para obtener los límites se encarga de obtener y actualizar los límites de configuración del invernadero desde FIWARE Orion.

1. Primero verifica la conexión Wi-Fi y, si está activa, realiza una petición HTTP GET al endpoint.
2. Si la respuesta es exitosa, procesa el payload JSON recibido:
 - a. Se limpian los arrays que almacenan las horas y minutos de riego.

- b. Se extrae y se asignan los valores de los límites desde el JSON a la estructura “límites”, incluyendo:
- i. Consumo máximo de agua.
 - ii. Horas programadas de riego, almacenando horas y minutos en arrays separados.
 - iii. Rangos de humedad ambiental.
 - iv. Rangos de humedad del suelo.
 - v. Rangos de los niveles de luz.
 - vi. Rango de temperaturas.
 - vii. Volumen mínimo de agua.

3. Si la operación es exitosa, se registra un log de confirmación; y en caso de error, se registra un log con el error específico. Finalmente, cierra la conexión HTTP.

```

void obtenerLimites() {
    if (WiFi.status() != WL_CONNECTED) return;

    HTTPClient http;
    http.begin(fiware_limites_url);
    int httpCode = http.GET();

    if (httpCode == HTTP_CODE_OK) {
        String payload = http.getString();
        DynamicJsonDocument doc(2048);
        deserializeJson(doc, payload);

        limites.horas_riego_horas.clear();
        limites.horas_riego_minutos.clear();

        limites.consumo_agua_max = doc["consumo_agua_max"]["value"].as<int>();

        JSONArray horasArray = doc["horas_riego"]["value"];
        for (JSONArray horaPair : horasArray) {
            limites.horas_riego_horas.push_back(horaPair[0].as<int>());
            limites.horas_riego_minutos.push_back(horaPair[1].as<int>());
        }

        limites.humedad_amb_min = doc["humedad_amb_min"]["value"].as<float>();
        limites.humedad_amb_max = doc["humedad_amb_max"]["value"].as<float>();
        limites.humedad_suelo_min = doc["humedad_suelo_min"]["value"].as<int>();
        limites.humedad_suelo_max = doc["humedad_suelo_max"]["value"].as<int>();
        limites.luz_min = doc["luz_min"]["value"].as<int>();
        limites.luz_max = doc["luz_max"]["value"].as<int>();
        limites.temp_min = doc["temp_min"]["value"].as<float>();
        limites.temp_max = doc["temp_max"]["value"].as<float>();
        limites.volumen_agua_min = doc["volumen_agua_min"]["value"].as<int>();

        LOG_DEBUG("Límites actualizados desde FIWARE");
    } else {
        LOG_ERROR("Error obteniendo límites: " + String(httpCode));
    }
    http.end();
}

```

Figura 3-4. Función para obtener los límites

3.1.3.5 Bloque de notificación al context broker

Este bloque está compuesto por 3 funciones análogas: la función para notificar el estado del motor, la función para notificar el estado de las luces y la función para notificar el estado del aire acondicionado. La única diferencia entre las funciones es el endpoint al que se envía, así que se va a explicar la función para notificar el estado del aire acondicionado, ya que el funcionamiento es el mismo para el resto de las funciones que conforman este bloque.

La función para notificar el estado del aire acondicionado se encarga de actualizar el estado del aire acondicionado en el context broker.

1. Primero, se verifica si hay conexión Wi-Fi; si no hay conexión, la función termina.
2. Si hay conexión, se inicia una petición HTTP al endpoint predefinido y se configura el encabezado para enviar los datos en formato JSON.
3. Luego, se construye un payload con el estado del aire (true o false) y un metadato que indica el origen del dato ("Arduino"). Se usa el método HTTP PATCH para enviar esta información al servidor FIWARE.
4. Si la respuesta es exitosa, se registra un mensaje de confirmación en los logs; en caso de error, se registra el problema con los detalles específicos. Finalmente, cierra la conexión HTTP para liberar recursos.

```
void updateAireStateInFIWARE(bool state) {
    if (WiFi.status() != WL_CONNECTED) return;

    HTTPClient http;
    http.begin(firmware_aire);
    http.addHeader("Content-Type", "application/json");

    char payload[128];
    snprintf(payload, sizeof(payload),
              "{\"estado\":%s,\"metadata\":{\"origen\":{\"value\":\"Arduino\"}}}",
              state ? "true" : "false");
    int code = http.PATCH(payload);

    if (code > 0) {
        LOG_INFO("Estado del aire actualizado en FIWARE");
    } else {
        LOG_ERROR("Error actualizando aire: " + String(http.errorToString(code)));
    }
    http.end();
}
```

Figura 3-5. Función de notificación del estado del aire acondicionado

3.1.3.6 Bloque de manejo de las suscripciones

Este bloque está compuesto por la función que maneja las notificaciones que le llegan al servidor web configurado. Esta función procesa las notificaciones POST entrantes que provienen del context broker para actualizar el estado de los dispositivos o de los límites del invernadero. Siguiendo estos pasos:

1. Como primer paso, se realiza la recepción y validación de los datos mediante los siguientes subpasos:
 - a. Se comprueba que sea el método HTTP POST.
 - b. Se procesa el cuerpo de la petición, para ello:
 - i. Se lee el cuerpo en formato JSON (server.arg("plain")).
 - ii. Se valida que el JSON esté bien formado (usa StaticJsonDocument<1024>).
 - iii. Se rechazan peticiones malformadas con código 400 Bad Request.
2. Una vez se ha comprobado que el mensaje es correcto, se filtran las notificaciones propias mediante una función que detecta y descarta actualizaciones que el propio Arduino envió al context broker. Esto se consigue comparando el campo de metadatos que contiene el origen de la notificación.

3. Después de descartar las notificaciones propias, se procesan el resto de las notificaciones:
 - a. Si la notificación contiene el atributo “estado”, actualiza el pin GPIO correspondiente, activa el control manual del dispositivo y registra el tiempo de inicio.
 - b. Si la notificación es de tipo "Limites", actualiza todos los parámetros recibidos, limpiando y rellenando los arreglos de horas y minutos de riego.
4. Por último, envía una respuesta HTTP, al elemento que hizo la petición:
 - a. Envía 200 OK si todo se procesó correctamente.
 - b. Envía 405 Method Not Allowed si no es una petición POST.

3.1.3.7 Bloque de depuración

En este bloque se explica cómo se ha logrado crear los elementos de depuración del código, es decir, cómo se ha conseguido crear tres niveles de registros y cómo lograr que aparezcan por el monitor serial.

Para implementar un sistema de registros flexible con tres niveles de prioridad (ERROR, INFO, DEBUG), se ha creado un enum LogLevel donde cada nivel incluye a los anteriores (DEBUG muestra todo, ERROR solo lo crítico). Las macros LOG_ERROR, LOG_INFO y LOG_DEBUG imprimen mensajes en el monitor serial únicamente si están habilitadas y si el nivel actual es igual o superior al del mensaje.

Además, la función que recibe los mensajes de la salida serial permite controlar el sistema en tiempo real: posibilita activar/desactivar registros y cambiar el nivel de detalle mediante comandos por el monitor serial, lo que facilita la depuración sin necesidad de modificar el código.

```
void handleSerialCommands() {
    if (Serial.available()) {
        String command = Serial.readStringUntil('\n');
        command.trim();
        command.toUpperCase();

        if (command == "ENABLE_LOGS") {
            loggingEnabled = true;
            Serial.println("Logs habilitados");
        } else if (command == "DISABLE_LOGS") {
            loggingEnabled = false;
            Serial.println("Logs deshabilitados");
        } else if (command == "LOG_LEVEL=ERROR") {
            currentLogLevel = ERROR_LEVEL;
            Serial.println("Nivel de logs: ERROR");
        } else if (command == "LOG_LEVEL=INFO") {
            currentLogLevel = INFO_LEVEL;
            Serial.println("Nivel de logs: INFO");
        } else if (command == "LOG_LEVEL=DEBUG") {
            currentLogLevel = DEBUG_LEVEL;
            Serial.println("Nivel de logs: DEBUG");
        }
    }
}

enum LogLevel { ERROR_LEVEL, INFO_LEVEL, DEBUG_LEVEL };
LogLevel currentLogLevel = INFO_LEVEL;
#define LOG_ERROR(x) if (loggingEnabled && currentLogLevel >= ERROR_LEVEL) { Serial.print("[ERROR] "); Serial.println(x); }
#define LOG_INFO(x) if (loggingEnabled && currentLogLevel >= INFO_LEVEL) { Serial.print("[INFO] "); Serial.println(x); }
#define LOG_DEBUG(x) if (loggingEnabled && currentLogLevel >= DEBUG_LEVEL) { Serial.print("[DEBUG] "); Serial.println(x); }
bool loggingEnabled = true;
```

Figura 3-6. Bloque de depuración

3.2 Context Broker

Como ya se ha explicado anterior anteriormente, el context broker de este proyecto es Fiware Orion. Los datos que recibe FIWARE deben estar en formato JSON y se distinguen por entidades que tienen cada una su propia URL. Para la realización del proyecto se han usado las dos funciones principales de FIWARE: las entidades en el endpoint /v2/entities/ (más la entidad específica) y las notificaciones en el endpoint /v2/subscription.

Ambos endpoint complementan a la URL base donde está Fiware que en este proyecto ha sido <http://192.168.1.113:1026/> en el OS Windows. Para los elementos que se ejecutan en Windows (como OpenHAB), se puede sustituir la IP por localhost.

Para persistir el último valor de las entidades, se enlaza Fiware con MongoDB, que es una BBDD que está en el mismo contenedor DOCKER.

3.2.1 Entidades

Los datos recopilados por los sensores del invernadero se envían al Orion Context Broker de Fiware, donde se almacenan y organizan en entidades que representan cada componente del sistema. Cada entidad se identifica únicamente por su id y type, y contiene atributos con los valores medidos o configurados, junto con metadatos que describen su origen. Para este proyecto, las entidades clave son:

- La entidad del sensor, que contiene los valores de los sensores y está compuesta por:
 - Id SensorData1 y Type Sensor:
 - Atributos: humedad_ambiental, humedad_suelo, luz, nivel, temperatura_C, temperatura_F.
 - Cada atributo tiene un campo de metadatos vacío.
- La entidad del motor, que contiene el estado del motor y está compuesta por:
 - Id MotorData1 y Type Motor:
 - Atributo: estado, una variable de tipo boolean que indica el estado del motor.
 - El atributo tiene un campo de metadatos que es una variable de tipo texto con un valor que indica el origen.
- La entidad de los límites, que contiene los valores de los límites y está compuesta por:
 - Id Limites1 y Type Limites:
 - Atributos: temp_min, temp_max, humedad_amb_min, humedad_amb_max, luz_max, luz_min, horas_riegos, humedad_suelo_max, volumen_agua_min y consumo_agua_max. Todos estos atributos son de tipo Number, excepto horas_riegos que es una StructruredValue.
 - Cada atributo tiene un campo de metadatos que es una variable de tipo texto con un valor que indica el origen.
- La entidad del aire acondicionado, que contiene el estado del aire acondicionado y está compuesta por:
 - Id AireAcondicionado y Type AireAcondicionado:
 - Atributo: estado, una variable de tipo boolean que indica el estado del aire acondicionado.
 - El atributo tiene un campo de metadatos que es una variable de tipo texto con un valor que indica el origen.
- La entidad de las luces, que contiene el estado de las luces y está compuesta por:
 - Id Luces y Type Luces:
 - Atributo: estado, una variable de tipo boolean que indica el estado de las luces.
 - El atributo tiene un campo de metadatos que es una variable de tipo texto con un valor que indica el origen.

Como se puede observar en la figura 3-7 aparte del id y el type, los atributos tienen su propia estructura compuesta por:

- El campo type que indica el tipo de dato (Number, Boolean, StructuredValue).
- El campo value que indica el valor actual.
- El campo metadata que indica información adicional, en este caso el origen de los datos. El campo metadata a su vez, está compuesto por un campo type y un campo value.

```

{
  "id": "MotorData1",
  "type": "Motor",
  "estado": {
    "type": "Boolean",
    "value": false,
    "metadata": {
      "origen": {
        "type": "Text",
        "value": "backend"
      }
    }
  },
  {
    "id": "Limites:1",
    "type": "Limites",
    "consumo_agua_max": {
      "type": "Number",
      "value": 100,
      "metadata": {
        "origen": {
          "type": "Text",
          "value": "openhab"
        }
      }
    },
    "horas_rie go": {
      "type": "StructuredValue",
      "value": [
        [20, 0]
      ],
      "metadata": {
        "origen": {
          "type": "Text",
          "value": "openhab"
        }
      }
    }
  }
}

```

Figura 3-7. Entidades Fiware

De otra forma, si queremos consultar solo una entidad solo tenemos que hacer una petición GET añadiendo su id al final del endpoint, por ejemplo, /v2/entities/Limites:1 para consultar los límites donde Limites:1 es el id de la entidad. Además, se pueden obtener solo los atributos de la entidad, siguiendo con el ejemplo anterior quedaría /v2/entities/Limites:1/attrs/consumo_agua_max, y si añadimos value solo devolverá el valor del atributo.

```

{
  "consumo_agua_max": {
    "type": "Number",
    "value": 100,
    "metadata": {
      "origen": {
        "type": "Text",
        "value": "openhab"
      }
    }
  }
}

```

Figura 3-8. Atributo de una entidad Fiware

Para eliminar una entidad se debe hacer una petición DELETE especificando el id de la entidad que queremos eliminar. Ahora se muestra un ejemplo usando el comando curl de Linux:

```

curl -X DELETE \
'http://192.168.1.113:1026/v2/entities/Limites:1/attrs/consumo_agua_ma
x'

```

3.2.2 Suscripciones

Las suscripciones son un mecanismo mediante el cual Fiware notifica a los dispositivos suscritos a una entidad cada vez que la entidad sea actualizada. En la notificación que envía Fiware se encuentra todo el JSON de la entidad actualizada, es decir, sería lo mismo que el resultado que devuelve Fiware cuando hacemos GET a esa entidad.

Cada suscripción se almacena en una lista que se puede observar en el endpoint /v2/subscription, de forma que si se envía una petición GET a la URL devuelve todas las suscripciones.

Cada suscripción consta de una serie de campos, que voy a resumir (si se quiere más información visitar [5]):

1. id: Es el identificador generado automáticamente para la suscripción.
2. description: Texto creado por el usuario que indica el propósito de la suscripción (ej: notificar cambios al arduino).
3. expires: Es la fecha de caducidad de la suscripción (en formato ISO 8601).
4. status: Es el estado actual de la suscripción (active, expired, paused).
5. entities: Define qué entidades deben ser monitoreadas:
 - a. idPattern: es el patrón regex para filtrar por ID (ej: "Limites:.*" cubre todos los IDs que empiezan con "Limites:").
 - b. type: es el tipo de entidad a suscribir (ej: "Limites").
 - c. condition: son condiciones adicionales:
 - i. attrs: Son los atributos específicos a observar (si está vacío, se monitorizan todos).
 - ii. notifyOnMetadataChange: Si este campo es true, se notifican cambios tanto en valores como en metadatos.
6. timesSent: Es el número total de notificaciones enviadas.
7. lastNotification: Es la fecha de la última notificación exitosa.
8. attrs: Son los atributos específicos a incluir en la notificación (vacío = todos).
9. onlyChangedAttrs: Si este campo es false, se envía el estado completo de la entidad; si es true, solo los atributos modificados.
10. attrsFormat: Es el formato de los datos (normalized = estándar FIWARE, keyValues = simplificado).
11. http + url: Es el endpoint del servidor que recibirá las notificaciones.
12. lastFailure + lastFailureReason: Es la Fecha y el motivo del último fallo.
13. lastSuccess + lastSuccessCode: Es la Fecha y el código HTTP de la última notificación exitosa.
14. covered: Si este campo es true, indica que la suscripción está siendo procesada por un servidor federado.
15. throttling: Es el tiempo mínimo (en segundos) entre notificaciones consecutivas. Ejemplo: 5 = no envía más de una notificación cada 5 segundos, incluso si hay múltiples cambios.

```
{
  "id": "67fcfaa55cb8f475b800f8fc",
  "description": "Notificar cambios de límites al backend",
  "expires": "2040-01-01T00:00:00.000Z",
  "status": "active",
  "subject": {
    "entities": [
      {
        "idPattern": "Limites:.*",
        "type": "Limites"
      }
    ],
    "condition": {
      "attrs": [],
      "notifyOnMetadataChange": true
    }
  },
  "notification": {
    "timesSent": 230,
    "lastNotification": "2025-05-06T19:02:37.000Z",
    "attrs": [],
    "onlyChangedAttrs": false,
    "attrsFormat": "normalized",
    "http": {
      "url": "http://192.168.1.155:9000/invernadero/limites/fiware-callback"
    },
    "lastFailure": "2025-05-02T15:20:16.000Z",
    "lastFailureReason": "Timeout was reached",
    "lastSuccess": "2025-05-06T19:02:37.000Z",
    "lastSuccessCode": 200,
    "covered": false
  },
  "throttling": 5
},
```

Figura 3-9. Ejemplo de suscripción Fiware

Para generar una suscripción se debe mandar una petición POST a la URL de Fiware junto al endpoint que se ha nombrado al inicio de este subapartado y se siguen las recomendaciones de la guía de Fiware Orion [6].

Aquí se muestra un ejemplo de cómo hacer una suscripción mediante el comando curl:

```
curl -iX POST http://192.168.1.113:1026/v2/subscriptions \
-H "Content-Type: application/json" \
-d '{
    "description": "Notificación al backend cuando cambia el estado
    del motor",
    "subject": {
        "entities": [
            {
                "id": "MotorData1",
                "type": "Motor"
            }
        ],
        "condition": {
            "attrs": ["estado"]
        }
    },
    "notification": {
        "http": {
            "url": "http://192.168.1.152:9000/invernadero/motor/estado/cambiar",
            "qs": "motivo=fiware"
        },
        "attrs": ["estado"],
        "attrsFormat": "keyValues"
    },
    "throttling": 1
}'
```

Para eliminar una suscripción se debe hacer un DELETE a la URL donde se hizo el POST más el id de la suscripción que queremos eliminar.

Las notificaciones se envían como mensajes POST a la URL que se haya definido en el campo “url” de la suscripción.

4 SERVIDOR WEB Y BASE DE DATOS

La mejor manera de predecir el futuro es inventarlo.

- Alan Kay-

En este capítulo se va a explicar el funcionamiento del servidor web, ubicado en la máquina virtual con dirección IP 192.168.1.155 y en el puerto 9000; y el esquema de la base de datos que se usa en este proyecto. Para ello se detallarán los métodos más importantes y las clases que se han implementado.

4.1 Clases

En esta sección se detallan las clases Java creadas para el servidor web, ubicadas en la denominada ruta principal (`invernadero\src\main\java\com\greenhouse`). Las clases están divididas en las siguientes partes:

4.1.1 Principal

Es la clase `RegistroInvernaderoApplication`, que contiene el método `main` del programa. Este es el primer método que se llama al ejecutar el programa. Anotada con `@SpringBootApplication`, combina tres configuraciones clave [7]:

- `@EnableAutoConfiguration`: Configura automáticamente beans y dependencias.
- `@ComponentScan`: Escanea componentes (controladores, servicios, repositorios) en el paquete `com.greenhouse` y subpaquetes.
- `@Configuration`: Permite definir beans adicionales.

Además, `@EnableScheduling` habilita la ejecución periódica de tareas [8]. El método `main()` inicia la aplicación usando `SpringApplication.run()`, levantando el servidor embebido (Tomcat) y cargando toda la configuración.

4.1.2 Configuración

En el subdirectorio `config` de la ruta principal se encuentran las clases que configuran servicios. En este proyecto solo se encuentra la clase `SwaggerConfig`, que configura Swagger, un servicio para generar la documentación. En el fichero se han configurado el título, la versión que aparece en la interfaz de Swagger.

4.1.3 Modelos

En el subdirectorio `model` de la ruta principal se encuentran las clases que representan la estructura de datos del proyecto, es decir, son clases POJO. Estas clases definen:

- Las entidades que modelan elementos reales (ej: Motor, Sensor, Limites).
- Las relaciones con la base de datos mediante anotaciones (`@Entity`, `@Table`, `@Column`) para mapear tablas en PostgreSQL.
- Los atributos y las validaciones mediante anotaciones como `@NotNull`, `@Size`, o formatos específicos (`@DateTimeFormat`).

4.1.3.1 Modelado del Motor

Para modelar el motor, se han usado dos clases Java llamadas: “EstadoMotor” y “EventosMotor”. Ambas clases presentan los mismos atributos, pero “EstadoMotor” se usa para obtener el estado actual del motor y permite monitorizar el tiempo que lleva en funcionamiento a tiempo real para vincularlo con la lógica del negocio. En cambio, “EventosMotor” contiene un registro de los eventos asociados al motor del invernadero y registra cada acción de encendido y apagado con los detalles de quién activó el evento.

Los atributos de las clases son:

- accion: Describe el tipo de evento (“encendido” o “apagado”).
- motivo: Explica la causa por la que ocurrió el evento.
- duracionSegundos: Indica la duración en segundos del estado.
- consumoAgua: Indica el consumo del agua del estado.
- fechaCreacion: Se autogenera al persistir el evento y no se puede modificar.
- El campo id como clave primaria autoincremental.

4.1.3.2 Modelado de Límites

Para modelar los límites del sistema, se ha usado una clase Java llamada: “Limites”. Esta clase modela los parámetros de configuración del sistema del invernadero, definiendo los umbrales y reglas que controlan el comportamiento automático del sistema. Mapeada a la tabla limites en la base de datos mediante JPA [9], incluye:

- Atributos para los umbrales ambientales:
 - a. Los rangos de temperatura (tempMin, tempMax).
 - b. Los niveles de humedad (humedadAmbMin, humedadAmbMax).
 - c. La humedad del suelo (humedadSueloMin, humedadSueloMax).
 - d. La intensidad lumínica (luzMin, luzMax).
- Atributos para la configuración de recursos:
 - a. El consumo máximo de agua (consumoAguaMax).
 - b. El volumen mínimo de agua (volumenAguaMin).
 - c. Los horarios programados de riego (horasRiego), almacenados en una tabla separada (limites_horas_riege) mediante @ElementCollection.
- Otros:
 - a. La fechaActualizacion que se autogenera al modificar los límites.
 - b. La relación con la clase Usuarios mediante @ManyToOne para registrar quién realizó los cambios.
 - c. El campo usuarioActualizacion para consultas rápidas.
 - d. El campo id como clave primaria autoincremental.

4.1.3.3 Modelado de sensores

Para modelar los sensores, se ha usado una clase Java llamada: “LecturaSensores”. Esta clase representa las mediciones periódicas capturadas por los sensores del invernadero, almacenando datos ambientales clave para el control del sistema. Mapeada a la tabla lecturas_sensores mediante JPA, incluye:

- Atributos para los datos leídos como:

- a. El nivel de agua en el depósito (nivelAgua).
- b. La intensidad lumínica (luzAmbiente en porcentaje).
- c. La humedad del suelo (humedadSuelo en porcentaje).
- d. La humedad ambiental (humedadAmbiental en porcentaje).
- e. La temperatura en grados Celsius (temperaturaC) y Fahrenheit (temperaturaF).
- Otros:
 - a. La fechaCreacion generada automáticamente al insertar datos (@CreationTimestamp).
 - b. El campo id como clave primaria autoincremental.

4.1.3.4 Modelado de usuarios

Para modelar a los usuarios del sistema, se ha usado una clase java llamada: Usuarios. Esta clase representa a los usuarios del sistema de gestión del invernadero, controlando el nivel de acceso y permisos que poseen para interactuar con la plataforma. Mapeada a la tabla usuarios mediante JPA, incluye las siguientes características clave:

- Atributos para los datos de autenticación y seguridad como:
 - a. nombreUsuario y email con restricciones para que la combinación de estos campos sea única (@UniqueConstraint).
 - b. contraseña almacenada de forma segura (deberá estar encriptada en la implementación real).
 - c. Campos para control de actividad (activo) y nivel de privilegios (nivelAcceso).
- Otros:
 - a. fechaCreacion generada automáticamente al registrar el usuario (@CreationTimestamp).
 - b. ultimoAcceso que se actualiza con cada inicio de sesión exitoso.
 - c. El campo id como clave primaria autoincremental.
- La configuración por defecto:
 - a. Si no se especifica los nuevos usuarios tienen nivelAcceso = 1 y activo = true.

```
@Entity
@Table(name = "estado_motor")
public class EstadoMotor {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "fecha_creacion", nullable = false, updatable = false)
    private LocalDateTime fechaCreacion = LocalDateTime.now();

    @Column(nullable = false)
    private Boolean estado;

    @Column(name = "duracion_segundos")
    private Integer duracionSegundos;

    @Column(name = "consumo_agua")
    private Double consumoAgua;

    @Column(nullable = false)
    private String motivo;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public LocalDateTime getFechaCreacion() {
        return fechaCreacion;
    }

    public void setFechaCreacion(LocalDateTime fechaCreacion) {
        this.fechaCreacion = fechaCreacion;
    }
}
```

Figura 4-1.Clase tipo modelo (EstadoMotor)

4.1.4 Repositorios

En el subdirectorio repository de la ruta principal se encuentran las clases que actúan como la capa de acceso a datos (DAL) y tienen el siguiente propósito clave:

- Son Intermediarias entre la aplicación y la BBDD, es decir, se encargan de realizar operaciones CRUD (Create, Read, Update, Delete) sin necesidad de escribir SQL manualmente. Además, utilizan Spring Data JPA (para bases de datos relacionales como PostgreSQL) o Spring Data MongoDB (para NoSQL) [10].

Cada clase de modelo tiene su correspondiente clase Java de tipo repositorio JPA con sus consultas personalizadas:

- EstadoMotorRepository con:
 - a. `findTopByOrderByFechaCreacionDesc()`: Obtiene el último estado del motor registrado ordenado por fecha descendente.
- EventosMotorRepository con:
 - a. `findAllOrderByFechaCreacionDesc()`: Obtiene todos los eventos del motor ordenados por fecha descendente.
 - b. `findTop5OrderByFechaCreacionDesc()`: Obtiene los 5 eventos más recientes del motor ordenados por fecha descendente.
- LimitesRepository con:
 - a. `findTopByOrderByFechaActualizacionDesc()`: Obtiene el límite más reciente actualizado.
 - b. `findByUsuarioId(Integer usuarioId)`: Busca límites asociados a un ID de usuario.
 - c. `findTopByUsuarioOrderByFechaActualizacionDesc(Usuarios usuario)`: Obtiene el límite más reciente de un usuario específico.
 - d. `findByUsuarioOrderByFechaActualizacionDesc(Usuarios usuario)`: Obtiene todos los límites de un usuario ordenados por fecha de actualización descendente.
- LecturasSensoresRepository con:
 - a. `findTopByOrderByFechaCreacionDesc()`: Obtiene la última lectura de sensores registrada.
 - b. `findByFechaCreacionBetween(Date inicio, Date fin)`: Busca lecturas entre dos fechas.
 - c. `findUltimaLectura()`: Obtiene la última lectura de sensores (alternativa a `findTopBy`).
 - d. `obtenerResumenHoy()`: Obtiene los promedios horarios de lecturas de sensores del día actual (últimas 12 horas).
 - e. `obtenerResumen7Dias()`: Obtiene promedios diarios de lecturas de sensores de los últimos 7 días con datos almacenados.
 - f. `obtenerResumen30Dias()`: Obtiene promedios diarios de lecturas de sensores de los últimos 30 días con datos almacenados.
- UsuariosRepository con:
 - a. `findByNombreUsuario(String nombreUsuario)`: Busca un usuario por su nombre de usuario.
 - b. `findByEmail(String email)`: Busca un usuario por su email.
 - c. `existsByNombreUsuario(String nombreUsuario)`: Verifica si existe un usuario con ese nombre.
 - d. `existsByEmail(String email)`: Verifica si existe un usuario con ese email.

```

@Repository
public interface EventosMotorRepository extends JpaRepository<EventosMotor, Integer> {
    List<EventosMotor> findAllOrderByFechaCreacionDesc();
    List<EventosMotor> findTop5OrderByFechaCreacionDesc();
}

```

Figura 4-2. Clase tipo repositorio (EventosMotorRepository)

4.1.5 Servicios

En el subdirectorio service de la ruta principal se encuentran las interfaces que describen la lógica de negocio. Además, dentro de service existe el subdirectorio impl que contiene la implementación de las interfaces. En resumen:

- Las interfaces definen los métodos abstractos que describen qué operaciones de negocio deben implementarse.
- Las implementaciones desarrollan la lógica de negocio concretas de los métodos abstractos definidos en las interfaces.

Al igual que los repositorios, cada clase de modelo tiene una clase Java de interfaz y de implementación correspondiente.

4.1.5.1 Servicio del Motor

El servicio del motor está formado por los servicios ofrecidos por la interfaz y la implementación de las clases modelos que conforman el motor. Estas clases son:

- EventosMotorService es la interfaz de eventos motor que define el método obtenerEventos(), que obtiene una lista de todos los eventos registrados del motor.
- EventosMotorServiceImpl desarrolla el método obtenerEventos() usando el repositorio EventosMotorRepository para devolver todos los elementos almacenados en la BBDD (findAll()).
- EstadoMotorService es la interfaz de los estados del motor que define los siguientes métodos:
 - a. obtenerUltimoEstado(): Devuelve el último estado registrado del motor.
 - b. registrarCambioEstado(boolean nuevoEstado, String motivo): Guarda un nuevo estado del motor y los eventos asociados.
- EstadoMotorServiceImpl desarrolla los métodos definidos por su interfaz:
 - a. obtenerUltimoEstado(): Retorna el último estado almacenado en memoria (ultimoEstado) o lo busca en la base de datos, mediante el repository(findTopByOrderByFechaCreacionDesc()), si es la primera vez.
 - b. registrarCambioEstado(boolean nuevoEstado, String motivo): Se define como una transacción (@Transactional), para que si hay algún error no se persista en la BBDD.
 - Se crea un nuevo estado con motivo y fecha actual.
 - Si hubo un cambio de estado previo, se calcula duración y el consumo de agua, y guarda un evento. El consumo de agua es tiempo por 25 ml.
 - Se notifica a FIWARE (si el motivo no es "fiware").
 - c. comprobarConsumoAgua(): Verifica cada 5 segundos (@Scheduled) si el consumo de agua supera el límite máximo. En caso de que se supere apaga el motor.
 - d. notificarAFiware(boolean estado): Envía el estado actual del motor a FIWARE usando PATCH.

4.1.5.2 Servicio de los límites

El servicio de los límites está formado por la interfaz e implementación de la clase modelo:

- LimitesService es la interfaz de los parámetros de configuración del sistema. En esta clase se definen los siguientes métodos:
 - a. findCurrentLimits(): Obtiene los límites más recientes configurados en el sistema.
 - b. findCurrentLimitsByUserId(): Busca los límites activos para un usuario específico.
 - c. updateLimits(): Actualiza los límites existentes de un usuario.
 - d. save(): Crear límites para un usuario
 - e. notificarCambioAFIWARE(): Envía los límites actualizados a FIWARE.
 - f. actualizarHorasRiego(): Modifica las horas de riego permitidas para un usuario.
 - g. validateLimits(): Valida que los valores de los límites sean coherentes.
- LimitesServiceImpl es la clase que implementa los métodos de la interfaz:
 - a. findCurrentLimits(): Devuelve los últimos límites globales registrados (usa findTopByOrderByFechaActualizacionDesc).
 - b. findCurrentLimitsByUserId(): Obtiene los límites vigentes para un usuario específico (usa findTopByUsuarioOrderByFechaActualizacionDesc).
 - c. updateLimits(): Primero, se recuperan los límites actuales del usuario. Luego, se actualizan los campos editables (temperatura, humedad, etc.). Por último, se valida y notifica a FIWARE. Si falla en algún paso al ser una transacción (@Transactional): no se persiste en la base de datos.
 - d. save(): Primero, se validan los límites, luego se notifica a Fiware y por último se guarda en la BBDD.
 - e. actualizarHorasRiego() (@Transactional): Primero, se actualiza la lista de horas de riego permitidas y luego se notifica a FIWARE y se persiste en la BBDD. Si falla en algún paso al ser una transacción (@Transactional): no se persiste en la base de datos.
 - f. notificarCambioAFIWARE(): Envía los nuevos límites a FIWARE usando PATCH.
 - f. validateLimits(): Se comprueban que los nuevos límites cumplan con la lógica del negocio (un mínimo no puede superar un máximo, un porcentaje no puede valer más de un 100%, ...).

4.1.5.3 Servicio de los sensores

El servicio de los sensores está formado por la interfaz e implementación de la clase modelo:

- LecturasSensoresService es la interfaz que define los métodos para gestionar las lecturas de los sensores:
 - a. guardarDesdeNotificacion(Map<String, Object> payload): Procesa y guarda datos recibidos de una notificación desde FIWARE.
 - b. obtenerUltimaLectura(): Consulta la última lectura registrada en la base de datos.
 - c. obtenerHistorial(): Devuelve todas las lecturas almacenadas (sin filtros).
 - d. obtenerResumenHoy(): Obtiene un resumen de promedios horarios de las últimas 12 horas.
 - e. obtenerResumenUltimosDias (String periodo): Obtiene un resumen de promedios diarios del número de días que indique el periodo.
- LecturasSensoresServiceImpl es la clase que implementa los métodos de su interfaz:
 - a. guardarDesdeNotificacion(Map<String, Object> payload): Primero, se extraen los datos del payload FIWARE. Luego, se mapean los valores a un objeto LecturasSensores. Por último, se persisten en la base de datos mediante el método del repositorio (save()).
 - b. obtenerUltimaLectura(): Usa el método del repositorio (findUltimaLectura()) para recuperar la lectura más reciente.

- c. obtenerHistorial(): devuelve todas las lecturas mediante el método del repositorio (findAll()).
- d. obtenerResumenHoy(): Ejecuta una consulta SQL definida en el repositorio (obtenerResumenHoy) que permite calcular el promedio horario de los valores de los sensores durante las últimas 12 horas.
- e. obtenerResumenUltimosDias(String periodo): Ejecuta una consulta SQL definida en el repositorio (obtenerResumen30Dias o obtenerResumen7Dias) que permite calcular el promedio diario de los valores de los sensores durante el número de días indicado en el periodo.

4.1.5.4 Servicio de los usuarios

El servicio de los usuarios está formado por la interfaz e implementación de la clase modelo:

- UsuarioService es la interfaz que define los métodos para gestionar a los usuarios del sistema:
 - registrarUsuario(Usuarios usuario): Crea un nuevo usuario con valores por defecto.
 - login(String nombreUsuario, String contraseña): Verifica credenciales y registra último acceso si son válidas.
 - findAll(): Obtiene todos los usuarios registrados.
 - findById(Integer id): Busca un usuario por su ID.
 - save(Usuarios usuario): Guarda o actualiza un usuario existente.
 - delete(Integer id): Elimina un usuario por ID.
 - existeNombreUsuario(String nombreUsuario): Verifica si un nombre de usuario ya está registrado.
 - existeEmail(String email): Comprueba si un email ya está en uso.
 - actualizarContrasena(int id, String nuevaContrasena): Cambia la contraseña de un usuario si existe.
- UsuarioServiceImpl es la clase que implementa los métodos de su interfaz:
 - registrarUsuario(Usuarios usuario): Asigna fecha de creación, activa el usuario y asigna el nivel de acceso básico antes de guardar.
 - login(String nombreUsuario, String contraseña): Compara usuario y contraseña con las que existen en la BBDD. Actualiza ultimoAcceso si el login es exitoso.
 - findAll(): Mediante el método del repositorio (findAll()).
 - findById(Integer id): Mediante el método del repositorio (findById(id)).
 - save(Usuarios usuario): Mediante el método del repositorio (save(usuario)).
 - delete(Integer id): Mediante el método del repositorio (deleteById(id)).
 - existeNombreUsuario(String nombreUsuario): Mediante el método del repositorio (existsByNombreUsuario(nombreUsuario)).
 - existeEmail(String email): Mediante el método del repositorio (existsByEmail(email)).
 - actualizarContrasena(int id, String nuevaContrasena): Busca por id y si existe se actualiza la nueva contraseña en la BBDD.

```

@Transacional
@Override
public EstadoMotor registrarCambioEstado(boolean nuevoEstado, String motivo) {
    LocalDateTime ahora = LocalDateTime.now();
    EstadoMotor nuevo = new EstadoMotor();
    nuevo.setEstado(nuevoEstado);
    nuevo.setMotivo(motivo);
    nuevo.setFechaCreacion(ahora);

    boolean notificarFiware = false;

    if (ultimoEstado != null && !ultimoEstado.getEstado().equals(nuevoEstado)) {
        Duration duracion = Duration.between(ultimoEstado.getFechaCreacion(), ahora);
        int duracionSegundos = (int) duracion.getSeconds();
        double consumo = 0.0;

        if (ultimoEstado.getEstado()) {
            consumo = duracionSegundos * 25.0;
        }

        // Guardamos el estado anterior
        ultimoEstado.setDuracionSegundos(duracionSegundos);
        ultimoEstado.setConsumoAgua(consumo);
        estadoMotorRepository.save(ultimoEstado);
    }
}

public interface EstadoMotorService {
    EstadoMotor obtenerUltimoEstado();
    EstadoMotor registrarCambioEstado(boolean nuevoEstado, String motivo);
}

```

Figura 4-3. Clases de interfaz e implementación (EstadoMotorService y EstadoMotorServiceImpl)

4.1.6 Controladores

En el subdirectorio controller de la ruta principal se encuentran las clases que definen el comportamiento del servidor cuando se recibe una petición en la URL que definen. Su propósito principal es recibir peticiones HTTP, delega la lógica a los servicios y devuelven las respuestas estructuradas en el formato que se necesite, en este caso JSON.

Cada clase de modelado tiene su propia clase java que sirve de controlador.

4.1.6.1 Controlador del Motor

Está compuesto por los controladores de los eventos y del estado del motor. Estas clases son:

- EventosMotorController que tiene como ruta base /invernadero/motor/eventos y gestiona los eventos de operación del motor. Solo tiene el método de respuesta a un GET a este endpoint / y obtiene todos los eventos del motor ordenados cronológicamente.
- EstadoMotorController que tiene como ruta base invernadero/motor/estado y controla el estado actual del motor. Tiene configurado las respuestas a los siguientes métodos:

Método HTTP	Endpoint	Descripción
GET	/	Consulta el último estado registrado del motor.
POST	/cambiar	Cambia el estado del motor mediante un JSON y registra el motivo, por defecto, “fiware”.

Tabla 4-1. Peticiones respondidas por EstadoMotorController

4.1.6.2 Controlador de los límites

Está compuesto por el controlador que corresponde a su clase modelo. Esta clase es LimitesController que tiene como ruta base /invernadero/limites y que gestiona la configuración de límites ambientales y de riego para el invernadero.

Método HTTP	Endpoint	Descripción
GET	/	Obtiene los límites actuales configurados en el sistema, mediante findCurrentLimits del servicio.
GET	/usuario/{usuarioId}	Obtiene los límites específicos de un usuario por su ID, mediante el findCurrentLimitsByUserId del servicio.
POST	/usuario/{usuarioId}	Crea nuevos límites para un usuario, mediante el save del servicio.
PUT	/usuario/{usuarioId}	Actualiza límites existentes de un usuario, mediante el updateLimits del servicio.
POST	/fiware-callback	Recibe notificaciones de FIWARE para actualizar límites.
POST	/actualizar-horas-riegos/{usuarioId}	Actualiza horas de riego permitidas para un usuario, mediante el actualizarHorasRiego del servicio.

Tabla 4-2. Peticiones respondidas por LimitesController

4.1.6.3 Controlador de los sensores

Está compuesto por la clase LecturasSensoresController que tiene como ruta base /invernadero/sensores y que gestiona las lecturas de sensores del invernadero.

Método HTTP	Endpoint	Descripción
POST	/fiware-callback	Recibe datos de sensores desde FIWARE y los almacena en la base de datos, mediante el método guardarDesdeNotificacion(payload) del servicio.
GET	/ultima	Obtiene la última lectura registrada de los sensores, mediante el método obtenerUltimaLectura() del servicio.
GET	/historial	Recupera el historial completo de lecturas de sensores, mediante el método obtenerHistorial() del servicio.
GET	/resumen-hoy	Devuelve 12 lecturas del día actual mediante el método obtenerResumenHoy() del servicio.
GET	/resumen-ultimos-dias	Obtiene promedios diarios del número de días con datos que se indica en el periodo, mediante el método obtenerResumenUltimosDias(String periodo) del servicio.

Tabla 4-3. Peticiones respondidas por LecturasSensoresController

4.1.6.4 Controlador usuarios

Está compuesto por la clase UsuariosController que tiene como ruta base /invernadero/usuarios y que gestiona el registro, la autenticación y la gestión de usuarios del sistema.

Método HTTP	Endpoint	Descripción
GET	/	Obtiene todos los usuarios registrados en el sistema, mediante el método findAll() del servicio.
GET	/{id}	Obtiene un usuario específico por su ID (excluye datos sensibles como contraseña), mediante el método findById(id) del servicio.
GET	/login	Autentica un usuario mediante nombre de usuario y contraseña, mediante el método login(nombreUsuario, contraseña) del servicio.
POST	/registro	Registra un nuevo usuario (valida duplicados de nombre/email), mediante los métodos existeNombreUsuario(), existeEmail(), registrarUsuario(usuario) del servicio.
GET	/{nombreUsuario}/existe	Verifica si un nombre de usuario ya está en uso, mediante el método existeNombreUsuario(nombreUsuario) del servicio.
GET	/email/{email}/existe	Verifica si un email ya está registrado, mediante el método existeEmail(email) del servicio.
PATCH	/{id}/contrasena	Actualiza la contraseña de un usuario existente, mediante el método actualizarContrasena(id, nuevaContrasena) del servicio.

Tabla 4-4. Peticiones respondidas por UsuariosController

API Documentation 3.2 OAS3

Documentación de mi API para mi proyecto de gestión de un Invernadero

Servers
http://localhost:9000 - Generated server url

Usuarios

Operaciones para la gestión de usuarios

- POST** /invernadero/usuarios/registro Registrar nuevo usuario
- PATCH** /invernadero/usuarios/{id}/contraseña Cambiar contraseña
- GET** /invernadero/usuarios Obtener todos los usuarios
- GET** /invernadero/usuarios/{nombreUsuario}/existe Verificar nombre de usuario
- GET** /invernadero/usuarios/{id} Obtener usuario por ID
- GET** /invernadero/usuarios/login Iniciar sesión
- GET** /invernadero/usuarios/email/{email}/existe Verificar email

Lecturas de sensores

Operaciones relacionadas con lecturas de sensores desde FIWARE

- POST** /invernadero/sensores/fiware-callback Recibir notificación desde FIWARE
- GET** /invernadero/sensores/ultima Obtener la última lectura registrada
- GET** /invernadero/sensores/resumen-ultimos-dias Obtener resumen de lecturas (periodo de días)
- GET** /invernadero/sensores/resumen-hoy Obtener 12 lecturas del dia actual
- GET** /invernadero/sensores/historial Obtener el historial completo de lecturas

Límites

Operaciones relacionadas con los límites de los parámetros configurados por el usuario

- GET** /invernadero/lmites/usuario/{usuarioId} Obtener límites del usuario
- PUT** /invernadero/lmites/usuario/{usuarioId} Actualizar límites
- POST** /invernadero/lmites/usuario/{usuarioId} Crear límites
- POST** /invernadero/lmites/actualizar-horas-riego/{usuarioId} Actualizar las horas de riego
- GET** /invernadero/lmites Obtener límites actuales

Estado del Motor

API para registrar y consultar el estado del motor

- POST** /invernadero/motor/estado/cambiar Cambia el estado del motor y registra la transición
- GET** /invernadero/motor/estado/ Obtiene el último estado del motor

Eventos del Motor

API para consultar eventos del motor

- GET** /invernadero/motor/eventos Obtiene todos los eventos del motor

Schemas

- Límites >
- LocalTime >
- Usuarios >
- EstadoMotor >
- LecturasSensores >
- EventosMotor >

Figura 4-4. Documentación de Swagger

4.2 Ficheros de configuración

En este apartado se detallan los ficheros de configuración del servidor web. Hay dos tipos de ficheros de configuración los que configuran las dependencias del proyecto (.xml) y los que configuran las propiedades de algunos elementos (.properties).

Los ficheros de origen properties se encuentran dentro del subdirectorio resources que está en la ruta invernadero\src\main\resources. En este proyecto solo se ha usado un fichero de este tipo llamado application y contiene:

- La configuración de la BBDD detallando el tipo de BBDD que en este caso es PostgreSQL, la URL para la conexión (jdbc:postgresql://localhost:5432/registro_invernadero), las credenciales y el controlador que se usa.
- Las propiedades de Hibernate (JPA) que se configuran. Se activa el mostrar consultas SQL para mostrar las consultas en el servidor y formatoSQL para mejorar legibilidad. Se define la estrategia de actualización de la BBDD y el dialecto que se usa para hablar con la misma.
- El puerto de la aplicación en este caso el 9000.
- La configuración de la URL donde se va a ejecutar el swagger (<http://localhost:9000/swagger-ui.html>)

Por otro lado, el fichero de las dependencias se encuentra al inicio de la ruta en el subdirectorio invernadero y se llama POM. Este fichero contiene las tecnologías usadas y las dependencias necesarias para el correcto funcionamiento del proyecto.

4.3 BBDD

En este apartado se va a explicar la estructura de la BBDD, es decir, el nombre de esta, las tablas que contiene y la relación entre ellas. Se ha configurado una base de datos PostgreSQL que es un sistema de bases de datos relacional de código abierto. El nombre de la BBDD es registro_invernadero. Para ver cómo se ha configurado la BBDD consulte el Anexo C: Configuración de la base de datos. La BBDD está compuesta por las siguientes tablas:

4.3.1 Tabla usuarios

Es la tabla que almacena la información de los usuarios del sistema. Además, tiene una relación 1:N con límites. Sus elementos son:

Column	Type	Collation	Nullable	Default
id	integer		not null	nextval('usuarios_id_seq'::regclass)
nombre_usuario	text		not null	
contraseña	text		not null	
email	character varying(100)		not null	
activo	boolean			true
fecha_creacion	timestamp without time zone			CURRENT_TIMESTAMP
ultimo_acceso	timestamp without time zone			
nivel_acceso	integer			1

Indexes:

```
"usuarios_pkey" PRIMARY KEY, btree (id)
"idx_usuarios_nombre" btree (nombre_usuario)
"uk_email" UNIQUE CONSTRAINT, btree (email)
"ukkfsp0s1tflm1cw1j8idhq5ad0" UNIQUE CONSTRAINT, btree (email)
"ukof5vabgukahdwmgxk4kjrbu98" UNIQUE CONSTRAINT, btree (nombre_usuario)
"usuarios_nombre_usuario_key" UNIQUE CONSTRAINT, btree (nombre_usuario)
```

Referenced by:

```
TABLE "limites" CONSTRAINT "limites_usuario_id_fkey" FOREIGN KEY (usuario_id) REFERENCES usuarios(id) ON DELETE CASCADE
```

Figura 4-5. Tabla usuarios

4.3.2 Tabla limites

Es la tabla que contiene la configuración de los parámetros ambientales y de riego del invernadero. Tiene una relación 1:N con limites_horas_riego. Sus elementos son:

Column	Type	Table "public.limites"	Collation	Nullable	Default
id	integer			not null	nextval('limites_id_seq'::regclass)
fecha_actualizacion	timestamp without time zone				CURRENT_TIMESTAMP
temp_min	double precision			not null	
temp_max	double precision			not null	
humedad_amb_min	double precision			not null	
humedad_amb_max	double precision			not null	
humedad_suelo_min	integer			not null	
humedad_suelo_max	integer			not null	
luz_min	integer			not null	
luz_max	integer			not null	
consumo_agua_max	double precision			not null	10.0
volumen_agua_min	double precision			not null	20.0
usuario_id	integer			not null	
usuario_actualizacion	text			not null	

Indexes:

- "limites_pkey" PRIMARY KEY, btree (id)
- "idx_limites_actualizacion" btree (fecha_actualizacion)

Foreign-key constraints:

- "limites_usuario_id_fkey" FOREIGN KEY (usuario_id) REFERENCES usuarios(id) ON DELETE CASCADE

Referenced by:

- TABLE "limites_horas_rie go" CONSTRAINT "limites_horas_rie go_limite_id_fkey" FOREIGN KEY (limite_id) REFERENCES limites(id) ON DELETE CASCADE

Figura 4-6. Tabla limites

4.3.3 Tabla limites_horas_rie go

Es la tabla que contiene las horas permitidas para el riego automático y tiene una FK a limites. Sus elementos son:

Column	Type	Table "public.limites_horas_rie go"	Collation	Nullable	Default
id	integer			not null	nextval('limites_horas_rie go_id_seq'::regclass)
limite_id	integer			not null	
hora_rie go	time without time zone			not null	

Indexes:

- "limites_horas_rie go_pkey" PRIMARY KEY, btree (id)
- "idx_horas_rie go_limite" btree (limite_id)

Foreign-key constraints:

- "limites_horas_rie go_limite_id_fkey" FOREIGN KEY (limite_id) REFERENCES limites(id) ON DELETE CASCADE

Figura 4-7. Tabla limites_horas_rie go

4.3.4 Tabla lecturas_sensores

Es la tabla que contiene los registros de los datos leídos por los sensores. Sus elementos son:

Column	Type	Table "public.lecturas_sensores"	Collation	Nullable	Default
id	integer			not null	nextval('lecturas_sensores_id_seq'::regclass)
fecha_creacion	timestamp without time zone				CURRENT_TIMESTAMP
nivel_agua	integer			not null	
luz_ambiente	integer			not null	
humedad_suelo	integer			not null	
humedad_ambiental	double precision			not null	
temperatura_c	double precision			not null	
temperatura_f	double precision			not null	

Indexes:

- "lecturas_sensores_pkey" PRIMARY KEY, btree (id)
- "idx_lecturas_fecha" btree (fecha_creacion)

Figura 4-8. Tabla lecturas_sensores

4.3.5 Tabla estado_motor

Es la tabla que contiene el estado actual del motor y el consumo de agua. Sus elementos son:

Table "public.estado_motor"				
Column	Type	Collation	Nullable	Default
id	integer		not null	nextrval('estado_motor_id_seq'::regclass)
fecha_creacion	timestamp without time zone		not null	CURRENT_TIMESTAMP
estado	boolean		not null	
duracion_segundos	integer			
consumo_agua	double precision			0.0
motivo	character varying(20)		not null	

Indexes:

- "estado_motor_pkey" PRIMARY KEY, btree (id)
- "idx_motor_estado" btree (estado)

Figura 4-9. Tabla estado_motor

4.3.6 Tabla eventos_motor

Es la tabla que mantiene el registro de los cambios de estado del motor.

Table "public.eventos_motor"				
Column	Type	Collation	Nullable	Default
id	integer		not null	nextrval('eventos_motor_id_seq'::regclass)
fecha_creacion	timestamp without time zone		not null	CURRENT_TIMESTAMP
accion	character varying(9)		not null	
motivo	character varying(20)		not null	
duracion_segundos	integer			
consumo_agua	double precision			0.0

Indexes:

- "eventos_motor_pkey" PRIMARY KEY, btree (id)
- "idx_eventos_accion" btree (accion)

Check constraints:

- "eventos_motor_accion_check" CHECK (accion::text = ANY (ARRAY['ENCENDIDO'::character varying, 'APAGADO'::character varying]::text[]))

Figura 4-10. Tabla eventos_motor

A continuación, se muestra el diagrama ER de la base de datos:

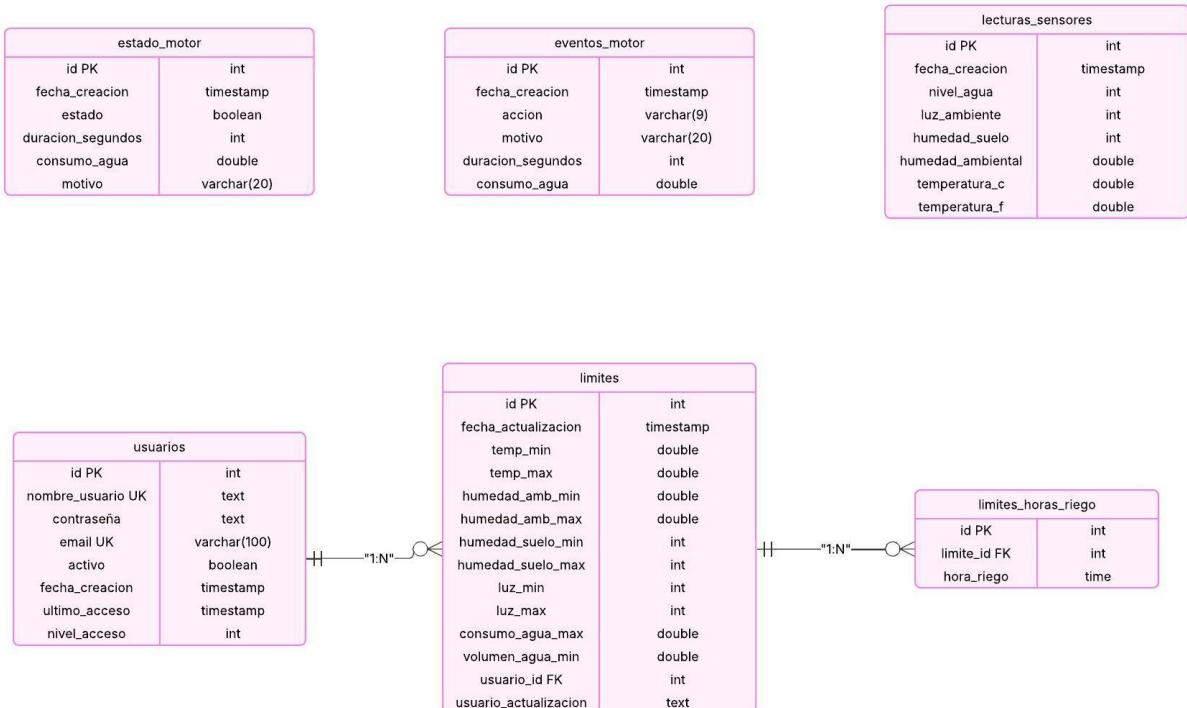


Figura 4-11. Diagrama ER

5 INTERFAZ DE USUARIO Y FUNCIONALIDAD

¿Qué somos las personas sino máquinas muy evolucionadas?

- Marvin Minsky -

En este capítulo se va a explicar el funcionamiento de las dos interfaces de usuario diseñadas en este proyecto. Por un lado, tenemos la aplicación Android y por otro lado tenemos la aplicación OpenHAB, ambas permiten gestionar el invernadero, pero cada una tiene ventajas e inconvenientes.

5.1 Aplicación Android

Gracias a la aplicación Android llamada ETSINatura, el usuario final podrá ver el estado actual del invernadero y controlar el motor de riego. Además, se podrá configurar los límites por los que se rigen las operaciones de control automático incluyendo la elección de las horas de riego según se haya creído conveniente después de ver el precio y el pronóstico del tiempo mediante las funcionalidades que se ofrecen. Por último, el usuario podrá obtener los registros del invernadero para analizar la evolución del mismo. Para comunicarse con el servicio REST, se usan los endpoints definidos en el punto 4.

A continuación, se muestran los casos de uso que tiene la aplicación y una explicación de cómo funciona cada uno.

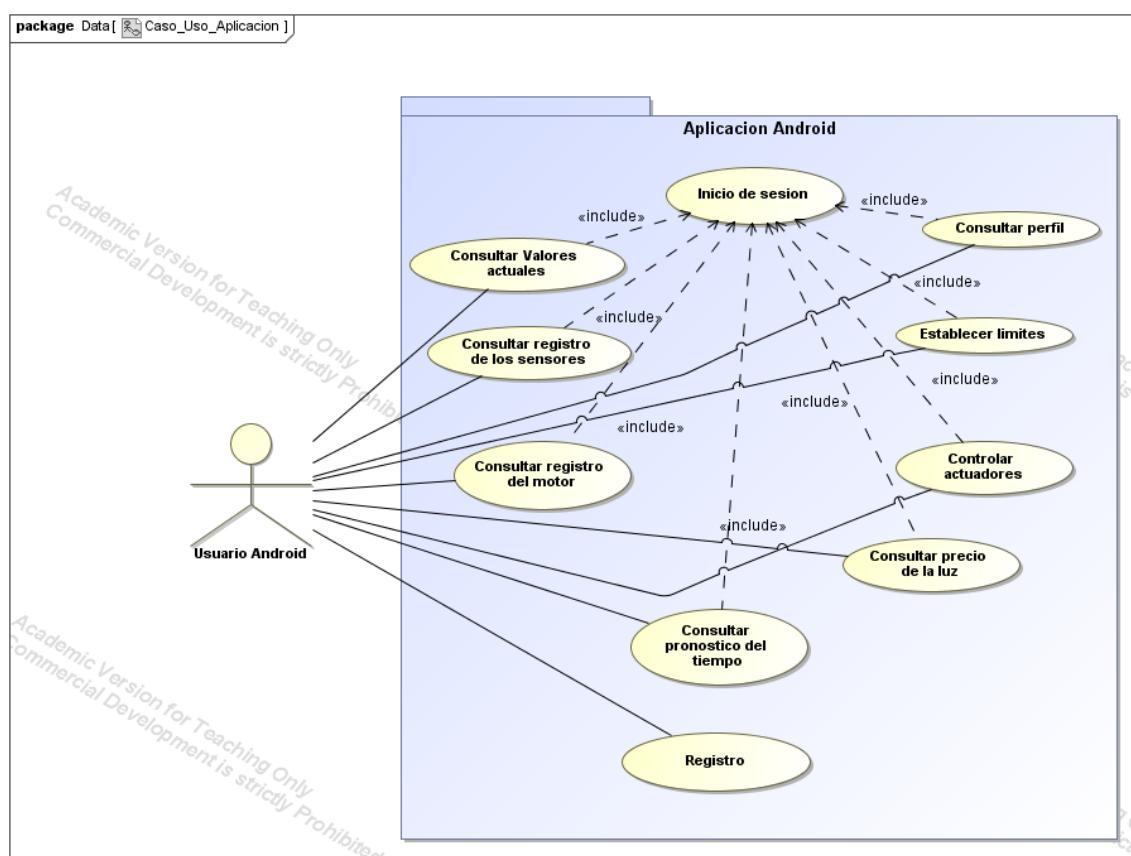


Figura 5-1. Diagrama de casos de uso

5.1.1 Inicio de sesión

En este apartado se explica el funcionamiento y los elementos que conforman el caso de uso “Inicio de Sesión” de la aplicación Android. Este caso de uso consiste en una pantalla con dos campos a llenar el usuario y la contraseña, además de dos botones “Acceder” y “Registrarse”.

Para implementar esta función se han necesitado los siguientes ficheros:

- **MainActivity.java:** Esta clase es la actividad principal de la aplicación y maneja el inicio de sesión de usuarios. Contiene los campos para ingresar usuario y contraseña, y realiza una petición HTTP GET a un servidor para validar las credenciales. Si el inicio de sesión es exitoso, muestra un texto Toast con “Inicio de sesión exitoso” y redirige a MainUsuarios enviando los datos del usuario. También incluye un botón para redirigir a la actividad de registro (RegistroActivity) y maneja errores de conexión con mensajes Toast.
- **activity_main.xml:** Define la interfaz de usuario de la actividad principal, que incluye:
 - Campos de texto (EditText) para usuario y contraseña.
 - Botones para iniciar sesión (Acceder) y registrarse (Registrarse).
 - Un fondo personalizado (@drawable/fondo_inicio) y estilos para los elementos (texto en negrita, colores semitransparentes, etc.).



Figura 5-2. Inicio de sesión

Además de este caso de uso en este apartado también se va a incluir la explicación de los ficheros que permiten las actividades de los usuarios en el resto de los casos de uso que necesitan primero que se inicie sesión.

- **MainUsuarios.java:** Esta clase representa la actividad principal del usuario después del inicio de sesión, implementando un menú de navegación lateral y una barra de herramientas. Gestiona la navegación entre diferentes fragmentos (inicio, perfil, precios de electricidad, límites, actuadores, gráficas y eventos del motor) utilizando el componente Navigation, estos fragmentos corresponden a un caso de uso cada uno. Recibe los datos del usuario en formato JSON desde la actividad de login y los pasa a los fragmentos correspondientes. También incluye un botón de cierre de sesión en el menú superior que redirige al usuario de vuelta a la pantalla de inicio de sesión.
- **activity_main_usuarios.xml:** Define la estructura visual de la pantalla principal de la aplicación, utilizando un DrawerLayout como contenedor principal. Incluye:
 - Un CoordinatorLayout que contiene una AppBar con Toolbar personalizada y un contenedor de fragmentos que muestra el contenido dinámico según la navegación.

- Un NavigationView (menú lateral) que muestra las opciones de navegación definidas en el archivo de menú (@menu/menu_nav) y un encabezado personalizado (@layout/nav_header).
- Configuración de navegación mediante el grafo definido en @navigation/usuario_navigation, permitiendo la transición fluida entre las diferentes secciones de la aplicación.
- **content_usuarios.xml:** Este archivo actúa como contenedor secundario para los fragmentos dentro de la actividad principal.
- **app_bar_usuarios.xml:** Define la estructura de la barra de aplicación y el contenedor principal para la navegación entre fragmentos en la actividad MainUsuarios.

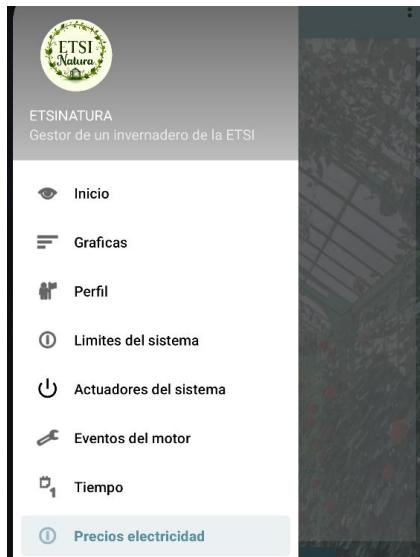


Figura 5-3. Barra de aplicación

5.1.2 Registro

En este apartado se explica el funcionamiento y los elementos que conforman el caso de uso “Registro”. Este caso de uso muestra los campos a llenar para registrar un usuario nuevo en la BBDD y dos botones: uno para cancelar el registro y otro para confirmarlo.

Para su implementación se han necesitado los siguientes ficheros:

- **RegistroActivity.java:** Esta clase gestiona el proceso de registro de nuevos usuarios en la aplicación. Contiene los campos a ingresar: nombre de usuario, contraseña y email. Se valida que todos estén completos antes de enviar una petición POST al servidor con los datos en formato JSON. Se muestra un diálogo de progreso durante la solicitud y maneja tanto la respuesta exitosa (mostrando un mensaje y cerrando la actividad) como los errores. También incluye un botón para cancelar el registro y volver a la pantalla anterior.
- **activity_registro.xml:** Define la interfaz de usuario para el registro, manteniendo el diseño visual coherente con la pantalla de inicio de sesión (mismo fondo y estilo). Incluye:
 - Tres campos de texto (EditText) para usuario, contraseña (con enmascaramiento) y email, cada uno con su etiqueta (TextView) correspondiente.
 - Dos botones (“Confirmar Registro” y “Cancelar Registro”) con estilos similares a los de inicio de sesión.
 - Diseño lineal vertical con márgenes y alineación centrada, optimizado para usabilidad en dispositivos móviles.



Figura 5-4. Registro

5.1.3 Consultar valores actuales

En este apartado se explica el funcionamiento y los elementos que conforman el caso de uso “Consultar valores actuales”. Este caso de uso muestra los valores actuales leídos por el sensor y una representación visual de lo que significa ese valor, por ejemplo, un recipiente lleno o vacío dependiendo del nivel del agua.

Para implementar esto se han necesitado los siguientes ficheros:

- **InicioFragment.java:** Fragmento que muestra el panel de inicio del usuario, con datos de sensores (luz, temperatura, humedad ambiental, humedad del suelo y nivel de agua). Recibe los datos del usuario en formato JSON desde la actividad principal y los utiliza para personalizar el mensaje de bienvenida. Realiza una petición HTTP GET al servidor para obtener las últimas lecturas de los sensores y actualiza la interfaz con los valores recibidos, mostrando iconos dinámicos (como termómetros de distintos colores) según los umbrales predefinidos.
- **fragment_inicio.xml:** Interfaz del fragmento de inicio, diseñada con un CoordinatorLayout y un ScrollView para permitir desplazamiento. Incluye un mensaje de bienvenida y cinco secciones (cada una con un ImageView y un TextView) para mostrar los datos de los sensores. Los elementos están organizados en un LinearLayout vertical con fondos semitransparentes. Los iconos y textos se actualizan dinámicamente desde el código según los valores de los sensores.



Figura 5-5. Consultar valores actuales

5.1.4 Consultar registros de los sensores

En este apartado se explica el funcionamiento y los elementos que conforman el caso de uso “Consultar registros de los sensores”. Este caso de uso muestra dos spinners a seleccionar: el sensor y el rango de días a consultar, además de un botón que permite la obtención y la carga de los datos.

Para su implementación se han necesitado los siguientes ficheros:

- **GraficasService.java:** Interfaz que define los endpoints de la API del servicio REST para obtener datos de sensores, utilizando Retrofit. Incluye dos métodos: `getResumenHoy()` para recuperar datos del día actual y `getResumenUltimosDias()` para obtener datos históricos según un período especificado (como parámetro `periodo`). Ambos devuelven una lista de mapas con pares clave-valor que representan las lecturas de los sensores y sus fechas asociadas, facilitando la integración con el servicio REST.
- **GraficaSensoresFragment.java:** Fragmento que gestiona la visualización de gráficas de datos de los sensores (temperatura, humedad, luz, etc.). Permite seleccionar un sensor y un período (hoy, 7 días o 30 días) mediante spinners, y al hacer clic en "Cargar datos", realiza una petición HTTP a través de Retrofit. Procesa la respuesta para mostrar los datos en un gráfico de líneas (para temperaturas) o de barras (para humedad/nivel de agua), con ejes X formateados según las fechas/horas. Incluye los métodos para parsear fechas, ordenar los datos y ajustar la visualización de etiquetas para evitar saturación.
- **fragment_grafica_sensores.xml:** Diseño del fragmento de gráficas, que utiliza un `ConstraintLayout` con fondo semitransparente. Contiene:
 - Dos Spinner para seleccionar el período y el sensor, con estilos consistentes (texto negro sobre fondo blanco semitransparente).
 - Un botón "Cargar datos" para activar la consulta.
 - Dos gráficos (LineChart y BarChart) de la librería MPAndroidChart, inicialmente ocultos (`visibility="gone"`), que se muestran dinámicamente según el tipo de datos.

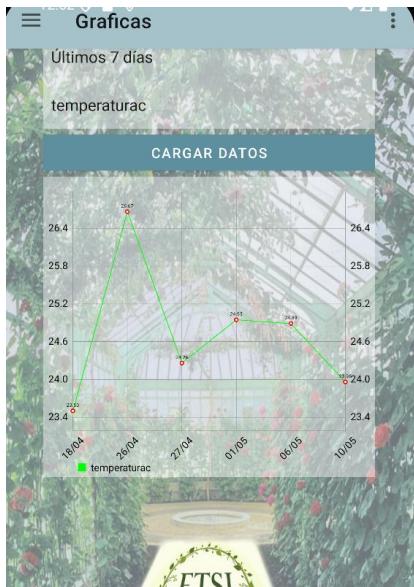


Figura 5-6. Consultar registro de los sensores

5.1.5 Consultar registro del motor

En este apartado se explica el funcionamiento y los elementos que conforman el caso de uso “Consultar registros del motor”. Este caso de uso muestra una gráfica de barras con el consumo de agua por evento de riego, es decir, por el tiempo en el que el motor esté encendido y todos los registros de los eventos del motor.

Para su implementación se han necesitado los siguientes ficheros:

- **EventosMotorAdapter.java:** Adaptador del RecyclerView que muestra la lista de eventos del motor. Vincula los datos de cada evento (acción, motivo, duración, consumo de agua y fecha) con las vistas del layout item_evento_motor.xml. Implementa los métodos esenciales de RecyclerView.Adapter para renderizar dinámicamente los elementos de la lista.
- **EventosMotorFragment.java:** Fragmento que gestiona la visualización de eventos del motor, combinando un gráfico de barras para mostrar el consumo de agua y un RecyclerView para listar los detalles de cada evento. Utiliza Retrofit para obtener datos paginados desde la API (EventosMotorService), procesa las fechas y actualiza tanto la gráfica como la lista. También, define el formato de las etiquetas del eje X del gráfico.
- **EventosMotorService.java:** Interfaz de Retrofit que define el endpoint para obtener eventos del motor. El método getEventosPaginados recibe parámetros page y size para controlar la paginación, devolviendo una lista de objetos Motor. Esta interfaz simplifica la comunicación con el servicio REST.
- **fragment_eventos_motor.xml:** Layout del fragmento que organiza la interfaz en un LinearLayout vertical. Contiene:
 - Un BarChart para visualizar el consumo de agua, con fondo semitransparente y estilo consistente.
 - Un RecyclerView para mostrar la lista detallada de eventos, ocupando el espacio restante.
 - Un TextView de título para la gráfica. Todos los elementos usan colores oscuros sobre fondos semitransparentes para mejorar la legibilidad.
- **item_evento_motor.xml:** Diseño de cada elemento individual en el RecyclerView. Usa un LinearLayout vertical con cinco TextView para mostrar los detalles del evento (acción, motivo, duración, consumo y fecha).
- **Motor.java:** Clase modelo que representa los eventos del motor en el sistema, con los atributos estado, acción, motivo, duracionSegundos, consumoAgua y fecha, con sus getters y setters.



Figura 5-7. Consultar registro del motor

5.1.6 Consultar perfil

En este apartado se explica el funcionamiento y los elementos que conforman el caso de uso “Consultar perfil”. Este caso de uso muestra un título del apartado y los campos de texto con los datos del usuario y un botón para cambiar la contraseña.

Para su implementación se han necesitado los siguientes ficheros:

- **PerfilFragment.java:** Fragmento que muestra y gestiona la información del perfil del usuario, incluyendo nombre, email, estado y nivel de acceso. Utiliza Retrofit para obtener los datos del usuario mediante su ID y actualiza la interfaz con la respuesta. Incluye un botón para cambiar la contraseña, que abre un diálogo (AlertDialog) para ingresar la nueva contraseña y realiza una petición al servicio REST mediante UsuarioService. Finalmente, muestra los errores y las respuestas del servidor con mensajes Toast.
- **UsuarioService.java:** Interfaz de Retrofit que define los endpoints para interactuar con el servicio REST. Incluye los siguientes métodos:
 - `getUsuarioById`: Obtiene los datos de un usuario específico mediante su ID (GET).
 - `cambiarContrasena`: Envía una nueva contraseña al servidor para actualizarla (PATCH), recibiendo el ID del usuario y un mapa con la nueva contraseña.
- **Usuario.java:** Clase modelo que representa un usuario en la aplicación. Contiene los campos `nombreUsuario`, `email`, `activo`, `nivelAcceso`, `ultimoAcceso` y `fechaCreacion`, junto con sus respectivos getters y setters. También incluye un campo `nuevaPassword` y métodos para gestionar cambios de contraseña. Este modelo se utiliza para parsear las respuestas JSON del servidor y manipular los datos del usuario en la aplicación.
- **fragment_perfil.xml:** Diseño del fragmento de perfil, estructurado con un ConstraintLayout y fondo semitransparente. Muestra:
 - Un título "Mi Perfil" (TextView) en la parte superior.
 - Cuatro TextView para mostrar el nombre, email, estado y nivel de acceso del usuario, con estilos consistentes (texto negro sobre fondo blanco semitransparente).
 - Un botón "Cambiar contraseña" con estilo destacado (fondo gris y texto blanco en negrita).



Figura 5-8. Consultar perfil

5.1.7 Establecer límites

En este apartado se explica el funcionamiento y los elementos que conforman el caso de uso “Establecer límites”. Este caso de uso muestra los límites actuales del sistema que se pueden cambiar, excepto las horas que solo se pueden modificar desde el caso de uso “Consultar precio de la luz” y un botón para guardar los cambios.

Para su implementación se han necesitado los siguientes ficheros:

- **LímitesService.java:** Interfaz Retrofit que define los endpoints para gestionar los límites del sistema. Incluye:
 - `getLimites`: Obtiene los límites actuales mediante una petición GET, usando el ID de usuario.
 - `actualizarLimites`: Envía nuevos límites al servidor con una petición PUT.
- **LímitesFragment.java:** Fragmento que muestra y permite editar los límites del sistema (temperatura, humedad, luz, etc.). Obtiene los datos actuales del servidor mediante LímitesService y los muestra en campos de texto. Solo usuarios con `nivelAcceso = 3`, es decir, los administradores pueden modificar los valores. Al pulsar el botón de guardar, envía los cambios al servidor y muestra notificaciones de éxito/error.
- **Límites.java:** Clase modelo que representa los límites configurables del sistema. Contiene los atributos rangos de temperatura (`tempMin`, `tempMax`), humedad (`humedadAmbMin`, `humedadSueloMax`), luz (`luzMin`, `luzMax`), consumo de agua (`consumoAguaMax`) y horas de riego (`horasRiego`). Incluye los getters/setters para cada campo y un constructor vacío, permitiendo el parseo automático de JSON con Gson.
- **fragment_limites.xml:** Diseño del fragmento de límites, estructurado con ConstraintLayout. Contiene:
 - 10 pares de `TextView` + `EditText`: Para mostrar/editar cada límite (ej. temperatura, humedad), con estilos consistentes (texto negro sobre fondo blanco semitransparente). Los `EditText` están deshabilitados por defecto y se activan solo para usuarios autorizados.
 - `TextView` para horas de riego: Muestra una lista de horas preconfiguradas.
 - Botón "Guardar cambios": Resaltado en celeste, visible solo para usuarios con permisos.



Figura 5-9. Consultar límites

5.1.8 Consultar precio de la luz

En este apartado se explica el funcionamiento y los elementos que conforman el caso de uso “Consultar precio de la luz”. Este caso de uso muestra el precio de la energía por hora y un spinner para elegir las horas de riego permitidas.

Para su implementación se han necesitado los siguientes ficheros:

- **Ree ApiService.java:** Interfaz Retrofit que define los endpoints para interactuar con la API de REE (Red Eléctrica Española) y el servicio REST. Incluye:
 - `getPrecios`: Realiza una petición GET a la API de REE (“es/datos/mercados/precios-mercados-tiempo-real”) para obtener precios horarios de electricidad, usando parámetros como fechas, zona geográfica y granularidad temporal.
 - `actualizarHorasRiego`: Envía una lista de horas de riego al servidor del invernadero mediante POST, asociadas a un ID de usuario.
- **PreciosElectricidadFragment.java:** Fragmento que muestra los precios de electricidad en tiempo real y permite a usuarios con nivelAcceso = 3 seleccionar horas óptimas para riego, las 5 horas más baratas. Usa la API de REE (["https://apidatos.ree.es/"](https://apidatos.ree.es/)) y el Ree ApiService, para obtener datos y los presenta en un TextView con horas más baratas en magenta. Por último, se pueden agregar horas (mediante TimePickerDialog) y enviarlas al servicio REST.
- **Precio.java:** Clase modelo que representa un precio horario de electricidad, con atributos `datetime` (marca temporal) y `value` (precio en €/MWh). Proporciona getters para acceder a estos datos, facilitando el parseo de JSON desde la API de REE.
- **ReeResponse.java:** Clase modelo que encapsula la respuesta de la API de REE. Contiene una lista de objetos, que a su vez almacenan los datos de precios.
- **fragment_precios_electricidad.xml:** Diseño del fragmento con un ScrollView y un LinearLayout vertical. Incluye:
 - `textoPrecios`: Muestra los precios en formato "HH:MM - precio €/MWh", con fondo semitransparente.
 - `btnAgregarHora` y `btnEnviarHoras`: Botones visibles solo para administradores (nivelAcceso = 3), para gestionar horas de riego.

- **textoHorasSeleccionadas:** Lista las horas elegidas por el usuario.



Figura 5-10. Consultar precio de la luz

5.1.9 Consultar pronóstico del tiempo

En este apartado se explica el funcionamiento y los elementos que conforman el caso de uso “Consultar pronóstico del tiempo”. Este caso de uso muestra dos campos para llenar con la ciudad y el país del que queremos el clima y un botón para enviar la petición.

Para su implementación se han necesitado los siguientes ficheros:

- **OpenWeatherService.java:** Interfaz Retrofit que define el endpoint ("data/2.5/forecast"), para obtener datos meteorológicos de OpenWeatherMap. Utiliza una petición GET con parámetros como ciudad, unidades (métricas), idioma (español) y la API key obtenida creando un usuario en OpenWeather. Retorna un JsonObject con la respuesta.
- **TiempoFragment.java:** Fragmento que muestra previsiones meteorológicas de 3 días (hoy, mañana y pasado) como máximo usando la API de OpenWeatherMap. Permite buscar por ciudad/país, procesa la respuesta JSON para extraer temperatura, descripción e iconos del clima (⽓ / ☀ / ⛅), y organiza los datos por franjas horarias. Usa la URL ("https://api.openweathermap.org/") y el OpenWeatherService para hacer la petición a la OpenWeather.
- **WeatherResponse.java:** Clase modelo que permite parsear la respuesta JSON de OpenWeatherMap. Contiene clases anidadas para mapear campos del JSON, esto facilita el acceso a datos como temperatura (temp), condición climática (main) y descripción (description).
- **fragment_tiempo.xml:** Diseño del fragmento con ConstraintLayout y un LinearLayout vertical. Incluye:
 - Dos EditText para ciudad y país.
 - Un botón "Buscar clima" que activa la consulta.
 - Un TextView para mostrar resultados.



Figura 5-11. Consultar pronóstico del tiempo

5.1.10 Controlar actuadores

En este apartado se explica el funcionamiento y los elementos que conforman el caso de uso “Controlar actuadores”. Este caso de uso muestra un texto que representa el estado del motor y un interruptor que permite cambiarlo.

Para su implementación se han necesitado los siguientes ficheros:

- **ActuadoresService.java:** Interfaz Retrofit que define los endpoints para interactuar con el sistema de actuadores del invernadero. Incluye dos métodos principales:
 - `getEstadoMotor`: Obtiene el estado actual del motor mediante una petición GET.
 - `cambiarEstadoMotor`: Envía un comando para modificar el estado del motor (encender/apagar) usando POST, aceptando un mapa de datos como payload y un parámetro de motivo.
- **ActuadoresFragment.java:** Fragmento que controla el estado del motor del invernadero. Muestra el estado actual (Encendido/Apagado) en un TextView y permite modificarlo mediante un Switch. Utiliza Retrofit para comunicarse con el servicio REST, manejando las interacciones del usuario en la aplicación para evitar actualizaciones no deseadas, es decir, tener en cuenta cuando se inicia el interruptor para que no se cambie el estado al por defecto al iniciar el fragmento y se envíe al servicio REST. Además, incluye la lógica para sincronizar el estado del switch con el servidor y muestra notificaciones Toast para confirmar cambios.
- **EstadoMotorRequest.java:** Clase modelo que representa una solicitud de cambio de estado del motor. Contiene dos campos: acción un booleano que indica si se debe encender (true) o apagar (false) el motor y motivo un String que describe la razón del cambio.
- **fragment_actuadores.xml:** Diseño del fragmento, organizado en un LinearLayout vertical centrado. Contiene:
 - `textoEstadoMotor`: Muestra el estado actual del motor.
 - `switch_motor`: Interruptor para control manual del motor.



Figura 5-12. Controlar actuadores

5.1.11 Ficheros de configuración

Para poder usar todos los elementos y que el proyecto funcione correctamente se deben configurar dos ficheros especiales, pues son los ficheros de configuración del proyecto:

- **AndroidManifest.xml:** Archivo de configuración principal que define la estructura básica de la aplicación Android. Declara los permisos necesarios (INTERNET y ACCESS_NETWORK_STATE), las actividades principales (MainActivity como actividad de lanzamiento, MainUsuarios y RegistroActivity), y configura aspectos globales como el ícono de la aplicación, tema y configuración de seguridad de red. También especifica que la aplicación es compatible con diseño RTL (Right-to-Left).
- **build.gradle:** Archivo de configuración de Gradle que gestiona las dependencias y opciones de compilación del proyecto. Establece las versiones mínimas (24) y objetivo (35) del SDK, configura el entorno de Java (versión 11), y declara las dependencias clave como Volley para peticiones HTTP, Material Design Components, Navigation Components, Retrofit para APIs REST, y MPAndroidChart para gráficos. Incluye también dependencias para testing (JUnit y Espresso).

```
implementation 'com.android.volley:volley:1.2.1'
implementation 'com.google.android.material:material:1.9.0'
implementation 'androidx.appcompat:appcompat:1.6.1'
implementation 'androidx.navigation:navigation-fragment:2.6.0'
implementation 'androidx.navigation:navigation-ui:2.6.0'
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'
```

Figura 5-13. Librerías del fichero build.gradle

5.2 Aplicación Openhab

La aplicación OpenHAB es un software que mediante canales conecta los ítems definidos en él y mediante reglas definimos cómo se va a actuar cuando ocurra un evento. La función de esta aplicación es permitir que a futuro se puedan conectar dispositivos IOT, sin necesidad de modificar la aplicación Android, de manera rápida y sencilla agregándolos como un ítem extra.

Lo primero que se ha definido ha sido el canal de donde va a escuchar los elementos que en este caso es la dirección y el puerto de FIWARE (<http://localhost:1026/v2/entities/>). Después de enlazar con FIWARE, se usa

este canal de enlace en las reglas que se han personalizado para los ítems del programa. Los ítems los podemos clasificar en 3 tipos, los valores de los sensores, los valores de los límites y los actuadores.

- Sensores: son los ítems que representan a los sensores.

Nombre	Tipo	Etiqueta	Categoría
OrionHumedadAmbiental	Number	Humedad Ambiente	humidity
OrionLuz	Number	Luz	light
OrionTemperaturaC	Number	Temperatura °C	temperature
OrionTemperaturaF	Number	Temperatura °F	temperature
OrionHumedadSuelo	Number	Humedad Suelo	humidity
OrionNivel	Number	Nivel	water

Tabla 5-1. Ítems de los sensores

- Límites: son los ítems que representan los límites todos son editables, excepto las horas de riego

Nombre	Tipo	Detalles
LímiteTempMin	Number	Límite de la temperatura mínima en °C
LímiteTempMax	Number	Límite de la temperatura máxima en °C
LímiteHumedadAmbMin	Number	Límite de la humedad ambiental mínima en %
LímiteHumedadAmbMax	Number	Límite de la humedad ambiental máxima en %
LímiteHumedadSueloMin	Number	Límite de la humedad del suelo mínima en %
LímiteHumedadSueloMax	Number	Límite de la humedad del suelo máxima en %
LímiteLuzMin	Number	Límite de la luz mínima en %
LímiteLuzMax	Number	Límite de la luz máxima en %
LímiteConsumoAguaMax	Number	Límite del consumo de agua por riego en litros
LímiteVolumenAguaMin	Number	Volumen de agua mínimo necesario para poder encender el motor
LímitesHorasRiego	String	Horas en las que se puede encender el motor (no editables).

Tabla 5-2. Ítems de los límites

- Actuadores: son los ítems que representan un actuador

Nombre	Tipo	Detalles
MotorEstado	Switch	Es el actuador que controla el motor de riego
LucesEstado	Switch	Es el actuador que controla la iluminación
AireEstado	Switch	Es el actuador que controla la climatización

Tabla 5-3. Ítems de los actuadores

Para controlar el funcionamiento de estos ítems se han definido las siguientes reglas:

- Lectura_Elementos es la regla que sirve para actualizar los estados cada 30 segundos consultando FIWARE (Motor, Sensores, Luces, Aire).
 - Detalle: Realiza peticiones HTTP GET al canal para obtener los datos de sensores (humedad, luz, temperatura, nivel), los estados de actuadores (ON/OFF) .
- ActualizarLimites es la regla que se ejecuta al modificar cualquier límite (temp, humedad, luz, agua). Se hacen las siguientes comprobaciones:
 - Rangos coherentes ($\text{min} < \text{max}$), Valores porcentuales (0-100%) y Horas de riego no vacías.
 - Se ignoran actualizaciones con metadatos origen: "openhab".
- AccionarLuces es la regla que controla las LucesEstado.
 - Acción: Envía estado (ON/OFF) a FIWARE via PUT al cambiar LucesEstado incluyendo los metadatos de origen para evitar bucles.
 - Endpoint: <http://localhost:1026/v2/entities/Luces/attrs/estado>.
- AccionarAire es la regla que controla el AireEstado
 - Usa PUT para enviar el estado al canal + AireAcondicionado/attrs/estado.
- Accionar Motor es la regla que controla MotorEstado.
 - Usa POST para enviar estado al canal + MotorData1/attrs.
- Inicio es la regla que se encarga de cargar todos los datos al arrancar OpenHAB.
 - Consulta: Obtiene valores actuales de sensores y límites de FIWARE.

Además de estas reglas tenemos un script de Python que cuando se recibe una notificación desde FIWARE cambia el formato al JSON que puede leer OpenHAB. Este fichero implementa un servidor Flask, en Docker, que actúa como intermediario para recibir notificaciones y actualizar ítems en OpenHAB. El servidor escucha en el puerto 5000 y define una ruta /notify que acepta solicitudes POST. Cuando recibe datos en formato JSON, extrae valores específicos (como límites de consumo de agua, humedad, temperatura, etc.) de un diccionario llamado MAPA_ITEMS, que mapea atributos recibidos a nombres de ítems en OpenHAB. Luego, envía estos valores a OpenHAB mediante solicitudes HTTP POST a su API REST. La dirección de OpenHAB está configurada para funcionar en un entorno Docker (host.docker.internal).

The screenshot shows the OpenHAB interface with the URL localhost:8080/settings/items. The left sidebar is titled 'Sin páginas' and contains the following navigation items:

- Administración**
 - Configuración
 - Things
 - Model
 - Items** (highlighted)
 - Pages
 - Rules
 - Scenes
 - Scripts
 - Schedule
- Tienda de complementos
- Herramientas de Desarrollo
- Ayuda y Acerca de

The main area is titled 'Items' and shows a list of 20 items:

Icon	Name	Type	Value	Action
	Aire Acondicionado	Switch	AireEstado	OFF >
	Consumo Agua Máx	Number · Point > Control	LímiteConsumoAguaMax	100 >
M	Estado del Motor	Switch	MotorEstado	OFF >
	Horas de Riego	String · Point > Control	LímiteHorasRiego	[[10.0],[11.0]] >
	Humedad Ambiental Máx	Number · Point > Control	LímiteHumedadAmbMax	99 >
	Humedad Ambiental Min	Number · Point > Control	LímiteHumedadAmbMin	20 >
	Humedad Ambiente	Number · Point > Measurement	OrionHumedadAmbiental	49 >
	Humedad Suelo	Number · Point > Measurement	OrionHumedadSuelo	0 >
	Humedad Suelo Máx	Number · Point > Control	LímiteHumedadSueloMax	99 >
				REFRESH

The bottom right corner of the main area has a blue circular button with a '+' sign.

A detailed view of the 'Humedad Suelo Mín' item is shown in a modal window:

Icon	Name	Type	Description	Value	Action
	Humedad Suelo Mín	Number · Point > Control	LímiteHumedadSueloMin	80 >	
	Luces	Switch	LucesEstado	OFF >	
	Luz	Number	OrionLuz	71 >	
	Luz Máxima	Number · Point > Control	LímiteLuzMax	80 >	
	Luz Mínima	Number · Point > Control	LímiteLuzMin	22 >	
	Nivel	Number	OrionNivel	0.0 >	
	Temperatura °C	Number	OrionTemperaturaC	24 >	
	Temperatura Máx	Number · Point > Control	LímiteTempMax	28 >	
	Temperatura Mín	Number · Point > Control	LímiteTempMin	15 >	
	Temperatura °F	Number	OrionTemperaturaF	72.86 >	
	Volumen Agua Mín	Number · Point > Control	LímiteVolumenAguaMin	0 >	

The bottom right of this modal also has a blue circular 'REFRESH' button.

Figura 5-14. Ítems de OpenHAB

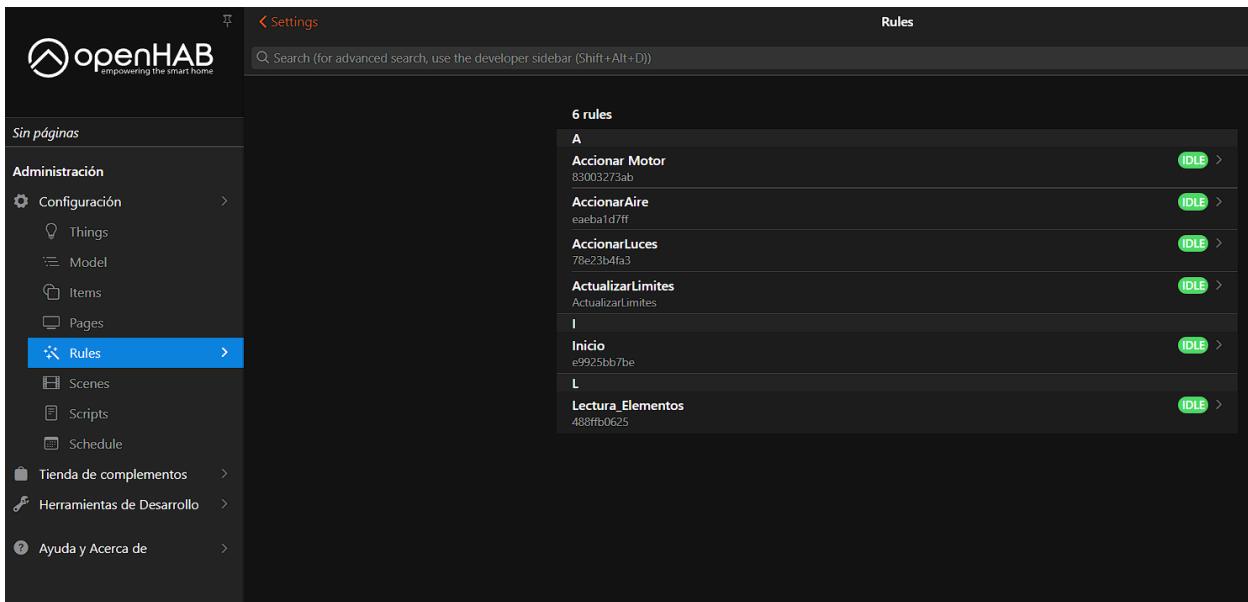


Figura 5-15. Reglas de OpenHAB

5.3 Resumen

En este apartado se redacta un resumen y una comparación de las dos interfaces de usuario que tenemos.

Por un lado, tenemos la aplicación móvil que permite el registro e inicio de sesión de los usuarios y permite observar los históricos de los elementos de la base de datos. Además, permite establecer las horas de riego permitidas y mirar el clima. Aunque solo permite controlar el motor.

Por otro lado, tenemos la interfaz de OpenHAB que, aunque permite editar los límites del sistema, no permite modificar las horas de riego permitidas ni consultar los históricos. Sin embargo, ofrece ventajas frente a la aplicación móvil, como la capacidad de controlar dispositivos IoT reales (por ejemplo, luces y aire acondicionado). En este proyecto, estos elementos se simulan mediante LEDs, pero en un entorno real podrían gestionarse directamente a través de canales de OpenHAB, simplificando su configuración en comparación con una implementación personalizada en la aplicación móvil.

6 CONCLUSIONES Y LÍNEAS FUTURAS

El arte desafía a la tecnología y la tecnología inspira al arte.

-John Lasseter -

El desarrollo del proyecto aunque diverso, tiene varios aspectos que se pueden mejorar. En este apartado se sacarán las conclusiones y se nombrarán las futuras líneas de mejoras que se pueden implementar sobre el proyecto.

El mundo actual está evolucionando a gran velocidad, y con ello las tecnologías y las formas en las que están presentes en nuestra vida. Cada vez los procesos son más automáticos y están controlados desde la distancia. Como se ha demostrado en este proyecto mediante la implementación de un sistema de riego y control de un invernadero, se puede controlar todo desde un móvil lo que permite una respuesta instantánea a los cambios.

Este proyecto ha resultado ser de bajo coste pues solo ha necesitado de los sensores, los actuadores y la placa ESP32.

Además, se ha demostrado que es posible conectar Fiware Orion Context Broker con OpenHAB, aunque OpenHAB no proporcione una integración nativa o directa. Esto se ha logrado mediante reglas, que permiten la automatización de los elementos del proyecto, y bindings configurados con las URLs de Fiware, que permiten obtener y actualizar los valores de los elementos.

Por otro lado, la aplicación Android ha demostrado que podemos conectarnos a la REE, para obtener los precios de la electricidad a tiempo real, y con OpenWeather, que permite obtener el pronóstico del tiempo. Ambas conexiones junto con las condiciones ambientales del invernadero que nos ofrece la aplicación nos permiten decidir qué límites queremos establecer para el invernadero. Por último, podemos obtener en forma de gráficas los registros temporales del invernadero para estudiar cómo mejorar la eficiencia de este.

Como líneas de interés para futuras mejoras se encuentran los siguientes apartados:

1. Seguridad:
 - a. Actualizar el paso de mensajes usando el protocolo HTTPS.
 - b. Implementar mecanismos que garanticen la autenticación de los usuarios y el cifrado de las comunicaciones, para evitar que un usuario no autorizado pueda cambiar los datos de algún elemento.
2. Funcionalidad:
 - a. Optimizar el uso de los metadatos en futuras versiones para evitar bucles de mensajes y mejorar la eficiencia de las entidades.
 - b. Añadir en los elementos del sistema la opción de modificar la dirección del servidor de manera dinámica y añadir la opción para que el administrador pueda eliminar información de la base de datos desde la aplicación Android.

ANEXO A: INSTALACIÓN DE FIWARE ORION USANDO DOCKER

En este anexo se explican los pasos a seguir para instalar y poner en funcionamiento Fiware Orion usando Docker y MongoDB como base de datos para Fiware.

Para instalar Docker, se necesita: un procesador de 64 bit con SLAT, 4GB de RAM, tener activado la virtualización en la BIOS y Windows 11 64-bit: Home o Pro versión 22H2 o mayor [11].

Los pasos a seguir para instalar Docker son:

1. Nos dirigimos a la página principal de Docker (<https://www.docker.com/>) y elegimos el OS donde nos queremos instalar Docker. En este caso se ha elegido Windows-AMD64

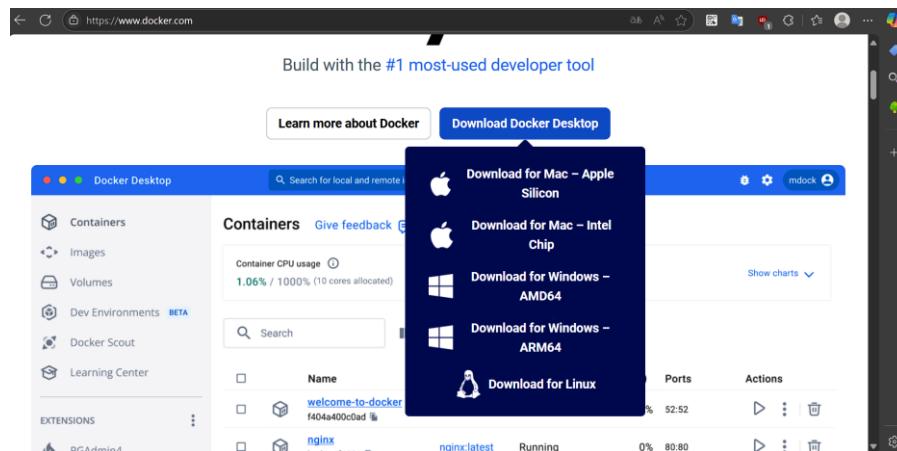


Figura 0-1. Descargar Docker

2. El siguiente paso es ejecutar el fichero .exe que instalará Docker en esta ruta por defecto, C:\Program Files\ Docker\ Docker. A partir de este momento seguimos las instrucciones del instalador y de la guía de Docker [11].

Una vez que tenemos instalado Docker procedemos a desplegar el contenedor de Fiware, que contiene al Orion y a MongoDB. Para lograr esto, se necesita lo siguiente:

1. Creamos el fichero docker-compose.yaml que contiene los elementos de nuestro contenedor. A continuación, se muestra el fichero:

```
version: "3.8"
services:
  orion:
    image: fiware/orion
    hostname: orion
    container_name: fiware-orion
    depends_on:
      - mongo-db
    networks:
      - fiware-net
    expose:
      - "1026"
    ports:
```

```

      - "1026:1026"
      command: -dbURI mongodb://mongo-db:27017/orion -logLevel
DEBUG

mongo-db:
  image: mongo:4.4
  hostname: mongo-db
  container_name: db-mongo
  networks:
    - fiware-net
  expose:
    - "27017"
  ports:
    - "27017:27017"
  command: --nojournal
  volumes:
    - mongo-db:/data/db

networks:
  fiware-net:
    driver: bridge

volumes:
  mongo-db:
    driver: local

```

- Una vez que tenemos el fichero, en la terminal de Docker ejecutamos el siguiente comando, indicando la ruta hasta el docker-compose:

```
docker-compose up -d .\fiware\docker-compose.yaml
```

- Una vez que tenemos el contenedor en la interfaz gráfica le damos al play para iniciar y a stop para detener

<input type="checkbox"/>	<input checked="" type="radio"/>	fiware	-	-	-	N/A 3 days ago		
<input type="checkbox"/>	<input checked="" type="radio"/>	orion	372f17452fcf	fiware/orion	1026:1026	N/A 3 days ago		
<input type="checkbox"/>	<input checked="" type="radio"/>	db-mongo	4034bb946f2e	mongo:4.4	27017:27017	N/A 3 days ago		

Figura 0-2. Contenedor apagado

<input type="checkbox"/>	<input checked="" type="radio"/>	fiware	-	-	-	N/A 0 seconds ago		
<input type="checkbox"/>	<input checked="" type="radio"/>	orion	372f17452fcf	fiware/orion	1026:1026	N/A 0 seconds ago		
<input type="checkbox"/>	<input checked="" type="radio"/>	db-mongo	4034bb946f2e	mongo:4.4	27017:27017	N/A 1 second ago		

Figura 0-3. Contenedor encendido

Una vez que tenemos todo instalado y configurado si pulsamos en el nombre del elemento podremos observar los logs de este y ver lo que ocurre.

ANEXO B: CONFIGURACIÓN DEL ESP32

En este anexo se detallará el procedimiento para obtener el software Arduino, y el proceso de montaje de los elementos físicos del proyecto.

En primer lugar, accedemos a <https://wwwarduino.cc/en/software/> y elegimos la versión 2.3.6.

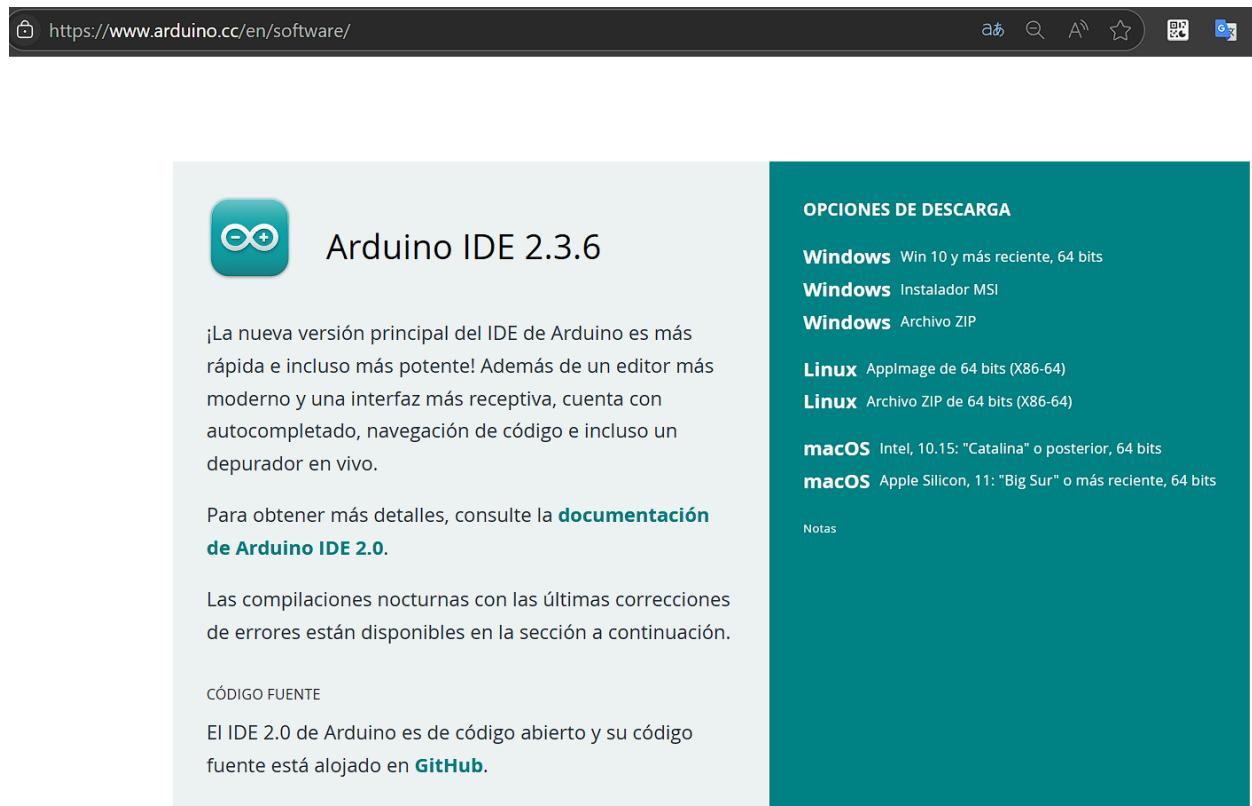


Figura 0-1. Descarga de la versión de Arduino

Una vez tengamos la versión de Arduino instalada entraremos en el IDE y en la sección Archivo->Preferencias añadimos la url que se muestra a continuación:

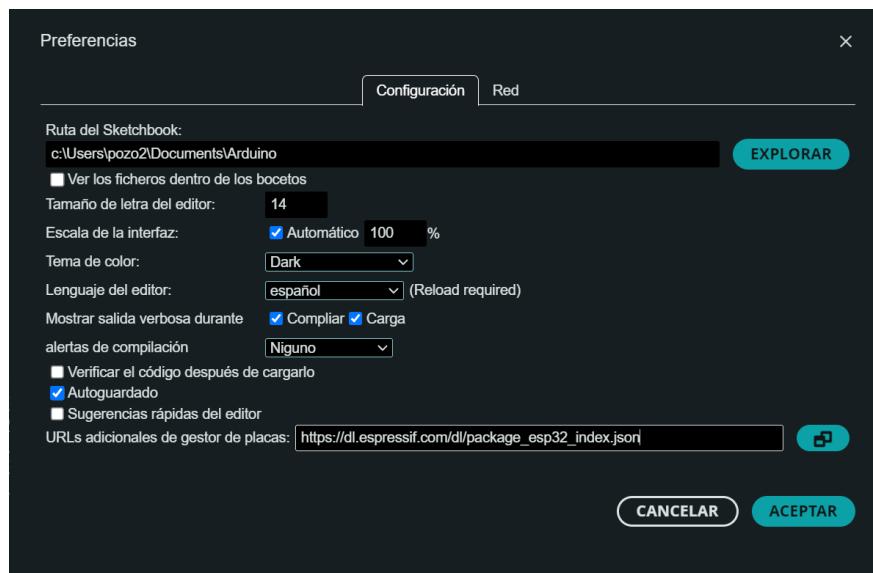


Figura 0-2. Configuración de la Url de Arduino

A continuación, procedemos a elegir, en las opciones de la izquierda, el gestor de placas y elegimos esp32

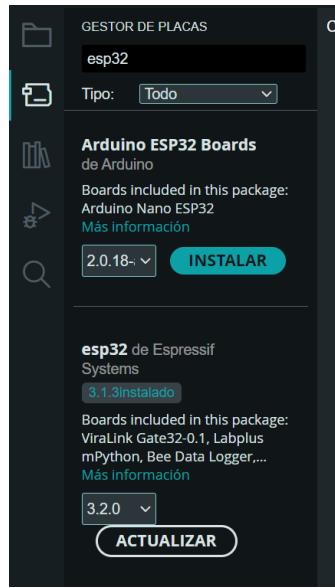


Figura 0-3. Gestor de placas

Una vez que tenemos el software de Arduino configurado, procedemos al montaje de los elementos físicos, actuadores y sensores. Para ello a continuación se muestran las figuras del montaje real del sistema y un esquema del montaje virtual.

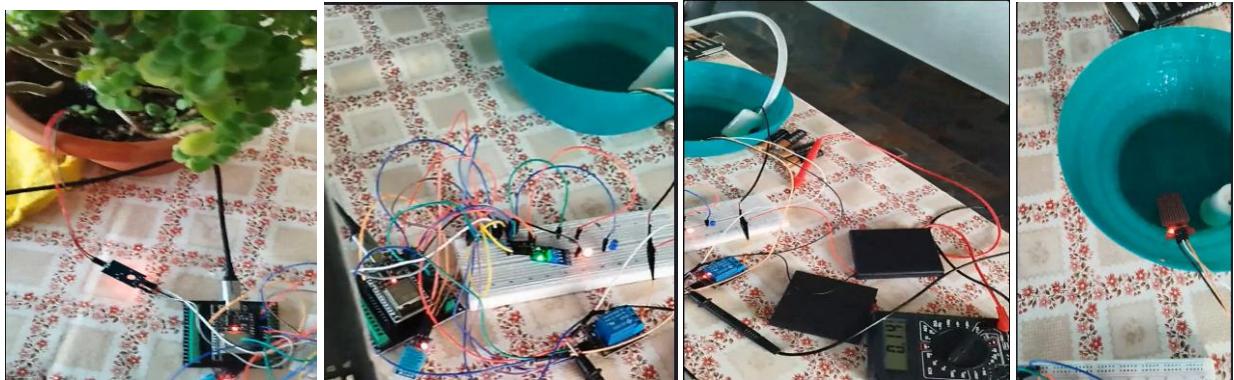


Figura 0-4. Montaje del sistema

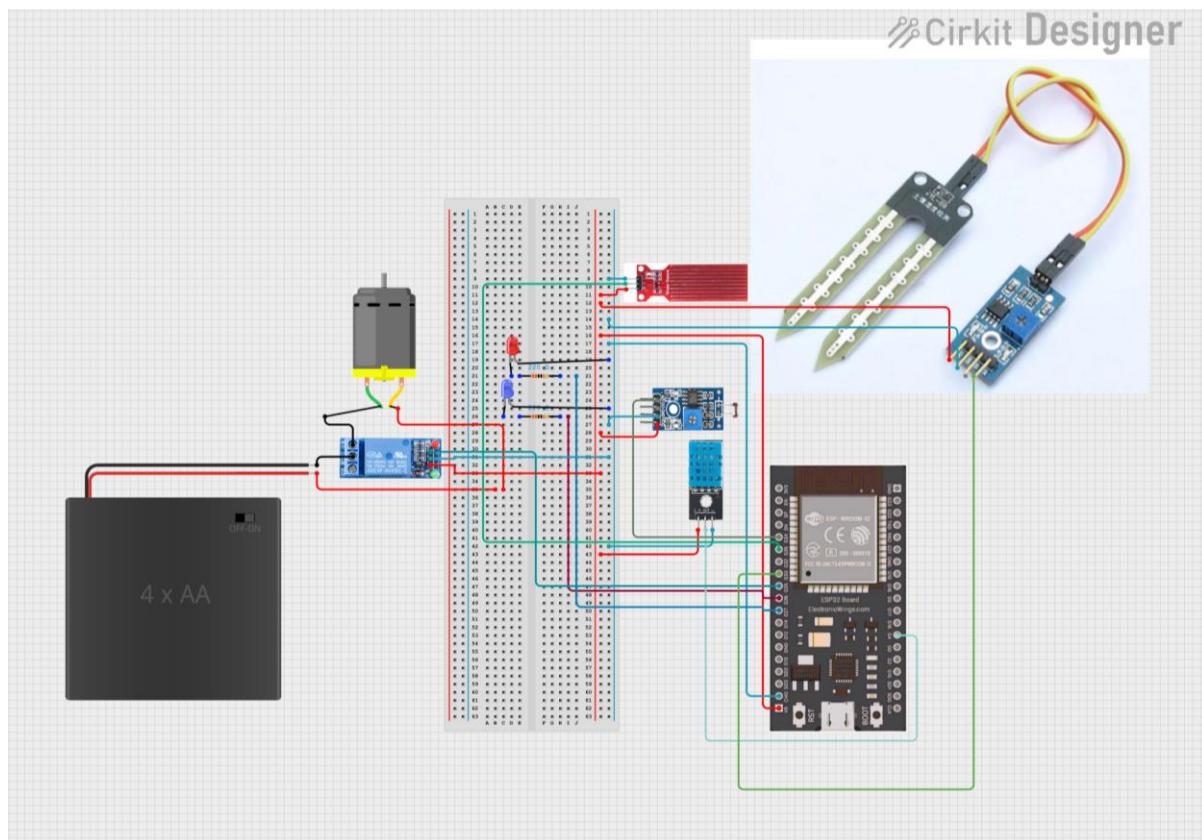


Figura 0-5. Esquema del sistema

ANEXO C: CONFIGURACIÓN DE LA BASE DE DATOS

En este anexo se explican los pasos a seguir para la configuración y la puesta en funcionamiento de la base de datos PostgreSQL.

Cuando se instala el servidor PSQL, se crea un superusuario postgres para el servidor de la base de datos. Como vamos a usar el usuario dit, debemos darle permisos modificando el fichero pg_hba.conf que se encuentra en la ruta /etc/postgresql/14/main/ y añadimos en local la línea que termina en md5 para cambiar la configuración de autenticación de los usuarios.

```
# Database administrative login by Unix domain socket
local  all      postgres          peer
# TYPE  DATABASE   USER      ADDRESS            METHOD
#
# "local" is for Unix domain socket connections only
local  all      all              md5
local  all      all              peer
# IPv4 local connections:
host   all      all      127.0.0.1/32        md5
# IPv6 local connections:
host   all      all      ::1/128           md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
#local  replication  postgres      peer
#host  replication  postgres      127.0.0.1/32    md5
#host  replication  postgres      ::1/128          md5
```

Una vez configurado los permisos ejecutamos el script bbdd.sql que nos crea nuestra base de datos necesaria para el proyecto. El script bbdd.sql se muestra a continuación:

```
DROP DATABASE IF EXISTS registro_invernadero;

CREATE DATABASE registro_invernadero;

GRANT ALL PRIVILEGES ON DATABASE registro_invernadero TO dit;

ALTER DATABASE registro_invernadero OWNER TO dit;

DROP TABLE IF EXISTS horas_riego CASCADE;
DROP TABLE IF EXISTS limites CASCADE;
DROP TABLE IF EXISTS limites_horas_riego CASCADE;
DROP TABLE IF EXISTS estado_motor CASCADE;
DROP TABLE IF EXISTS eventos_motor CASCADE;
DROP TABLE IF EXISTS lecturas_sensores CASCADE;
DROP TABLE IF EXISTS usuarios CASCADE;

DROP INDEX IF EXISTS idx_limites_actualizacion;
DROP INDEX IF EXISTS idx_usuarios_nombre;
DROP INDEX IF EXISTS idx_lecturas_fecha;
DROP INDEX IF EXISTS idx_motor_estado;
DROP INDEX IF EXISTS idx_eventos_accion;
DROP INDEX IF EXISTS idx_horas_riego_limite;
```

```

ALTER TABLE IF EXISTS usuarios DROP CONSTRAINT IF EXISTS uk_email;
-- Tabla de Usuarios
CREATE TABLE IF NOT EXISTS usuarios (
    id SERIAL PRIMARY KEY,
    nombre_usuario TEXT UNIQUE NOT NULL,
    contraseña TEXT NOT NULL,
    email VARCHAR(100) NOT NULL,
    activo BOOLEAN DEFAULT TRUE,
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    ultimo_acceso TIMESTAMP,
    nivel_acceso INTEGER DEFAULT 1
);

ALTER TABLE usuarios
ADD CONSTRAINT uk_email UNIQUE (email);

-- Tabla para el registro de las lecturas de los sensores
CREATE TABLE IF NOT EXISTS lecturas_sensores (
    id SERIAL PRIMARY KEY,
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    nivel_agua INTEGER NOT NULL,
    luz_ambiente INTEGER NOT NULL,
    humedad_suelo INTEGER NOT NULL,
    humedad_ambiental DOUBLE PRECISION NOT NULL,
    temperatura_c DOUBLE PRECISION NOT NULL,
    temperatura_f DOUBLE PRECISION NOT NULL
);

-- Tabla para el registro del estado del motor
CREATE TABLE IF NOT EXISTS estado_motor (
    id SERIAL PRIMARY KEY,
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    estado BOOLEAN NOT NULL,
    duracion_segundos INTEGER,
    consumo_agua DOUBLE PRECISION DEFAULT 0.0,
    motivo VARCHAR(20) NOT NULL
);

-- Tabla para el registro de los eventos del motor
CREATE TABLE IF NOT EXISTS eventos_motor (
    id SERIAL PRIMARY KEY,
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    accion VARCHAR(9) CHECK (accion IN ('ENCENDIDO', 'APAGADO')) NOT
NULL,
    motivo VARCHAR(20) NOT NULL,
    duracion_segundos INTEGER,
    consumo_agua DOUBLE PRECISION DEFAULT 0.0
);

-- Tabla Límites
CREATE TABLE IF NOT EXISTS limites (
    id SERIAL PRIMARY KEY,
    fecha_actualizacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    temp_min DOUBLE PRECISION NOT NULL,

```

```
temp_max DOUBLE PRECISION NOT NULL,  
humedad_amb_min DOUBLE PRECISION NOT NULL,  
humedad_amb_max DOUBLE PRECISION NOT NULL,  
humedad_suelo_min INTEGER NOT NULL,  
humedad_suelo_max INTEGER NOT NULL,  
luz_min INTEGER NOT NULL,  
luz_max INTEGER NOT NULL,  
consumo_agua_max DOUBLE PRECISION NOT NULL DEFAULT 10.0,  
volumen_agua_min DOUBLE PRECISION NOT NULL DEFAULT 20.0,  
usuario_id INTEGER NOT NULL REFERENCES usuarios(id) ON DELETE  
CASCADE,  
    usuario_actualizacion TEXT NOT NULL  
);  
  
-- Tabla secundaria para las Horas de riego  
CREATE TABLE IF NOT EXISTS limites_horas_riego (  
    id SERIAL PRIMARY KEY,  
    limite_id INTEGER NOT NULL REFERENCES limites(id) ON DELETE  
CASCADE,  
    hora_riego TIME NOT NULL  
);  
  
CREATE INDEX idx_lecturas_fecha  
ON lecturas_sensores(fecha_creacion);  
CREATE INDEX idx_motor_estado ON estado_motor(estado);  
CREATE INDEX idx_eventos_accion ON eventos_motor(accion);  
CREATE INDEX idx_limites_actualizacion  
ON limites(fecha_actualizacion);  
CREATE INDEX idx_usuarios_nombre ON usuarios(nombre_usuario);  
CREATE INDEX idx_horas_riego_limite  
ON limites_horas_riego(limite_id);  
  
INSERT INTO usuarios (nombre_usuario, contraseña, email,  
nivel_acceso)  
VALUES ('admin', 'admin','joseantoniopozo03@gmail.com', 3);
```

ANEXO D: CONSTRUCCIÓN DEL SISTEMA

En este anexo se detallarán los pasos a seguir para conseguir la puesta en marcha del sistema.

En primer lugar, debemos acceder a este repositorio GitHub <https://github.com/jospozpri1/TFG> y descargarnos las carpetas o los .zip como se prefiera, que contienen los códigos de los programas.

Una vez tengamos los ficheros vamos a ir usándolos de uno en uno:

- Docker contiene el fichero docker-compose y la carpeta fiware-openhab-bridge. Usando el Docker-compose y siguiendo los pasos del Anexo A se instalará en Docker el contenedor para el Context Broker. En la carpeta se encuentran los ficheros necesarios para el puente que conecta con OpenHab, para ello nos situamos en la carpeta y ejecutamos “docker build -t fiware-openhab-bridge”.
- Codigo_TFG contiene el código necesario para el funcionamiento del esp32. Una vez que hemos seguido los pasos del Anexo B y tenemos configurado todos los elementos físicos, conectamos el esp32 y configuramos el id y la contraseña del wifi, así como la ip que se le asigna al esp32 de manera que concuerde con nuestra red wifi. Por último, configuramos la zona horaria en el código y lo cargamos a nuestro esp32.
- Src es la carpeta que contiene todas las clases .java y los layouts necesarios para la ejecución de la aplicación Android. Para su uso, en Android Studio se crea un proyecto Empty Views Activity, con nombre Invernadero y quedando el package como com.dam.invernadero y se selecciona lenguaje JAVA y lenguaje de configuración Groovy DSL. A continuación, se muestra un ejemplo de creación de un proyecto:

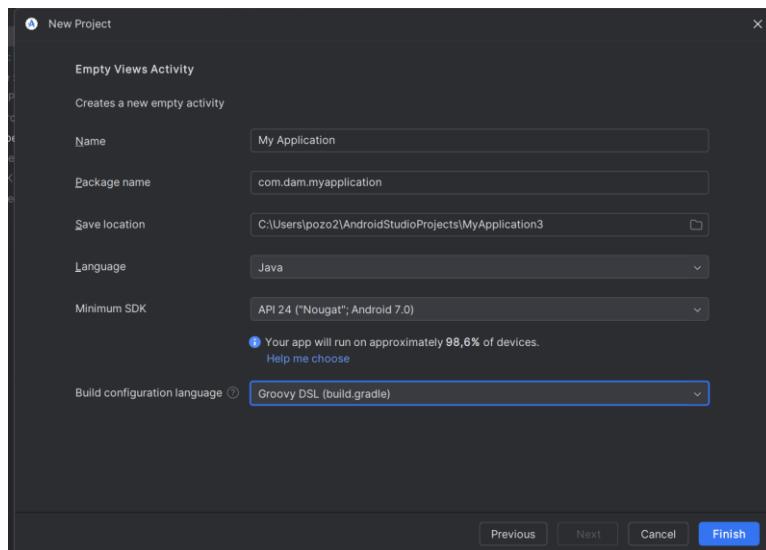


Figura 0-1. Nuevo proyecto Android Studio.

Una vez creado el proyecto sustituimos los elementos de la carpeta src por los nuestros y añadimos en el build.gradle las librerías que aparecen en la Figura 5 13. Librerías del fichero build.gradle. Por último, modificamos las direcciones ip del servidor por la dirección ip de nuestro servidor.

- TFG contiene los ficheros necesarios para la ejecución del servidor Rest y la base de datos. En primer lugar, seguimos los pasos del Anexo C y una vez tengamos configurado la base de datos, modificamos el fichero properties que se encuentra en la siguiente ruta TFG\invernadero\src\main\resources si queremos hacer algún cambio. Además, modificamos en los ficheros de implementación de los servicios la ip del Context Broker por nuestra ip. Por último, se inicia el servidor moviéndonos a la carpeta de invernadero desde la terminal e introduciendo el siguiente comando “mvn spring-boot:run”.

```

dit@localhost:~$ cd TFG/invernadero/
dit@localhost:~/TFG/invernadero$ mvn spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] -----> com.greenhouse:registro-invernadero <-----
[INFO] Building registro-invernadero 0.0.1-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] >>> spring-boot-maven-plugin:2.7.0:run (default-cli) > test-compile @ registro-invernadero >>>
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:resources (default-resources) @ registro-invernadero ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.10.1:compile (default-compile) @ registro-invernadero ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:testResources (default-testResources) @ registro-invernadero ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] skip non existing resourceDirectory /home/dit/TFG/invernadero/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.10.1:testCompile (default-testCompile) @ registro-invernadero ---
[INFO] No sources to compile
[INFO]
[INFO] <<< spring-boot-maven-plugin:2.7.0:run (default-cli) < test-compile @ registro-invernadero <<<
[INFO]
[INFO] --- spring-boot-maven-plugin:2.7.0:run (default-cli) @ registro-invernadero ---
[INFO] Attaching agents: []

```

:: Spring Boot :: (v2.7.0)

Figura 0-2. Inicio del Servidor Rest

Para obtener la aplicación de OpenHAB debemos descargarnos de la página oficial de OpenHAB la versión disponible para Windows (Stable actualmente la 4.3.5).

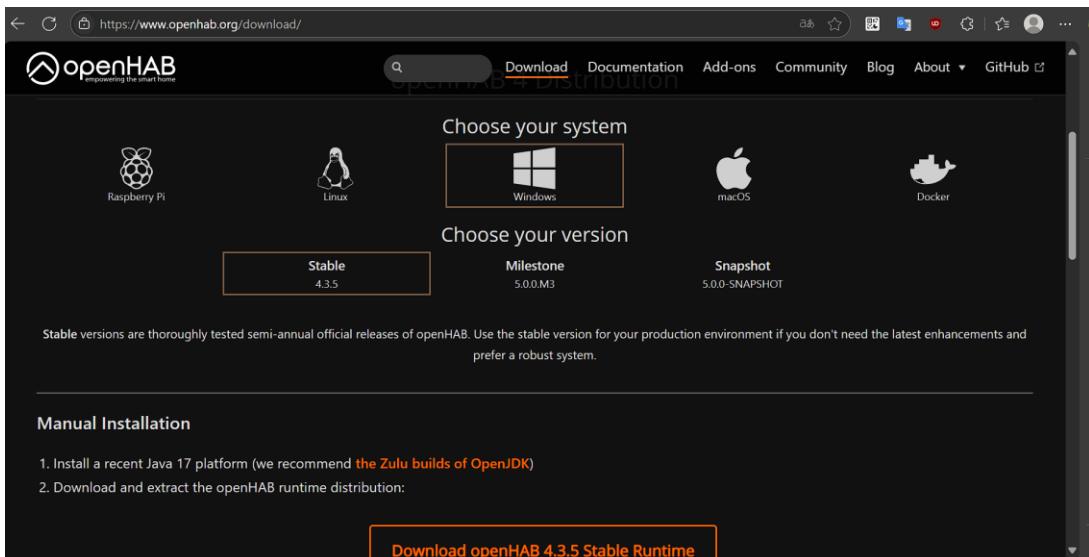


Figura 0-3. Instalar OpenHAB desde la página oficial

Una vez nos hemos vez descargado el zip y extraído se pulsa en el start de Windows y se inicia Openhab como se indica en la Figura 0 4. Iniciar OpenHab. Una vez iniciado la aplicación, nos conectamos a la dirección ip (o al localhost) y a su puerto y creamos el usuario y la contraseña y en la ruta openhab\userdata metemos la carpeta jsondb disponible en git que contiene las reglas y los ítems usados.

Nombre	Fecha de modificación	Tipo	Tamaño
addons	15/02/2025 13:01	Carpeta de archivos	
conf	15/02/2025 13:37	Carpeta de archivos	
runtime	15/02/2025 12:54	Carpeta de archivos	
userdata	15/02/2025 13:41	Carpeta de archivos	
LICENSE	15/02/2025 12:55	Documento de texto	14 KB
start	15/02/2025 12:55	Archivo por lotes de Windows	1 KB
start	15/02/2025 12:55	Archivo de origen SH	1 KB
start_debug	15/02/2025 12:55	Archivo por lotes de Windows	1 KB
start_debug	15/02/2025 12:55	Archivo de origen SH	1 KB

Figura 0-4 Iniciar OpenHab

REFERENCIAS

- [1] ibm, «¿Qué es el Internet de las cosas (IoT)? | IBM,» [En línea]. Available: <https://www.ibm.com/mx-es/topics/internet-of-things>. [Último acceso: 2025 06 19].
- [2] Oracle, «Oracle España,» [En línea]. Available: <https://www.oracle.com/es/internet-of-things/>. [Último acceso: 2025 06 19].
- [3] marketsandmarkets, «Agriculture IoT Market Size & Share Industry Growth Analysis 2032,» [En línea]. Available: <https://www.marketsandmarkets.com/Market-Reports/iot-in-agriculture-market-199564903.html>. [Último acceso: 06 05 2025].
- [4] Docker, «What is a Container? | Docker,» [En línea]. Available: <https://www.docker.com/resources/what-container/>. [Último acceso: 07 05 2025].
- [5] Fiware-Orion, «FIWARE NGSIV2 Orion API Specification - Fiware-Orion,» [En línea]. Available: <https://fiware-orion.readthedocs.io/en/master/orion-api.html#subscription>. [Último acceso: 08 05 2025].
- [6] Fiware-Orion, «API Walkthrough - Fiware-Orion,» [En línea]. Available: https://fiware-orion.readthedocs.io/en/master/user/walkthrough_apiv2.html#subscriptions. [Último acceso: 08 05 2025].
- [7] Spring, «Using the @SpringBootApplication Annotation :: Spring Boot,» [En línea]. Available: <https://docs.spring.io/spring-boot/reference/using/using-the-springbootapplication-annotation.html>. [Último acceso: 08 05 2025].
- [8] Spring, «Task Execution and Scheduling :: Spring Framework,» [En línea]. Available: <https://docs.spring.io/spring-framework/reference/integration/scheduling.html#scheduling-enable-annotation-support>. [Último acceso: 08 05 2025].
- [9] Oracle, «Java Persistence API,» [En línea]. Available: <https://www.oracle.com/java/technologies/persistence-jsp.html>. [Último acceso: 09 05 2025].
- [10] Spring, «Spring Data,» [En línea]. Available: <https://spring.io/projects/spring-data>. [Último acceso: 09 05 2025].
- [11] Docker, «Windows | Docker Docs,» [En línea]. Available: <https://docs.docker.com/desktop/setup/install/windows-install/>. [Último acceso: 18 05 2025].

