Universidad de Valladolid

Escuela de Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática
(Mención Tecnologías de la Información)

Sistema de sensorización para el control de clima en cultivos

Autor:
D. Kave Heidarieh Sorosh

Universidad de Valladolid

Escuela de Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática
(Mención Tecnologías de la Información)

Sistema de sensorización para el control de clima en cultivos

Autor:
D. Kave Heidarieh Sorosh
Tutores:
Dr. Arturo González Escribano
Dr. Diego R. Llanos Ferraris

Dedicado a todas las personas que luchan por conseguir sus objetivos

Agradecimientos

Quiero agradecer,

a los profesores que me han dado clase, por el conocimiento que me han transmitido, al departamento de Matemáticas por su apoyo explícito y reconocimiento de mis esfuerzos en la materia.

A mis compañeros, de los cuales he aprendido mucho, con los que he compartido muchas horas de esfuerzo y otras de alegrías.

A mis amigos, por aguantar mis momentos de tensión, apoyarme, comprenderme y alentarme.

A Isabel, que me ha acompañado en este, y otros largos viajes, por su comprensión y guía.

A mi hermana, por ser la persona que siempre está ahí, como un faro.

A mi padre, por ser un pilar en mi vida.

Resumen

El presente proyecto pretende ser un acercamiento al mundo de Internet of Things a través de una serie de experimentos que van mutando en su estructura, que no en su funcionalidad, a lo largo del desarrollo del mismo. El objetivo principal ha sido recoger información del entorno más próximo mediante sensores y actuar sobre el entorno a través de actuadores conectados a diferentes elementos eléctricos o electrónicos que puedan ser accionados y puestos en marcha de manera remota. Lo que ofrece Internet of Things, y esta aplicación en concreto, es una especie de ubicuidad al poder situar estos Objetos en cualquier entorno con conectividad Wi-Fi y alimentación eléctrica. Esto permite que dichos espacios adquieran cierta inteligencia al tener recuerdo de las situaciones pasadas de manera que podremos actuar sobre ellos con mayor conocimiento.

El presente trabajo ha sido ideado para conseguir datos relevantes de un cultivo mediante su sensorización y tener la posibilidad de actuar sobre él de manera remota accionando elementos eléctricos o electrónicos. Algo a resaltar de este trabajo, es la polivalencia que presenta el sistema al permitir ser adaptado e implementado a otros entornos con relativa facilidad. Mediante cambios en el Modelo de Negocio puede ser utilizado incluso en áreas como la domótica.

Abstract

This project encompasses the Internet of Things with a series of experiments, by continually developing the structure, but not its functionality. The aim of the project is to gather information from the surrounding environment using sensors which feed information to the application. Actuators are placed to allow electronic components to be controlled. Considering the Internet of Things, by using this application it allows a kind of ubiquity to acquire intelligence using a WiFi enabled environment where data can be stored and in effect learned.

This system has a number of integrated devices which can obtain a myriad of data by using sensors which relay information back to the application. Further, by using remote access we can activate electronical devices such as irrigation to water plants at specified intervals. A key benefit of the system, is its sheer versatility and adaptability to a variety of environments. This can be achieved by simply adding different functionalities within the business model.

Índice

Índice de figuras

Índice de tablas

Listings

1. Introducción

1.1. Contexto

La informática está presente en la actualidad en un sinfín de campos y su introducción en el día a día aún se va a intensificar más en los próximos años con una presencia mucho más acusada y esta, no será tan obvia como la informática a día de hoy. En su mayoría está ligada a una pantalla y un teclado, sin embargo, evolucionará para pasar más desapercibida, estará oculta, integrada en los *Objetos* cotidianos.

Uno de los campos con mayor capacidad de crecimiento en la informática se prevé que sea el de Internet de las Cosas/Objetos, *IoT* (Internet of Things), íntimamente relacionada con *Computación pervasiva* ya estas promueven al ubicuidad de la informática. Este crecimiento en *IoT* puede suponer una gran revolución en el día a día de los usuarios que ahora verán integrada la tecnología a su alrededor y pasará inadvertido porque los dispositivos no llevarán pantalla y no se verán [8].

Así, las casas serán *inteligentes*, habrá un sinfín de dispositivos conectados a internet. Estos, se podrán comunicar entre sí, con los personas o viceversa. De esta manera cuando dos dispositivos se comuniquen entre sí, uno de ellos podrá dar órdenes al otro sin la intervención humana [1]. El *IoT* se podrá encargar de muchas labores que a día de hoy realizamos nosotros y nos liberará de tediosas labores del hogar. Así, nuestro frigorífico, que tendrá múltiples sensores, *sabrá* el contenido que tiene en su interior, *sabrá* qué alimentos nos gustan y dónde puede comprarlos cuando sea necesario para no quedarnos sin aquellos productos que queremos tener en nuestro frigorífico. La limpieza será relegada a las maquinas autónomas que serán las encargadas de hacer el mantenimiento de nuestros hogares. Aun más, antes de levantarnos, la casa estará caldeada en invierno, tendremos preparado el café en la cocina y según nos aseamos en el baño algún *Objeto* nos recordará la agenda para el día, al cepillarnos los dientes, el cepillo analizará varios parámetros a través de la saliva relacionados con nuestra salud, etc. Después de habernos vestido nuestra ropa *hablará* entre ella y empezarán a cambiar de color para ir bien conjuntados y todo dependiendo de nuestro estado de ánimo ya que los sensores conectados a nuestro *Objeto* cama y almohada habrán registrado determinados parámetros del sueño y *sabrán* cómo hemos dormido y por la manera de levantarnos también *sabrán* nuestro estado de ánimo. Una vez en la calle llevaremos otros *Objetos* que estarán midiendo nuestro pulso o la concentración de glucosa en la sangre. Todos estos *Objetos*, estarán conectados a Internet y transferirán todos esos datos a uno a varios servicios en la nube, donde se analizarán mediante *Big Data* y *Aprendizaje automático* con redes neuronales sabrán más de nosotros, aprendiendo de nuestros hábitos y podrán hacernos la vida más fácil.

Para mantener la confidencialidad de los datos recogidos, que serán muy abundantes, se desarrollará una estricta legislación que garantizará a los clientes/usuarios la seguridad de los mismos y permitirá que no puedan ser usados para otros fines que no sean los para los cuales se hayan obtenido.

En la agricultura se desarrollará mucho más lo que hoy en día se conoce como agricultura de precisión. En la actualidad se empieza a extender el uso de estaciones meteorológicas que miden un sinfín de parámetros como los ambientales, los de la tierra, la lluvia, la radiación solar o la velocidad del viento y su dirección. Estos *Objetos*, los diferentes sensores, envían la información en crudo a la nube donde se procesan y son transformados adquiriendo valor al proporcionar al agricultor índices que le facilitan la toma de decisiones como, por ejemplo, la evapotranspiración que ayuda a calcular la cantidad de riego necesario o la probabilidad de que el cultivo adquiera algunas enfermedades como el oídio o el mildiu, permitiendo tomar medidas antes de que suceda.

En cuanto a la industria, en la actualidad se habla de *industria 4.0* o *industria inteligente*. Las factorías prescindirán prácticamente de la intervención de operarios y serán las máquinas *inteligentes* las que mediante sus sensores conectados harán todo el trabajo repetitivo dejando otras funciones a las personas. Esto supone una gran revolución a nivel social ya que las manufacturas las podrán hacer casi de manera autónoma las máquinas dejando a las sociedades más tiempo libre para el ocio o la reflexión.

Todos estos cambios, que pueden parecer ciencia ficción, se pueden dar en muy pocos años ya que la evolución tecnológica cada vez se está acelerando más. El *Internet de las Cosas/Objetos* ha venido para quedarse y revolucionar la vida de las personas, de nosotros depende la dirección que tome la evolución, si a un mayor control social y con más problemas de desempleo o hacia una sociedad más justa y donde las máquinas trabajen por nosotros.

1.2. Motivación

Este proyecto nace a partir de una experiencia personal con un huerto casero y la sospecha de estar usando más agua en el riego de lo necesario para el cultivo. Me preguntaba cosas como ¿cuánta agua necesita una tomatera?, ¿hasta que profundidad llegan las raíces?, ¿cuánta humedad es recomendable para un buen crecimiento?, ¿cuánta agua estoy mal utilizando porque se va a más profundidad de lo que llegan las raíces?, o incluso ¿cuánto me cuesta, en litros de agua, cada kilogramo de tomates?

Con estas estas preguntas se empieza a dar forma a un sistema para poder medir algunos parámetros mediante dispositivos de bajo coste como microcontroladores Arduinos, ESP8266, sensores de humedad y temperatura, caudalímetros, relés, electroválvulas, módulos Bluetooth, cables, protoboards, leds, etc.

Además generalizando el sistema se podría usar para dotar de *inteligencia* una casa, *domótica*, y controlar la calefacción con el mismo sistema con unos pequeños cambios, lo que permitiría, por ejemplo, programar la calefacción con un calendario.

En el mercado hay soluciones desarrolladas ya maduras que funcionan muy bien pero la ventajas que tiene implementar un desarrollo propio es que se puede adaptar a unas necesidades concretas, utilizar dispositivos de bajo coste para su implementación, alimentar la curiosidad en el proceso de desarrollo e incluso darle un enfoque de producto al ser un proyecto eminentemente útil.

1.3. Propuesta

Una vez que se han detectado las necesidades de la aplicación a desarrollar se hace una propuesta a grandes rasgos sobre las funcionalidades del sistema.

Se propone *IoT* para implementar un sistema con el cual poder monitorizar un cultivo mediante sensores y actuadores. Estos pueden ser sensores de temperatura y humedad ambiental, humedad de la tierra para saber cuánto regar o caudalímetros para saber cuánta agua hemos utilizado en cada riego. En cuanto a los actuadores, nos focalizaremos principalmente en relés, que accionan el riego y en caso de instalarse en un invernadero podría usarse para el control del clima, accionando extractores, humificadores, etc.

Otra parte importante del proyecto, es la persistencia de los datos obtenidos, que pueden servir para utilizarlos y ofrecer servicios de agricultura de precisión. Aún así, sin procesar los datos, se puede ver la curva de la temperatura y humedad durante el día o saber cuántos litros de agua hemos usado durante el verano para regar nuestro huerto puede servir para ayudarnos a tomar decisiones en el futuro y sacar conclusiones.

La interfaz del sistema es una *API REST* con la cual pueden interactuar diferentes *clientes* como una aplicación web, una aplicación de un *smart phone* o desde cualquier otro servicio que pueda hacer peticiones HTTP en cualquiera de sus variantes.

Además, es un sistema flexible ya que se pueden añadir más dispositivos *inteligentes* con facilidad a los ya conectados según necesidades.

1.4. Objetivos

El objetivo principal de este proyecto es implementar un prototipo totalmente funcional con el cual se pueda dotar de *inteligencia* a un cultivo mediante el uso de *Internet de las Cosas/Objetos*. Se recogerán tanto datos datos ambientales como del suelo y se utilizarán actuadores para activar el riego y medir la cantidad de agua utilizada.

1. Hardware:
    - *a*) de bajo coste,
    - *b*) fácilmente accesible,
    - *c*) con una gran comunidad de desarrolladores,
    - *d*) mucha documentación en la red.
2. Software:
    - *a*) Desarrollar un sistema al se pueda añadir más módulos hardware con facilidad,
    - *b*) Que disponga de una interfaz con la cual un *cliente* pueda interactuar.
3. Prototipo:
    - *a*) que permita medir temperatura y humedad ambiental,
    - *b*) que permita medir temperatura y humedad de la tierra,
    - *c*) que permita acceder a los datos persistentes,
    - *d*) que permita accionar el riego,
    - *e*) que permita recoger la cantidad de agua suministrada.

2. Tecnologías Relacionadas

2.1. Internet de las Cosas/Objetos

Cuando se habla genéricamente de *IoT* se entiende por los objetos del día a día que se encuentran interconectados a través de Internet y que normalmente están equipados con *inteligencia ubicua*, es decir, estos *Objetos* pueden encontrase en todas partes y al mismo tiempo, no el mismo *Objeto*, y proporcionar inteligencia al obtener datos que se pueden procesar y proporcionar conocimiento. *IoT* incrementará la ubicuidad de Internet integrando cada *Objeto* que interactuará mediante sistemas embebidos lo que proporcionará una intensa red distribuida de *Objetos* que se comunicarán entre ellos así como con las personas.

2.1.1. Definición

Hoy en día no existe una definición única de todo lo que comprende *IoT*. El origen del término se atribuye a *Auto-Id Labs* del *Massachusetts Institute of Technology* en un estudio que hicieron sobre la tecnología de identificación por radiofrecuencia, RFID, en el año 1999 [9]. A partir de entonces la tecnología del *IoT* ha ido creciendo y extendiéndose más allá de la tecnología RFID.

Según la Unión Internacional de Telecomunicaciones, UIT, organismo especializado en telecomunicaciones dependiente del la Organización de las Naciones Unidas se define a *IoT* como, «Infraestructura mundial para la sociedad de la información que propicia la prestación de servicios avanzados mediante la interconexión de objetos (físicos y virtuales) gracias a la interoperatividad de tecnologías de la información y la comunicación presentes y futuras.» [10]

Existes otras definiciones sobre *IoT* que hacen énfasis sobre los *Objetos* conectados a Internet mientras que otros resaltan otras características como los protocolos utilizados o las tecnologías de red [11].

También se puede entender como una infraestructura global de la sociedad de la información que permite ofrecer servicios avanzados a través de la interconexión de *Objetos*, tanto físicos como virtuales, favorecidas por las *TIC*. Además se espera que *IoT* integre gran cantidad de las tecnologías avanzadas como M2M, la minería de datos y la toma autónoma de decisiones, tecnologías avanzadas de detección y accionamiento [3], etc.

Las *TIC* tradicionalmente ofrecen comunicación en cualquier *Lugar* y en cualquier *Momento* y ahora al usar la tecnología de *IoT* se añade una nueva dimensión que es, con cualquier *Objeto*, tal como se muestra en la figura 1.

2.1.2. Descripción Técnica

En *IoT* los objetos pueden ser del mundo físico o del virtual con la capacidad de integrarse en redes de comunicación. Los objetos físico existen en el mundo físico y se pueden detectar y actuar sobre ellos mientras que los objetos virtuales existen en el mundo de la información y se pueden almacenar, procesar y acceder a ellos. Los objetos físicos pueden ser representados

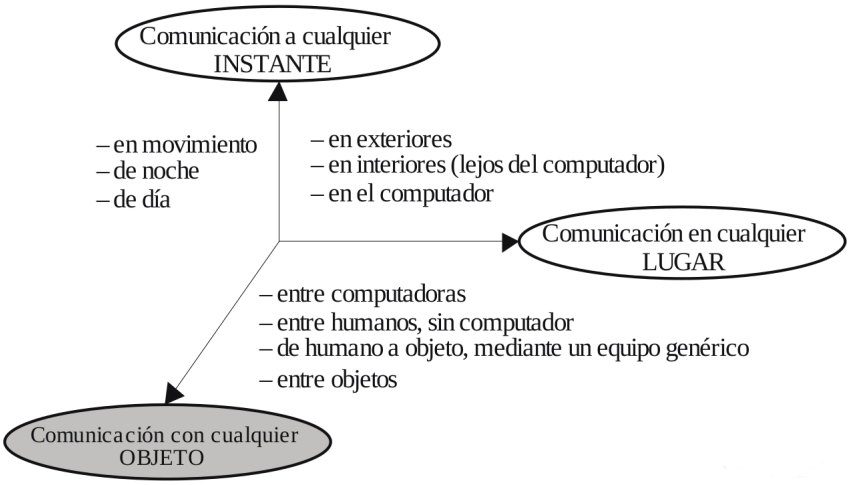

Figura 1: Nueva dimensión introducida por *IoT*

en el mundo de la información por uno o varios objetos virtuales y el objeto virtual puede existir sin tener asociado ningún objeto físico.

Un dispositivo es un pieza que tiene que poder comunicarse y además puede detectar, accionar y procesar datos. Los dispositivos recogen datos y los envían a través de la red para que se puedan procesar. Además dentro de sus capacidades está la de ejecutar operaciones en función de la información recibida.

Los objetos físicos se pueden representar en el mundo de la información por uno o varios objetos virtuales y los objetos virtuales pueden existir sin tener asociado ningún objeto del mundo físico. Los dispositivos, son piezas del sistema con capacidad de comunicación obligatoria y otras capacidades opcionales como de detección, accionamiento, almacenamiento o procesamiento de datos entre otras. Algunos dispositivos pueden ejecutar operaciones en función de los datos recibidos de otros dispositivos.

Según muestra la figura 2 los dispositivos se pueden comunicar con otros a través de una pasarela, por una red sin pasarela o directamente sin utilizar la red de comunicación. Existen otras formas de comunicación que pueden ser combinación de las anteriores.

Las redes actuales basadas en tecnologías convencionales como TCP/IP proporcionan fiabilidad y eficiencia y la infraestructura de *IoT* puede crearse sobre estas redes o sobre otras que se empiezan a implantar como NGN (Next Generation Networking).

Como se ha mencionado anteriormente los dispositivos *IoT* deben tener al menos capacidad de comunicación. En la figura 3 se puede ver cómo se pueden clasificar estos dispositivos.

- Dispositivo de transporte de datos: anexo a un objeto físico y que conecta al objeto físico con las redes de comunicación.
- Dispositivo de adquisición de datos: es un dispositivo de lectura y escritura con capacidad para interactuar con objetos físicos

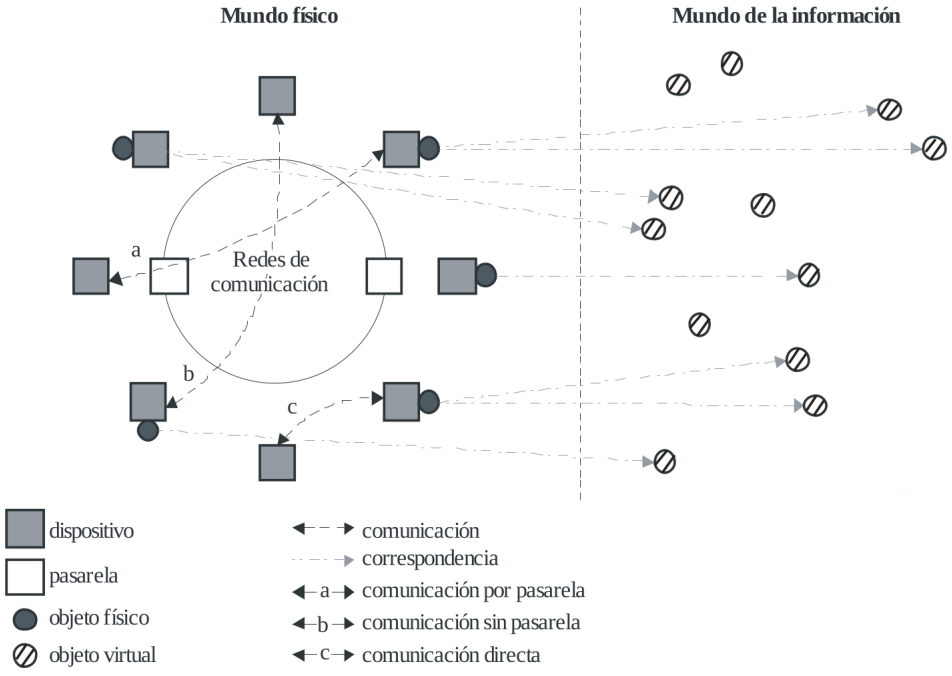

Figura 2: Descripción Técnica

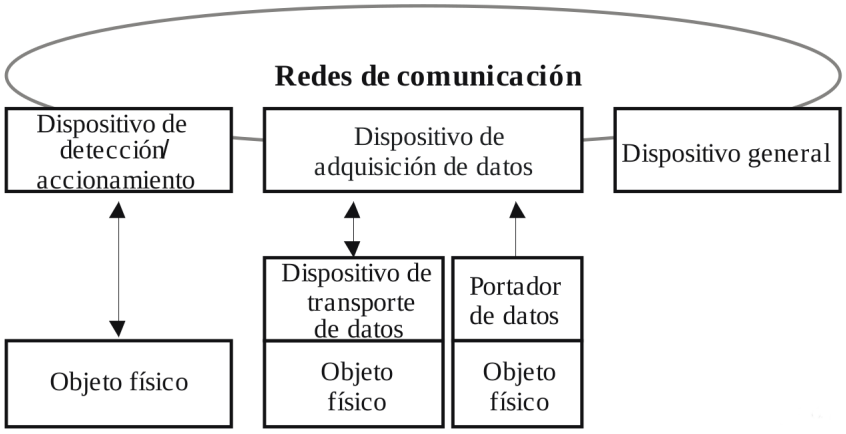

Figura 3: Tipos de dispositivos y su relación con objetos físicos

- Dispositivo de detección y accionamiento: detectan o miden información de su entorno convirtiendo estas en señales digitales que posteriormente puede transmitir de manera inalámbrica a otros dispositivos. Normalmente estos dispositivos forman una red y se comunican entre ellos.

- Dispositivo genérico: Es un dispositivo que cuenta con capacidades de procesamiento y comunicación y se comunica con las redes de manera inalámbrica. Se pueden usar en máquinas industriales, electrodomésticos o teléfonos inteligentes.

2.1.3. Características fundamentales

Las características fundamentales de *IoT* son las siguientes:

- Interconectividad: todos los *Objetos* pueden estar interconectados con la infraestructura global de la información y comunicación.

- Servicios relacionados con objetos: capacidad de ofrecer servicios relacionados con los *Objetos*

- Heterogeneidad: pueden interactuar con otros dispositivos basados en otras tecnologías a través de la red.

- Cambios dinámicos: el estado de los dispositivos pueden variar de manera dinámica así como el contexto.

- Escala enorme: el número de dispositivos que se puede llegar a gestionar puede ser mucho mayor al número de equipos que ahora están conectados a la red de Internet.

2.1.4. Modelo de Referencia IoT

El modelo de referencia de IoT consta de cuatro capas y de capacidades de gestión y seguridad que están relacionadas con estas capas como se puede ver en la figura 4.

Capa de Aplicación
En esta capa se encuentran las aplicaciones IoT.

Capa de soporte de de servicios y aplicaciones En esta capa se pueden encontrar dos grupos de de capacidades.

- capacidades de soporte genéricas: capacidades comunes que pueden utilizarlas diferentes aplicaciones de IoT como el procesamiento y almacenamiento de datos,

- capacidades de soporte específicas: capacidades que atienden necesidades particulares de diversas aplicaciones.

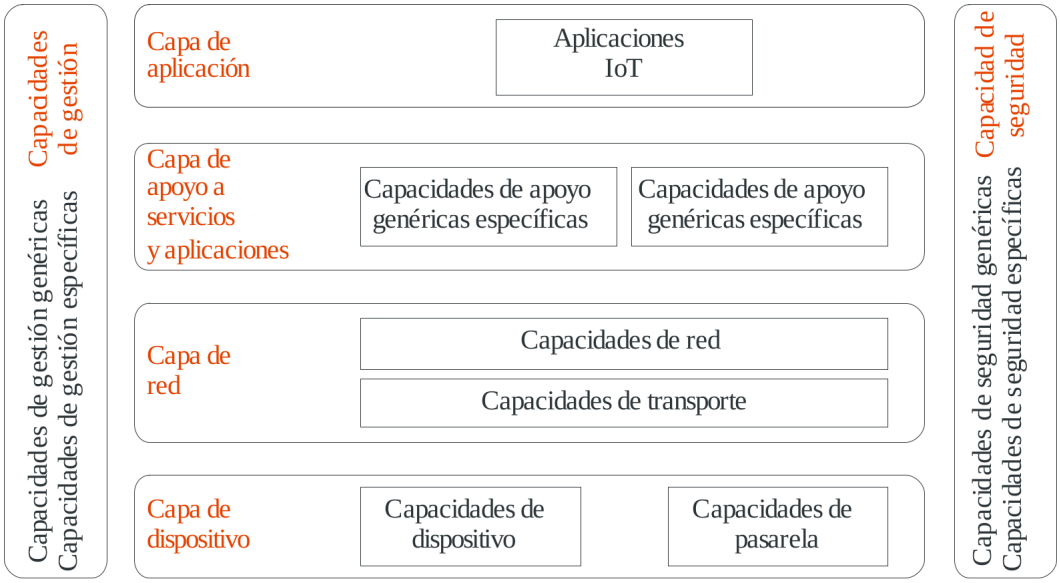

Figura 4: Modelo de referencia de IoT

Capa de red

Dos tipos de capacidad de red:

- capacidad de red: ofrecen funciones de control de la conectividad en la red,
- capacidad de transporte: se centran en ofrecer conectividad para el transporte de la información y datos específicos se servicios y aplicaciones IoT.

Capa de dispositivo

Las capacidades de los dispositivos pueden ser la interacción directa con la red de comunicaciones, la interacción indirecta con la red de comunicación o los dispositivos pueden construir redes de manera *ad-hoc* en algunas circunstancias.

Las capacidades de pasarela pueden ser el soporte de interfaces múltiples en la capa de dispositivo que se conectan a través de diferentes tecnologías alámbricas o inalámbricas como Bluetooth o Wi-Fi. En la capa de red las capacidades de pasarela se pueden comunicar por redes telefónicas como 2G, 3G o 4G o redes Ethernet entre otras.

2.2. Plataformas IoT

Existen plataformas que dan soporte a los *Objetos IoT* ya implementados y que soportan el protocolo MQTT. Entre ellos se puede encontrar Ubitots, HiveMQTT, ThingsBoard que son clientes de escritorio. Para dispositivos móviles también se pueden encontrar muchos clientes.

Estas plataformas son muy útiles porque evitan al desarrollador crear entornos web para mostrar los datos y este, puede dedicar sus esfuerzos a la implementación del *Backend* y el *Modelo de Negocio*.

2.3. Hardware

2.3.1. Arduino

Arduino es una plataforma de *software* y *hardware* bajo licencia GPL de GNU [14]. Con la gama de microcontroladores de Arduino se pueden crear proyectos complejos con un costo económico bajo ya que el hardware se puede adquirir en infinidad de sitios y son de bajo coste. Dispone de un entorno de desarrollo de fácil manejo, con el cual podemos programar toda la gama de microcontroladores oficiales o sus derivados. También dispone de un sinfín de bibliotecas, Open Source y privativas, en la red a disposición de los desarrolladores como la biblioteca de Adafruit para los sensores DHT.

La de gama se microcontroladores de Arduino están íntimamente ligados al IoT ya que permiten crear esos *Objetos* que pueden obtener información de su entorno o accionar elementos y establecer comunicación con la red, de manera inalámbrica habitualmente. Hay que tener en cuenta que el propósito de estos dispositivos es educativo, esto quiere decir que normalmente no son usados para proyectos profesionales pero su uso está muy extendido para proyectos para todo tipo de proyectos y existe una comunidad muy grande y activa que facilita el desarrollo de cualquier aplicación basada en Arduino.

La elección del uso del modelo Nano para este trabajo ha sido porque posee todo lo necesario para su desarrollo, su bajo coste además de su reducido tamaño.

Se ha empleado junto al sensor DHT-22 y al módulo Bluetooth HC-05 para la que la comunicación fuera inalámbrica (Figura 37) o junto al relé SRD-05VDC-SL-C con comunicación mediante el puerto serie, a través de un cable USB-mini_USB (Figura 42).

El mayor inconveniente en el empleo o implementación de estos dispositivos es que por sí mismos no poseen conectividad inalámbrica sino que hay que conseguirlo mediante módulos separados.

2.3.2. ESP8266

Qué es Es un chip de bajo coste de Wi-Fi con una pila TCP/IP completa y un microcontrolador [23]. Es un *SOC, System On a Chip*, que describe la tendencia de fabricación más reciente de usar tecnologías de fabricación que integran todos o gran parte de lo módulos que componen dispositivos de este tipo.

En este proyecto se han usado dos versiones de ESP8266, el primero el ESP-01 y el segundo ha sido ESP8266 NodeMcu LoLin [24]. El primer modelo no llegó a funcionar correctamente como se explica en 6.1.1 y por ello se recurrió a un segundo modelo. Este segundo modelo es LoLin, cuenta con una placa de desarrollo, NodeMCU, con pines a los cual podemos conectar de manera muy simple los sensores o actuadores que vayamos a utilizar. Podemos observar las conexiones del ESP8266 con el sensor DHT-22 (figura37, con DHT-22 y relé (figura38 y con el caudalímetro (figura39.

2.3.3. Rapsberry Pi

Raspberry Pi es una computadora de bajo coste con Arquitectura ARM. En sus últimas versiones cuenta con Wi-Fi y Bluetooth lo que posibilita las conexiones inalámbricas. Además de las prestaciones normales de una computadora hay que destacar que dispone de pines GPIO (General Purpose Input/Output) con el cual controlar dispositivos.

Existen diferentes Sistemas Operativos destinados a este dispositivo casi todos basados en *GNU-Linux*. Algunos de estos son:

- Raspbian: Basado en Debian. Se puede instalar tanto con soporte para entorno de ventanas o sin ellas. No pertenece a la *Raspberry Pi Fundation*. El *SO* lo desarrollan un pequeño grupo de entusiastas de *Raspberr Pi*.
- Ubuntu Core: Es un derivado de Debian, usa el mismo kernel que el Ubuntu tradicional pero está diseñado y pensado para las aplicaciones IoT y la seguridad, el planteamiento es confinar las aplicaciones y sus dependencias en paquetes aislados unos de otros. Además el SO está muy aligerado.
- Fedora: Versión de Fedora desarrollada para arquitectura ARM.
- Arch Linux: Para los amantes de archlinux existe esta versión para arquitectura ARM encaminada hacia la simplicidad, ligereza, y el control total del usuario. No dispone de entorno de ventanas.

También hay una versión, ¡gratuita!, de Microsoft de Windows 10.

Integración en el Proyecto

Solamente se ha usado para la primera versión del proyecto, *prototipo 1*. El *SO* utilizado ha sido *Raspbian Lite* y sobre él se han instalado los servicios necesarios para el desarrollo. La misión fundamental, dentro sistema, ha sido la de hacer de centralizador entre los sensores y actuadores y la salida a Internet. Con más detalle la Raspberry Pi tenía un *recolector de datos* corriendo como servicio del sistema que era controlado mediante *systemctl*, este servicio se encargaba de esperar a que le llegasen los datos de los diferentes sensores. Una vez que le llegaba un dato, cada 5 minutos, los enviaba al servidor de base de datos mediante el servicio de colas *RabbitMQ* 2.4.6. También disponía de otro servicio, este mediante *sockets*, que se encargaba de comunicar con los actuadores cuando le llegaba la señal desde el entorno web desarrollado para prestar este servicio. Se puede ver cual es la función del dispositivo Raspberry Pi en el sistema en la topología de la arquitectura del prototipo 1 (figura 17).

2.3.4. DHT-22

Es un sensor capacitivo [26] que mide la la temperatura y humedad relativa del ambiente y devuelve una señal digital con la medición. Sus características [25] más destacadas son:

- rango de humedad de 0-100 % con un error de ± (2-5 %),
- rango de temperatura -40 a 80°C (±0.5°C),

- lectura de datos cada 2 segundos como máximo.

2.3.5. HC-05

Es un módulo Bluetooth que usa el protocolo Bluetooth SPP (Serial Port Protocol) y está diseñado para que la configuración inalámbrica sea transparente para el usuario al utilizar el protocolo del puerto serie [27]. Esto simplifica la programación del microcontrolador que utilice este módulo.

2.3.6. SRD-05VDC-SL-C

El relé es un dispositivo electromagnético que sirve como interruptor en circuitos eléctricos. Este modelo soporta un máximo de intensidad de 10A [29].

2.3.7. YF S402

Es un caudalímetro que consta de una turbina cuya salida es en pulsos digitales. Los pulsos son proporcionales al paso del caudal del líquido [28].

2.4. Servicios

2.4.1. MySQL

MySQL es un sistema gestor de bases de datos relacional con dos tipos de licencia, una con GNU/GPL y otra de tipo privativa.

Está compuesta por las siguientes partes:

- Administrador de almacenamiento: controla el acceso a la información de la base de datos almacenada en el disco,
- procesador de consultas: recibe las peticiones de consultas o actualización y busca la mejor manera de ejecutarlas,
- gestor de transacciones: conserva la integridad de la base de datos a través de las propiedades, *ACID*, Atomomicidad, Consistencia, Aislamiento (Isolation), Persistencia (Durability)

Es uno de los gestores más utilizados y su primera versión fue lanzada en 1999, contando con una larga trayectoria, lo que promueve la fiabilidad del producto.

En este proyecto se ha usado este gestor para administrar la base de datos ya que su uso está muy extendido y cuenta con una gran comunidad. La documentación es muy extensa y completa facilitando el desarrollo de la aplicación.

2.4.2. MQTT

MQTT (Message Queue Telemetry Transport) es un protocolo de comunicación para dispositivos IoT que está construido sobre la pila TCP/IP y está considerado como el estándar en las comunicaciones de IoT al permitir la comunicación de los dispositivos con los servicios de *backend*.

MQTT fue desarrollado por IBM a finales de los 90 para conectar sensores de los oleoductos con los satélites. Es un protocolo de mensajería asíncrona que utiliza el patrón de publicación/suscripción [30].

Ventajas que ofrece MQTT como protocolo de comunicación para IoT:

- Es un protocolo liviano lo que le permite implementarse en hardware de dispositivos altamente limitados y en redes con ancho de banda de alta latencia y limitado,
- es flexible haciendo que soporte varios escenarios de aplicaciones para dispositivos y servicios IoT.

El Modelo de publicación y suscripción

El protocolo MQTT define los tipos de entidades en la red: un intermediario y un número de clientes. El intermediario es un servidor que recibe todos los mensajes de los cliente y luego los redirige a clientes de destinos relevantes. Un cliente es cualquier cosa que pueda interactuar con el *broker* para enviar y recibir mensajes [31], puede ser un sensor *Objeto* IoT o una aplicación del *backend* del servicio donde se procesan los datos de IoT.

Los pasos que se sigue el en envío y recepción de un mensaje:

1. El cliente se conecta con el *broker* y se puede suscribir a cualquier *topic* de mensajes del *broker*,
2. el cliente publica el mensaje en un *topic* y envía el mensaje y el *topic* al *broker*,
3. el *broker* redirige el mensaje a todos los clientes que están suscritos a ese *topic*.

Como los mensajes MQTT están organizados por temas, el desarrollador puede especificar que ciertos clientes sólo puedan interactuar con determinados mensajes. Por ejemplo, los sensores publicarán las lecturas sobre el *topic* sensor_data y se suscribirán al *topic* config_change. Las aplicaciones que procesan los datos de los sensores y los guardan en una base de datos se suscribirán al tema sensor_data. En una consola de administración se pueden escribir comandos de configuración de los sensores y publicarlos en el *topic* config_change como se puede ver en la figura 5.

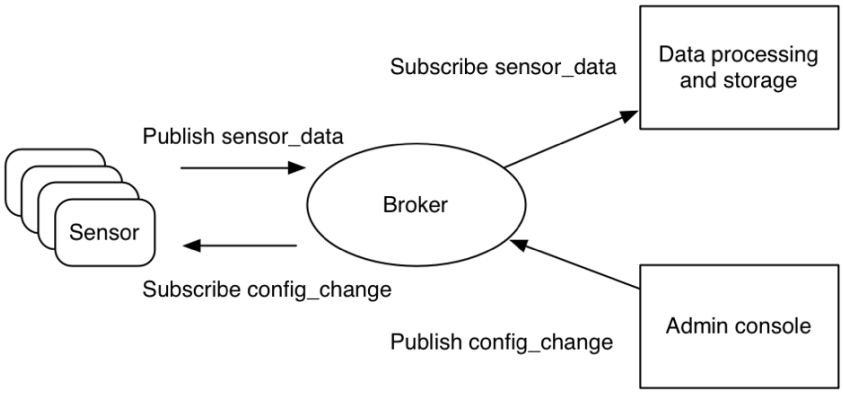

Figura 5: Modelo Publicación Suscripción

Otro ejemplo de interacción del modelo de publicación/suscripción puede ser el de los sensores publicando mensajes, cada uno con su propio *topic*, y enviándolos al *broker*, este los redirige a los clientes que estén suscritos a los *topic*. A su vez puede haber clientes que tengan ambos roles, caso del *backend* y por lo tanto envíen mensajes a otros clientes, dispositivos, haciendo que accionen algún elemento (figura 6).

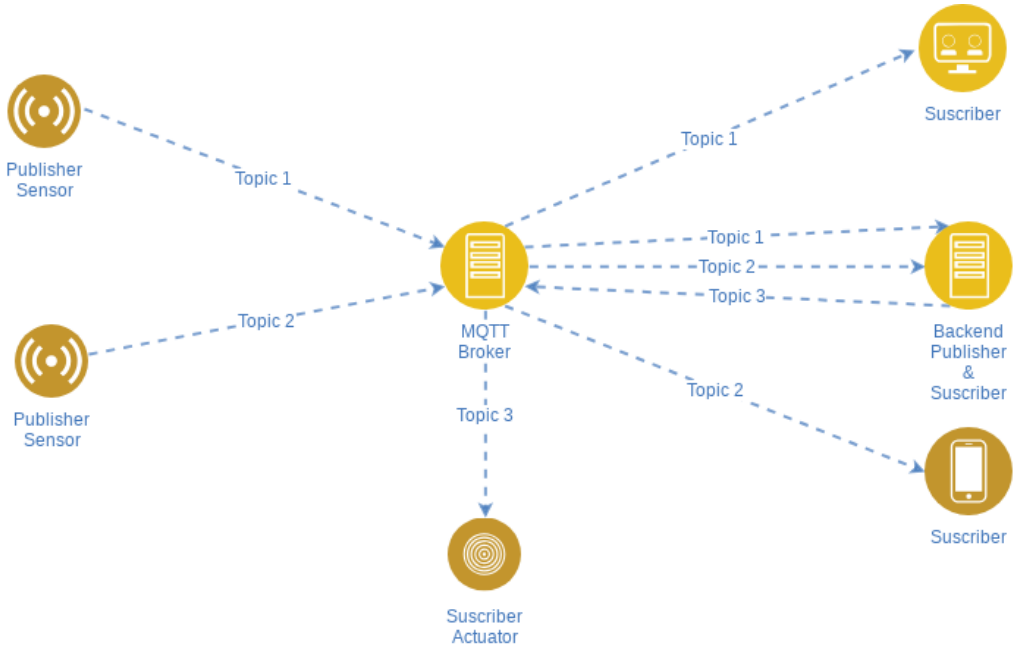

Figura 6: Modelo Publicación Suscripción

2.4.3. Mosquitto

Eclipse Mosquitto es un *Broker* de mensajes que implementa el protocolo *MQTT* y es *Open Source*. *Mosquitto* es muy ligero y su uso es adecuado para dispositivos con poco poder de procesamiento o en servidores de muchos recursos. Su

configuración es sencilla y su funcionamiento se ha detallado en la sección anterior ya que es una implementación del *broker* MQTT.

2.4.4. Sistema Gestor de Versiones

Como sistema gestor de versiones se ha usado *Git* que es un software diseñado por Linus Torvalds y con licencia *GNU/GPL*.

2.4.5. No-IP

El servicio *no-ip* se ha utilizado en este proyecto para poder conectar con la Raspberry Pi desde Internet. La IP de este dispositivo es dinámica y por lo tanto podría ser modificada en cualquier momento. Para resolver este problema *no-ip* proporciona una dirección con el cual se puede acceder siempre al dispositivo, para ello hay que instalar un programa en la Raspberry Pi que cada cierto tiempo se conecta con el servidor de *no-ip* y le da su IP.

2.4.6. RabbitMQ

RabbitMQ es un *Broker de mensajes* que sirve para el intercambio de datos entre procesos, aplicaciones y servidores (figura7)[15].

Además permite tener una capa de abstracción en las comunicaciones entre diferentes máquinas sin necesidad de gestionar las comunicaciones. En el caso más simple, los datos los crea un *Productor* y los envía al *Broker* que gestiona la cola y un *Consumidor* se suscribe a la cola y recibe los datos que el productor deja en la cola. Una vez los datos han sido *consumidos* dejan de estar en la cola. La complejidad de este sencillo caso puede aumentar, utilizando más de una cola, con consumidores para cada cola o también que la asignación de la cola se haga por el contenido del mensaje, de manera selectiva[16].

Figura 7: Diagrama funcionamiento RabbitMQ

Para las comunicaciones utiliza el protocolo de *AMQP* que se distribuye bajo licencia de *Open Source*[17] que implementa el estándar AMQP.

2.4.7. Flask

Flask es un *Micro Framework* concebido para facilitar el desarrollo de Aplicaciones Web en Python bajo el patrón MVC [4].

En este proyecto se ha usado en el prototipo 1 en la parte de Aplicación Web y en el prototipo 2 para la implementación de la APIRest.

2.5. Bibliotecas

2.5.1. gh-tools

gh-tools es una bilbioteca propia desarrollada para su uso en el proyecto. Es instalable vía pip y se ha creado en un paquete Python[19].

2.6. Entorno de Desarrollo Integral (IDE)

Como *IDE* principalmente he usado PyCharm con una licencia de estudiante que proporciona la empresa que desarrolla el producto, JET BRAINS, que además cuenta con una amplia gama de productos para el lenguaje que necesites con licencia de estudiante. Es un entorno de desarrollo que tiene las características de otros como, resaltado dela sintaxis, auto completado, etc, pero tiene otras características muy potentes como la integración continua o el control de versiones, conexión con base de datos Otra función a destacar es la creación automática del *Modelo entidad relación* o el diagrama de clases.

Otro *IDE* utilizado en el proyecto para programar los microcontroladores, tanto los Arduino como los ESP8266, ha sido el de Arduino en su versión de Escritorio. Al ser un *Software Libre* permite la inclusión de infinidad de *bibliotecas* que facilitan la programación.

Vim es un editor de textos muy configurable con licencia GPL. Dispone con *Plugins* instalables vía manejadores de instalación como Vundle.

3. Planificación

3.1. Metodología de Desarrollo Ágil basada en Scrum

Las metodologías ágiles para el desarrollo de software se vienen imponiendo frente al desarrollo mediante las técnicas tradicionales. En este proyecto se va a usar un método adaptado a las condiciones de desarrollo del proyecto basado en la metodología de Scrum. Se elige Scrum ya que el alcance del proyecto no está perfectamente definido, interesa añadir funcionalidades gradualmente y si fuera necesario cambiar requisitos entre iteraciones. Aunque se recomienda que al menos el equipo de desarrollo sea de tres miembros y aunque en este caso solamente sea uno se decide usar Scrum por las ventajas que conlleva como por ejemplo el tener una interacción más cercana con el Product owner lo que redunda en un software que se ajusta más a lo que quiere el cliente.

La metodología Scrum no establece el análisis ni el diseño en sus especificaciones pero en esta metodología adaptada sí se ha realizado aunque no en toda su envergadura. De esta manera, se añaden los diagramas que se creen más importantes para un entendimiento a nivel técnico del proyecto.

Como se ha mencionado anteriormente el *equipo de desarrollo* es un único desarrollador y la responsabilidad de *Product owner* también recae en la misma persona. No es conveniente que una persona adopte todos los roles en Scrum y desde luego, no es lo que proponen las guías de desarrollo ágil.

A continuación se van a detallar los elementos que conforman Scrum, y sí están presentes, en la memoria del proyecto.

Se establece un *Product backlog* con todas las *Historias de usuario* que se van a desarrollar priorizadas por el *Product owner*. En cada iteración, *Sprint*, se implementarán una o varias historias de usuario que vendrán recogidas en un *Sprint backlog*. Las iteraciones serán de una duración de dos semanas.

La herramienta Taiga permite tener todo centralizado en una misma aplicación además de en unas hojas de cálculo, es por este motivo que se eligió para el seguimiento de este proyecto.

Los *stakeholders* (Tablas 1, 2 y 3) son cualquier organización o persona que esté relacionado con el desarrollo del proyecto o vaya a ser usuario de la aplicación. En este caso se puede considerar como *stakeholders*:

Organización	Universidad de Valladolid
Dirección	Plaza de Santa Cruz. 8
E-mail	admon.secretaria.general@uva.es

Tabla 1: Stakeholder UVa

Participante	Arturo González Escribano
Organización	Universidad de Valladolid
Rol	Tutor TFG
E-mail	arturo@infor.uva.es

Tabla 2: Stakeholder Tutor TFG

Participante	Kave Heidarieh Sorosh
Organización	Universidad de Valladolid
Rol	Desarrollador, Product Owner
E-mail	kave.heidarieh@alumnos.uva.es

Tabla 3: Stakeholder desarrollador, product owner

3.2. Product Backlog

En el *Product backlog* se incluyen una lista de funcionalidades, productos y acciones que forman el artefacto que se va a construir. El responsable del mantenimiento de este listado es el *Product owner* [2].

HU	Como	Me gustaría	Prioridad	Riesgo	Puntos	Estado
1	Product owner	Controlar un cultivo a través de una web	Alta	Alto	8	Terminado
2	Product owner	Saber qué dispositivos son necesarios	Alta	Alto	5	Terminado
3	Product owner	Leer los datos de los sensores	Alta	Bajo	3	Terminado
4	Product owner	Que los datos de lectura de los sensores sean persistentes	Media	Bajo	8	Terminado
5	Product owner	Ver un sketch de la web	Alta	Bajo	2	Terminado
6	Desarrollador	Saber qué tecnologías y cómo usarlas para desarrollar una aplicación web	Alta	Alto	8	Terminado
7	Desarrollador	Establecer la estructura de la web con las vistas	Alta	Medio	13	Terminado
8	Product owner	Ver una gráfica con la temperatura y otra con la humedad de los sensores	Media	Bajo	21	Terminado
9	Desarrollador	Dar seguridad a la base de datos	Media	Alto	8	Terminado
10	Product owner	Que los sensores sean inalámbricos	Media	Medio	8	Terminado
11	Product owner	Poder accionar un relé desde la web para accionar el riego	Alta	Medio	8	Terminado
12	Product owner	Saber si el riego está accionado	Media	Alto	8	Terminado
13	Product owner	Programar riego mediante unos relojes en la vista de riego	Media	Bajo	8	Terminado
14	Product owner	Saber cómo adaptar aplicación a modelo IoT	Alta	Alto	13	Terminado
15	Desarrollador	Utilizar dispositivos ESP8266 y protocolo MQTT	Alta	Alto	21	Terminado
16	Desarrollador	Establecer paradigma de Programación Orientada a Objetos en la aplicación	Alta	Bajo	8	Terminado
17	Product owner	Que otra aplicaciones puedan utilizar esta aplicación a través de una APIRest	Alta	Alto	5	Terminado
18	Product owner	Controlar el consumo de agua	Alta	Medio	5	Terminado
19	Product owner	Poder programar riegos	Alta	Medio	5	Terminado
20	Desarrollador	Redactar la memoria	Alta	Alto	34	En curso

Tabla 4: Product backlog

3.3. Sprints

3.3.1. Sprint 1

Sprint	Fecha inicio	Fecha final
1	16/04/2018	29/04/2018

Sprint Backlog

Historia de Usuario		
Número: 1		Usuario: Product owner
Prioridad en negocio: Alta		Riesgo en desarrollo: Alto
Nombre	Control de un cultivo a través de web	
Descripción	Quiero poder conectarme a una web y ver desde un móvil o un ordenador datos del cultivo como la temperatura y la humedad ambiental y poder abrir el riego. Se van a buscar soluciones técnicas para ver cómo se puede implementar un artefacto	
Validación	Se le presenta al product owner que tecnologías se van a utilizar y debe aceptarlas	

Tabla 5: Historia de usuario número 1

Historia de Usuario		
Número: 2		Usuario: Product owner
Prioridad en negocio: Alta		Riesgo en desarrollo: Alto
Nombre	Dispositivos necesarios	
Descripción	El product owner quiere saber que dispositivos se pueden necesitar para poder desarrollar el artefacto que ha pedido	
Validación	Se le enumeran los dispositivos que creemos necesarios para desarrollar el producto y debe aceptar	

Tabla 6: Historia de usuario número 2

Historia de Usuario		
Número: 3		Usuario: Product owner
Prioridad en negocio: Alta		Riesgo en desarrollo: Bajo
Nombre	Envío de datos de los sensores	
Descripción	Hacer una demo donde se pueda ver cómo están recogiendo datos los sensores y visualizarlos a través de la monitor del IDE de Arduino o de la terminal	
Validación	El product owner puede ver la temperatura y humedad ambiental a través de la pantalla y comprobar que cambia al calentar el sensor	

Tabla 7: Historia de usuario número 3

Sprint Backlog

	Time (esti)	Time (spent)	Time (left)	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Historia Usuario 1																	
Buscar tecnologías para desarrollo aplicación web y ver su funcionamiento	24	27	0	4	4	2	4		4	4	5						
Historia Usuario 2																	
Buscar qué dispositivos son necesarios para implementar el sistema que el product owner quiere y ver su funcionamiento		17	0									5	4	4	3	1	
Historia usuario 3																	
Investigar sobre los dispositivos	4	4	4														
Adquirir los dispositivos necesarios	4	4	4														
Investigar cómo se programan los microcontroladores Arduino	5	5	5														
Cablear dispositivos	8	8	8														
Pogramar Arduinos para lectura de datos por puerto serial	6	6	6														
	51	71	27	4	4	2	4	0	4	4	5	5	4	4	3	1	0
Daily burnout	3	23		4	4	4	4	0	4	4	4	4	4	4	4	4	0
Total time left (from estimate)			51	47	43	39	35	35	31	27	23	19	15	11	7	3	3
Total time left (from spent)			71	67	63	61	57	57	53	49	44	39	35	31	28	27	27

Tabla 8: Burnout sprint 1

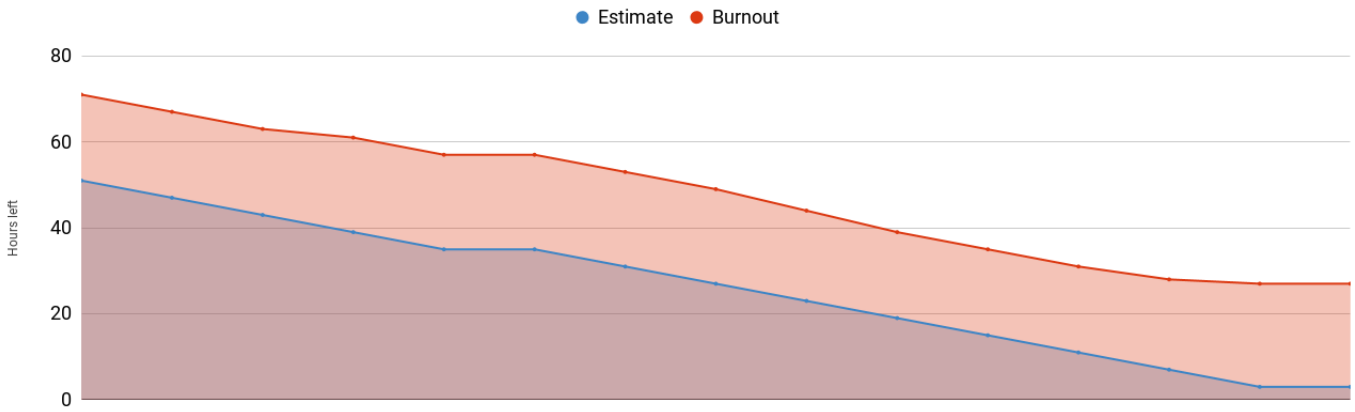

Figura 8: Burnout sprint 1

3.3.2. Sprint 2

Sprint	Fecha inicio	Fecha final
2	30/04/2018	13/05/2018

Historia de Usuario	
Número: 4	Usuario: Product owner
Prioridad en negocio: Media	Riesgo en desarrollo: Bajo
Nombre	Guardar las mediciones en una base de datos
Descripción	Almacenar las mediciones que recogen los sensores y almacenarlos en una base de datos e incorporar otros datos que son útiles
Validación	Se le muestra al product owner las mediciones que se han ido recogiendo en un plazo de tiempo

Tabla 9: Historia de usuario número 4

Sprint backlog

	Time (esti)	Time (spent)	Time (left)	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Historia usuario 3																	
Investigar sobre los dispositivos	4	17	0	4	4	5	4										
Adquirir los dispositivos necesarios	4	6	0			1	1	1	3								
Investigar cómo se programan los microcontroladores Arduino	5	6	0							6							
Cablear dispositivos	8	8	4									4					
Pogramar Arduinos para lectura de datos por puerto serial	6	8	0										8				
Historia usuario 4																	
Configurar máquina virtual con IP fija	4	4	1												3		
Instalar servcios necesarios	4	4	2												2		
Configurar MyQSL con usuario	3	3	0													3	
Diseñar base de datos	4	4	0														4
Crear base de datos con tablas	2	2	2														
Instalar SO en Raspberry Pi	1	1	1														
Configurar Raspberry Pi	1	1	1														
Implementar programa que recoge los datos de puerto serie y los inserta en la base de datos	4	4	4														
Crear base de datos con tablas	2	2	2														
Instalar SO en Raspberry Pi	1	1	1														
Configurar Raspberry Pi	1	1	1														
Implementar programa que recoge los datos de puerto serie y los inserta en la base de datos	4	4	4														
	58	76	23	4	4	6	5	1	3	6	0	4	8	0	5	3	4
Daily burnout	6	24		4	4	4	4	4	4	4	0	4	4	4	4	4	4
Total time left (from estimate)			58	54	50	46	42	38	34	30	30	26	22	18	14	10	6
Total time left (from spent)			76	72	68	62	57	56	53	47	47	43	35	35	30	27	23

Tabla 10: Burnout sprint 2

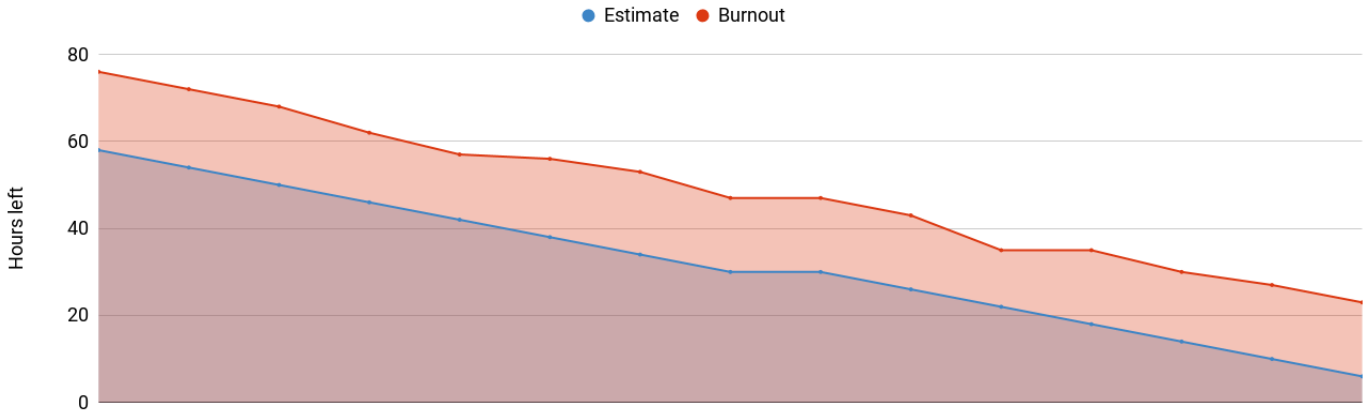

Figura 9: Burnout sprint 2

3.3.3. Sprint 3

Sprint	Fecha inicio	Fecha final
3	14/05/2018	27/05/2018

Historia de Usuario		
Número: 5		Usuario: Product owner
Prioridad en negocio: Alta		Riesgo en desarrollo: Bajo
Nombre	Sketch de la web	
Descripción	Quiero ver gráficamente cómo sería la web antes de empezar con la implemetación	
Validación	Se le muestra al product owner los sketchs de cómo se ha diseñado la web. Le gusta el modelo y se puede empezar con su implementación	

Tabla 11: Historia de usuario número 5

Historia de Usuario		
Número: 6		Usuario: Desarrollado
Prioridad en negocio: Alta		Riesgo en desarrollo: Alto
Nombre	¿Cómo desarrollar la aplicación web?	
Descripción	Buscar información sobre desarrollo de aplicaciones web y tecnologías a usar	
Validación	Se le explica al product owner qué tecnologías se van a usar y por qué	

Tabla 12: Historia de usuario número 6

	Time (esti)	Time (spent)	Time (left)	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Historia Usuario 4																	
Diseño de la base de datos	4	4	2	2													
Crear base de datos con tablas	3	4	0	4													
Instalar SO en Raspberry Pi	2	3	0		3												
Configurar Raspberry Pi	1	2	0		2												
Implentar programa que recoge los datos de puerto serie y los inserta en la base de datos	6	8	0		4	4											
Historia Usuario 5																	
Ralización de un sketch del sitio web	4	8	0				5	3									
Historia Usuario 6																	
Investigación funcionamiento Flask	10	36	0					2	2	8	8	8		8			
Investigación aplicación web mediante Flask	20	32	0									2		2	10	10	8
	50	97	2	4	9	4	5	5	2	8	8	10	0	10	10	10	8
Daily burnout	2	49		4	4	4	4	4	4	4	4	4	0	4	4	0	4
Total time left (from estimate)			50	46	42	38	34	30	26	22	18	14	14	10	6	6	2
Total time left (from spent)			97	91	82	78	73	68	66	58	50	40	40	30	20	10	2

Tabla 13: Burnout sprint 3

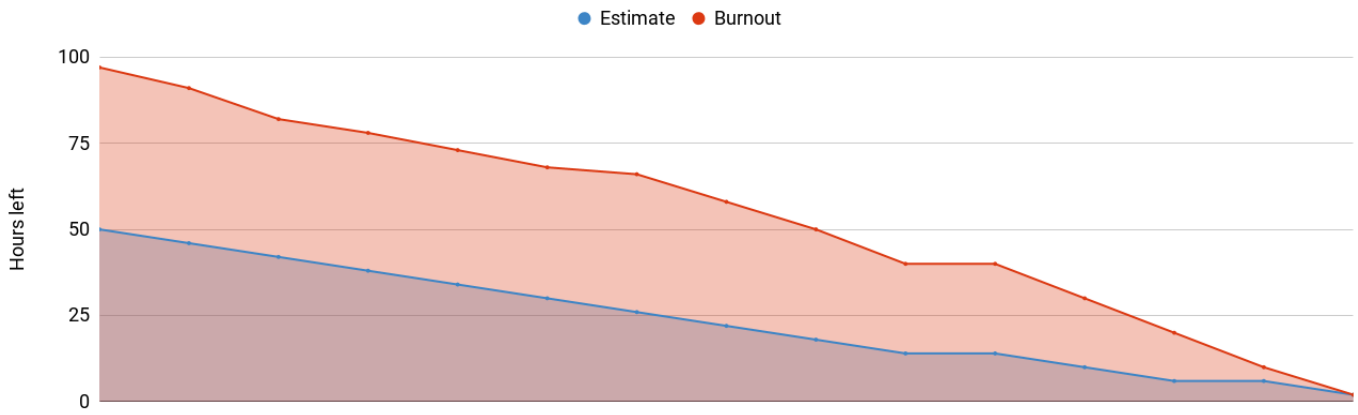

Figura 10: Burnout sprint 3

3.3.4. Sprint 4

Sprint	Fecha inicio	Fecha final
4	28/05/2018	10/06/2018

Historia de Usuario		
Número: 7		Usuario: Desarrollador
Prioridad en negocio: Alta		Riesgo en desarrollo: Medio
Nombre	Desarrollo de estructura de la web	
Descripción	Se desarrolla la estructura de la web sin funcionalidades solo con la vista principal	
Validación	Se le muestra a product owner como es la estructura de la web, como se pueden ir añadiendo vistas y los colores que se han elegido.	

Tabla 14: Historia de usuario número 7

	Time (esti)	Time (spent)	Time (left)	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Historia Usuario 7																	
Busqueda de plantilla para dashboard	8	10	0	6	4												
Adaptación de código html a entorno Flask	16	21	0		5	6		6	4								
Creación de plantilla base html en Flask	12	24	0					2	3	6	5	8					
Quitar errores de la plantilla	8	23	0											5	8	7	3
Implementar vista de temperatura	8	8	8														
Implementar vista de humedad	2	2	2														
	54	88	10	6	9	6	0	8	7	6	5	8	0	5	8	7	3
Daily burnout	6	40		4	4	4	0	4	4	4	4	4	0	4	4	4	4
Total time left (from estimate)			54	48	44	40	40	36	32	28	24	20	20	16	12	8	4
Total time left (from spent)			88	82	73	67	67	59	52	46	41	33	33	28	20	13	10

Tabla 15: Burnout 4

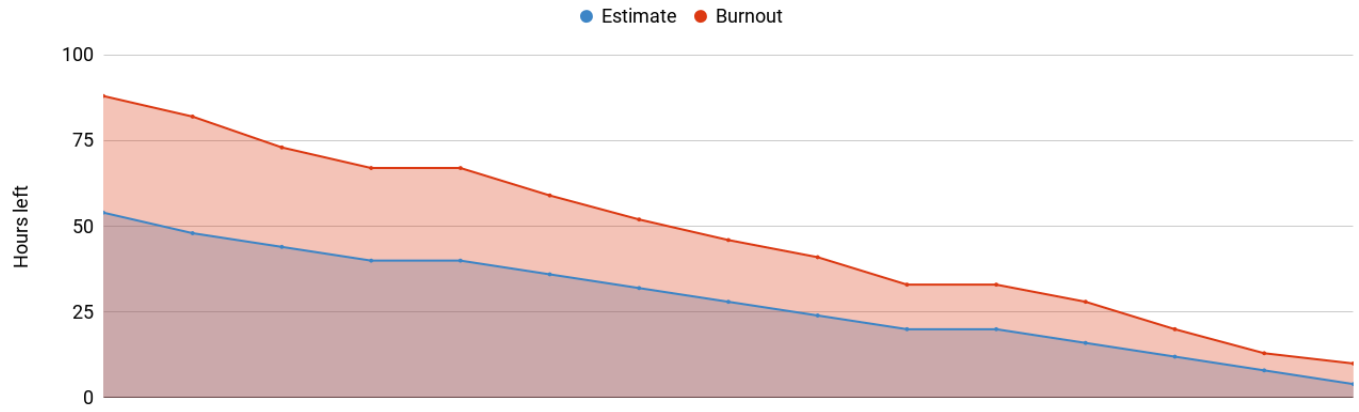

Figura 11: Burnout sprint 4

3.3.5. Sprint 5

Sprint	Fecha inicio	Fecha final
5	11/06/2018	24/06/2018

Historia de Usuario		
Número: 8		Usuario: Product owner
Prioridad en negocio: Media		Riesgo en desarrollo: Bajo
Nombre	Vista con una gráfica de la temperatura y la humedad	
Descripción	Se desarrolla dos vistas que muestren las mediciones de un intervalo de 24 horas en forma de gráfica, una con las mediciones de la temperatura y otra con las mediciones de la humedad.	
Validación	El producto owner comprueba que se muestran los datos de un día en las dos vistas y cómo, la temperatura es más alta en las horas diurnas que en las nocturnas.	

Tabla 16: Historia de usuario número 8

	Time (esti)	Time (spent)	Time (left)	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Historia Usuario 7																	
Implementar vista de temperatura	8	11	0	5		4	2										
Implementar vista de humedad	2	2	0				2										
Historia Usuario 8																	
Implementar estructura de la aplicación web	8	8	4				4										
Lanzar servidor con la aplicación web	4	14	0					6	6	2							
Implementar código en la aplicación web para haga petición a la base de datos y devuelva los datos de las últimas 24 de temperatura	4	7	0						2	5							
Implementar código en la aplicación web para haga petición a la base de datos y devuelva los datos de las últimas 24 de humedad	1	1	0								1						
Investigar cómo pasar datos de la aplicación web a las vistas	2	8	0								5	3					
Recoger los datos en las vistas	1	2	0									2					
Investigar formas se mostrar gráficas en al aplicación web	4	5	0										5				
Mostrar en la vista temperatura la gráfica de la temperatura	6	15	0											4		8	3
Mostrar en la vista temperatura la gráfica de la humedad	1	1	0														1
	41	74	4	5	0	4	8	6	8	7	6	5	5	4	0	8	4
Daily burnout	-7	26		4	0	4	4	4	4	4	4	4	4	4	0	4	4
Total time left (from estimate)			41	36	36	32	28	24	20	16	12	8	4	0	0	-4	-8
Total time left (from spent)			74	69	69	65	57	51	43	36	30	25	20	16	16	8	4

Tabla 17: Burnout 5

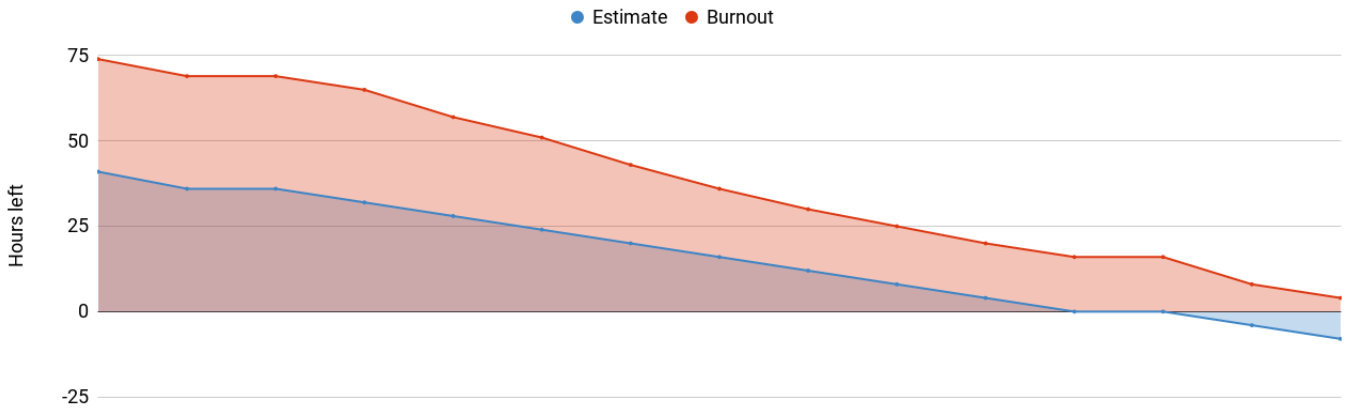

Figura 12: Burnout sprint 5

3.3.6. Sprint 6

Sprint	Fecha inicio	Fecha final
6	25/06/2018	8/07/2018

Historia de Usuario	
Número: 9	Usuario: Desarrollador
Prioridad en negocio: Media	Riesgo en desarrollo: Alto
Nombre	Seguridad base de datos
Descripción	Evitar que un usuario del gestor de base de datos pueda acceder desde fuera del servidor
Validación	El desarrollador comprueba que no se pueden hacer inserciones con el usuario anterior si se trata de hacer desde una máquina que no sea el servidor donde está el gestor de base de datos.

Tabla 18: Historia de usuario número 9

Historia de Usuario	
Número: 10	Usuario: Product owner
Prioridad en negocio: Media	Riesgo en desarrollo: Medio
Nombre	Sensor inalámbrico
Descripción	Quiero que los sensor de temperatura y humedad sean inalámbricos y que se conecten con la Raspberry Pi por Bluetooth
Validación	El product owner comprueba que los sensores ahora son inalámbricos

Tabla 19: Historia de usuario número 10

	Time (esti)	Time (spent)	Time (left)	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Historia Usuario 9																	
Investigar soluciones	8	8	4	4													
Invesitigar RabbitMQ	4	6	0	2	4												
Instalar y configurar el servicio en la misma máquina que esta el servidor de base de datos	4	6	0		2	4											
Instalar y configurar servicio en la Raspberry Pi	2	2	1			1											
Adaptar programa que recibe datos de los sensores y hace las inserciones en la base de datos para que envíes los datos por la cola	4	4	0					4									
Implementar un programa que gestione los datos recibidos en la cola y haga las insercciones	2	2	0					2									
Quitar privilegios al usuario del gestor de base de datos para que no pueda acceder solo desde la misma máquina	1	1	0							1							
Historia Usuario 10																	
Investigar funcionamiento de módulo bluetooth HC-05 para Arduino	4	5	0							4	1						
Conectar mediante cable Arduino con HC-05 y sensor DHT	3	4	0							1	3						
Configurar conexión Bluetooth de Raspberry Pi para poder emparejarlo con el módulo HC-05 conectado a Arduino	4	8	0									4	4				
Programar Arduino con la nueva rutina	2	2	0													2	
Modificar programa recolector de datos que reside en la Raspberry Pi	4	9	0													4	5
	42	57	5	6	6	5	0	6	0	6	4	4	4	0	0	6	5
Daily burnout	2	17		4	4	4	0	4	0	4	4	4	4	0	0	4	4
Total time left (from estimate)			42	36	32	28	28	24	24	20	16	12	8	8	8	4	0
Total time left (from spent)			57	51	45	40	40	34	34	28	24	20	16	16	16	10	5

Tabla 20: Burount sprint 6

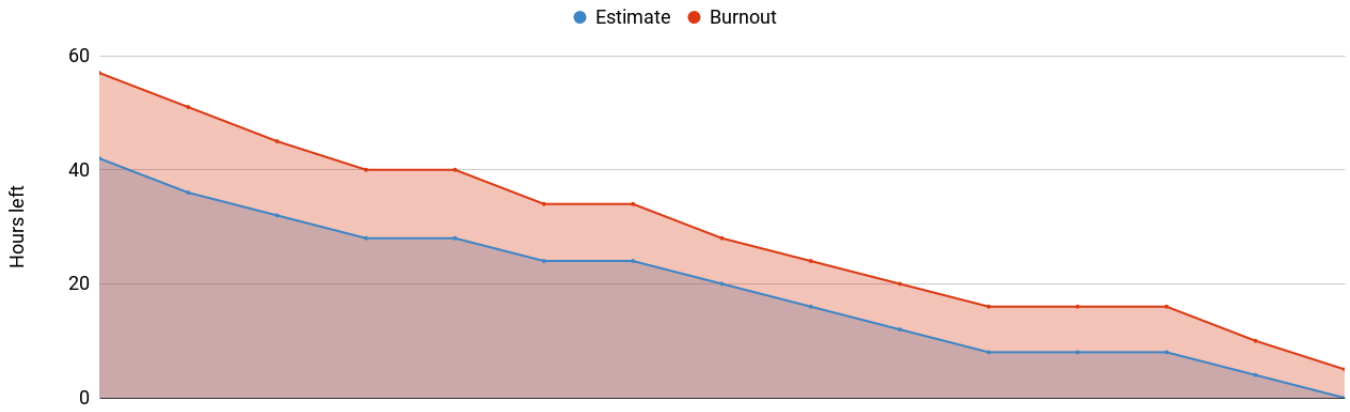

Figura 13: Burnout sprint 6

3.3.7. Sprint 7

Sprint	Fecha inicio	Fecha final
7	9/07/2018	22/07/2018

Historia de Usuario		
Número: 11		Usuario: Product owner
Prioridad en negocio: Alta		Riesgo en desarrollo: Medio
Nombre	Accionar un relé para controlar el riego	
Descripción	Quiero poder abrir y cerrar el riego desde de la web	
Validación	A modo de prototipo se monta un circuito, para que el product owner, al pulsar un elemento de la web accione un relé que maneja el encendido y apagado de un led	

Tabla 21: Historia de usuario número 11

Historia de Usuario		
Número: 12		Usuario: Product owner
Prioridad en negocio: Media		Riesgo en desarrollo: Alto
Nombre	Saber si el riego está accionado	
Descripción	Saber en todo momento si el circuito que controla el relé está cerrado o abierto y poder verlo en la web	
Validación	El product owner acciona el relé y enciende el led, a continuación se reinicia el servidor web, se accede a la vista de riego se pulsa el botón de estado y se comprueba que el estado que muestra es el correcto	

Tabla 22: Historia de usuario número 12

Historia de Usuario		
Número: 13		Usuario: Product owner
Prioridad en negocio: Media		Riesgo en desarrollo: Bajo
Nombre	Programar riego mediante reloj en la web	
Descripción	Insertar en la web unos relojes con los cuales se pueda programar el riego poniendo la hora y estableciendo la duración del riego	
Validación	El product owner ve como el la web se muestran uno relojes en los cuales puede poner la hora de riego y la duración de la misma	

Tabla 23: Historia de usuario número 13

	Time (esti)	Time (spent)	Time (left)	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Historia Usuario 11																	
Investigar funcionamiento de relé conectado a Arduino	8	8	4		2	2											
Montar circuito de Arduino, relé y led en protoboard	2	3	0		3												
Crear rutina en el micrcontrolador que maneje los mensajes que le llegan de accionamiento del relé	4	5	0			5											
Accionar el relé desde terminal en Raspberry Pi	2	2	0				2										
Programar servidor en Raspberry Pi que abre un socket y que cuando le llegue un mensaje para el relé se envíe se lo envíe	2	3	0				3										
Programar un cliente para el servidor socket en el servidor web	2	3	0						3								
Crear una vista riego en la aplicación web	2	2	0						2								
Insertar un elemento en la vista con el cual poder accionar un encendido o apagado	2	4	0							4							
Dar funcionalidad al botón de la vista para que se conecte con el cliente y envíe la señal de accionamiento del relé	3	3	0							3							
Historia Usuario 12																	
Modificar la rutina del microcontrolador para que cada segundo envíe una señal con su estado	2	5	0									5					
Implementar un servidor socket en el servidor web que reciba el estado del relé	2	2	1										1				
Implementar un cliente del servidor en Raspberry Pi que envíe el estado	1	1	0										1				
Modificar la vista del riego para que cargue el estado del del relé sin cargar la página completa	4	7	0										3	4			
Historia Usuario 13																	
Buscar elementos implementados de tipo clockPicker	4	12	0												6	4	2
Añadir dos elementos clockPicker en la vista de riego	2	5	0													2	3
Recoger los datos del los relojes	2	2	2														
	44	67	7	0	5	7	5	0	5	7	0	5	5	4	6	6	5
Daily burnout	0	23		0	4	4	4	0	4	4	4	0	4	4	4	4	4
Total time left (from estimate)			44	44	40	36	32	32	28	24	20	20	16	12	8	4	0
Total time left (from spent)			67	67	62	55	50	50	45	38	38	33	28	24	18	12	7

Tabla 24: Burnout sprint 7

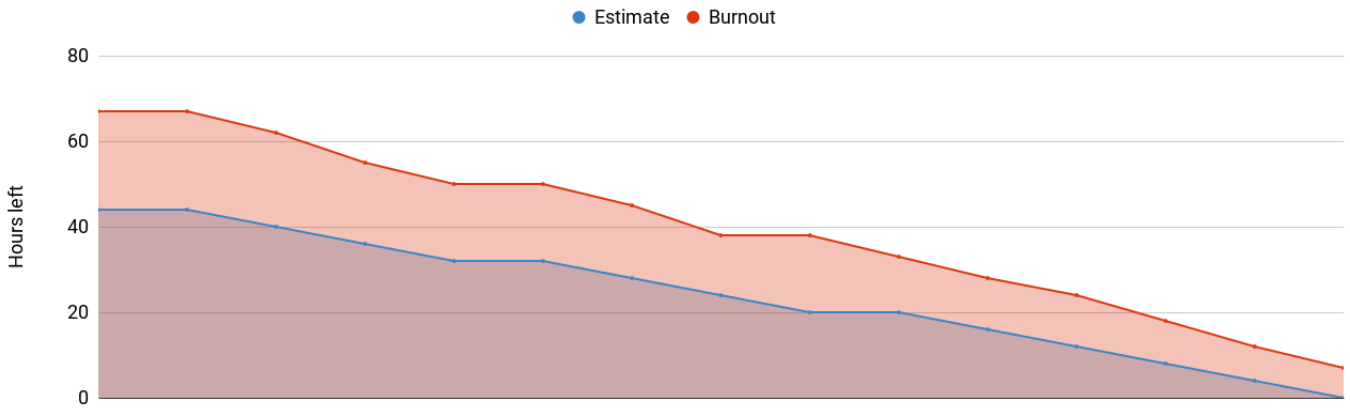

Figura 14: Burnout sprint 7

3.3.8. Sprint 8

Sprint	Fecha inicio	Fecha final
8	23/07/2018	5/08/2018

Historia de Usuario	
Número: 14	Usuario: Product owner
Prioridad en negocio: Alta	Riesgo en desarrollo: Alto
Nombre	¿Se puede adaptar el sistema a IoT?
Descripción	El product owner nos pregunta como sería la transición de la topología que hemos diseñado si creáramos un modelo orientado a IoT y con las comunicaciones a través de Wi-Fi
Validación	Se muestra una topología orientada a IoT. Se muestran los dispositivos que se pueden usar con el fin de tener las comunicaciones vía Wi-Fi

Tabla 25: Historia de usuario número 14

Historia de Usuario	
Número: 15	Usuario: Desarrollador
Prioridad en negocio: Alta	Riesgo en desarrollo: Alto
Nombre	Uso de dispositivos ESP8266 y MQTT
Descripción	En una primera parte de adaptación del sistema a IoT, se cambian los Arduinos por ESP8266 y se implementa el protocolo MQTT, se envían los datos directamente al servidor de base de datos
Validación	Se comprueba cómo se han cambiado los dispositivos anteriores por ESP8266 y que los datos, son mandados directamente al broker MQTT

Tabla 26: Historia de usuario número 15

	Time (esti)	Time (spent)	Time (left)	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Historia Usuario 13																	
Recoger los datos del los relojes	2	6	0	4	2												
Historia Usuario 14																	
Investigar sobre IoT	12	15	0	5	5		5										
Desarrollar topología nueva con los elementos nuevos y la estructura de la aplicación	10	10	4					6									
Historia Usuario 15																	
Adquirir dispositivos nuevos	1	1	0						1								
Programar ESP8266	4	30	0						4	5	5	6		5	5		
Conectar con cables los ESP8266 con los sensores	2	2	1														1
Conectar con cables ESP8266 con sensor y relé	1	1	0														1
Instalar y configurar broker MQTT en el servidor de base de datos	2	2	0														2
Programa en el servidor de base de datos que recoja los datos de los sensores y los inserte en la base de datos y que gestione los mensajes al relé	3	3	3														
	37	70	8	9	7	0	5	6	5	5	5	6	0	5	5	0	4
Daily burnout	-7	26		4	4	0	4	4	0	4	4	4	4	4	4	0	4
Total time left (from estimate)			37	28	24	24	20	16	16	12	8	4	0	-4	-8	-8	-12
Total time left (from spent)			70	61	54	54	49	43	38	33	28	22	22	17	12	12	8

Tabla 27: Burnout sprint 8

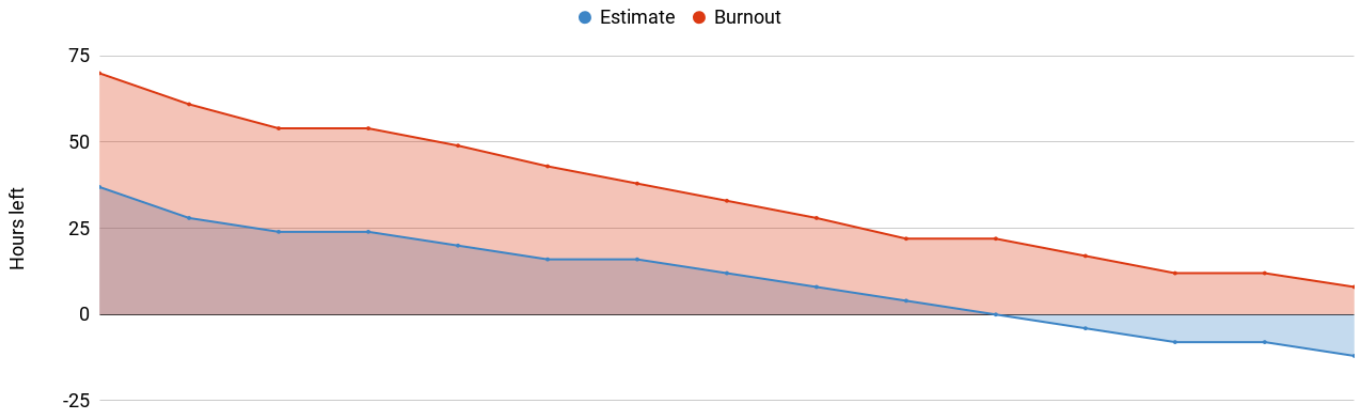

Figura 15: Burnout sprint 8

3.3.9. Sprint 9

Sprint	Fecha inicio	Fecha final
9	6/08/2018	19/08/2018

Historia de Usuario		
Número: 16		Usuario: Desarrollador
Prioridad en negocio: Alta		Riesgo en desarrollo: Bajo
Nombre	Orientación a Objetos de la Aplicación	
Descripción	Rediseñar la aplicación para que la Programación sea Orientada a Objetos	
Validación	A través del código se puede comprobar la Orientación a Objetos	

Tabla 28: Historia de usuario número 16

Historia de Usuario		
Número: 17		Usuario: Product owner
Prioridad en negocio: Alta		Riesgo en desarrollo: Alto
Nombre	Desarrollo de APIRest	
Descripción	Las necesidades cambian y ahora se pretende que la aplicación de servicio a otras aplicaciones mediante peticiones http	
Validación	Se hacen varias peticiones http con los servicios que se tenían anteriormente y se comprueba que llegan los datos de manera correcta y que se puede accionar el relé	

Tabla 29: Historia de usuario número 17

Historia de Usuario		
Número: 18		Usuario: Product owner
Prioridad en negocio: Alta		Riesgo en desarrollo: Medio
Nombre	Control del consumo de agua	
Descripción	Añadir un dispositivo nuevo con un sensor que mida el caudal del agua una vez que se haya activado el relé.	
Validación	El product owner comprueba que una vez activado el relé el caudalímetro empieza a contar los litros de agua que pasan por el sensor (esto se hace mediante una simulación y lo que hace mover la turbina del sensor es la acción de soplar por el mismo) viéndolo a través de una APP de Android	

Tabla 30: Historia de usuario número 18

Historia de Usuario		
Número: 19		Usuario: Product owner
Prioridad en negocio: Alta		Riesgo en desarrollo: Medio
Nombre	Programar riegos	
Descripción	Quiero poder programar los riegos y que si me equivoco y solapo dos riegos el sistema lo reconozca y que el sistema almacene de maneara persistente los riego y cuánto he regado en cada riego	
Validación	El product owner comprueba cómo se programan varios riegos, alguno se solapa y el sistema no lo permite dando un mensaje, y una vez que llega el tiempo del riego se ve cómo se activa el relé cerrando el circuito y activa el led en vez de la electroválvula y se sopla por el sensor de que mide el caudal del agua y se comprueba que la medición del riego se almacena correctamente en la base de datos.	

Tabla 31: Historia de usuario número 19

	Time (esti)	Time (spent)	Time (left)	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Historia Usuario 15																	
Programa en el servidor de base de datos que recoja los datos de los sensores y los inserte en la base de datos y que gestione los mensajes al relé	3	4	0	4													
			0														
Historia Usuario 16			0														
Modelo de análisis de clases necesarias para la aplicación	8	8	0		4	4											
Implementación de las clases	4	5	0				5										
Implementar un paquete Python con las clases necesarias para el funcionamiento de la aplicación	4	4	1						3								
Instalar el paquete Python en el servidor de base de datos	1	1	1														
Crear un programa en el servidor de base de datos que utilice las clases implementadas y que gestione los datos que recibe el broker MQTT	4	5	0							5							
Historia Usuario 17																	
Instalar paquete Python con clases en el servidor web	1	1	0								1						
Desarrollo de APIRest con los servicios que anteriormente se tenían en la web	4	10	0								4	2	4				
Historia Usuario 18																	
Programar ESP8266	2	2	0												2		
Cablear ESP8266 y caudalímetro	1	1	0												1		
Instalar APP Android MQTT para ver cómo manda la señal el ESP8266	1	1	0												1		
Historia Usuario 19																	
Añadir tabla en la base de datos	1	1	0													1	
Modificar programa backend para que mida el caudal del caudalímetro y añada los datos a la base de datos	2	3	0													3	
Añadir servicio en la APIRest	2	3	0														3
	38	49	2	4	4	4	5	0	3	5	5	2	4	0	4	4	3
Daily burnout	-10	1		4	4	4	4	0	4	4	4	4	4	0	4	4	4
Total time left (from estimate)		E	38	34	30	26	22	22	18	14	10	6	2	2	-2	-6	-10
Total time left (from spent)			49	45	41	37	32	32	29	24	19	17	13	13	9	5	2

Tabla 32: Burnout sprint 9

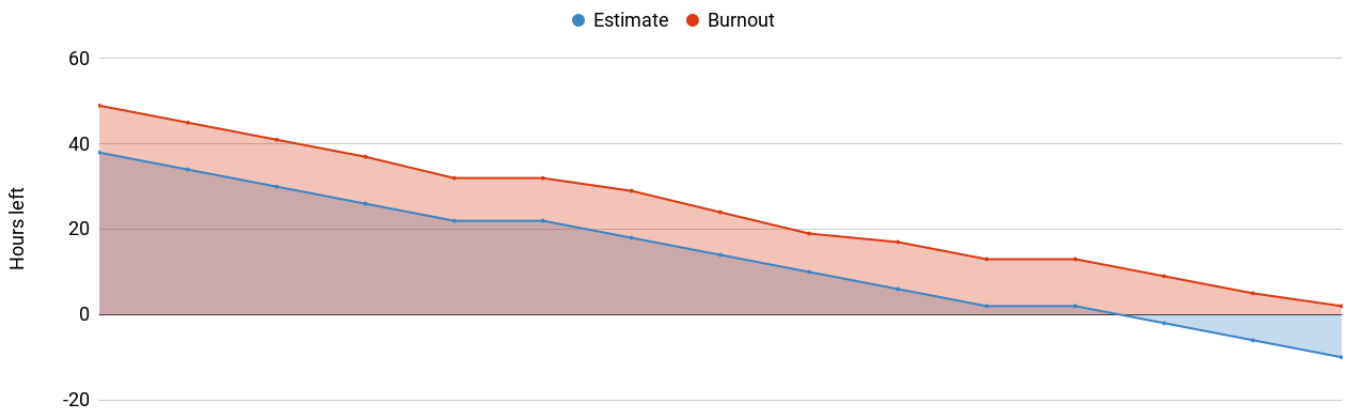

Figura 16: Burnout sprint 9

3.3.10. Sprint 10

Sprint	Fecha inicio	Fecha final
10	20/08/2018	2/09/2018

Historia de Usuario	
Número: 20	Usuario: Desarrollador
Prioridad en negocio: Alta	Riesgo en desarrollo: Alto
Nombre	Memoria del proyecto
Descripción	Memoria completa del proyecto con todas las partes establecidas
Validación	El proyecto es entregado en la secretaría de dirección el día 3 de septiembre.

Tabla 33: Historia de usuario número 20

4. Análisis

En la ingeniería del software, la fase de análisis, es la que se corresponde con la obtención de toda la información para la implementación del sistema. Sin embargo, en este proyecto, se ha utilizado un método de desarrollo ágil basado en Scrum y por lo tanto desde del inicio no se dispone de todos los requisitos que va a tener la aplicación. Para la realización de los diagramas se hace referencia a las Historias de Usuario para un mejor entendimiento.

4.1. Topología del Sistema

La topología del sistema se modela a través de los nodos que lo conforman, siendo estos, elementos físicos que existen en tiempo de ejecución y representan un recurso computacional.

Para la representación de la topología del prototipo 1 se va a tener en cuenta que el proyecto se encuentra en el Sprint 7 3.3.7. En este momento del proyecto ya están todos los elementos que conforman el sistema y por ello se puede hacer un diagrama de él.

En cuanto a la representación de la topología del segundo prototipo, el proyecto se encuentra en el Sprint 8 3.3.8.

4.1.1. Prototipo 1

En la figura 17 se aprecia cómo se ha usado el diferente hardware necesario para conseguir un sistema funcional de *IoT*. Para ello se ha utilizado una notación no formal. Los sensores DHT-22 van unidos mediante cable a Arduino. El sensor está continuamente enviando los datos al microcontrolador, a través del pin de datos, y este solamente los envía al dispositivo que hace de concentrador, *recolector de datos*, cada cierto tiempo. Los datos se envían a Rapsberry Pi a través de conexión *Bluetooth*. Para ello se ha añadido un HC-05 al sistema que está conectado con cada *Arduino* vía cableado.

Una vez los datos llegan a *Recolector de datos*, Rapsberry Pi, este los pone en la cola de mensajes de RabbitMQ y de esta manera llegan hasta el *Servidor RabbitMQ* que se está ejecutando en la misma máquina donde se encuentra el servidor de base de datos MySQL.

El *Consumidor*, cuando le llegan los datos, los va insertando el la base de datos. Ahora que los datos se encuentran de manera persistente en la base de datos, a estos puede acceder la aplicación web. Además la aplicación web puede enviar la señal de encendido o apagado al *Actuador* mediante un *socket* que conecta con la Raspbery Pi, y esta le envía dicha señal al Arduino pasando al relé accionando el mecanismo, en este caso el riego.

4.1.2. Prototipo 2

La topología del prototipo 2 (figura 18) ha variado bastante en cuanto a la parte de los sensores y actuadores, *IoT*. Ahora nos encontramos que los dispositivos utilizados ya no son los *Arduino* sino los microcontroladores ESP8266 que cuentan con

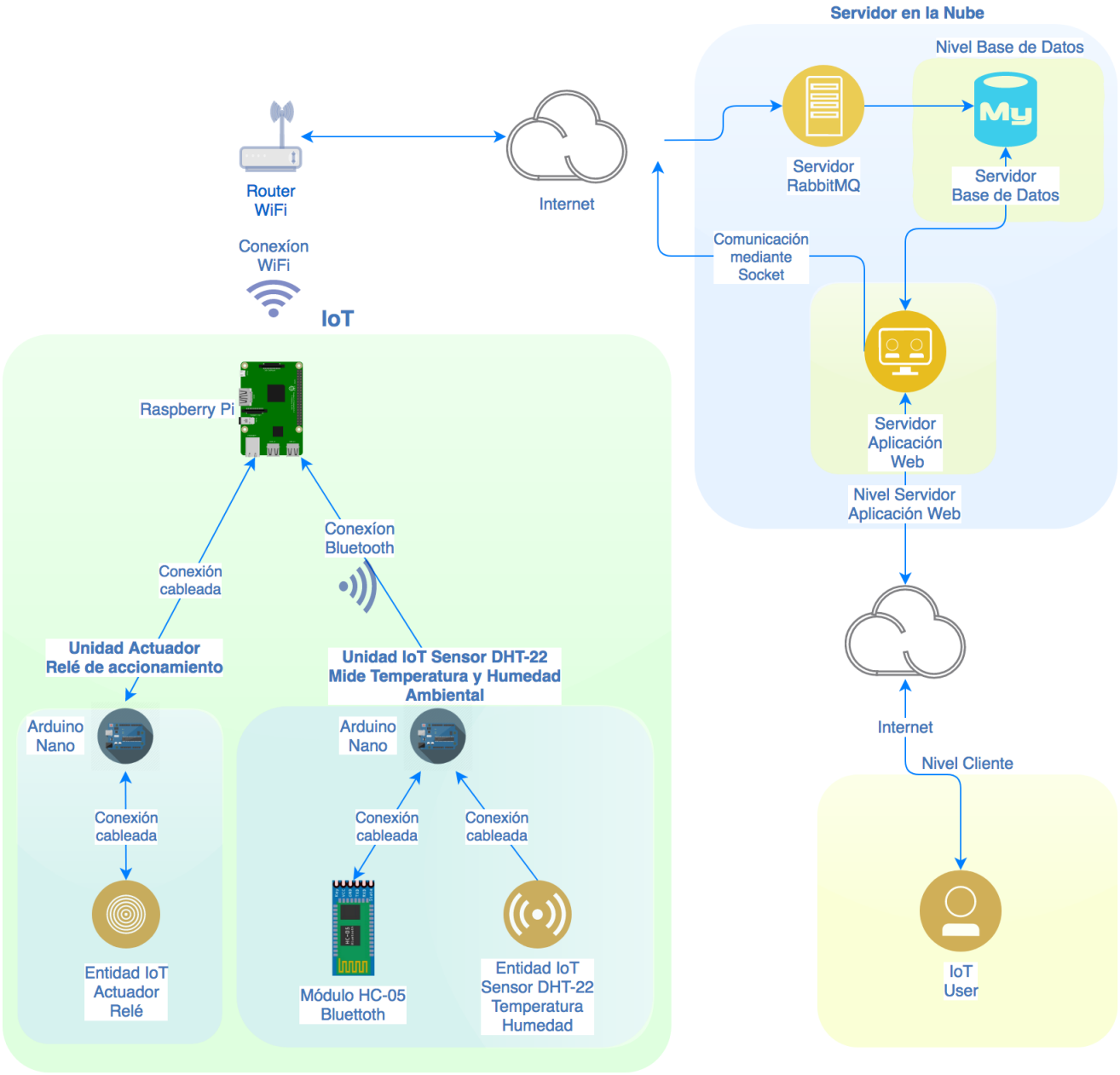

Figura 17: Arquitectura prototipo 1

módulo Wi-Fi. Además se ha utilizado el protocolo de comunicación para *IoT* MQTT que proporciona una capa de abstracción que permite la comunicación entre los microcontroladores *ESP8266* y el *Servidor MQTT*. A su vez el *Servidor MQTT* tiene varios manejadores para que una vez que le llegan los datos los inserte en la base de datos. A diferencia del prototipo 1, el actual proporciona una interfaz en forma de *API Rest* al cual pueden hacer peticiones http cualquier aplicación que tenga acceso a la misma.

Ventajas del prototipo 2:

- Mayor alcance de las unidades *IoT* ya que la conexión es vía Wi-Fi.
- No es necesario más que el microcontrolador ESP8266 para la comunicación ya que tiene integrado el módulo Wi-Fi evitando tener que añadir el módulo Bluetooth HC-05. Esto evita la gestión de la comprobación de la conexión Bluetooth quitando complejidad al conjunto además de eliminar una fuente de *bugs*.
- Tampoco es necesario ya la Raspberry Pi. Al utilizar el protocolo de comunicación *MQTT* el ESP8266 se conectada directamente con el *servidor MQTT*. De nuevo se quita complejidad al sistema con la consiguiente eliminación de puntos de fallo.
- Ahora es mucho más sencillo añadir sensores o actuadores al sistema. La modularidad esstá notablemente mejorada. Para añadir un nuevo elemento solamente hay que conectarlo a la red Wi-Fi y programarlo con los mismos parámetros que los anteriores, únicamente hay que cambiar los *topic* y añadir manejadores para los nuevos *topic*.

Para el prototipo 2 se han conseguido una mejoras muy relevantes en comparación al primer prototipo. En la figura 18

4.2. Modelo de Dominio

En la ingeniería de software el modelo de dominio es un modelo conceptual para la resolución de problemas. En él se reflejan las distintas entidades y relaciones que lo conforman.

Como ya se ha señalado con anterioridad, en Scrum no se dispone de todos los requisitos, ni siquiera de todos los elementos que pueden conformar el modelo de dominio desde el inicio. Por ello se va a hacer referencia al Sprint 3.3.7 que es cuando se conocen todos los elementos que intervienen en el modelo de dominio que se presenta a continuación.

4.2.1. Prototipo2

Modelo de Dominio

En la figura 19 se muestra el diagrama de clases que representa el modelo de dominio de las clases afectadas en este proyecto.

Estas clases son las siguientes:

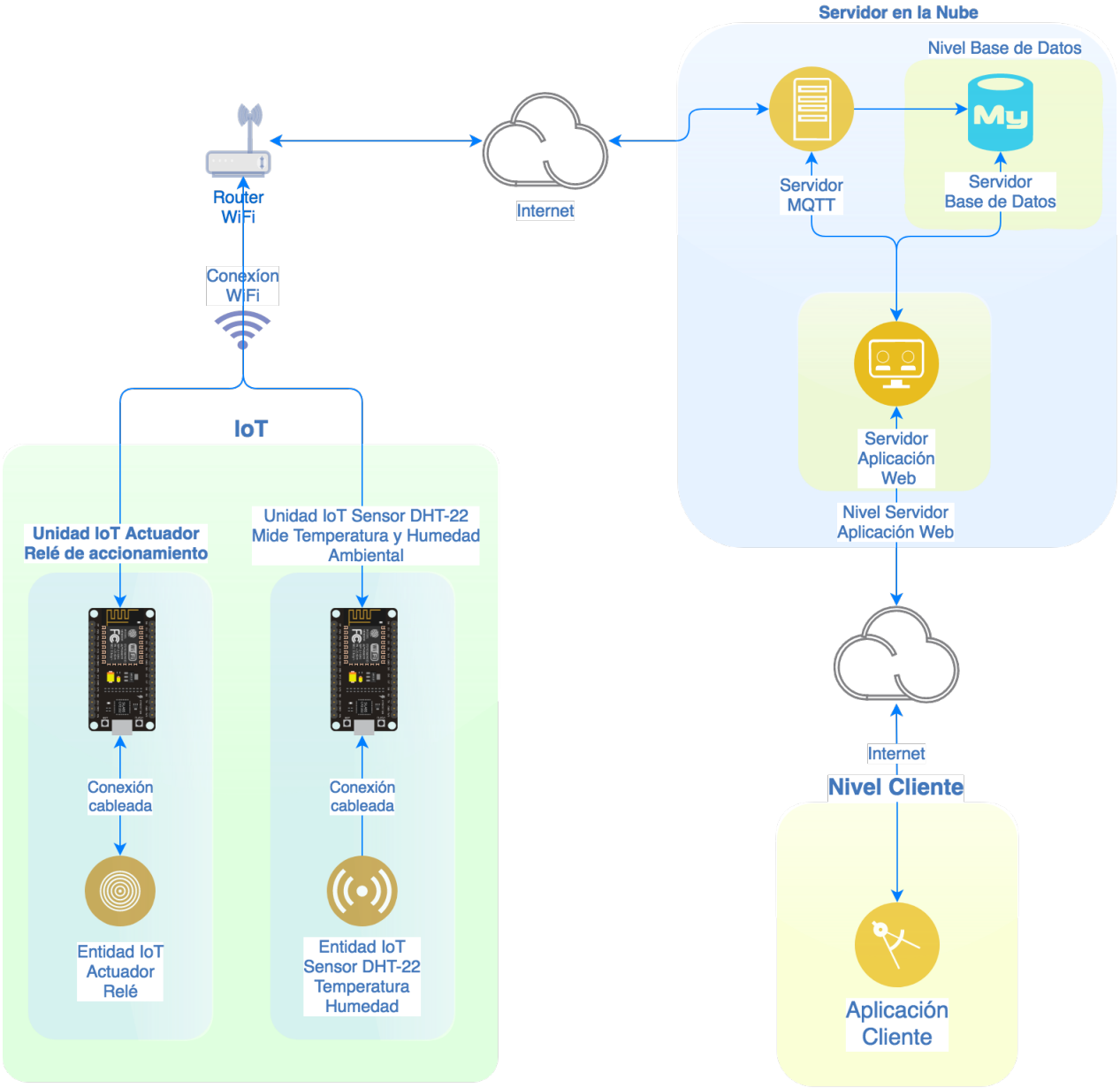

Figura 18: Arquitectura prototipo 2

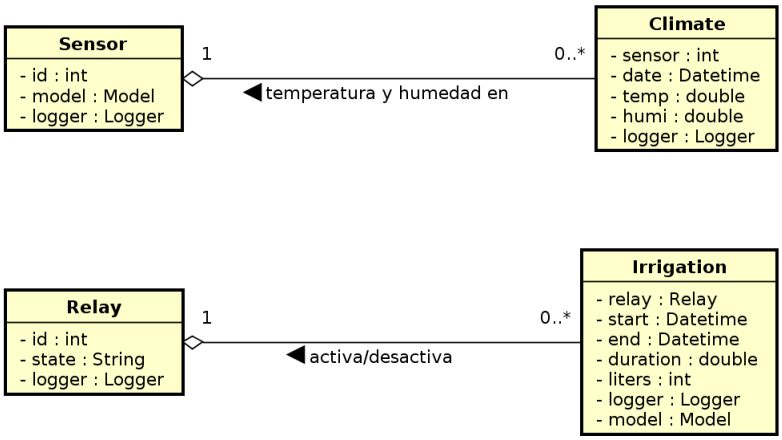

Figura 19: Modelo de Dominio

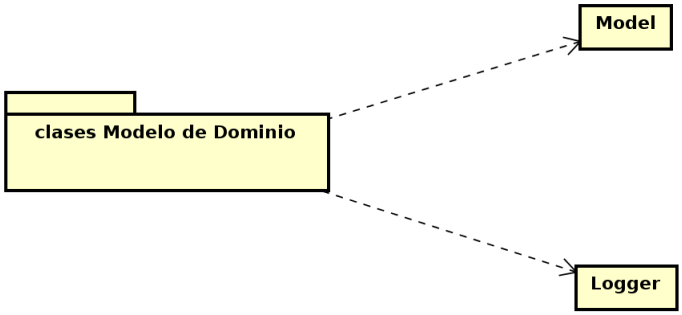

Figura 20: Interacción Modelo de Dominio con la clase Model

- Sensor: Es un modelo que representa un sensor del mundo físico que mide temperatura y humedad. Sus operaciones son del tipo leer los datos que ha proporcionado al sistema.
- Climate: Es la representación de los datos que recoge el sensor y su misión es guardar esos datos en la base de datos.
- Relay: Es un clase que representa un relé del mundo físico. Su función es activarse o desactivarse.
- Irrigation: Representa un riego en el sistema. Principalmente puede comunicarse con el relé al que está asociado para que se active/desactive e insertar los datos del riego en la base de datos.

Para la interacción con la base de datos se ha pensado en modelar una clase que haga de interfaz de la base de datos con la cual interaccionan las otras clases (figura 20, también se dispone de la clase Logger que es un manejador de los logs de sistema.

Diagrama de Paquetes

Es un mecanismo de propósito general para organizar el modelo utilizado para que sea un mecanismo de agrupación lógica a través de una agrupación física de los elementos en un componente [7]. En la figura 21 se puede ver como está estructurada la aplicación y *gh-backend* y *gh-APIRest* clientes de los servicios que proporciona el paquete *gh-tools*.

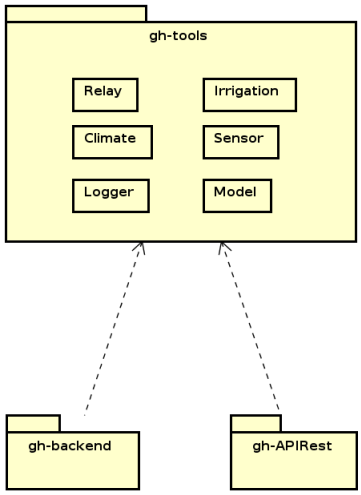

Figura 21: Modelo de Dominio

5. Arquitectura y Diseño

En la fase de diseño se especifican en detalle todos los componentes de la arquitectura del sistema, así como la definición de todas las clases que intervienen en la implementación y sus interacciones.

En esta sección solamente se van a dar unas pinceladas de la documentación que puede llevar esta parte de la memoria si se hubiera hecho mediante el Proceso Unificado, por ejemplo. Una de las ventajas que tienen los métodos ágiles es que se presenta menos documentación que otros procesos de desarrollo como el anteriormente reseñado. Por ello, y siguiendo una filosofía ágil basada en Scrum, se van a detallar los diagramas que se han considerado más importantes.

5.1. Arquitectura

El proceso de diseño arquitectónico identifica los principales componentes de un sistema y sus comunicaciones [5].

El subsistema que conforman los *Objetos IoT* junto al *Broker MQTT* utilizan el patrón arquitectónico de Publicación-Suscripción. El funcionamiento de este modelo está definido en la sección 2.4.2.

Sin embargo, el subsistema con el cual interactúan los clientes finales de la aplicación, que pueden ser cualquier tipo de aplicación que utilice peticiones HTTP en cualquiera de sus variantes, implementa la arquitectura clásica de *Cliente-Servidor* donde el servidor es un programa que proporciona servicios y el cliente es un programa que representa entidades que solicitan un servicio. Este modelo se puede aplicar a programas que residen en la misma máquina o en máquinas diferentes. En nuestro caso, el cliente y el servidor se encuentran en máquinas diferentes y se comunican a través de la red y sobre la capa de aplicación del modelo TCP/IP a través de peticiones HTTP. En la figura 18 se puede ver la topología del sistema.

En el diagrama de despliegue (figura 22) se puede ver como se comunican las diferentes partes y como es su división. La primera, como se ha mencionado anteriormente, la forman los *Objetos IoT* y el *Broker MQTT*. Entre ellos, las comunicaciones se basan en el patrón *Publicacíon/Suscripción* sobre el protocolo de comunicaciones *Message Queuing Telemetry Transport*.

El siguiente patrón arquitectónico utilizado es el de *Cliente/Servidor* y se utiliza entre el Servidor Web y las Aplicaciones que le piden servicios a través de peticiones HTTP.

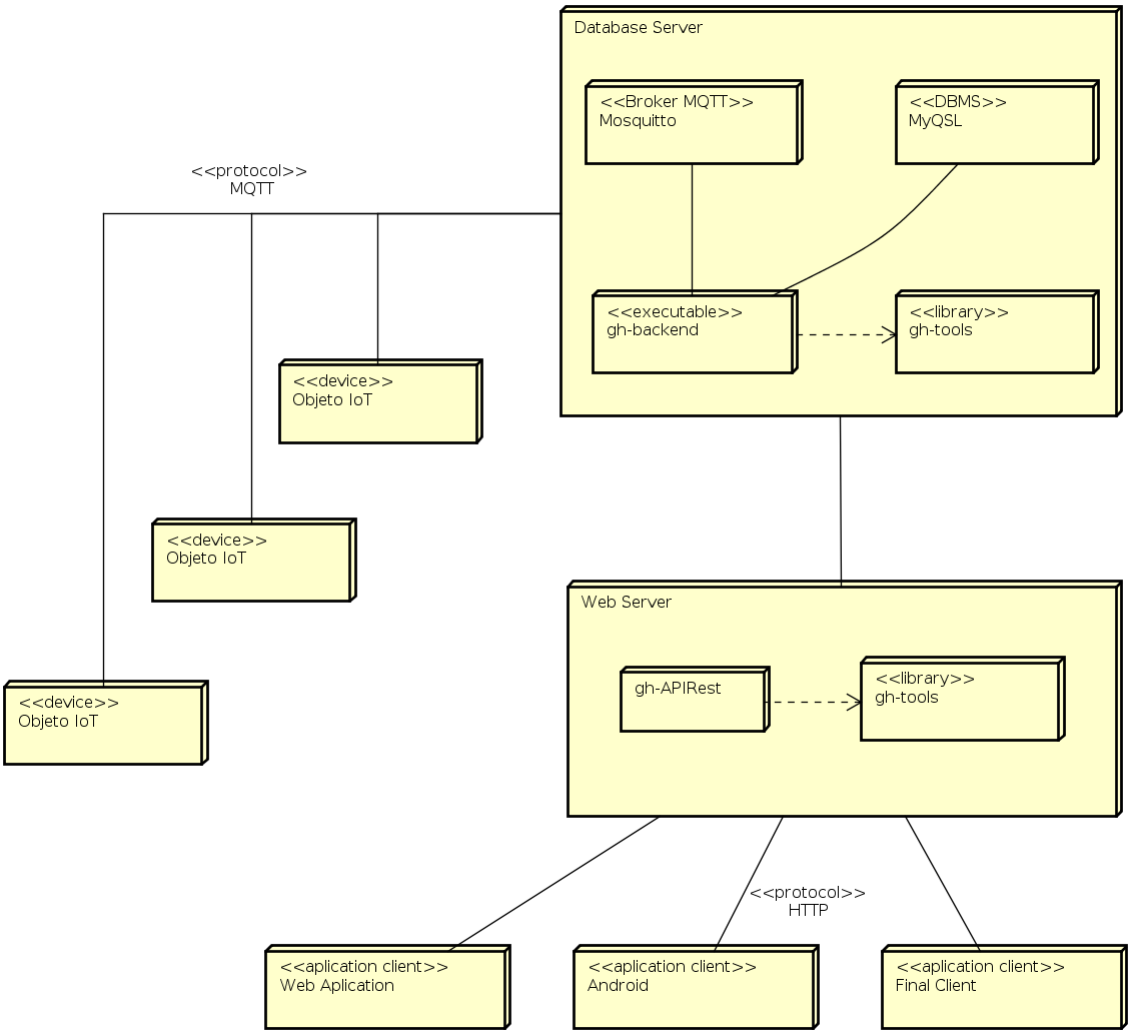

Figura 22: Diagrama de despliegue

5.2. Diseño de la Base de Datos

El gestor de bases de datos que se ha usado en el sistema es MySQL para la persistencia de los datos que se almacenan de los diferentes sensores y actuadores que conforman el sistema. En la figura 23 se puede ver cómo se guardan los datos y cómo se conforman las relaciones entre las tablas.

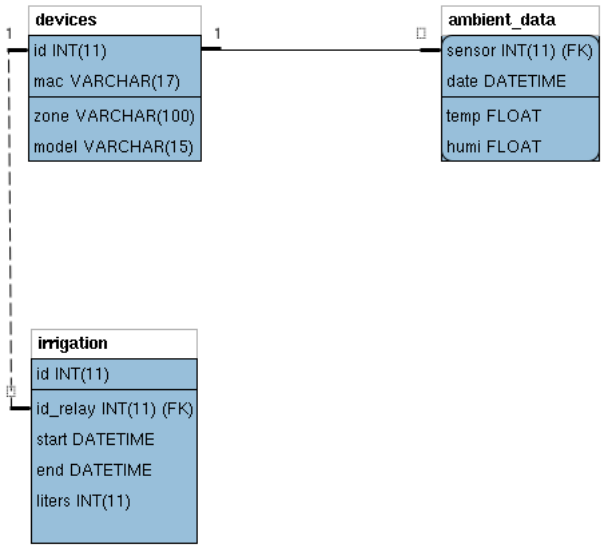

Figura 23: Modelo entidad relación

Descripción de las tablas:

- devices: representan los dispositivos que se dan de alta en el sistema,
  - se les asigna un id,
  - mac del dispositivo,
  - zona donde está situado,
  - el tipo de dispositivo que es
- ambient_data: recoge los datos ambientales que le suministran los sensores que estén dados de alta en devices,
- irrigation: almacena los datos de los riegos que han hecho los dispositivos que pueden accionar el riego y almacena los datos relevantes.

5.3. Diseño de las Clases

En la parte de diseño, además de aparecer las clases del modelo de dominio, deben aparecer las que la aplicación necesita para su funcionamiento interno. Las clases que aparecen en la figura 24 son las que se han diseñado específicamente para esta aplicación. Hay otras clases de Python o librerías que no aparecen en este diagrama de clases, aunque sí intervienen en la aplicación.

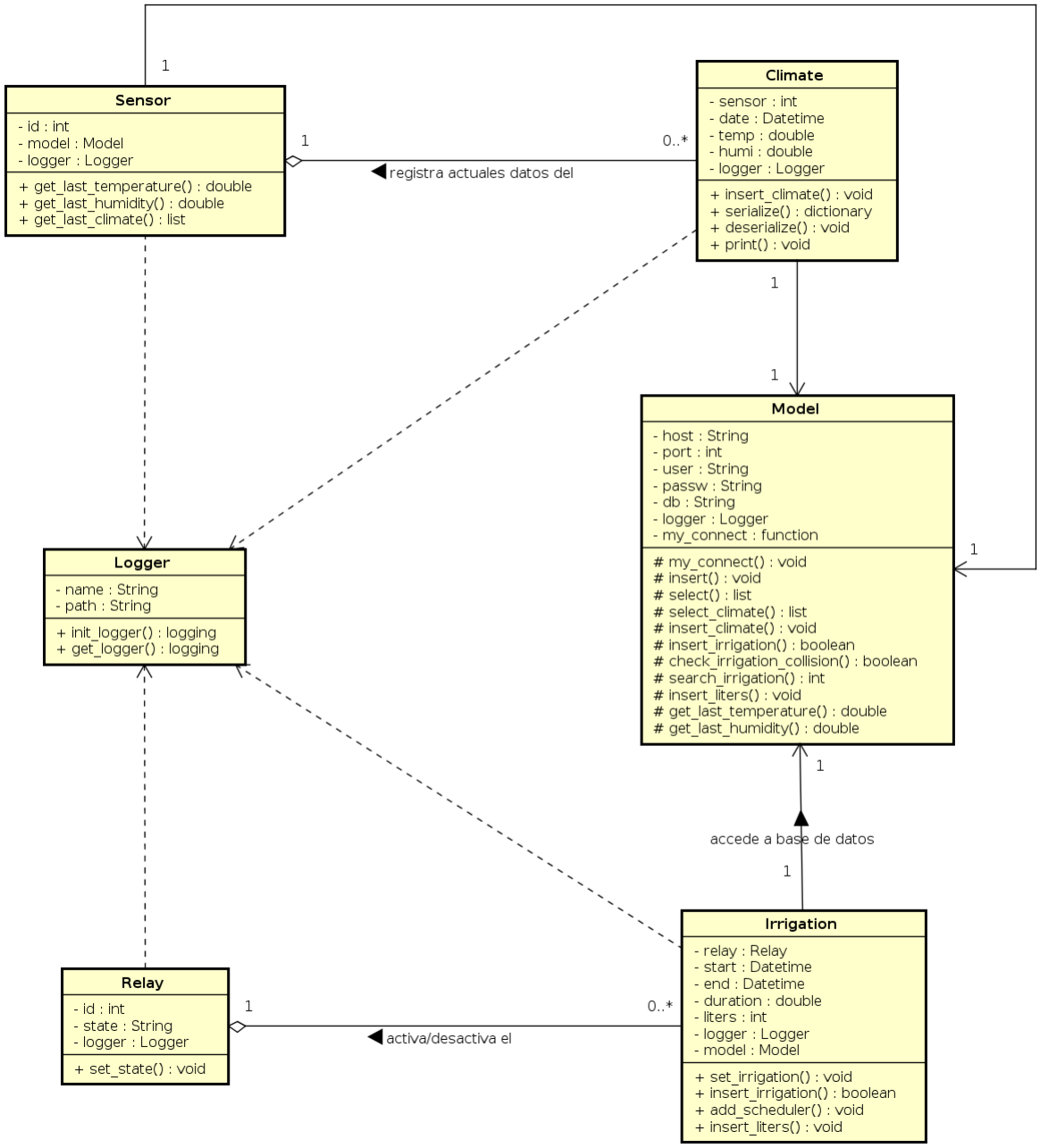

Figura 24: Modelo entidad relación

6. Implementación

En esta sección se van a detallar los problemas que han surgido en el transcurso del desarrollo de la aplicación y cómo se han resuelto. Así, se ha subdivido en dos partes, una *hardware* y otra de *software*.

6.1. Hardware

6.1.1. ESP8266 ESP-01

Es la primera versión y la más económica de la gama ESP8266. Para programarlos las conexiones se deben hacer tal como se muestra en la tabla 34 [20].

Número Pin	Nombre Pin	Cargar programas	Normalmente Utilizado para
1	Ground	Ground	Conectado a la Tierra del circuito
2	TX	RX	Conectado al pin RX del microcontrolador para cargar el programa
3	GPIO-2	Sin conectar	Pin de uso general Entrada/Salida
4	CH_EN/CH_PD	3.3 v	Acitvacion del Chip, activo en alta
5	GPIO-0	Sin conectar	Pin de uso general Entrada/Salida
6	Reset	3.3 v	Resetear módulo
7	RX	TX	Conectado al pin TX del microcontrolador para leer
8	Vcc	3.3 v	Alimentación

Tabla 34: Pines de ESP01

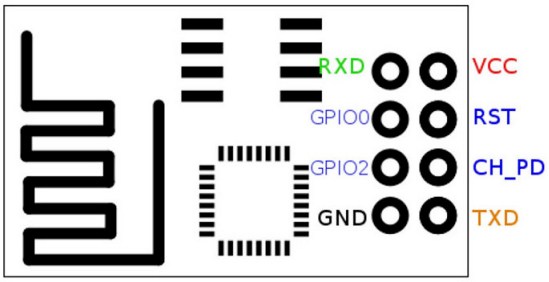

Figura 25: Pinout de ESP01

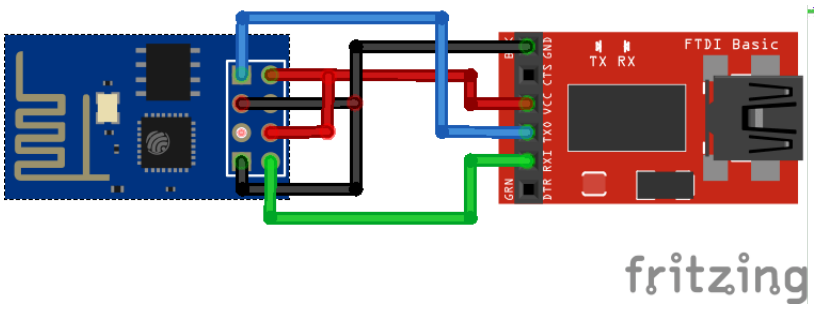

Figura 26: Conexiones para cargar programas en el dispositivo

Después de muchos intentos y pruebas no se consigue cargar en los dispositivos adquiridos en la tienda online AliExpress los programas. Se han hecho las conexiones tal como muestran diferentes webs y tutoriales sin ningún éxito con el siguiente mensaje de error:

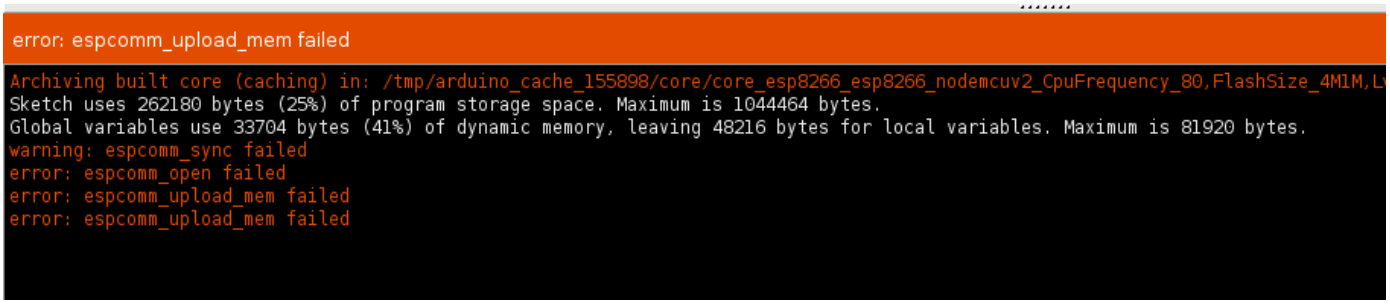

Figura 27: Error que muestra al tratar de cargar un programa

La solución que se ofrece [22], como se puede ver en esta *Issue* en la página de ESP8266 en GitHub y en un sinfín de diferentes webs y post, a este error es conectar *GPIO-0* a la toma de tierra. Después de probar a hacer dicha conexión se produce el mismo resultado de error al cargar un programa al dispositivo. Estos intentos se han hecho en un Arduino UNO. También se ha intentado en un entorno profesional con FTDI con el mismo resultado.

Los dispositivos sí funcionan ya que el led de encendido se mantiene encendido cuando el dispositivo se encuentra encendido y otro led parpadea cuando se está subiendo el programa. Además se han probado mediante comandos AT y responden como es de esperar.

Tras todos estos intentos no se llega una conclusión de por qué no se pueden cargar programas al ESP01. La solución pasó por utilizar los Nodemcu V3 ESP8266 12e LoLin, facilitados por Argotec. En estos dispositivos se cargan los programas sin ningún problema.

6.1.2. DHT-11

Este sensor es la versión básica del DHT-22. Las características técnicas son algo inferiores tales como la precisión de las medidas o el refresco de las mediciones. El problema que surge con esto modelo, al menos en este proyecto, es que el sensor deja de funcionar por completo. Esto puede ser debido a que estos no llevaban un circuito integrado añadido, en el cual han puesto ya una resistencia, como sí viene en los DHT-22 que se han utilizado para el proyecto.

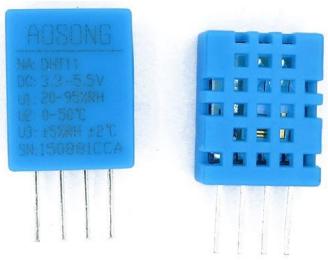

Figura 28: Sensor DHT-11

La solución pasó por adquirir unos sensores de mejor calidad, DHT-22, que vienen con un circuito integrado en cual hay una resistencia para limitar la corriente y evitar que el sensor tenga un pico de tensión dejando de funcionar.

6.1.3. Rapspberry Pi 2 Model B

Ciertamente la dificultad de utilizar Raspberry PI 2 no es más que, entre sus características no están las conexiones inalámbricas que se utilizan en el proyecto como son el Bluetooth o el Wi-Fi y por lo tanto se tuvo que adquirir una Raspberry Pi 3.

También se podrían haber utilizado adaptadores USB de Bluetooth y Wi-Fi con los cuales se habría conseguido la misma solución inalámbrica pero se optó por la anterior ya que además, el model 3, ofrece unas mejoras en hardware que son notables, principalmente el procesador contando con un *Quad core* de 1.2GHz y una arquitectura de 64 bits.

6.2. Software

En este apartado se van a precisar los dificultades que se han encontrado tanto en la configuración de sistemas como en la implementación del código contenido en la aplicación.

Muchas veces estas dificultades se han encontrado por no tener una correcta configuración de los sistemas utilizados o querer utilizarlos de una manera para la cual no están pensados.

Esto se debe a que dichos sistemas no se conocen lo suficiente con anterioridad, y por lo tanto, se es dado a tener planteamientos incorrectos o a no configurar correctamente los mismos.

6.2.1. MySQL

En la primera versión del prototipo 1 las inserciones en la base de datos se hacían directamente desde la Rapsberry Pi. Esto está considerado como una mala práctica por temas de seguridad y por lo tanto se cambió en las siguientes versiones. La configuración por defecto del gestor de base de datos MySQL no permite conexiones remotas. Para habilitarlas es necesario crear un usuario y darle los permisos que se quieran sobre la base de datos y sobre las tablas de manera explícita y además estableces desde qué IP se va a conectar. Como las IPs desde donde se conecta con el servidor de base de datos son dinámicas, entonces se ha de permitir conexiones desde cualquier IP. Como ya se ha mencionado anteriormente se han de dar los permisos al usuario. Tal como se puede observar en la figura 29 se listan los usuarios existentes en el *Gestor*. Cabe destacar que puede haber usuarios con el mismo nombre pero con distinta dirección siendo usuarios diferentes. También se puede ver como ha sido necesario poner que kave puede conectarse desde culquier dirección mediante el símbolo *%*. En las siguiente figura 30 se puede ver cómo el usuario kave tiene permiso para todas la bases de datos y todas sus tablas. Además con *ALL PRIVILEGES* puede hacer cualquier cosa sobre cualquier base de datos, algo bastante peligroso, ya sea por un ataque o por un error. En una primera etapa en la cual se *trastea* mucho en la base de datos se puede permitir esto pero una vez que la base de datos está creada con sus tablas y no se van a modificar lo correcto es revocar todos los permisos y dejar únicamente los necesarios. Actualmente como las inserciones se hacen desde la misma máquina lo correcto sería que el usuario *kave* solo pudiera acceder desde *localhost*. No se modifica esto ya que actualmente se siguen modificando algunas cosas desde otras máquinas que no son la local pero se es plenamente consciente de ello.

```
mysql> SELECT DISTINCT CONCAT('SHOW GRANTS FOR ''', user, '''@''', host, ''';') FROM mysql.user;
+-----------------------------------------------------------+
| CONCAT('SHOW GRANTS FOR ''', user, '''@''', host, ''';') |
+-----------------------------------------------------------+
| SHOW GRANTS FOR 'kave'@'%';                               |
| SHOW GRANTS FOR 'root'@'127.0.0.1';                       |
| SHOW GRANTS FOR 'root'@'::1';                             |
| SHOW GRANTS FOR 'root'@'kavemachine';                     |
| SHOW GRANTS FOR 'debian-sys-maint'@'localhost';           |
| SHOW GRANTS FOR 'jose'@'localhost';                       |
| SHOW GRANTS FOR 'root'@'localhost';                       |
+-----------------------------------------------------------+
7 rows in set (0.00 sec)
```

Figura 29: Usuarios en MySQL

```
mysql> show grants for kave@'%';
+-----------------------------------------------------------------------------------------------------------------+
| Grants for kave@%                                                                                               |
+-----------------------------------------------------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'kave'@'%' IDENTIFIED BY PASSWORD '*3F50515DDEE62F18A2B1CE3BE819CFB2F3C869F1' |
| GRANT ALL PRIVILEGES ON `temperatura`.* TO 'kave'@'%'                                                           |
+-----------------------------------------------------------------------------------------------------------------+
2 rows in set (0.02 sec)
```

Figura 30: Permisos del usuario kave

6.2.2. Organización del Proyecto

En la *versión 0.1.0* del proyecto, prototipo 1, todo el proyecto está implementado en un único proyecto. Como se puede ver el la figura 17 están claramente divididas las partes pero el proyecto se implementó en uno solo como se puede ver en la

figura 31. Los programas se lanzan desde diferentes niveles en el árbol y utilizan los mismo herramientas y clases.

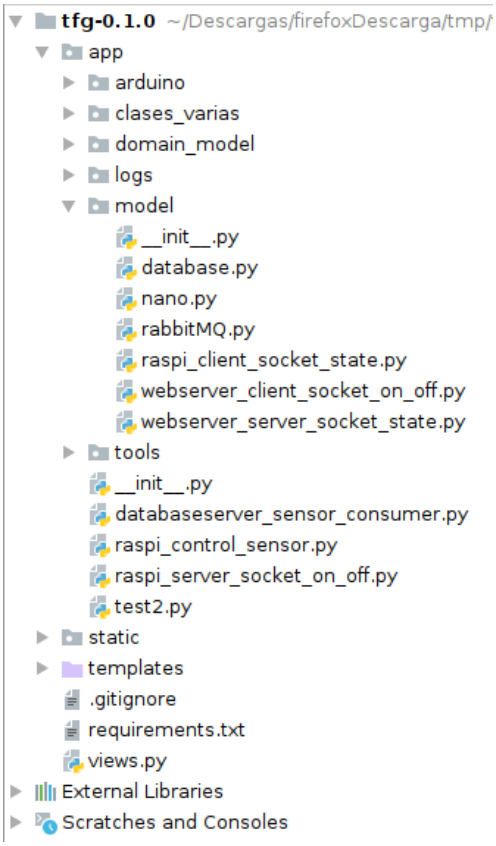

Figura 31: Árbol de ficheros del prototipo 1

Al hacer los *import* hay que decir dónde están y según qué programa los va a usar la dirección no es la misma o según en qué máquina se ejecutan, como se puede ver en el siguiente código 6.2.2. La solución, mala, pasó por añadir un bloque *try except* e introducir los *import* en ella. Funcionaba, pero parece una mala práctica para solucionar un problema que está mal enfocado. Algo similar pasa al crear el *logger*, dependiendo dónde de qué programa se lance y cree el objeto hay que configurarlo de manera diferente.

```
import socket

try:
    from app.tools.logger import create_log
    from app.tools.config import *
except ImportError:
    from tools.logger import create_log
    from tools.config import *

HOST, PORT = raspi_ip, raspi_socket_port_on_off

try:
    logger = create_log(webserver_logger)
except:
    logger = create_log(raspi_logger)
```

```
16
17
18 def relay_on_off(state: str):
19     data = state.encode()
20
21     # Create a socket (SOCK_STREAM means a TCP socket)
22     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
23
24     try:
25         # Connect to server and send data
26         sock.connect((HOST, PORT))
27         sock.sendall(bytes(data))
28     except OSError as err:
29         logger.error(err)
30     finally:
31         sock.close()
```

Listing 1: webserver_client_socket_on_off

En general tener todas las partes del proyecto en un único proyecto no parece una buena idea y por lo tanto en el prototipo 2 se divide la aplicación en varias partes. Como se ha indicado la solución es dividir la aplicación en varias partes. Estás divisiones se pueden ver en la topología de la aplicación en la figura 18. Así queda dividido en:

- gh-backend: Los programas de los microcontroladores ESP8266, y el recolector de datos de los sensores/actuadores, los *Objetos IoT*, e interactuar con el *Gerstor de Base de Datos*.
- gh-APIRest: Es un servidor web al que se pueden hacer peticiones *http*
- gh-tools: Es la caja de herramientas, las clases, que utilizan tanto gh-backend como gh-APIRest.

De esta manera se consigue independizar los sistemas según sus funciones y el proyecto queda mucho mejor definido.

6.2.3. Paquetes Python

En el prototipo 2, como se ha descrito en el apartado anterior, *gh-backend* y *gh-APIRest* utilizan las mimas clases y por lo tanto había que añadir a los dos proyectos las mismas clases e importarlas para poder usarlas. Esto supone que si modificas algo del código en un proyecto en el otro queda sin modificar por lo tanto es muy difícil de mantener, además de tener código repetido y otras desventajas.

La solución es crear un paquete Python [19] y instalarlo vía pip.

6.2.4. Logger

La clase *Logger* es una configuración propia de la clase de Python Logging. Sirve para crear ficheros *log*. Al usarlo en el prototipo 1 se repetían líneas en el *log*. El problema era que cada clase, al llamar al *logger*, creaba una instancia nueva. En el siguiente parte del fichero se puede ver cómo se repiten de tres en tres las entrads del log.

```
2018-03-06 11:13:33,517  ERROR    database.py    send_query      :87   (0, '')
2018-03-06 11:13:33,517  ERROR    database.py    send_query      :87   (0, '')
2018-03-06 11:13:33,517  ERROR    database.py    send_query      :87   (0, '')
2018-03-06 11:13:33,564  ERROR    nano.py        connect_bluetooth:33   (16, 'Device or resource busy')
2018-03-06 11:13:33,564  ERROR    nano.py        connect_bluetooth:33   (16, 'Device or resource busy')
2018-03-06 11:13:33,564  ERROR    nano.py        connect_bluetooth:33   (16, 'Device or resource busy')
2018-03-06 11:13:33,565  ERROR    nano.py        connect_bluetooth:33   (16, 'Device or resource busy')
2018-03-06 11:13:33,565  ERROR    nano.py        connect_bluetooth:33   (16, 'Device or resource busy')
2018-03-06 11:13:33,565  ERROR    nano.py        connect_bluetooth:33   (16, 'Device or resource busy')
2018-03-06 11:13:33,567  ERROR    nano.py        read_nano_bluetooth:92   (9, 'Bad file descriptor')
2018-03-06 11:13:33,567  ERROR    nano.py        read_nano_bluetooth:92   (9, 'Bad file descriptor')
2018-03-06 11:13:33,567  ERROR    nano.py        read_nano_bluetooth:92   (9, 'Bad file descriptor')
2018-03-06 11:13:35,223  ERROR    nano.py        connect_bluetooth:33   (104, 'Connection reset by peer')
2018-03-06 11:13:35,223  ERROR    nano.py        connect_bluetooth:33   (104, 'Connection reset by peer')
2018-03-06 11:13:35,223  ERROR    nano.py        connect_bluetooth:33   (104, 'Connection reset by peer')
2018-03-06 11:13:35,262  ERROR    nano.py        connect_bluetooth:33   (111, 'Connection refused')
2018-03-06 11:13:35,262  ERROR    nano.py        connect_bluetooth:33   (111, 'Connection refused')
2018-03-06 11:13:35,262  ERROR    nano.py        connect_bluetooth:33   (111, 'Connection refused')
```

Lo solución fue al hacer la llamada al *logger*, que esta a su vez, hiciera una llamada a la instancia superior, la primera, y utilizar la misma instancia, como se puede ver en la línea 16 del siguiente código.

```
1  import logging
2
3
4  class Logger():
5      def __init__(self, name: str = 'greenhouse', path: str = './log/'):
6          self.name = name
7          self.path = path
8
9      def init_logger(self) -> logging:
10         file_debug = self.path + self.name + '_debug.log'
11         file_info = self.path + self.name + '_info.log'
12         file_hist = self.path + self.name + '_error.log'
13         formatter = logging.Formatter('%(asctime)s %(levelname)-8s %(filename)-14s '
14                                       '%(funcName)-15s:%(lineno)-3s %(message)s')
15
16         logger = logging.getLogger(__name__)
17         logger.setLevel(logging.DEBUG)
18
19         fh1 = logging.FileHandler(filename=file_debug, mode='w+')
20         fh2 = logging.FileHandler(filename=file_hist, mode='a+')
21         fh3 = logging.FileHandler(filename=file_info, mode='a+')
22
23         fh1.setLevel(logging.DEBUG)
24         fh2.setLevel(logging.ERROR)
25         fh3.setLevel(logging.INFO)
26
27         fh1.setFormatter(formatter)
28         fh2.setFormatter(formatter)
29         fh3.setFormatter(formatter)
30
31         logger.addHandler(fh1)
32         logger.addHandler(fh2)
33         logger.addHandler(fh3)
```

```
34
35          return logger
36
37      def get_logger(self):
38          return logging.getLogger(__name__)
```

Listing 2: classs Logger

6.2.5. Fichero de Configuración

En el prototipo 1 el fichero de configuración tampoco era accesible para todos los ficheros que lo necesitaban debido a que las llamadas se hacían desde distintos ficheros que se encuentran en diferentes partes del árbol de ficheros como ya se explicó en 6.2.2.

La solución vino con nueva organización del proyecto 6.2.2. Ahora cuando se instala gh-tools los datos para la conexión con la base de datos así como otros, los carga del fichero de configuración que tiene el proyecto en el directorio ./etc evitando tener ningún dato no público en el paquete.

Como ejemplo de cómo es el fichero de configuración que debe haber en el directorio ./etc

```
1  [mqtt]
2  user = nombre_de_usuario
3  passw = contrasena_del_servicio
4
5  [topic]
6  sensor1 = greenhouse/sensor1/both
7  sensor2 = greenhouse/sensor2/both
8  sensor3 = greenhouse/sensor3/both
9  sensor4 = greenhouse/sensor4/flow
10 general = greenhouse/#
11
12 [server]
13 host = direccion_ip_del_servidor_mqtt
14 port = 9999
```

Listing 3: config.ini

6.2.6. RabbitMQ

No es posible conectarse al broker de RabbitMQ desde otra máquina después de haber instalado todo.

Para ello es necesario crear un usuario y dar los permisos necesarios con los siguientes comandos:

```
1 ~$ rabbitmqctl add_user NOMBRE_USUARIO CONTRASENA
2 ~$ rabbitmqctl set_user_tags NOMBRE_USUARIO administrator
3 ~$ rabbitmqctl set_permissions -p / NOMBRE_USUARIO ".*" ".*" ".*"
```

1. Primero se crea un usuario,
2. a continuación se le da rol de administrador, ya que es el usuario por defecto,
3. se le da permisos para configurar, leer y escribir.

6.2.7. Perdida de Conexiones

En las primeras versiones del prototipo 1, las conexiones se podían perder con facilidad. La conexión del microcontrolador, Arduino Nano, con la Raspberry Pi mediante Bluetooth y continuamente había que comprobar si la conexión está viva. En el caso de que no lo estuviese, volver a conectar el microcontrolador que la hubiera perdido, había más de uno conectados con el mismo sistema. También se podía perder la conexión con la base de datos, recordar que en esta fase del prototipo las inserciones se hacían directamente en la base de datos desde la Raspberry Pi, con lo cual también había que comprobarlo. La comprobación de las conexiones de todos los dispositivos así como con la base de datos era un verdadero quebradero de cabeza que se resolvía mediante dos bucles y banderas con el estado de conexión. Mediante este sistema se resolvía los problemas de pérdida de conexión pero no era una solución aceptable. Además para incluir nuevos sensores o actuadores había que añadir a los bucles las comprobaciones con sus respectivas banderas lo que hacía que fuera complicado añadir nuevos dispositivos. En la siguiente parte de código se puede ver cómo se resolvían las pérdidas de conexión y cómo se hacían las comprobaciones.

```
import pymysql
from configparser import ConfigParser
from app.modules.logger import create_log
from app.model.database import connect_db, send_data
from app.model.nano import connect_bluetooth, read_nano_bluetooth
from app.modules.flags import Flag

config = ConfigParser()
config.sections()
config.read('config.ini')
config.sections()
bluetooth_config = config['bluetooth']

db_addr1 = bluetooth_config['module1']
db_addr2 = bluetooth_config['module2']
port1 = int(bluetooth_config['port1'])
port2 = int(bluetooth_config['port2'])

logger = create_log('prototipo.log')


def main():
    cnx = pymysql.connect(host='ip_database_host', port=9999, user='data_base_user', password='pass',
                          db='data_base_name')
    while True:
        while Flag.connect_db and Flag.connect_db:
            sock1 = connect_bluetooth(db_addr1, port1)
            sock2 = connect_bluetooth(db_addr2, port1)

            while Flag.read_nano and Flag.send_data \
                    and Flag.connect_nano and Flag.connect_db:
```

```
                ambient1 = read_nano_bluetooth(sock1)
                send_data(cnx, ambient1)
                ambient2 = read_nano_bluetooth(sock2)
                send_data(cnx, ambient2)

                logger.info('sensor: {}'.format(ambient1['sensor']))
                logger.info('temp: {}'.format(ambient1['temperature']))
                logger.info('humi: {}'.format(ambient1['humidity']))
                logger.info('sensor: {}'.format(ambient2['sensor']))
                logger.info('temp: {}'.format(ambient2['temperature']))
                logger.info('humi: {}'.format(ambient2['humidity']))


if __name__ == '__main__':
    main()
```

Listing 4: raspi_control_sensor_v1

En este otro fichero se puede ver la misma funcionalidad pero ya implementado RabbitMQ con lo cual ya la conexión con la base de datos no se gestionaba desde la Raspberry Pi.

```
from threading import Thread
from modGreenhouse import Ambient

try:
    from app.clases_varias.element import *
    from app.clases_varias.connection import *
    from app.clases_varias.rabbitMQ import Sender
    from app.sockets.client_relay_state import relay_state
    from app.tools.config import *
    from app.tools.logger import create_log
except ImportError:
    from clases_varias.element import *
    from clases_varias.connection import *
    from clases_varias.rabbitMQ import Sender
    from sockets.client_relay_state import relay_state
    from tools.config import *
    from tools.logger import create_log

try:
    logger = create_log(webserver_logger)
except:
    logger = create_log(raspi_logger)


def main():
    # init objects
    bluetooth1 = M_bluetooth(bluetooth_module1)
    bluetooth2 = M_bluetooth(bluetooth_module2)

    sensor1 = Dht_22(1)
    sensor2 = Dht_22(2)

    rabbit_sender = Sender()

    # init socket to send relay state
```

```
    t1 = Thread(target=relay_state)
    t1.start()

    # Connect Bluetooth socks
    bluetooth1.connected()
    sleep(0.1)
    bluetooth2.connected()

    while True:

        logger.debug('outer while')
        if bluetooth1.get_state() == False:
            sleep(0.1)
            logger.debug('bluetooth1.get_state(): {}'.format(bluetooth1.get_state()))
            bluetooth1.sock.close()
            bluetooth1 = M_bluetooth(bluetooth_module1)
            bluetooth1.connected()
        elif bluetooth2.get_state() == False:
            sleep(0.1)
            logger.debug('bluetooth2.get_state(): {}'.format(bluetooth2.get_state()))
            bluetooth2.sock.close()
            bluetooth2 = M_bluetooth(bluetooth_module2)
            bluetooth2.connected()

        while Flag.inner_while:
            ambient1 = sensor1.read_ambient(bluetooth1)
            ambient2 = sensor2.read_ambient(bluetooth2)

            # Connect to rabbitMQ queue
            rabbit_sender.connected()
            rabbit_sender.send(ambient1)
            rabbit_sender.send(ambient2)
            rabbit_sender.connection.close()

            if (ambient1.temperature == 100 or ambient2.temperature == 100
                    or bluetooth1.get_state() == False or bluetooth2.get_state() == False):
                Flag.inner_while = False

        Flag.inner_while = True


if __name__ == '__main__':
    main()
```

Listing 5: raspi_control_sensor_v2

La solución paso por cambiar la topología de la aplicación, prototipo 2, donde ya no está la Raspberry Pi y son los propios *Objetos IoT* los que envían los datos mediante la conexión Wi-Fi y el protocolo MQTT. Así, todo el código que se ejecutaba en la Raspberry Pi ya no es necesario eliminando posibles puntos de fallo. Además cabe destacar que la modularidad ahora es una de las características de la nueva topología pudiéndose añadir nuevos *Objetos* con mucha facilidad. Solamente hay que poner al *Objeto* en la red convenientemente configurado y programado y añadir el *topic* al *gh-backend* como se puede ver en el siguiente código que es el servidor Mosquitto. Tal como se explica en su sección, es un servicio en el que se publica y se suscribe.

En el siguiente fichero se ve cómo cambia la forma de recolectar los datos y cómo ya no es necesario gestionar mediante bucles y banderas el estado de las conexiones. La biblioteca tiene un *client.loop_forever()* que gestiona el bucle y queda de manera transparente para el programador su gestión.

```
import json
from datetime import datetime
import configparser

import paho.mqtt.client as mqtt
from requests import get, put, post

from ghTools.irrigation import Irrigation
from ghTools.logger import Logger
from ghTools.climate import Climate

logger = Logger().init_logger()

config = configparser.ConfigParser()
config.read('./etc/config.ini')

list_flow = []
list_date_flow = []


def on_connect(client, userdata, flags, rc):
    print('Connected with result code {}'.format(rc))


def on_message(client, userdata, msg):
    print('Received message {} on topic {} with QoS {}'
          .format((msg.payload).decode(), msg.topic, msg.qos))


def on_message_flow(client, userdata, msg):
    data = msg.payload.decode()
    if data[0] == '{':
        try:
            data_dict = json.loads(data)
            ir = Irrigation(id_relay=data_dict['sensor'])
            ir.liters = data_dict['liters']

            if data_dict['liters'] is 0 and len(list_flow) > 1:
                ir.start = list_date_flow.pop(0)
                ir.end = list_date_flow.pop()
                item_tmp = list_flow.pop()
                logger.debug(item_tmp)
                item_dict = json.loads(item_tmp)
                ir.liters = item_dict['liters']

                logger.debug('start:\t{}'.format(ir.start))
                logger.debug('end:\t{}'.format(ir.end))
                logger.debug('liters:\t{}'.format(ir.liters))

                ir.insert_liters()

                list_flow.clear()
```

```
                list_date_flow.clear()

            else:
                list_flow.append(data)
                list_date_flow.append(datetime.now())
                logger.debug(data)

        except Exception as err:
            logger.error(err)
    else:
        logger.debug(data)
        print(data)


def on_message_sensor(client, userdata, msg):
    data = msg.payload.decode()
    if data[0] == '{':
        try:
            data_dict = json.loads(data)
            logger.debug(data_dict)
            climate = Climate(sensor=data_dict['sensor'])
            climate.temp = data_dict['temp']
            climate.humi = data_dict['humi']
            climate.insert_climate()
        except Exception as err:
            logger.error(err)
    else:
        logger.debug(data)
        print(data)


client = mqtt.Client()

client.message_callback_add(config.get('topic', 'sensor1'), on_message_sensor)
client.message_callback_add(config.get('topic', 'sensor2'), on_message_sensor)
client.message_callback_add(config.get('topic', 'sensor3'), on_message_sensor)
client.message_callback_add(config.get('topic', 'sensor4'), on_message_flow)

client.on_connect = on_connect
client.on_message = on_message

client.username_pw_set(username=config.get('mqtt', 'user'), password=config.get('mqtt', 'passw'))
client.connect(host=config.get('server', 'host'), port=config.getint('server', 'port'), keepalive=60)

client.subscribe(config.get('topic', 'general'), 1)

client.loop_forever()
```

Listing 6: mqtt-collector

6.2.8. Sockets

Las comunicaciones entre la Aplicación Web y la Raspberry Pi, prototipo 1, se hacían mediante socket. Esto servía para que desde la Aplicación web, el usuario pudiera apagar o encender el relé que hay en el sistema. Recordar que este relé se supone que activa una electroválvula con la cual se abre el riego. Además a través de otro socket se enviaba desde la Raspberry Pi hacía la Aplicación Web el estado del relé, de esta manera siempre se podía saber si estaba activo o no. Cada socket tiene su servidor siempre activo y su cliente. Esto supone tener más puertos abiertos en el router y contar con un servicio en medio por si cambia la IP ya que pueden ser fijas o no. En este caso la IP del servidor de la Aplicación Web sí lo es pero no el de la Raspberry Pi y por lo tanto hay que usar un servicio como No-IP. Otro gran inconveniente es que, no en todas partes es posible abrir puertos. Por ejemplo para hacer una demostración en la E.I.I no se podría hacer de esa manera.

La solución, como en el apartado anterior ha sido cambiar al prototipo 2, en donde ya no son necesarios los sockets ya que las comunicaciones entre *Objetos* y sus *manejadores* se hace a través del servidor *Mosquitto* y el protocolo *MQTT* añadiendo una capa de abstracción que permite no tener que implementar nada en esa parte debido a que ya existe una tecnología desarrollada por *IBM* y con muchos años de funcionamiento que además es *Open Source* que es *MQTT*. Esto nos permite centrar el desarrollo en el modelo de negocio.

7. Pruebas

La realización de esta sección se basa en validación de las historias de usuario de la metodología de desarrollo ágin Scrum.

7.1. Tipos de Pruebas

7.1.1. Pruebas de Caja Blanca

Las pruebas de caja blanca están ligadas al código fuente y su función es la comprobación del correcto funcionamiento de las funciones que se han implementado en la aplicación.

Estas pruebas se han ido haciendo durante la fase de implementación de las funcionalidades que el product owner solicitaba a través de las historias de usuario.

7.1.2. Pruebas de Caja Negra

Las pruebas de caja negra se centran en las funcionalidades de la aplicación. Estás pruebas se han hecho con el product owner o el desarrollador.

7.2. Resultados de las Pruebas

Para representar la validación de las historias de usuario se ha remarcado la casilla de validación en verde. Con ello se expresa que el Usuario afectado ha dado su conformidad a la misma y por lo tanto, se ha podido seguir con las siguientes historias de usuario.

Historia de Usuario	
Número: 1	Usuario: Product owner
Prioridad en negocio: Alta	Riesgo en desarrollo: Alto
Nombre	Control de un cultivo a través de web
Descripción	Quiero poder conectarme a una web y ver desde un móvil o un ordenador datos del cultivo como la temperatura y la humedad ambiental y poder abrir el riego. Se van a buscar soluciones técnicas para ver cómo se puede implementar un artefacto
Validación	Se le presenta al product owner que tecnologías se van a utilizar y debe aceptarlas

Tabla 35: Historia de usuario número 1

Historia de Usuario		
Número: 2		Usuario: Product owner
Prioridad en negocio: Alta		Riesgo en desarrollo: Alto
Nombre	Dispositivos necesarios	
Descripción	El product owner quiere saber que dispositivos se pueden necesitar para poder desarrollar el artefacto que ha pedido	
Validación	Se le enumeran los dispositivos que creemos necesarios para desarrollar el producto y debe aceptar	

Tabla 36: Historia de usuario número 2

Historia de Usuario		
Número: 3		Usuario: Product owner
Prioridad en negocio: Alta		Riesgo en desarrollo: Bajo
Nombre	Envío de datos de los sensores	
Descripción	Hacer una demo donde se pueda ver cómo están recogiendo datos los sensores y visualizarlos a través de la monitor del IDE de Arduino o de la terminal	
Validación	El product owner puede ver la temperatura y humedad ambiental a través de la pantalla y comprobar que cambia al calentar el sensor	

Tabla 37: Historia de usuario número 3

Historia de Usuario		
Número: 4		Usuario: Product owner
Prioridad en negocio: Media		Riesgo en desarrollo: Bajo
Nombre	Guardar las mediciones en una base de datos	
Descripción	Almacenar las mediciones que recogen los sensores y almacenarlos en una base de datos e incorporar otros datos que son útiles	
Validación	Se le muestra al product owner las mediciones que se han ido recogiendo en un plazo de tiempo	

Tabla 38: Historia de usuario número 4

Historia de Usuario		
Número: 5		Usuario: Product owner
Prioridad en negocio: Alta		Riesgo en desarrollo: Bajo
Nombre	Sketch de la web	
Descripción	Quiero ver gráficamente cómo sería la web antes de empezar con la implemetación	
Validación	Se le muestra al product owner los sketchs de cómo se ha diseñado la web. Le gusta el modelo y se puede empezar con su implementación	

Tabla 39: Historia de usuario número 5

Historia de Usuario		
Número: 6		Usuario: Desarrollado
Prioridad en negocio: Alta		Riesgo en desarrollo: Alto
Nombre	¿Cómo desarrollar la aplicación web?	
Descripción	Buscar información sobre desarrollo de aplicaciones web y tecnologías a usar	
Validación	Se le explica al product owner qué tecnologías se van a usar y por qué	

Tabla 40: Historia de usuario número 6

Historia de Usuario		
Número: 7		Usuario: Desarrollador
Prioridad en negocio: Alta		Riesgo en desarrollo: Medio
Nombre	Desarrollo de estructura de la web	
Descripción	Se desarrolla la estructura de la web sin funcionalidades solo con la vista principal	
Validación	Se le muestra a product owner como es la estructura de la web, como se pueden ir añadiendo vistas y los colores que se han elegido	

Tabla 41: Historia de usuario número 7

Historia de Usuario		
Número: 8		Usuario: Product owner
Prioridad en negocio: Media		Riesgo en desarrollo: Bajo
Nombre	Vista con una gráfica de la temperatura y la humedad	
Descripción	Se desarrolla dos vistas que muestren las mediciones de un intervalo de 24 horas en forma de gráfica, una con las mediciones de la temperatura y otra con las mediciones de la humedad.	
Validación	El producto owner comprueba que se muestran los datos de un día en las dos vistas y cómo, la temperatura es más alta en las horas diurnas que en las nocturnas.	

Tabla 42: Historia de usuario número 8

Historia de Usuario		
Número: 9		Usuario: Desarrollador
Prioridad en negocio: Media		Riesgo en desarrollo: Alto
Nombre	Seguridad base de datos	
Descripción	Evitar que un usuario del gestor de base de datos pueda acceder desde fuera del servidor	
Validación	El desarrollador comprueba que no se pueden hacer inserciones con el usuario anterior si se trata de hacer desde una máquina que no sea el servidor donde está el gestor de base de datos.	

Tabla 43: Historia de usuario número 9

Historia de Usuario		
Número: 10		Usuario: Product owner
Prioridad en negocio: Media		Riesgo en desarrollo: Medio
Nombre	Sensor inalámbrico	
Descripción	Quiero que los sensor de temperatura y humedad sean inalámbricos y que se conecten con la Raspberry Pi por Bluetooth	
Validación	El product owner comprueba que los sensores ahora son inalámbricos	

Tabla 44: Historia de usuario número 10

Historia de Usuario		
Número: 11		Usuario: Product owner
Prioridad en negocio: Alta		Riesgo en desarrollo: Medio
Nombre	Accionar un relé para controlar el riego	
Descripción	Quiero poder abrir y cerrar el riego desde de la web	
Validación	A modo de prototipo se monta un circuito, para que el product owner, al pulsar un elemento de la web accione un relé que maneja el encendido y apagado de un led	

Tabla 45: Historia de usuario número 11

Historia de Usuario		
Número: 12		Usuario: Product owner
Prioridad en negocio: Media		Riesgo en desarrollo: Alto
Nombre	Saber si el riego está accionado	
Descripción	Saber en todo momento si el circuito que controla el relé está cerrado o abierto y poder verlo en la web	
Validación	El product owner acciona el relé y enciende el led, a continuación se reinicia el servidor web, se accede a la vista de riego se pulsa el botón de estado y se comprueba que el estado que muestra es el correcto	

Tabla 46: Historia de usuario número 12

Historia de Usuario		
Número: 13		Usuario: Product owner
Prioridad en negocio: Media		Riesgo en desarrollo: Bajo
Nombre	Programar riego mediante reloj en la web	
Descripción	Insertar en la web unos relojes con los cuales se pueda programar el riego poniendo la hora y estableciendo la duración del riego	
Validación	El product owner ve como el la web se muestran uno relojes en los cuales puede poner la hora de riego y la duración de la misma	

Tabla 47: Historia de usuario número 13

Historia de Usuario		
Número: 14		Usuario: Product owner
Prioridad en negocio: Alta		Riesgo en desarrollo: Alto
Nombre	¿Se puede adaptar el sistema a IoT?	
Descripción	El product owner nos pregunta como sería la transición de la topología que hemos diseñado si creáramos un modelo orientado a IoT y con las comunicaciones a través de Wi-Fi	
Validación	Se muestra una topología orientada a IoT. Se muestran los dispositivos que se pueden usar con el fin de tener las comunicaciones vía Wi-Fi	

Tabla 48: Historia de usuario número 14

Historia de Usuario		
Número: 15		Usuario: Desarrollador
Prioridad en negocio: Alta		Riesgo en desarrollo: Alto
Nombre	Uso de dispositivos ESP8266 y MQTT	
Descripción	En una primera parte de adaptación del sistema a IoT, se cambian los Arduinos por ESP8266 y se implementa el protocolo MQTT, se envían los datos directamente al servidor de base de datos	
Validación	Se comprueba cómo se han cambiado los dispositivos anteriores por ESP8266 y que los datos, son mandados directamente al broker MQTT	

Tabla 49: Historia de usuario número 15

Historia de Usuario		
Número: 16		Usuario: Desarrollador
Prioridad en negocio: Alta		Riesgo en desarrollo: Bajo
Nombre	Orientación a Objetos de la Aplicación	
Descripción	Rediseñar la aplicación para que la Programación sea Orientada a Objetos	
Validación	A través del código se puede comprobar la Orientación a Objetos	

Tabla 50: Historia de usuario número 16

Historia de Usuario		
Número: 17		Usuario: Product owner
Prioridad en negocio: Alta		Riesgo en desarrollo: Alto
Nombre	Desarrollo de APIRest	
Descripción	Las necesidades cambian y ahora se pretende que la aplicación de servicio a otras aplicaciones mediante peticiones http	
Validación	Se hacen varias peticiones http con los servicios que se tenían anteriormente y se comprueba que llegan los datos de manera correcta y que se puede accionar el relé	

Tabla 51: Historia de usuario número 17

Historia de Usuario		
Número: 18		Usuario: Product owner
Prioridad en negocio: Alta		Riesgo en desarrollo: Medio
Nombre	Control del consumo de agua	
Descripción	Añadir un dispositivo nuevo con un sensor que mida el caudal del agua una vez que se haya activado el relé.	
Validación	El product owner comprueba que una vez activado el relé el caudalímetro empieza a contar los litros de agua que pasan por el sensor (esto se hace mediante una simulación y lo que hace mover la turbina del sensor es la acción de soplar por el mismo) viéndolo a través de una APP de Android	

Tabla 52: Historia de usuario número 18

Historia de Usuario		
Número: 19		Usuario: Product owner
Prioridad en negocio: Alta		Riesgo en desarrollo: Medio
Nombre	Programar riegos	
Descripción	Quiero poder programar los riegos y que si me equivoco y solapo dos riegos el sistema lo reconozca y que el sistema almacene de maneara persistente los riego y cuánto he regado en cada riego	
Validación	El product owner comprueba cómo se programan varios riegos, alguno se solapa y el sistema no lo permite dando un mensaje, y una vez que llega el tiempo del riego se ve cómo se activa el relé cerrando el circuito y activa el led en vez de la electroválvula y se sopla por el sensor de que mide el caudal del agua y se comprueba que la medición del riego se almacena correctamente en la base de datos.	

Tabla 53: Historia de usuario número 19

Historia de Usuario		
Número: 20		Usuario: Desarrollador
Prioridad en negocio: Alta		Riesgo en desarrollo: Alto
Nombre	Memoria del proyecto	
Descripción	Memoria completa del proyecto con todas las partes establecidas	
Validación	El proyecto es entregado en la secretaría de dirección el día 3 de septiembre.	

Tabla 54: Historia de usuario número 20

8. Caso de Uso

El caso de uso que va a presentar es la integración de la aplicación *greenhouse*, este proyecto, con la plataforma *Ubidots* que es un servicio para la presentación de datos e interacción con dispositivos IoT. Presenta grandes ventajas ya que ofrecen almacenamiento de datos recogidos por los *Objetos* y su posterior presentación en diversos formatos a través de su web. De esta manera simplifica en gran medida la presentación de los datos. Al usar esta plataforma se puede evitar el desarrollo de una web propia para mostrar gráficas y datos ya que es el servicio que ofrecen. Al usar esta plataforma se puede dejar de lado la parte de presentación de datos y hacer mayor esfuerzo en desarrollo de dispositivos con más funcionalidades y también en el modelo de negocio de la aplicación.

A continuación se va mostrar cómo con un mínimo cambio en el código de la aplicación *backend* se integra *greenhouse* con *Ubidots*.

Sólamente se han de hacer una pequeñas modificaciones en el código del *recolector de datos mqtt* (fichero6.2.7). Han continuación se van a mostrar los cambios hechos en el fichero anterior.

```
from ubidots import ApiClient

api = ApiClient(token=config.get('ubidots', 'default_token'))


def on_message_sensor(client, userdata, msg):
    data = msg.payload.decode()
    if data[0] == '{':
        try:
            data_dict = json.loads(data)
            logger.debug(data_dict)
            climate = Climate(sensor=data_dict['sensor'])
            climate.temp = data_dict['temp']
            climate.humi = data_dict['humi']
            climate.insert_climate()

            temp_id = config.get(str(data_dict['sensor']), 'temp_id')
            humi_id = config.get(str(data_dict['sensor']), 'humi_id')
            var_temp = api.get_variable(temp_id)
            var_humi = api.get_variable(humi_id)
            var_temp.save_value({'value': data_dict['temp']})
            var_humi.save_value({'value': data_dict['humi']})

        except Exception as err:
            logger.error(err)
    else:
        logger.debug(data)
        print(data)
```

Listing 7: mqtt-collector-ubidots

Para la intergración con *Ubidots* sólamente han hecho falta añadir 6 líneas de código en la función, además en el fichero de configuración hay que añadir los *ids* de las variables así como el *default key*. Con estos mínimos cambios en nuestra aplicación podemos visualizar los datos de nuestros sensores vía web.

Construir una vista con los datos que queramos resaltar de un sensor o varios en muy sencillo. De esta manera se pueden mostrar gráficas y datos de medias, máximas, mínimas u operaciones más complejas de una manera transparente para nosotros (figura 32). Nuestra parte es subministrar los datos de nuestros dispositivos a *Ubidots* y la plataforma nos mostrará los datos según los pidamos.

Ubidots dispone de cuenta para estudiantes lo cual permite su uso sin coste para desarrollos como el de esta aplicación.

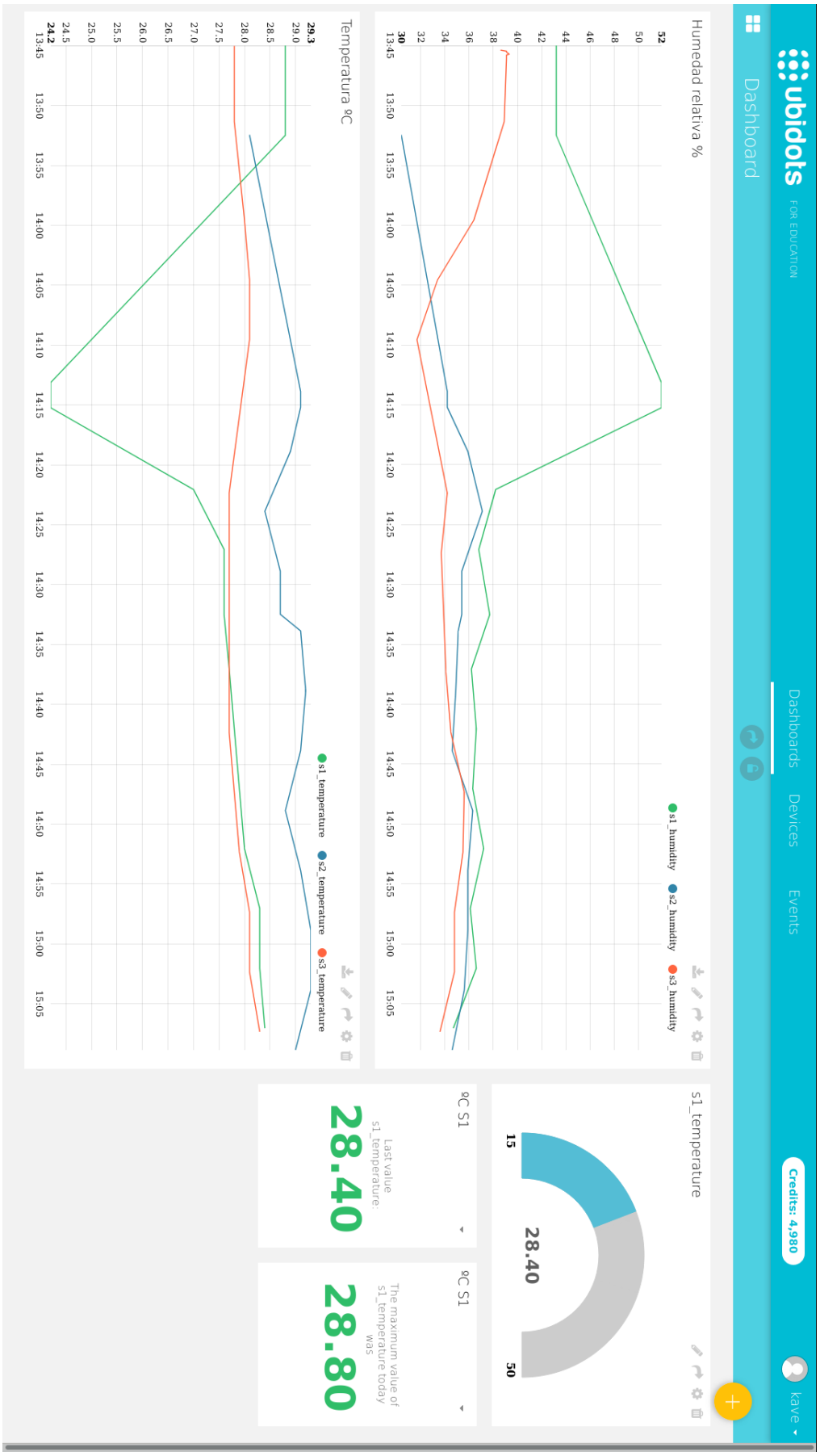

Figura 32: Vista de un Dashboard de Ubidots

Otro caso de uso es hacer varia peticiones http a la APIRest, como insertar un riego, pedir los últimos datos de algún sensor

Otra caso de uso va a ser hacer peticiones a la APIRest, para ello se va a usar la aplicación Postman. Las peticiones van a consistir en:

- insertar un riego correctamente mediante una petición *http post* (figura 33) con el resultado correcto,
- una vez iniciado la hora del riego, se activará el caudalímetro y registrará los litros que se han consumido en el riego, se podrá ver en la base de datos,
- el siguiente caso insertar un riego haciendo coincidir parte del riego con otro que ya esta programado (figura 34) con el resultado de error,
- ahora se pide mediante *GET* el último dato del sensor 3 de humedad (figura 35) con resultado correcto,
- y por último se pide el último dato de temperatura al sensor 3 (figura 36) con resultado correcto.

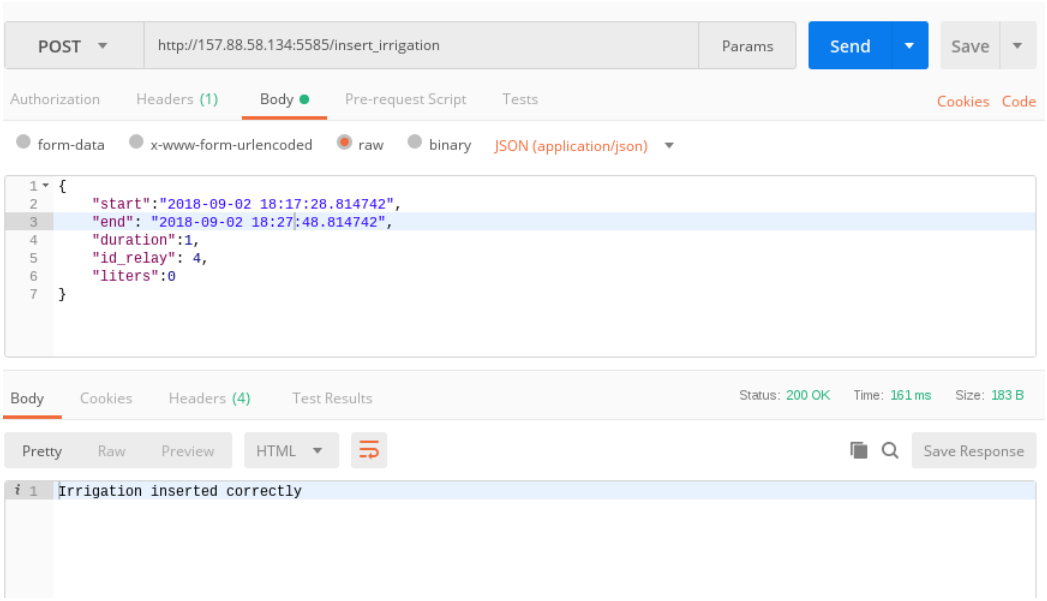

Figura 33: Insertar un riego

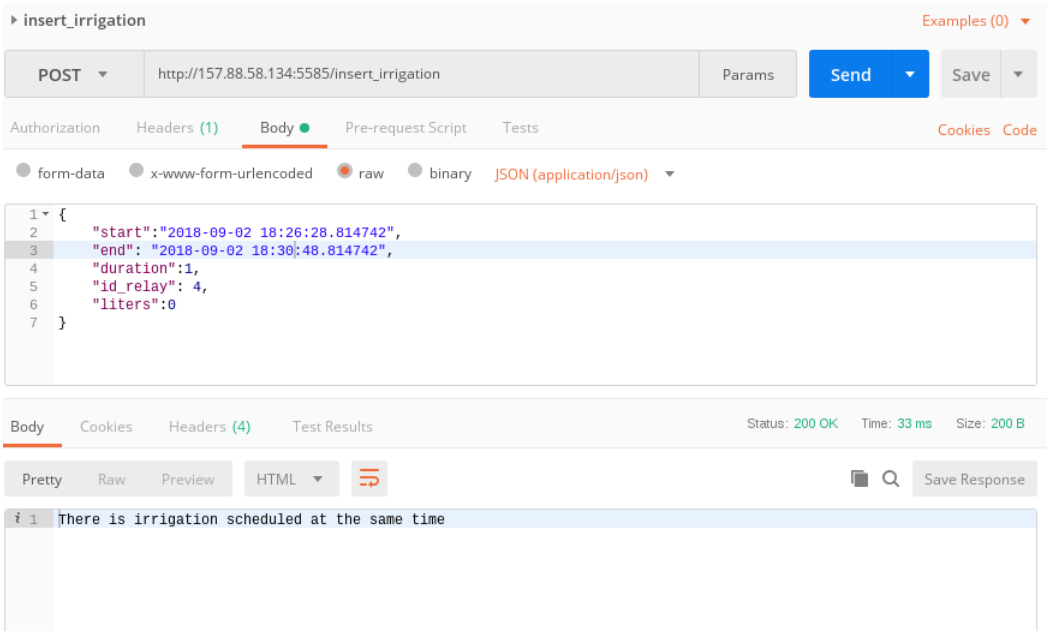

Figura 34: Insertar un riego error

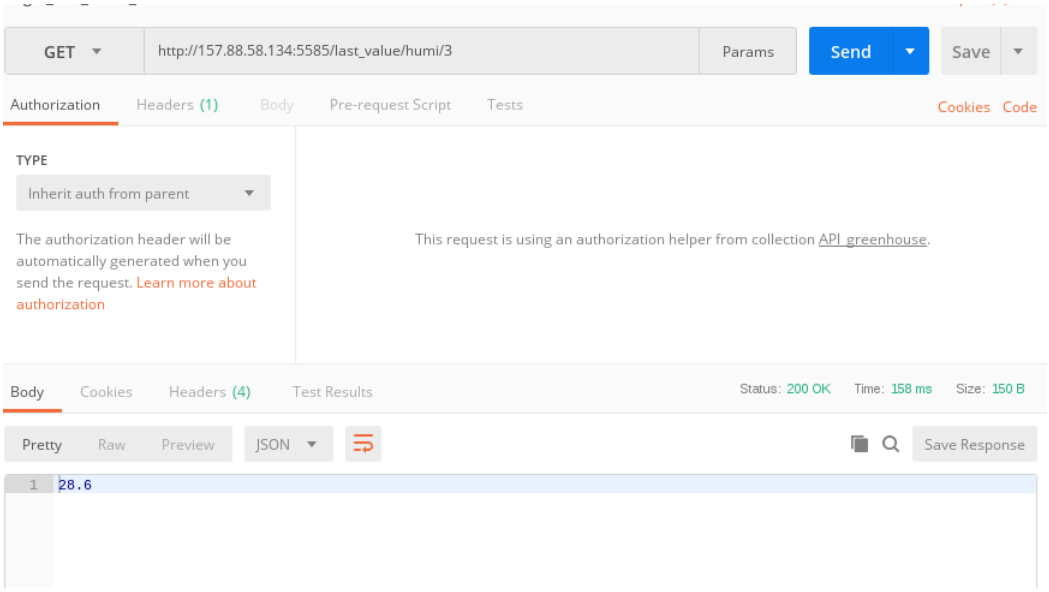

Figura 35: Último valor de humedad del sensor 3

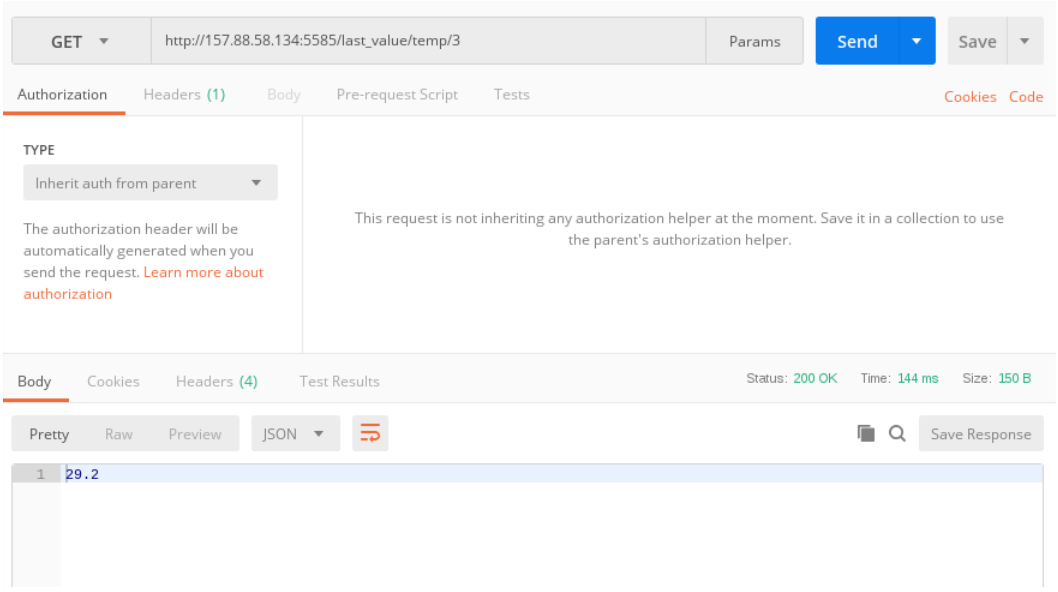

Figura 36: Último valor de temperatura del sensor 3

9. Conclusiones y Trabajo Futuro

En el siguiente apartado se indica el grado de cumplimiento de los objetivos propuestos en la subsección de Objetivos dentro de la introducción. Además, a modo de conclusiones se va a valorar los conocimientos adquiridos durante el desarrollo del proyecto tanto técnicos, a nivel de tecnologías, como conocimientos a nivel organizativos y de estructuración de proyectos. También se van a indicar que lineas se podrían seguir desarrollando en la aplicación implementada a rasgos generales, cuales han sido sus carencias y cómo se podría mejorar.

Lo primero se van a indicar qué objetivos se han cumplido de los propuestos en los *Objetivos*:

1. Hardware:
   - para adquirir todo el hardware que se ha usado en este proyecto, sin contar con la computadora con la cual se ha hecho la implementación, se habrá gastado no más de 110€.
   - todos los dispositivos utilizados se pueden adquirir en páginas web como `https://es.aliexpress.com/` u otras similares.
   - las comunidades que utilizan estos dispositivos son muy amplias y la red está trufada de webs, tutoriales, videotutoriales, etc. donde encontrar la solución a casi todo lo que busques.
   - la documentación para Arduino es muy completa y se puede usar la misma para otras placas.
2. Software:
   a) Cumplido, se pueden añadir módulos con pequeños cambios en el software,
   b) Cumplido, dispone de una interfaz para aplicaciones tipo *cliente*,
3. Prototipo:
   a) Cumplido, permite medir temperatura y humedad ambiental,
   b) No cumplido, no he podido conseguir un sensor que haga ese tipo de mediciones,
   c) Cumplido,los datos son accesibles vía peticiones http a través de la APIRest,
   d) Cumplido, se puede accionar el riego mediante programación del riego vía petición a la APIRest, o a través de un *Smartphone* con una aplicación para Android,
   e) Cumplida, los datos de riego quedan almacenados en la base de datos y se pueden visualizar en tiempo real mediante un *Smartphone* con una aplicación para Android.

El desarrollo de software es una disciplina harto compleja que hace que desarrollar software de *calidad* con todas las características que la definen como *software de calidad, funcionabilidad, confiabilidad, usabilidad, portabilidad, mantenibilidad y eficiencia*, sea complejo de hacer. Aún así considero que este proyecto cumple estas características. Se ha conseguido cumplir con estos requisitos después de haber cambiado la topología del sistema al haber encontrado soluciones ya existentes y muy utilizadas.

A lo largo de la implementación del proyecto se han adquirido grandes conocimientos sobre cómo elaborar un proyecto desde cero hasta su conclusión. Para ello han hecho falta todos los conocimientos teóricos adquiridos durante los años

transcurridos en la Escuela de Ingeniería Informática ya que con una mayor o menor dificultad se han comprendido los modelos de arquitectura con los cuales están diseñados las herramientas y tecnologías que se han utilizado a lo largo del presente proyecto. Más concisamente, si bien no se conocían la mayoría de las tecnologías que que se han usado sí entendía su funcionamiento tras leer documentación.

Ahora más concretamente, sobre este proyecto, que partió de una idea propia y a la cual se le han ido dando vueltas durante varios meses hasta iniciar su implementación, se podría decir que ha sido un proceso complejo. Ha habido cambios importantes en él, como por ejemplo la mutación que ha sufrido su topología durante el transcurso de la implementación del proyecto, debido a que mientras se desarrollaba, a la vez, también se investigaban soluciones existentes que resolvieran la problemática a resolver.

Así pues se han eliminado elementos de la topología inicial al utilizar el protocolo de comunicación estándar para *IoT MQTT* que es un protocolo de comunicación que ha permitido eliminar puntos de fallo en la aplicación. Anteriormente las comunicaciones se hacía mediante sockets y servicios de colas. Además, la utilización de esta topología permite, el poder añadir al sistema *Objetos* de IoT con mucha facilidad. Otro de los valores de este sistema es que se puede integrar en otras plataformas que dan soporte a aplicaciones IoT con relativa facilidad.

Líneas futuras de la aplicación:

- seguridad MQTT: proteger de manera correcta una aplicación da para otro TFG. En este caso destacar por ejemplo que la transmisión de datos entre los *Objetos* y el Broker MQTT se transmiten en plano y se podría securizar mediante *TLS/SSL*,
- seguridad APIRest: permitir el acceso a la APIRest a través de un token utilizando por ejemplo *JWT*,
- añadir sensores al sistema que den más información sobre el entorno,
- procesar los datos de la base de datos que para que estos datos obtengan un valor,
- añadir más servicios en la APIRest entre ellos la obtención de datos procesados,
- añadir otras clases con para ofrecer más funcionalidades.

Finalmente destacar que a nivel personal ha sido una experiencia muy satisfactorio el haber emprendido un proyecto personal y haberlo transformado en un TFG y con ello haber adquirido una gran cantidad de conocimientos.

10. Apendice

10.1. Manual de Usuario

Cómo se instala

Cómo se pone en marcha

En el siguiente manual se explica cómo se instalan y se lanzan las diferentes partes de la aplicación.

Los elementos se necesitan pueden ser variables ya que dependen del número de sensores y actuadores que se quieran conectar al sistema. Se van a listar los elementos que se han usado en este proyecto:

1. 4 unidades de ESP8266
2. 4 adaptadores de corriente con salida micro usb
3. 1 unidad de SRD-05VDC-SL-C
4. 1 unidad de YF S402
5. 3 unidades de DHT-22
6. 1 placa de pruebas (protoboard)
7. 1 conjunto de cables de tipo macho-macho, hembra-hembra y hembra-macho
8. 1 conjunto de resistencias
9. 1 led
10. 1 servidor o máquina virtual con IP fija
11. red Wi-Fi para la conectividad de los *Objetos*
12. 1 ordenar
13. 1 cable mini-usb usb

10.1.1. Backend

El backend se va considerar la parte del proyecto que se instala en el Servidor que debe tener IP fija. En el siguiente repositorio de GitHub, `https://github.com/kave06/tfg-gh-backend` en la rama develop, nos vamos a encontrar con los siguietes directorios y con el contenido que se va a detallar a continuación:

- database: hay un script para crear las tablas de la base de datos

- esp8266-12e: los programas para cargar en los dispositivos esp8266
- etc: fichero de configuración, hay que quitarle el guión bajo al nombre del fichero y poner los datos de nuestro servidor y nombre se usuario de *mosquitto* y su contraseña. También hay poner los topics a los cuales nos vamos a suscribir.
- mqtt-collector.py: el fichero que se encarga de manejar los datos que le llegan desde el servidor *mosquitto*
- requirements: el fichero donde están las dependencias del proyecto

Pasos a seguir para preparar los servicios que necesita el servidor:

1. el servidor debe tener un SO GNU/Linux, preferiblemente Debian o alguno de sus derivados ya que es en Debian donde se ha desarrollado el proyecto,
2. el servidor debe tener instalado el gestor de base de datos MySQL o MariaDB, preferentemente MySQL ya que es donde se han desarrollado el proyecto,
3. se instalará el *Broker MQTT* Mosquitto,
4. el servidor debe tener instalado *virtualenv* para la gestión de entornos virtuales de Python,
5. el servidor debe tener instalado *pip* para la gestión de paquetes Python.

Con estos pasos tendremos preparado el sistema con los servicios necesarios para que la aplicación pueda funcionar correctamente.

A Continuación se va a detallar cómo cargar los progrmas en el ESP8266. Antes de cargar los programas se tendrá que poner los datos de la red Wi-Fi, los datos del servidor *MQTT* que son los de *Mosquitto* y los topic de cada *Objeto* en el programas que se vaya a cargar. También se puede configurar cada cuanto envía los datos. Cada dispositivo debe tener un nombre único de cliente.

Cómo cargar los programas en los *ESP8266*:

1. instalar en el ordendor el IDE de Arduino y cargar las bibliotecas necesarias que están en los ficheros gh-backend/esp8266-12e/*
2. conectaremos el *ESP8266* al ordenador y cargaremos el programa que queramos de gh-tools/backend/esp8266-12e
   - para el ESP8266 y DHT-22 cargaremos el programa esp8266_mqtt_dht_relay_json
   - para el ESP8266 y DHT-22 y relé cargaremos el programa esp8266_mqtt_dht_relay_json
   - para el ESP8266 y caudalímetro cargaremos el programa esp8266_mqtt_flow_json
3.

Una vez que tenemos cargados los programas podemos cablear los ESP8266 con los sensores o actuadores.

A continuación se va a detallar cómo cablear los ESP8266 con el sensor DHT-22, con el caudalímetro, el relé y los demás componentes. Para ello se van a detallar los pasos a seguir:

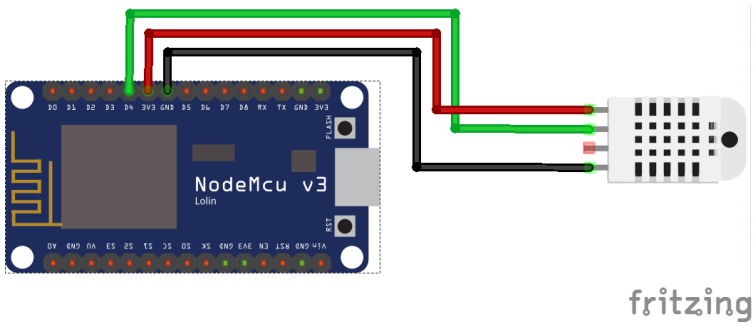

Figura 37: Cableado de ESP8266 con DHT-22

1. conectar los cables entre el *ESP8266* y el sensor de temperatura y humedad *DHT-22* tal como se ve en la figura 37,
2. conectar los cables entre el *ESP8266*, el relé y el *DHT-22* tal como se muestra en la figura 38,
3. conectar los cables entre el *ESP8266* y el caudalímetro tal como se muestra en la figura 39,
4.

Una vez que están cableados los dispositivos podemos ver a través del monitor del IDE de Arduino si se leen los datos de manera adecuada ya que los programas que se han cargado con anterioridad están en modo *debug* y por lo tanto tenemos lecturas de los dispositivos.

Una vez preparado el sistema y los *Objetos* vamos a crear la base de datos con sus tablas y lanzar el programa que se va a encargar de recolectar los datos de los sensores y los va a guardar en la base de datos. También se va a encargar de la comunicación con los actuadores.

Pasos a seguir para poner en funcionamiento la base de datos y el *backend*, *gh-backend*, de la aplicación.

1. crear una base de datos,
2. crear usuario con contraseña en MySQL y darle permisos al menos de *INSERT* y *SELECT* en la base de datos creada. Hay que tener en cuenta que el *recolector* debe de tener acceso a la base de datos y por lo tanto si no reside en el mismo servidor que el servidor de Base de Datos el usuario creado debe ser configurado para que tenga acceso desde el exterior,
3. cargar el fichero create_tables.sql que se encuentra en gh-backend/database, con ello tendremos las tablas necesarias en las base de datos. Antes de poder empezar a recibir datos de los sensores y actuar sobre los actuadores, tendremos que darlos de alta en la base de datos en la tabla *devices*.
4. se creará un usuario con un contraseña en el servicio del *Mosquitto*,
5. crear un entorno virtual de Python,
6. instalar mediante pip las dependencias necesarias que se encuentran en el fichero requirements,
7. descargar el paquete *gh-tools* de `https://github.com/kave06/tfg-gh-tools/tree/develop` o desde los ficheros adjuntos al proyecto,

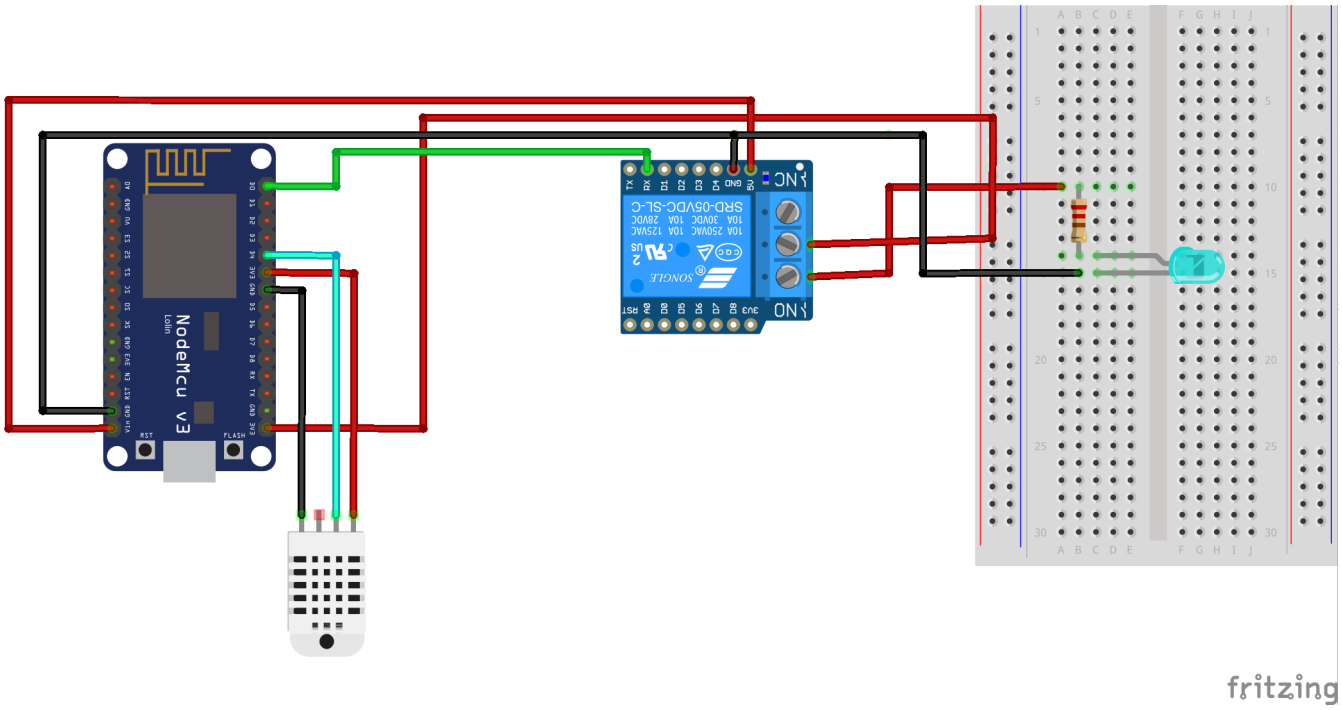

Figura 38: Cableado de ESP8266 con DHT-22 y relé

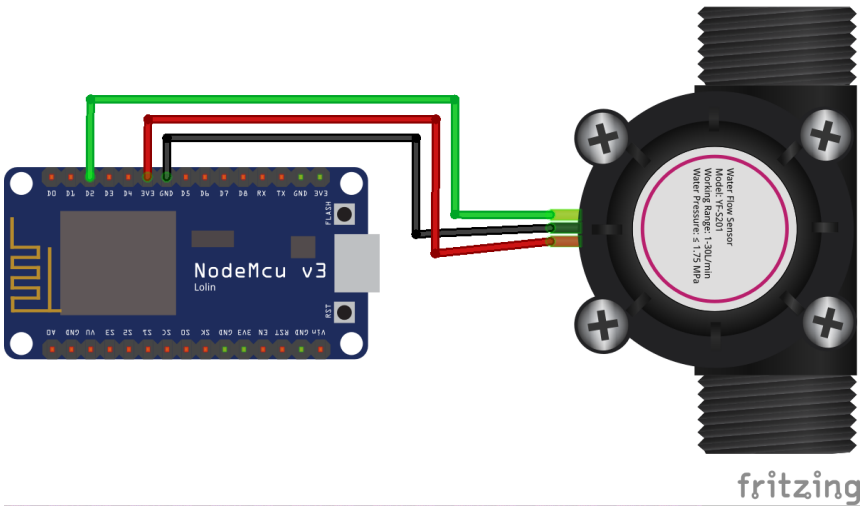

Figura 39: Cableado de ESP8266 con DHT-22

8. instalar el paquete Python *gh-tools* mediante pip,
9. poner los datos en el fichero de configuración;
    - Mosquitto:
        - nombre de usuario,
        - contraseña del usuario,
        - IP del servidor,
        - puerto donde escucha el servidor,
        - los diferentes *topic* que usan los dispositivos
    - MySQL:
        - nombre de usuario,
        - contraseña del usuario,
        - IP del servidor,
        - puerto donde escucha el servidor,
        - nombre de la base de datos.
10. se recomienda crear un servicio de sistema que se gestione mediante *systemctl* y que arranque al inicio del sistema con el programa gh-backend/mqtt-collector.py, si no lanzarlo como cualquier programa Python.

Llegados a este punto el servidor está listo esperando a que le lleguen datos para insertarlos en la base de datos o para interactuar con los actuadores.

10.1.2. APIRest

Pasos a seguir para poner en marcha el servidor con la APIRest:

1. descargar el proyecto de https://github.com/kave06/gh-APIRestFull/tree/develop o desde los ficheros adjuntos en el proyecto,
2. crear entorno virtual de Python,
3. instalar mediante pip las dependencias necesarias que se encuentran en el fichero requirements,
4. descargar el paquete gh-tools de https://github.com/kave06/tfg-gh-tools/tree/develop y con el entorno virtual acitvado intalar el paquete,
5. cambiar el nombre del fichero ./etc/config_.ini a config.ini,
6. poner los datos en el fichero de configuración;
    - Mosquitto:
        - nombre de usuario,
        - contraseña del usuario,

        - IP del servidor,
        - puerto donde escucha el servidor,
        - los diferentes *topic* que usan los dispositivos
    - MySQL:
        - nombre de usuario,
        - contraseña del usuario,
        - IP del servidor,
        - puerto donde escucha el servidor,
        - nombre de la base de datos.

7. activar el entorno virtual,
8. instalar mediante pip las dependencias necesarias que se encuentran en el fichero requirements,
9. descargar el paquete gh-tools de https://github.com/kave06/tfg-gh-tools/tree/develop y con el entorno virtual acitvado intalar el paquete,
10. lanzar el programa pricipal ./app.py mediante el comando:

```
(venv) user@debian: python app runserver -h 0.0.0.0 -p 5000 --threaded

```

Con estos pasos estaría en funcionamiento el servidor web con la APIRest funcionando y se podrían hacer peticiones http.

10.2. Manual del Programador

Estructura de ficheros y explicación

Qué es lo que debería saber un programador que quisiera utilizar tu TFG como punto de partida para desarrollos adicionales.

El proyecto se ha hecho en dos fases que han dado lugar a dos proyectos muy diferentes aunque las funcionalidades son las mismas o muy parecidas. En primer lugar se desarrollo un sistema llamado prototipo1 que pretende recoger los datos de unos sensores y interactuar con unos actuadores. Se desarrolló mediante funciones y utilizando diferentes ficheros, es decir no se hizo la Orientación a Objetos. También se utilizó diferentes soluciones para la comunicación entre los diferentes sistemas dando lugar a un *sistema distribuido* y para ello se utilizaron *sockets* y un servicio de colas como *RabbitMQ*.

La segunda versión del proyecto, llamado prototipo2, la funcionalidad es muy parecida sólo que ahora el proyecto tiene una arquitectura mucho más adecuada y se evitan mucho puntos de fallo al eliminar elementos que no son necesarios en esta nueva arquitectura. Además el enfoque ahora es *Orientado a Objetos*

10.2.1. Prototipo1

El proyecto se ha desarrollado en un único proyecto clasificando sus partes a través de directorios como método de división. A diferencia del prototipo2 17 la interfaz para usar la aplicación es una interfaz web destinada a usuarios finales.

A continuación se va a hacer una descripción de los directorios y ficheros que componen el proyecto.

app

En el directorio app se encuentran todos los ficheros que no son de la aplicación web. Dentro de este directorio se encuentran otros directorios y algunos ficheros que se van a detallar a continuación

Ficheros en este directorio:

- databaseserver_sensor_consumer: este fichero lanza al consumidor del servicio de colas. Se ejecuta en el servidor de bases de datos. Recibe los datos de los sensores y mediante los *callbacks* procesa los mensajes,
- raspi_control_sensor: es el programa principal, collector de datos, se ejecuta en la Raspberry Pi y recibe los datos de los sensoresy los envía. Una tarea muy importante que tiene que cumplir este programa es que si falla cualquier elemento pueda recobrarse de fallo, es decir tiene que estar continuamente comprobando las conexiones tanto con los sensores, que es a través de Bluetooth, como con el servidor de bases de datos, que es a través del servicio de colas. También tiene que poder recuperase si algún elemento hardware falla como el módulo HC-05, el microcontrolador Arduino Nano o el sensor DHT-22. Esto se consigue mediante bucles infinitos y banderas, tal como se puede ver en el fichero situado en el apartado 6.2.7,

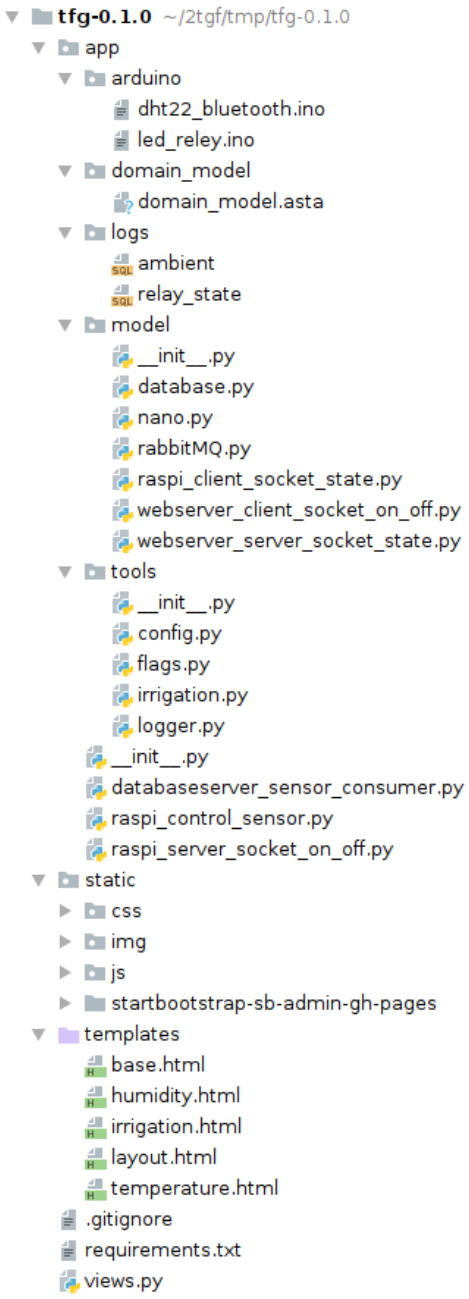

Figura 40: Árbol de ficheros del prototipo1

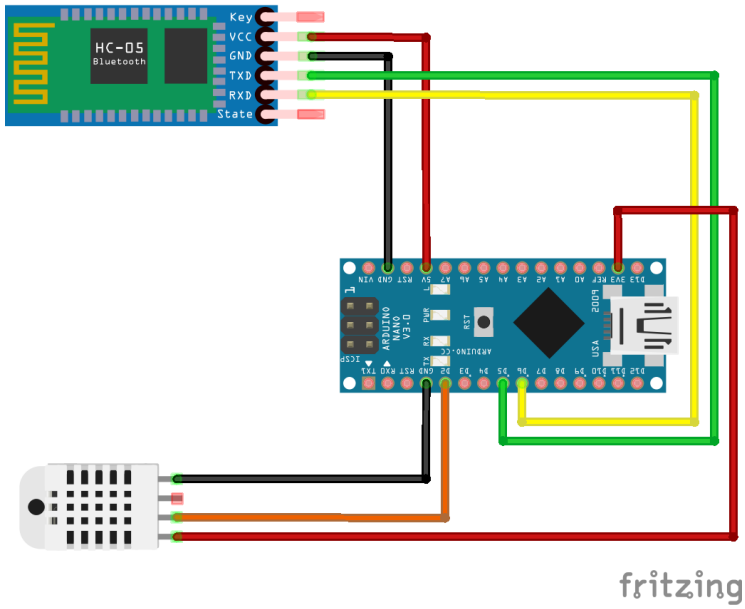

Figura 41: Esquema de cableado Ardunino Nano, sensor DHT-22 y módulo bluetooth HC-05

- raspi_server_socket_on_off: el servidor que escucha continuamente en la Raspberry Pi para apagar o enceder el relé los mensajes que le llegan desde la aplicación web, la comunicación se hace a través de un socket.

arduino

En este directrio se encuentran los programas para los microcontroladores Arduino.

- dht22_bluetooth: es el programa que se ejecuta en el Arduino y que lee los datos del sensor DHT-22 y los envía a través de módulo Bluetooth HC-05.
- led_relay: este Arduino está conectado mediante cable serie con la Raspberry Pi que es quien le manda el mensaje de encender o apagar el relé.

La siguiente figura 41 es el esquema de conexiones entre el Arduino Nano, el módulo de Bluetooth HC-05 y el sensor DHT-22. se puede ver cómo están cableados los elementos que miden la temperatura y humedad ambiente y la envían a la Raspberry Pi de manera inalámbrica a través de Bluetooth.

En la figura 42 se puede ver cómo es el esquema de conexión de los diferentes dispositivos. La Raspberry Pi envía la señal a través del puerto serie por un cable USB-mini_usb y el Arduino envía la señal a pin D7 a la entrada de señal del relé RX y el relé al llegarle la señal cierra el circuito y enciende el led. Con este mecanismo se podría activar cualquier otro elemento electrónico por ejemplo una electroválvula situada en una tubería y que abriera el riego una vez cerrado el circuito.

model

En este directorio se encuentran diversos ficheros que se van a detallar a continuación sus funciones.

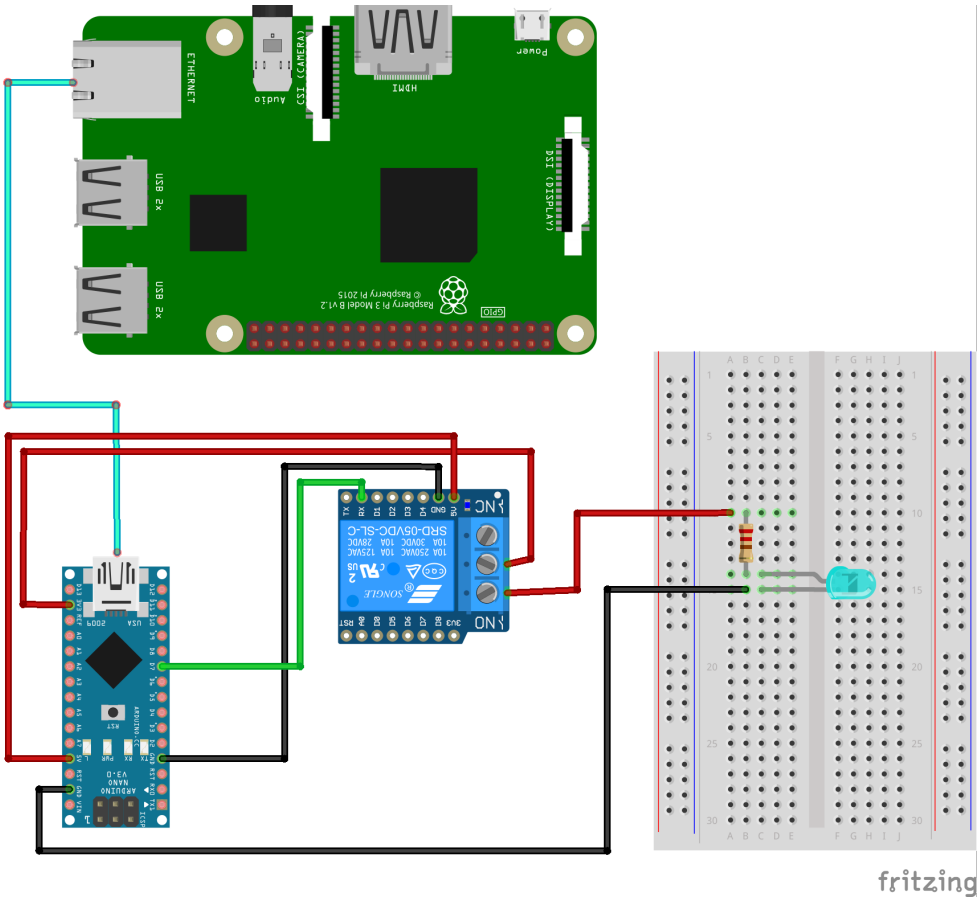

Figura 42: Esquema de cableado Raspberry Pi, Ardunino Nano y relé

- database: es la encargada de interaccionar con la base de datos,
- nano: contiene todas las funcionalidades que debe tener el dispositivo Arduino Nano para el funciomiento de la aplicación. Este programa se ejecuta en la Raspberry Pi según se ve en la figura 17. Se van a detallar sus funciones.
  - connect_bluetooth(): conecta con un dispositivo Nano a mediante la MAC del módulo HC-05,
  - connect_serial(): conecta con un dispositivo Nano mediante conexión Serial,
  - send_signal(): envía una señal al relé para cerrar o abrir el circuito,
  - read_nano_bluetooth(): lee los datos del sensor DHT-22 y se envían a través del módulo HC-05 conectado al dispositivo Nano,
  - read_serial_state(): devuelve el estado del relé, si está activado o no.
- rabbitMQ: Tiene las funcionalidades para interactuar con el servicio de colas RabbitMQ, tanto para enviar como para recibir. Estas funcionalidades se usan tanto en la Raspberry Pi como en el servidor de bases de datos,
- raspy_client_socket_state: este fichero es el cliente de un socket. Está continuamente leyendo el estado del relé y enviándolo a través de un socket a la aplicación web que a su vez tiene un servidor escuchando continuamente y enviando la señal. De esta manera si cambia de estado el relé la aplicación lo sabe porque está leyendo el estado directamente desde el Arduino Nano conectado al relé, a través de la Raspberry Pi. Si no llegase esta señal sería muestra de que algo está fallando.
- webserver_client_socket_on_off: es el cliente de un socket que desde la aplicación web manda una señal al servidor que escucha en ese socket con señal de apagado o encendido del relé,
- webserver_server_socket_state: un servidor en la aplicación web que está escuchando y continuamente recibe el estado del relé,

tools

El directorio tools esta a modo caja de herramientas que usa la aplicación. A continuación se van a detallar los ficheros contenidos en el directorio:

- config: contiene configuraciones de las distintas partes de la aplicación,
- flags: banderas para controlar los estados de las conexiones de Bluetooth entre la Raspberry Pi los Arduino Nano,
- irrigation: es una clase que puede enceder o apagar el relé,
- logger: crea un log para el sistema.

10.2.1.1. Aplicación web A continuación están los ficheros de la aplicación web desarrollada con el microframework para Python Flask. La distribución de directorios es las siguiente:

- static: se encuentran los css, imágenes y los ficheros JavaScript,
- templates: los ficheros html auque el código está desarrado en Jinja2,

A continuación esta el fichero views con el cual se lanza la aplicación es donde se crean las URLs. El el código se puede ver cómo se crean las URLs y cómo se devuelven los diferentes html y los datos necesarios para cada vista y cómo se hacen las llamadas a los métodos de la aplicación.

```
from threading import Thread

from flask import Flask, render_template, request, jsonify
from flask_script import Manager
from flask_bootstrap import Bootstrap

from app.model.database import ambient_days
from app.model.webserver_client_socket_on_off import relay_on_off
from app.tools.logger import create_log

from app.tools.flags import Var
from app.model.webserver_server_socket_state import launch_socket_relay_state
from app.tools.config import *


app = Flask(__name__)

app.secret_key = 'secret key'

manager = Manager(app)
bootstrap = Bootstrap(app)

try:
    logger = create_log(webserver_logger)
except:
    logger = create_log(raspi_logger)


@app.route('/temperature')
def temperature():
    data_list1 = ambient_days(1, 'sensor1_per_hour')
    data_list2 = ambient_days(1, 'sensor2_per_hour')
    list_temp1 = data_list1[0]
    list_temp2 = data_list2[0]
    list_hour1 = data_list1[2]
    list_hour2 = data_list2[2]
    return render_template('temperature.html', list_temp1=list_temp1, list_hour1=list_hour1,
                           list_temp2=list_temp2, list_hour2=list_hour2)


@app.route('/humidity')
def humidity():
    data_list1 = ambient_days(1, 'sensor1_per_hour')
    data_list2 = ambient_days(1, 'sensor2_per_hour')
    list_humi1 = data_list1[1]
    list_humi2 = data_list2[1]
    list_hour1 = data_list1[2]
    list_hour2 = data_list2[2]

    return render_template('humidity.html', list_humi1=list_humi1, list_hour1=list_hour1,
                           list_humi2=list_humi2, list_hour2=list_hour2)

```

```
53
54 @app.route('/handle_data', methods=['POST'])
55 def handle_data():
56     state = request.form['irrigation_state']
57     logger.info('state is: {}'.format(state))
58     relay_on_off(state)
59     return render_template('irrigation.html', relay_state=Var.RELAY_STATE)
60
61
62 @app.route('/irrigation')
63 def irrigation():
64     return render_template('irrigation.html', relay_state=Var.RELAY_STATE)
65
66
67 @app.route('/')
68 def dashboard():
69     data_list1 = ambient_days(1, 'sensor1_per_hour')
70     data_list2 = ambient_days(1, 'sensor2_per_hour')
71     list_temp1 = data_list1[0]
72     list_temp2 = data_list2[0]
73     logger.info(list_temp1)
74     list_hour1 = data_list1[2]
75     logger.info(list_hour1)
76     list_hour2 = data_list2[2]
77     logger.info(list_hour2)
78     return render_template('temperature.html', list_temp1=list_temp1, list_hour1=list_hour1,
79                            list_temp2=list_temp2, list_hour2=list_hour2)
80
81
82 @app.route('/relay_state')
83 def relay_state():
84     return jsonify(state=Var.RELAY_STATE)
85
86
87 @app.route('/irrigation_hour')
88 def add_numbers():
89     hour = request.args.get('hour')
90     print('hour: {}'.format(hour))
91     return jsonify(result=hour)
92
93
94 @app.route('/duration')
95 def duration():
96     duration = request.args.get('duration')
97     print('duration: {}'.format(duration))
98     return jsonify(result=duration)
99
100
101 if __name__ == '__main__':
102     t1 = Thread(target=launch_socket_relay_state)
103     t1.start()
104
105     manager.run()
```

Listing 8: views

10.2.2. Prototipo2

El proyecto está divido en tres partes que comprenden las diferentes partes del mismo que son las siguientes:

- gh-tools: comprende las clases y las herramientas que necesita la aplicación para funcionar,
- gh-backend: comprende la parte del servidor de la aplicación,
- gh-APIRest: hace de interfaz el los clientes de la aplicación.

A continuación se van a explicar estas partes en detalle, su contenido y función.

10.2.2.1. gh-tools Es un paquete Python con todas las clases que necesita la aplicación para funcionar a modo de herramientas. Está pensado para que se pueda instalar mediante el gestor de paquetes de Python *pip*. El poder instalarlo como un paquete en estructura del proyecto es una gran ventaja ya que permite tener este código con las clases necesarias en las otras partes del proyecto de manera más cómoda que si se tuviera que tener como parte del proyecto. El mantenimiento es más sencillo ya que sólo hay que modificar en un lugar aunque luego hay que instar la nueva versión del paquete en el entorno virtual.

A continuación de va a detallar el contenido del paquete con sus ficheros y su funcionalidad.

setup

Es un fichero de configuración para la creación de paquetes Python.

Climate

Es la representación de los datos de una medición del sensor DHT, además de la fecha y hora en que se han tomado esos datos.

Atributos de la clase:

- sensor: número del sensor del que vienen los datos,
- date: fecha y hora a la que llega el dato a la aplicación,
- temp: temperatura ambiental recogida por el sensor medida en ºC,
- humi: humedad relativa ambiental recogida por el sensor medida en %,
- logger: recoge la instancia de logger del programa que haya creado la primera instancia del Logger.
- model: Inicializa una instancia de la clase privada _Model() con el cual se puede comunicar con la base de datos.

Funciones de la clase:

- get_last_items(): devuelve los últimos items que se han insertado en la base de datos con el número de sensor del objeto, como parámetro se le pasan los días que se quieran,
- insert_climate(): inserta los datos del objeto en la base de datos,
- serialize(): convierte un objeto Climate en formato JSON,
- deserialize(): convierte un JSON en un objeto Climate,
- print():

Irrigation

La clase Irrigation representa un riego en el sistema. Está asociado a un relé que es el que acciona y de esta manera puedes tener varias instancias de Irrigation que pueden accionar cada una un relé diferente pudiendo controlar el riego de varias lineas de riego. Cada riego se almacena en la base de datos de modo que se puede obtener un histórico de riegos. Además el sistema almacena los riegos en un calendario y los acciona en el momento fijado y durante el tiempo que se fije.
También queda almacenado en la base de datos cuántos litros se han regado en cada riego pudiendo saber los litros de riego mensuales por ejemplo.

Atributos de la clase:

- relay: inicializa una instancia de la clase Relay con el cual se asocia un riego a un relé determinado para poder accionar ese relé,
- star: cuándo se va ha iniciar el riego en formato de fecha y hora,
- duration: durante cuánto tiempo va a estar el relé tiene el circuito cerrado,
- liters: cuántos litros ha medido el caudalímetro en ese ciclo de riego,
- logger: recoge la instancia de logger del programa que haya creado la primera instancia del Logger.
- model: Inicializa una instancia de la clase privada _Model() con el cual se puede comunicar con la base de datos.

Funciones de la clase:

- set_irrigation(): se conecta con el relé asociado a la clase y lo acciona,
- insert_irrigation(): inserta en la base de datos los datos del riego pero antes de hacerlo comprueba que en ese intervalo no haya otra riego programado con ese relé,
- add_scheduler(): inserta en el planificador los datos del riego para que se ejecuten en el momento fijado,
- insert_leters(): inserta en la base de datos los litros que ha medido el caudalímetro asociados a esa instancia de riego,

Logger

Logger es una clase que sirve para tener un único *log* en cada parte de la aplicación. Cada clase tiene un atributo de este tipo lo que permite su uso de manera muy sencilla. Tal y como está configurada la clase crea tres distintos *logs* con diferentes niveles cada uno, debug, info y error.

Model

Model es la clase privada que se encarga de la interacción con la base se datos. Es utilizado por las otras clases que lo usan para interactuar con la base de datos. Los objetos que utilizan esta clase crean una instancia de la misma con la cual utilizan sus funciones.

La configuración de los parámetros con los que una instancia de esta clase se conecta con la base de datos encuentra en el fichero de configuración ./etc/config.ini, que es donde están todos los datos de configuración que necesita al aplicación.

Atributos de la clase:

- host: dirección IP del servidor de base de datos,
- port: puerto en el que escucha el sistema gestor de base de datos,
- user: usuario que se usa para interactuar con la base de datos,
- passw: contraseña del usuario,
- db: nombre de la base de datos,
- logger: recoge la instancia de logger del programa que haya creado la primera instancia del Logger.
- __my_connect(): inicializa la conexión a la base de datos.

Funciones de la clase:

- __my_connect(): crea una conexión nueva con la base de datos. Cada instancia _Model tiene una conexión con la base de datos,
- __insert(): hace una inserción en la base de datos,
- __select(): hace un select sobre la base de datos,
- select_climate(): devuleve un result set con una lista de los datos de temperatura, humedad, timestamp de un sensor determinado. Como parámetro se le pasa un entero con los días que se quieren los datos,
- insert_climate(): se inserta en la base de datos los datos de una instacia de Climate,
- insert_irrigation(): inserta en la base de datos lo datos de una instancia de Irrigation comprobando antes que en ese intervalo no haya ya uno insertado,
- __check_irrigation_collision(): comprueba con los datos de la instancia de Irrigation si en la base de datos ya existe otro riego en ese intervalo,

- __search_irrigation(): busca con los datos de la instancia de Irrigation si en la base de datos hay un riego con esos datos. Se usa para insertar es esa línea de la base de datos los litros de agua que se han regado,
- insert_liters(): inserta en la base de datos los litros correspondientes a los datos de la instancia de Irrigation,
- get_last_temperature(): devuelve el dato de la última temperatura de la instancia de la clase Sensor con la que se ha hecho la llamada,
- get_last_humidity():devuelve el dato de la última de humedad de la instancia de la clase Sensor con la que se ha hecho la llamada,
- add_liter_irrigation(): inserta los litros de un riego una vez que el planificador ha lanzado el riego y el caudalímetro haya medido los litros,
-

Relay

Relay es la abstracción de un objeto relé de la realidad. Se puede accionar, cerrar y abrir su circuito, es decir encender o apagar el elemento al que esté asociado. La conexión con el relé se hace a través del protocolo MQTT. Actualmente solamente lo utiliza la clase Irrigation para iniciar y terminar el riego pero se podría asociar a cualquier otra clase que necesitara accionar un relé.

Atributos de la clase:

- id: es el identificador por el que se lo conoce en la aplicación y por el cual se puede conectar a él a través del protocolo MQTT. Este dato esta en la base de datos cuando se le da de alta en la tabla *devices*,
- state: el estado de circuito, si esta cerrado o abierto, se codifica como *ON* o *OFF*
- logger: recoge la instancia de logger del programa que haya creado la primera instancia del Logger.
-

Funciones de la clase:

- set_state(): conecta mediante el protocolo de MQTT con el *Broker de Mosquitto* y publica en el *topic* el estado del relé.

Sensor Sensor es la clase de modela un sensor físico en el mundo real dentro de la aplicación con el cual te puedes comunicar pedirle datos que ha almacenado en la base de datos.

Atributos de la clase:

- id: es el identificador por el que se lo conoce en la aplicación y por el cual se puede conectar a él a través del protocolo MQTT. Este dato esta en la base de datos cuando se le da de alta en la tabla *devices*,

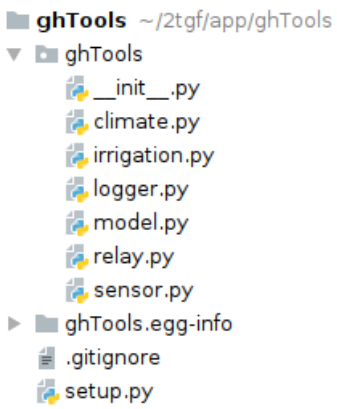

Figura 43: Árbol de ficheros de gh-tools

- __model: Inicializa una instancia de la clase privada _Model() con el cual se puede comunicar con la base de datos.

Funciones de la clase:

- get_last_temperature(): hace una llamada a la función get_last_temperature() de la clase _Model,
- get_last_humidity(): hace una llamada a la función get_last_humidity() de la clase _Model,
- get_last_climate(): hace una llamada a la función select_climate() de la clase _Model.

10.2.2.2. gh-backend Es la parte de la aplicación que reside en el servidor de MQTT, donde está instalado el *Broker Mosquitto*. Principalmente se encarga de recolectar los datos que le envían los sensores y almacenarlos en la base de datos. También es el encargado de interactuar con los actuadores a través del Mosquitto.

A continuación se van a explicar en detalle los ficheros que componen gh-backend, en la figura 44 se pueden ver.

database

En este directorio se encuentra el script para la estructura de la Base de Datos que creará las tablas con sus restricciones. Se creará la tabla *devices* en la cual hay que dar de alta los dispositivos que se usan en la aplicación. Se le asignará un id único, se insertará dónde esta situado dicho elemento y el tipo del dispositivo así la mac del esp al que está conectado.

esp8266-12e

Es este directorio se encuentran los programas que se ejecutan en los dispositivos ESP8266. A continuación se puede ver el código completo del programa que se ejecuta en el microcontrolador, el fichero esp8266_mqtt_dht_relay_json.

```
1 /*
2  Author Kave Heidarieh Sorosh
3
4  This program is a part of an End of Degree Project developed already, that
```

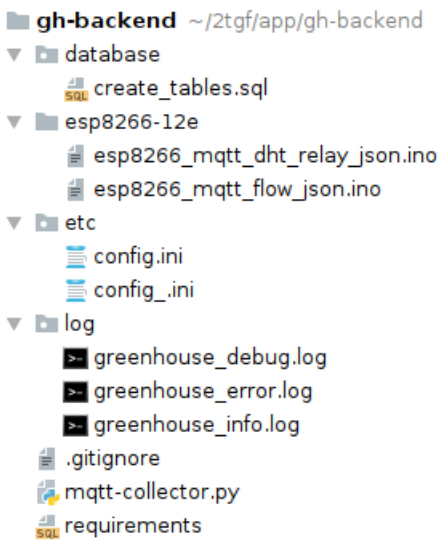

Figura 44: Árbol de ficheros de gh-backend

```
 controls a greenhouse with IoT devices using MQTT protocol and an APIRest.
 For further information about the code, check the URLs shown bellow:
 https://github.com/kave06/tfg-gh-backend/tree/develop
 https://github.com/kave06/tfg-gh-tools/tree/develop
 url con API

This programm is based on a project that have MIT licence:
https://projetsdiy.fr/esp8266-dht22-mqtt-projet-objet-connecte/

Licence: GNU GPLv3

*/


#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include "DHT.h"

#define wifi_ssid "xxxxxxxxxx"
#define wifi_password "xxxxxxxxx"

#define mqtt_server "xxxxxxxxxx"
#define mqtt_user "xxxxxxxxxx"
#define mqtt_password "xxxxxxxxxx"

/************ Change number of sensor and clietName with the correct number of sensor **************/
int sensor = 3;
#define temperature_topic "greenhouse/sensor3/temperature"  //Topic temperature
#define humidity_topic "greenhouse/sensor3/humidity"        //Topic humidity
#define both_topic "greenhouse/sensor3/both"                //Topic both
#define relay_topic "greenhouse/relay4"                     //Topic of relay
char *clientName = "ESP8266-12e_3";
/*********************************************************************************************/

```

```
//Buffer to decode MQTT messages
char message_buff[100];

long lastMsg = 0;
long lastRecu = 0;
long timeBetweenSend = 60 * 5;  // in seconds
bool debug = true;              //Display log message if True
String both = "";


// Un-comment you sensor
#define DHTTYPE DHT22
#define DHTPIN D4               // DHT Pin
#define D0 16                   // Relay Pin

// Create abjects
DHT dht(DHTPIN, DHTTYPE);
WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
  Serial.begin(9600);
  pinMode(D0,OUTPUT);
  setup_wifi();                           //Connect to Wifi network
  client.setServer(mqtt_server, 5578);    // Configure MQTT connexion
  client.setCallback(callback);           // callback function to execute when a MQTT message
  dht.begin();
}
//Connetion to WiFi network
void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(wifi_ssid);

  WiFi.begin(wifi_ssid, wifi_password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi OK ");
  Serial.print("=> ESP8266 IP address: ");
  Serial.print(WiFi.localIP());
}

//Reconnetion
void reconnect() {

  while (!client.connected()) {
    Serial.print("Connecting to MQTT broker ...");
    if (client.connect("ESP8266Client", mqtt_user, mqtt_password)) {
      Serial.println("OK");
    } else {
```

```
      Serial.print("KO, error : ");
      Serial.print(client.state());
      Serial.println(" Wait 5 secondes before to retry");
      delay(5000);
    }
  }
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  long now = millis();
  // Send a message every ...
  if (now - lastMsg > 1000 * timeBetweenSend) {
    lastMsg = now;
    // Read humidity
    float h = dht.readHumidity();
    // Read temperature in Celcius
    float t = dht.readTemperature();
    both = String(t) + " " + String(h);
    String one = "{\"sensor\"";
    String two = "\"temp\"";
    String three = "\"humi\"";
    String two_points = ":";
    String coma = ", ";
    String send_both = one + two_points + String(sensor) + coma + two + two_points + String(t) + coma + three +
     two_points + String(h) + "}";


    // Oh, nothing to send
    if ( isnan(t) || isnan(h)) {
      Serial.println("KO, Please check DHT sensor !");
      // return to setup()
      return;
    }

    if ( debug ) {
      Serial.print("Temperature : ");
      Serial.print(t);
      Serial.print(" | Humidity : ");
      Serial.println(h);
      Serial.println("\t" + both);
    }
    client.publish(temperature_topic, String(t).c_str(), true);   // Publish temperature on temperature_topic
    client.publish(humidity_topic, String(h).c_str(), true);      // and humidity
    client.publish(both_topic, send_both.c_str(), true);               // temperature and humidity
  }
  if (now - lastRecu > 100 ) {
    lastRecu = now;
    client.subscribe(relay_topic);
  }
}

```

```
150 // MQTT callback function
151 void callback(char* topic, byte* payload, unsigned int length) {
152
153   int i = 0;
154   if ( debug ) {
155     Serial.println("Message recu =>  topic: " + String(topic));
156     Serial.print(" | longueur: " + String(length,DEC));
157   }
158   // create character buffer with ending null terminator (string)
159   for(i=0; i<length; i++) {
160     message_buff[i] = payload[i];
161   }
162   message_buff[i] = '\0';
163
164   String msgString = String(message_buff);
165   if ( debug ) {
166     Serial.println("Payload: " + msgString);
167   }
168
169   if ( msgString == "ON" ) {
170     digitalWrite(D0,HIGH);
171   } else {
172     digitalWrite(D0,LOW);
173   }
174 }
```

Listing 9: ESP8266_mqtt_dht_relay_json

Esto programa está basado en otro con licencia MIT que permite ser integrado en otro con licencia GNU GPLv3 que es la licencia de este proyecto.

A continuación se va a explicar lo más relevante del código.

El la línea 18-21 están los *topic* a los que se suscribe o en los que publica. En las conexiones con *Mosquitto* es necesario un nombre de cliente, en la línea 32 hay una variable *clientName*, este nombre debe ser único y si más de un dispositivo se conectasen con el mismo nombre no funcionaria correctamente ninguno ya que se conectarían y desconectarían constantemente.

En la configuración, *setup()*, se inicializa la conexión al Wi-Fi, se configura la conexión con *Mosquitto* con la IP y el puerto donde está escuchando el *Broker*, a continuación se define el nombre de la función que va a hacer de *callback* para el cliente MQTT, esta función saltará en cualquier momento en que llegue un mensaje, se encargará del accionamiento de relé y por último la inicialización del sensor DHT-22.

A continuación está la función que se encarga de manejar la conexión Wi-Fi. Primeramente hay un *delay* de 10 milisegundos para que el sistema estabilice la señal D2 referente al relé y a continuación utilizando la biblioteca *ESP8266WiFi* se conecta a la red Wi-Fi mediate el *SSID* y la contraseña, mientras se conecta escribe a través del monitor para saber que no se ha conseguido conectar.

La función *reconect()* es la encargada de conectar con el *Broker de Mosquitto* y está en un bucle infinito hasta que consiga conectar, si no escribe mensajes de error en el monitor de IDE.

Seguidamente esta la función principal del microcontrolador que se ejecuta en un bucle infinito y lo primero que hace es conectar con el *Broker de Mosquitto*, ya tiene conexión Wi-Fi, se lanza el bucle del cliente MQTT que escucha si le llegan mensajes para manejarlos con el *callback*. A continuación se encuentra la parte que hace reloj para enviar los datos según se configure, si cumple la condición de tiempo entra y recoje los datos del sensor utilizando la biblioteca DHT y creando un *String* con el formato de *JSON*, si no llegan datos del sensor escribe un mensaje en el monitor, y hace un *return* que vuelve al *setup()* para inicializar al sensor. Si llegan correctamente los datos del sensor se crean los *Strings* y el JSON para que el cliente, MQTT, publica en el *Broker* en varios *topic* los datos. Con esto termina el *if* al que entra según la configuración que se haga de tiempo entre envíos. Seguidamente cada 100 milisegundos comprueba si le ha llegado algún mensaje al *topic* al cual está suscrito, en este casa al relé y con esto termina el bucle principal.

¿Qué sucede si hay un mensaje a la cola a la cual está suscrita? *client.loop()* está escuchando continuamente y al recibir un mensaje lanza el manejador, *callback* al que se ha unido en la configuración, *setup()*, esto se debe que el cliente está en un bucle continuamente escuchando, *client.loop()*.

El *callback* lo que hace es decodificar el mensaje que le llega y si el mensaje es *ON* entonces activa el pin correspondiente, con cualquier otro mensaje apaga.

11. Referencias

[1] Designing the Internet of Things (2014)
Adrian McEwen & hakim Cassimally

[2] Scrum Manager Versión 2.6
Obra colectiva creada y coordinada por Iubaris Info 4 Media SL
Autores de esta versión: Alexander Menzinsky, Gertrudis López, Juan Palacio

[3] Internet of Things: A hands-On Approach (2014)
Arshdeep Bahga & Vijay Madisetti

[4] Flask Web Development: Developing Web Applications with Python (2014)
Miguel Grinberg

[5] Ingeniería del Software (7º Edición - 2005)
Ian Sommerville

[6] UML y Patrones (2º Edición - 2003)
Craig Larman

[7] Apuntes de la Asignatura Diseño, Integración y Adaptación de Software (2016-2017)
José Manuel Marqués

[8] Charla - La Informática del futuro - NAUKAS (2017)
Diego Llanos

[9] Auto-Id Labs (MIT)
http://autoid.mit.edu/iot_research_initiative
Visitado: 31 de Julio 2018

[10] Descripción general de Internet de los objetos
https://www.itu.int/rec/T-REC-Y.2060-201206-I
Visitado: 31 de Julio 2018

[11] Internet of Things
Technology and Value Added
Felix Wortmann, Kristina Flütcher
https://www.alexandria.unisg.ch/252999/1/s12599-015-0383-3.pdf
Visitado: 31 de Julio 2018

[12] Long Rage Wi-Fi
https://en.wikipedia.org/wiki/Long-range_Wi-Fi
Visitado: 31 de Julio 2018

[13] A LoRaWAN: Long range wide area networks study
https://ieeexplore.ieee.org/document/8123360/
Visitado: 31 de Julio 2018

[14] Arduino
https://es.wikipedia.org/wiki/Arduino
Visitado: 7 de agosto 2018

[15] RabbitMQ
https://www.cloudamqp.com/blog/2015-05-18-part1-rabbitmq-for-beginners-what-is-rabbitmq.html
Visitado: 10 de Agosto 2018

[16] RabbitMQ Tutorial
https://www.rabbitmq.com/getstarted.html
Visitado: 10 de Agosto 2018

[17] AMQP Licence
https://www.amqp.org/legal
Visitado: 10 de Agosto 2018

[18] AMQP
https://www.digitalocean.com/community/tutorials/an-advanced-message-queuing-protocol-amqp-walkthrough
Visitado: 10 de Agosto 2018

[19] python-packaging
https://python-packaging.readthedocs.io/en/latest/
Visitado: 11 de Agosto 2018

[20] ESP8266 wiring
https://www.elec-cafe.com/esp8266-temperature-humidity-webserver-with-a-dht11-sensor/
Visitado: 16 de Agosto 2018

[21] ESP8266 ESP01
https://components101.com/wireless/esp8266-pinout-configuration-features-datasheet
Visitado: 18 de Agosto de 2018

[22] Issue error ESP01 en GitHub
https://github.com/esp8266/Arduino/issues/770
Visitado: 18 de Agosto de 2018

[23] ESP8266
https://es.wikipedia.org/wiki/ESP8266
Visitado: 26 de Agosto de 2018

[24] ESP8266 NodeMcu LoLin
https://www.luisllamas.es/esp8266-nodemcu/
Visitado: 26 de Agosto de 2018

[25] Ficha técnica DHT-22
https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf
Visitado: 26 de Agosto de 2018

[26] Sensor capacitivo
https://es.wikipedia.org/wiki/Sensor_capacitivo
Visitado: 26 de Agosto de 2018

[27] ficha técnica módulo Bluetooth HC-05
https://www.gme.cz/data/attachments/dsh.772-148.1.pdf
Visitado: 26 de Agosto de 2018

[28] ficha técnica caudalímetro YF S402
https://5.imimg.com/data5/SJ/LH/MY-1833510/yf-s402-g1-4-pvc-water-flow-sensor.pdf
Visitado: 26 de Agosto de 2018

[29] ficha técnica relé SRD-05VDC-SL-C
http://www.circuitbasics.com/wp-content/uploads/2015/11/SRD-05VDC-SL-C-Datasheet.pdf
Visitado: 26 de Agosto de 2018

[30] Patrón de arquitectura Publicación Suscripción
https://aws.amazon.com/es/pub-sub-messaging/
Visitado: 26 de Agosto de 2018

[31] Descripción de MQTT
https://www.ibm.com/developerworks/ssa/library/iot-mqtt-why-good-for-iot/index.html
Publicado por Michael Yuan en 04-10-2017
Visitado: 26 de Agosto de 2018