

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 esta el problema de la brecha digital es especialmente problemática en los entornos rurales.

Pensando en esta situación se han ideado los objetivos de realizar el diseño, implementación y prueba de un sistema capaz de las siguientes acciones:

- Centralizar la recogida de datos en distintos lugares de la geografía y tener estos datos centralizados
- Realizar acciones en esos lugares como forma de retroalimentación de los datos recibidos
- Permitir que se sigan recogiendo los datos aunque la red este caída de forma temporal
- Sea fácil de manejar por las personas que tengan pocas habilidades tecnológicas y pocos recursos para invertir en tecnología

Para cumplir con estos objetivos primarios sera necesario administrar una base de datos distribuida. Esta base de datos debe contar con una central que puede estar replicada todos los datos y además en cada lugar estén 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 mandados a la central cuando la conexión sea restablecida.

También se debe cumplir que los componentes físicos del sistema deben de poderse conseguirse fácilmente, que sean 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 parte debe estar aislada lo mas posible del resto y las tecnologías usadas para realizar el código fuente no deben de 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 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 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 podrán acceder a todos los datos de 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 mas actuales y además el historial de los datos. Si el proyecto va bien de tiempo se podrán crear mejores gráficas para mostrar los historiales 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 examinar mas 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.

Dadas estas características se ha decidido que la visualización y manipulación de los datos y la maquinaria se realizara 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.

Lista de requisitos funcionales:

- 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 además del 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 y otros dentro de la aplicación. Los roles serán los siguientes: de usuario, de operario y de administrador.
- El rol de usuario tiene los permisos para visualizar los datos de sus propios datos generados por sus sensores en los lugares que tiene asignados.
- El rol de operario tiene los permisos para visualizar los datos de todos los usuarios. Esto tiene el fin de que los datos de todos los lugares estén vigilados y gestionados. Esto puede ser de vital importancia por que puede haber usuarios que no controlas debidamente sus lugares
- El rol de administrador además tiene permisos para gestionar los usuarios. De esta forma no tiene restricciones en las cosas que puede hacer la aplicación. Este es el principal responsable de la aplicación aparte de los desarrolladores.

ESTUDIO DE LAS POSIBLES TECNOLOGÍAS

TECNOLOGÍAS PARA LA WEB: Para elegir las tecnologías de este proyecto primero hay que diferenciar las diferentes partes que lo van a componer. En parte de la aplicación web hay tres capas. El modelo, la vista y el controlador. El modo de diferenciarlos suele depender de la tecnología usada. Actualmente es muy común la tecnología de los micro servicios. Esta es actualmente la forma mas usada 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 ademas desacoplar unas partes de otras y además desacoplar las diferentes capas.

Otras estructuras anteriormente usadas en las aplicaciones web, es la de generar el código HTML directamente desde el código ejecutado en el servidor como hacía PHP y otras tecnologías. Actualmente se usan frameworks que consumen código y generan código JavasCript para ser ejecutado en el navegador. Por lo tanto la aplicación web se suele componer de alguno de estos frameworks para la capa de la vista que se ejecuta en el navegador en su mayor parte y la parte que se ejecuta enteramente en el servidor que usa microservicios para la comunicación. Además estos frameworks contienen una buena cantidad de componentes que permiten hacer cosas como simular ventanas emergentes en todo tipo de navegadores. Hacer estas cosas desde HTML básico y además que ese código funcionase en todos los navegadores actuales sería terriblemente largo y tedioso.

Por lo tanto se escogerá este tipo de estructura: la de los microservicios. Una vez tomada esta decisión queda decidir que framework se usará para la parte frontal en el navegador y que tecnologías se usarán para generar los microservicios.

Para la parte frontal del navegador estas son algunas de los frameworks mas populares: Angular, React y Vue:

	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

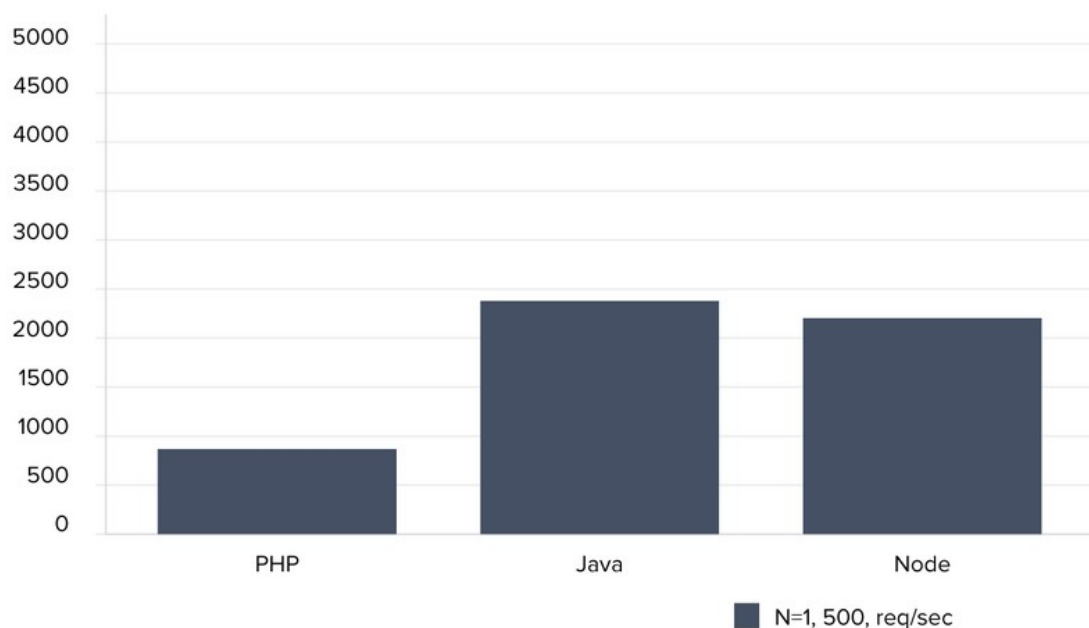
Por lo tanto la opción escogida es Angular por la mayor cantidad de características que hacen prescindir de la necesidad de la dependencia de terceros.

TECNOLOGÍAS PARA MICROSERVICIOS: En el caso de la generación de microservicios también hay que escoger tecnologías. En este caso unas buenas opciones de estudio son Node.js, Java y PHP. Esta comparación tiene que ser pensada exclusivamente en el tratamiento de microservicios y no en las otras capacidades que tiene cada uno de estos lenguajes para otras cosas.

En esta tabla se pueden ver algunas diferencias de estos lenguajes. Para tener esta tabla clara hay que diferenciar entre hilos y procesos. Un proceso es la instancia de un programa en ejecución. Es administrado por el sistema operativo, si un proceso se queda esperando a una operación de entrada/salida como un acceso a una base de datos, fichero o a una red y en ese intervalo de tiempo esta ocupando recursos de memoria y de procesados. En cambio un solo proceso 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 un único proceso de programa, mientras que en PHP se genera un nuevo proceso consumiendo mas recursos por cada llamada.

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. 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 gran alta carga 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 y eso es una desventaja en unas ocasiones como programas complejos 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 mas 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 esto nos facilitará mas las cosas. Por lo tanto el lenguaje elegido para realizar los microservicios sera JavaScript.

Dentro de JavaScript en el lado del servidor hay que elegir tecnologías para implementar los microservicios y acceder a las bases de datos. Además hay que escoger alguna tecnología para manejar la base de datos que use la plataforma web para persistir los datos. La base de datos de la página será la misma que se use como central en el proyecto. Se ha decidido por el conocimiento previo de las tecnologías 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.

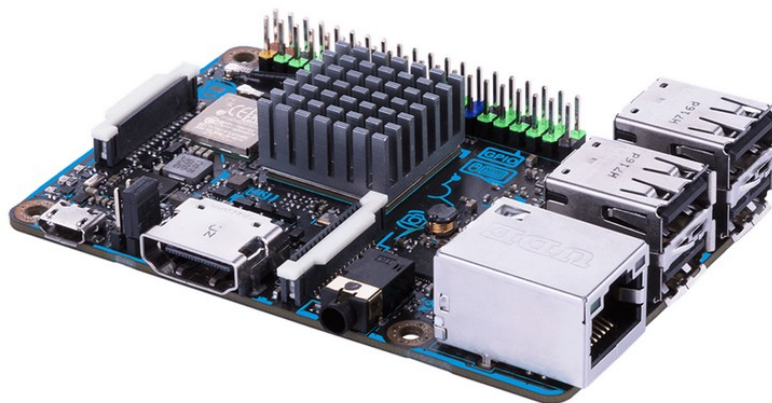
TECNOLOGÍAS PARA RECOPIRAR LOS DATOS: como se ha comentado antes este va a ser un sistema distribuido y los datos se leerán en un lugar remoto y como en ese lugar la conexión puede fallar potencialmente los datos se guardarán en el lugar donde sean tomados. Por lo tanto necesitamos un hardware que sea capaz de mandar y recibir datos de la red, almacenar datos de forma relativamente fiable y de 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 ejecutar Node.js de forma eficiente para usar el mismo lenguaje en las comunicaciones con los microservicios. Estas son las alternativas:

Raspberry Pi: es el micro ordenador actual mas famoso y con mas soporte con mucha diferencia. Además el precio recomendado es uno de los mas 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 mas recomendable. En el aspecto negativo esta la actual disponibilidad: debido a la falta de chips su precio se ha multiplicado y además por la falta de stock es difícil conseguir una. 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 muchas de sus competidoras.



ASUS Tinker Board S: es una alternativa mas cara y potente. Para este proyecto no será necesaria esa potencia extra pero es una opción a tener en cuenta. 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.

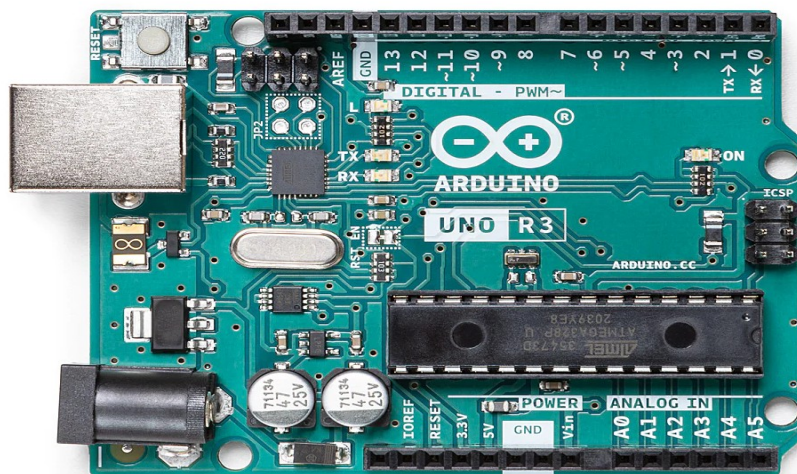


Orange Pi: una alternativa muy similar a Raspberry pi en características y a un precio muy competitivo. También se puede prescindir de tarjeta SSD y que sería su sucesora mas inmediata.



Como se puede ver en las imágenes los micro ordenadores tienen patillas que no suelen tener un ordenador normal. Estas patillas se pueden configurar como entradas o salidas digitales. Esto es una opción muy buena y 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 mas aconsejable para un proyecto grande: usar otro componente intermedio a los sensores para proteger al micro ordenador de sobre voltajes y no complicar el montaje con muchos componentes diferentes.

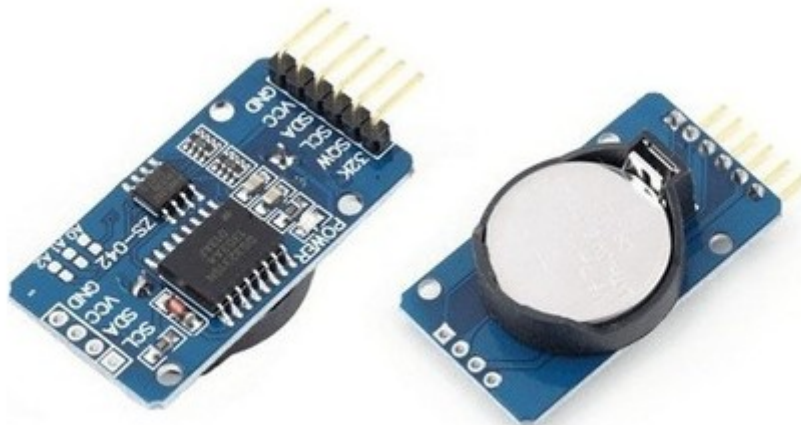
Arduino: es una pequeña placa barata y fácil de encontrar y si es necesario de reemplazar. Tiene múltiples formatos que se pueden escoger según las necesidades del proyecto. El modelo mas común y el que usaremos en el proyecto es el de la foto, el Arduino Uno:



Hay varios motivos para usar esta placa como ampliación de un micro ordenador con entradas y salidas integradas. Además de contar directamente con entradas y salidas analógicas es mas barato y funciona a una frecuencia muchísimo mas baja que un micro ordenador. Estas dos características hacen que al sufrir sobre tensiones el reemplazo de hardware sea mas 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 sobre calentarse.

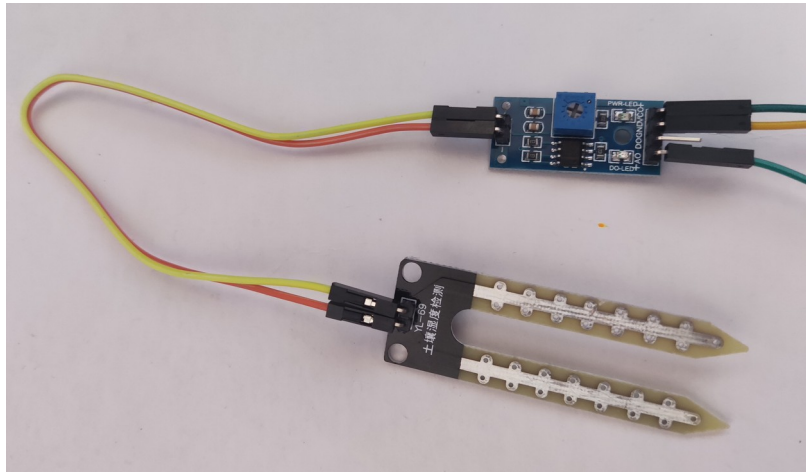
OTROS COMPONENTES Y SENSORES

En el caso 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 micro ordenador se distribuye sin ningún tipo de batería no tiene forma de conservar la hora si es apagada y no esta conectada a alguna red de donde saque la hora actual. Para esto se utilizará el módulo DS1307. Este módulo se conecta directamente a la Raspberry pi y con la correspondiente configuración es usado para mantener la hora en el caso de apagado y falta de red para escanear la hora en ella:

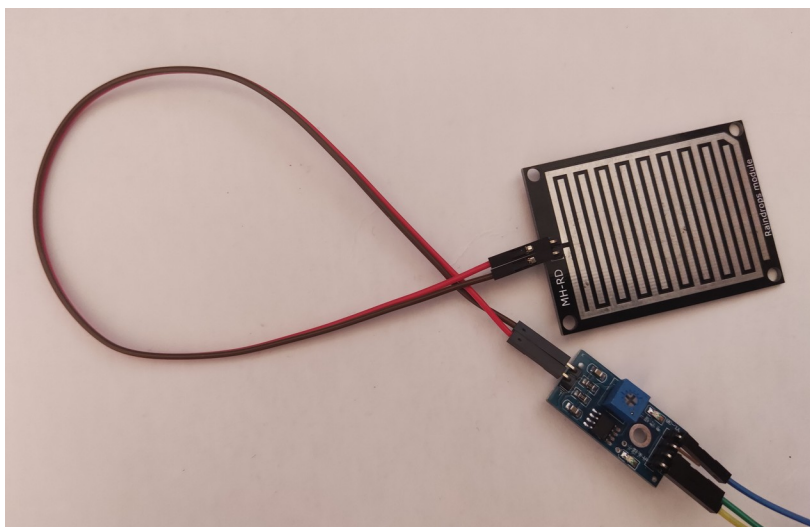


En el caso de Arduino: se usarán un buen número de sensores y una placa de relés para mandar salidas al exterior. A excepción de uno las mediciones realizadas por el Arduino son simplemente lecturas analógicas o digitales y se almacenarán como tales. En el caso del sensor de humedad y temperatura se usara una librería externa. Pero esto no será un problema ya que esta distribuida bajo licencia MIT.

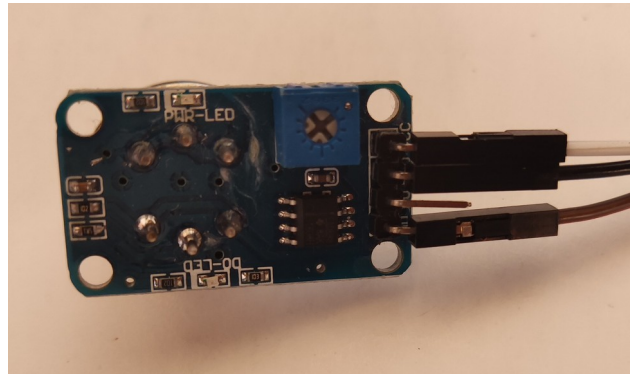
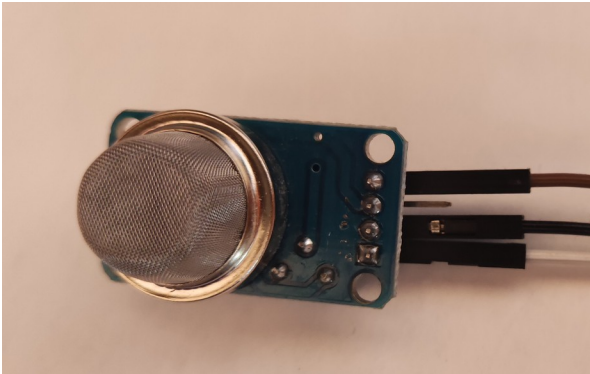
Sensor de aceite: es bastante simple: consta de dos conductores conectados a un regulador electrónico el cual retorna una salida analógica que indica la resistencia que hay entre los dos conductores y una salida digital que manda un pulso alto o bajo si la resistencia sobrepasa un cierto límite que se 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.



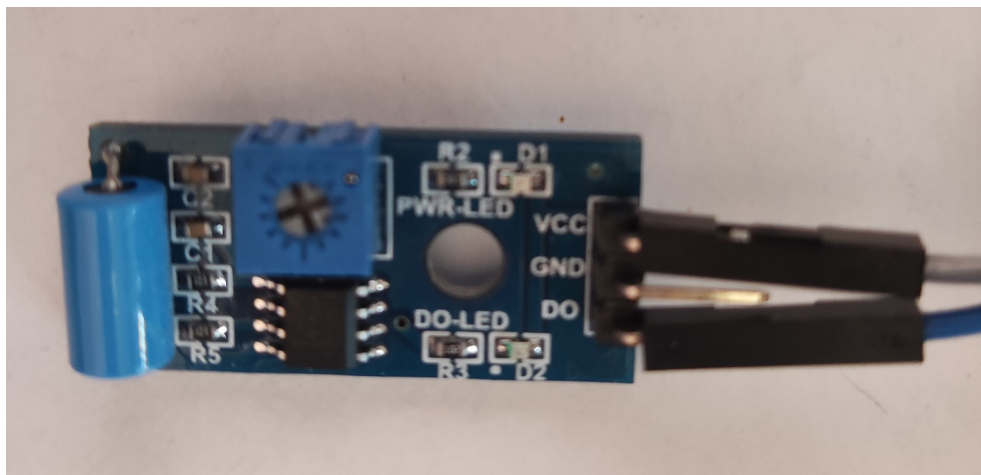
Sensor de lluvia: muy similar al anterior. La diferencia esta en la forma de los dos conductores. Con esta forma la resistencia entre los conductores baja cuando caen gotas en el sensor ya que dada la forma cada gota hace contacto en los dos conductores a la vez.



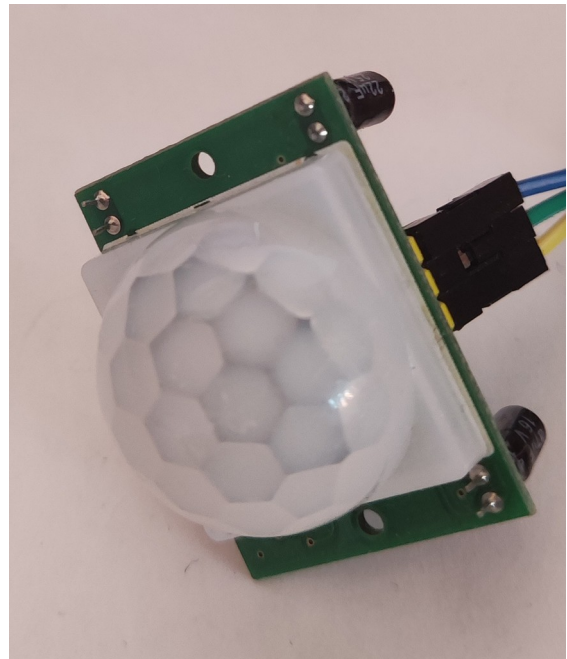
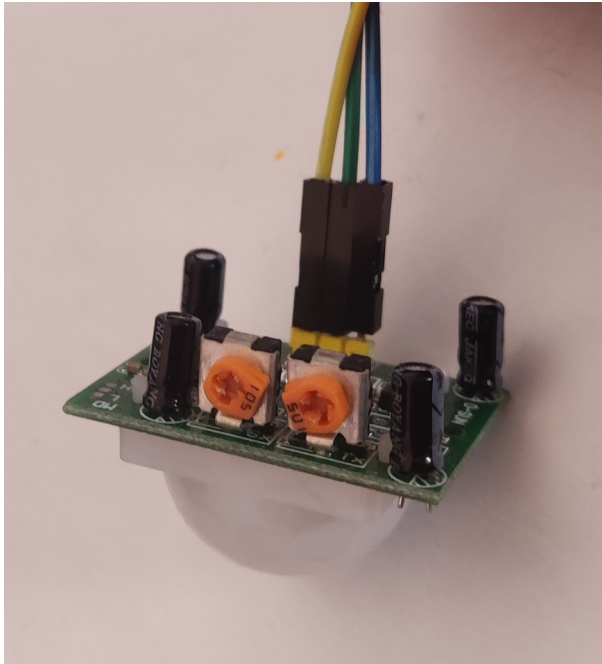
Sensor de gases inflamables MQ-2: este sensor detecta los típicos gases inflamables usados por mecheros o cocinas de gas. Las sustancias detectadas son metano, butano, GLP o humo. Sus salidas son similares a las salidas de los sensores anteriores, por lo tanto lleva un potenciómetro para regular el nivel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



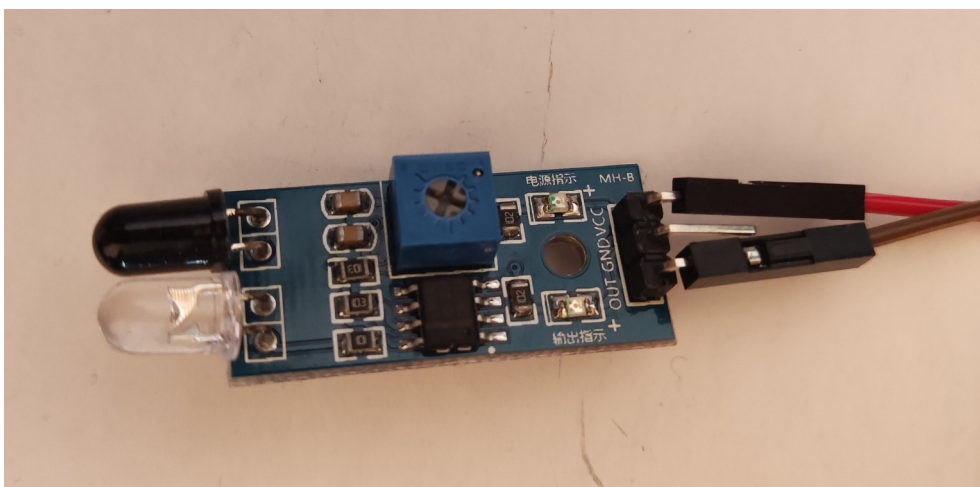
Sensor de vibración: se activa en el caso de que sea agitado con fuerza. En este caso solamente tiene una salida digital. La sensibilidad de esta salida esta regulada por el potenciómetro de la imagen. Por lo tanto de este solo se podrán guardar valores analógicos:



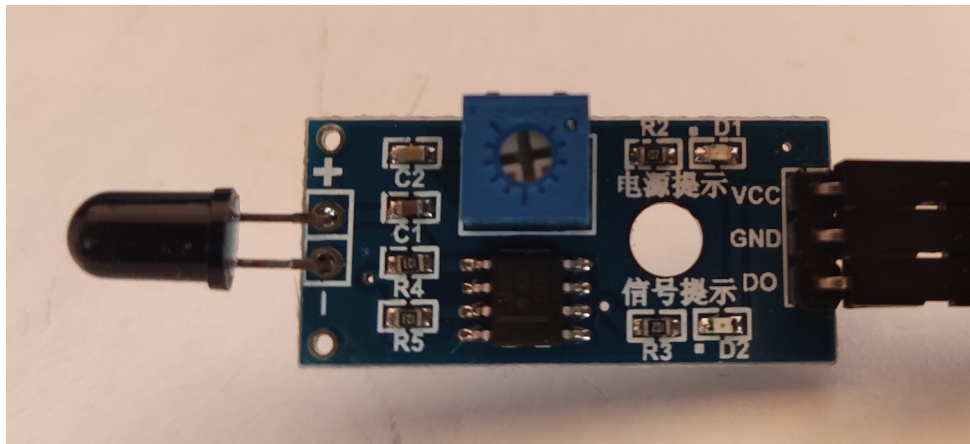
Sensor de personas HC-SR501: detecta personas o animales mediante infrarrojos. Se activa cuando alguien se mueve en la franja donde es sensible. Puede detectar movimiento entre 3 y 7 metros de distancia. Los dos potenciómetros que lleva son para regular la sensibilidad de detección de entre 3 a 7 metros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



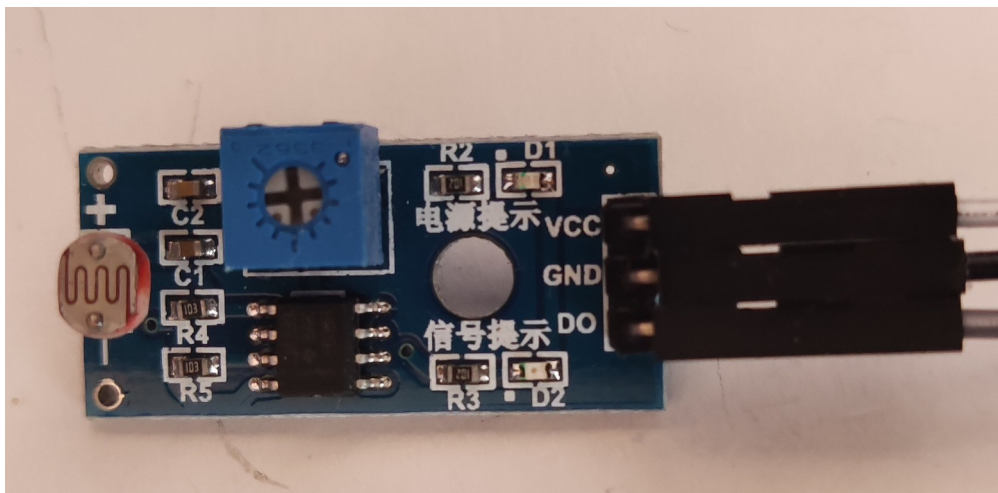
Sensor de obstáculos IR: consiste en dos diodos: uno emisor de luz y otro receptor. El emisor emite una luz de infrarrojos que no puede ver el ojo humano que es reflejada si hay algo justo enfrente y entonces el receptor la recibe y se activa. El nivel de sensibilidad es ajustado desde el potenciómetro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



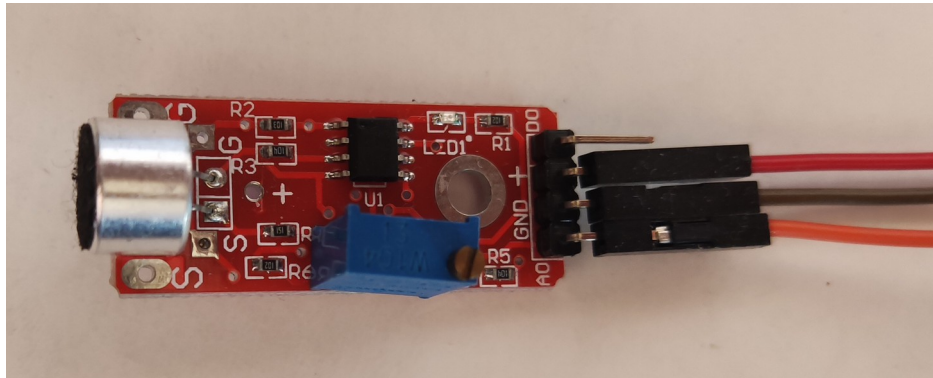
Sensor de llama por infrarrojos para incendios: detecta la luz de las llamas y se activa cuando tiene una llama cerca. Solo tiene una salida digital cuya sensibilidad es regulada por un potenciómetro.



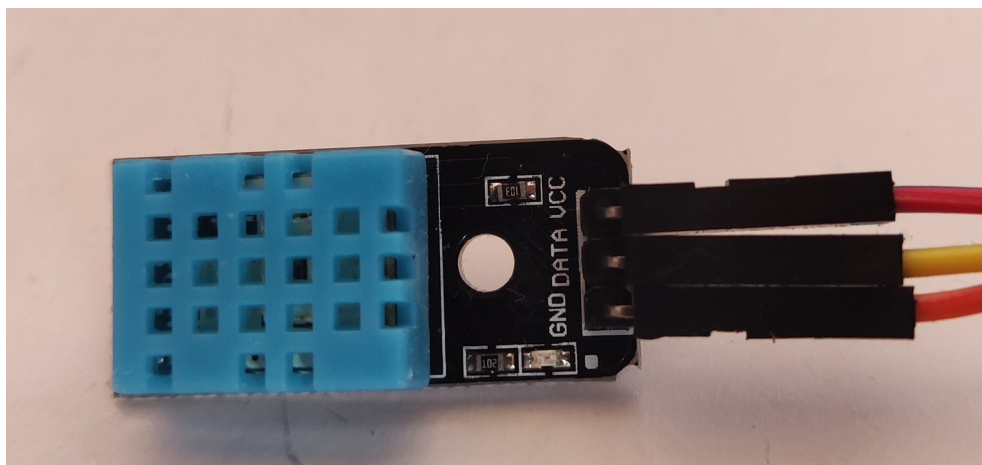
Sensor de luz para el día y la noche: usa un foto resistor para detectar si hay luz del sol o de luces visibles por las personas. Tiene una señal digital que puede ser regulada por el potenciómetro. Hay que tener en cuenta que es muy importante que el sensor este orientado a la luz que se quiera detectar.



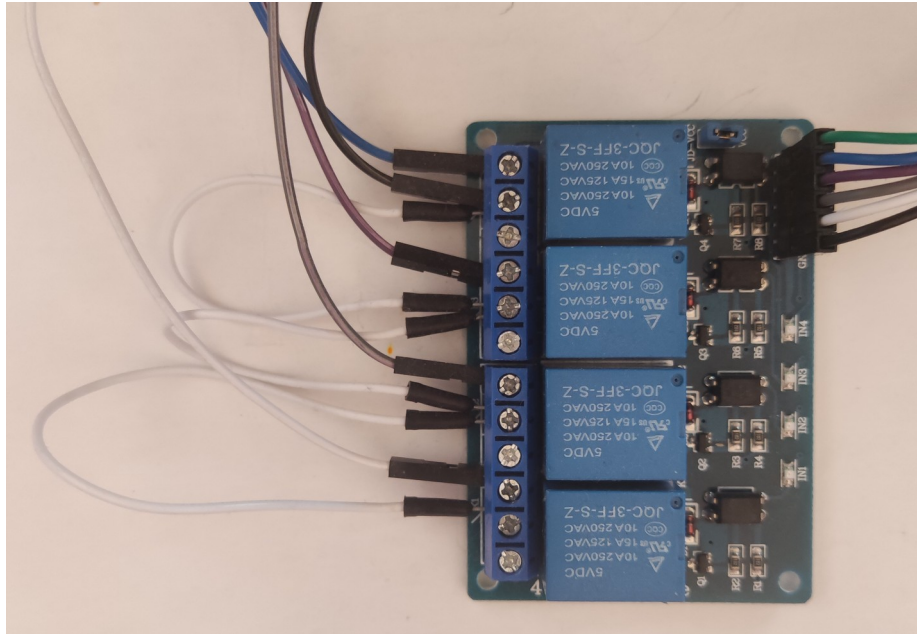
Sensor de ruidos fuertes: no graba sonido. Solamente se activa cuando su sensor detecta que hay sonidos fuertes. En este caso tiene salida digital como analógica y la salida digital se puede regular mediante un potenciómetro al que se le pueden dar varias vueltas para que ajustar el valor deseado sea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



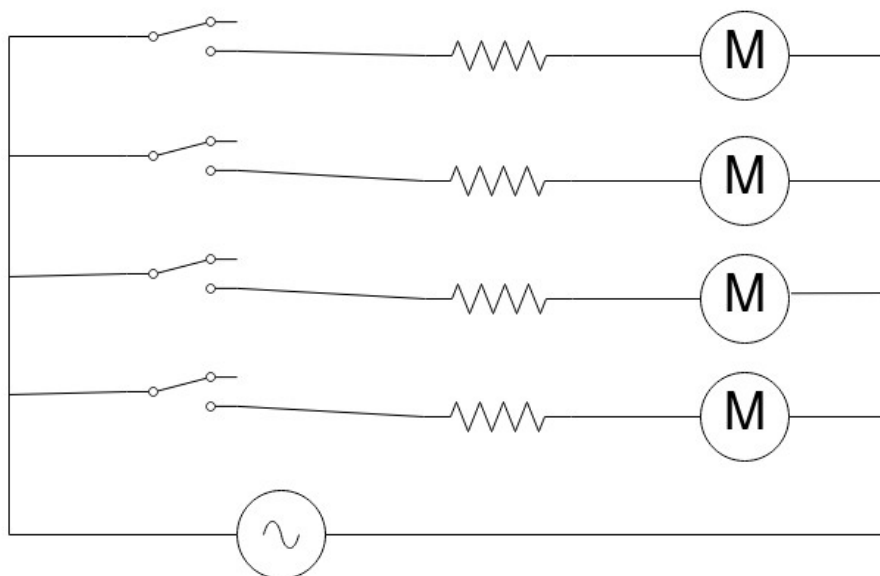
Sensor de humedad y temperatura DHT11: este sensor es diferente a los demás. Lee dos valores diferentes: la humedad en el ambiente y la temperatura. Solamente tiene una patilla digital para estos dos valores, por lo que es aconsejable usar una librería dedicada para manejar este sensor. Además hay que tener en cuenta que esta librería tardará unos mili segundos 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 cuando 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 y pudiendo activar cuatro líneas de tensión.



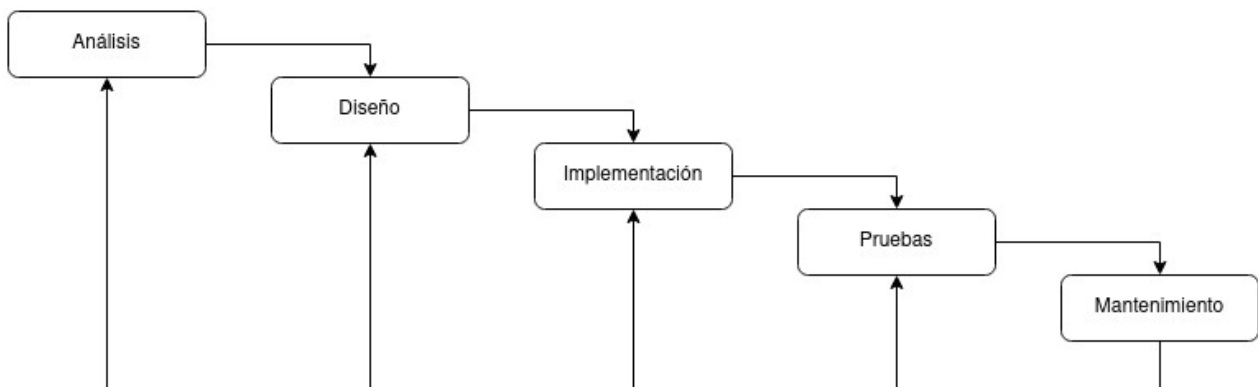
Este es el circuito electrónico que puede activar cuatro líneas de maquinaria. Cada relé esta conectado a un línea de tensión en un cada de cada relé representado por cada interruptor en la imagen. Por el otro lado la maquinaria esta conectada al conector normalmente desconectado del interruptor. De esta forma cuando se activa el interruptor se enciende la maquinaria conectada:



METODOLOGÍA PARA EL DESARROLLO

Existen varias metodologías para desarrollar software. Incluso varias clasificaciones. Estas son las metodologías mas comunes y sus ventajas e inconvenientes para este proyecto:

Metodología en cascada: en esta metodología no empieza cada fase del desarrollo hasta haber terminado la anterior. En cada fase se tratan todas las funcionalidades del programa. Para añadir un poco mas de flexibilidad se introdujo la opción de volver para atrás en las etapas de desarrollo. A pesar de esto la metodología sigue siendo muy poco flexible. Hay mucho tiempo transcurrido entre que la mayor parte de las funcionalidades son planificadas hasta que son realizadas. Por lo tanto cuando estas funcionalidades son implementadas ya pueden no tener interés o puede que no quede tiempo para realizarlas y su planteamiento haya sido un gasto de tiempo inútil. Esta metodología es útil en la actualidad para proyectos relativamente pequeños y cerrados que se quedarán casi concluidos al implantarlos. Este proyecto no es lo suficientemente pequeño para esta metodología así que no se usará esta. Se utilizará alguna metodología ágil que usará ciclos en espiral definidos al comienzo.



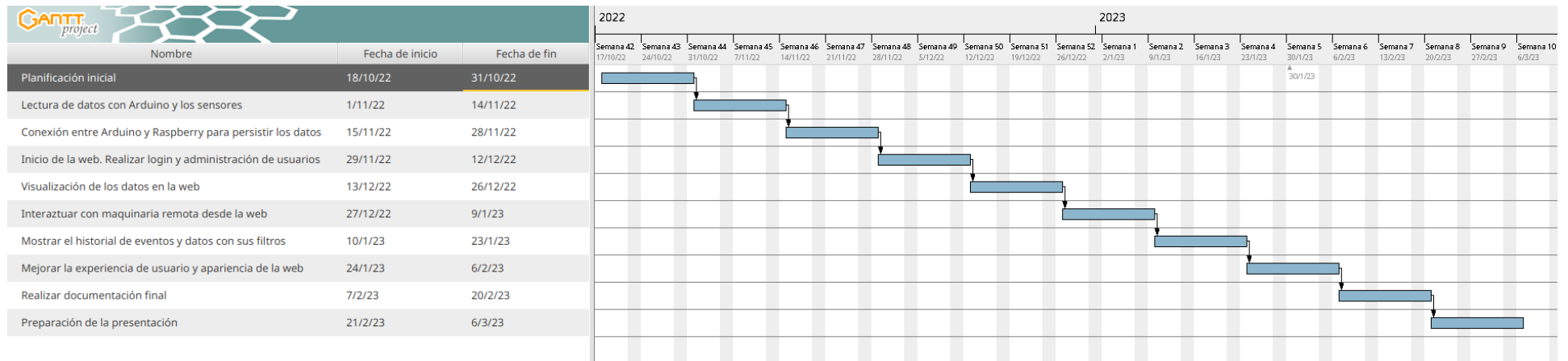
Metodologías ágiles: están basados principalmente para separar funcionalidades en orden de desarrollo e importancia. De forma que cada una de estas partes se planea y se desarrolla dejando al resto para planificaciones posteriores. Estas son las metodologías mas comunes aunque siempre hay muchas variantes de cada una:

Extreme programming: normalmente abreviado a XP. Esta basado en varias elementos: usar varias cascadas seguidas, integrar al cliente durante muchas horas para interactuar con el cada cambio en el programa y adecuarse a cualquier cambio, en muchas ocasiones usar dos desarrolladores a la vez programando en un solo ordenador entre otras. Es aconsejado para desarrolladores con experiencia. Dadas las características de este proyecto no es la más aconsejable.

Kanban: esta basado en tarjetas repartidas en columnas. En cada tarjeta esta una funcionalidad o cosa por hacer. Estas tarjetas se reparten en columnas con títulos como los siguientes: Bloqueadas, Realizadas, Por hacer. Este es un método de desarrollo donde la comunicación con el cliente se basa en pequeñas tareas que pueden repartirse en tiempos muy diferentes y con distinto nivel de prioridad. No es el caso de este proyecto aunque usar estas tarjetas podría ser útil.

Scrum: probablemente el mas popular de los tres. Tiene un gran número de variantes. Esta compuestos por los denominados “sprints” que consisten en una iteración con todos los pasos del software para una cantidad de funcionalidades determinadas y que tienen una duración determinada. Al inicio de cada uno de estos sprints se acuerda con el cliente los elementos a desarrollar en ese sprint y cuando termina se le entrega el resultado. Hay convenciones de roles de cada persona del proyecto. Escogeremos esta metodología ya que es muy versátil y la dirección es muy clara y los resultados son rápidamente visibles por el cliente.

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 cada semana esta compuesta por 15 horas hace un total de 30 horas por tarea. Por lo tanto hace un total de 300 horas.

Se han puesto las partes de la planificación por el orden en la que se ejecutarán. En la realización de algunas tareas son necesarias las tareas anteriores, en otras no. Sin embargo se ha puesto que cada tarea no comience hasta que empieza la anterior para cumplir con la metodología Scrum y por que el proyecto esta realizado solamente por una persona.

DESGLOSE DE PARTES

TAREA	SUBTAREA	HORAS
Planificación inicial	Recopilar las intenciones del proyecto, su alcance y la forma de afrontarlo	30
Investigación y formación	Investigación de las tecnologías y la viabilidad de cada una. Contrastar las ventajas e inconvenientes de cada tecnología	10
	Formación en cada una de estas tecnologías	20
T 1 Lectura de datos con Arduino y los sensores	T 1.1 Montaje del cableado entre Arduino y los sensores	10
	T 1.2 Lecturas de datos de cada sensor desde el programa de Arduino	10
T 2 Conexión entre Arduino y Raspberry pi para persistir los datos	T 2.1 Comunicar Arduino y Raspberry pi desde sus respectivos programas. Paso de los datos leídos a la Raspberry pi	10
	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 Visualización de los datos en la web	T 3.1 Diseño de la base de datos para la web y generar la base de datos	10
	T 3.2 Inicio del backend con los micro servicios 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	20
	T 3.3 Inicio del frontend que lee los datos mediante micro servicios y mostrar sus últimos valores en la web	20

TAREA	SUBTAREA	HORAS
T 4 Realización del login en la web y gestión de los usuarios	T 4.1 Implementación de la creación de tablas visibles en la página. Estas se podrán rehusar rápidamente en Angular	20
	T 4.2 Implementación de la creación de formularios emergentes rápidamente con Angular	20
	T 4.3 Creación de la tabla con sus relaciones en la base de datos y sus correspondientes micro servicios administrando el identificador de sesión.	5
	T 4.4 Realización del CRUD mediante llamadas en el frontend	5
	T 4.5 Gestión de los usuarios con sus visualizaciones, filtros y su correspondiente CRUD y login desde el frontend	5
T.5 Interactuar con maquinaria remota desde la web	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 Modificar los micro servicios 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	10
	T 6.3 Mostrar gráficos de los datos requeridos en el frontend	10
T.7 Realizar simulaciones con muchos lugares remotos donde se toman datos	T 7.1 Realizar programa que simule las llamadas de múltiples lugares que envían datos	10
	T 7.2 Combinar las entradas del programa con el prototipo físico realizado con Raspberry pi, Arduino y sensores	5
	T 7.3 Examinar el comportamiento del sistema y mejora de la simulación	15

TAREA	SUBTAREA	HORAS
T.8 Valoración y mejora de la experiencia de usuario	T 8.1 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 Mejora de cada acción que puede hacer el usuario. Mejora del rendimiento en cada acción de la forma posible	10
TOTAL HORAS		300

PRIMER SPRINT

Objetivos:

T 1.1 Montaje del cableado entre Arduino y los sensores	10
T 1.2 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

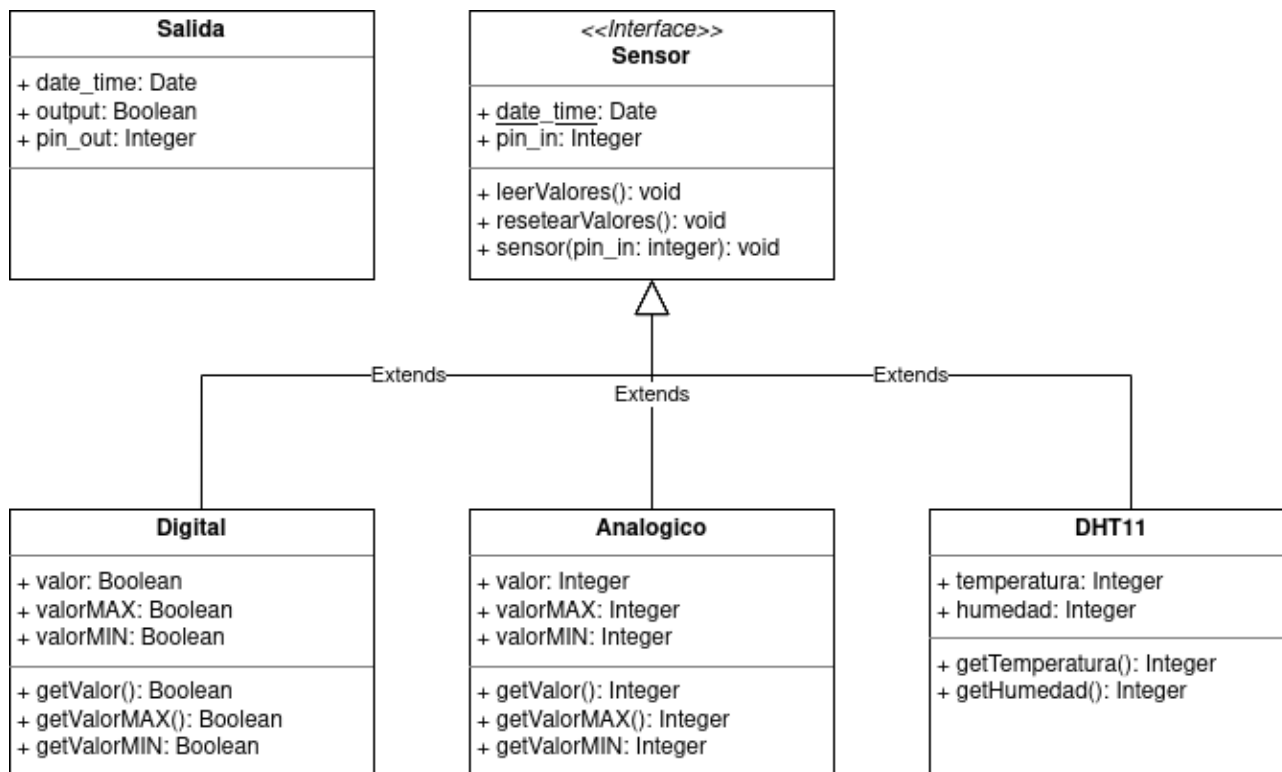
T 1.1 Montaje del cableado entre Arduino y los sensores

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.

FALTA DE FOTO CON LA REALIZACIÓN

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

Lo primero que se ha realizado es la estructura de los datos de los sensores y las salidas:



Esta será la estructura de las entidades del programa que residirá en el Arduino. Tendremos cuatro salidas y varios sensores. Estos sensores tendrán varios tipos y serán inicializados con el valor del pin que usarán. Se ha dividido los sensores en tres tipos: Digital, Analógico y otro tipo mas para el sensor DHT11 que tiene sus propias características. En el supertipo de todos los sensores está el campo del pin que usan y los métodos para leer los valores actuales del sensor físico y los almacene en los campos del objeto. Por lo tanto este método tiene que ser virtual. También está el método `resetearValores` que reinicia los valores máximos y mínimos de los valores. 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. Sí que se usará en los otros sensores 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 si no el valor mas alto o mas 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 por que 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: 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 ocupada al Arduino durante unos cuantos mili segundos 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 dispositivo que vamos a usar es 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 lo tanto 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 300 líneas y este es el resultado de la compilación:

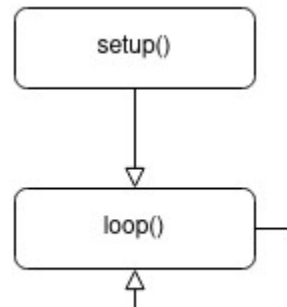
Compilado

El Sketch usa 10046 bytes (31%) del espacio de almacenamiento de programa. El máximo es 32256 bytes.
Las variables Globales usan 521 bytes (25%) de la memoria dinámica, dejando 1527 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 mas.

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 mas para usar variables globales. Al arrancar el programa se ejecuta la función setup() y cuando

termina se ejecuta la función `loop()` constantemente. Probablemente esto se haya realizado así para facilitar la programación a los novatos en el mundo de la programación. 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.

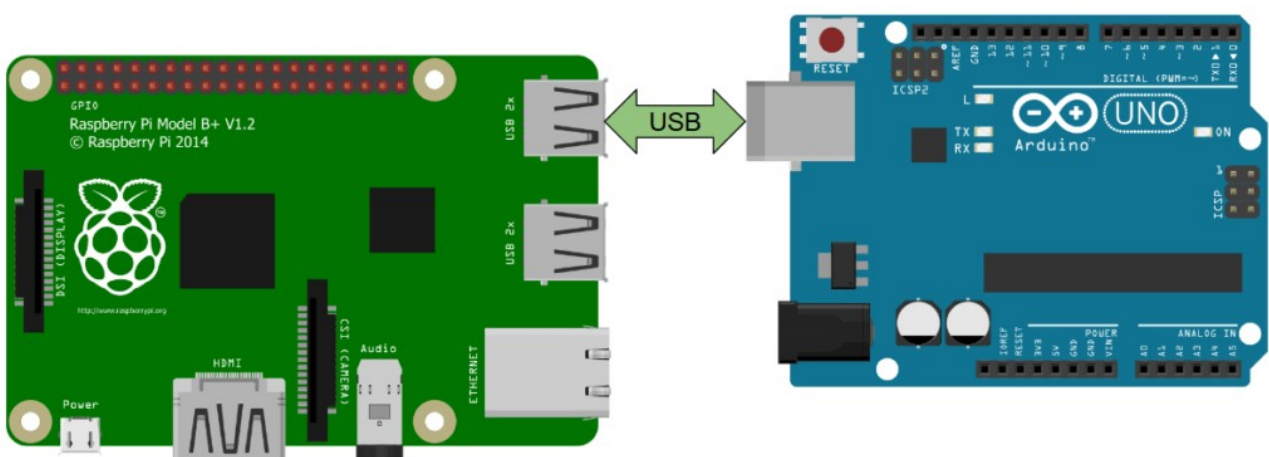
T 2.1 Comunicar Arduino y Raspberry pi desde sus respectivos programas.

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 mandará 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

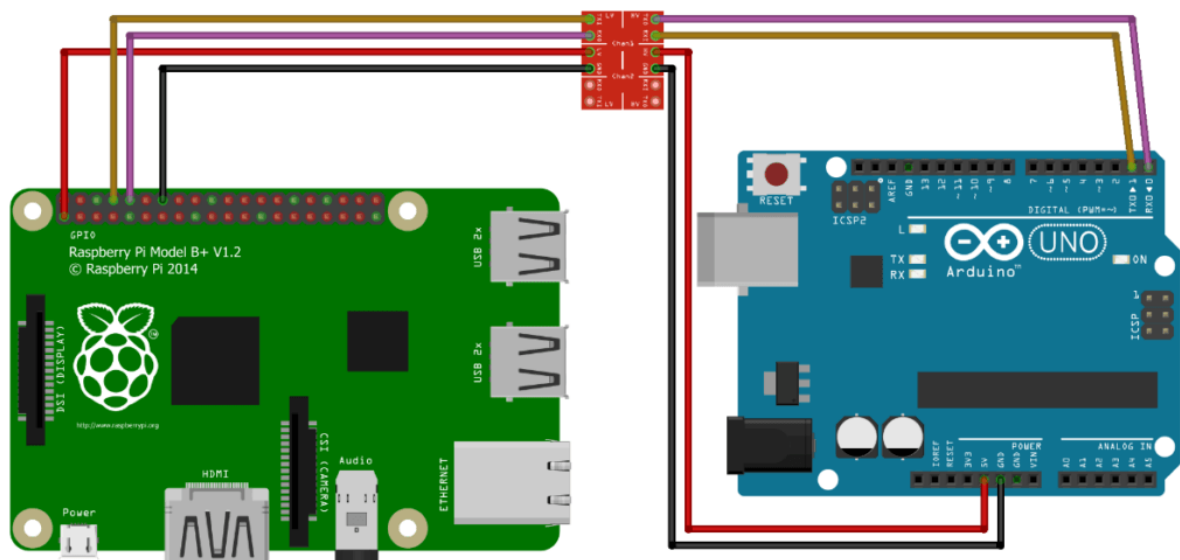
expuestos. Hay varias opciones para comunicar una Raspberry un un Arduino. Principalmente tres:

Conexión serie mediante USB: la mas aconsejada y la usada en el proyecto. Es simple físicamente y también tiene el código mas 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 resets 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 mas simple este sistema tiene otra ventaja sobre las demás: se puede cambiar rápidamente el sistema Raspberry por otro tipo de micro ordenador que funcione con Linux. Esto es por que 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 Arduino a una sola Raspberry. Pero esto esta fuera del alcance del proyecto y antes de aumentar la complejidad física sería mas práctico cambiar esta placa Arduino por otra con mas entradas y salidas. El caso de que con la mayor placa Arduino no sería 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 mas 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.

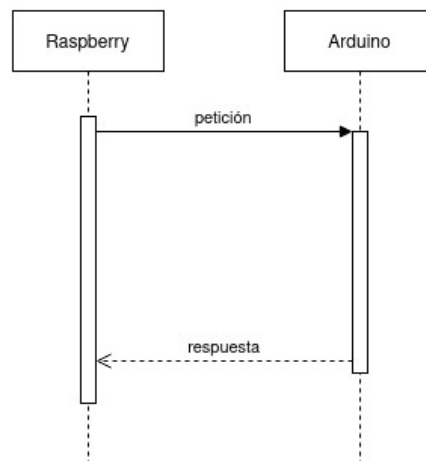
Como se puede ver en la imagen físicamente es mas complicado y difícil de mantener. Conecta las entradas de un componente con las salidas de otro. El componente intermedio es un convertidor de niveles de tensión ya que las patillas de una Raspberry funcionan a un nivel de 3.3 voltios mientras el nivel de funcionamiento de las patillas de un Arduino son 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 micro ordenador 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 esta 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 esta 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 pondrán las salidas que tenga la placa

Arduino y se recibirá 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. De la respuesta que recibirá la Raspberry tendrá su propio checksum para comprobar si la respuesta es correcta o no.

ESTABILIDAD DEL SISTEMA RASPBERRY PI

Un aspecto importante es el estado de la Raspberry 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:~ $
```

The image shows a terminal window titled '(pi) 192.168.0.222 — Konsole'. The window has a menu bar with 'Archivo', 'Editar', 'Ver', 'Marcadores', 'Preferencias', and 'Ayuda'. The terminal content shows the user running two commands: 'vcgencmd measure_temp' which outputs 'temp=44.5'C', and 'vcgencmd get_throttled' which outputs 'throttled=0x0'. The prompt 'pi@raspberrypi:~ \$' is visible at the end of each line.

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.

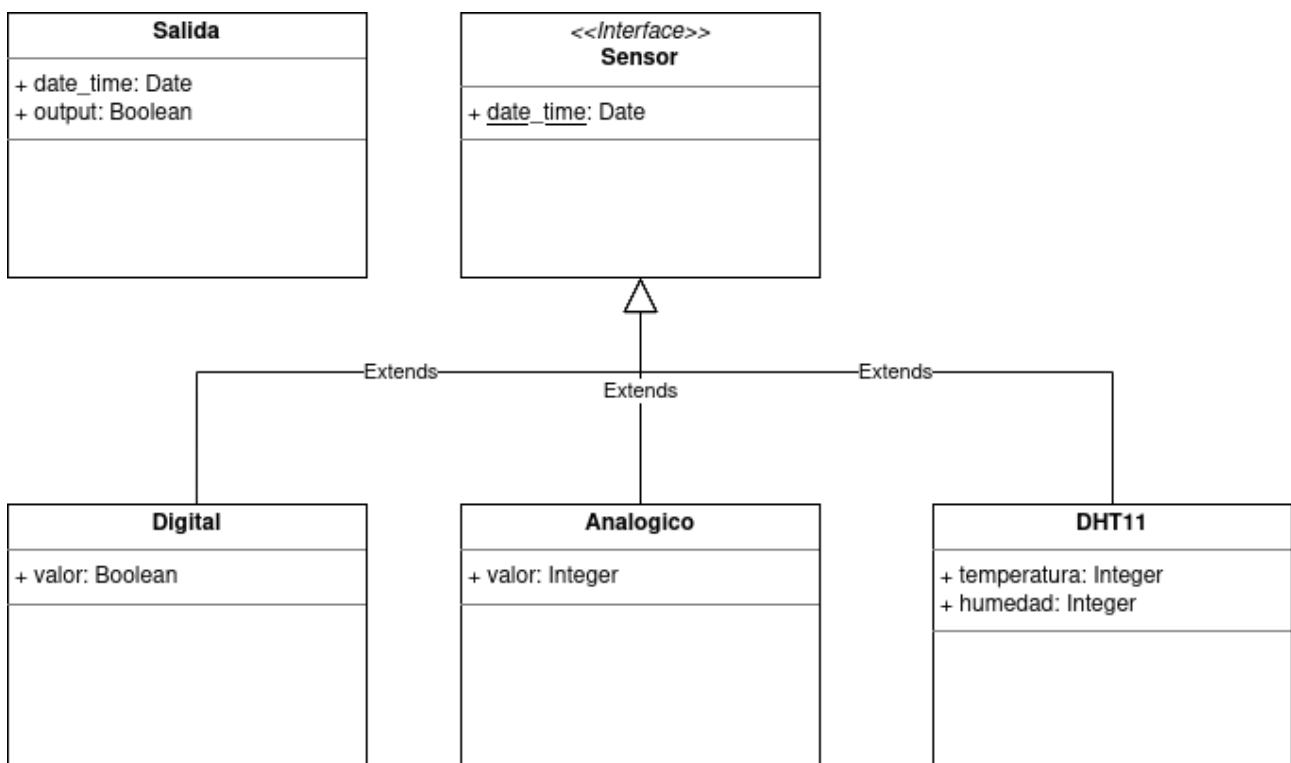
El segundo comando examina la tensión. El valor 0x0 significa que la tensión ha sido estable desde el encendido de la Raspberry pi.

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

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 Diseño de la base de datos para la web y generar la base de datos	10
T 3.2 Inicio del backend con los micro servicios 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 (inicio)	20

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. La estructura representada 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.

Para un ahorro del espacio se ha tomado esta estructura para ser guardada en la base de datos:

Actions	Sensors
+ <u>date_time</u> : Date + binary_values: integer	+ <u>date_time</u> : 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 tenemos las acciones que serán las salidas del Arduino. La clave es el tiempo, de forma que para obtener los valores que hay que enviar al Arduino se extrae la fila con la fecha mas alta, la cual se extrae rápidamente por que es el campo clave.

Por último la segunda fila 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 tanto para 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 estará el valor “has_sended” compuesto por un booleano. Será false si la final 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 sera false.

Este es un ejemplo de los datos que se guardan en la base de datos por el funcionamiento del sistema:

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

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

En cuanto a la creación de las tablas en la base de datos se ha recurrido al ORM de Node.js llamado Sequelize que una vez definidas las tablas en JavaScript con su estructura las genera automáticamente si es que no están generadas ya. Para hacer esto primero hay que crear un proyecto de Node.js y configurar Sequelize y la definición de las tablas.

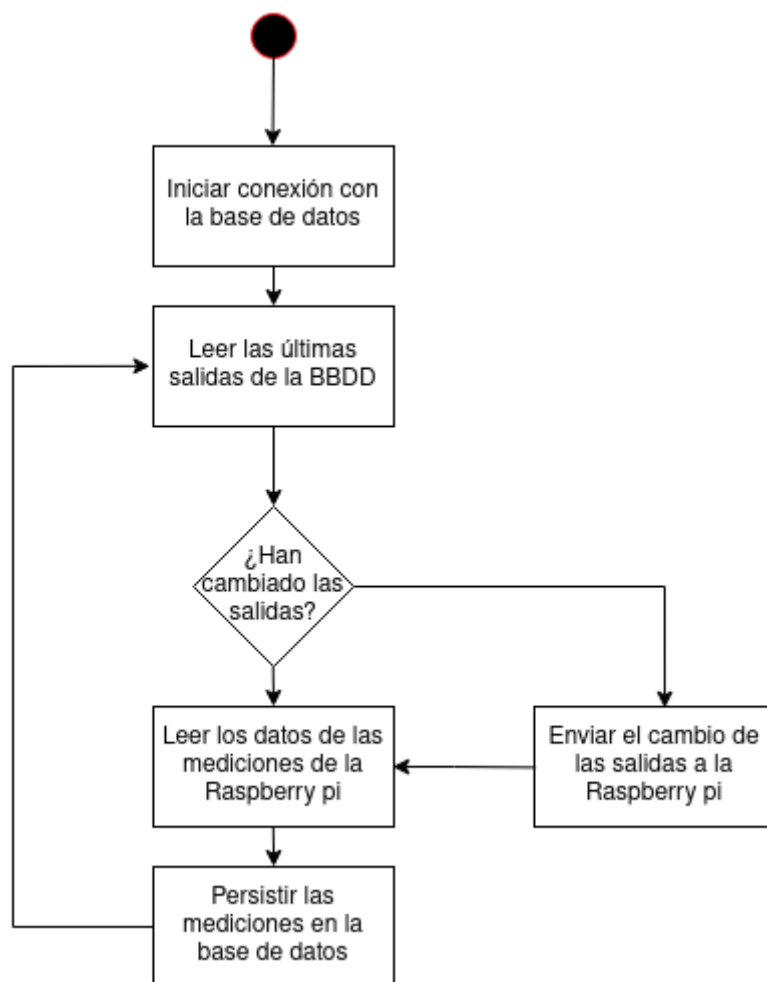
T 2.3 Hacer que el programa que se ejecuta en la Raspberry pi persista los datos en la base de datos

Para interactuar con la base de datos en Postgress con la Raspberry pi usaremos la librería para language 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 de las bases de datos entre la Raspberry y el servidor central.

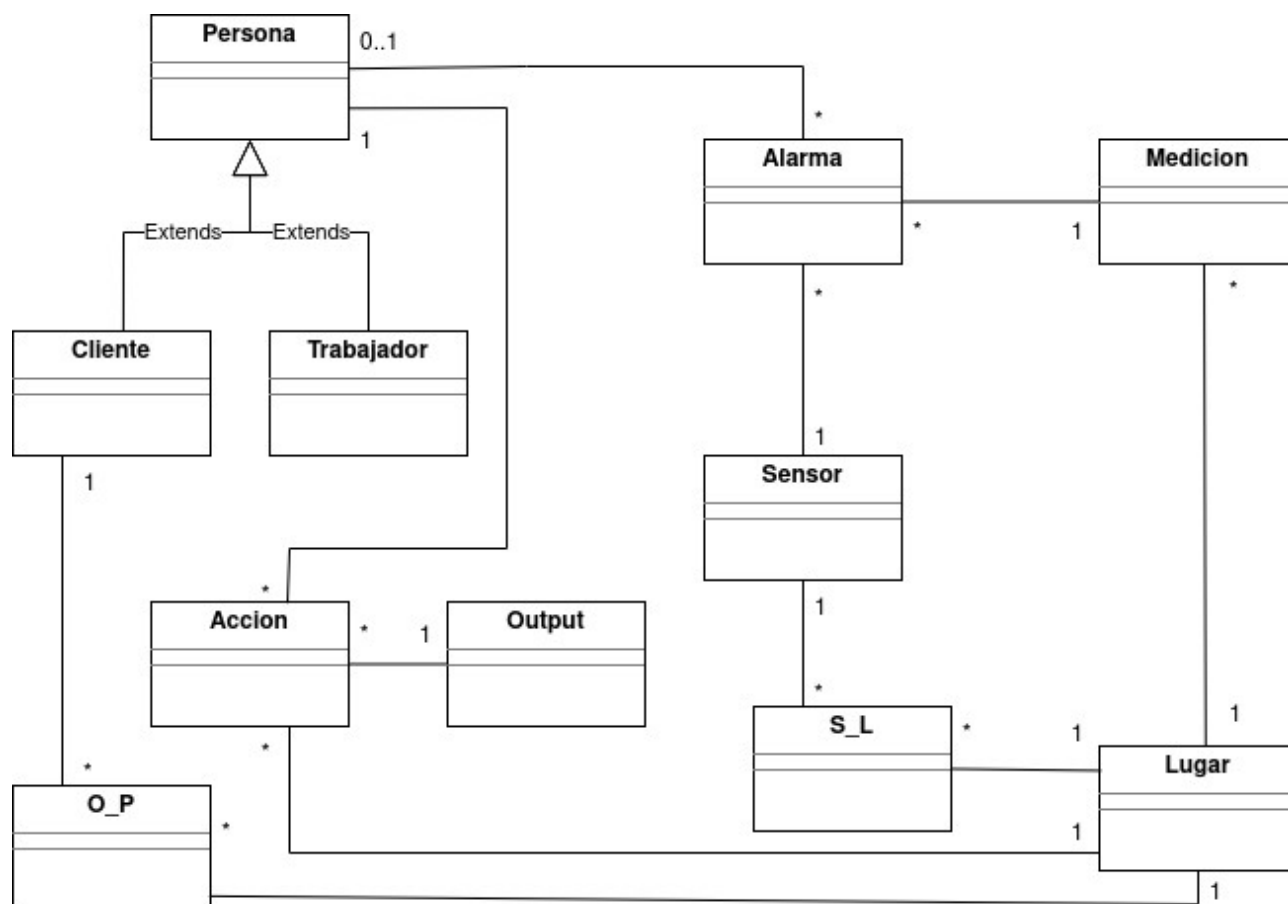
El flujo del programa sería el siguiente:



Como el programa no tiene final no hay punto final.

T 3.1 Diseño de la base de datos para la web y generar la base de datos

Se ha ideado la siguiente estructura de entidades:



Son unas cuantas tablas que exigen una severa explicación.

Tabla Medición: es equivalente a la tabla Sensors 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.

Tabla Alarma: se refiere a un evento generado por una medición en un sensor. Como la tabla Medición 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, 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.

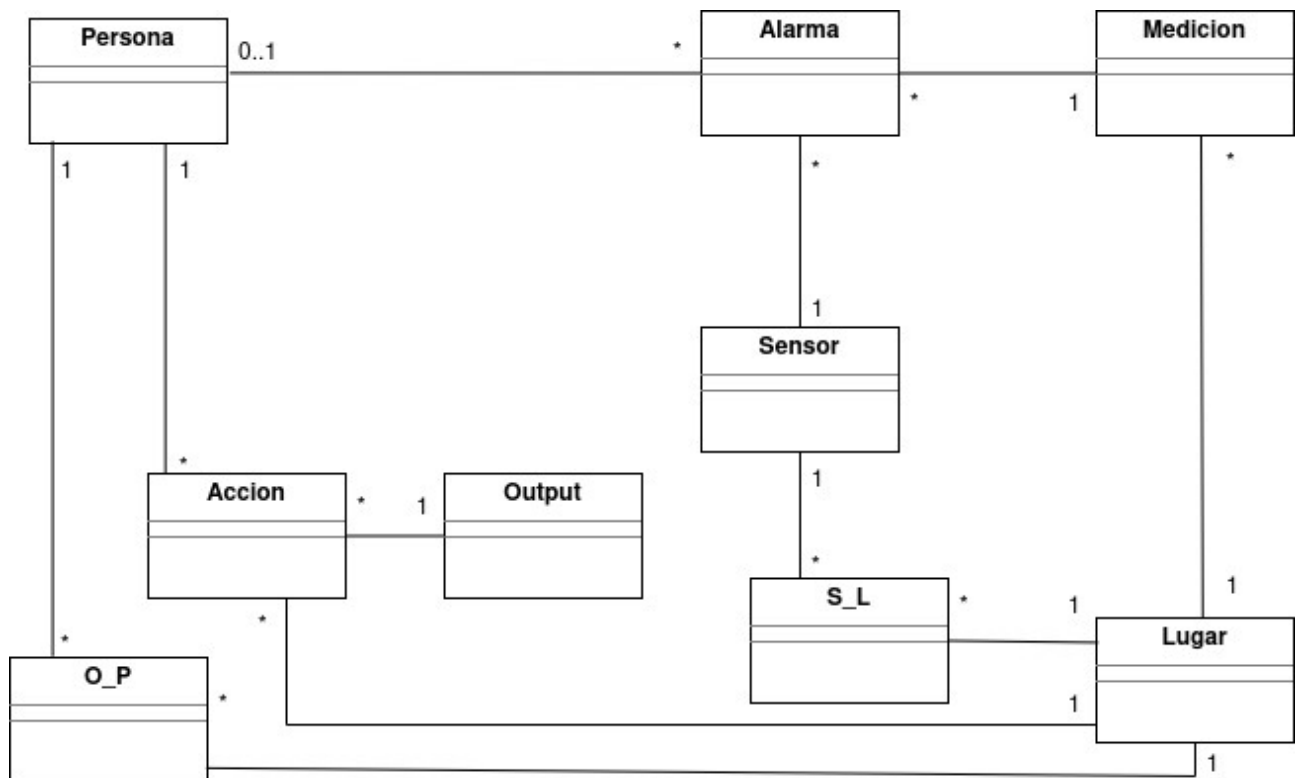
Tabla sensor: representa un modelo de sensor. De momento tendremos 10 predefinidos pero se pueden añadir mas. Dado que en cada lugar puede haber varios modelos de sensores y cada modelo de sensor puede estar en varios lugares hay una relación varios a varios con la tabla Lugar. Dado que la tabla Medición no esta normalizada si queremos incluir mas modelos de sensores será necesario incluir mas campos en esta tabla. Esto puede ser un problema pero es uno de los precios a pagar por el rendimiento.

Tabla Output: 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 que salidas tiene cada lugar, como en el caso de los modelos de sensores.

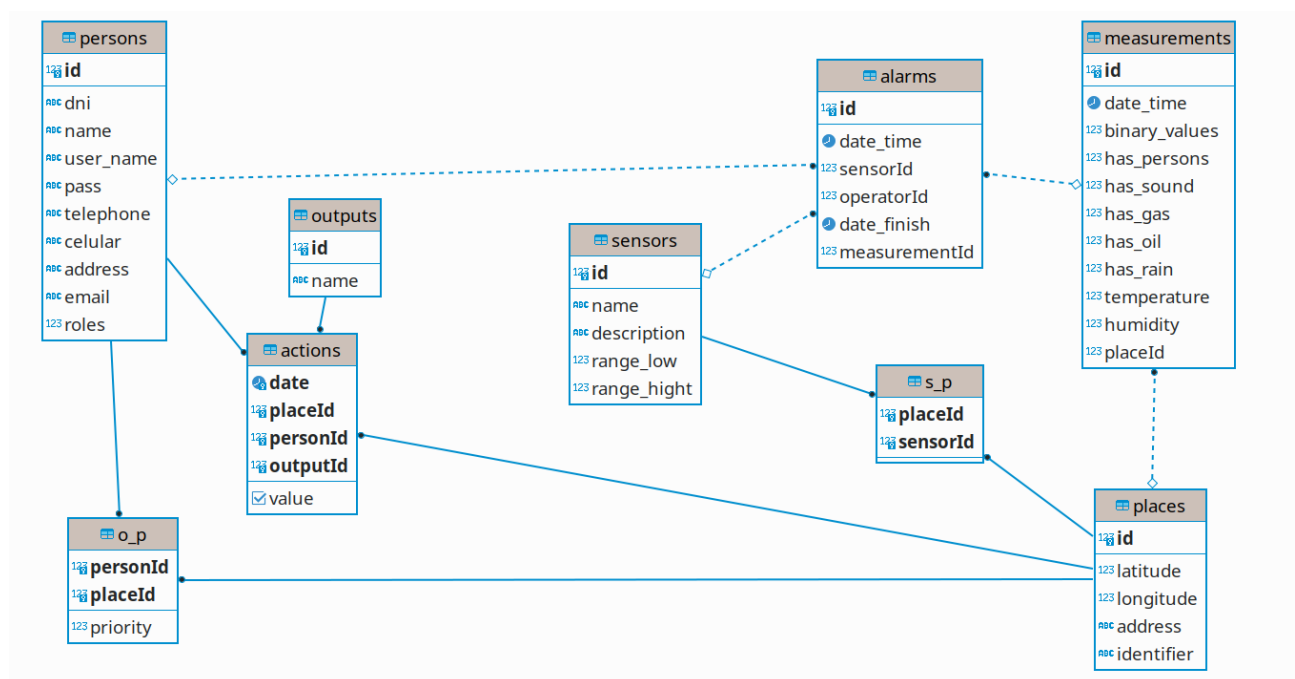
Tabla Acción: representa un cambio en una determinada salida de un determinado lugar realizado por una determinada persona. Esto resulta en una relación ternaria. Esta relación entre tablas está normalizada. Se podría haber simplificado haciendo una desnormalización como se ha hecho con la tabla Medición 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.

Tabla Persona y sus subTablas: representa a un usuario del sistema. Una persona puede ser un cliente o un trabajador 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 esta en que un cliente solo puede acceder a los lugares que tiene asignados mientras un trabajador puede acceder a todos los lugares. Teniendo en cuenta que los temas de contabilidad están mas allá del alcance de este proyecto por lo que los campos de ambas tablas serán los mismos. Teniendo esto en cuenta en el modelo final estas tres tablas en una sola con la inclusión del campo "rol" en la tabla para diferenciar entre cliente y trabajador. Para el alcance el proyecto es una buena solución pero hay que tener esta simplificación presente.

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



Esta una representación con los datos (a modificar):



T 3.2 Inicio del backend con los micro servicios 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 (inicio)

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. Se usa una estructura de capas similar a la estructura Modelo-Vista-Controlador. Cada microservicio esta estructurado en un directorio que contiene 4 archivos:

<nombre>-model.js: es donde esta 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.

<nombre>-repository.js: 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. Esta clase y el modelo es la que compone la capa del modelo de la estructura Modelo-Vista-Controlador. Solamente responde a una llamada HTTP si ocurre un error en la capa de persistencia. Si todo va bien pasa los datos obtenidos de la base de datos y de los parámetros de entrada a la capa de servicio.

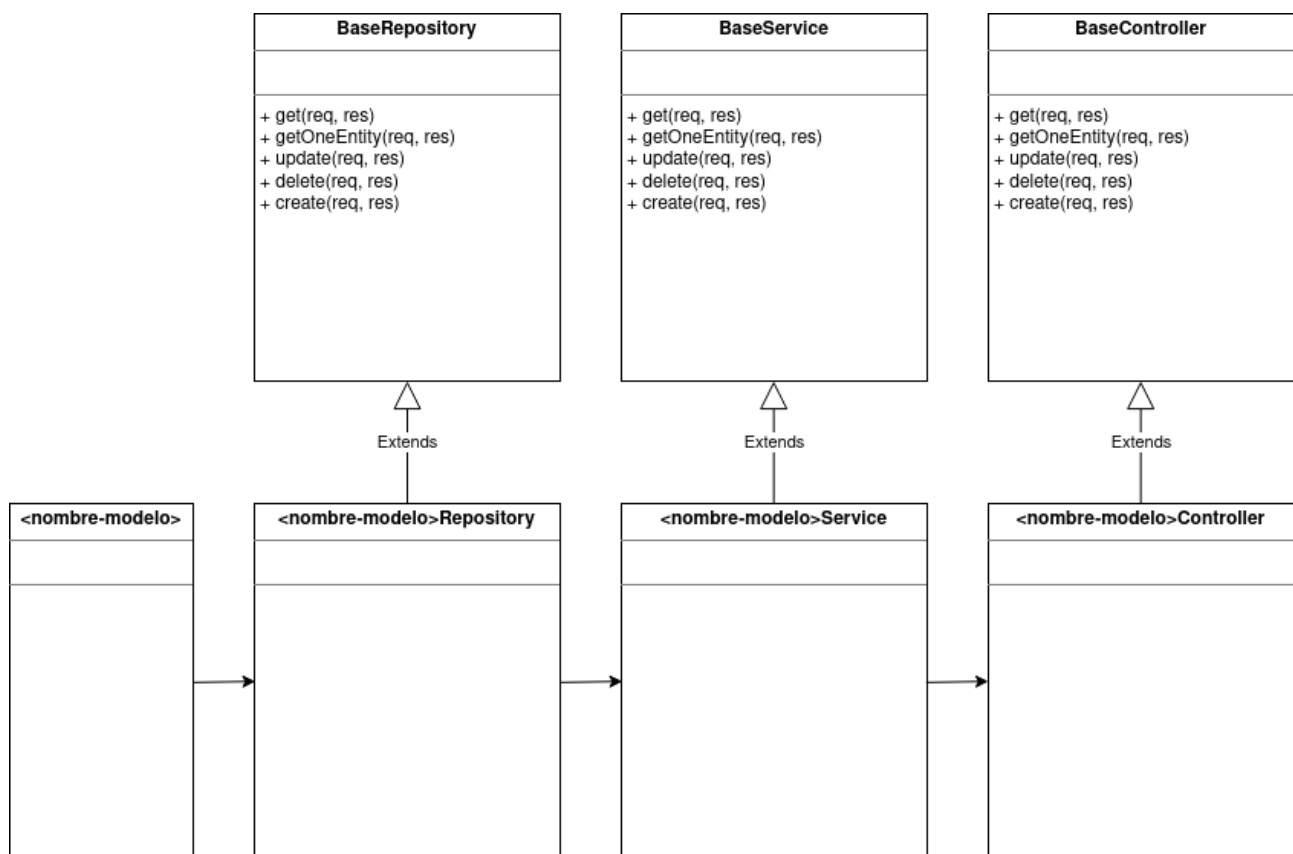
<nombre>-service.js: hereda de BaseService. Es el equivalente a la capa del controlador donde reside toda la lógica de negocio de la entidad. Si no se sobreescribe ningún método de la clase padre entonces no se usa ningún tipo de lógica especial.

<nombre>-controlador.js: es el equivalente a la capa de la vista de un microservicio. Toma los datos que le llegan del servicio y con ellos responde a las llamadas HTTP que le llegan.

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

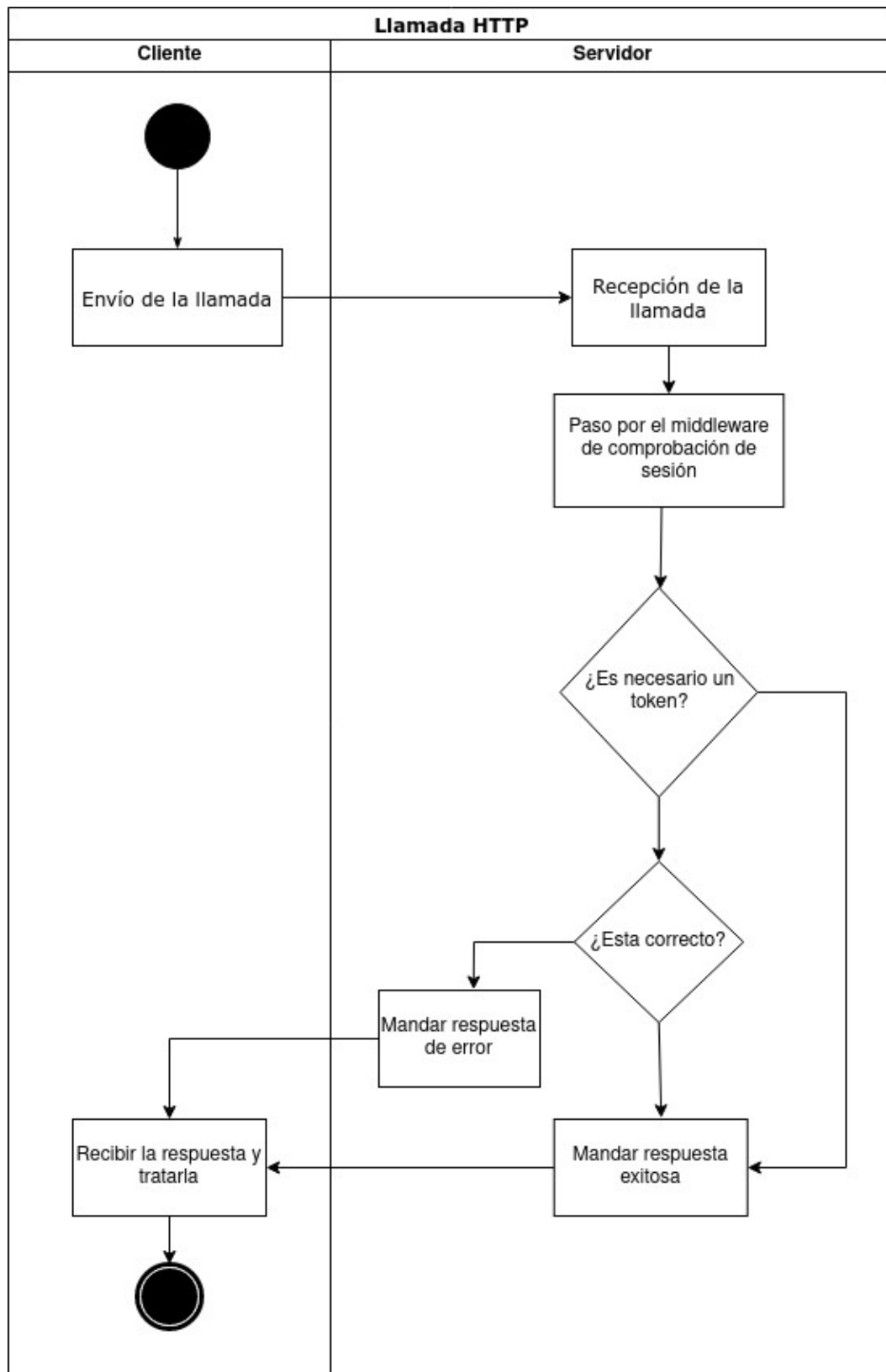
En la estructura de las clases esta basada en una cascada: cada objeto recibe una instancia del objeto anterior en su creación. De esta forma llama a los métodos de la instancia que ha recibido en su creación.

Este es un diagrama de la estructura de cada microservicio:



De esta forma cada capa del microservicio hereda de una clase predefinida. Consiguiendo que no sea necesario repetir código en cada microservicio.

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 automáticamente 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.



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.

