



# UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

## TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Recogida, almacenamiento y uso de datos proporcionados por sensores localizados de manera descentralizada.

Realizado por: Alberto Díez Frías

Tutelado por: María Vico Pascual Martínez Losa

Logroño, Junio, 2023

## **RESUMEN DE LA MEMORIA**

El objetivo de este proyecto es la recolección de datos y activación de maquinaria en lugares remotos que tengan conexión a internet. Los usuarios tendrán una cuenta en una página web donde podrán ver los lugares que tienen asignados, manipular interruptores para activar maquinaria en esos lugares remotos y visualizar los datos recolectados en esos lugares. Algunos de estos lugares pueden tener conexiones a internet deficientes por lo que puede haber caídas. Por lo tanto, los datos deben ser guardados localmente en los propios lugares y en un servidor central. El servidor incorporará una plataforma web cuyo funcionamiento estará basado en microservicios para lograr una alta modularidad y capacidad para futuras migraciones. Para desarrollar el software solamente serán utilizadas herramientas de libre uso y, en el caso del hardware, solo serán empleados elementos que podrán ser fácilmente sustituibles para prevenir una posible falta futura de existencias.

## **ABSTRACT**

The aim of this project is to collect data and activate machinery at remote locations that have an internet connection. Users will have an account on a website where they will be able to view their assigned locations, manipulate switches to activate machinery at these remote locations and view the data collected at these locations. Some of these locations may have poor internet connections and therefore may have outages. Therefore the data must be stored both locally and on a central server. The server will include a web platform based on microservices to achieve both high modularity and capacity for future migrations. In order to develop the software the any tools used will be free software. Regarding to the hardware, to prevent a possible future lack of stock, only easily replaceable components will be used

# INDICE

CAPÍTULO 1: INTRODUCCIÓN.....	1
1.1 OBJETIVOS DEL PROYECTO.....	1
1.2 ARQUITECTURA DEL SISTEMA.....	2
1.3 ESTUDIO DE LAS DIFERENTES TECNOLOGÍAS.....	3
1.4 COMPONENTES HARDWARE PARA RECOPILAR LOS DATOS.....	4
1.5 OTROS COMPONENTES Y SENsoRES.....	6
1.6 METODOLOGÍA ESCOGIDA PARA EL DESARROLLO.....	9
1.7 TOPOLOGÍA DEL SISTEMA.....	10
1.8 PLANIFICACIÓN MEDIANTE DIAGRAMA DE GANTT.....	11
1.9 PLANIFICACIÓN.....	12
CAPÍTULO 2: DESARROLLO.....	13
2.1 PRIMER SPRINT.....	13
2.2 SEGUNDO SPRINT.....	18
2.3 TERCER SPRINT.....	26
2.4 CUARTO SPRINT.....	29
2.5 QUINTO SPRINT.....	30
2.6 SEXTO SPRINT.....	32
2.7 SEPTIMO SPRINT.....	36
2.8 OCTAVO SPRINT.....	40
2.9 NOVENO SPRINT.....	44
CAPÍTULO 3: APARTADOS FINALES.....	49
3.1 ESTRUCTURA Y RELACIONES DE LOS MÓDULOS DE ANGULAR.....	49
3.2 COMPARACIÓN DE LA PLANIFICACIÓN Y LA REALIDAD.....	50
3.3 CONCLUSIONES.....	51
3.4 BIBLIOGRAFÍA.....	52
ANEXO 1: TÉRMINOS Y DEFINICIONES POR ORDEN ALFABÉTICO.....	54
ANEXO 2: APORTACIÓN DE LOS DIFERENTES ENPOINTS.....	57

## CAPÍTULO 1: INTRODUCCIÓN

### 1.1 OBJETIVOS DEL PROYECTO

Las zonas rurales tienen una gran limitación en conectividad, y comunicación. Hay grandes áreas de terreno prácticamente despoblado, en muchas ocasiones muy distante y con baja cobertura de conexión a internet. Hay soluciones en el mercado para esto, pero en la mayoría de ocasiones estas requieren pagar una cuota mensual o una conexión a internet constante y cualquiera de estas dos condiciones no siempre se cumplen. Además, está el problema de la brecha digital, que es especialmente problemática en los entornos rurales.

Para intentar minimizar lo anterior, se ha pensado en diseñar e implementar un sistema que permita:

- Realizar la recogida de datos en distintos lugares de la geografía y almacenar estos datos de forma centralizada.
- Realizar determinadas acciones en esos lugares como forma de retroalimentación de los datos recogidos.
- Permitir que se sigan recogiendo los datos aunque la red esté caída de forma temporal.
- Sea fácil de manejar por las personas que tengan pocas habilidades tecnológicas y pocos recursos para invertir en tecnología.

Un ejemplo sería el de un agricultor que quiere comprobar el estado de su plantación. Puede ver en tiempo real y desde cualquier lugar, si su plantación está inundada, si hay fuego o si hay un escape de gas. Incluso puede hacer funcionar maquinaria de forma remota, como encender un motor que evaque el agua si el sensor de humedad detecta que una zona está inundada. La posición de los sensores y la maquinaria puede ser configurada por o para cada usuario, por lo que podrá ser aplicado en otros muchos ámbitos como hogares o fábricas. Otro objetivo del proyecto es la internacionalización, ya que con una traducción al inglés se consigue expandir mucho los posibles mercados, además de que tener una aplicación desarrollada para ser traducida casi desde el principio hace que traducir a otros lenguajes sea mucho más fácil.

Para cumplir con estos objetivos será necesario administrar una base de datos distribuida. Esta base de datos debe contar con una central en la que estarán replicados todos los datos. Además, en cada lugar deben estar almacenados los datos recogidos en ese lugar para que, en el caso de que la conexión caiga de forma temporal, los datos recogidos en ese tiempo sean enviados a la central cuando la conexión sea restablecida.

También se debe cumplir que los componentes físicos del sistema deben poderse conseguir fácilmente, ser fácilmente sustituibles por otros modelos similares, por si llega el momento de no poder ser encontrados en el mercado en un momento dado. (Lo cual no es nada raro en la situación actual y probablemente también en el futuro). Para conseguir esto, cada componente debe estar aislada lo más posible del resto y las tecnologías usadas para desarrollar el sistema no deben depender de otros códigos propietarios en la medida de lo posible. Además, también se debe conseguir que la central donde se almacenen los datos pueda ser cualquier tipo de servidor u ordenador.

Para poder cumplir con el requisito de que el sistema pueda ser manejado por pocas habilidades tecnológicas y no demasiados recursos, se debe conseguir, por un lado, que se pueda acceder al sistema desde casi cualquier dispositivo, consiguiendo de esta forma que se pueda acceder desde cualquier lugar que tenga acceso a internet y también que se pueda ver en pantallas grandes. También se debe conseguir que cada usuario solo pueda ver sus correspondientes datos pero, que los responsables con suficiente nivel de permisos pueda acceder a los datos de todos los usuarios. El login en el sistema debe poder hacerse de forma fácil, a ser posible que se puedan guardar las credenciales en el dispositivo del usuario pero sin perder seguridad. Además, debe ser posible conectarse desde un gran número de sistemas operativos y a ser posible sin ningún tipo de instalación de programas.

Un aspecto muy importante del sistema será la visualización de los datos. Este tiene que mostrar rápidamente los datos más actuales, así como el historial de los datos. El proyecto podrá ampliarse creando mejoras gráficas para mostrar los históricos de los datos a lo largo del tiempo. También se podrán crear mejores gráficas de las acciones sobre maquinaria realizadas durante el tiempo. Si hay suficiente tiempo, también, se podrán crear representaciones propias de las medidas de cada sensor. También se replanteará estudiar más detenidamente la seguridad del sistema descentralizado ya que no es una de las prioridades del proyecto y, podría ser necesario revisar la seguridad, en caso de poner el proyecto en producción. En el caso de tener el suficiente tiempo, sería bastante útil hacer un diseño responsive de la aplicación para poder usarla correctamente en un móvil.

Teniendo en cuenta los objetivos planteados se ha decidido que la visualización y manipulación de los datos y la maquinaria se realizará mediante una aplicación web. De esta forma se garantiza la distribución de la aplicación en casi todo tipo de sistemas operativos, hardware y pantallas, sin que sea necesaria instalación. Hay una gran cantidad de tecnologías actualmente para desarrollar las aplicaciones web en cada una de sus múltiples capas. Por este hecho hay muchas tecnologías donde escoger en cada capa, que son suficientemente independientes de cualquier empresa que puede cambiar de estrategia en cualquier momento.

De todo lo anterior se extraen la siguiente lista de requisitos:

- El sistema controlará el acceso y lo permitirá solamente a los usuarios autorizados. Los usuarios deben ingresar al sistema con un nombre de usuario y contraseña.
- El software podrá ser utilizado en los sistemas operativos Windows y Linux.
- La aplicación debe poder utilizarse sin necesidad de instalar ningún software adicional, solamente con el navegador web.
- La aplicación debe poder utilizarse con los navegadores web Chrome, Firefox y Edge.
- Cada usuario tendrá un rol perteneciente a un grupo de roles predefinido. Dependiendo del rol tendrá unos permisos u otros dentro de la aplicación. Los roles serán los siguientes: usuario, operario y administrador.
- El rol de usuario tendrá permisos para visualizar sus propios datos generados por sus sensores en los lugares que tiene asignados.
- El rol de operario tendrá permisos para gestionar los datos de los distintos lugares, con el fin de que los datos de todos los lugares estén vigilados y gestionados. Esto puede ser de vital importancia, puesto que puede haber usuarios que no controlen debidamente sus lugares. También podrá visualizar las mediciones de los sensores y modificar las salidas.
- El rol de administrador, además, tiene permisos para gestionar los usuarios. De esta forma no tiene restricciones dentro de la aplicación. Este es el principal responsable de la aplicación.

## 1.2 ARQUITECTURA DEL SISTEMA

Para elegir las tecnologías de este proyecto primero hay que diferenciar las diferentes partes que lo van a componer.

Para la aplicación web vamos a usar el patrón modelo, vista, controlador. Se ha escogido esta arquitectura porque es una forma adecuada de estructurar las partes del sistema, sobre todo para una aplicación distribuida. Actualmente, es muy común utilizar la tecnología de los microservicios para estructurar las aplicaciones distribuidas. Este es un método usado principalmente, para estandarizar las comunicaciones entre las partes de la aplicación que están en diferente lugar y, además, desacoplar unas partes de otras.

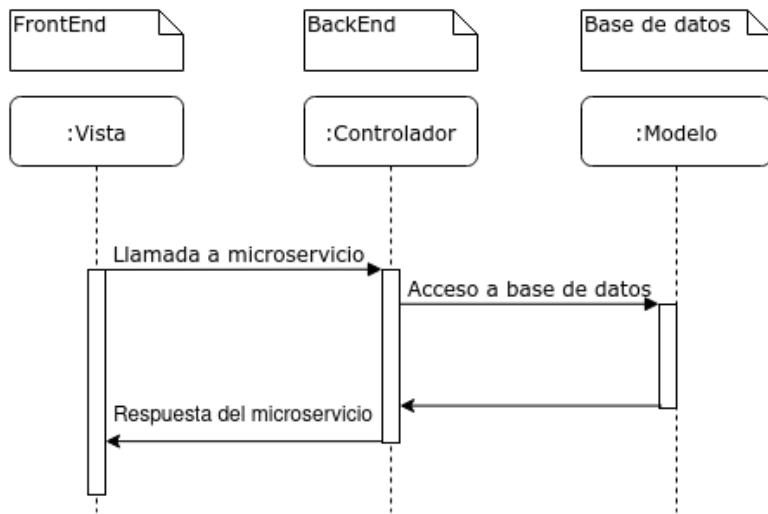
Otra posibilidad para desarrollar aplicaciones web, es generar el código HTML directamente desde el código ejecutado en el servidor, como hacen PHP u otras tecnologías.

Actualmente, para la vista se usan frameworks que consumen código y generan código JavaScript para ser ejecutado en el navegador, dejando en desuso las tecnologías que generaban el código HTML en tiempo real desde el servidor. Hay una explicación de estos términos en uno de los anexos que están al final del documento.

Hoy en día una aplicación web se suele componer de alguno de estos frameworks para el componente de la vista, que se ejecuta en el navegador en su mayor parte y, de microservicios para la comunicación con el servidor. Además, estos frameworks contienen una buena cantidad de componentes que permiten realizar tareas como simular ventanas emergentes en todo tipo de navegadores. Realizar este tipo de tareas con HTML básico y, además, que ese código funcione en todos los navegadores actuales, sería terriblemente largo y tedioso.

Por lo tanto, se escogerá como arquitectura: la de los microservicios. Una vez elegida esta arquitectura, queda decidir qué framework se usará para la parte frontal en el navegador y qué tecnologías se usarán para generar los microservicios. Un micro servicio es una llamada HTTP que se hace normalmente a un servidor para intercambio de datos. En esta arquitectura, la vista, que se ejecutará en el navegador del cliente, se comunicará con el servidor, donde residirá la mayor parte de la lógica y el acceso a la base de datos mediante microservicios.

Así, la aplicación web está dividida en backEnd y frontEnd. En el frontEnd está la vista y en el backEnd se encuentra el modelo del dominio y la mayor parte de la capa del controlador. Ahora trataremos las tecnologías de la parte del frontEnd. Esta parte es ejecutada en su mayor parte en el navegador del usuario.



### 1.3 ESTUDIO DE LAS DIFERENTES TECNOLOGÍAS

- Para desarrollar el front:

Algunos de los frameworks más populares para desarrollar el frontend son Angular, React y Vue. A continuación se incluye una comparativa entre ellos:

	Angular	React	Vue
Apoyo	Google	Facebook	Un equipo de colaboradores internacional
Filosofías	Tiene muchas características integradas en la base	Es minimalista. Tiene pocas características integradas de base. Por lo tanto, suele ser necesario integrar dependencias de terceros	La cantidad de características es intermedia a las otras dos
Sintaxis	TypeScript con template	Mezcla HTML con JavaScript	JavaScript con template
Dificultad de aprendizaje	Alta. Tiene una estructura de proyectos compleja	Alta. Proyecto con estructura propia y HTML y JavaScript mezclado	Menos compleja al crear el proyecto

A partir de esta comparativa, la opción escogida es Angular, por la mayor cantidad de características que hacen prescindir de la necesidad de la dependencia de terceros.

- Tecnologías para desarrollar microservicios

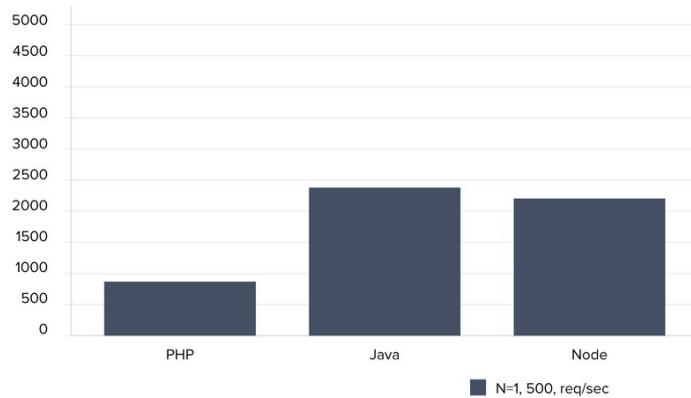
Para generar microservicios también existen diferentes tecnologías. En este caso, unas buenas opciones de estudio son Node.js, Java y PHP. Se va a realizar una comparación pensando exclusivamente en el tratamiento de microservicios, y no en las otras capacidades que tiene cada uno de estos lenguajes para otras cuestiones.

En la siguiente tabla se pueden ver algunas diferencias entre los lenguajes. Para entender la tabla hay que tener presente la diferencia entre hilos y procesos. Un proceso es la instancia de un programa en ejecución. Es administrado por el sistema operativo. Muchos procesos similares duplican memoria en la RAM para hacer acciones similares y es un gasto de recursos. Otro sistema más óptimo es usar un proceso que puede manejar muchos hilos. Estos hilos comparten parte de la memoria del proceso, por lo que comparten recursos y tiempo de ejecución de procesador. Si un hilo se queda esperando a un proceso de entrada/salida, los demás usan el tiempo de procesador que queda libre. Cuando una llamada llega a Node.js o Java estos las responden mediante la ejecución de un hilo perteneciente al único proceso de programa, mientras que en PHP se genera un nuevo proceso consumiendo más recursos por cada llamada.

Esta es una comparación entre diferentes lenguajes en cuanto a estas filosofías para gestionar microservicios:

Lenguaje	Hilos vs Procesos	No-Bloqueos	Facilidad de uso
Node.js	hilos	Sí	Requiere llamadas
Java	hilos	Sí	Requiere llamadas
PHP	procesos	No	

Este es un gráfico de rendimiento con una prueba de estress con 500 llamadas a la vez. La gráfica representa el número de llamadas respondidas por segundo. Esta gráfica se ha realizado comparando diferentes rendimientos sacados de internet. Por lo tanto, los mayores valores son los mejores:



Con estos datos y esta gráfica, el caso de PHP queda descartado, ya que su rendimiento es menor con una carga alta de trabajo dado sus bloqueos de entrada/salida en los multiprocesos.

Llegamos a la conclusión de que con una gran carga de trabajo, Java y Node son parecidos. Cada uno tiene sus ventajas sobre el otro: Java tiene mayor facilidad de uso, ya que es un lenguaje fuertemente tipado; mientras Node, que usa código JavaScript, no es tipado. Eso es una desventaja cuando los programas son complejos por la gran cantidad de errores de ejecución que se generan, pero también puede ser una ventaja, para programas simples. Por el otro lado, JavaScript y TypeScript son casi el mismo lenguaje y eso es una gran ventaja, ya que usaremos lenguajes muy parecidos tanto en el servidor como en el navegador, y esto nos facilitará mucho las cosas. También hay que tener en cuenta que Node.js usa una licencia MIT que es más libre que la usada por Java. Además, JavaScript usa JSON de forma nativa y este es el formato en el que se manejan los datos la mayoría de las veces en los microservicios, por lo tanto, facilitará la tarea. Por todo lo anterior, el lenguaje elegido para realizar los microservicios será JavaScript.

Para usar JavaScript del lado del servidor se usa el entorno Node.js. Una vez decidido este entorno hay que elegir tecnologías para implementar los microservicios y acceder a las bases de datos. Además, hay que escoger alguna tecnología que use la plataforma web para manejar los datos. La base de datos de la página será la misma que se use como central en el proyecto. Se ha decidido, por la adecuación de las mismas para este proyecto y por el conocimiento previo de ellas, usar Express para levantar los microservicios y Sequelize como tecnología ORM para acceder a la base de datos. Estas usan la licencia MIT, por lo que se pueden usar incluso de manera comercial. Hay que recordar que estos términos están definidos en uno de los anexos de este documento.

#### 1.4 COMPONENTES HARDWARE PARA RECOLLECTAR LOS DATOS

Como se ha comentado antes, este va a ser un sistema distribuido y los datos se leerán en un lugar remoto. Como en ese lugar la conexión puede fallar potencialmente, los datos se guardarán en el mismo lugar donde sean tomados. Por lo tanto, necesitamos un hardware que sea capaz de enviar y recibir datos de la red, almacenar datos de forma relativamente fiable y comunicarse con otros pequeños sistemas. Además, es muy importante que sea barato, que consuma poca electricidad y que sea fácilmente reemplazable por si hay falta de stock de esa plataforma en el mercado, tanto actual como futura. Es conveniente que este hardware sea capaz de mandar y recibir a los microservicios de forma eficiente. A día

de hoy, las alternativas son: Raspberry Pi, Asus Tinker Board S, Orange Pi y Arduino para la conexión con los elementos electrónicos:

**Raspberry Pi**: es el microordenador actual más famoso y con más soporte con mucha diferencia. Además, el precio recomendado es uno de los más competitivos. El resto de alternativas están en mayor o menor medida basadas en este. Desde, casi todos los puntos de vista, esto hace que sea la más recomendable. En el aspecto negativo está la actual disponibilidad, ya que debido a la falta de chips, es difícil conseguir una y su precio se ha multiplicado. Pero su larga trayectoria y la futura estabilización del mercado debería hacer que volviera a haber stock a precios aceptables. Y aunque esto no ocurriera, sería relativamente sencillo reemplazarla por alguna de sus competidoras.



**ASUS Tinker Board S**: es una alternativa más cara y potente. Para este proyecto no será necesaria esa potencia extra, pero es una opción a considerar. Hay que tener en cuenta que no necesita una tarjeta SD para funcionar, ya que tiene su propia memoria de 16GB por lo que se podría descontar la tarjeta SD del precio.

**Orange Pi**: una alternativa muy similar a Raspberry Pi en características, y a un precio muy competitivo. También se puede prescindir de tarjeta SD. Sería su sucesora más inmediata.

Como se puede ver en las imágenes, los microordenadores tienen patillas que no suelen tener un ordenador normal. Estas patillas se pueden configurar como entradas o salidas digitales. Para entradas o salidas digitales es posible usar fácilmente baratos convertidores digitales a analógicos y viceversa. Sin embargo, se ha barajado y aceptado otra opción más aconsejable para un proyecto grande: usar otro componente intermedio a los sensores para proteger al microordenador de sobre voltajes y no complicar el montaje con muchos componentes diferentes.

**Arduino**: es una pequeña placa barata y fácil de encontrar y, si es necesario, de reemplazar (no sería demasiado difícil porque su programación es muy simple). Además de que es muy poco probable que sea necesario reemplazar, ya que su licencia es totalmente libre y es fabricado por muchas empresas diferentes en múltiples países. Tiene múltiples formatos que se pueden escoger según las necesidades del proyecto. El modelo más común y el que usaremos en el proyecto es el de la foto, el Arduino Uno:



Hay varios motivos para usar esta placa como ampliación de un microordenador con entradas y salidas integradas. Además de contar directamente con entradas y salidas analógicas, es más barato y funciona a una frecuencia muchísimo más baja que un microordenador. Estas dos características hacen que al sufrir sobre tensiones el reemplazo de hardware sea más barato y, como su frecuencia y consumo son mucho menores, es posible estar leyendo las entradas constantemente y captar casi cualquier cambio en los valores, aunque estos sean muy cortos en el tiempo, sin sobrecalentarse.

## 1.5 OTROS COMPONENTES Y SENSORES

Además de microordenadores y microcontroladores, es necesario otro tipo de componentes electrónicos para interactuar con el mundo real. Se ha escogido un grupo de sensores que cubren un buen número de tipos de mediciones, cubriendo así un gran número de casos, así como una placa de relés para tener influencia sobre el mundo real. Además, está el problema de mantener la hora en los dispositivos electrónicos, ya que muchos de los dispositivos anteriores no tienen batería, con lo que no pueden mantener la hora en caso de desconexión, y esto es necesario para saber la hora en la que fueron tomadas las mediciones.

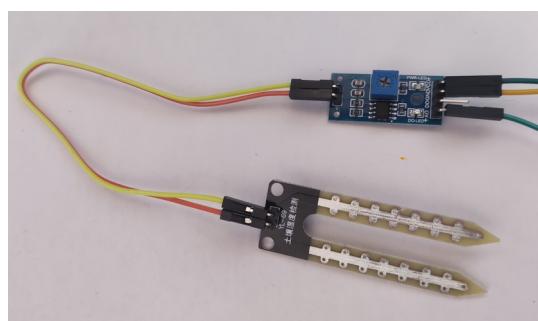
En el caso de la hora de la Raspberry Pi, usaremos un componente extra para que conserve la hora actual en caso de que haya una caída de la red eléctrica y de internet. Como este microordenador se distribuye sin ningún tipo de batería, no tiene forma de conservar la hora si es apagada y no está conectada a ninguna red de donde pueda obtener la hora actual. Para esto se utilizará el módulo DS1307. Este módulo es usado para mantener la hora:



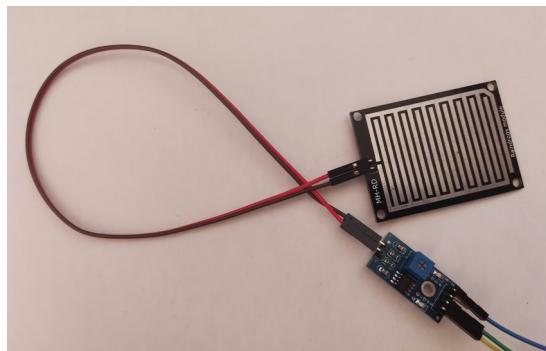
En el caso de Arduino se usarán un buen número de sensores, para cubrir múltiples tipos de mediciones, y una placa de relés, para mandar salidas al exterior. Las diferentes lecturas que serán realizadas con Arduino, serán, simplemente, lecturas analógicas o digitales y se almacenarán como tales. Por lo tanto, el sistema de lectura para todos estos sensores será el mismo. Un caso especial es el sensor de humedad y temperatura, se usará una librería externa. Pero esto no será un problema, ya que está distribuida bajo licencia MIT.

A continuación se listarán los diferentes sensores escogidos con una breve explicación:

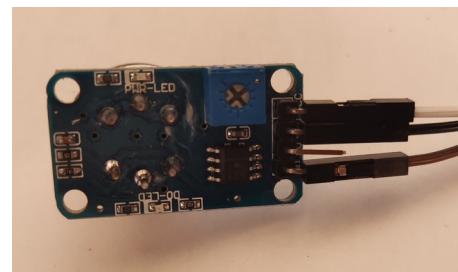
- Sensor de aceite: es bastante simple. Consta de dos conductores conectados a un regulador electrónico, el cual retorna una salida analógica, que indica la resistencia que hay entre los dos conductores, y, una salida digital, que manda un pulso alto o bajo si la resistencia sobrepasa un cierto límite que se ajusta mediante el potenciómetro de la imagen. De esta forma, la resistencia entre los conductores que se ven en la parte de abajo de la imagen, depende de lo que se encuentra entre ellos. También tiene otra salida analógica que puede ser usada. Usaremos la salida analógica. Será usado para detectar líquidos que, en el caso de existir, estarán en contacto con los dos conductores y, por lo tanto, circulará la electricidad.



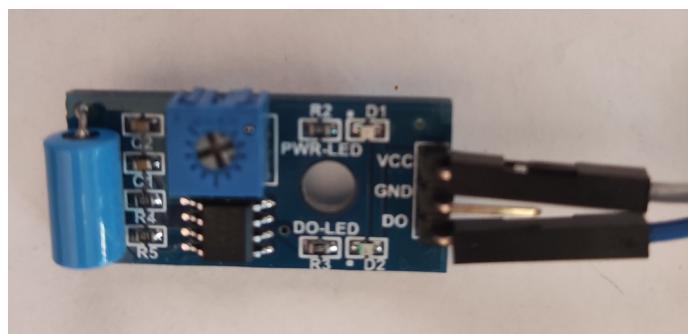
- Sensor de lluvia: muy similar al anterior. La diferencia está en la forma de los dos conductores. Con esta forma, la resistencia entre los conductores baja cuando caen gotas en el sensor, ya que dada la forma, cada gota hace contacto en los dos conductores a la vez. También tiene salida analógica y digital con potenciómetro para regular el nivel en el que se pasa del estado alto al estado bajo. De este se tomarán medidas analógicas.



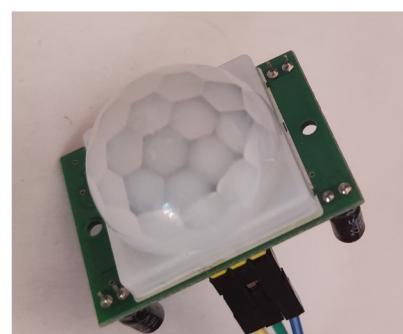
- Sensor de gases inflamables MQ-2: este sensor detecta los típicos gases inflamables, como los usados por mecheros o cocinas de gas. Las sustancias detectadas son metano, butano, GLP o humo. Sus salidas son similares a las salidas de los sensores anteriores, por lo tanto, lleva un potenciómetro para regular el nivel analógico en el que la salida se pone al nivel alto o bajo. De este se tomarán medidas analógicas.



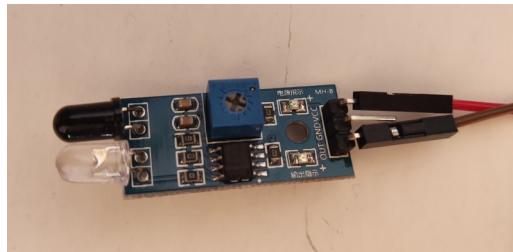
- Sensor de vibración: se activa en el caso de que sea agitado con fuerza. En este caso solamente tiene una salida digital. La sensibilidad de esta salida está regulada por el potenciómetro de la imagen. Por lo tanto, de este sensor solo se podrán guardar valores digitales:



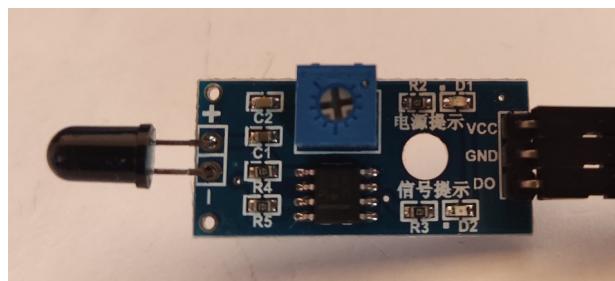
- Sensor de personas HC-SR501: detecta personas o animales mediante infrarrojos. Se activa cuando alguien se mueve en la franja donde es sensible. Puede detectar movimiento entre 3 y 7 metros de distancia. Los dos potenciómetros que lleva son para regular la sensibilidad de detección de entre 3 a 7 metros y la demora de la detección, que es el tiempo necesario de activación para que su salida se ponga a nivel alto. De este sensor se tomarán medidas analógicas.



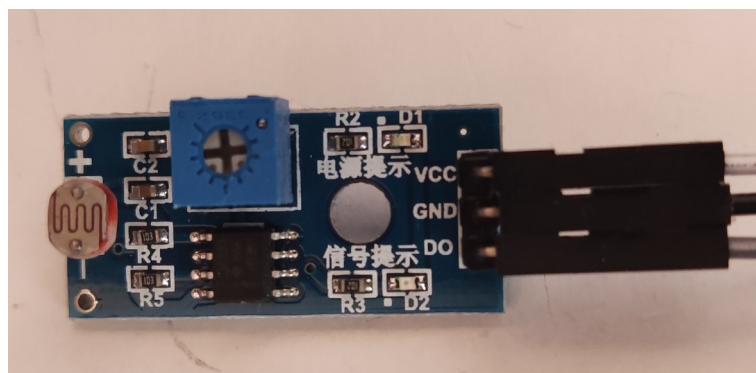
- Sensor de obstáculos IR: consiste en dos diodos: uno emisor de luz y otro receptor. El emisor emite una luz de infrarrojos que no puede ver el ojo humano, que es reflejada si hay algo justo enfrente y, entonces, el receptor la recibe y se activa. El nivel de sensibilidad es ajustado desde el potenciómetro y tiene una única salida digital.



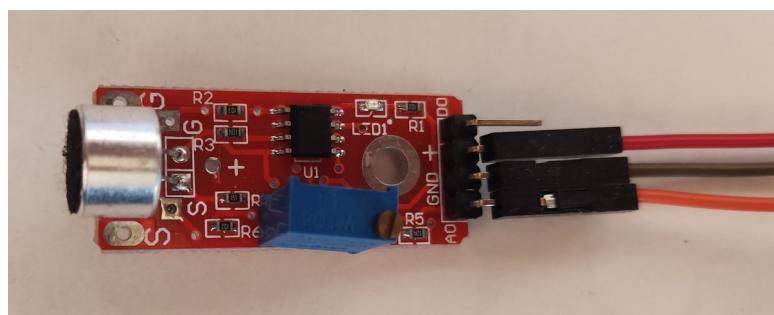
- Sensor de llama por infrarrojos para incendios: detecta la luz de las llamas y se activa cuando tiene una llama cerca. Solo tiene una salida digital cuya sensibilidad es regulada por un potenciómetro.



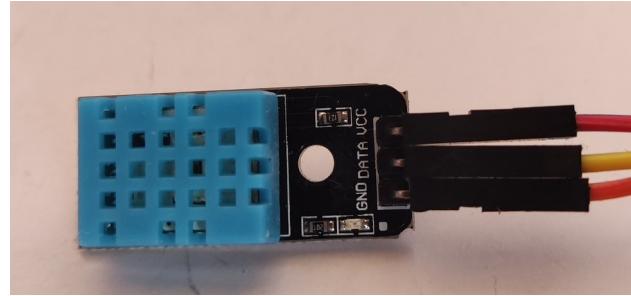
- Sensor de luz para el día y la noche: usa un foto resistor para detectar si hay luz del sol o generadas por las personas. Tiene una señal digital que puede ser regulada por el potenciómetro. Hay que tener en cuenta que es muy importante que el sensor esté orientado a la luz que se quiera detectar. Solamente tiene una salida digital.



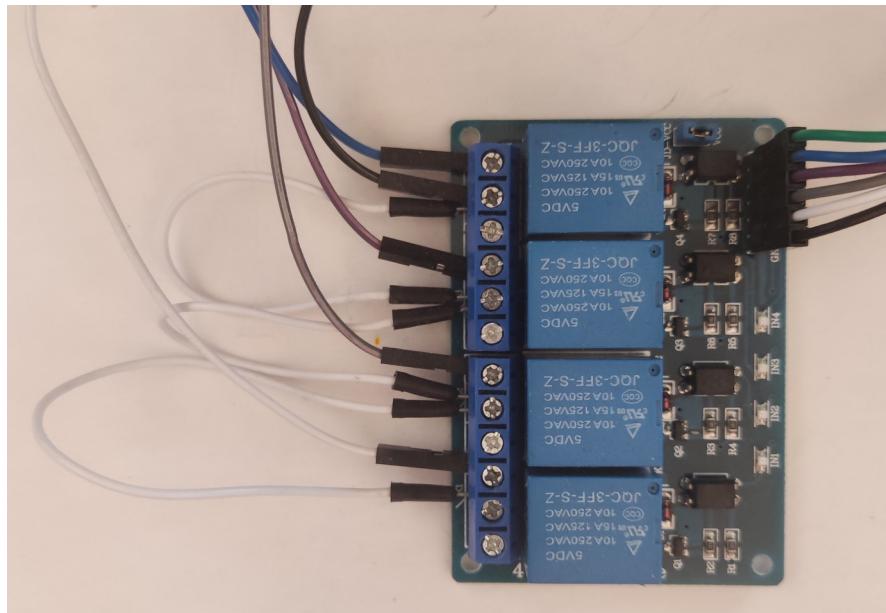
- Sensor de ruidos fuertes: no graba sonido. Solamente se activa cuando su sensor detecta que hay sonidos fuertes. En este caso, tiene salida digital como analógica. La salida digital se puede regular mediante un potenciómetro al que se le pueden dar varias vueltas para ajustar el valor deseado sea más fácil. De este se tomarán medidas analógicas.



- Sensor de humedad y temperatura DHT11: este sensor es diferente a los demás. Lee dos valores diferentes: la humedad en el ambiente y la temperatura. Solamente tiene una patilla digital para estos dos valores, por lo que es aconsejable usar una librería dedicada para manejar este sensor. Además, hay que tener en cuenta que esta librería tardará unos milisegundos en comunicarse con el sensor. Durante este tiempo las comunicaciones y el resto de componentes electrónicos conectados al Arduino quedarán desatendidos, por lo que habrá que decidir cuándo llamar a las funciones de esta librería.



- Módulo de relé para manipular maquinaria externa: todos los anteriores elementos electrónicos conectados a Arduino han sido sensores. Los sensores son entradas para Arduino mientras que este será una salida. Esta placa electrónica maneja cuatro relés que pueden ser usados para activar casi cualquier tipo de maquinaria. La placa tiene cuatro relés, por lo tanto, ocupará cuatro salidas digitales de Arduino, pudiendo activar cuatro líneas de tensión diferentes. Las salidas tienen una capacidad de 1 Amperio, por lo que con esta capacidad se pueden encender una gran cantidad de aparatos.



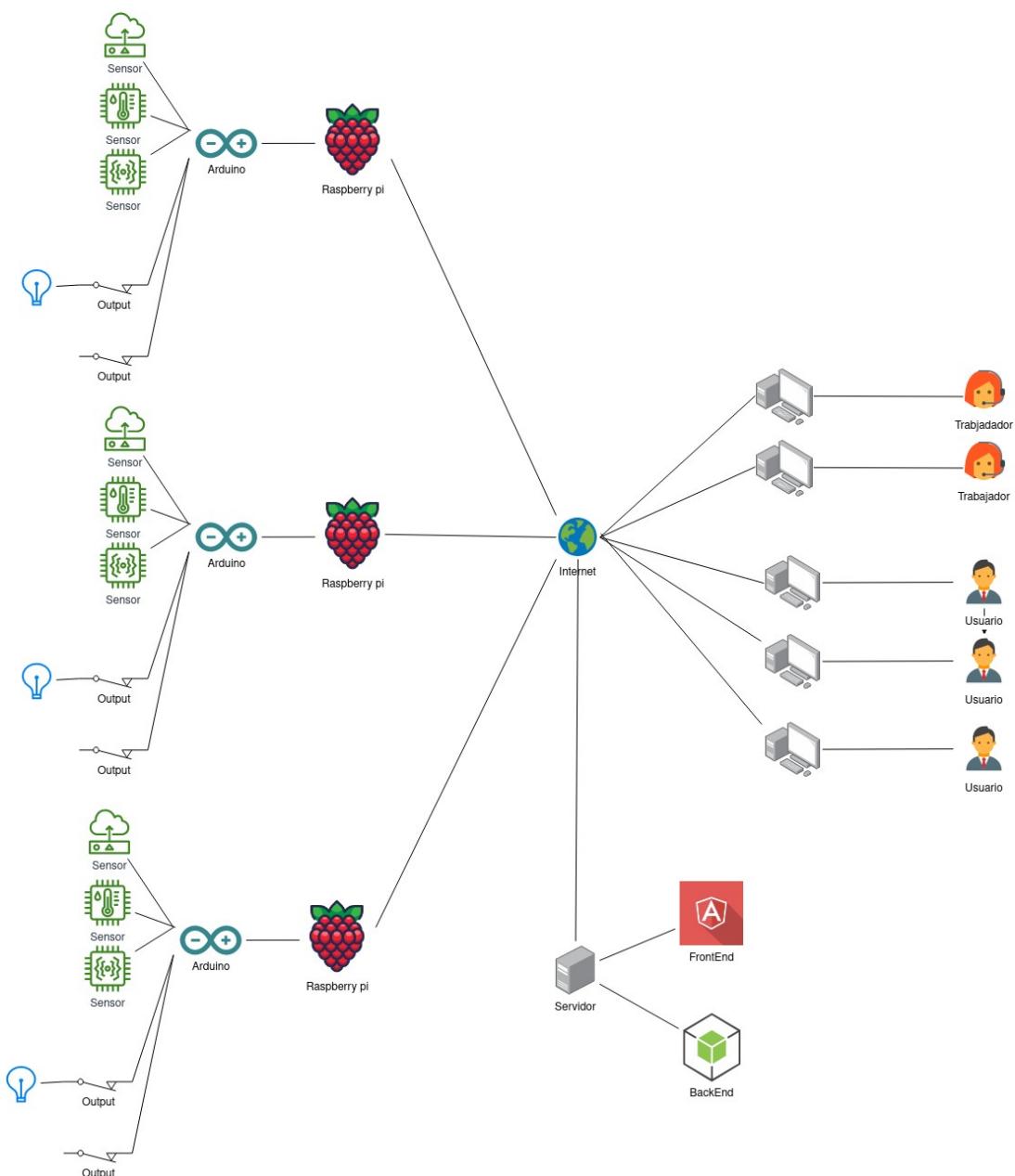
## 1.6 METODOLOGÍA ESCOGIDA PARA EL DESARROLLO

Se ha decidido utilizar una metodología incremental, puesto que, al tener una arquitectura distribuida, nos iremos centrándonos en cada una de las componentes. Además, aunque los requisitos los hemos establecido desde un proyecto propio, al no existir un cliente real, pudiera interesarnos en algún momento incluir o modificar alguno de ellos. Por ello usaremos sprints para seguir el desarrollo.

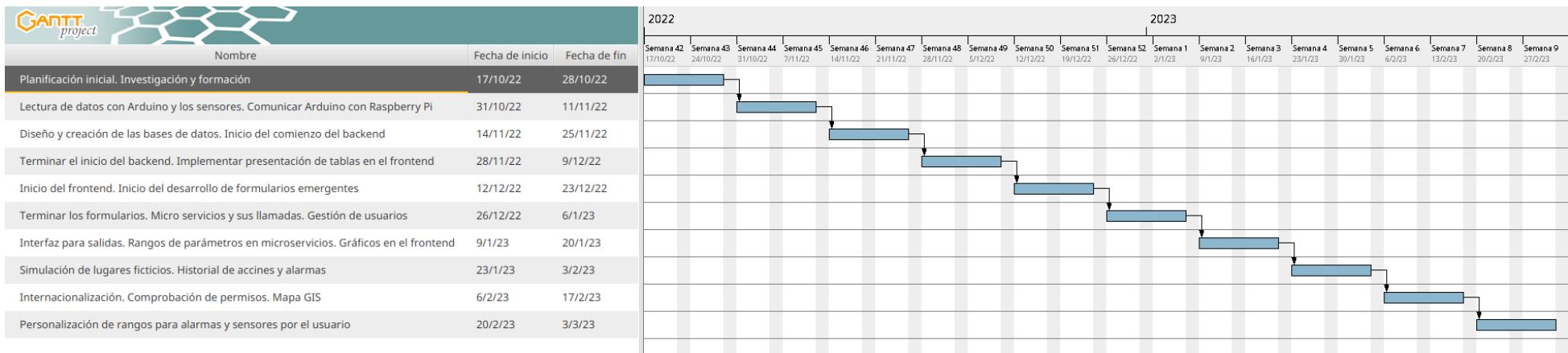
## 1.7 TOPOLOGÍA DEL SISTEMA

A continuación, se incluye un diagrama con la topología del sistema. En cada lugar donde se toman mediciones, habrá una Raspberry conectada a un Arduino y este a los otros componentes electrónicos. Las Raspberry Pi estarán conectadas a internet y se comunicarán con el servidor central. Los usuarios interactuarán con estos sistemas conectando con el servidor mediante la aplicación web. De esta forma, cuando las personas entren a la aplicación web, podrán acceder a todos los datos a los que tienen permiso, sin necesidad de acceder a los lugares donde se toman mediciones, puesto que el servidor tiene todos los datos recogidos hasta ese momento (o al menos hasta el momento en el que cada Raspberry tenía acceso a la web).

Este es un ejemplo gráfico de la topología. Cada Raspberry está conectado con un Arduino y cada Arduino está conectado a un grupo de sensores de propósito general o particular y distribuidos físicamente como el usuario responsable del lugar decida. En el servidor está todo lo necesario para almacenar los datos y para la aplicación web. Dado que la aplicación web esta compuesta por backEnd y frontEnd, en el servidor estará el backEnd y todo lo necesario para enviar el frontEnd al navegador de los usuarios. En la puesta en producción, también son necesarios otros elementos para usar el protocolo HTTPS como son el correspondiente certificado, entre otros elementos.



## 1.8 PLANIFICACIÓN MEDIANTE DIAGRAMA DE GANTT



Esta es la planificación prevista. Se han repartido las tareas en diez hitos y se han asignado dos semanas a cada uno. Teniendo en cuenta que en cada semana se invertirán 15 horas, hace un total de 30 horas por hito. Por lo tanto, hace un total de 300 horas.

En los rangos de tiempo se han puesto fechas en lugar de horas. Se han dejado los tiempos como una estimación de la fecha de finalización del proyecto, pero la verdadera condición es la cantidad de horas usadas.

## 1.9 PLANIFICACIÓN

TAREA	SUBTAREA	HORAS PREVISTAS
Planificación inicial	Recopilar las intenciones del proyecto, su alcance y la forma de afrontarlo	15
Investigación y formación	Investigar de las tecnologías y la viabilidad de cada una. Contrastar las ventajas e inconvenientes de cada tecnología	5
	Compra de los componentes, instalación del software necesario que se necesita para manipularlos	10
T 1 Lectura de datos con Arduino y los sensores	T 1.1 Montar del cableado	10
	T 1.2 Realizar las lecturas de datos de cada sensor	10
T 2 Conexión entre Arduino y Raspberry pi para persistir los datos	T 2.1 Comunicar Arduino y Raspberry pi	10
	T 2.2 Diseñar y crear de la base de datos	5
	T 2.3 Persistir datos en Raspberry pi	5
T.3 Visualización de los datos en la web	T 3.1 Diseñar la base de datos para la web	10
	T 3.2 Iniciar del backend	20
	T 3.3 Implementar la creación de tablas visibles en la página web	20
	T 3.4 Iniciar del frontend	20
T 4 Realización del login en la web y gestión de los usuarios	T 4.1 Realizar utilidad para formularios emergentes	20
	T 4.2 Crear de los microservicios	5
	T 4.3 Creación de las llamadas en frontend	5
	T 4.4 Gestión de los usuarios	10
T.5 Interacción con maquinaria remota	T 5.1 Hacer la interfaz para que el usuario pueda manejar las salidas de Arduino	10
T.6 Mostrar el historial de eventos y datos con sus filtros	T 6.1 Realizar la aceptación de intervalos de variables en los microservicios	5
	T 6.2 Realizar los filtros para recibir rangos de fechas en el frontend	5
	T 6.3 Mostrar gráficos de los datos requeridos en el frontend	10
T.7 Realización simulaciones con muchos lugares remotos donde se toman datos	T 7.1 Simular las llamadas	10
	T 7.2 Realizar un historial de acciones	10
	T 7.3 Realizar un historial de alarmas	10
T.8 Valoración y mejora de la experiencia de usuario	T 8.1 Internacionalizar la aplicación web	15
	T 8.2 Probar varios tipos de usuario	5
	T 8.3 Realizar un mapa con posición de los lugares	10
T.9 Personalización de sensores y alarmas	Personalizar el tipo de cada sensor.	15
	Personalizar las alarmas.	15
<b>TOTAL</b>		<b>300</b>

## CAPÍTULO 2: DESARROLLO

### 2.1 PRIMER SPRINT

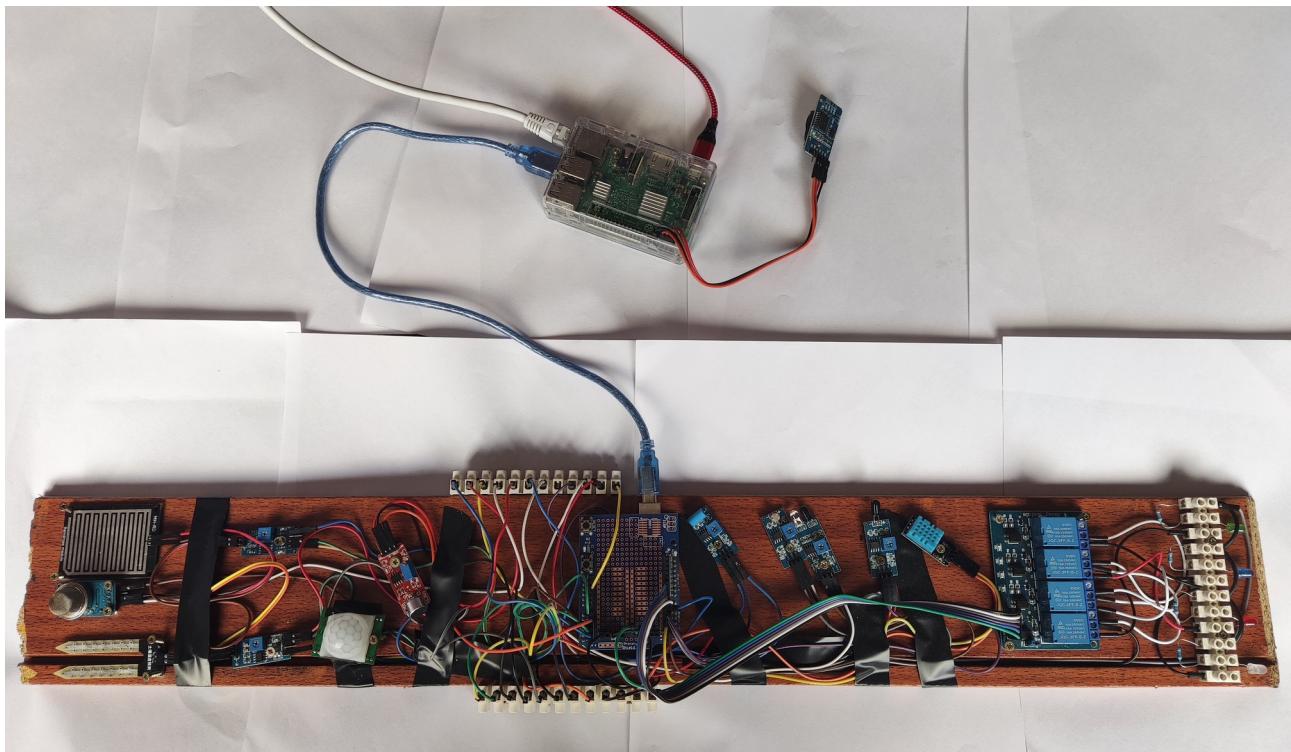
Objetivos:

T 1.1 Montar del cableado entre Arduino y los sensores	10
T 1.2 Realizar las lecturas de datos de cada sensor desde el programa de Arduino	10
T 2.1 Comunicar Arduino y Raspberry pi desde sus respectivos programas. Paso de los datos leídos a la Raspberry pi	10

#### T 1.1 Montaje del cableado entre Arduino y los sensores

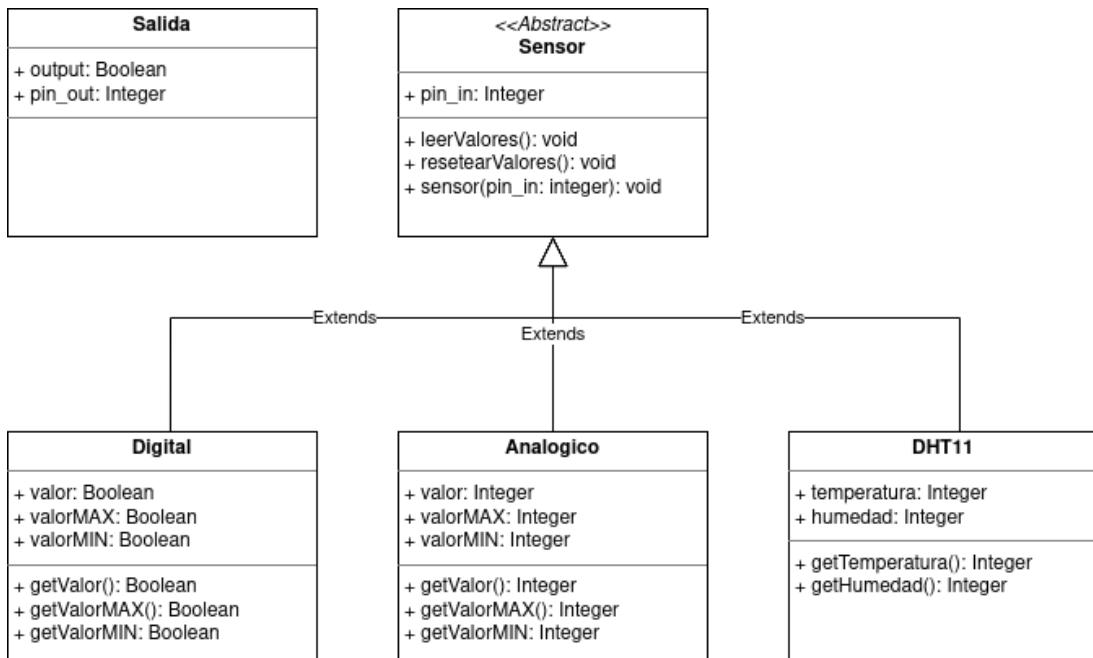
En esta parte del sprint nos encargaremos de conectar los sensores y la unidad de relé al Arduino. Hay que conectar todos los cables a los componentes y empezar a desarrollar el código que generará el programa que residirá en el Arduino. Tenemos 10 sensores que serán las entradas a Arduino y 4 salidas para el relé que serán todas las salidas.

En la imagen se muestra todo el montaje. En la parte de arriba está la Raspberry con el cable de red, cable de alimentación y cable USB conectado al Arduino. También es capaz de acceder a la red mediante WIFI. Abajo está el Arduino conectado a la Raspberry y al resto de componentes mediante cables directos. Las regletas son para conectar las tensiones a cada componente, que se conecta directamente al Arduino, ya que cada componente necesita su tensión de alimentación.



#### T 1.2 Lecturas de datos de cada sensor desde el programa de Arduino:

Lo primero que se ha hecho ha sido modelar conceptualmente los sensores y las salidas:



El diagrama anterior corresponde con las entidades del programa que residirá en el Arduino. Los sensores pueden ser de distintos tipos y serán inicializados con el valor del pin que usarán. Se han definido tres tipos: Digital, Analógico y otro tipo más para el sensor DHT11 que tiene sus propias características. Esto se ha clasificado por los tipos de entradas que puede tener un Arduino. En el supertipo de todos los sensores está el campo del pin que usan y el método para leer los valores actuales del sensor físico y almacenarlos en campos del objeto. Por lo tanto, este método tiene que ser sobreescrito en todas las clases hijas. También está el método resetearValores que reinicia los valores máximos y mínimos de los valores en ese rango de tiempo de medición. Este método no se usará para el tipo de sensor DHT11, ya que no se considerará que sus valores cambien rápidamente a lo largo del tiempo (en este caso tendrá una implementación vacía). Este método se usará en los sensores digitales y analógicos, ya que interesa captar los picos de valores en los rangos de tiempo. Esto será necesario para captar eventos que ocurren en un intervalo de tiempo muy pequeño como el paso de una persona. En este caso no interesa el último valor leído en un determinado rango de tiempo sino el valor más alto o más bajo leído en un determinado rango de tiempo.

Hay que tener en cuenta que se ha tomado una única patilla como conexión a cada sensor y a cada salida porque en los elementos usados ha sido suficiente con esto. Pero en el mercado puede haber elementos que necesiten un número de patillas mayor. En ese caso habría que crear un nuevo tipo de Sensor y sobrescribir su constructor.

**Sensores de tipo Digital:** guardan valores booleanos. Por lo tanto, al reiniciar sus valores de máximo y mínimo, el máximo se pondrá a false y el mínimo a true y serán cambiados cuando tengan un valor diferente.

**Sensores de tipo Analógico:** guardan valores enteros. Estos valores estarán entre los rangos: [0..1023]. Esto es debido a que el Arduino usa 10 bits para almacenar estos valores. Al reiniciar los valores, el valor máximo se pondrá a 0 y el mínimo a 1023. Así estos valores se cambiarán en las próximas lecturas.

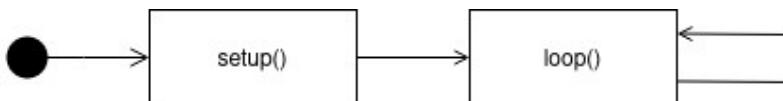
**Sensor de tipo DHT11:** guarda la temperatura y la humedad ambiente. Usa una sola patilla pero almacena dos valores. Para hacer esto se usa una librería que tiene licencia MIT. Hay que tener presente que esta librería al hacer lecturas deja ocupado al Arduino durante unos cuantos milisegundos y, por lo tanto no se puede leer del resto de sensores. Por lo tanto, se leerán los valores de humedad y temperatura al recibir la petición de la Raspberry pi para enviar los datos de los sensores.

Hay que tener en cuenta que un Arduino es un dispositivo muy limitado y tiene una cantidad de memoria muy limitada: solamente dos KiloBytes de memoria RAM para las variables. El tamaño máximo para el almacenamiento del programa es de 32 KB. Por ello, usaremos la mínima cantidad de memoria posible, el mínimo código posible y las mínimas librerías posibles, actualmente una. También es una buena opción usar variables e incluso objetos globales para ahorrar memoria en el tamaño de la pila y no es un problema dada la alta simplicidad y baja extensión del código. El código del programa almacenado en el Arduino tiene menos de 400 líneas y este es el resultado de la compilación:

El Sketch usa 8752 bytes (27%) del espacio de almacenamiento de programa. El máximo es 32256 bytes.  
Las variables Globales usan 393 bytes (19%) de la memoria dinámica, dejando 1655 bytes para las variables locales. El máximo es 2048 bytes.

Por lo tanto, hemos ocupado un buen porcentaje de la capacidad de la memoria, aunque todavía queda sitio para bastante más.

Un programa en C++ en Arduino, a diferencia de otros programas en C++ tiene 2 funciones: `setup()` y `loop()`. Estas funciones no tienen parámetros, otro motivo más para usar variables globales. Al arrancar el programa se ejecuta la función `setup()` y cuando termina se ejecuta la función `loop()` constantemente. Dado el propósito de la placa Arduino, el flujo del programa no termina, solamente termina cuando la placa deja de tener alimentación o es reprogramada:

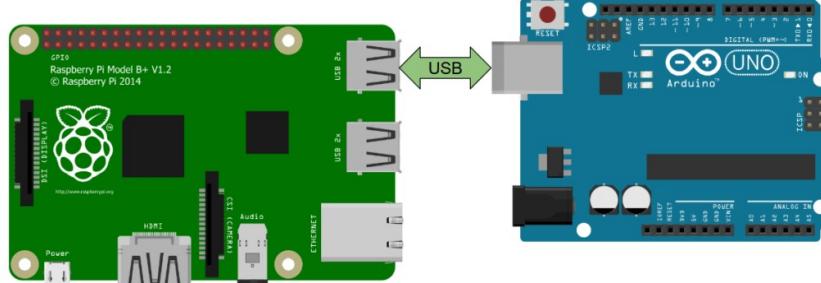


En la función `setup()` se inicializan los objetos ya creados y se ponen las salidas del relé a nivel bajo que es considerado el valor por defecto. En la función `loop()` se comprueba si hay una petición de la Raspberry pi. Si no la hay, se hace una lectura de todos los sensores a excepción del DHT11. Es necesario leer los valores de estos sensores, para detectar los valores máximos y mínimos de las lecturas. En el caso de recibir una petición de lectura de sensores se hacen las lecturas del sensor DHT11 y se envían. En caso de recibir una petición para cambiar los valores de las salidas, se cambian según los valores llegados en la petición y se contesta a la petición para dar constancia de que ha funcionado.

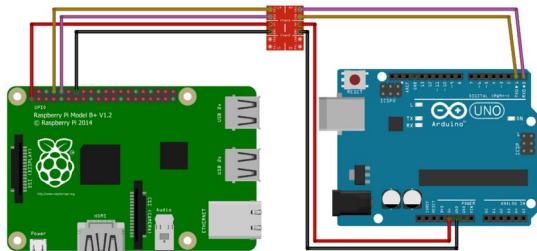
### T 2.1 Comunicar Arduino y Raspberry pi desde sus respectivos programas. Paso de los datos leídos a la Raspberry pi

Una vez leídos los datos de los sensores desde el Arduino llega el momento de traspasarlos. El primer paso será pasar esos datos al componente que tenga conectado directamente. Este será una Raspberry que almacenará esos datos y, a intervalos de tiempo, los enviará a la base de datos central. Hay que tener presente que es necesario llevar un pequeño control de errores, ya que, al arrancar, los primeros mensajes no llegan correctamente. El cifrado y la seguridad no se tratarán, ya que esta comunicación es llevada a cabo entre dos componentes que están uno al lado de otro, por lo que esto no se considera necesario. Esto se cumple para todos los sistemas de comunicación aquí expuestos. Hay distintas opciones para comunicar una Raspberry un Arduino. Los fundamentos son los siguientes:

- **Conexión serie mediante USB:** es la más aconsejada y la usada en el proyecto. Es simple físicamente y también tiene el código más simple. Con este sistema se puede alimentar al Arduino mediante esta conexión USB, pero se añadirá otra alimentación aparte, ya que aportar esta energía es demasiado para una Raspberry y aunque no se han experimentado reseteos ni caídas en la Raspberry sí que se han visto alertas de tensión baja si no se añadía alimentación aparte al Arduino. Además de ser la más simple, este sistema tiene otra ventaja sobre las demás: se puede cambiar rápidamente el sistema Raspberry por otro tipo de microordenador que funcione con Linux. Esto es porque simplemente se usa un conector USB que tiene cualquier ordenador y el código es genérico de Linux. Gracias a esto también se puede probar el programa desde un PC con la única condición de que esté en el SO Linux y que tenga un puerto USB libre. En teoría se podrían conectar varias placas Arduino a una sola Raspberry. Pero esto está fuera del alcance del proyecto y antes de aumentar la complejidad física sería más práctico cambiar esta placa Arduino por otra con más entradas y salidas. El caso de que con la mayor placa Arduino no fuera suficiente, se podría optar por usar varias. En ese caso, la alimentación externa para cada Arduino sería algo totalmente necesario. La velocidad de transferencia que se usará será de 9600 bps, mucho más alta de lo que se necesita. En caso necesario se puede aumentar.

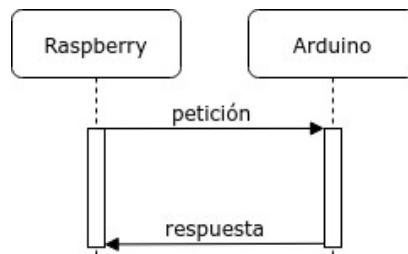


- **Conexión serie mediante GPIOs y el protocolo TX/RX:** en vez de usar un cable USB se usa un protocolo serie desde los propios cableados de la Raspberry y Arduino. Como se puede ver en la imagen, físicamente es más complicado y difícil de mantener. Conecta las entradas de un componente con las salidas de otro. El componente intermedio es un convertidor de niveles de tensión, ya que las patillas de una Raspberry funcionan a un nivel de 3.3 voltios mientras el nivel de funcionamiento de las patillas de un Arduino es de 5 voltios. Otra desventaja de este sistema es que el código de parte del Raspberry usa una librería exclusiva de Raspberry así que si se tuviera que cambiar de microordenador habría que cambiar este código. Otro problema con el que no se ha llegado a probar es que ya estamos usando este protocolo TX/RX con la placa externa que mantiene la hora del Raspberry cuando está apagada, así que tal vez este sistema podría estar directamente descartado.



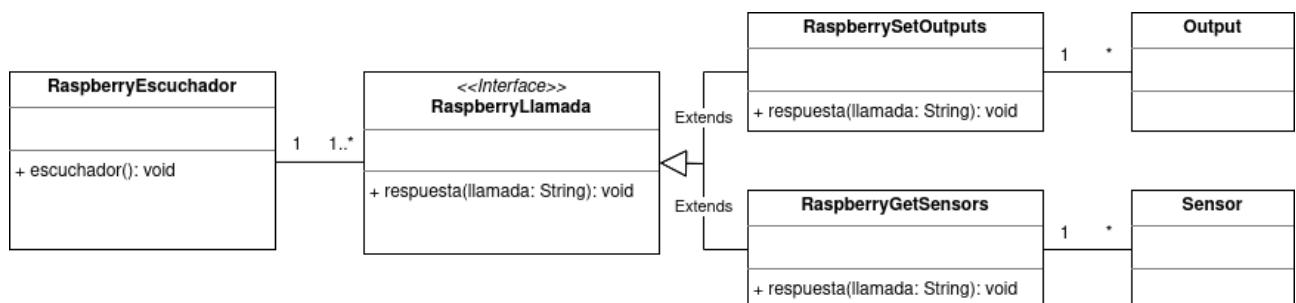
- **Conexión serie realizada completamente desde código mediante las patillas de entrada y salida:** físicamente es similar a la anterior. La diferencia está en que en lugar de usar el protocolo TX/RX se puede implementar el protocolo desde el inicio mediante código en cada uno de los sistemas. Esto lleva varias desventajas como una gran cantidad de trabajo desarrollando este código, fuente de errores y probablemente lentitud. Pero tiene la ventaja potencial de poder usar muchos Arduinos por cada Raspberry, pero esto en comparación no compensa y menos para este proyecto.

Una vez decidido el protocolo y el sistema físico es el turno de pensar en el funcionamiento a nivel de aplicación. Se usará el sistema de petición-respuesta similar al usado por el protocolo HTTP. En la petición se enviarán las salidas que tenga la placa Arduino y se recibirán los valores de los sensores. Al final de todos estos valores se usará un valor de checksum para el control de errores.



Se utilizará una llamada para cambiar una salida que representará un cambio en los relés y otra llamada para obtener los valores de los sensores. La respuesta que recibirá la Raspberry tendrá su propio checksum para comprobar si la respuesta es correcta o no.

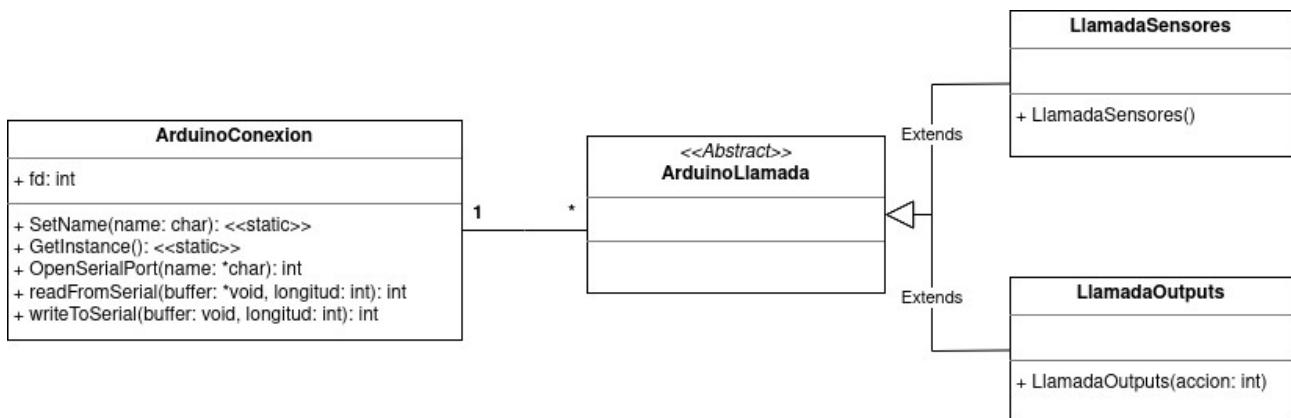
Diagrama de clases realizado para la comunicación en Arduino:



En el programa hay una única clase RaspberryEscuchador. Esta clase tiene un método que es llamado constantemente, llamado escuchador. Este método comprueba si hay algún String que ha llegado por el puerto serie. Si la encuentra, la lee, vaciando el puerto y según su contenido llama a un objeto del supertipo RaspberryLlamada u otro. Actualmente, tendremos dos subtipos de RaspberryLlamada: uno para la llamada relacionada con los outputs, que pone las salidas a los valores que han llegado en la llamada, y, otra, que lee los valores de los sensores y manda una respuesta a la Raspberry pi con sus valores. Por cada llamada, habrá una respuesta para que el programa de la Raspberry compruebe el éxito. Esta respuesta también tendrá su propio checksum para la comprobación de que esté todo correcto. Como se puede ver en el diagrama, la clase RaspberrySetOutputs se relaciona con varias clases Output para cambiar sus valores y la clase RaspberryGetSensors está relacionada con las clases de tipo Sensor para leer sus valores.

La forma en la que se intercambian los datos es mediante un String, que es lo que se pasa entre los dos programas, mediante el puerto serie. Para pasar los datos mediante un String se hace decodificación de los valores numéricos a una lista de caracteres numéricos, y luego se concatenan para meter varios valores en un mismo String. Los números se separan por un espacio, y al final, se pone un último número con la suma de todos los números a modo de checksum para hacer una comprobación de los errores. Al pasar una cadena, se suman todos los números que tiene la cadena separados por espacios menos el último. Finalmente, se compara la suma con el último número y, si son iguales, se acepta la transmisión como correcta y como incorrecta en caso contrario. Los valores booleanos se pasan como 1 y 0 en los casos de verdadero o falso, respectivamente. En la clase RaspberryLlamada el método respuesta es virtual y debe ser implementado por las clases que lo heredan y que pueden ser instanciadas. Por lo tanto, estas dos clases hijas deben implementarlas, ya que cada respuesta es diferente.

Diagrama de objetos en la parte de comunicación con Arduino en la parte de la Raspberry pi:



De esta forma usamos un solo objeto para gestionar la conexión con el Arduino y este objeto será utilizado por todos los subtipos de ArduinoLlamada para hacer su llamada propia al Arduino. Tendremos un tipo de llamada para los sensores y otro para las salidas del Arduino. La clase ArduinoConexión abre el puerto USB que está conectado con el Arduino. Para no inicializar el puerto varias veces se ha optado por el patrón de diseño singleton. De esta forma, solamente se crea una instancia de la clase y solo se inicializa el puerto USB una sola vez. Esta instancia es usada por las instancias de LlamadaSensores y LlamadaOutputs. También hay que tener presente que no se pueden realizar dos llamadas a la vez. Esto se soluciona realizando la llamada y esperando la respuesta cada vez que se llama a un método de llamada.

## ESTABILIDAD DEL SISTEMA RASPBERRY PI

Un aspecto importante es el estado de la Rasberry al cabo del tiempo. Es importante saber si su temperatura y si su tensión ha sido la correcta en todo momento. Esto se puede saber con los siguientes comandos:

```

(pi) 192.168.0.222 — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
pi@raspberrypi:~ $ vcgencmd measure_temp
temp=44.5'C
pi@raspberrypi:~ $ vcgencmd get_throttled
throttled=0x0
pi@raspberrypi:~ $

```

El primer comando retorna la temperatura actual. El resultado es 44.5 C, lo cual es una buena temperatura para una CPU. Esto se ha logrado dejando el procesador ocioso la mayor parte del tiempo.

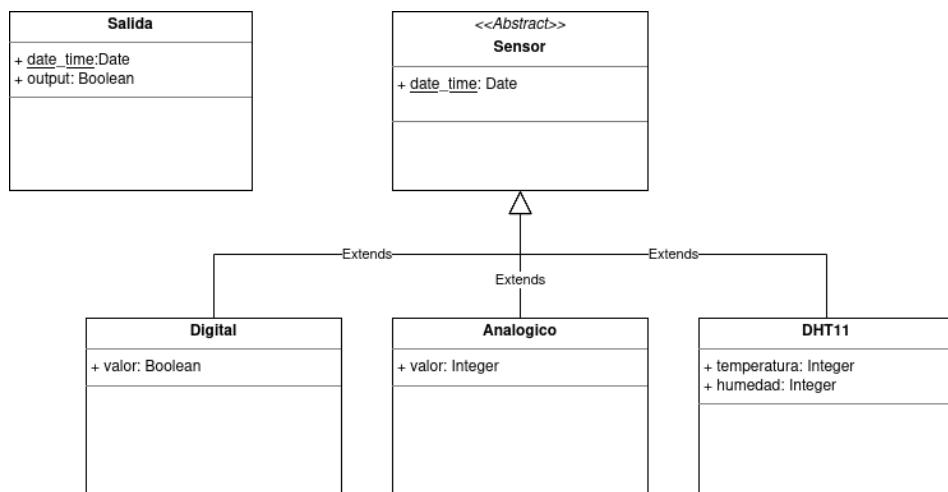
El segundo comando examina la tensión. El valor 0x0 significa que la tensión ha sido estable desde el encendido de la Raspberry pi.

**Revisión del Sprint:** se han realizado todas las tareas con éxito en el tiempo previsto.

## 2.2 SEGUNDO SPRINT

T 2.2 Diseño y creación de la base de datos que residirá en la Raspberry pi	5
T 2.3 Hacer que el programa que se ejecuta en la Raspberry pi persista los datos en la base de datos	5
T 3.1 Diseñar la base de datos para la web y generación la base de datos. Esta base de datos estará contenida en el servidor.	10
T 3.2 Iniciar el backend con los microservicios que admiten los datos por microllamadas y los retornan a petición de otras microllamadas. Habrá backend para la web mediante Node.js que recibirá llamadas HTTP desde la Rasberry. En esta parte se tratará la parte del servidor implementada en JavaScript	10

### T 2.2 Diseño y creación de la base de datos que residirá en la Raspberry pi:



Este es el diagrama que representa el concepto de un sensor y sus diferentes tipos. De cada sensor se guarda su medición y la fecha en la que ha sido tomada. Los valores a almacenar dependen del tipo de sensor. Para almacenarlos físicamente, lo requerido por las formas normales es usar una tabla por cada tipo. El modelo representado en la imagen cumpliría las reglas básicas de normalización de tablas en las bases de datos. Pero esto ocuparía mucho espacio en el caso de grabar un gran número de filas por unidad de tiempo y un gran número de datos al replicar el valor de la fecha. Por lo tanto, se ha elegido usar una estructura que no cumple las formas normales, pero que ocupará mucho menos espacio.

Con el objetivo de ahorrar espacio se ha diseñado el siguiente modelo de datos:

Actions	Sensors
+ date_time: Date + binary_values: int	+ date_time: Date + has_sended: Boolean + binary_values: Integer + has_persons: Integer + has_sound: Integer + has_gas: Integer + has_rain: Integer + temperature: Integer + humidity: Integer

En la primera tabla solamente se tienen las acciones que serán las salidas del Arduino. La clave primaria es el tiempo, de forma que, para obtener los valores que hay que enviar al Arduino, se extrae la fila con la fecha más alta, la cual se extrae rápidamente porque es el campo clave.

Por último, la segunda entidad es donde se guardan los valores de los sensores. Hay que tener presente que esta tabla puede llegar a ocupar una gran cantidad de memoria en el disco, por lo que reducir su longitud, se ha usado un solo campo para los sensores digitales. Incluso se podrían haber compartido los valores de los sensores analógicos, pero no se ha hecho por simplicidad. El campo clave será la fecha y existirá un campo “has\_sended” de tipo booleano cuyo valor será false si la en su última medición no ha sido recibida por la base de datos central, y, true en el caso de que sí haya sido recibida. Por lo tanto, inicialmente su valor será false.

La siguiente imagen muestra un ejemplo de los datos que se guardan en la base de datos:

	date_time	has_sended	binary_values	has_persons	has_sound	has_gas	has_oil	has_rain	temperature
1	2022-11-13 19:47:09.719	[ ]	10	1.023	333	84	996	995	1.023
2	2022-11-13 19:47:07.710	[ ]	10	1.023	333	84	994	994	1.022
3	2022-11-13 19:47:05.702	[ ]	10	990	333	84	994	994	1.021
4	2022-11-13 19:47:03.694	[ ]	10	990	333	84	994	994	1.021
5	2022-11-13 19:47:01.686	[ ]	10	1.023	333	84	995	995	1.021
6	2022-11-13 19:46:59.673	[ ]	10	1.011	333	84	995	995	1.021
7	2022-11-13 19:46:57.663	[ 1 ]	10	1.023	333	84	995	995	1.021

Como se puede ver se están grabando los datos cada dos segundos. Y ninguna de estas filas se ha enviado al servidor central.

### T 2.3 Hacer que el programa que se ejecuta en la Raspberry Pi persista los datos en la base de datos ubicada en la Rasberry:

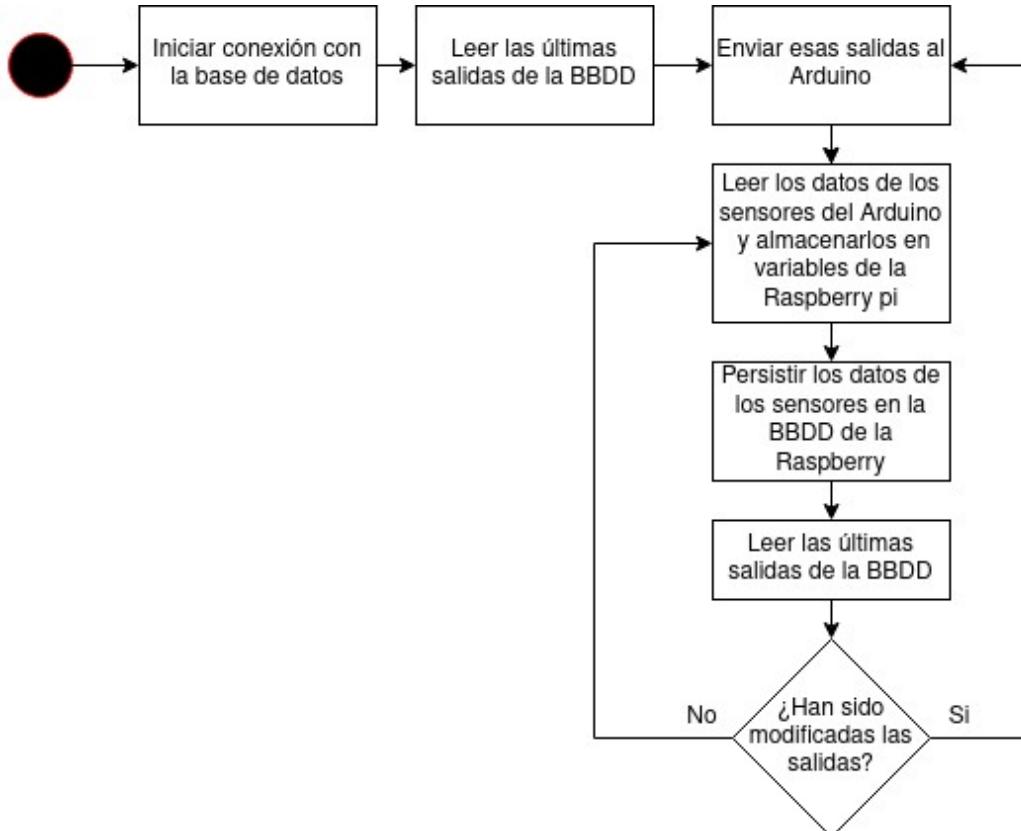
Para almacenar datos en la Raspberry Pi, y también en el servidor, usaremos el SGBD llamado PostgreSQL. Es un sistema de gestión de base de datos relacional de código abierto bajo la licencia PostgreSQL, similar a BSD o la MIT. Por lo tanto, se puede usar libremente para desarrollos comerciales, por lo que es muy usado.

Para interactuar con la base de datos en PostgreSQL con la Raspberry Pi usaremos la librería para lenguaje C libpq. Esta librería tiene una licencia similar a la BSD, por lo que puede ser distribuida y se puede hacer uso comercial de ella. Para abstraer esta capa de persistencia se ha encapsulado en una clase llamada Bdconexión:

DBconexion
- connInfo: String - conexion: *PGconn
+ BDconexion(String) + insertarMedicion(valoresBinarios int, personas int, sonido int, gas int, aceite int, temperatura int, humedad int): int + getUltimaAccion(): int

Esta clase será ampliada posteriormente para la interacción entre las bases de datos de la Raspberry y el servidor central. De momento tenemos guardado la conexión a la base de datos, su configuración y algunas acciones para interactuar con la base de datos.

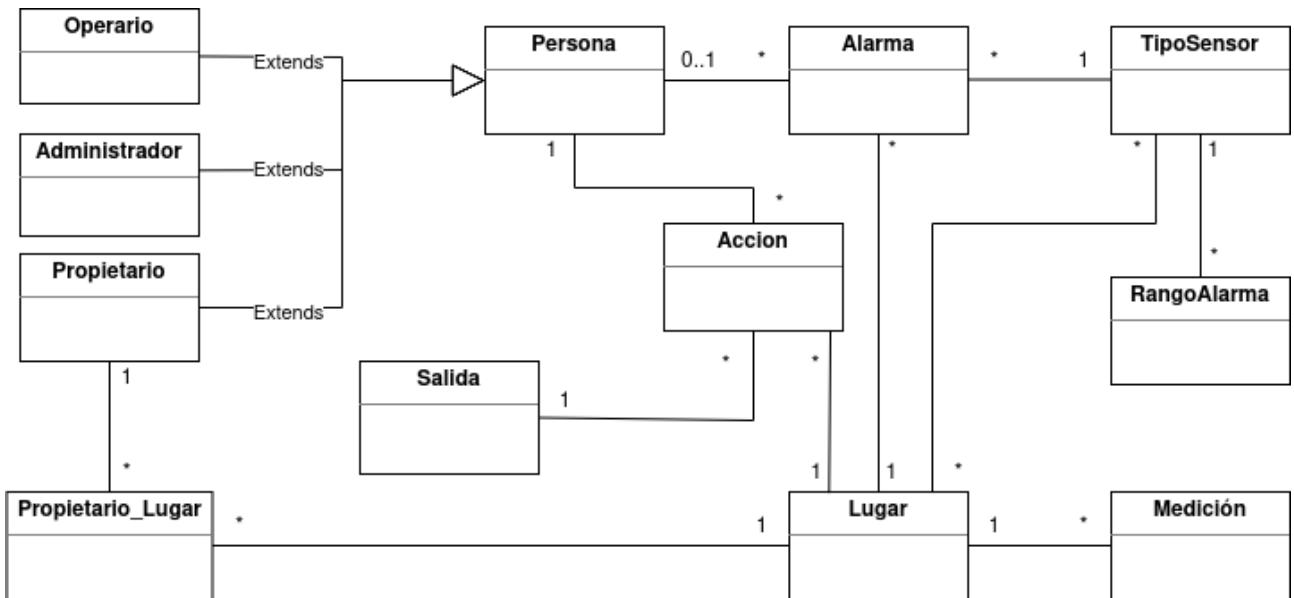
El flujo del programa:



Como el programa no tiene final, no hay punto final.

**T 3.1 Diseño de la base de datos para la web y generación la base de datos.** Esta base de datos estará contenida en el servidor web.

Se ha ideado el siguiente esquema de entidades:



Son unas cuantas tablas con unas relaciones bastante complejas que exigen una severa explicación.

**Tabla Medicion:** es equivalente a la tabla Sensor ubicada en la Raspberry Pi. Hay que recordar que esta tabla no está normalizada como ya se comentó antes. Esta tabla contiene las mediciones en un determinado momento. Una medición es de un determinado lugar y puede generar varias alarmas o ninguna según sus mediciones. Una alarma es creada cuando se detecta fuego o gas en los sensores, aunque está la posibilidad de añadir la funcionalidad para que se pueda configurar.

**Tabla Lugar:** representa un lugar donde hay un sistema con una Raspberry Pi, un Arduino y los correspondientes sensores. Esta tabla tiene su propio identificador único y muchas relaciones porque es una de las tablas más importantes de toda la base de datos.

**Tabla Alarma:** se refiere a un evento generado por una medición en un sensor. Como la tabla Mediciones no está normalizada y la medición de un solo sensor puede generar una alarma, se llega a la conclusión de que una sola fila de las mediciones puede generar varias alarmas. Para tener presente el sensor que ha generado cada alarma, se ha generado una relación entre cada alarma y un tipo de sensor en concreto. Una alarma puede ser tratada por una persona o por ninguna, por lo tanto, también se ha creado una relación para esto. Esta relación puede no existir, ya que en el caso de que se genere la alarma ninguna persona la habrá revisado todavía. Existe la posibilidad de relacionar una alarma con una medición en vez de con un lugar, pero la información es prácticamente la misma y el rendimiento será mucho mejor con la relación a la tabla lugar por tener muchas menos filas.

**Tabla TipoSensor:** representa un modelo de sensor. De momento tendremos 10 sensores predefinidos, aunque está la posibilidad de poder añadir más en el futuro. Dado que en cada lugar, puede haber varios modelos de sensores y cada modelo de sensor puede estar en varios lugares hay una relación varios a varios con la tabla Lugar. Dado que la tabla Medición no está normalizada, si queremos incluir más modelos de sensores será necesario incluir más campos en esta tabla. Esto puede ser un problema, pero es uno de los precios a pagar por el rendimiento. Actualmente, la implementación da por hecho que todos los tipos de sensores están en todos los lugares.

**Tabla RangoAlarmas:** en esta tabla reside la información de los rangos, por cada tipo de sensor, en los cuales se disparan las alarmas. Actualmente, se usan valores predefinidos, pero, ante la posibilidad de que se puedan configurar en un futuro.

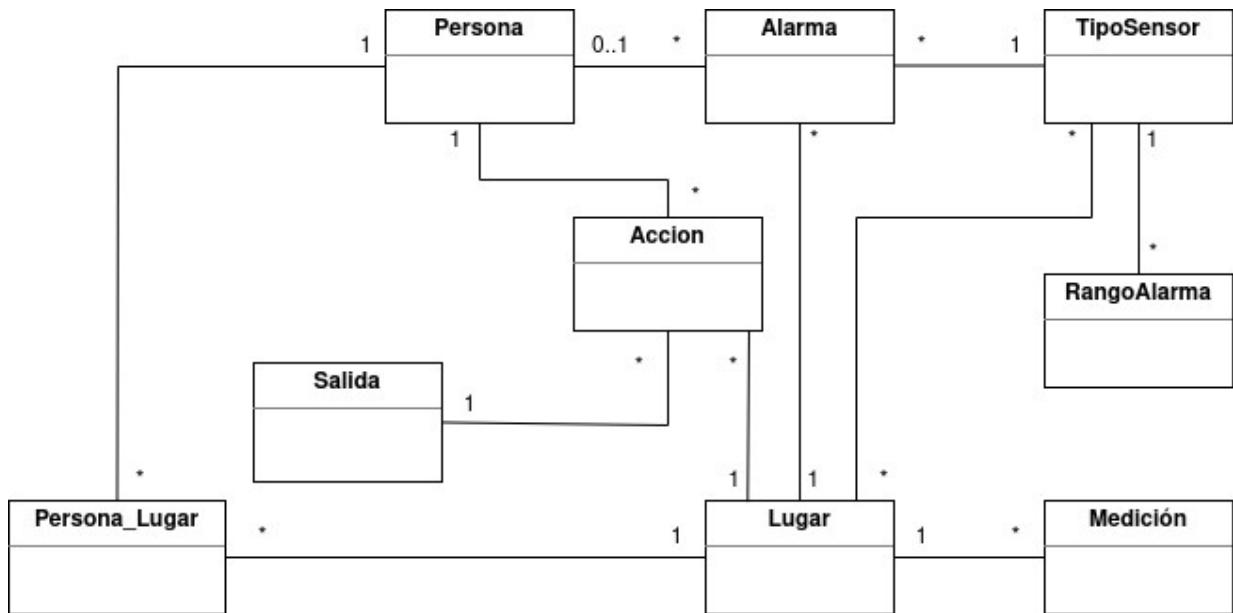
**Tabla Salidas:** se refiere a las salidas que hay en el Arduino. En el alcance del proyecto se han predefinido 4 salidas, pero se ha generado esta tabla para obligar a la base de datos a tener consistencia y para tener la posibilidad de futuras ampliaciones. Si hay ampliaciones, podría ser necesario relacionar esta tabla con la tabla Lugar para saber qué salidas tiene cada lugar, como en el caso de los modelos de sensores.

**Tabla Acciones:** representa un cambio en una determinada salida de un determinado lugar realizado por una determinada persona. Esta relación entre tablas está normalizada. Se podría haber simplificado haciendo una desnormalización como se ha hecho con la tabla Medición, pero se ha decidido dejarla así ya que, como tendremos muchísimas menos acciones que mediciones, la necesidad de

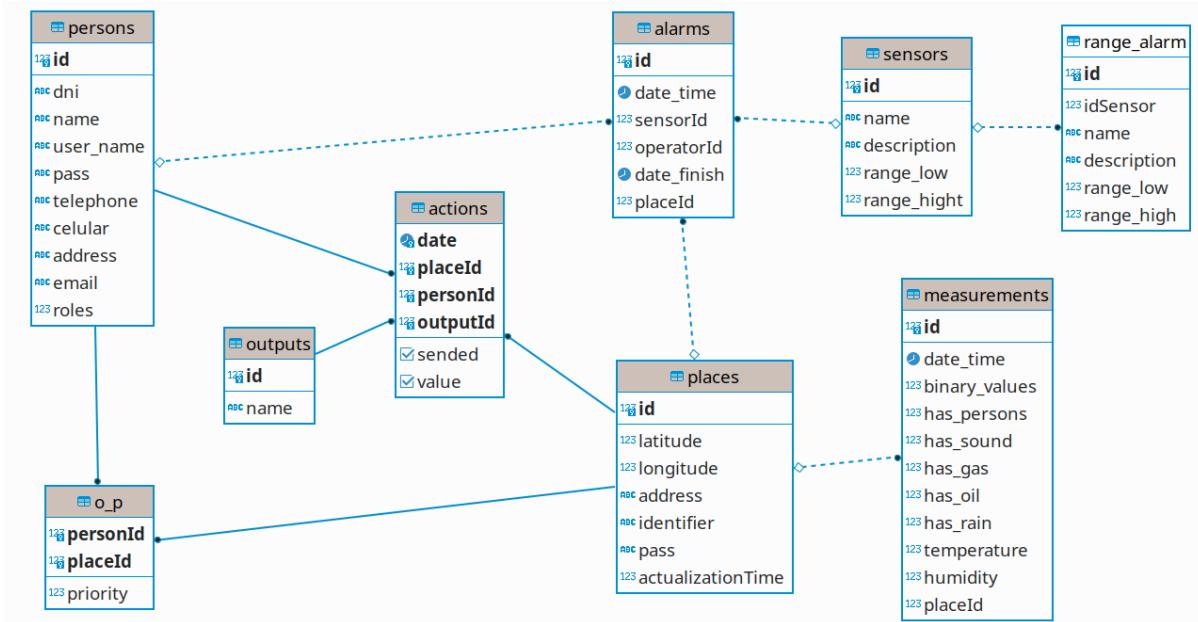
rendimiento en este caso no es tan importante. El contenido de esta tabla puede visualizarse por el usuario para que tenga información de cuáles han sido las acciones a lo largo del tiempo.

**Tabla Persona y sus subtablas:** representa a un usuario del sistema. Una persona puede ser un cliente, un trabajador o un administrador de la organización que mantiene el sistema. Estos roles no serán excluyentes, por lo tanto, una misma persona puede ser tanto cliente como trabajador. La diferencia está en que un cliente solo puede acceder a los lugares que tiene asignados, mientras un trabajador puede acceder a todos los lugares. Además, un administrador puede administrar los usuarios que hay en el sistema. Un operador tendrá todos los permisos de un cliente más los suyos propios y un administrador tendrá todos los permisos que tiene un operario más los suyos propios. Por lo tanto, es suficiente para representar esta característica, el valor de un campo en la fila de usuario. Este campo será el rol del usuario. Una persona tendrá el rol más alto que pueda tener en este campo en la fila que lo representa. No es necesario hacer más tablas. La tabla intermedia Persona\_Lugar ha sido nombrada o\_p en la implementación final, el motivo de este nombre ha sido la traducción al inglés de operation\_place. Es una tabla que implementa la relación N:N que relaciona los lugares con las personas. Una persona en caso de ser un propietario, puede estar relacionado con varios lugares, y, un lugar, puede estar relacionado con varios propietarios.

Por lo tanto el modelo final de la base de datos en el servidor central queda con el siguiente diseño:



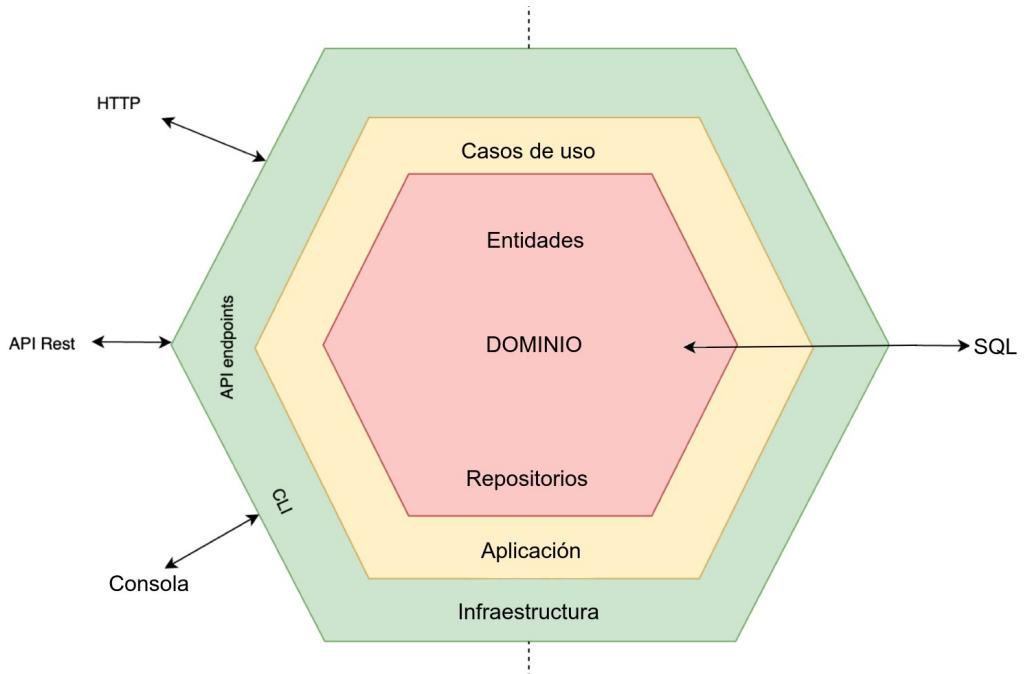
Esta es una representación del modelo de datos con los metadatos generados en el SGBD:



Para representar la clave primaria de la mayoría de las tablas se ha optado por un identificador autoincremental como clave primaria. Esto es debido a que es la forma indicada por el funcionamiento del ORM para lenguaje JavaScript llamado Sequelize, que es el usado en el acceso a la base de datos en este proyecto. Para compensar esto, se han generado índices en la base de datos en los campos necesarios, como en la mayoría de los campos de fecha.

**T 3.2 Inicio del backend:** con los microservicios que admiten los datos por microlllamadas y los retornan a petición de otras micro llamadas. El backend para la web usará la tecnología Node.js que recibirá llamadas HTTP desde la Raspberry y desde el frontend. En esta parte del sprint, se tratará la parte de las llamadas que llegarán desde la Raspberry Pi. En la siguiente parte del sprint se tratarán las llamadas que llegan desde el frontend de la página web.

Como ya se ha comentado al inicio, para el servidor se ha escogido la tecnología Node.js y Express como librería para recibir las llamadas. Para la implementación, en este proyecto se usa una estructura de capas basada en la arquitectura hexagonal. Esta arquitectura está dividida en las siguientes capas:



De esta forma se usa una clasificación en tres capas aisladas. El aislamiento implica que cada capa solo puede interactuar con las capas anterior y siguiente. También las capas son referenciadas en cascada en la aplicación. La capa de las entidades hace referencia a la estructura de los datos y su persistencia. La capa de los casos de uso se refiere a la capa que contiene la lógica de negocio. Y esta capa no interactúa directamente con la librería de persistencia (en nuestro caso Sequelize) ni con la comunicación con el exterior (en nuestro caso Express). La capa controlador o infraestructura se comunica con las interfaces que se comunican con el exterior y envía sus entradas y salidas a la capa de servicio.

La intención de esta arquitectura tiene en común con la estructura de capas, que las capas interiores no sepan nada de las capas exteriores. Diciéndolo de otra forma: las capas interiores no son dependientes de las capas exteriores. De esta forma, si el cliente cambia los requisitos del proyecto y el sistema se tiene que comunicar con el exterior de forma diferente, solamente habrá que cambiar la última capa. Esto es debido a que solamente la capa exterior tiene comunicación con el mundo exterior. También la capa interior es la única que tiene acceso a la base de datos, por lo que si queremos cambiar de SGBD o de ORM solamente tendremos que cambiar la capa interior, ya que el resto de capas no usa estas tecnologías. La peculiaridad de esta arquitectura es que está estructurada a los casos de uso de cada entidad, de forma que todas las entradas y salidas de una entidad están repartidos en tres archivos, uno por cada capa. De esta forma, cada entidad dentro de cada capa está separada del resto, aunque cada una de estas partes puede usar otras entidades, pero siempre desde su propia capa para no romper la estructura de capas. Este es el añadido de la estructura de capas.

Este tipo de arquitecturas tienen las ventajas:

- **Independencia de los Frameworks:** unas librerías no se acoplan con las otras, ya que cada capa usa su conjunto de Frameworks. Esta condición no está en otras tecnologías de capas.
- **Testable:** cada capa es fácilmente de probar, ya que, cada capa solamente interactúa con las capas anteriores y siguientes. También, se puede comprobar el resultado final comprobando la capa exterior. También se puede comprobar cada caso de uso por separado.
- **Independiente de la base de datos:** no hay más dependencias de las necesarias con la base de datos.
- **Buen mantenimiento para proyectos de vida media y larga:** al estar, los componentes separados son más fáciles de mantener y modificar.

Sin embargo, también se encuentran algunas desventajas, como la complejidad que se crea cuando existen lógicas complejas. En algunas ocasiones una entidad necesita relacionarse mucho con otras entidades y en esos casos esta estructura puede crear complejidades innecesarias.

Para este diseño, cada tabla de la base de datos tiene varios archivos que manipulan desde dominio de la tabla hasta los microservicios, pasando por la lógica:

**<nombre>-model.js:** es el archivo donde está definida la tabla estructurada por el servicio. Contiene una estructura para el ORM llamado Sequelize que genera la tabla en la base de datos y sus relaciones. Con esta estructura se pueden hacer los accesos a la tabla en la base de datos. Cada uno de estos archivos tiene la información para generar una tabla en la base de datos que se genera si es que no existe al ejecutar el programa.

**<nombre>-repository.js:** si todo va bien, pasa los datos obtenidos de la base de datos y de los parámetros de entrada a la capa de servicio. Este archivo es dependiente de la parte de persistencia y, también, accede a la librería Sequelize. Este archivo y el anterior forman la capa central, la cual es la encargada de la parte de persistencia. Por eso es la única capa que utiliza el ORM llamado Sequelize. Hereda de BaseRepository. Aquí se heredan todos los métodos, por lo que no es necesario hacer cambios de la clase base. Solamente cuando la naturaleza de la entidad lo requiere. Solamente responde a una llamada HTTP si ocurre un error en la capa de persistencia. En esta parte no debe hacer ninguna lógica de negocio ya que tiene que estar totalmente encargada de manejar los datos de la base de datos mediante el ORM. Un cambio del SGBD o del ORM solamente debe afectar a esta parte

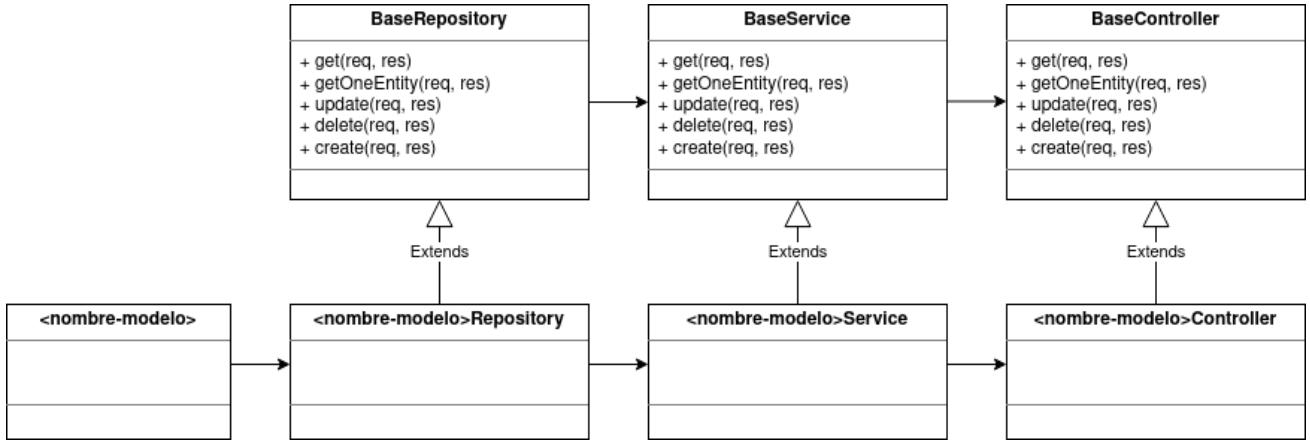
**<nombre>-service.js:** hereda de BaseService. Es donde reside toda la lógica de negocio de la entidad. Si no se sobreescribe ningún método de la clase padre, entonces no se usa ningún tipo de lógica especial. Por lo tanto, el comportamiento por defecto es llamar a la capa siguiente. Este es el archivo que equivale a la capa llamada casos de uso. Depende de las entidades y se ocupa de lógicas en concreto de la aplicación.

**<nombre>-controlador.js:** usa los interfaces externos para interactuar con el exterior. En este caso, usa la librería Express de Node.js para recibir y responder a las llamadas HTTP. El contenido mínimo de este archivo relaciona las rutas URL con los métodos que las gestionan. Por lo tanto, llama a los métodos de Express para hacerlo.

Dada la naturaleza de JavaScript es posible dejar las clases hijas prácticamente vacías. Esto es porque en este lenguaje no hay tipos y en una variable se puede almacenar cualquier dato.

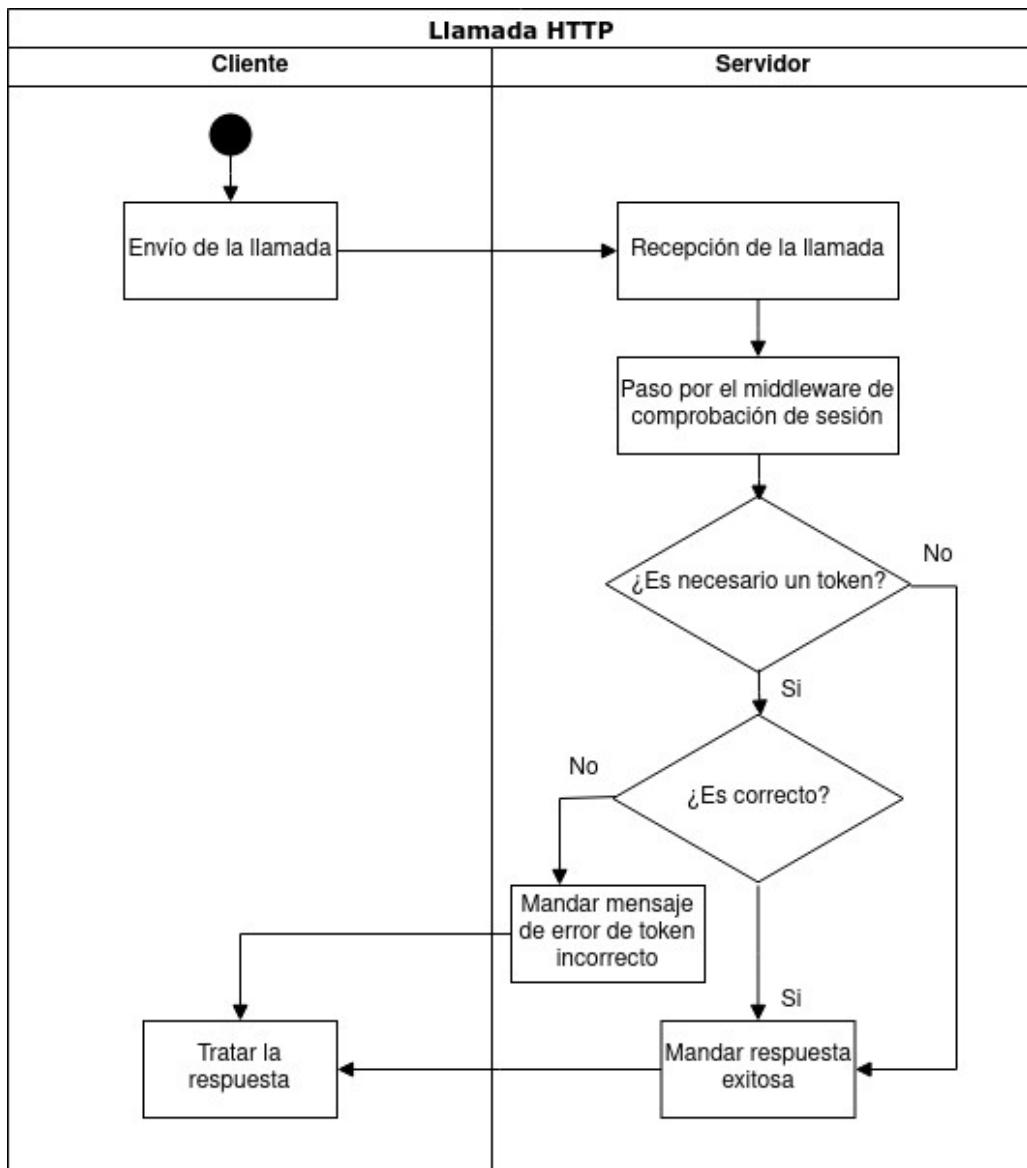
La estructura de las clases está basada en una cascada: cada objeto recibe una instancia del objeto anterior en su creación. De esta forma llama a los métodos de la instancia que ha recibido en su creación.

Este es un diagrama de la estructura de clases de cada entidad:



De esta forma, cada capa de la entidad hereda de una clase predefinida, consiguiendo que no sea necesario repetir código en cada entidad.

Como estructura de los middlewares para la gestión de los permisos se usa la estructura de los middlewares típica de los servidores web. Cada petición HTTP pasa por middlewares hasta llegar a los objetos anteriores. En cada uno de estos middlewares la petición puede ser respondida. Esto se usa para realizar el control de las sesiones de forma automática. Para realizar la gestión de sesiones se usa un middleware que recibe un identificador de sesión llamado token y, si este token no existe o no es correcto, se responde a la petición con un error. Esto se puede deshabilitar en las variables de entorno para facilitar el desarrollo. También se puede deshabilitar en algunos servicios para que no necesiten autenticación para ser usados, como el servicio de autenticación. Estos middlewares se encuentran en la capa de los controladores en la parte externa. De forma que es un lugar por donde pasan las llamadas HTTP antes de llegar al resto de la capa.



Este es el ciclo que sigue una llamada. Esto se realiza de forma automática, por lo que no es necesario comprobar la identidad del emisor de la llamada en cada implementación de un servicio, ya que se hace automáticamente en el middleware por el que pasan todas las llamadas.

En uno de los anexos hay una lista con los endpoints a los que se pueden hacer llamadas. Algunos de estos endpoints son para ser llamados por el backend y otros por la Raspberry Pi. Cada uno de estos tipos tiene un sistema de seguridad diferente, pero similar. Siempre se usa un token, pero la forma de obtener la clave y la contraseña son diferentes.

**Revisión del Sprint:** se han realizado todas las tareas con éxito en el tiempo previsto. La última tarea solo estaba planificado hacer los servicios para que se realizasen las llamadas desde la Raspberry Pi. Por lo que se continúa con esta tarea en el siguiente sprint realizando los endpoints para ser usados por el frontend. Todos los endpoints han sido documentados en uno de los anexos.

## 2.3 TERCER SPRINT

T 3.2 Iniciar del backend con los microservicios que admiten los datos por micro llamadas y los retornan a petición de otras micro llamadas. Habrá backend para la web mediante Node.js que recibirá llamadas HTTP desde el Arduino. Se realiza la parte que hace las llamadas desde la Raspberry que quedaba pendiente. En esta ocasión hacemos la parte que hace las llamadas desde la Raspberry	10
T 3.3 Implementar de la creación de tablas visibles en la página. Estas se podrán rehusar rápidamente en Angular	20

**T 3.2 Iniciar del backend con los microservicios que admiten los datos por micro llamadas y los retornan a petición de otras micro llamadas.**

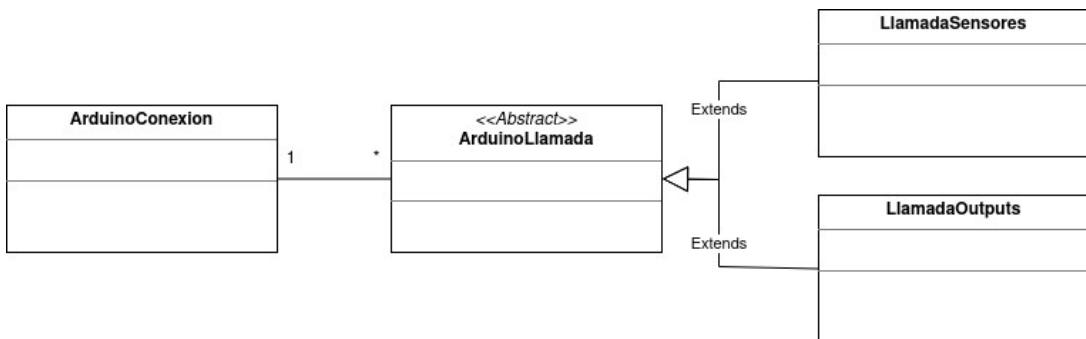
Se han realizado los endPoints que realizan todas las operaciones CRUD de todas las tablas en el servidor. Dado el diseño de objetos en el backend, realizado en JavaScript, se pueden realizar las modificaciones de una tabla con muy poca cantidad de código, ya que solamente hay que heredar las clases anteriormente descritas para tener toda la funcionalidad que realiza un CRUD.

El planteamiento del sistema es que el servidor tenga constantemente las mismas URLs y reciba peticiones HTTP desde la Raspberry Pi. De esta forma tendremos tres tipos de llamadas:

- Llamadas que envían mediciones al servidor.
- Llamadas que piden los últimos estados de los outputs para el Arduino.
- Llamadas que piden la velocidad de actualización de las mediciones hechas por la Raspberry Pi al servidor.

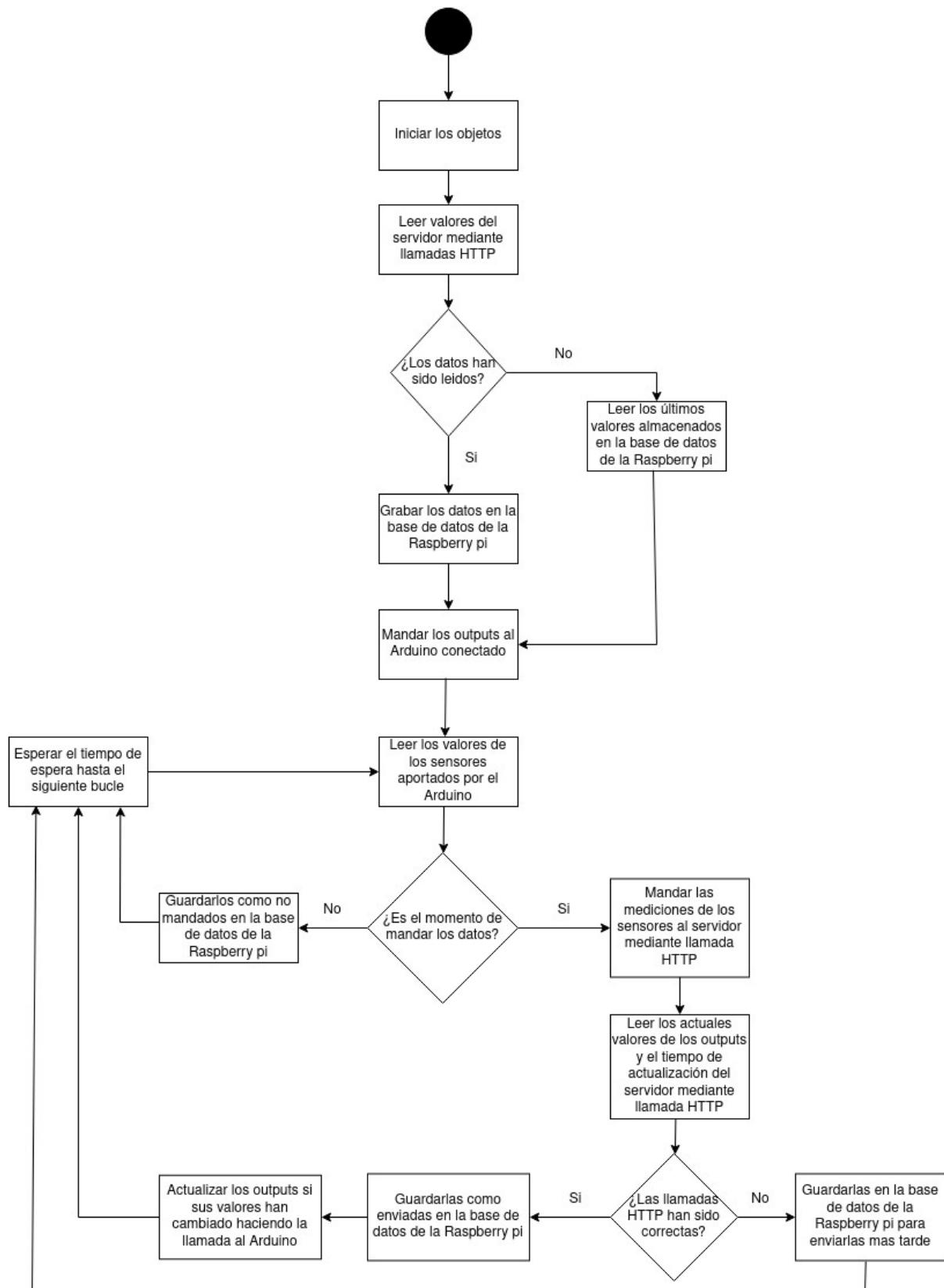
Por lo tanto, este programa que se ejecuta en la Raspberry gestionará llamadas HTTP al servidor, llamadas al Arduino y la gestión de la base de datos. Por ello, hay que decidir el diseño de cada tarea y decidir como harán interacciones entre sí.

Este es el esquema simplificado que hemos visto antes de clases de la interacción con el Arduino:



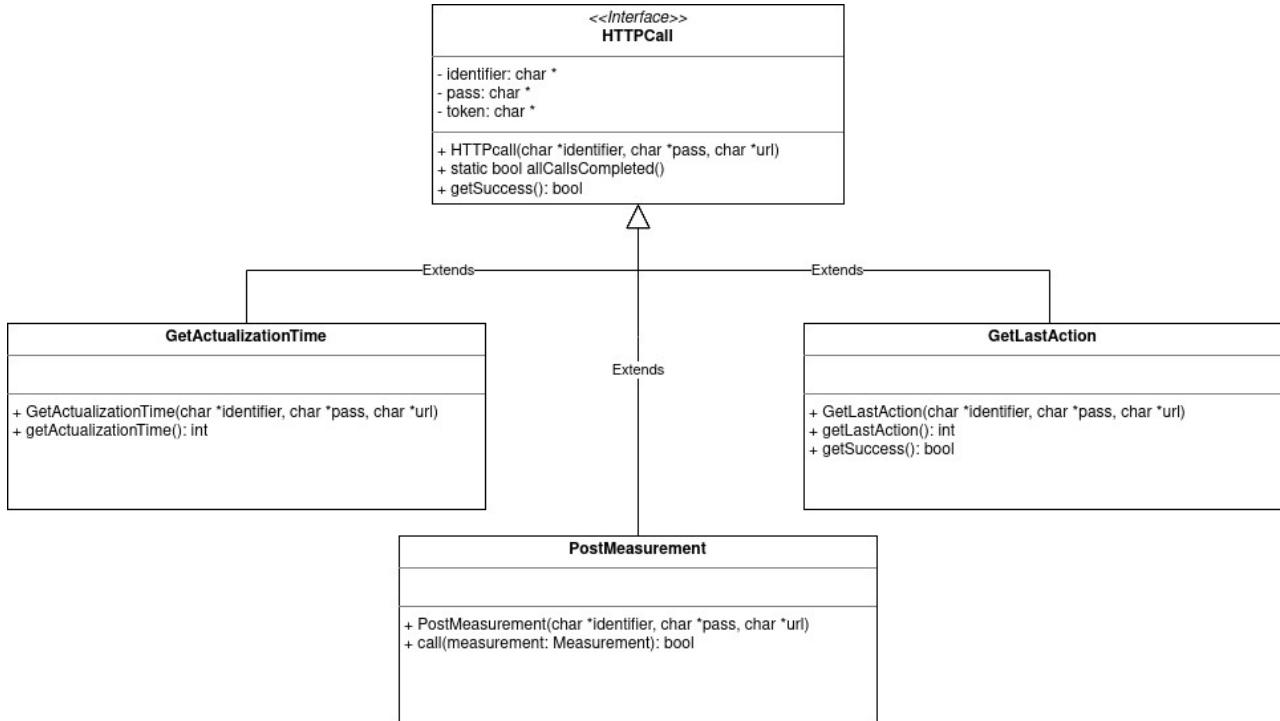
De esta forma tendremos solamente una sola instancia de **ArduinoConnexion**. Esta inicializa el puerto USB que conecta al Arduino y envía y recibe los datos. La siguiente es la interfaz **ArduinoLlamada**, en una inicialización se le pasa la instancia de **ArduinoConexion** para que puedan acceder al puerto serie. De este derivan las otras dos clases. Una para hacer llamadas a los sensores y otra a los outputs. En los outputs la entrada es un entero en el que se encuentran los valores de cada salida en binario y, en el caso de los sensores, se le pasa una variable que contiene los segundos que se puede esperar a recibir los datos. Esto es usado para poder llevar el tiempo de actualización lo más bajo posible. Se ha conseguido un tiempo de lectura de dos segundos entre lecturas de sensores.

Para poner todo esto junto hay que tener muy claro la secuencia del programa.



En el caso del acceso a la base de datos, se ha optado por una única clase donde se han implementado todas las acciones que se realizarán sobre la base de datos.

En el caso de las llamadas HTTP al servidor se ha realizado una interfaz `HTTPCall` para encapsular lo más posible la librería utilizada y se ha derivado una clase por cada tipo de llamada realizada.

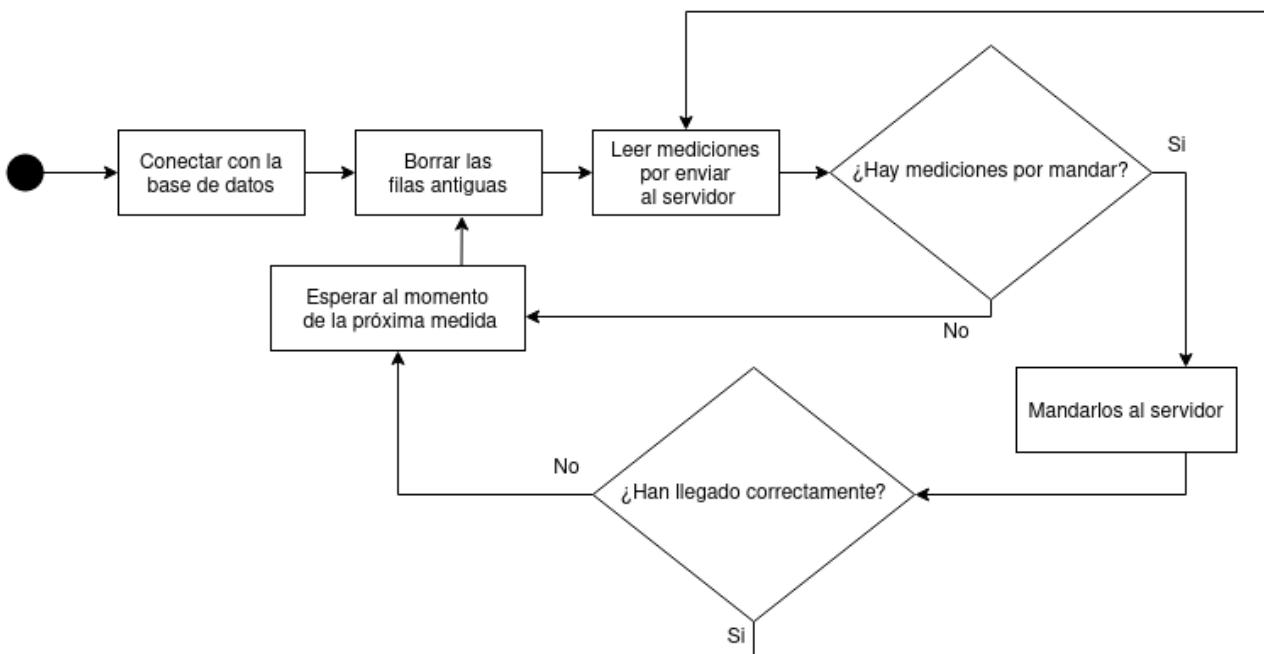


Son muchos componentes juntos interactuando entre sí. Al iniciar el programa se inicializan todos los objetos, incluyendo la conexión con la base de datos. Lo siguiente es leer los datos necesarios del servidor mediante llamadas HTTP. Estos datos son los últimos valores de los outputs y el tiempo de espera en las actualizaciones de los datos que se pueden configurar en el servidor por cada lugar. Si la llamada resulta fallida, se asigna a los outputs los últimos valores almacenados en la Raspberry Pi. Después comienza el escuchador. En este escuchador se van leyendo los datos de los sensores que se almacenan en la base de datos. Cuando es el momento indicado, estos valores se mandan al servidor y se consultan los últimos valores de los outputs para mandarlos a la Raspberry desde el servidor. En el caso de que los valores no hayan podido ser enviados, se guardan para mandarlos posteriormente. El encargado de esto es otro programa que se encarga del mantenimiento de los datos y que se ejecuta en la Raspberry al mismo tiempo que el programa que escanea los datos. Se ha optado por esta opción para simplificar el programa. Estos dos programas compiten únicamente en el acceso a la base de datos, pero esto no será un problema, ya que cada programa accederá a la base de datos cada varios segundos y el propio SGBD se encargará de los posibles problemas de concurrencia, bloqueando las tablas en los momentos necesarios. Dada la simpleza no ocurrirán bloqueos mutuos.

El programa de mantenimiento se encarga, de mandar los datos de los sensores que no han sido enviados por fallos de las llamadas HTTP. Esto puede ser algo común porque puede haber determinados lugares que se queden sin cobertura y no puedan acceder a la red. También se encarga de eliminar las filas de mediciones que han sido tomadas hace tiempo. Esto es para que la base de datos que reside en la Raspberry pi no se haga demasiado grande con el tiempo. Si la base de datos se hace demasiado grande, el rendimiento del sistema podría bajar demasiado y el almacenamiento podría llenarse.

Estos dos programas se ejecutan en el arranque de la Raspberry Pi. Su sistema operativo ya ha sido configurado para ejecutarlos en su arranque. De esta forma, al conectar la alimentación a la Raspberry Pi, está arranca su sistema operativo y luego estos dos programas. Por lo tanto, no hace falta ninguna responsabilidad por parte del usuario en su ejecución.

Este es el flujo del programa que se encarga del mantenimiento de los datos de las mediciones. Las mediciones que no se han podido mandar se buscan en la base de datos y se intentan mandar al servidor, hasta que se consiguen mandar o hasta que no queda ninguna fila por mandar. Las mediciones que se mandan en cada llamada HTTP son las que tienen una fecha más alta, para que se vayan mandando por prioridad de fecha. Si no hay filas por mandar o no se consigue conectar con el servidor, se espera un tiempo prudencial hasta el siguiente intento. Así se evitan procesos innecesarios.



**Revisión del sprint:** esta tarea se ha retrasado. En el caso de realizar las llamadas HTTP en C se ha tardado el triple de tiempo, ya que ha sido muy costoso porque ha habido que utilizar librerías no utilizadas antes: para manejar acceso a bases de datos SQL, hacer llamadas HTTP y leer las estructuras JSON utilizadas en las tablas. Como el tiempo en completar esta tarea ha sido de 30 horas, se considera como un sprint completo. Así quedará el tiempo realizado en el cálculo final:

T 3.2 Inicio del backend con los microservicios que admiten los datos por micro llamadas y los retornan a petición de otras micro llamadas. Habrá backend para la web mediante Node.js que recibirá llamadas HTTP desde el Arduino. Se realiza la parte que hace las llamadas desde la Raspberry que quedaba pendiente.	30
---	----

## 2.4 CUARTO SPRINT

### T 3.3 Implementación de la creación de tablas visibles en la página. Estas se podrán rehusar rápidamente en Angular

La estructuración de Angular está basada en módulos y componentes. Una aplicación web en Angular tiene un módulo raíz. Un módulo puede contener otros módulos y componentes. Un componente tiene un sistema de rutas o enlaces URL que definen qué componente se ocupa de cada URL disponible en la página. Cada componente contiene un registro de los módulos y componentes con los que está relacionado, un código en TypeScript que se ocupa de la lógica de negocio, un archivo HTML y otro SCSS (el cual genera código CSS para los estilos en cascada) que se ocupan de la vista. Un componente solamente tiene los archivos de código, HTML y SCSS. Un componente se puede insertar en cualquier módulo del proyecto, poniendo la etiqueta HTTP que lo representa. Una de las ventajas de este sistema es que se pueden pasar parámetros a los componentes, que se llaman mediante etiquetas HTML incluyéndoles parámetros como atributos. Los atributos pueden ser de cualquier tipo, ya que en TypeScript se puede usar cualquier tipo de dato. Esto puede ser muy útil para generalizar los componentes. También hay que tener claro que todos los componentes están contenidos en un módulo, y que cada módulo tiene su propia configuración. Además, cada módulo puede ser cargado en el cliente de forma individual para no hacer la carga inicial de la página demasiado larga.

Con todas estas condiciones es muy útil crear componentes básicos que se pueden reutilizar como si fuesen los típicos controles de una aplicación. Crearemos un componente tabla que use las tablas incorporadas en Angular. Dado que estos componentes aceptan parámetros, el componente tabla aceptará una lista con los datos a mostrar y una variable con los metadatos que describen la configuración con que estos datos deben ser mostrados. Aquí está una muestra del componente:

Nombre de usuario	Nombre	Correo electrónico	Teléfono	Móvil	DNI	Dirección	Rol	Acciones
Alberto	Andrea Valderrama3	aa2@yahoo.com	222	2222	16603541	Gran vía	Usuario	
Juan	Juan	Juan@gmail.com	222222	22222	1234	Gran Vía, Madrid	Administrador	
Maria	Maria Pérez	maria@gmail.com	12345678	12345678	16693534H	Calle Valcuerna 4, 5°C	Operario	

Con el código del componente tabla es necesario muy poco código para tener un componente tan complicado. En la parte de HTML solo es necesario el siguiente código:

```
<app-table #tableUsers [config]="usersConfig" [data]="usersData" [isLoadingTable]="isLoadingUsers"
(action)="tableEvent($event)">
</app-table>
```

Los estilos CSS de la tabla están predeterminados, pero pueden ser modificados en el CSS del componente donde se usa el componente tabla. Los otros parámetros son referentes al código en el lenguaje TypeScript de Angular. Estos parámetros son la configuración de la tabla donde se le pasan sus campos y forma de representación de cada campo, los datos de las filas de la tabla, una variable booleana que indica si los datos se están cargando o no y, por último, un método para recibir los eventos de la tabla. Estos eventos son disparados al pulsar en los botones de la columna “Acciones” y también se pueden configurar con el primer parámetro. Con esta tabla también se pueden hacer filtros modificando los datos que están en la variable que se le pasa a la tabla como datos de la tabla.

**Revisión del Sprint:** en esta ocasión se ha tardado más en completar la tarea de lo que estaba previsto. Para implementar el componente table en Angular también ha costado algo más de tiempo, ya que ha sido necesario comprobar muchos comportamientos diferentes en la propia tabla. Como el tiempo de esta tarea ha sido de 30 horas, se ha considerado un sprint completo. El cómputo de horas de esta tarea se considerará como la siguiente:

T 3.3 Implementación de la creación de tablas visibles en la página. Estas se podrán rehusar rápidamente en Angular	30
---	----

## 2.5 QUINTO SPRINT

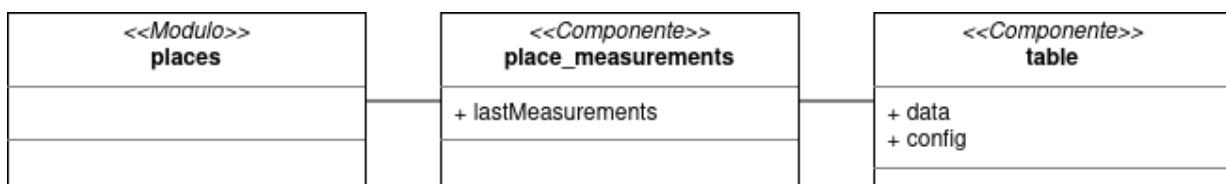
T 3.4 Iniciar el frontend que lee los datos mediante microservicios y mostrar sus últimos valores en la web	20
T 4.1 Implementación de la creación de formularios emergentes rápidamente con Angular. Implementación del HTML y CSS y dejando pendiente la programación	10

### T 3.4 Inicio del frontend que lee los datos mediante microservicios y mostrar sus últimos valores en la web

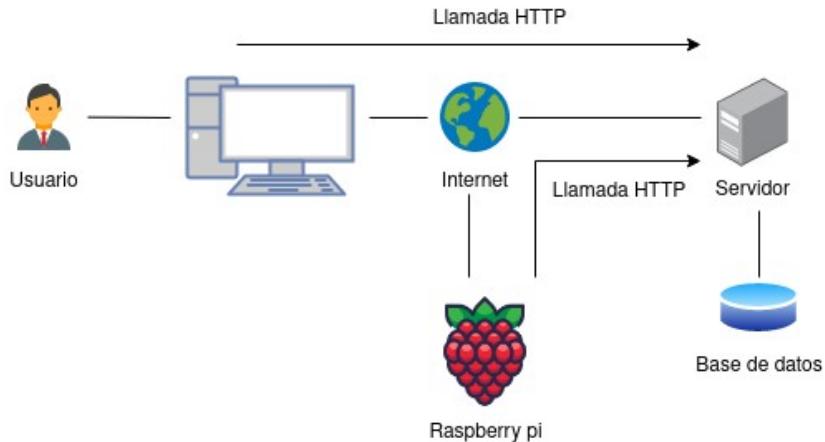
Representación de las mediciones en la página web: aquí está una muestra de la representación:

placeId	date_time	Vibración	Obstáculos	Luz	Fuego	binary_values	has_persons	has_sound	has_oil	has_gas	has_rain	temperature	humidity
2	12/17/22, 15:50:20 PM	No	No	Si	No	10	1023	1023	997	85	1023	17	71
1	12/18/22, 18:26:46 PM	Si	No	Si	No	11	1	354	1007	109	1023	16	67

Con todo esto presente se ha usado el siguiente diseño para representar los datos:



Se ha realizado un módulo para los lugares, ya que este tendrá su propia ruta URL. Este módulo usa el componente place\_measurements, que lo invoca mediante una etiqueta HTML, y, este, a su vez, usa el componente table, el cual accede mediante su etiqueta HTML y añade como atributo los valores de las mediciones. Estos valores se actualizan cada pocos segundos y se actualizan automáticamente en la tabla. La forma de obtener los datos es por medio de una llamada HTTP al back y, en la respuesta, se retornarán los datos sacados de la base de datos. En esta base de datos, la Raspberry Pi de cada lugar habrá puesto los datos de sus mediciones también mediante llamadas HTTP, tal y como se ha visto en los anteriores sprints.



#### T 4.1 Implementación de la creación de formularios emergentes rápidamente con Angular. Implementación del HTML y CSS y dejando pendiente la programación de los eventos. La programación de los eventos se implementará en cada componente que use este componente

Angular tiene utilidades para simular ventanas emergentes en las páginas web que lo usan. También tiene utilidades para manejar formularios, de forma que se pueden manejar simplemente las validaciones de cada campo del formulario y comprobar si la validación de todos los campos está realizada. Otras utilidades permiten comprobar si el conjunto de los campos son válidos. También tiene etiquetas HTML para simplificar la comunicación con el usuario, mostrarle si los campos están en un estado válido y cuál es su nombre. Aquí hay una muestra:

**Datos del usuario**

Nombre de usuario *	Alberto
Nombre *	Andrea Valderrama3
Correo electrónico *	@yahoo.com
Teléfono fijo *	222
Teléfono móvil *	2222
DNI *	16603541
Dirección *	Gran vía
Contraseña	
Role *	Usuario

[Cancelar](#) [Aceptar](#)

Lo que se ha hecho en este componente es unificar y generalizar todas estas utilidades de Angular para realizar formularios emergentes en los que introducir, modificar datos previamente introducidos o incluso introducir valores para buscar filas en concreto. En este caso, para acudir a este componente no se usará una etiqueta HTML, sino que se invocará el formulario emergente con un código como este:

```
const dialogRef = this.dialog.open(DialogComponent, {
  data: this.dialogConfig,
});
dialogRef.afterClosed().subscribe(async result => {
```

El campo “dialogConfig” está configurado de la siguiente manera:

```
export const dialogConfig: IDialogConfig = {
  editable: true,
  columns: [
    { name: 'id', prop: 'id', type: 'text', canView: false },
    { name: 'usersTable.Username', prop: 'user_name', type: 'text', validators: 'nameUser' },
    { name: 'usersTable.name', prop: 'name', type: 'text', validators: 'nameUser' },
    { name: 'usersTable.correoElectronico', prop: 'email', type: 'email' },
    { name: 'usersTable.phone', prop: 'telephone', type: 'text', validators: 'validatePhone' },
    { name: 'usersTable.celular', prop: 'celular', type: 'text', validators: 'validatePhone' },
    { name: 'DNI', prop: 'dni', type: 'text', validators: 'validateDNI' },
    { name: 'usersTable.direccion', prop: 'address', type: 'text', validators: 'validateDireccion' },
    { name: 'usersTable.pass', prop: 'pass', type: 'password', canSearch: false, validators: 'password' },
    { name: 'usersTable.rol', prop: 'roleText', type: 'combo', arrayValues: [0, 1, 2], arrayShows: ['Administrador', 'Operario', 'Usuario'] },
  ],
  title: 'usersTable.userData',
  action: 'insert'
}
```

Cada campo tiene su propia configuración y, antes de abrir el formulario emergente, también hay que pasar el valor que va a tener el campo. Una vez abierto el formulario emergente, se define el evento de su cierre para interceptar sus datos en el momento de cierre y cuál ha sido la forma en la que se ha cerrado para actuar en consecuencia. Este evento también tiene que implementarse desde el componente que lo usa. También se han definido métodos de validación para cada campo. El estado de la validación es visible por el usuario mientras los campos se van llenando.

La parte más compleja del componente está en el código HTML, porque hay que ajustar muchos parámetros por todos los atributos que tiene cada campo. Aun así, quedan bastantes tareas por hacer, como validaciones de los controles de los botones para que actúen de una forma u otra si todos los campos están completos o no. También hay que tener en cuenta diferentes comportamientos como que la contraseña es un campo necesario al crear un nuevo registro, pero no lo es al actualizarlo.

**Revisión del Sprint:** todas las tareas se han completado en el tiempo previsto.

## 2.6 SEXTO SPRINT

T 4.1 Implementar la creación de formularios emergentes rápidamente con Angular. Terminar la parte de la implementación en TypeScript	10
T 4.2 Crear la tabla con sus relaciones en la base de datos y sus correspondientes microservicios administrando el identificador de sesión.	5
T 4.3 Realizar el CRUD mediante llamadas en el frontend	5
T 4.4 Gestionar los usuarios con sus visualizaciones, filtros y su correspondiente CRUD e inicio de sesión desde el frontend	10

#### **T 4.1 Implementación de la creación de formularios emergentes rápidamente con Angular. Terminando esta parte completando la implementación.**

Como se puede ver en la imagen, se ha implementado el código y hay más funcionalidades, como la deshabilitación del botón de aceptar si no están todos los campos correctos:

**Datos del usuario**

Latitud \*

23

---

Longitud \*

3

---

Dirección \*

Barrón

---

Identificador \*

Alberto

---

Contraseña \*

\*\*\*

---

Tiempo de actualización \*

▼

---

Personas

---

#### **T 4.2 Creación de la tabla con sus relaciones en la base de datos y sus correspondientes microservicios administrando el identificador de sesión.**

La creación de la tabla, consiste simplemente en la creación de la tabla por el ORM llamado Sequelize. En el caso, el identificador de sesión hay que implementarlo en JavaScript en la parte del Backend. Para generar el identificador de sesión, se usa una librería de JavaScript llamada bcrypt. Con esta librería se realiza el hash de la contraseña para guardarla en la base de datos. Cuando un usuario se autentica en la página, se comprueba si la contraseña se corresponde con ese hash. En el que será el modo producción, la contraseña se transportará por la red cifrada mediante el protocolo HTTPS. Para generar el identificador de sesión se genera un token introduciendo los datos del usuario, a excepción del hash de la contraseña y, este token, que estará cifrado, se manda al cliente como respuesta. El cliente almacena el token en el navegador y lo manda en las cabeceras de cada llamada HTTP. Esta cabecera es examinada por el servidor cuando recibe las llamadas, el cual comprueba que contiene el token. El token se valida comprobando que no está caducado y que se corresponde con algún usuario existente en la base de datos. Si estas comprobaciones no tienen éxito, se retorna un error al cliente, y si tiene éxito, se hace la operación. No todas las llamadas HTTP requieren un token, la autenticación para obtener el token es una de ellas. Aquí tenemos un ejemplo de la llamada que genera un token. Esta llamada está incluida en la lista de endpoints del anexo y es una de las más importantes.

Esta llamada es usada para hacer la autenticación en la página:

The screenshot shows the Postman application interface. At the top, there's a header bar with 'POST' and the URL `((host)):(port)/api/persons/authenticate`. On the right side of the header are 'Send' and 'Cookies' buttons. Below the header, there are tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Body' tab is currently selected, indicated by a green dot. Under the 'Body' tab, there are dropdowns for 'none', 'form-data', 'x-www-form-urlencoded', 'raw', 'binary', 'GraphQL', and 'JSON'. The 'JSON' option is selected and expanded, showing a JSON object with fields: 'user\_name' (value: "Alberto") and 'pass' (value: "abc"). Below the body editor, there are tabs for 'Body', 'Cookies', 'Headers (7)', and 'Test Results'. The 'Test Results' tab is selected, showing a status message: 'Status: 200 OK Time: 144 ms Size: 913 B'. To the right of the status message are 'Save Response' and search/filter icons. The main content area displays the response body as a JSON object:

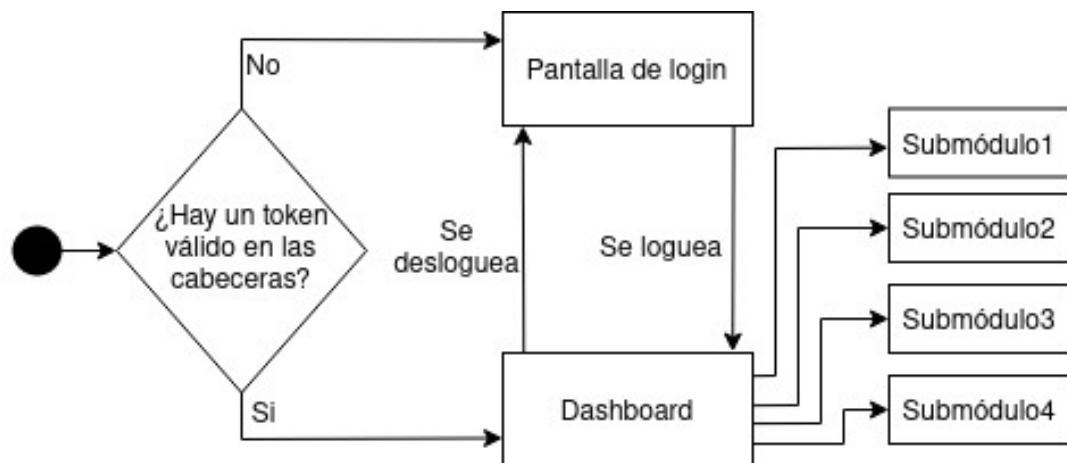
```
1 "ok": true,
2 "usuario": {
3     "id": 1,
4     "dni": "16603541",
5     "name": "Andrea Valderrama3",
6     "user_name": "Alberto",
7     "telephone": "222",
8     "celular": "2222",
9     "address": "Pequeña via 112, 4 B",
10    "email": "aa2@yahoo.com",
11    "roles": 2
12 },
13 "token": "eyJhbGciOiJIUzI1NiIsInRcCjI6IkpxVCJ9.
14 eyJlcVhcmIiJp7imKjoxLCjkbmki01ixNjYmZu0MSisIm5hbWUi01JBbmRyZGvcmFzYTMIcJ1c2VyX25hbWUo1JBbGj1cnRvIiwicGFzcI6Ii0yYiQxMCRGcX1YNGlsSXNweEk4M1VmVm9uenQuMuDyemZja1RPLmF4en1jZTF
15 JNUNaJbaSjXRgVBZSisInRlGVwaG9uZS16IiyM1isInNlbHVsYX1i01IiyM1iyIiwiYWRkcmVzci6IlBlcXVlw7FhIHbDzWEgMTEyLCA0IEi1LCj1bWFpbD6ImFhMkB5YWhvbby5jb201LCJyb2x1cyI6Mn0sImlhdC16HTY3Mj05Ng10CwizX
hwIjoxNjcyNjY4NjU4fQ.OImtgNtuBuZonWHKznH_B-VTCeaFjNt1c0g15rspws"
```

#### T 4.3 Realización del CRUD mediante llamadas en el frontend

Las llamadas son las típicas de una tabla, con las cualidades especiales explicadas en el punto anterior. Esto hace cambiar unas cuantas cosas en la implementación del frontend. Hay un método para interceptar las llamadas HTTP antes de que las haga el navegador e injectar el identificador de sesión para que la pueda reconocer el servidor. También hay que incluir en esta parte la implementación del guardado del token en las cookies de sesión del navegador.

**T 4.4 Gestión de los usuarios con sus visualizaciones, filtros y su correspondiente CRUD e inicio de sesión desde el frontend**

Esta parte consiste en dos pasos bien diferenciados: realización del inicio de sesión y gestión de los usuarios. En el caso del inicio de sesión, requiere una estructuración de los módulos y rutas del frontend. Se han estructurado de la siguiente forma:



De esta forma, cuando se accede a la aplicación web lo primero que se hace es comprobar si hay un token guardado en las cookies de sesión del navegador. Si hay uno, se hace una llamada al servidor para comprobar si el token es válido. Si lo es, se va a la URL solicitada. En caso contrario, se va a la pantalla de autenticación. En el caso de que la URL solicitada sea la de pantalla de inicio de sesión, pero ya se tenga un token válido, se redirige a la URL principal de Dashboard, ya que no tiene sentido autenticarse dos veces. Todas las funcionalidades de la aplicación web están en las URLs que cuelgan de la URL del Dashboard. De esta forma, una vez que se está autenticado, la única forma de volver a la pantalla de login es pulsar el botón de cerrar sesión o que caduque el token de sesión.

La parte de la gestión de los usuarios se ha completado como se puede ver en la pantalla:

Nombre de usuario	Nombre	Correo electrónico	Teléfono fijo	Teléfono móvil	DNI	Dirección	Rol	Acciones
juanito	Juan Balderama	Juas@gmail.com	222	222	222	Calle Chile 2º B	Usuario	
Alberto	Andrea Valderrama3	aa2@yahoo.com	222	2222	16603541	Pequeña vía 112, 4 B	Administrador	

Esta es la pantalla desde la que se administran los valores de los usuarios. Cada una de las filas representa un usuario y sus datos. En cada fila tenemos tres acciones: modificar los datos, eliminar el usuario y ver todos sus valores. Al pulsar en la acción de modificar los valores nos sale la ventana emergente, como se puede ver en la captura, se ve en tiempo real la validación de los datos:

#### Datos del usuario

Nombre de usuario \*Alberto

Nombre \*  
Andrea Valderrama3

Correo electrónico \*  
aa2@yahoo.com

Teléfono \*  
222

Móvil \*  
2222

DNI \*  
16603541

Dirección \*  
Gran vía

Contraseña

Rol \*  
Usuario

Encima de la tabla hay más botones. El de la derecha es para crear un nuevo usuario. Sale una ventana emergente como la anterior para llenar los datos. El botón de la izquierda es para realizar una búsqueda:

Nombre de usuario	Nombre	Correo electrónico	Teléfono fijo	Teléfono móvil	DNI	Dirección	Rol	Acciones
Alberto	Andrea Valderrama3	aa2@yahoo.com	222	16603541	Pequeña vía 112, 4 B	Administrador		
Juanito Balderama	Juanito	juanito@gmail.com	2222	16644	Calle Chile	Usuario		

Se puede llenar uno o varios campos para buscar similitudes con los datos introducidos en cada campo. Como se puede ver, aquí no está la contraseña. Al introducir parte del nombre y pulsar en “Aceptar” en la tabla quedan las filas coincidentes con todos estos parámetros. Las columnas no introducidas se ignoran:

Nombre de usuario	Nombre	Correo electrónico	Teléfono fijo	Teléfono móvil	DNI	Dirección	Rol	Acciones
Alberto	Andrea Valderama3	aa2@yahoo.com	222	2222	16603541	Pequeña vía 112, 4 B	Administrador	

Items per page: 10 | < > | 1 - 1 of 1

Solo hay una fila que coincide con todos estos parámetros. Para reiniciar el filtro y ver todas las filas otra vez, solamente hay que pulsar en el botón situado en el centro.

**Revisión del Sprint:** todas las tareas se han completado en el tiempo previsto.

## 2.7 SEPTIMO SPRINT

T 5.1 Hacer la interfaz para que el usuario pueda manejar las salidas de Arduino	10
T 6.1 Modificar los microservicios de acceso a los datos para que admitan parámetros de intervalos de fechas	5
T 6.2 Realizar los filtros para recibir rangos de fechas en el frontend	5
T 6.3 Mostrar gráficos de los datos requeridos en el frontend	10

### T 5.1 Hacer la interfaz para que el usuario pueda manejar las salidas de Arduino

Se ha realizado una vista para mostrar las últimas salidas de un determinado lugar registradas en el servidor:

The screenshot shows a user interface for managing outputs at a specific location. The location details are: Nombre: aaaa, Dirección: Pequeña vía 112, 4 B. Below this, there are four output controls labeled Primera salida, Segunda salida, Tercera salida, and Cuarta salida. Each control has a blue 'ON' button and a grey 'OFF' button. To the right of each button is its current state: Enviado for the first and fourth, and No enviado for the second and third.

Cada lugar tiene cuatro salidas. Se muestra el último estado grabado en el servidor. Este estado es modificado por el usuario al pulsar en el interruptor. En la columna de la derecha pone una indicación para cada salida. Estos dos estados son “Enviado” y “No enviado”. Un estado en la salida enviada significa que la Raspberry Pi de ese lugar ha leído ese estado, por lo tanto el mensaje de “No enviado” significa que la Raspberry de ese lugar todavía no ha leído este estado y que, por lo tanto, ese estado no ha llegado al lugar, ya que todavía no ha sido transmitido. Hay que tener claro que un estado de una salida esté enviado solo quiere decir que ha llegado al Arduino, pero no es seguro que esté en el mundo real, ya que puede haber algún problema en la parte del sistema que va entre la Raspberry Pi hasta el relé. Esta vista ha sido guardada, en un componente para reutilizarla más tarde. Será reutilizado, para mostrar las salidas en la misma pantalla que las entradas, para que el usuario pueda ver los defectos de las salidas sobre las entradas. El funcionamiento de esta parte está basado en el endpoint que modifica la tabla de acciones. También está incluido en la lista de endpoints del anexo.

### T 6.1 Modificar los microservicios de acceso a los datos para que admitan parámetros de intervalos de fechas

Para lograr esto, en las llamadas HTTP lo que se ha hecho es poner sufijos a los parámetros que se llaman como los campos a filtrar. Aquí tenemos un ejemplo con la llamada: “api/measurements?placeId=1&date\_timeFINISH=2023-01-1T17:33:20.007Z&date\_timeBEGIN=2022-12-17T12:33:10.007Z&date\_timeORDERDESC=0&LIMIT=10”

La respuesta a esta llamada viene en formato JSON como se puede ver en la captura. En esta sentencia se obtienen las mediciones del lugar con el id=1 y que están en el intervalo de tiempo indicado. Hay una limitación de registros para que la llamada no sea demasiado larga, hay que tener cuidado con este valor, ya que es posible que no obtengamos todos los datos. Los sufijos para hacer esta funcionalidad son:

FINISH: para obtener los registros anteriores a la fecha que se le pasa como valor en el campo cuyo nombre está anterior al FINISH

BEGIN: para obtener los registros posteriores a la fecha indicada en el campo indicado anteriormente

ORDERDESC: para obtener un orden descendente en ese campo

LIMIT: como se puede ver en el ejemplo, es un parámetro completo, limita el número de registros en la respuesta para que la respuesta no sea demasiado grande

```

{
  "data": [
    {
      "id": 60517,
      "date_time": "2023-01-01T17:33:02.000Z",
      "binary_values": 10,
      "has_persons": 758,
      "has_sound": 325,
      "has_gas": 117,
      "has_water": 90,
      "has_rain": 983,
      "temperature": 17,
      "humidity": 69,
      "placeId": 1
    },
    {
      "id": 60516,
      "date_time": "2023-01-01T17:33:32.000Z",
      "binary_values": 10,
      "has_persons": 760,
      "has_sound": 325,
      "has_gas": 117,
      "has_water": 90
    }
  ]
}
  
```

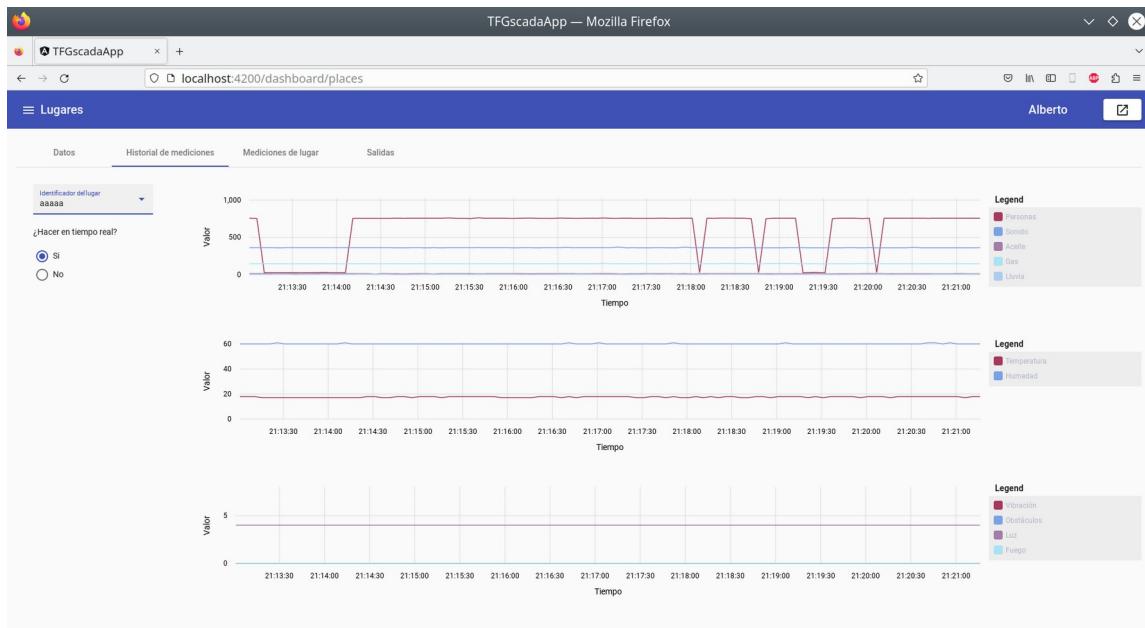
## T 6.2 Realizar los filtros para recibir rangos de fechas en el frontend

Antes de hacer estos controles hay que decidir las posibles intenciones del usuario. Se ha puesto un selector para que el usuario escoja el lugar del que se va a mostrar el gráfico de mediciones. También se le va a dar a escoger al usuario entre ver el historial de un rango de lecturas hasta un determinado momento o las más actuales. Esta opción ha sido la llamada “En tiempo real”. En el caso de escoger ver las mediciones hasta un momento en concreto, se le da al usuario la posibilidad de escoger una fecha y una hora para ver el gráfico de las mediciones anteriores:

### T 6.3 Mostrar gráficos de los datos requeridos en el frontend

Se ha escogido la librería para Angular llamada gnx-charts que usa una licencia MIT, este tipo de licencia puede ser consultada en el anexo de definiciones. Se han realizado tres gráficas: una con las mediciones analógicas, otra para las señales digitales y otra para las lecturas de humedad y temperatura realizadas por el sensor DHT11. En el caso de las mediciones digitales, para que no se solapen una sobre la otra, se ha escogido un rango diferente para cada medición.

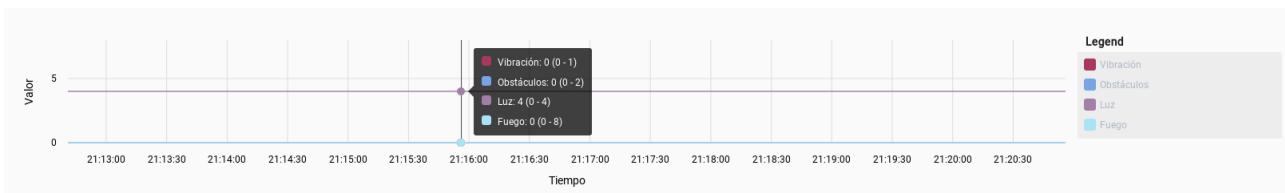
En la escala temporal se ha puesto una hora, que es el momento en el que se hace cada medición. La cantidad del tiempo representado en las gráficas depende del número de mediciones realizadas, esto se ha hecho así para que los valores expuestos no sean demasiado altos y no queden demasiado juntos. En la opción de ver las mediciones en tiempo real, las lecturas se van actualizando cada varios segundos. Cuando eso ocurre el gráfico va avanzando a la izquierda y los tiempos se van actualizando:



En el caso de escoger la opción de ver las mediciones anteriores a un determinado momento, se pondrán las opciones como se muestra en la imagen:



En este caso, los valores no se van actualizando. Solo se actualiza cuando se cambia la fecha y la hora de las opciones.



Una utilidad de estos menús es que al colocar el ratón encima de un momento dado se pueden ver los valores de ese momento y sus intervalos.

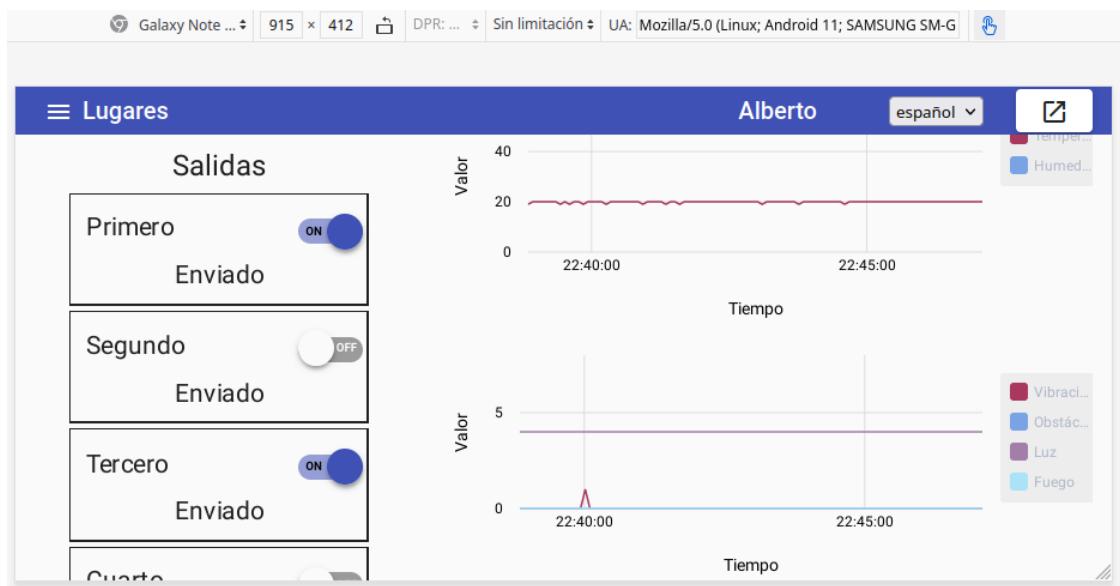
Finalmente, se han unido los componentes de modificar las salidas del relé con visualizar las salidas en los gráficos. De esta forma, al modificar los estados de las salidas se pueden ver las modificaciones de las entradas de los sensores en tiempo real, sin ser necesario cambiar de lugar. Por ejemplo: en el caso en el que el nivel de humedad sea demasiado bajo, se puede activar la salida que está conectada al riego en ese lugar e ir comprobando el sensor de humedad para ver cómo se incrementa esta medida.



Esta pantalla ha sido especialmente complicada de mostrar en pantallas pequeñas. Se han hecho concesiones, pero se ha logrado un resultado aceptable:



Es necesario ver la página en el modo ancho de los dispositivos móviles y usar el scroll vertical, pero se pueden ver todos los datos y usar todas las funcionalidades:



Hay que tener en cuenta que este no es el modo preferente de usar esta aplicación web, aunque sí es posible usarla de esta manera.

**Revisión del Sprint:** todas las tareas se han completado en el tiempo previsto.

## 2.8 OCTAVO SPRINT

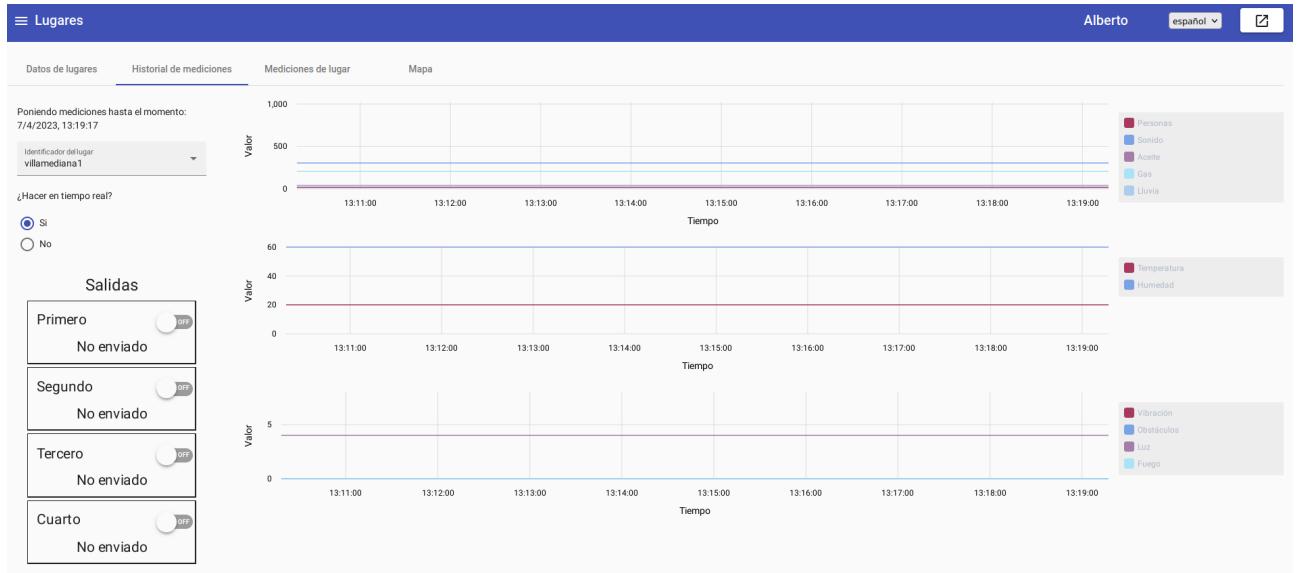
T 7.1 Realizar programa que simule las llamadas de múltiples lugares que envían datos	10
T 7.2 Realizar un historial de acciones que pueden ser examinadas por el usuario	10
T 7.3 Realizar un historial de alarmas que pueden ser examinadas por el usuario	10

### T 7.1 Realizar programa que simule las llamadas de múltiples lugares que envían datos

Se ha realizado un prototipo para simular las operaciones de un lugar físico. De esta forma, se hace una buena prueba de la lectura de datos y el cambio de salidas. Sin embargo, no podemos comprobar cómo se comporta el servidor al recibir mediciones de muchos lugares a la vez. Para simular esto se ha realizado un algoritmo que se ejecuta cada cinco segundos y que hace una llamada HTTP, al propio servidor, por cada lugar que está en la base de datos, a excepción del lugar en el que está realizado físicamente. Los valores de las mediciones de los lugares que están en la base de datos, pero que no existen, se extraen de la última medición del lugar que sí existe. En las capturas se puede ver como en varios lugares están las mismas mediciones:



Esta es una prueba de estrés para el sistema y que demuestra que funciona perfectamente. Sin embargo, para comprobarlo correctamente habría que generar tantos lugares en el prototipo como lugares tengamos en el sistema real.



En estas capturas están varios lugares diferentes. Todos tienen datos de variables, pero solamente el último es un lugar real. Los datos del resto de lugares han sido simulados.

Este sistema se usa o no dependiendo de la configuración de las variables de entorno del backend. En el caso de puesta en producción, este algoritmo quedará deshabilitado en la producción. Dado que esta inserción de datos se hace mediante llamadas HTTP, este algoritmo es una buena prueba de estrés para la posterior puesta en producción.

### T 7.2 Realizar un historial de acciones que pueden ser examinadas por el usuario

Las acciones son las manipulaciones de los usuarios sobre las salidas de los lugares. Estas manipulaciones quedan registradas en la base de datos y en la aplicación puede ser revisado este historial en la siguiente sección:

Fecha	Usuario	Salida	Dirección	Lugar	Enviado	Valor
3/18/23, 18:56:36 PM	Alberto	Cuarto	Avenida Perez Galdós	aaaaaa	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3/18/23, 18:56:31 PM	Alberto	Cuarto	Avenida Perez Galdós	aaaaaa	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
3/18/23, 18:54:33 PM	Alberto	Segundo	Avenida Perez Galdós	aaaaaa	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3/18/23, 18:54:22 PM	Alberto	Segundo	Avenida Perez Galdós	aaaaaa	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
3/18/23, 18:54:13 PM	Alberto	Segundo	Avenida Perez Galdós	aaaaaa	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3/18/23, 18:39:36 PM	Alberto	Segundo	Avenida Perez Galdós	aaaaaa	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
3/18/23, 18:39:33 PM	Alberto	Primero	Avenida Perez Galdós	aaaaaa	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
3/11/23, 11:06:26 AM	Alberto	Primero	Avenida Perez Galdós	aaaaaa	<input type="checkbox"/>	<input type="checkbox"/>
2/5/23, 15:16:47 PM	Alberto	Segundo	Avenida Perez Galdós	aaaaaa	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2/5/23, 15:16:46 PM	Alberto	Tercero	Avenida Perez Galdós	aaaaaa	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Elementos por página: 10 | 1 – 10 of 46 | < > >>

De esta forma podemos visualizar la salida que ha accionado cada usuario con los datos de su fecha, el lugar donde ha accionado la salida, si ha sido enviada correctamente a ese lugar y el valor que se ha mandado. El recuadro está marcado para salidas de nivel alto y desmarcado para salidas de nivel bajo. También podemos ver si la acción ha sido enviada al lugar. Encima de la tabla tenemos filtros para ver estas salidas. Podemos filtrar por usuario, lugar, nivel de salida e intervalo de la fecha en la que está registrada. Pulsando en el nombre de los campos, también podemos hacer ordenaciones por cada campo, tanto ascendente como descendente.

Hay que tener claro que esta tabla es solamente de consulta, así que solamente podemos consultar los datos.

### T 7.3 Realizar un historial de alarmas que pueden ser examinadas por el usuario

Hay valores límite de los sensores que son peligrosos y son valores que además pueden cambiar. Para reflejar estos eventos están las alarmas. Desde el servidor se van examinando constantemente los valores de los sensores y, en el caso de que estos valores estén dentro de unos rangos predeterminados, se crean las alarmas. Estas pueden ser visualizadas en la siguiente sección:

Fecha de inicio	Finalizado	Sensor	Operador encargado	Lugar	Acciones
19/2/23, 10:40:35 AM	-	Detector de un obstáculo delante del sensor	bbbbbb		
18/2/23, 21:32:17 PM	18/2/23, 21:32:52 PM	Detector de un obstáculo delante del sensor	aaaaaa		
18/2/23, 21:32:07 PM	18/2/23, 21:32:12 PM	Detector de un obstáculo delante del sensor	aaaaaa		
18/2/23, 20:39:47 PM	18/2/23, 20:40:22 PM	Detector de un obstáculo delante del sensor	aaaaaa		
18/2/23, 20:36:17 PM	18/2/23, 20:36:47 PM	Detector de un obstáculo delante del sensor	aaaaaa		
18/2/23, 20:32:17 PM	18/2/23, 20:32:52 PM	Detector de un obstáculo delante del sensor	Andrea Valderrama3	bbbbbb	
18/2/23, 20:32:07 PM	18/2/23, 20:32:12 PM	Detector de un obstáculo delante del sensor	bbbbbb		
18/2/23, 20:32:07 PM	18/2/23, 20:32:22 PM	Detector de un obstáculo delante del sensor	aaaaaa		
18/2/23, 20:31:22 PM	18/2/23, 20:31:57 PM	Detector de un obstáculo delante del sensor	Andrea Valderrama3	aaaaaa	
18/2/23, 20:30:17 PM	18/2/23, 20:31:02 PM	Detector de un obstáculo delante del sensor	aaaaaa		

Items per page: 10 | 1 – 10 of 33 | < > >>

Las alarmas tienen un momento de inicio, que es el momento en el que son detectadas, y un momento de finalización, que es el momento en el que el valor del sensor de la alarma ha vuelto a un valor fuera del rango considerado como peligroso. También tiene un tipo de sensor que ha generado la alarma, un usuario que es el responsable de tratar la alarma y un lugar donde ha ocurrido. Tanto la fecha de finalización como el usuario, tienen valores indeterminados cuando nadie ha acusado la alarma y si el rango de las mediciones está en el rango de valores de alarma para ese tipo de sensor. El proceso de asignar un usuario a una alarma se hace pulsando en el botón de la tabla marcado con un pulgar arriba. Cuando se hace eso, el usuario autenticado en la página queda encargado de la alarma, ya que es él mismo quien ha pulsado en

el botón. Como se puede ver en las capturas, cada alarma puede tener tres colores: rojo, verde y amarillo. Cuando una alarma se muestra en color rojo significa que el valor de ese sensor está en un rango peligroso, por lo que no tiene fecha de finalización y, además, ningún usuario está encargado de tratar la alarma. Cuando está mostrada en amarillo significa que no tiene un usuario encargado de tratar esa alarma. Si la alarma tiene fecha de finalización es mostrada en color verde, ya que no es peligrosa. El estado de las alarmas se puede ver en los datos de las tablas y no es necesario ver el código de colores, pero es una forma de diferenciar la clasificación rápidamente.

The screenshot shows a table titled 'Alarms' with the following columns: Fecha de inicio, Finalizado, Sensor, Operador encargado, Lugar, and Acciones. The table contains 10 rows of data, each representing an alarm. The rows are colored green, yellow, and blue, corresponding to different sensor types or states. At the top of the table, there are five filter dropdowns: 'Rango para la fecha de inicio' (08/02/2023 – 22/02/2023), 'Rango para la fecha de fin' (empty), 'Sensor' (Todos), 'Usuario' (Todos), and 'Identificador del lugar' (Todos). Below the table are pagination controls: 'Items per page: 10', '1 – 10 of 33', and navigation arrows.

Como se puede ver, esta tabla también tiene filtros. Se puede filtrar por los cinco campos de la tabla. Los campos de las fechas son casos especiales. Estos se filtran por rangos de fechas y tienen una interfaz correspondiente para ello:

This screenshot shows the same 'Alarms' table interface, but the 'Rango para la fecha de inicio' field is populated with '08/02/2023 – 22/02/2023'. The date range selector shows a calendar for February 2023, with the range highlighted. The rest of the table and filters are identical to the previous screenshot.

Se puede escoger entre intervalos de días, tanto para la fecha de inicio como la fecha de fin. Si se escoge un rango donde no hay alarmas, el resultado es que no se encuentran alarmas:

This screenshot shows the 'Alarms' table with the 'Rango para la fecha de inicio' field set to '21/02/2023 – 23/02/2023'. The table below displays the message 'No results found'. The rest of the interface, including filters and pagination, remains the same.

Una utilidad especial de esta tabla es que está siendo actualizada cada varios segundos. Gracias a esto podemos ver sin ser necesario actualizar la página como estos registros están progresando.

Los rangos de valores en los que se crean las alarmas para cada tipo de sensor están guardados en la base de datos y estos son los correspondientes valores:

123 idSensor	ABC name	ABC description	123 range_low	123 range_high
5	Gas detectado	Se ha detectado un nivel alto de gas inflamable	300	1.024
8	Temperatura excesiva	La temperatura es demasiado alta	50	200
11	Fuego detectado	Se ha detectado fuego en el sensor de fuego	1	1
4	Obstaculo detectado	Se ha detectado un obstáculo en el sensor	1	1

Según estos rangos de valores, la alarma de gas detectado se genera cuando el valor de este tipo de sensor es alto, la alarma de temperatura se genera cuando la temperatura es igual o mayor de 50 grados y en el caso de sensor de fuego y obstáculo se genera cuando su valor de entrada está activado, ya que el valor de entrada de estos sensores es digital.

**Revisión del Sprint:** todas las tareas se han completado en el tiempo previsto en conjunto. La primera ha tardado 5 horas menos y la última 5 horas más.

## 2.9 NOVENO SPRINT

T 8.1 Realizar la internacionalización de la aplicación web. Poner los textos a varios idiomas.	15
T 8.2 Probar varios tipos de usuario abarcando todos los roles. Y todas las acciones para cada uno.	5
T 8.3 Realizar un mapa representando la posición de los lugares	10

### T 8.1 Internacionalización de la aplicación web. Poner los textos a varios idiomas.

Para realizar esta tarea se ha utilizado una librería de Angular para la internacionalización. Esta librería simplifica mucho las cosas en la mayoría de los casos, ya que sustituye todos los textos que se le pasa por su equivalente en el idioma seleccionado, tanto en el código HTML como en el código TypeScript de Angular. Hay algunos casos donde ha sido necesario hacer algunas modificaciones extras para traducir el texto en las gráficas, incluir las traducciones en las tablas y las ventanas emergentes.

### T 8.2 Probar varios tipos de usuario abarcando todos los roles. Y todas las acciones para cada uno.

Se han comprobado los permisos con usuarios con diferentes roles. Este es el ejemplo de los diferentes usuarios que tenemos:

Nombre de usuario	Nombre	Correo electrónico	Teléfono	Móvil	DNI	Dirección	Rol	Acciones
Francisca	Guadalupe	sfsdf@gmail.com	123456789	123456789	16603534H	Calle Juanola 1º B	Operario	
Maria	Maria Pérez	maria@gmail.com	123456789	123456789	16603535L	Calle Valcuena 4, 5ºC	Operario	
Alberto	Andrea Valderrama	aa2@yahoo.com	1234567890	1234567890	16603536C	Gran vía	Administrador	
Juan	Juan Frías Andres	Juan@gmail.com	113456789	123456789	16603537K	Gran Vía, Madrid	Usuario	

Como se puede ver, hay tres roles diferentes: "Administrador", "Usuario" y "Operario". El usuario de rol "Administrador" tiene permisos para todo: manejar los usuarios y ver todos los lugares. El rol "Operario" no puede administrar los usuarios, pero puede ver todos los lugares. Los usuarios con el rol "Usuario" solamente puede ver los lugares que tiene asignados. En la captura se ha entrado con un usuario administrador y se pueden ver los usuarios. En el caso de los lugares se puede ver todos los disponibles:

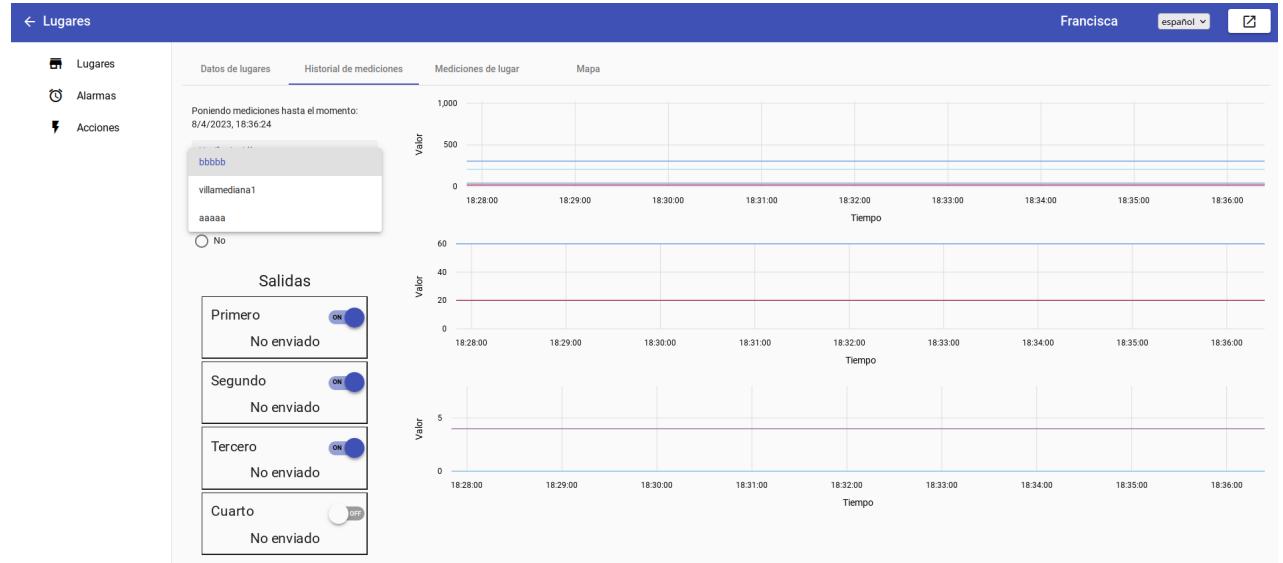
Lugares							
Usuarios Lugares Alarmas Acciones	Datos de lugares		Historial de mediciones		Mediciones de lugar		Mapa
	Latitud	Longitud	Dirección	Identificador	Tiempo de actualización	Usuarios	Acciones
	42.46054701798568	-2.4552900439949643	Calle Chile	bbbbbb	5	Andrea Valderrama Guadalupe	
	42.42795309363744	-2.41947541202865	Plaza central, Villamediana de Iregua	villamediana1	5	Andrea Valderrama Juan Frias Andres Guadalupe	
42.46230448169748	-2.4514472709770523	Avenida Perez Galdós	aaaaaa	5	Maria Pérez		
Elementos por página 10							1 - 3 of 3   < < > >

Con el usuario administrador se puede ver todos los lugares que hay.

En el caso de un usuario con un rol de “Operario” el resultado es el siguiente:

Lugares							
Lugares Alarmas Acciones	Datos de lugares		Historial de mediciones		Mediciones de lugar		Mapa
	Latitud	Longitud	Dirección	Identificador	Tiempo de actualización	Usuarios	Acciones
	42.46054701798568	-2.4552900439949643	Calle Chile	bbbbbb	5	Andrea Valderama Guadalupe	
	42.42795309363744	-2.41947541202865	Plaza central, Villamediana de Iregua	villamediana1	5	Andrea Valderrama Juan Frias Andres Guadalupe	
42.46230448169748	-2.4514472709770523	Avenida Perez Galdós	aaaaaa	5	Maria Pérez		
Elementos por página 10							1 - 3 of 3   < < > >

Como se puede ver, en la captura, en este caso no se puede ver el acceso a los usuarios y, además, la URL está protegida mediante guardas en Angular. De forma que si escribimos la URL de los usuarios, en el navegador, tampoco podremos acceder a ella. También se puede ver que se pueden visualizar todos los lugares y sus medidas. También en el caso de la selección de los desplegables:



La comprobación de las alarmas y las acciones también ha sido la indicada.

Alarmas								
Lugares Alarmas Acciones	Rango de la fecha de inicio		Rango de la fecha de fin		Sensor	Operador	Acciones	
	08/04/2022 - 08/04/2023		Finalizado		TODOS	TODOS	TODOS	
	Fecha de inicio	Finalizado	Sensor	Operador encargado	TODOS	TODOS	aaaaa	bbbbb
	10/2/23, 11:40:34 AM	19/2/23, 11:41:59 AM	Detector de un obstáculo delante del sensor	Andrea Valderrama	aaaaa	aaaaa	villamediana1	

En el caso de un usuario con el rol de “Usuario” el resultado es el siguiente:

Como se puede ver en las capturas, solamente se puede acceder al lugar que tiene asignado y no se puede acceder tampoco a la configuración de los usuarios. Se puede ver que un mismo lugar puede tener varios usuarios y, varios usuarios pueden tener diferentes lugares.

En el caso de las alarmas el resultado es el siguiente:

Como se puede ver, solamente es posible acceder al único lugar que tiene asignado y, que este no tiene alarmas.

En el caso de las acciones:

Como se puede ver el resultado es el mismo.

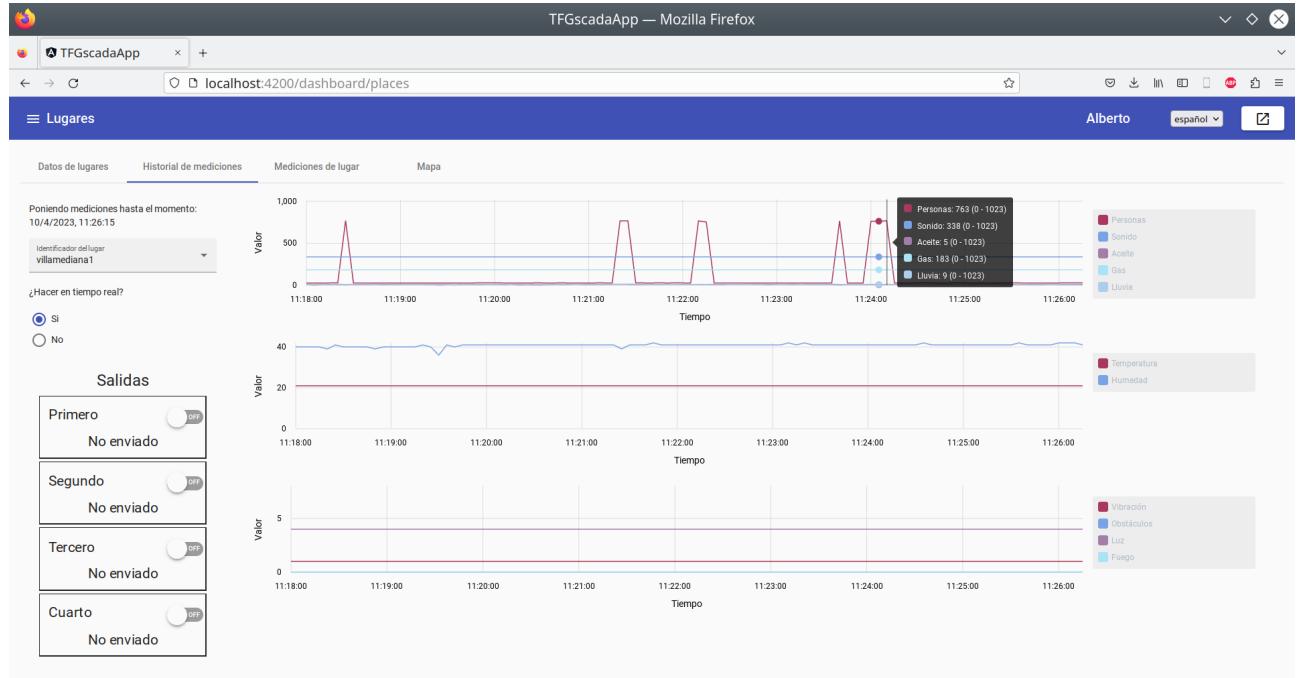
### T 8.3 Realizar un mapa representando la posición de los lugares

Cada lugar tiene una latitud y una longitud, por lo que pueden ser visualizados en su posición en un mapa. Para realizar esto se ha utilizado la librería OpenLayers, la cual tiene una licencia BSD, por lo que puede ser utilizada libremente. Se usan mapas en la pantalla y en ellos se ubican los lugares. Aquí se puede ver una muestra:

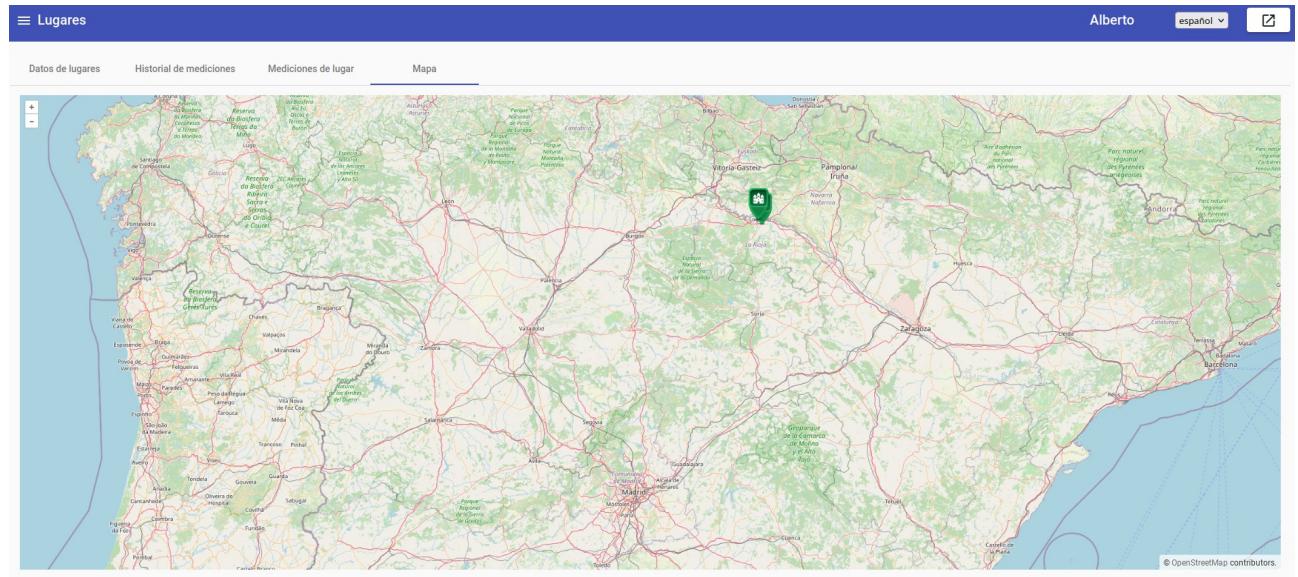
La vista de mapa muestra a todos los lugares en su correspondiente lugar, sacados de su correspondiente latitud y longitud. De esta forma se tiene una representación de las posiciones de los lugares en el mapa.

Al poner el ratón encima del icono de un lugar aparece el identificador de ese lugar:

En este caso, se puede ver el tooltip con el identificador “villamediana1”. Si se pulsa en el lugar, el tooltip se mueve al “Historial de mediciones” con ese lugar seleccionado:



Esta es una forma rápida de recurrir a un lugar sabiendo su posición. En el caso de que haya muchos lugares y estos estén muy separados, se puede cambiar la visión del mapa para verlos claramente. Para eso están los iconos de zoom en la parte superior e izquierda de la pantalla. También se puede hacer zoom con la rueda del ratón. Aquí se puede ver un ejemplo de un zoom:



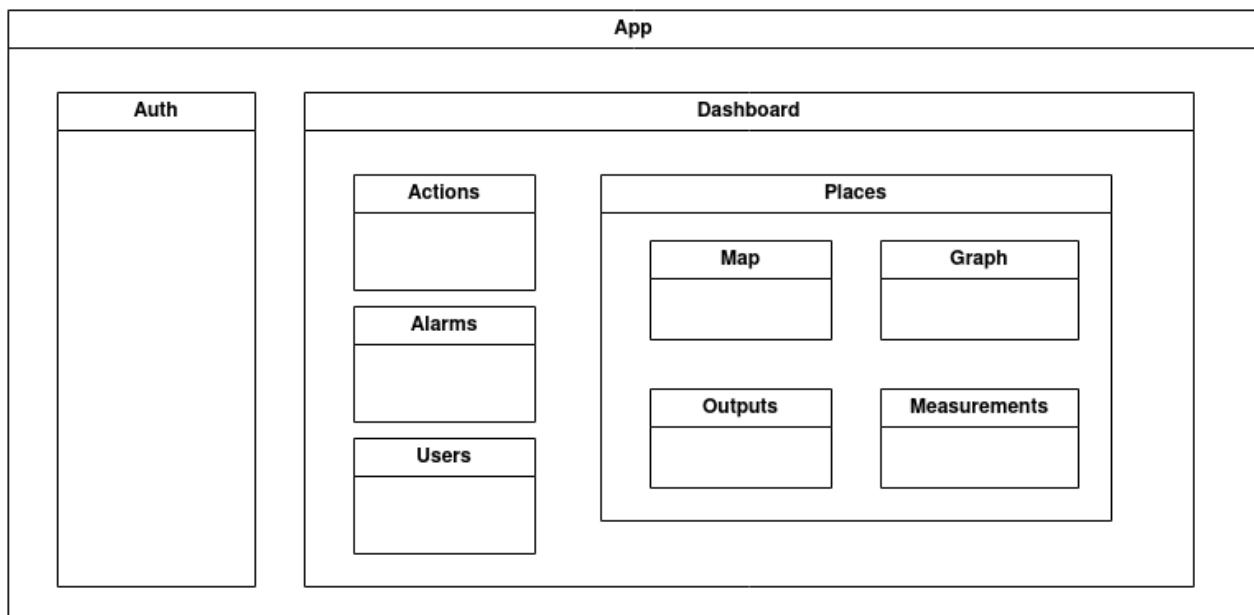
Como se puede ver en esta captura, es necesario cambiar la perspectiva para seleccionar un lugar cuando tiene otros muy juntos.

**Revisión del Sprint:** todas las tareas se han completado en el tiempo previsto.

## CAPÍTULO 3: APARTADOS FINALES

### 3.1 ESTRUCTURA Y RELACIONES DE LOS MÓDULOS DE ANGULAR

Esta es la forma en la que se ha estructurado la parte de frontend realizada con Angular. Como ya se ha comentado, Angular está estructurado en módulos y componentes. Cada módulo contiene componentes y otros módulos en una estructura en forma de árbol. Esta es la estructura que se ha elegido para la parte de frontend:



Cada módulo se carga por separado en el navegador del cliente. De forma que en la primera carga de la página solamente se cargan los módulos necesarios en ese momento. Por esta razón se han estructurado los módulos de esta forma. Dentro del frontend, en el primer nivel cuelgan Auth que es para abrir la sesión de la página y el Dashboard que contiene el resto de funcionalidades que requieren una sesión. Dentro del Dashboard está cada elemento del menú principal. Dentro de estos cuatro elementos de menú principales, los módulos se cargan al escoger el elemento del menú, y no por la estructura de los datos que manejan. Dentro del menú “Lugares” hay otros cuatro módulos, uno por cada pestaña del submenú. De esta forma, dentro de cada etiqueta de cada pestaña habrá una etiqueta de un componente diferente. Para relacionarlos con más facilidad se han metido estos cuatro componentes dentro del componente Places.

### 3.2 COMPARACIÓN DE LA PLANIFICACIÓN Y LA REALIDAD

TAREA	SUBTAREA	HORAS PREVISTAS	HORAS REALIZADAS
Planificación inicial	Recopilar las intenciones del proyecto, su alcance y la forma de afrontarlo	15	15
Investigación formación y	Investigar de las tecnologías y la viabilidad de cada una. Contrastar las ventajas e inconvenientes de cada tecnología	5	5
	Compra de los componentes, instalación del software necesario que se necesita para manipularlos	10	10
T 1 Lectura de datos con Arduino y los sensores	T 1.1 Montar del cableado	10	10
	T 1.2 Realizar las lecturas de datos de cada sensor	10	10
T 2 Conexión entre Arduino y Raspberry pi para persistir los datos	T 2.1 Comunicar Arduino y Raspberry Pi	10	10
	T 2.2 Diseñar y crear de la base de datos	5	5
	T 2.3 Persistir datos en Raspberry Pi	5	5
T.3 Visualización de los datos en la web	T 3.1 Diseñar la base de datos para la web	10	10
	T 3.2 Iniciar del backend	20	40
	T 3.3 Implementar la creación de tablas visibles en la página web	20	30
	T 3.4 Iniciar del frontend	20	20
T 4 Realización del login en la web y gestión de los usuarios	T 4.1 Realizar utilidad para formularios emergentes	20	20
	T 4.2 Crear de los microservicios	5	5
	T 4.3 Creación de las llamadas en frontend	5	5
	T 4.4 Gestión de los usuarios	10	10
T.5 Interacción con maquinaria remota	T 5.1 Hacer la interfaz para que el usuario pueda manejar las salidas de Arduino	10	10
T.6 Mostrar el historial de eventos y datos con sus filtros	T 6.1 Realizar la aceptación de intervalos de variables en los microservicios	5	5
	T 6.2 Realizar los filtros para recibir rangos de fechas en el frontend	5	5
	T 6.3 Mostrar gráficos de los datos requeridos	10	10
T.7 Realización simulaciones con muchos lugares remotos donde se toman datos	T 7.1 Simular las llamadas	10	5
	T 7.2 Realizar un historial de acciones	10	10
	T 7.3 Realizar un historial de alarmas	10	15
T.8 Valoración y mejora de la experiencia de usuario	T 8.1 Internacionalizar la aplicación web	15	15
	T 8.2 Probar varios tipos de usuario	5	5
	T 8.3 Realizar un mapa con posición de los lugares	10	10
T.9 Personalización de sensores y alarmas	Personalizar el tipo de cada sensor.	15	0
	Personalizar las alarmas.	15	0
TOTAL		300	300

### 3.3 CONCLUSIONES

Al trazar el plan del desarrollo del sistema, el principal objetivo fue que desde una aplicación web se pudiese visualizar el historial de los valores tomados en diferentes lugares remotos, además de poder interactuar en los interruptores ubicados en esos lugares. El objetivo principal se ha cumplido, dado que se puede visionar en cada lugar los últimos valores de los sensores que hay, su historial y se pueden modificar los valores de las salidas como si fueran interruptores.

Se ha conseguido hacer muchas opciones para los usuarios, como un control de permisos para usuarios que tengan a cargo unos determinados lugares, trabajadores y administradores. También hay otras opciones como la de ver la posición de los lugares en el mapa, ver el historial de los valores de los sensores y ver el historial de quienes han manipulado las salidas de un determinado lugar. Otros objetivos conseguidos han sido recuperar los datos que no se han podido enviar por caídas temporales en internet, que múltiples usuarios puedan usar el sistema a la vez con múltiples lugares y llevar un historial de alarmas básico en el que se muestre cuándo unos determinados valores de unos tipos de sensores están dentro de un rango de valores peligroso.

El rendimiento de la aplicación web es bueno en general, ya que en la mayoría de opciones la respuesta es casi inmediata. Un caso especial es la lectura de los datos históricos de los sensores. Como hay que leer bastantes datos a la vez de un histórico que puede llegar a ser muy grande, el tiempo de respuesta llega a ser bastante apreciable. Se ha comprobado el correcto funcionamiento en una situación de estrés, haciendo una simulación de varios lugares con sensores que envían datos a la vez. Sin embargo, no se ha llegado a simular un gran número de recopilaciones de datos, así la carga de trabajo en un entorno de producción real podría ser mayor.

Debido al retraso producido en uno de los sprint, no se ha dado la posibilidad al usuario de poder personalizar más sus lugares como se había pretendido en un principio. La personalización pretendida era dar la capacidad al usuario de configurar los sensores que pudiera tener en cada lugar del que fuera responsable. Aunque con las opciones implementadas, el sistema se puede utilizar en muchos casos de uso diferentes debido a los múltiples tipos de sensores y hay total libertad de utilizar las salidas como estime cada usuario en cada lugar. También se tenía pensado dar la opción al usuario de personalizar los rangos de valores en los que las alarmas se generan, pero debido al retraso no se ha podido.

Las limitaciones de tiempo y presupuesto han supuesto que el alcance del proyecto no incluyese la puesta en producción de la aplicación web. Este alcance habría incluido demasiados pasos: generar imágenes para ser arrancadas en contenedores de docker en un servidor, además de otras configuraciones. Estas tareas suelen encargarse a otros responsables que no suelen ser los desarrolladores. Aunque los desarrolladores suelen hacer estos pasos una vez que están lo bastante automatizados. Hay otras opciones más baratas que se podían haber usado, pero esas opciones también estaban fuera del alcance de este proyecto. Una de estas opciones es la de conectar el servidor que estuviera en una red local con las Raspberries mediante conexiones de VPN. Otra opción que no se ha probado y que también podría ser suficiente para un particular, es que el servidor se ejecutase en el mismo microordenador que estuviese recopilando los datos y que, por lo tanto, no fuese necesaria una conexión a internet. En este caso el usuario podría entrar a la aplicación web, desplazándose al lugar donde está el microordenador y conectándose simplemente por wifi a un router donde estuviera conectado el propio microordenador. Las modificaciones y variaciones de este sistema son muchas y pueden servir a muchos propósitos diferentes.

Desde el punto de vista personal he ideado este proyecto para combinar múltiples tecnologías muy diferentes y que además se pueden complementar entre sí. También he combinado diferentes tipos de hardware, empezando por la simple Arduino, continuando con Raspberry Pi, luego el funcionamiento de un servidor centralizado y terminando por el funcionamiento del navegador del usuario. También he combinado múltiples lenguajes de programación: desde C++ hasta Typescript, pasando por JavaScript y otros tipos de lenguajes como HTML, SCSS y SQL. También se ha estructurado todo en la forma de un programa distribuido, por lo que también se han utilizado varias tecnologías para combinar los elementos. A pesar de no haber completado toda la planificación, la gran mayoría de objetivos han sido desarrollados, por lo que estoy satisfecho con el proyecto. Mención especial se merece la interfaz de usuario, es bastante completo y funciona con un buen rendimiento.

### 3.4 BIBLIOGRAFÍA

Para la realización del proyecto se han utilizado una gran cantidad de tecnologías diferentes. Internet ha sido la gran fuente de recopilación de documentación sobre las tecnologías. Este es un recopilatorio de cada parte

Ámbito	Tecnología	Fuente
Web parte de frontend	Angular	<a href="https://material.angular.io/">https://material.angular.io/</a> : página de documentación de los controles de Angular para la web
	OpenLayers	<a href="https://openlayers.org/">https://openlayers.org/</a> : página oficial de la librería para visualizar mapas en la web. Contiene ejemplos y documentación
Web parte de backend	Node.js	<a href="https://nodejs.dev">https://nodejs.dev</a> : la página oficial de node.js donde también hay documentación
	Express	<a href="https://expressjs.com/">https://expressjs.com/</a> : página oficial de la librería de node.js usada para implementar los microservicios
	Sequelize	<a href="https://sequelize.org/">https://sequelize.org/</a> : página oficial del ORM para JavaScript
	Otras librerías para Node.js	<a href="https://www.npmjs.com/">https://www.npmjs.com/</a> : página oficial de las distintas librerías para node
Base de datos	PostgreSQL	<a href="https://www.postgresql.org/">https://www.postgresql.org/</a> : página oficial del Sistema Gestor de Bases de datos de código abierto llamado PostgreSQL. Esto ha sido de utilidad para instalar y configurar este sistema en el ordenador de desarrollo, la Raspberry Pi y obtener documentación para manejar las llamadas a esta base de datos. También tiene documentación para realizar la conexión y llamadas desde el lenguaje C con la librería libpq.
Lenguaje C usado desde la Raspberry Pi	Realización de llamadas HTTP	<a href="https://curl.se/">https://curl.se/</a> : página oficial de la librería para hacer llamadas HTTP. Tiene documentación para su uso en el lenguaje C.
	Comunicación con Arduino	<a href="https://forum.arduino.cc/">https://forum.arduino.cc/</a> : contiene ejemplos para la comunicación serie entre Arduino y un ordenador con linux con las funciones POSIX desde la parte de la Raspberry y librerías de Arduino desde la parte de Arduino. En este caso será una Raspberry Pi con la distribución Raspian
Configuración y uso de Raspberry Pi	Instalación del sistema operativo. Diferentes configuraciones	<a href="https://www.raspberrypi.com/">https://www.raspberrypi.com/</a> : contiene documentación, foros y ejemplos. También se ha buscado información en todo tipo de foros para cada configuración
Lenguaje C usado desde Arduino	Manejo de entradas y salidas desde sus pines	<a href="https://www.arduino.cc/">https://www.arduino.cc/</a> : hay todo tipo de ejemplos y documentación. Para las funcionalidades básicas usadas ha sido más que suficiente

En las asignaturas estudiadas en el grado se han obtenido diferentes conocimientos usados en este proyecto de varias asignaturas:

Concepto	Asignaturas
Programación en lenguaje C++ y estructuración del código en clases	Programación orientada a objetos, Tecnología orientada a objetos
Diseño, desarrollo de bases de datos y manipulación de los datos	Bases de datos, Diseño de bases de datos. Programación de bases de datos.
Diseño de circuitos electrónicos básicos	Física
Configuración de la manipulación de programas en linux y uso de la API de POSIX	Sistemas Operativos
Comunicación entre dispositivos a través de la red	Redes de computadoras, Sistemas distribuidos, Seguridad
Fundamentos básicos en el desarrollo de aplicaciones web	Programación de aplicaciones web
Desarrollo de documentación	Bases de datos, Ingeniería del software, Diseño tecnológico de sistemas de información, Proyectos de informática, Taller transversal I, Profesión de ingeniero en informática, Taller transversal II

## ANEXO 1: TÉRMINOS Y DEFINICIONES POR ORDEN ALFABÉTICO

**Analógico:** se refiere a los niveles de tensión que pueden tomar un gran número de valores. Técnicamente infinitos pero en nuestro caso usaremos valores de 0 a 1023.

**BackEnd:** en aplicaciones web distribuidas se refiere a la parte que se ejecuta en el servidor. Actualmente, está compuesto por microservicios.

**Bit:** es un elemento que puede ser 0 o 1.

**Booleano:** un tipo que puede ser verdadero o false. Muchas veces se hace un equivalente con los ceros y unos de los bits

**BPS:** Bits Per Second, es el número de impulsos elementales que pueden ser 0 o 1 transmitidos por cada segundo. Esta medida es usada para medir la velocidad de una transmisión,

**C:** es un lenguaje de programación de propósito general. Es apreciado por la eficiencia de código que produce.

**C++:** es una extensión del lenguaje C que aporta del uso de clases y objetos en la programación.

**CPU:** Central Procesor Unit o Unidad Central de Procesamiento. Es uno de los componentes de los ordenadores y otros dispositivos programables. Es el encargado de ejecutar las instrucciones del programa

**CRUD:** es el acrónimo de “Crear, leer, actualizar y borrar”. Este término se suele utilizar para referirse a las operaciones básicas al tratamiento de datos en una base de datos.

**CSS:** siglas de Cascading Style Sheets u Hojas de Estilo en Cascada. Es un lenguaje de marcado para definir los estilos de visualización en una página web. De esta forma se definen los atributos de visualización de cada elemento en una página web.

**Checksum:** también llamada suma de verificación. Es un valor de redundancia para hacer una comprobación para detectar si todos los datos son correctos. De forma que este valor se saca como resultado de un conjunto de valores. En la transmisión de los valores se vuelve a generar otro checksum y se compara con el transmitido para saber si todos los datos han llegado correctamente.

**Compilación:** es el proceso de generar un ejecutable a partir del código fuente.

**Contraseña:** es un texto más o menos complicado y que solo sabe el correspondiente usuario para poder demostrar frente a un servidor su identidad.

**Cookies:** son pequeños archivos guardados en los navegadores por las páginas web. Estos pequeños archivos guardan datos usados por esas páginas web para múltiples motivos. En nuestro caso para guardar la sesión del usuario.

**Dashboard:** en inglés significa tablero de instrumentos. Es el nombre que se le suele dar al lugar donde el usuario encuentra todas las funcionalidades que necesita.

**Digital:** se refiere a una señal que puede tomar una cantidad limitada de valores. En nuestro caso solamente valor alto o valor bajo.

**Express:** es también llamado Express.js. Es un framework para Node.js de código abierto y licencia MIT. Se utiliza para desarrollar aplicaciones web y escuchas de llamadas HTTP, también llamados microservicios.

**Filtro:** es el componente en el que se puede configurar para poder ver unos registros en concreto.

**Formulario:** es el componente web donde se pueden introducir los datos de un determinado registro.

**Framework:** es una utilidad para programar que tiene como características de ahorrar mucho código, ya que aporta funcionalidades directamente al programa

**FrontEnd:** en una aplicación web es la parte que se ejecuta en el navegador del cliente. En nuestro caso, para implementarlo se usará el framework llamado Angular.

**GPIO:** General Purpose Input/Output también llamado Entradas/Salidas de propósito general. Consiste en patillas de un componente electrónico que pueden ser configuradas para propósitos generales mediante programación. En el caso de la Raspberry pi se refiere a las patillas que pueden ser configuradas como entrada o salida.

**HTML:** HyperText Markup Language o lenguaje marcado de hipertexto. Hace referencia al lenguaje de marcado para elaborar páginas web.

**HTTP:** HyperText Transfer Protocol. Es un protocolo de comunicación que permite transferencias de comunicación en el canal de internet. Es un protocolo sin estado, por lo que es necesario varias tecnologías para mantener los estados en las páginas web.

**HTTPS:** HyperText Transfer Protocol Secure o protocolo seguro de transferencia de hipertexto. Está basado en el protocolo HTTP y es su versión segura. Esta seguridad se consigue mediante un cifrado basado en SSL/TLS. De esta forma se consigue que la información sensible como claves, contraseñas y otros datos puedan circular sin ser usados por un atacante.

**JSON:** JavaScript Object Notation o notación de objeto JavaScript. Es un formato de texto sencillo para el intercambio de datos. Es usado nativamente por JavaScript y es el formato más común para transferir datos en las páginas web.

**Java:** es un lenguaje de programación usado con múltiples propósitos. Uno de ellos es realizar microservicios que pueden ser utilizados por aplicaciones web.

**JavaScript:** es un lenguaje de programación creado para ser ejecutado en navegadores web. Es abreviado muchas veces como JS. Es un lenguaje interpretado, por lo que puede ser usado independientemente del sistema operativo. También puede ser usado en el lado del servidor en las aplicaciones web.

**KB:** es el acrónimo de KiloBytes.

**KiloBytes:** se refiere a una unidad de cantidad de almacenamiento. Esta cantidad equivale a 1024 bytes dado que los múltiplos son binarios. Un ordenador hoy en día tiene millones de KiloBytes de memoria y varios gigas de almacenamiento.

**Licencia BSD:** es una licencia de software permisiva como la licencia MIT. También permite usar software con esta licencia en software comercial.

**Servidor:** es el ordenador o conjunto de ordenadores a los que se conectan los clientes para acceder al sistema.

**MIT:** re refiere a una licencia de software permisiva. Por lo tanto, está permitido utilizar software con esta licencia en software propietario.

**Microllamadas:** son las llamadas HTTP que se hacen desde los llamados lugares del sistema distribuido. Estas llamadas se realizan para aportar o recibir datos u otras cosas.

**Microservicios:** son los elementos que escuchan en una determinada dirección de red con un puerto para recibir y dar información.

**Middleware:** es conocido como la lógica de intercambio de información entre aplicaciones. En nuestro caso se refiere concretamente a las zonas de código por las que pasan los datos de una comunicación entre el cliente y el servidor.

**Node.js:** es un entorno en tiempo de ejecución multiplataforma, de código abierto para la capa del servidor. Está basado en el lenguaje de programación JavaScript. Fue creado para ser útil en programas de red altamente escalables. Ese es el caso en el que lo usamos, aunque también se puede usar para otras cosas.

**ORM:** en nuestro caso son las siglas de Object Relational Mapping. Es una técnica para convertir datos entre un sistema de tipos utilizando un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional como motor de persistencia. El objetivo de un sistema ORM es pasar los datos de un sistema a otro con poca cantidad de código y de forma más o menos automatizada. Otra característica de estos sistemas es poder cambiar el SGBD de un programa sin cambios en el código, ya que es el propio ORM el que se encarga de ello.

**Output:** se refiere a la patilla que puede encender un aparato externo por decisión del usuario.

**PHP:** es un lenguaje de programación interpretado del lado del servidor y de uso general que se adapta especialmente al desarrollo web.

**Patilla:** es un conector electrónico que puede ser conectado.

**Persistir:** es la acción de guardar datos en la base de datos.

**Prueba de estrés:** se refiere a una simulación de la cantidad de procesos que tendría el sistema cuando estuviera funcionando en producción normalmente.

**RAM:** Random Access Memory o memoria de acceso aleatorio. Es una memoria de corto plazo usado por los dispositivos electrónicos para almacenar datos que van a ser usados en poco tiempo y que se suelen perder al apagarse.

**SCSS:** es la extensión de Sass. Sass es un lenguaje de hojas de estilos en cascada para usar un lenguaje de script simple para generar hojas de estilo en cascada o CSS. Esto es usado por los desarrolladores web para simplificar y flexibilizar la creación de archivos CSS.

**SGBD:** se refiere a un Sistema Gestor de Bases de Datos. Es el software que permite administrar una base de datos. Hoy en día hay muchos gestores de bases de datos de múltiples fabricantes y licencias.

**SQL:** Structured Query Language o lenguaje de consulta estructurada. Es un lenguaje para administrar y recuperar información de sistemas de gestión de bases de datos relacionales. Está basado en el álgebra y el cálculo relacional. Es el lenguaje más común para manipular bases de datos.

**Scroll:** es el movimiento que se puede hacer en una página web cuando la página es más grande que la pantalla.

**Sequelize:** es un ORM para ser usado en JavaScript desde el lado del servidor.

**Sesión:** en una página web se refiere a una sesión HTTP. Las cuales asocian información con visitantes individuales.

**Singleton:** es un patrón de diseño en el software

**Software:** es la parte lógica del sistema que manda instrucciones a la parte física llamada hardware para que sea posible su funcionamiento.

**Stock:** se refiere al número de existencias disponibles en el mercado.

**String:** es el tipo de dato en lenguajes de programación que representa una lista de letras.

**TX/RX:** es un protocolo para la transmisión de datos en telecomunicaciones.

**Tensión:** es la diferencia de potencial entre dos puntos. En los aparatos electrónicos es una medida eléctrica que se debe cumplir para que un sistema electrónico funcione.

**Tipado:** en un lenguaje de programación es una característica de las variables. Si una variable está tipada el valor que contiene debe tener unas características y si no está tipada su valor puede tener cualquier característica. Una variable tipada suele generar más errores de compilación, ya que el compilador siempre sabe el contenido que se le puede asignar, mientras que una variable no tipada suele generar más errores de ejecución, ya que su contenido puede no adecuarse a la operación a realizar. Otra diferencia es que definir tipos para las variables suele requerir más código. Por lo tanto, cada caso tiene sus ventajas e inconvenientes.

**Token:** es una cadena de caracteres que tiene un significado. En aplicaciones web distribuidas se suelen usar tokens de seguridad para la autenticación de usuarios.

**Topología:** es la rama de la matemática dedicada a las propiedades de los cuerpos geométricos. En los sistemas distribuidos en red se refiere a la estructura geométrica de los componentes del sistema y su relación entre sí.

**TypeScript:** es un lenguaje de programación libre y de código abierto. Es un superconjunto de JavaScript que añade tipos estáticos y objetos basados en clases. Es una forma de usar JavaScript definiendo tipos a las variables. Normalmente, los códigos escritos en TypeScript suelen ser convertidos a JavaScript antes de ser ejecutados o distribuidos.

**URL:** Uniform Resource Locator. Es una secuencia de caracteres que se usa para nombrar o identificar un recurso en internet como puede ser una imagen o una página web.

**USB:** Universal Serie Bus. Es un estándar de un bus de comunicaciones que define los cables. En el proyecto es usado para comunicar la Raspberry Pi con el Arduino.

**Valores booleanos:** en la lógica booleana usada en programación son los dos valores que puede tomar una variable de tipo booleano. Estos dos valores se suelen representar como verdadero y falso.

**Ventana emergente:** en inglés es llamado pop-up. Es una ventana que aparece de manera superpuesta sobre las demás. Es usada para hacer una determinada operación para luego cerrarse.

**Voltio:** es la unidad de medida del potencial eléctrico. En nuestro caso es el nivel de tensión necesario para que los componentes electrónicos tengan energía para funcionar. También es la medida que usan las entradas y las salidas.

**WIFI:** es una tecnología que permite la interconexión inalámbrica de dispositivos electrónicos.

## ANEXO 2: APORTACIÓN DE LOS DIFERENTES ENPOINTS

Esta es la lista de endpoints el backend. Hay que tener en cuenta que los endpoints de tipo get se le pueden pasar parámetros para filtrar unas determinadas filas. También hay que tener en cuenta que no solamente pueden llegar llamadas desde el frontend, también de los microordenadores que están en sus correspondientes lugares. Por lo tanto, ha sido necesario modificar el sistema de seguridad para diferenciar entre unos y otros, ya que su autenticación depende de tablas de la base de datos diferentes.

Tabla acciones, esta tiene un endpoint para un caso especial:

- **Get api/actions/:id\_action**: retorna los datos de una acción por su id
- **Get api/actions/place/place**: retorna la última acción por un determinado lugar, está configurado para que este endpoint sea llamado por la Raspberry de un determinado lugar para preguntar el estado de las salidas que debe tomar. Por lo tanto esta llamada recibe los datos de autenticación de la propia Raspberry
- **Get api/actions**: retorna todas las acciones de la base de datos
- **Path api/actions/:id\_action**: modifica una acción de la base de datos
- **Delete api/actions/:id\_action**: borra una acción mediante su id
- **Post api/actions**: crea una nueva acción con los datos que se le pasan. Retorna los datos que se guardaron en la base de datos

Tabla alarmas: esta tabla no tiene todos los endpoints de un CRUD completo, ya que las alarmas son generadas por el propio backend y tampoco se pueden eliminar. Solo se pueden ser consultadas y modificadas. Estos son los endpoints disponibles:

- **Get api/alarms/:id**: retorna todas las alarmas de la base de datos
- **Get api/alarms**: retorna todas las alarmas de la base de datos
- **Path api/alarms/:id**: modifica una determinada fila

Tabla mediciones, esta es otro caso un poco especial, ya que está pensada para hacer filtrados en todas las llamadas por que su número de filas es muy grande. Tampoco tiene endpoints por un CRUD normal, ya que las mediciones son creadas por los lugares y esto tiene unas características especiales. Tampoco se pueden eliminar mediante un endpoint. Estos son los endpoints disponibles:

- **Get api/measurements**: está pensado para usar los filtros con los siguientes parámetros: ?placeId=1&date\_timeFINISH=2023-01-21T17:33:20.007Z&date\_timeBEGIN=2023-01-16T22:33:10.007Z&date\_timeORDERDESC=0 &LIMIT=100000. Hay que recordar que en el caso de que el número de filas a retornar sea demasiado alto se limitan automáticamente.
- **Get api/measurements/multiple**: retorna las mediciones de los lugares que cumplen la condición de estar relacionados con el usuario que hace la llamada. En caso de que el usuario sea el administrador, se retornan todos las mediciones que cumplen con el filtro de los parámetros que se le pasan en la llamada.
- **Post api/measurements/multiple**: es otro endpoint para ser usado por una Raspberry que usará sus propias credenciales. Inserta varias mediciones en el mapa. Insertar varias mediciones en una sola llamada no cumple con el estándar de las llamadas de HTML, pero es mucho más óptimo, sobre todo a la hora de mandar muchas mediciones que se han retrasado en el tiempo.

Tabla outputs: están los tipos de salidas. Actualmente, se han limitado las salidas en cada lugar a cuatro, así que actualmente solo se usa una lectura que podría haber sido reemplazada a una enumeración. No se ha hecho más funcionalidad con esta tabla por falta de tiempo y se ha dejado para posibles implementaciones futuras:

- **Get api/outputs**: retorna los tipos de salida que tiene cada lugar.

Tabla tipo de sensor: los tipos de sensores están predefinidos y, por lo tanto, la única funcionalidad es la de leer esta tabla. Por lo tanto este es el único endpoint:

- **Get api/sensors**: lee los tipos de sensor que tiene cada lugar

Tabla personas: esta es una tabla además de tener la funcionalidad CRUD para manipular sus filas, también tiene un endpoint para retornar un token al frontend que se usará posteriormente en los otros endpoints como identificador de sesión. Por seguridad, con este token se puede identificar al usuario y este caducará en un tiempo predeterminado.

- **Get api/persons**: retorna los datos de las personas. No se incluye el valor referente a la contraseña
- **Get api/persons/:id**: retorna los datos de la persona con ese identificador. No se incluye el valor referente a la contraseña

- **Path api/persons/:id:** modifica los datos de esa persona
- **Delete api/persons/:id:** elimina la fila de la persona con ese identificador
- **Post api/persons/authenticate:** recibe nombre de usuario y contraseña y los comprueba con la base de datos. Primero busca el usuario con ese nombre y si lo encuentra compara la contraseña usando la librería bcrypt, ya que en la base de datos solo se guarda el hash de la contraseña. En el caso de que este todo correcto se genera un token y se envía en la respuesta para que el front lo use posteriormente en las siguientes llamadas en los endpoints que necesitan token

Tabla lugares: además de los endpoints habituales para el CRUD de la tabla, se usa otro endpoint para generar un token para las llamadas realizadas por los lugares, teniendo un funcionamiento similar a los usuarios.

- **Get api/places:** retorna los datos de los lugares
- **Get api/places/:id:** retorna los datos del lugar que tiene ese identificador
- **Path api/places/:id:** cambia los datos de la fila del lugar con ese identificador
- **Delete: api/places/:id:** elimina la fila del lugar con ese identificador
- **Post api/places:** crea un nuevo lugar con los datos que se le pasan en el body. En la respuesta se retornan los datos guardados con el identificador de la fila generada por la base de datos.
- **Post api/places/authenticate:** se le pasa un identificador y contraseña del lugar para autenticar al microordenador que está en ese lugar. Por lo tanto, el identificador y la contraseña deben estar guardados tanto en el microordenador como en el servidor y estos deben de coincidir.
- **Get api/places/actualization/actualization:** este endpoint es para ser llamado desde el microordenador de un lugar, por lo tanto requiere un token generado para un lugar que esté en la base de datos. Este endpoint retorna los datos grabados en el servidor de ese lugar. El microordenador usa estos datos para diferentes configuraciones