UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Desarrollo de un sistema de monitorización y automatización de cultivos

TRABAJO DE FIN DE GRADO

Borja Iñesta Hernández

Tutor: **Juan Carlos Guerri Cebollada**
Cotutor: **Francisco José Martínez Zaldívar**

Trabajo de Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2018-2019

Resumen

El siguiente proyecto consiste en el diseño e implementación de un sistema para el cuidado de un cultivo. Se llevará a cabo mediante la toma de datos y la ejecución de tareas. Una pequeña computadora Raspberry Pi actuará como nodo central de control y recopilación de datos, y será un microcontrolador Arduino el que recoja los datos de los sensores. Mediante mensajes automáticos o a demanda del usuario, la plataforma Raspberry Pi se comunicará con Arduino inalámbricamente, pudiendo solicitar datos o activar mecanismos para la automatización del huerto. Se utilizará también una cámara como dispositivo de seguridad para el sistema. De esta manera se obtiene también un banco de datos que, utilizando técnicas de análisis, proporcionará valiosa información al usuario.

Palabras clave: IoT, Sensores, Telemedida, Telegestión, Agricultura inteligente.

Resum

Aquest projecte consisteix en el disseny i la implementació d'un sistema per a l'atenció d'un cultiu. Es farà a través de la presa de dades i l'execució de tasques. Una computadora Raspberry Pi actuarà com a node central de control i recopilaciò de dades, mentre que un microcontrolador Arduino serà el que arreplegue les dades dels sensors. Utilitzant missatges automàtics o a demanda de l'usuari, la plataforma Raspberry Pi es comunicarà amb Arduino sense filment, podent sol·licitar dades o activar mecanismes per a la automatització de l'hort. També es farà ús d'una càmera com a dispositiu de seguretat per al sistema. D'aquesta manera s'obté un banc de dades que, utilitzant tècniques d'analisi, proporcionarà valuosa informació.

Paraules clau: IoT, Sensors, Telemesura, Telegestió, Agricultura intel·ligent.

Abstract

The following project involves the design and implementation of a crop-caring system. The main tools will be data collection and task execution. A Raspberry Pi computer will play as the central control node collecting data, and an Arduino microcontroller will get the data provided by the sensors. Using automatic or on demand messages, the Raspberry Pi will communicate wirelessly with Arduino being able to ask for data and activate the convenient devices in the automated garden. Besides, a camera will be used as a security system. Thus the user obtains a data bank which, through analysis techniques, will provide valuable information.

Keywords: IoT, Sensors , Telemeasurement, Telemanagement, Smart gardening.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1: Introducción

1.1 Introducción

Desde la aparición del término IoT (Internet of things), acuñado por Kevin Ashton en una conferencia en 1999 [1], el crecimiento de este campo ha sido notable. Aunque no fue hasta entre 2008 y 2009 cuando se pudo ver el verdadero ritmo de crecimiento. Según un estudio de Cisco IBSG [2], en estas fechas el número de dispositivos conectados a Internet superó al número de personas. Y aunque la predicción para 2015 de este mismo estudio era de 25 mil millones de dispositivos, realmente fue de unos 17.6 mil millones de dispositivos en 2016 [3]. Tras la publicación de estos resultados, los estudios han estimado una cifra más baja para sus predicciones, aunque de muy elevado crecimiento.

Estas cifras de crecimiento, que empresas como Gartner estiman en 20.4 mil millones de dispositivos IoT en 2020 [4], junto con la necesidad de ser más productivas, son las que llevan cada vez a más empresas a apostar por este campo.

La denominada Industria 4.0 o lo que se entiende como la siguiente revolución industrial, ve en IoT un pilar clave para su desarrollo. Aportando interoperatividad entre distintas redes y sistemas, recolección de datos mediante sensores y la toma de decisiones descentralizada gracias a la automatización de procesos, es como desde grandes empresas hasta particulares, pueden beneficiarse de la optimización aportada por las tecnologías basadas en IoT.

Como cualquier otro sector, la agricultura tiene que evolucionar con el paso del tiempo y adaptarse a los cambios y desafíos producidos por este. En el año 2017 España ocupaba la primera posición en el ranking de área cultivada por cultivos orgánicos de la Unión Europea con 2.08 millones de hectáreas [5]. Pese a que la agricultura solamente representa el 2.7% del PIB en España y en torno al 3.5% de media del PIB del mundo [6], la agricultura ecológica experimenta crecimiento en los últimos años, un 18.5% más de 2012 a 2017, como se puede observar en la figura 1.1. La contribución de los procesos no ecológicos en la agricultura al deterioro del medio ambiente, como al deterioro de la salud del consumidor son las principales razones por las que el usuario final opta por alimentos de origen ecológico. La preocupación medioambiental, la producción de alimentos de alta calidad y la generación de empleos dignos son algunos de los objetivos de la agricultura ecológica [7].

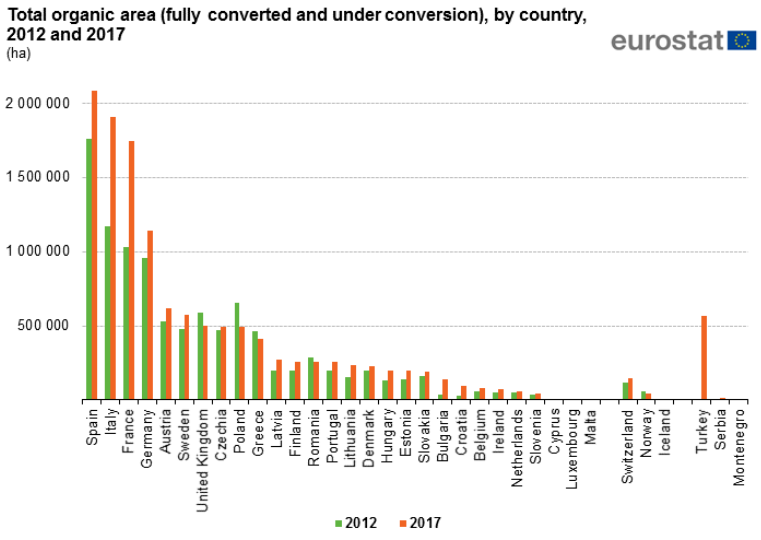


Figura 1.1 Área de cultivo orgánico en hectáreas. Fuente: eurostat.

1.2 Motivación y objetivos

El cultivo ecológico requiere de mayor planificación y cuidados que el cultivo convencional, en el que se emplean productos químicos sintéticos como fertilizantes y productos para el control de plagas. Es por esto por lo que el cultivo ecológico sería el mayor beneficiado de la implantación de sistemas IoT para la toma de datos mediante sensores, que midan parámetros desde la temperatura y la humedad del suelo hasta parámetros como el pH del agua utilizada o el índice NDVI (Normalized Difference Vegetation Index) que son de vital importancia para el crecimiento de los cultivos. Precisamente, esta corresponde con la principal motivación que origina el presente Trabajo de Fin de Grado, el cuidado y la automatización de los cultivos.

En este sentido existen proyectos realizados por personas de la comunidad DIY (Do it yourself) como Garduino [8], que sirven de inspiración para este Trabajo de Fin de Grado. Con la motivación también de que más personas se inicien en la agricultura ecológica de una manera fácil y de mejorar los procesos existentes en el cultivo de la agricultura ecológica actual este proyecto tiene el objetivo principal de diseñar e implementar una infraestructura IoT para la monitorización y la automatización del cultivo ecológico. Para cumplir este objetivo principal, se han establecido los siguientes objetivos secundarios que se detallan a continuación:

- Diseño e implementación de un dispositivo de medición y control de servicios inalámbrico.
- Diseño e implementación de un dispositivo de control, videovigilancia y recopilación de datos inalámbrico.
- Diseño e implementación de un servidor central con gestión de usuarios y acceso a los datos recopilados.
- Diseño e implementación de una interfaz de acceso a todo lo anterior.

Se espera que utilizando hardware económicamente asequible, creando software que automatice tareas y recopile datos y creando una interfaz intuitiva y amigable, la agricultura ecológica siga creciendo en España y el resto del mundo. Permitiendo así que empresas dedicadas a la agricultura hagan una mejor gestión de los recursos naturales y que cada vez más personas se interesen por cultivar ecológicamente, mejorando el medio ambiente.

1.3 Metodología

1.3.1 Fases del proyecto

1. Definición de objetivos: En esta fase se define el proyecto, creando hitos que posteriormente se dividirán en tareas tras la fase de investigación y planificación.

2. Fase de investigación y planificación: Tras la definición de los objetivos, cada hito se divide en tareas en la fase de investigación y se asigna a cada tarea un coste temporal. La suma total del tiempo asignado a cada tarea, nos dará la primera estimación del tiempo total del proyecto.

3. Seguimiento: El seguimiento del proyecto se iniciará tras la definición de objetivos, mediante tutorías bajo demanda para la resolución de dudas y la muestra de cada uno de los hitos conseguidos.

4. Análisis: La última fase del proyecto consiste en la valoración final del proyecto y en valorar las posibles líneas futuras del proyecto.

1.3.2 Diagrama de Gantt

El diagrama de Gantt del diseño y desarrollo del proyecto junto con el seguimiento mediante tutorías y las mejoras aplicadas es el siguiente:

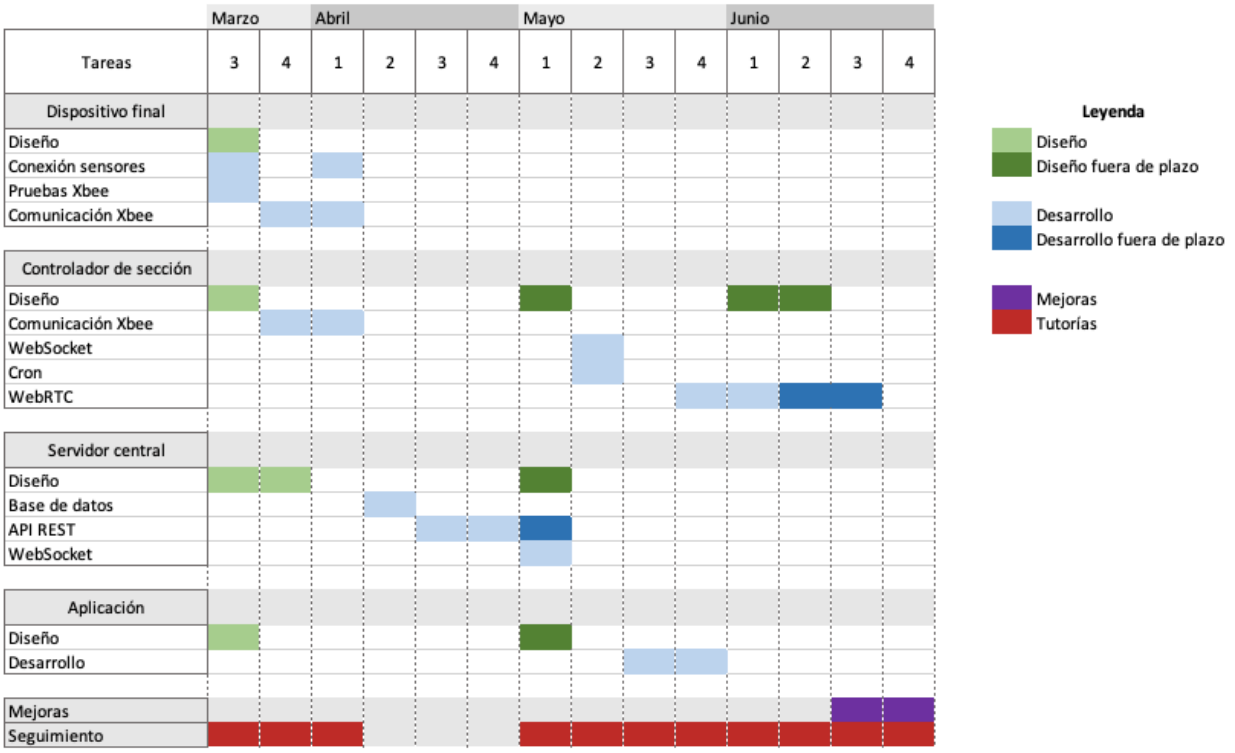


Figura 1.2 Diagrama de Gantt.

1.4 Estructura de la memoria

El siguiente apartado explica la estructura del presente documento:

- Capítulo 2 (Estado del arte): En este apartado se realiza una comparativa de las propuestas para el hardware utilizado en el proyecto y se justifican las decisiones tomadas, también se justifican las decisiones tomadas a nivel de software. Y por último se realiza una pequeña introducción de los conceptos necesarios para el correcto entendimiento de este proyecto.

- Capítulo 3 (Diseño): Muestra la arquitectura planteada para el proyecto con los diferentes elementos del sistema, las diferentes funcionalidades y los diferentes mensajes que intercambian los dispositivos. También se comentan algunas las tecnologías utilizadas para ayudarnos en el desarrollo y su importancia en el proceso de desarrollo.

- Capítulo 4 (Desarrollo): Aborda todo el proceso de creación de cada uno de los elementos que conforman el sistema, el esquema de la base de datos, así como pruebas realizadas con tecnologías descartadas e incluye diagramas de flujo de los distintos programas creados e imágenes explicativas de la función de cada dispositivo.

- Capítulo 5 (Conclusiones): Se realiza un análisis de lo conseguido por el proyecto, las posibles mejoras y los presupuestos de los prototipos realizados.

Capítulo 2: Estado del arte

2.1 Sistemas IoT

2.1.1 Infraestructura de red

Los sistemas IoT (véase figura 2.1) suelen estar caracterizados por ser una red o múltiples redes de dispositivos capaces de comunicarse entre ellos, adquirir datos mediante sensores o activar servicios mediante actuadores. También están caracterizados por necesitar de un gateway (puerta de enlace) entre los diferentes protocolos utilizados por los dispositivos finales y el elemento central de la red que permite su control, almacenamiento de los datos adquiridos y otras funciones.

Figura 2.1 Ejemplo de infraestructura de una red IoT.

Los elementos principales de este sistema son los dispositivos finales, estos dispositivos permiten tener una conexión directa con el mundo real. A través de sensores proporcionan datos en tiempo real o para su almacenamiento y a través de actuadores permiten realizar acciones interactuando con el mundo físico.

El elemento central de la arquitectura suele ser el que se encarga de dar las órdenes a estos dispositivos finales, dar acceso al servicio ofertado por el sistema, recopilar los datos y almacenarlos, extraer la información de estos datos y proporcionarles un valor añadido mediante técnicas de análisis de datos. Estas funciones pueden variar en función del objetivo del proyecto, en otros sistemas, los dispositivos finales pueden ser lo suficientemente autónomos como para tomar sus propias decisiones.

Un elemento muy importante de los sistemas IoT es el gateway, sin este elemento los diferentes dispositivos del sistema no podrían comunicarse, puesto que suelen utilizar protocolos diferentes y por esto, el gateway actúa como traductor entre distintos protocolos.

Cada uno de los elementos de esta infraestructura se dividen en una arquitectura de protocolos y tecnologías diferente (véase figura 2.2), aunque este proyecto sólo se centrará en la arquitectura de capas del dispositivo final.

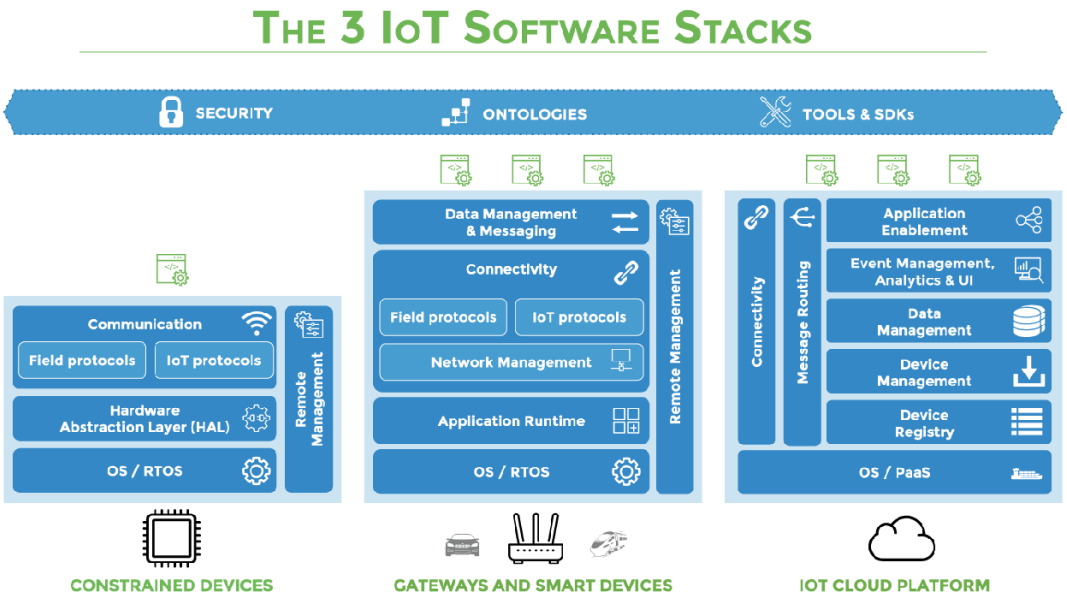


Figura 2.2 Arquitectura de cada elemento de la red IoT. Fuente: iot.eclipse.org.

2.1.2 Arquitectura de capas

Existen varias arquitecturas de capas utilizadas para IoT debido a que se trata de un concepto, más allá que una tecnología concreta. Es por esto, que cada tecnología presenta su modelo de capas. Para describir la arquitectura de capas de los dispositivos finales, se utilizará el modelo de arquitectura de cuatro capas (véase figura 2.3), utilizado por el protocolo ZigBee [9]. Este modelo está dividido en las siguientes partes:

1. Capa física: Este nivel se encarga de la modulación y demodulación de las señales, la potencia de transmisión, sensibilidad de recepción, el número de canales y el rechazo a los canales adyacentes.

2. Capa de enlace de datos: Hace uso de CSMA/CA para transmitir paquetes punto a punto entre dispositivos cercanos. También se encarga de enviar los paquetes de sincronización.

3. Capa de red: Esta capa proporciona un sistema de enrutamiento para poder realizar transmisiones de múltiples saltos entre dispositivos.

4. Capa de aplicación: Esta compuesta por varias subcapas. El framework de aplicación que define los objetos necesarios para comunicarse con la subcapa de soporte de aplicación (APS), la APS proporciona una estructura unificada de comunicación y por último el objeto del dispositivo ZigBee (ZDO) que realiza las funciones de administración de la red.

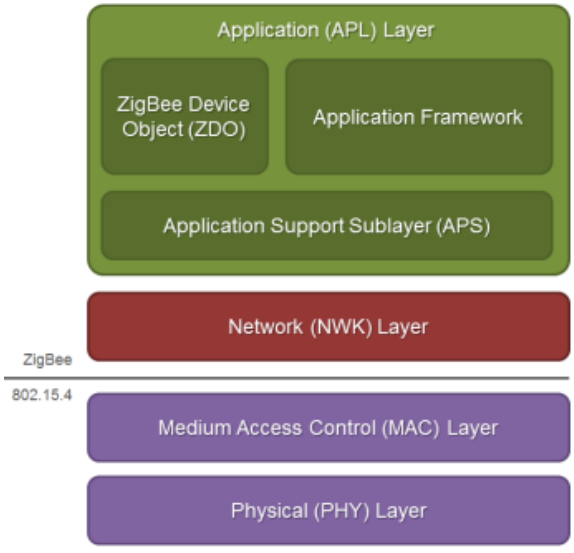


Figura 2.3 Arquitectura de capas de ZigBee. Fuente: Digi.

2.2 Hardware

En este apartado se muestran las opciones más comunes y actualmente existentes de hardware para el prototipo de los componentes de nuestra infraestructura. De estos elementos se elegirán dos y esa decisión será justificada en profundidad en el Capítulo 3: Diseño.

Para esta infraestructura se necesita de un dispositivo de bajo coste y poco consumo, capaz de manejar varios sensores y transmitir datos inalámbricamente (véase tabla 2.1) y también un dispositivo de bajo coste con mayor capacidad de computación para soportar flujos de vídeo y de consumo reducido (véase tabla 2.2). Las opciones son las siguientes:

Nombre	Arduino	STM32	Teensy	Intel Galileo
Procesador	Atmega328P	ARM Cortex-M3	ARM Cortex-M4	Intel Pentium de un núcleo
Reloj	16 MHz	72 MHz	180 MHz	400 MHz
Memoria Flash	32 KB	64 KB	1 MB	8 MB
Memoria SRAM	2 KB	20 KB	256 KB	512 KB
Memoria EEPROM	1 KB	Emulada	4 KB	8 KB
Pines analógicos	6	10	25	6
Pines digitales	14	26	42	20
Precio	20 €	< 8 €	29 €	79 €

Tabla 2.1 Comparativa microcontroladores.

Nombre	Raspberry pi	Tinker Board	Orange pi	BeagleBone
Procesador	ARM Cortex-A53	ARM Cortex-A17	ARM Cortex-A53	ARM Cortex-A8
Núcleos	4	4	4	2
Reloj	1.2 GHz	1.8 GHz	1.2 GHz	1 GHz
Memoria RAM	1 GB	2 GB	2 GB	512 MB
Pines	40	40	40	92
WIFI	802.11n	802.11n	802.11n	802.11n
Ethernet	10/100	1000	100/1000	10/100
USB	4	2	3	1
Precio	35 €	70 €	32 €	62 €

Tabla 2.2 Comparativa ordenadores de placa reducida.

Tras la comparativa realizada en las dos tablas, el microcontrolador seleccionado ha sido Arduino UNO (véase figura 2.4), siendo el microcontrolador con especificaciones más modestas en todos los campos analizados y el segundo microcontrolador más barato tras el STM32.

Pese a que en comparación de los campos analizados la elección lógica sería la de STM32, la gran comunidad existente que proporciona documentación y apoya el proyecto Arduino, la sencillez de su IDE y del proceso de instalación de librerías, su filosofía de software y hardware libre y su bajo consumo hacen que este proyecto se decante por esta opción finalmente. Su gran popularidad hace que encontremos fácilmente gran cantidad de shields y software compatible con este microcontrolador.

Figura 2.4 Microcontrolador Arduino UNO.

Por otra parte, el ordenador de placa reducida seleccionado ha sido Raspberry pi 3 model B (véase figura 2.5). Sólo por detrás de la opción de ASUS Tinker Board en muchas de las características y de Orange pi en memoria RAM y precio. Ocurre el mismo fenómeno que en la tabla de microcontroladores, en esta tabla, la opción lógica por potencia y precio sería Orange pi, pero la elección ha sido de Raspberry pi por su comunidad, los HAT disponibles y otro hardware disponible en su tienda.

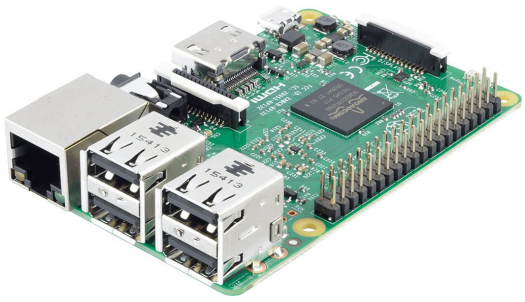


Figura 2.5 Ordenador de placa reducida Raspberry pi 3 model B.

Una vez elegido el hardware para realizar el prototipo de los dispositivos finales es momento de elegir el protocolo de comunicaciones inalámbricas que se utilizará, ya que de esta decisión dependerá elegir un hardware u otro. Como todas las decisiones mostradas en este capítulo, será de nuevo justificada por motivos de infraestructura de la red que queremos diseñar en el apartado Capítulo 3: Diseño.

Para mostrar esta decisión se mostrará una tabla comparando los protocolos de comunicación típicos tanto para WLAN, WMAN, WWAN y LPWAN, omitiendo tecnologías WPAN que no son útiles para nuestro diseño (véase tabla 2.3).

Protocolo	WIFI	ZigBee	SigFox	LoRa	NB-IoT
Cobertura	Rural: 100 m Urbana: 50 m	Rural: 1.2 km Urbana: 60 m	Rural: 50 km Urbana: 5 km	Rural: 15 km Urbana: 5 km	Rural: 10 km Urbana: 1 km
Tasa de transmisión	600 Mbit/s	250 kbit/s	100 bit/s	50 kbit/s	200 kbit/s
Banda	2.4 o 5 GHz	2.4 GHz	868 MHz	868 MHz	800, 1800 o 2600 GHz
Número de nodos	250	65536	10^6	10^4	10^4
Precio por shield/HAT	25 €	20 €	45 €	20 €	45 €

Tabla 2.3 Comparativa protocolos de comunicación inalámbrica.

La elección del protocolo contrastando los datos de la anterior comparativa ha sido de ZigBee. Siendo la segunda opción con mayor tasa de transmisión, primera opción en cuanto a precio junto con LoRa y teniendo una cobertura de 1.2 km en zona rural y 60 m en zona urbana, son las razones por las que se ha escogido este protocolo. Opciones como SigFox o NB-IoT son descartadas por el uso de una red no gestionada, ambos protocolos utilizan sus propias redes y funcionan mediante pagos mensuales por su uso, en el caso de SigFox tras superar la versión de prueba o utilizarlo de manera comercial. Junto con ZigBee, una opción muy interesante es LoRa que ha sido descartada simplemente por su menor tasa de transmisión de datos. Y por último, cabe señalar que el coste de WIFI es por el uso de un shield necesario para Arduino UNO y que esta opción ha sido descartada por su cobertura y número de nodos.

El hardware utilizado para el uso de ZigBee será XBee (véase figura 2.6), tecnología de la compañía Digi, con la posibilidad de utilizar protocolos para la creación de redes como ZigBee o el protocolo propietario DigiMesh. Los shields disponibles para Arduino y los HAT disponibles para Raspberry pi, junto con las bibliotecas existentes para el software de Arduino y Python, son las razones de peso en esta elección.

Figura 2.6 Módulo radio XBee PRO S2C.

Con el fin de presentar un prototipo mínimamente funcional, se han escogido una serie de sensores para proporcionar medidas interesantes para el proyecto. Los sensores han sido escogidos por su utilidad en relación con el proyecto, pero teniendo siempre en cuenta la perspectiva económica ya que se trata solamente de un prototipo. Sensores que midan parámetros diferentes o de mejor calidad que los actualmente seleccionados serán fundamentales para el desarrollo final del proyecto.

Los dispositivos finales serán capaces de realizar las siguientes funciones:

- Tomar medidas de la temperatura y humedad del aire
- Tomar medidas de la humedad de la tierra
- Tomar medidas de la luminosidad del ambiente
- Tomar medidas del nivel de agua del depósito
- Activar o desactivar una bomba de agua
- Activar o desactivar la luz artificial

Para ello se ha escogido el siguiente hardware:

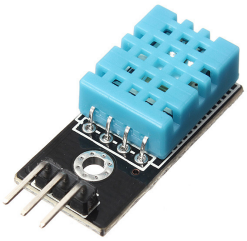


Figura 2.7 Sensor DHT11.

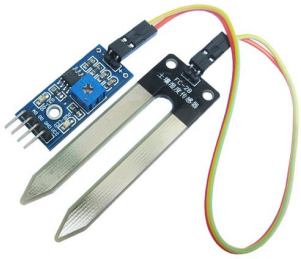


Figura 2.8 Sensor FC28.

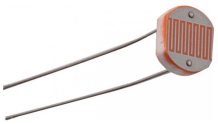


Figura 2.9 Resistencia LDR.

Figura 2.10 Sensor HW38.

Figura 2.11 Relé.

El sensor DHT11 (véase figura 2.7) permite tomar medidas de temperatura y humedad del aire, el sensor FC28 (véase figura 2.8) introducido en el sustrato proporcionará medidas de humedad del sustrato. Realizando un pequeño circuito y tratando la medida obtenida por la resistencia LDR (véase figura 2.9) se podrá realizar un pequeño luxómetro. Con el sensor HW38 (véase figura 2.10) se obtendrá el nivel de agua en el tanque en todo momento. Y gracias a los relés (véase figura 2.11) se controlarán los servicios mediante Arduino.

También se instalará una pequeña cámara en cada una de las Raspberry pi de la infraestructura, esta decisión cobrará sentido en el Capítulo 3: Diseño, donde se mostrará la infraestructura real del proyecto y la función de cada elemento. La cámara seleccionada es la Raspberry pi Camera V2, principalmente por su económico precio y por su slot dedicado en la placa de la Raspberry pi, que evita el uso innecesario de un USB.

2.3 Software

Se han de tomar decisiones sobre qué software se utilizará para implementar las interconexiones del sistema. Debido al diseño del sistema algunas decisiones tendrán un amplio margen de maniobra mientras que otras vendrán prácticamente impuestas por la tecnología utilizada.

Dispositivo final: Al tratarse de un dispositivo diseñado sobre el microcontrolador Arduino UNO, el lenguaje de programación será Arduino, basado en C/C++. Con el uso de las librerías necesarias para el control de los sensores y la utilización del dispositivo XBee. Este dispositivo será programado con la ayuda del Arduino IDE, su entorno de desarrollo oficial.

Controlador de sección: Este dispositivo tendrá instalada una distribución de Raspbian como su sistema operativo. Raspbian se trata de un sistema operativo GNU/Linux basado en Debian y es el sistema operativo oficial para Raspberry pi, aunque existen otras muchas opciones. Se usará Python 3 para implementar el programa principal que permitirá al controlador de sección comunicarse con Arduino mediante XBee y comunicarse a su vez con el nodo central o la plataforma en la nube que almacenará los datos, haciendo de gateway. Se utilizará el proyecto WebRTC para la captura y streaming de vídeo de la cámara de la Raspberry pi. El IDE a utilizar será cualquiera de los preinstalados en la distribución Raspbian.

Nodo central: Para el desarrollo de las funciones de este elemento son pocas las limitaciones que se tienen. El desarrollo se realizará en un ordenador portátil MacBook Air de buenas prestaciones y su futuro despliegue será en un servidor local o en la nube, por lo que, por el momento no hay limitaciones a la hora de elegir el lenguaje de programación o framework a usar. Se utilizará Python 3 al igual que en el controlador de sección para facilitar la comunicación entre el nodo central y el gateway, al utilizar el mismo lenguaje de programación y posiblemente los mismos frameworks, implementar la comunicación será mucho más simple. Para la base de datos, se utilizará el motor sqlite, viene preinstalado en Python 3 y con él se tendrá una base de datos relacional embebida. Algunas de las ventajas de este modelo son su sencillez, gracias a ello requiere de poco mantenimiento y que está integrado en la aplicación software en la que será utilizado.

Servicios: En la infraestructura los servicios serán proporcionados vía web. Para facilitar la implementación de esta web se usará el framework de desarrollo de aplicaciones Angular [10]. Este framework, desarrollado por Google, combina los elementos de programación web tradicional como HTML, CSS y JavaScript con elementos más avanzados y novedosos como Sass para mejorar CSS, TypeScript añadiendo tipos algunas variaciones a JavaScript y otros elementos propios. Se trata del principal framework para aplicaciones web a Junio de 2019.

XCTU: Todos los XBee utilizados en este proyecto tendrán que ser previamente configurados con los parámetros que se justificarán en el Capítulo 3: Diseño. Para facilitar la configuración de estos dispositivos, la empresa Digi proporciona una aplicación multiplataforma llamada XCTU [11] que permitirá modificar parámetros y realizar tests de conexión de los distintos elementos de nuestra red XBee.

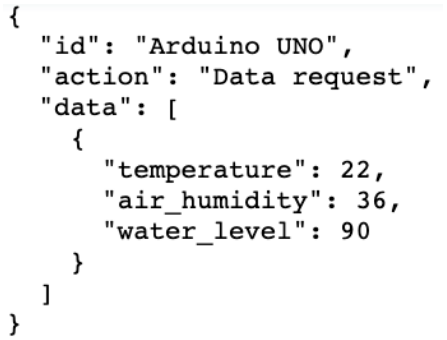
En el proyecto también se utiliza otro software como gestor de paquetes, herramientas para migración de bases de datos, creación de entornos virtuales, software de control de revisiones, múltiples librerías de varios lenguajes de programación o herramientas de creación de variables de entorno que serán comentadas durante el desarrollo de la memoria del proyecto.

2.4 Fundamentos

A continuación se proporcionará una pequeña descripción de algunos de los conceptos necesarios para la correcta comprensión de este proyecto.

JSON:

Se trata de un formato de texto para el envío de datos por redes. Derivó del lenguaje de programación JavaScript donde los objetos son escritos en este formato, de ahí su nombre JavaScript Object Notation. Su formato fácilmente legible por humanos (véase figura 2.12), su reducida sintaxis de seis tipos de datos que hace fácil su aprendizaje y su formato más compacto en comparación con XML, hacen que JSON sea cada vez una alternativa más frecuente. Su elección para este proyecto además de por las razones anteriores ha sido por la utilización de TypeScript (superconjunto de JavaScript) y Python 3 como lenguajes de programación.



```
{
  "id": "Arduino UNO",
  "action": "Data request",
  "data": [
    {
      "temperature": 22,
      "air_humidity": 36,
      "water_level": 90
    }
  ]
}
```


Figura 2.12 Ejemplo JSON.

API REST:

Por una parte hace referencia al componente API (Application Programming Interface) que se refiere a una interfaz de programación de aplicaciones y por otra a REST (REpresentational State Transfer) que se refiere a un tipo de arquitectura de software que cumple las siguientes reglas de diseño [12]:

- Cliente/Servidor: El modelo principal será el de un servidor que ofrece un servicio y un cliente que hace uso de este servicio.
- Sin estado: Cada petición contendrá la información necesaria para que el servidor pueda proporcionar la respuesta. No se almacenará información entre peticiones.
- Cacheable: El servidor indicará al cliente si las peticiones pueden ser almacenadas para ser utilizadas en el futuro.
- Sistema dividido en capas: La comunicación entre el cliente y el servidor tiene que estar estandarizada, de tal forma que permita a un intermediario responder a las solicitudes del cliente en vez del servidor.
- Interfaz uniforme.
- **(Opcional)** El servidor podrá suministrar código ejecutable a los clientes.

Debido a la segunda regla, no se pueden utilizar elementos de almacenamiento de sesión u otra información como las cookies. En caso de querer securizar algunos de los servicios proporcionados por la API REST se deberá hacer uso de la siguiente técnica de seguridad, la autenticación basada en tokens.

Autenticación basada en tokens:

Un token de seguridad suele ser un String codificado y encriptado que facilita la autenticación de un usuario. Generalmente el usuario accede al sistema introduciendo su nombre de usuario y contraseña y el sistema le responde con un token que le autoriza a hacer uso de los servicios que proporciona. Este token tiene un tiempo limitado de uso, permitiendo así que el usuario no tenga que autenticarse cada vez que va a hacer uso de uno de los servicios del sistema. Una vez recibido el token de autenticación, el usuario ha de incluirlo en todas las peticiones que requieran autenticación y el sistema se encargará de validar el token, si es un token expirado, incorrecto o falso, el sistema tendrá la obligación de denegar el acceso al recurso solicitado.

La diferencia con las cookies está en que el sistema no tiene que almacenar ninguna información mas allá que el mecanismo de cifrado de este token, en cambio en los sistemas basados en cookies tanto el usuario como el sistema han de poseer la cookie.

CRUD:

CRUD (Create, Read, Update and Delete) hace referencia a las funciones de crear, leer, actualizar y borrar realizadas por las bases de datos de un sistema. Se utilizará este acrónimo en el Capítulo 3: Diseño, cuando se tengan que definir algunos de los procesos de la aplicación.

WebRTC:

Es un proyecto de software libre que proporciona la capacidad de realizar comunicaciones en tiempo real con el uso de APIs [13]. Este proyecto realizado por Google permite la comunicación mediante el intercambio de mensajes peer to peer o el intercambio de audio y vídeo en tiempo real, todo esto sin la necesidad de instalar plugins o aplicaciones. La infraestructura permite la conexión de dos máquinas vía peer to peer o a través de un servidor central. Para entender WebRTC necesitamos definir una serie de elementos clave en el funcionamiento de esta tecnología:

- ICE (Interactive Connectivity Establishment): Es el framework encargado de gestionar la conexión realizada por WebRTC. Las redes que atravesamos para realizar una conexión se componen de elementos muy variados que a veces dificultan la conexión. El uso de NAT para la gestión de direcciones IP públicas, firewalls, routers y otros elementos, a su vez con el uso de diferentes protocolos de transporte como TCP o UDP hacen que el camino o la forma de conexión con un mismo punto no sea única. ICE se encarga de buscar la mejor opción a la hora de conectar dos elementos entre sí, y lo hace apoyándose de otros elementos que conforman la tecnología WebRTC, como son los servidores STUN y TURN.

- STUN (Session Traversal Utilities for NAT): Es el servidor responsable de proveer a un cliente que se encuentra detrás de un router NAT su dirección IP pública. Información necesaria para el establecimiento de la conexión. Una vez proporcionada, la comunicación se realiza entre los clientes directamente.

- TURN (Traversal Using Relay NAT): Es el servidor al que WebRTC recurre cuando todo lo anterior ha fallado. A diferencia de STUN, este servidor se mantiene entre los clientes una vez la conexión se ha establecido. Si WebRTC requiere de este servidor, la comunicación ya no será peer to peer.

El establecimiento de la conexión se lleva a cabo compartiendo información en formato SDP. SDP (Session Description Protocol) es un protocolo para describir sesiones de comunicación multimedia. Mediante ICE, se generan candidatos, que son opciones de conexión y estas opciones se envían en formato SDP.

El proceso de envío de esta información se llama señalización y no está definido por WebRTC.

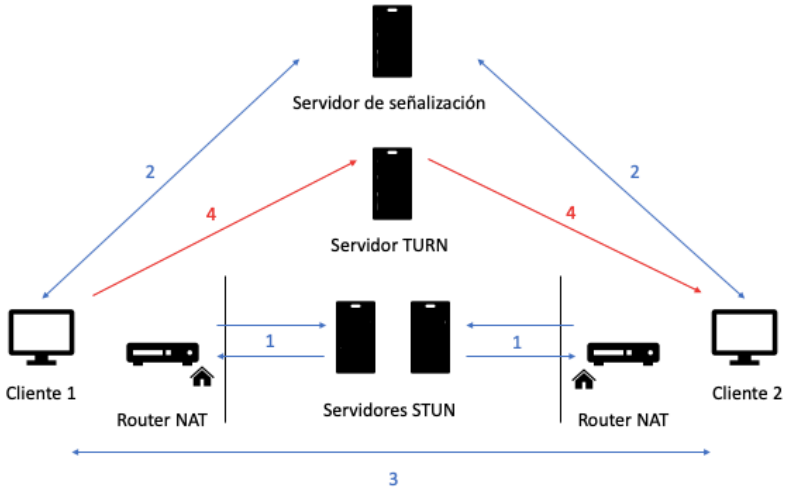


Figura 2.13 Comunicación con WebRTC.

En la figura 2.13 se muestra el proceso típico de comunicación con WebRTC. En ella se pueden ver dos clientes tras un router NAT que solicitan la información al servidor STUN (1), acto seguido inician la comunicación con el servidor de señalización para intercambiar la información con el otro cliente (2) y una vez intercambiada esta información son conectados el uno con el otro eliminando cualquier otro intermediario (3). En rojo, a diferencia de todos los pasos anteriores se encuentra el proceso de conexión cuando los mecanismos anteriores no son posibles, mediante un servidor TURN (4).

WebSocket:

WebSocket es un protocolo de comunicaciones que proporciona un canal bidireccional sobre una única conexión TCP. Está implementada para ser utilizada entre navegadores y servidores web, aunque otras tecnologías cliente/servidor también pueden utilizarlo. Es un elemento clave para la comunicación por notificaciones o push, en la que el servidor envía datos al cliente de manera asíncrona. Con esta tecnología el usuario o suscriptor puede iniciar un WebSocket y ser informado cuando ocurran eventos en el servidor.

La conexión entre el cliente y el servidor se comprueba constantemente gracias a un mecanismo de heartbeat, en el que el servidor envía mensajes constantemente y en caso de no recibir respuesta, cierra la conexión con el cliente en cuestión. El coste computacional para el servidor de mantener una conexión abierta sin hacer uso de ella es prácticamente nulo.

Red ZigBee:

Se tratan de redes compuestas por elementos que implementan la tecnología ZigBee. Dentro de estas redes existen tres tipos de dispositivos según su papel en la red:

- Coordinador: Es el dispositivo con más funcionalidad. Dado que sus funciones son las de control de la red, conexión de dispositivos y creación de rutas, es obligatoria la existencia de uno de estos elementos en cada red.

- Router: Gracias a este dispositivo se puede extender la cobertura de la red. Su misión es la de reenvío de datos y la creación de rutas alternativas.

- Dispositivo final: Es el dispositivo con funcionalidad más reducida, sólo puede comunicarse con su nodo padre (un coordinador o un router).

Los conceptos modos de transmisión y topologías serán comentados en el siguiente capítulo.

Capítulo 3: Diseño

El sistema que se diseñará en este apartado debe aportar una solución escalable y lo más eficiente posible para cumplir con los objetivos definidos por el proyecto. A continuación se mostrará como esta solución con una única implementación puede servir para cumplir con los requisitos de aficionados a la agricultura y con requisitos más exigentes como son los de profesionales de la agricultura. Para ello se definirán conceptos utilizados en el proyecto y se plantearán dos casos de uso, el primero de ellos de un aficionado a la agricultura y el segundo de una empresa de agricultura.

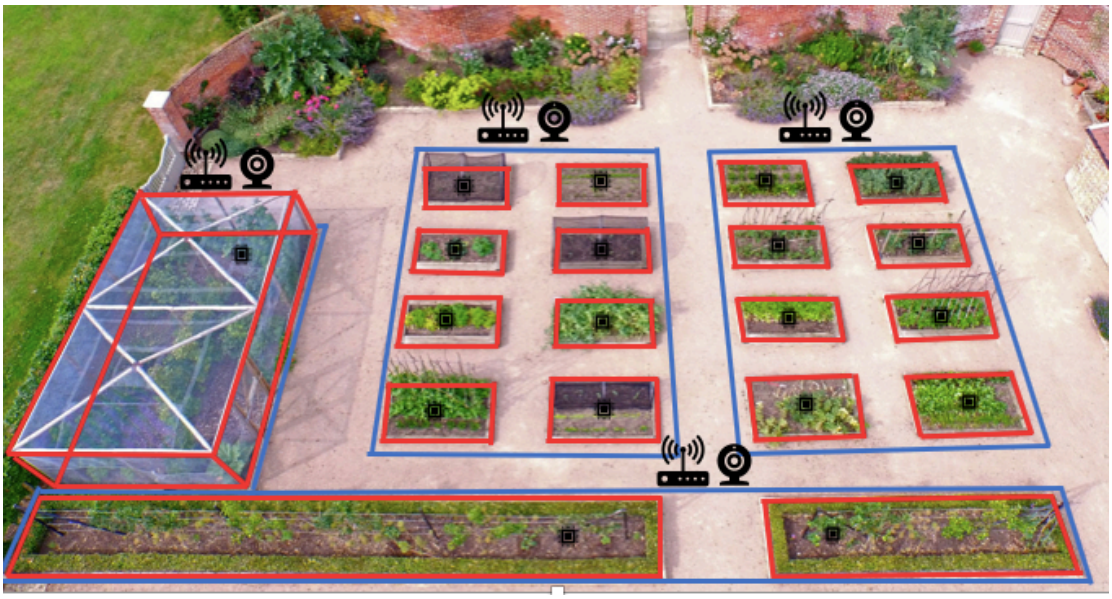


Figura 3.1 Sistema aplicado a un huerto.

La figura 3.1 muestra como sería la distribución de los elementos de este proyecto en un huerto más grande que el huerto particular típico y más pequeño que un huerto destinado a la agricultura comercial. Así se combinan ambas implementaciones de manera gráfica en la misma imagen. Cada zona roja es un cultivo diferente, debido a eso en el caso más simple tendrá un sensor o dispositivo final (Arduino UNO) por cada uno de los cultivos. En azul se puede ver lo que se define en nuestro proyecto como secciones. Cada sección cuenta con un controlador de sección (Raspberry pi) diferente, al que están conectados los sensores pertenecientes a esa misma sección. Instalado en el mismo controlador de sección se encuentra una cámara de videovigilancia, capaz de transmitir vídeo en tiempo real si el cliente lo solicita. El conjunto de todas las secciones define un huerto o zona.

Recapitulando, un huerto puede tener una o varias secciones y que cada una de estas secciones pueden tener uno o varios sensores. Estos sensores recopilarán datos que serán entregados a un servidor central para su análisis o al usuario final, gracias a la capacidad de tomar datos en tiempo real de nuestro sistema.

Por otra parte, la aplicación central que se encargará de guardar todos los datos tendrá un sistema de usuarios. Este sistema definirá qué huertos tiene cada uno de los usuarios. Mediante relaciones en las tablas de base de datos, se realizará de tal forma que varios usuarios puedan poseer el mismo huerto, para así permitir la gestión múltiple de un pequeño huerto o el acceso a recursos de los empleados de una empresa de agricultura.

Por último, una aplicación web será creada para la gestión de cada uno de estos elementos, la utilización de los servicios y la visualización de los datos.

3.1 Dispositivo final

El dispositivo final o sensor del proyecto tendrá la capacidad de recibir y responder a mensajes de cualquier dispositivo ZigBee conectado en la misma red ZigBee, además de obtener medidas de los sensores instalados y de poder activar o desactivar otros mecanismos. El hardware principal constará de un Arduino UNO, conectado a la red ZigBee mediante el dispositivo XBee. Conectados a este Arduino UNO estarán los sensores DHT11, FC28, HW38, el circuito luxómetro compuesto por la LDR y dos relés para el control de mecanismos. El dispositivo final será controlado por el controlador de sección (Raspberry pi), dispositivo que hace de coordinador de la red ZigBee y de gateway, mediante el uso de mensajes.

La comunicación por la red ZigBee se realizará en formato JSON. Se han definido dos tipos de mensajes para obtener la funcionalidad requerida. Estos mensajes serán mostrados en el apartado de comunicaciones de este capítulo.

Todos los scripts para este dispositivo estarán programados en lenguaje Arduino.

3.2 Controlador de sección

Este elemento clave para el proyecto realiza las funciones de comunicación y control de la red ZigBee, videovigilancia a través de la cámara instalada, petición y recopilación de los datos de los dispositivos finales de la red y la comunicación con el servidor central. Este dispositivo está formado por una Raspberry pi 3 model B con la cámara oficial V2 conectada y un dispositivo XBee en modo coordinador para comunicarse y gestionar con la red ZigBee. A su vez estará conectado a una red con acceso a Internet para la transmisión de los datos recopilados al servidor central y la emisión de vídeo en tiempo real.

Para la recopilación de datos, el controlador de sección ejecutará un programa, tipo Cron [14], que se encargará de regular la ejecución de los procesos de recopilación de datos en segundo plano a intervalos regulares. Estos datos serán enviados al servidor central para su almacenamiento en base de datos en formato JSON. Si la conexión con el servidor central se pierde por cualquier motivo, excepto por la interrupción de la ejecución de los procesos del controlador de sección, estos datos permanecerán almacenados en el controlador de sección hasta que se restablezca la conexión. Por otra parte, dada la posibilidad de que el usuario solicite datos en tiempo real, el servidor central se encargará de enviar la orden al controlador de sección. El usuario sólo tendrá que acceder a una web, vía navegador en un ordenador o vía smartphone, para realizar la solicitud de datos de los sensores en tiempo real o el inicio de la videovigilancia.

El programa principal para este dispositivo estará programado en Python 3.

3.3 Servidor central

El servidor central tendrá una base de datos con todos los datos de la aplicación, un sistema de usuarios que controlará el acceso a estos datos mediante la autenticación basada en tokens y una API REST que expondrá un CRUD mediante rutas HTTP para que otras aplicaciones puedan hacer uso de estos datos. La API REST para el uso de sus operaciones, aceptará y enviará mensajes en formato JSON. También permitirá la subida de datos a la base de datos mediante un archivo JSON.

Además de las acciones permitidas por las operaciones de la API REST, este servidor tendrá soporte para WebSockets, que permitirán el uso de tecnología push en la aplicación de navegador que se desarrollará como producto final, para el envío de información y otros mensajes. Este servidor con el uso de WebSockets será el encargado de señalizar la comunicación previa necesaria en WebRTC, haciendo de servidor de señalización.

El servidor será programado en Python 3 usando el microframework Flask [15]. La gran cantidad de extensiones existentes para este framework y su simplicidad, hacen de Flask una herramienta muy competitiva. El servidor a utilizar basado en WSGI será el proporcionado por la librería eventlet [16], necesaria para la utilización de WebSockets en Flask.

3.4 Comunicaciones

En esta parte se comentarán las características de diseño y se mostrarán esquemas de las comunicaciones entre los distintos elementos de la infraestructura diseñada.

3.4.1 Red ZigBee

En primer lugar, la primera red a diseñar será la red ZigBee. Esta red se encargará de comunicar los dispositivos finales con el controlador de sección en cada una de las secciones de nuestro huerto. El controlador de sección (Raspberry pi) estará conectado al XBee que en la red ZigBee ocupará el papel de coordinador, mientras que el dispositivo final tendrá el papel de router o dispositivo final en la red ZigBee, en función del tamaño de nuestra red de sensores.

Un aspecto no comentado de los dispositivos XBee son los modos de operación. Existen dos modos de operación, transparente o API. El modo transparente se utiliza para comunicaciones y redes sencillas, generalmente punto a punto. Este modo viene por defecto en los XBee, ya que no necesita programación alguna para utilizarse, de este modo al obtener los XBee, están listos para ser utilizados en su modo más básico. En cambio el modo API debe ser programado, este modo ofrece características como la posibilidad de envío de paquetes a varios destinatarios, la identificación de los dispositivos remotos conectados a la misma red, posibilidad de la lectura de parámetros o la configuración de dispositivos remotos, entre otra características [17]. Por eso, para la red se ha escogido el modo API.

Uno de los puntos más interesantes de la tecnología propietaria de Digi incluida en XBee, DigiMesh, es la posibilidad de que todos los dispositivos de la red se desconecten para ahorrar energía. En el caso de las redes ZigBee, sólo los dispositivos finales pueden desconectarse para ahorrar energía, el coordinador de la red siempre ha de estar conectado y los routers también. Las redes basadas en DigiMesh son redes más sencillas, donde sólo existe un tipo de dispositivo [18]. A pesar de estas ventajas proporcionadas por DigiMesh con respecto a ZigBee, se ha optado por ZigBee para facilitar la interoperatividad con otro tipo de dispositivos basados en esta tecnología, sin necesidad de utilizar XBee. Otro punto a favor se trata de ser un protocolo estandarizado por el IEEE.

Como ya se ha comentado anteriormente, la comunicación entre el controlador de sección y los dispositivos finales se realizará a través de mensajes en formato JSON. Se han definido dos formatos de mensajes de petición que siempre enviará el controlador de sección y dos mensajes de respuesta enviados por los dispositivos finales (véase figura 3.2).

El primer tipo de mensaje se encargará de solicitar datos de todos los sensores y mecanismos conectados a un dispositivo final y el segundo tipo de mensaje permitirá la activación de mecanismos conectados a los relés, como una bomba de agua o luz artificial. En las siguientes imágenes se muestran la estructura de la red ZigBee, con cada elemento y su función dentro del proyecto y ejemplos de los mensajes utilizados, tanto de petición como de respuesta.

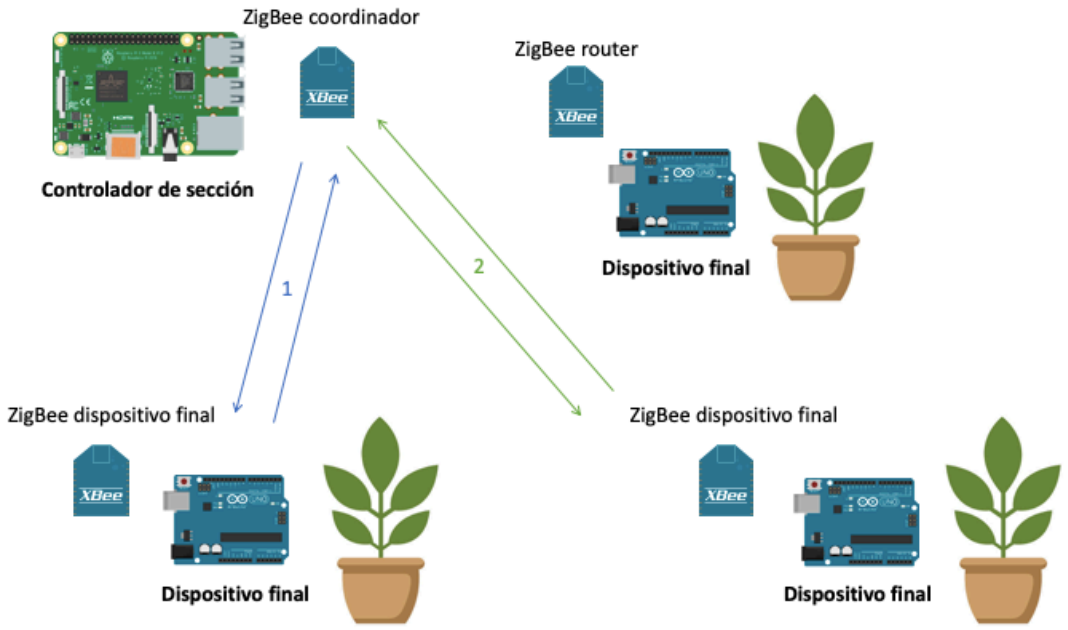


Figura 3.2 Red ZigBee del proyecto.

En la imagen anterior, se puede ver como el controlador de sección está conectado al coordinador ZigBee, mientras que los dispositivos finales de nuestro proyecto son o routers ZigBee o dispositivos finales ZigBee. La elección entre router o dispositivo final dependerá de la proximidad del dispositivo al controlador de sección. También se muestra el flujo de mensajes a distintos dispositivos, siendo el flujo de mensajes de tipo 1 (solicitud de datos) el siguiente:

```
{
  "act": 1
}
```

```
{
  "temperature": 23,
  "air_humidity": 31,
  "soil_humidity": 70,
  "light_intensity": 1290,
  "water_level": 82,
  "pump": false,
  "led": false
}
```

Figura 3.3 Petición (izquierda) y respuesta (derecha) de mensajes tipo 1.

El controlador de sección realiza la petición de datos enviando el mensaje con *act* (proviene de action) igual a 1. El sensor DHT11 proporciona su medida en los campos de la respuesta JSON llamados *temperature* y *air_humidity,* el sensor FC28 en el campo *soil_humidity,* el circuito con LDR en el campo *light_intensity,* el sensor HW38 en el campo *water_level* y por último los campos *pump* y *led* que proporcionan información sobre el estado de los mecanismos de bombeo de agua y luz artificial.

El flujo de mensajes de tipo 2 (activación de mecanismos) es el siguiente:

```
{
  "act": 2,
  "pump": true,
  "led": false
}
```

```
{
  "pump": true,
  "led": false
}
```

Figura 3.4 Petición (izquierda) y respuesta (derecha) de mensajes tipo 2.

En este tipo de mensajes, el controlador de sección envía el mensaje con *act* igual a 2 e incluyendo los campos *pump* y *led* con el valor booleano a true, si quiere activar dicho mecanismo o false, si quiere desactivarlo. El dispositivo final recibe el mensaje, activa o desactiva los mecanismos solicitados por el controlador de sección y devuelve el mensaje tras aplicar los cambios.

Cabe realizar ciertos apuntes, los *air_humidity, soil_humidity* y *water_level* son de tipo int y representan el valor en porcentaje de dicha medida. El campo *temperature* devuelve el valor en grados centígrados, el campo *light_intensity* devuelve el valor en luxes y los campos *pump* y *led,* que representan el estado de los mecanismos, son de tipo booleano, con valor true si está activado o valor false si no.

En el formato de los mensajes no se incluye la información del dispositivo que envía el mensaje debido a que esa información se puede extraer de los campos del protocolo. En el diseño del mensaje sólo se incluyen los datos.

3.4.2 Red Sección-Servidor central

Se trata de la red que conecta los controladores de sección con el servidor central. A través de esta red viajarán datos de control para las medidas en tiempo real, flujos de vídeo y señalización con WebRTC y los datos recopilados para ser almacenados en la base de datos del servidor central.

Para la señalización con WebRTC y la petición de datos en tiempo real, se ha seleccionado la tecnología WebSocket. Utilizando la librería Flask-SocketIO [19] y el servidor proporcionado por eventlet, se diseñará el servidor central que tendrá soporte para WebSockets. Por otra parte, para conectar los controladores de sección con el servidor central, se usará la librería SocketIO para Python 3. Ambas librerías proporcionan los mismos mecanismos de conexión, la diferencia reside en que Flask-SocketIO es una extensión de SocketIO para Flask.

SocketIO define una serie de mecanismos para mantener una comunicación bidireccional basada en eventos mediante clientes y un servidor. Estos mecanismos proporcionan eventos de conexión, desconexión y otros eventos básicos, además de la posibilidad de crear eventos personalizados por el usuario. Junto a esto, proporciona una forma de conexión mediante rutas HTTP y las llamadas “salas” en las que se pueden conectar varios clientes para mantener una comunicación múltiple.

Se crearán los métodos necesarios para la conexión de controladores de sección con el servidor central, y en el servidor central se mantendrá un listado de todos los controladores de sección que permanecen conectados. Una vez realizadas las conexiones, se proporcionarán eventos de petición de datos en tiempo real y activación de mecanismos en tiempo real. El servidor emitirá este evento al controlador de sección seleccionado y éste retransmitirá el mensaje al dispositivo final seleccionado, a través de la red ZigBee.

Para proporcionar la funcionalidad de servidor de señalización, cuando un controlador de sección se conecte al servidor principal, será incluido en una “sala”, cuyo nombre será un identificador único para ese controlador de sección. El usuario que quiera conectar con el controlador de sección para iniciar el proceso de señalización de WebRTC, tratará de enviar un mensaje a esta “sala”. El servidor central emitirá un mensaje informando sobre el estado de la conexión de este controlador de sección y, en caso de que esté conectado, se iniciará el proceso de intercambio de mensajes SDP a través del servidor central. Si se puede realizar la conexión, se realizará entre los dos participantes directamente (peer to peer).

Finalmente para la recopilación de los datos se facilitará una ruta HTTP en el servidor central para que, una vez recopilados todos los datos de los dispositivos conectados a la red ZigBee y en formato JSON, puedan ser subidos mediante una petición HTTP POST, adjuntando el archivo. Si el controlador de sección ha perdido la conexión con el servidor central, se encargará de mantener el archivo para tratar de subirlo en otro momento. Una vez subido, el archivo es parseado en el servidor central y añadido a la base de datos.

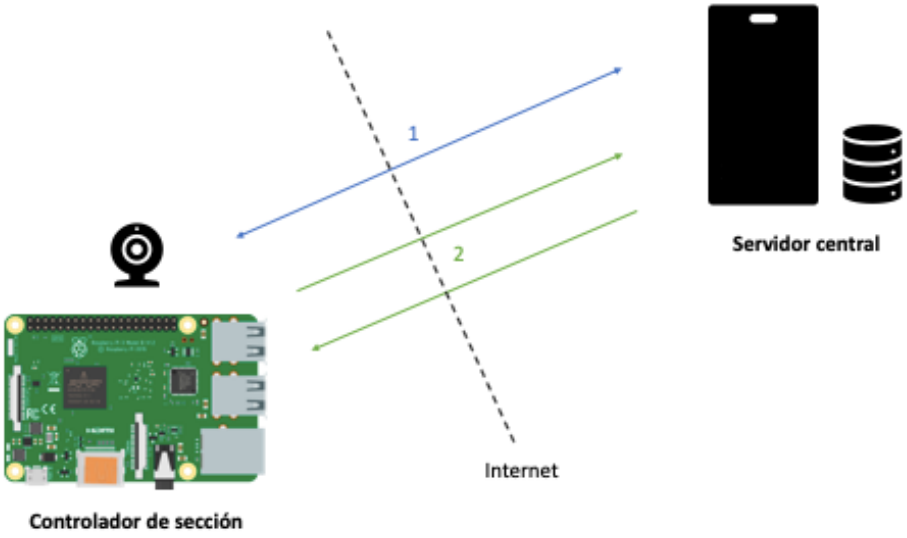


Figura 3.5 Red Sección-Servidor central.

En la anterior imagen, la línea discontinua se utiliza para marcar la desconexión de las redes en las que se encuentran el controlador de sección y el servidor central. En esta imagen se necesita atravesar redes a través de Internet para alcanzar el servidor central. La conexión marcada con el número 1, muestra la conexión mediante WebSocket, bidireccional y activo en todo momento. Por esta conexión el controlador de sección recibirá las solicitudes de medidas o de activación de mecanismos que tendrá que retransmitir a la red ZigBee y también realizará la señalización para WebRTC. El intercambio de mensajes con el número 2, muestra un mensaje de petición de subida de datos, cuando se ha producido la recopilación y la respuesta a esta petición por parte del servidor central.

3.4.3 Aplicación

Aunque dentro del apartado de comunicaciones, aquí se explicará el diseño de la aplicación utilizada para el acceso a todos estos servicios proporcionados por el proyecto. Se sitúa aquí debido a que sin conexión, esta aplicación no proporcionaría ningún servicio y también porque esta aplicación en sí, es básicamente, conexión entre distintos componentes.

El desarrollo de esta aplicación se realizará en Angular 7, con la ayuda de los componentes diseñados por Material y algunos otros componentes de otras compañías. El objetivo de esta aplicación es proporcionar un punto de acceso al usuario que posee un huerto o varios y que quiere visualizar los datos proporcionados por los sensores instalados. En principio, esta web estará alojada en el mismo servidor central que proporciona la API REST.

Esta aplicación obtendrá los datos y todos los servicios creados en el servidor central a través de la API REST, realizando peticiones GET, POST, PUT o DELETE (CRUD) a las rutas HTTP correspondientes. Mediante una interfaz simple e intuitiva, tendrá funcionalidades para registrar, editar y borrar todo tipo de elementos que forman parte de nuestro huerto. El usuario accederá con su usuario y contraseña, previamente registrado, y podrá visualizar sus huertos, secciones, cámaras en cada una de las secciones, sensores y datos de cada uno de los sensores. Al igual que el controlador de sección, mantendrá un WebSocket conectado al servidor central en todo momento, con este obtendrá notificaciones push del estado del sistema o podrá iniciar la señalización con el controlador de sección que solicite e iniciar vídeo en tiempo real. También con el uso de WebSockets podrá hacer uso de las diferentes funcionalidades que proporciona este proyecto.

Un objetivo de esta aplicación además de gestionar los huertos es el de visualizar los datos, por ello, la aplicación contará con apartados en los que se mostrarán tablas con los últimos datos obtenidos y gráficas del histórico de medidas.

A continuación se mostrará el proceso completo de comunicación para una petición de datos en tiempo real y para el inicio de una sesión WebRTC.

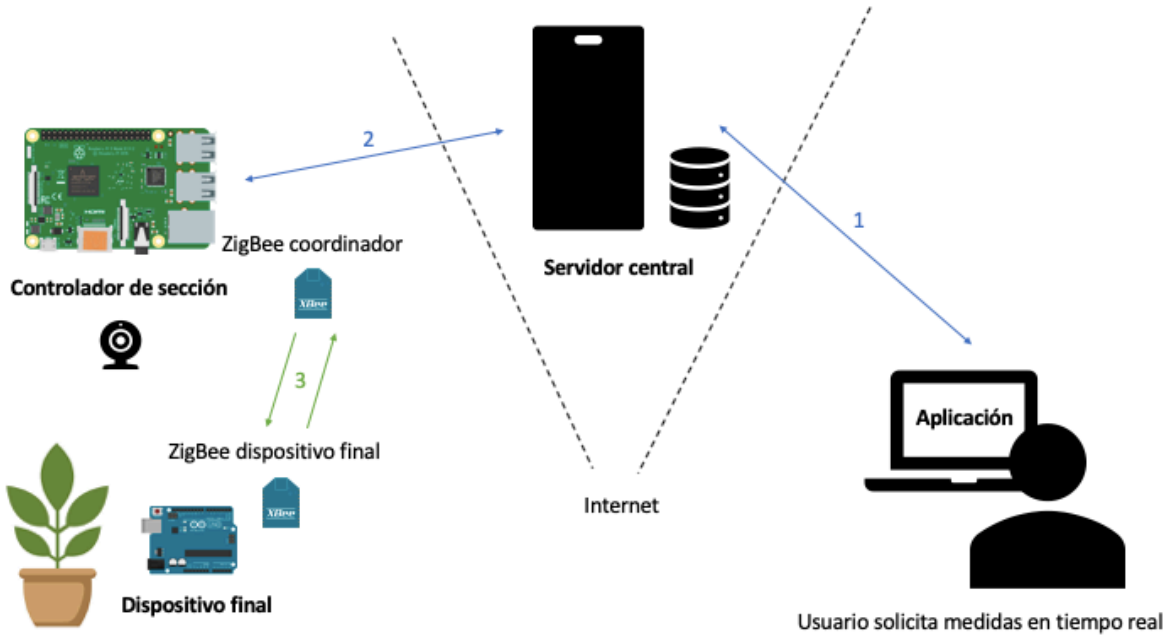


Figura 3.6 Solicitud de datos en tiempo real.

Como se puede ver en la figura 3.6, el usuario accede a la aplicación a través de Internet y realiza la petición de medidas, esa solicitud viaja por el WebSocket 1 (flecha bidireccional azul) al servidor central y es redirigido al controlador de sección correspondiente por el WebSocket 2. Al llegar al controlador de sección, busca el dispositivo final del que se trata y realiza la petición, con *act* igual a 1 al tratarse de una petición de datos, por la red ZigBee marcada con el número 3. La respuesta viaja en sentido contrario por todos los caminos antes mencionados.

Para el proceso de WebRTC entra en juego un servidor gestionado por Google, como se puede ver en la figura 3.7.

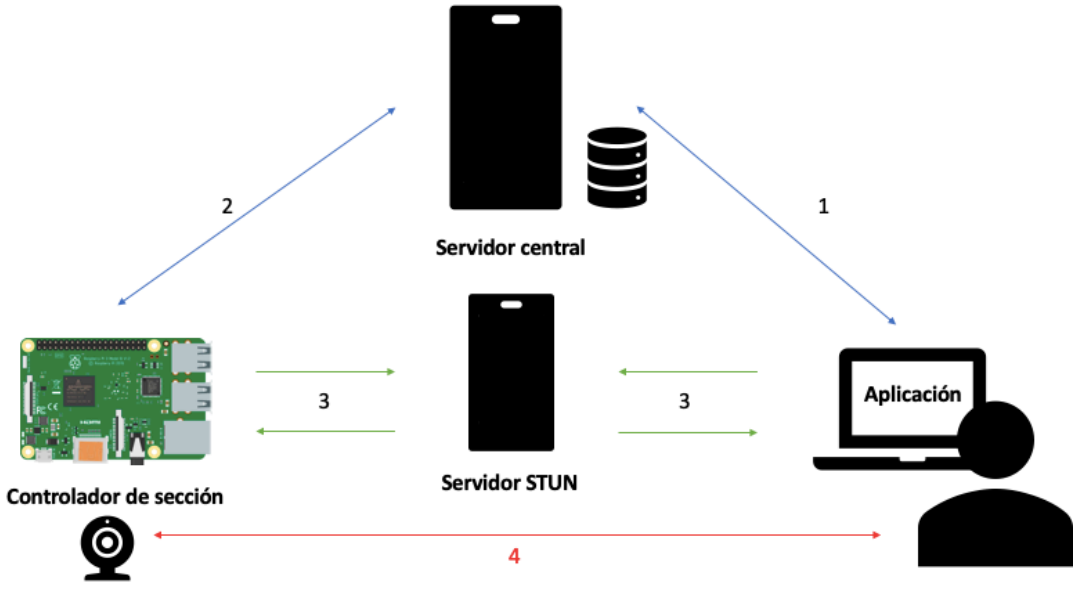


Figura 3.7 Proceso de señalización.

En la imagen no se han delimitado las redes puesto que todo este proceso ocurre a través de Internet.

El proceso de señalización se inicia con la conexión del controlador de sección al servidor central mediante un WebSocket, el que, tras conectar crea una “sala” para la espera de solicitudes de vídeo. Tras todo esto se inicia lo que se ve en la imagen superior, un usuario accede a la aplicación y decide iniciar una sesión WebRTC con un controlador de sección. La solicitud viaja desde el WebSocket del usuario (marcado con el número 1) hasta el WebSocket del controlador de sección (marcado con el número 2). Si el controlador de sección está conectado, se incluye en la misma “sala” al usuario y ambos realizan la petición a sus respectivos servidores STUN, en este caso es el mismo, propiedad de Google (número 3) para obtener su dirección IP pública. Si el controlador no está conectado, el usuario recibe un mensaje a través del WebSocket indicando que la sección no está disponible. Tras obtener la información del servidor STUN, ambos dispositivos (el navegador donde se ejecuta la aplicación y el controlador de sección) intercambian los mensajes SDP para crear la conexión directa. Si todo lo anterior ha funcionado correctamente, se inicia la conexión peer to peer entre ambos dispositivos (número 4).

En este ejemplo, a diferencia del ejemplo mostrado para explicar el proceso de conexión mediante WebRTC, no se utiliza un servidor TURN. Se prescinde de TURN porque en este proyecto solamente se quiere utilizar la funcionalidad peer to peer de WebRTC y con el uso de TURN, se necesitaría un servidor haciendo de intermediario durante toda la comunicación. En caso de que la comunicación peer to peer no sea posible, no se podrá mantener la comunicación.

Este proyecto realiza un uso responsable de los recursos por cada conexión iniciada para que el servidor pueda mantener un mayor uso de conexiones, por esto el uso de una API REST minimalista y el uso de WebRTC en modo peer to peer.

3.4.4 ORM

En este proyecto se ha utilizado un ORM (mapeo objeto-relacional), el cual permite que los mismos objetos que utilizamos en el programa principal sean los componentes de nuestra base de datos. Se ha escogido este modelo porque con la existencia de librerías como SQLAlchemy [20], se facilita muchísimo el proceso del manejo de la base de datos y mapeo a objetos. En el Capítulo 4: Desarrollo se mostrará el contenido de la base de datos diseñada mediante este proceso.

3.5 Infraestructura

Al principio del Capítulo 3: Diseño, se ha mostrado una imagen simplificada para explicar el papel de cada elemento en esta red. Realmente este proyecto se aplicará en dos escenarios distintos, dependiendo del escenario algunos de los elementos de la red estarán ubicados en lugares diferentes. El objetivo de este proyecto es acercar la agricultura ecológica al usuario aficionado mediante un producto atractivo que permita la modificación de sus componentes al estar basado en hardware libre, pero también es objetivo proveer a los agricultores comerciales de una herramienta útil que permita la mejora de sus procesos de cultivo. En función del objetivo final, la red utilizada será diferente. En este apartado se muestran ambas propuestas de diseño y se comenta el papel de cada uno de los elementos.

3.5.1 Uso aficionado

Para proporcionar un servicio sencillo, el usuario final sólo tendrá que obtener un controlador de sección y un dispositivo final para hacer uso del proyecto. Conectando el controlador de sección a Internet podrá hacer uso de la infraestructura proporcionada por este proyecto.

Registrando su usuario en el servidor central mediante la aplicación web diseñada podrá acceder a todo el sistema de gestión del huerto. Tras registrarse podrá crear su primer huerto y añadir los elementos que conforman su huerto, como son las secciones (una por cada controlador de sección) y como son los sensores o dispositivos finales, los que serán asignados a cada sección. Una vez puesto en marcha todo, esperando el tiempo necesario para que el dispositivo final haya tomado sus primeras medidas, se podrá ver accediendo a la aplicación los datos tomados por el sistema.

Figura 3.8 Huerto aficionado de ejemplo.

En la figura 3.8 se puede ver una sección delimitada en azul formada por las tres pequeñas áreas de cultivo y supervisadas por el controlador de sección (Raspberry pi). Dentro de esa sección tres dispositivos finales (Arduino UNO), uno en cada área de cultivo, se encargarán de tomar medidas y enviarlas al controlador de sección. Estos tres elementos, que para su instalación necesitarán una protección hermética, formarán la red ZigBee. Para la comunicación del huerto con el servidor central, el controlador de sección utilizará peticiones HTTP y un WebSocket, por estas razones es necesaria la conexión a Internet del controlador de sección.

El servidor central que se encargará de proporcionar la API REST y la aplicación en Angular será parte de la infraestructura gestionada por el proyecto en sí. El proyecto dará la posibilidad de utilizar la infraestructura de manera gratuita para la gestión de un único huerto formado por un único controlador de sección y varios sensores. Si el usuario quiere hacer uso de más huertos y/o secciones se proporcionarán unas tarifas para permitir el uso de más recursos del servidor gestionado por el proyecto.

3.5.2 Uso comercial

Debido a la posible utilización de elevados recursos del sistema la versión comercial del proyecto instalará una micro infraestructura igual que la utilizada por los usuarios aficionados pero gestionada de manera privada por los dueños del comercio en cuestión. Gracias a ello, los servidores centrales públicos de la aplicación no tendrán que manejar tanta carga y se podrán incorporar servicios particulares para cada comercio con un desarrollo a medida.

La instalación del proyecto se puede realizar sobre servidores ya existentes en la empresa, si no hiciera falta un servidor dedicado por la elevada carga de la red existente. Esto facilita la gestión de la red por parte del comercio y permite la instalación de servidores relay para que, en caso de que la comunicación peer to peer no funcione, se hagan cargo de que la comunicación mediante WebRTC siempre funcione.

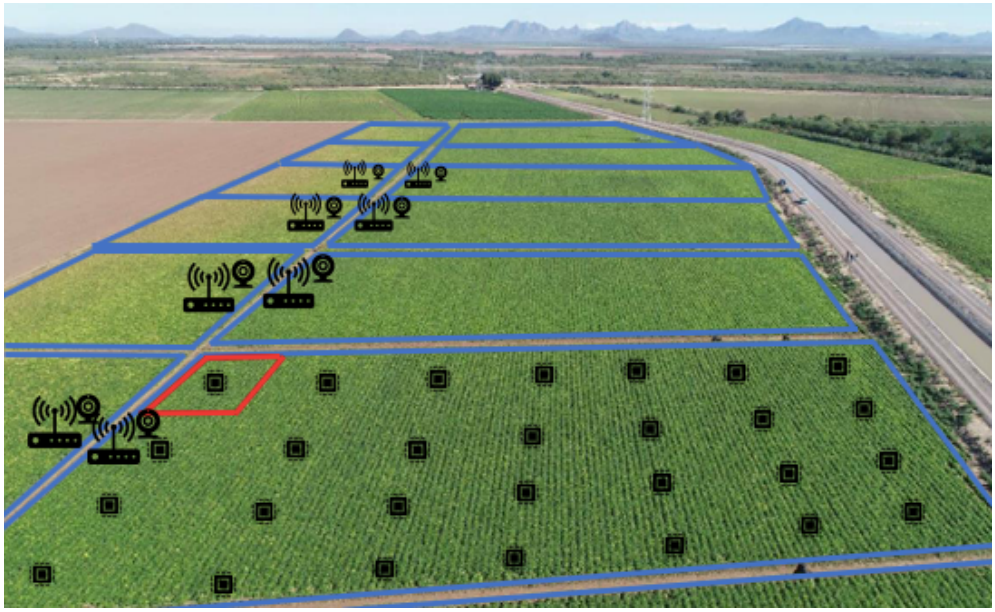


Figura 3.9 Secciones de un huerto comercial.

Arriba, en la figura 3.9, se muestra la distribución de un huerto comercial. En azul, como siempre, las secciones de dicho huerto supervisadas por el controlador de sección. La zona roja delimita el área que cubre un dispositivo final, puesto que conectar un dispositivo final a cada planta cultivada sería excesivamente caro. Por eso de cada sección se escogen plantas que representan a todo un sector dentro de esa sección y a esas plantas se les conecta el dispositivo final. En la imagen se han conectado veintiocho dispositivos finales en la sección de muestra. Esta muestra representaría la red ZigBee del huerto y junto con la red formada por los dispositivos finales y el servidor central, tendríamos la representación de un huerto comercial de ejemplo. La figura 3.10, muestra los diferentes huertos de una misma empresa, en amarillo.

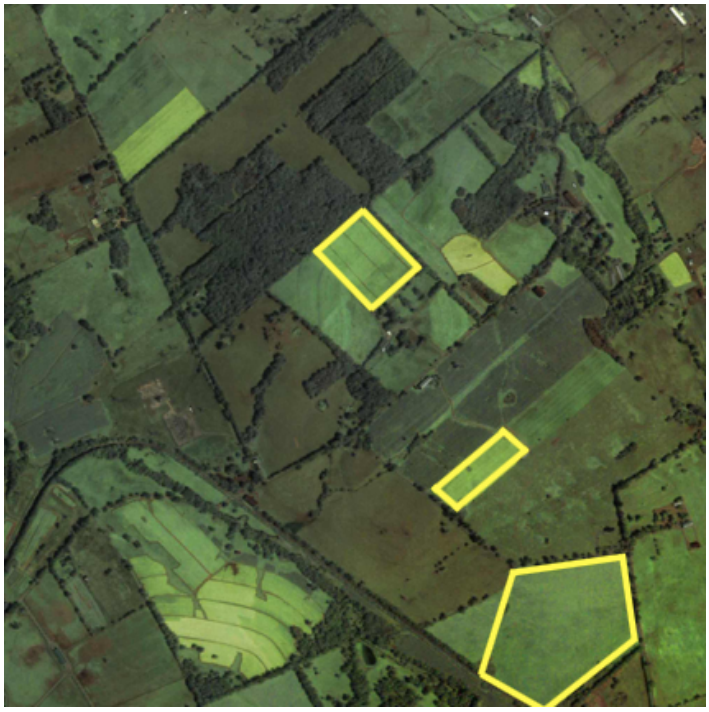


Figura 3.10 Huertos comerciales pertenecientes a la misma empresa.

Capítulo 4: Desarrollo

Este capítulo se dividirá en apartados, siendo cada apartado un hito en el desarrollo del proyecto. El orden de los apartados sigue el orden seguido en el proceso de desarrollo.

4.1 Dispositivo final

Se marcó como primer objetivo obtener las medidas de los sensores incorporados al dispositivo final. En las siguientes figuras se muestran los circuitos:

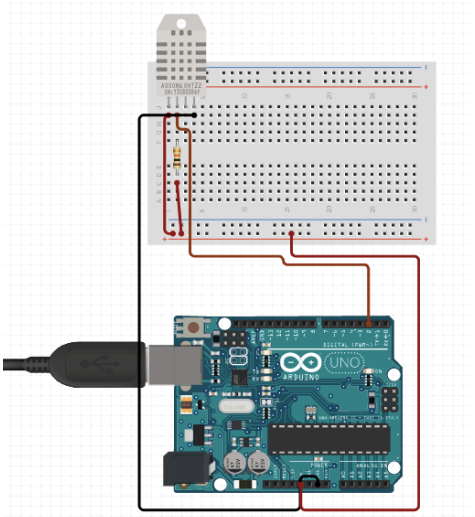


Figura 4.1 Circuito sensor DHT11.

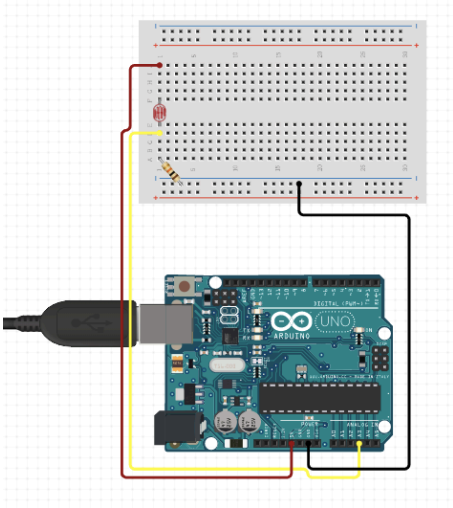


Figura 4.2 Circuito sensor luxómetro LDR.

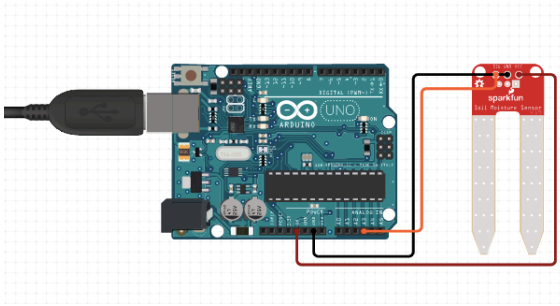


Figura 4.3 Circuito sensor FC28.

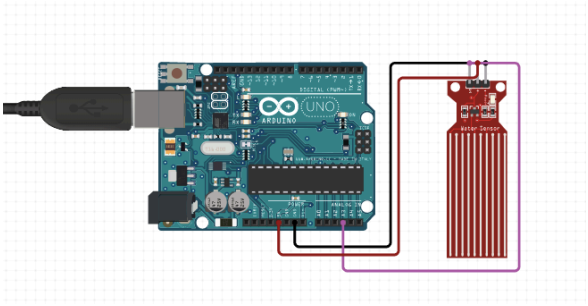


Figura 4.4 Circuito sensor HW38.

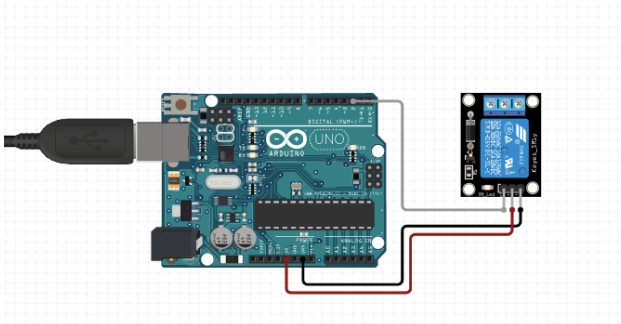


Figura 4.5 Circuito relé.

Para cada sensor de los mostrados arriba, se realizó un código particular para obtener la medida. Los circuitos son sacados del datasheet de cada componente o de ejemplos en Internet, como el luxómetro con LDR (véase figura 4.2). En las imágenes se puede ver en función de si el sensor proporciona una medida analógica o digital, el pin al que está conectado en Arduino varía, todos los circuitos que utilizan resistencias usan una resistencia de valor 10 kΩ, salvo el luxómetro que utiliza una resistencia de 5.03 kΩ. Para los valores de medidas proporcionadas en porcentaje se hace uso de la función map() y también se hizo uso de la librería para DHT11 creada por la empresa Adafruit.

El único punto a comentar sobre este desarrollo, es el cálculo de los luxes a partir del circuito de la LDR, que se adjunta aquí debajo.

Para obtener los luxes a partir del voltaje que cae en la resistencia de 5.03 kΩ se tendrá que reconvertir a voltios el valor medido por Arduino, puesto que al estar conectado a un pin analógico este valor está entre 0 y 1023. Esto es debido a que el conversor analógico-digital de Arduino UNO es de 10 bits. Para escalar este valor a los 5 V utilizados para alimentar el circuito se hace lo siguiente:

$$Voltaje\ resistencia = \frac{(float)\ valorDigital}{1023} \times 5V$$

Con esta operación se consigue pasar el valor *valorDigital* que es el obtenido de la medida de Arduino a la resistencia de 5.03 kΩ de un valor entre 0-1023 a un valor entre 0-5 V. Obtener el voltaje en la resistencia LDR es tan fácil como restarle a 5 V el valor obtenido en *Voltaje resistencia.* Aunque el valor verdaderamente interesante es el de la resistencia LDR, para ello se ha de hacer el siguiente cálculo:

$$Voltaje\ LDR = 5V - Voltaje\ resistencia$$

$$Resistencia\ LDR = \frac{Voltaje\ LDR}{Voltaje\ resistencia} \times 5.03\ k\Omega$$

Una vez calculado el valor de la resistencia LDR, para sacar los luxes se hace uso de los valores obtenidos de la recta utilizada para caracterizar a la función que define el valor de la iluminación en luxes frente al valor de la resistencia en ohms. Estos valores han sido sacados para este ejemplo [21].

$$Lux = 12518931 \times Resistencia\ LDR^{-1.405}$$

Aunque este dispositivo no resulta tan fiable como un luxómetro comercial, debido a variaciones de las mediciones con la temperatura o diferencias en la fabricación del propio dispositivo, permite tener una aproximación de un circuito luxómetro para el prototipo.

4.2 Red ZigBee

Debido a que las librerías utilizadas para el manejo de los dispositivos XBee sólo ofrecían soporte para el modo API el primer paso fue realizar pruebas de funcionamiento en modo transparente sin programar nada, solamente haciendo uso del software XCTU. Con estas pruebas se comprueba el correcto funcionamiento de los dispositivos y sirvieron también para familiarizarse con el programa XCTU.

El siguiente paso fue programar los dispositivos en el modo API, como se muestra en la figura 4.6.

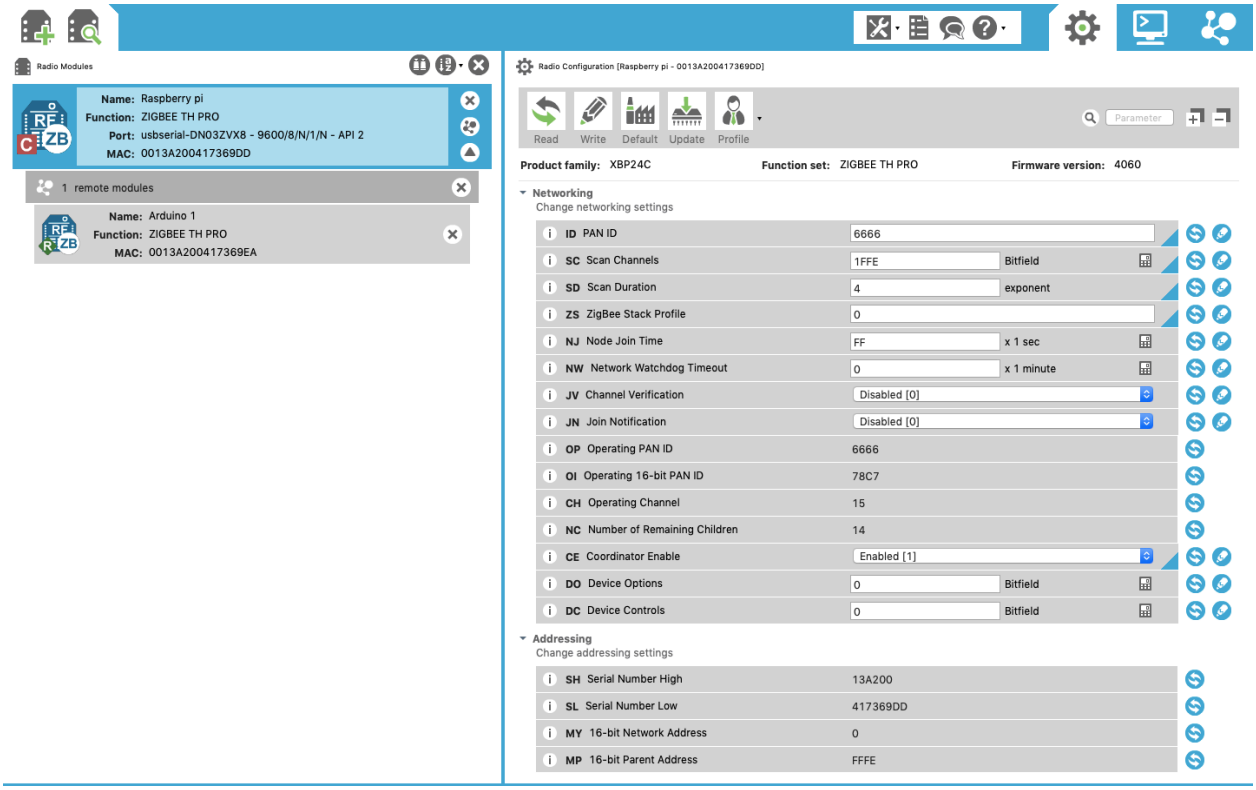


Figura 4.6 Programación en modo API con XCTU.

El *ID* es el identificador de la red que se utilizará en el proyecto, el valor *SC* define los canales a escanear por el dispositivo a la hora de unirse a una red. En el caso del coordinador, deja a su elección el canal en el que iniciar la red ZigBee. De todos los demás comandos, los detalles más interesantes son que el *CE* que define al dispositivo como coordinador y que en el caso de la red ZigBee, se ha decidido por no encriptar las comunicaciones por el incremento de bytes en la trama. La librería utilizada en Arduino para el manejo de XBee limita el tamaño máximo de paquete a 100 bytes, por eso se ha de prescindir de la encriptación en este desarrollo.

Después de configurar el coordinador con los parámetros comentados y todos los demás por defecto, se configuró el otro dispositivo de nuestra red como router ZigBee. En la siguiente imagen (véase figura 4.7) se puede ver la topología de la red creada.

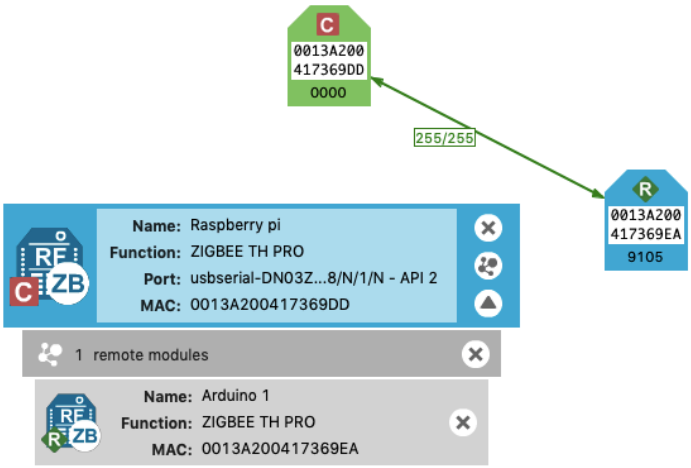


Figura 4.7 Topología en XCTU.

Una vez creada la red, se testó utilizando XCTU. Tras esto, el siguiente hito del proyecto fue conectar el dispositivo final al XBee configurado como router ZigBee y probar la comunicación con el coordinador de la red mediante XCTU. XCTU proporciona una interfaz de comunicaciones para enviar paquetes a los demás dispositivos conectados a la red. Primero se creó el primer código capaz de recibir paquetes, haciendo uso de la librería xbee-arduino [22]. Más tarde se implementó el código capaz de responder a estos mensajes. Y por último, como la comunicación de nuestro proyecto es mediante JSON se instaló un parseador de JSON, haciendo uso de la librería ArduinoJson [23].

4.3 Controlador de sección I

Para poder cerrar la fase del proyecto de desarrollo de la red ZigBee, se comenzó con la interconexión entre el controlador de sección y el dispositivo final. Debido a que el controlador de sección es el dispositivo que más depende del desarrollo de otros elementos, se dividirá este apartado en varios, para continuar con la explicación de este elemento cuando la infraestructura de la que necesita esté desarrollada y comentada.

En este apartado del desarrollo del controlador de sección, se utiliza la librería para Python 3 de Digi, python-xbee [24]. Con esta librería se realizó la conexión con el dispositivo XBee conectado al dispositivo final, se envío un mensaje JSON y se obtuvo una respuesta.

Una vez realizada la red ZigBee y teniendo el código del dispositivo final terminado y la parte de conexión con el dispositivo final del controlador de sección terminada también, se optó por desarrollar el servidor central para una vez terminado, escoger como conectar el controlador de sección a él.

4.4 Servidor central

Para el desarrollo de este servidor se utilizó Flask y varias de sus extensiones disponibles, muchas de ellas creadas por Miguel Grinberg, al que este proyecto debe mucho.

Antes de iniciar el desarrollo de esta parte del proyecto se utilizaron herramientas de control de revisiones como Git [25] y el repositorio online GitHub [26] para mantener varias copias del proyecto y aplicar los cambios realizados al código. También se utilizó Flask-Migrate [27] para las migraciones de bases de datos, es decir, aplicar cambios en bases de datos ORM existentes, sin tener que borrar la base de datos e iniciarla de nuevo. Estas herramientas junto con linters (correctores y predictores) para los diferentes lenguajes de programación utilizados han facilitado enormemente el desarrollo de este proyecto.

Lo primero que se realizó en Flask fue una url de prueba para familiarizarse con el framework. Una vez probado, con ayuda de Postman [28] que es un entorno de desarrollo de APIs, se comenzaron a realizar las primeras rutas que recibían JSON y respondían JSON en función de esa petición. A la par que el desarrollo de la API REST, se modelaron los objetos necesarios para la creación de la base de datos y el funcionamiento deseado de nuestra aplicación.

Debido a que el servidor tiene un sistema de usuarios, con la extensión de Flask Flask-HTTPAuth se restringe el acceso a las rutas que lo requieran si la petición no incluye el token de autenticación. Para generar el token de autenticación se creó un método en la clase User, que modela la tabla *User*, mediante la clase TimedJSONWebSignatureSerializer de la librería itsdangerous. Esto permite cifrar un dato que identifique al usuario de manera única (como el identificador de la base de datos) con una clave que solo conozca el servidor y que tenga una validez de un tiempo determinado por el servidor también, así nadie podrá conocer el token que se está utilizando en el sistema, puesto que caduca cada cierto tiempo. Para este proyecto, el token expira a los 600 segundos.

Junto al método de creación del token, se ha de crear un método de validación del token para comprobar si el token es válido o no. En función de la respuesta de este método, el servidor servirá la respuesta a la petición de manera satisfactoria o denegará el acceso. Este método tiene la peculiaridad de ser estático, puesto que tiene que devolver un objeto de la clase User, para saber que usuario está autenticándose, pero sin tener un objeto definido desde el que llamarlo. Así, mediante el token, se obtendrá el usuario que está autenticándose. Una vez desarrollado el método de creación y verificación, mediante el uso de un decorador aportado por Flask-HTTPAuth se proveerá de verificación con token a las rutas que se desee. Este ejemplo ha sido basado en el tutorial de Miguel Grinberg [29], para securizar APIs REST.

A la vez que el desarrollo de las primeras rutas y junto con el desarrollo de la verificación por token en servidor, se crearon los objetos en Python 3 que gracias a SQLAlchemy, modelarán las tablas de la base de datos ORM. El objeto User además de tener todos los datos de la tabla *User* tendrá los métodos antes mencionados para poder identificar a un usuario registrado cuando entrega un token de verificación. Tras desarrollar todos los objetos utilizados por la aplicación, la base de datos es la mostrada en la figura 4.8:

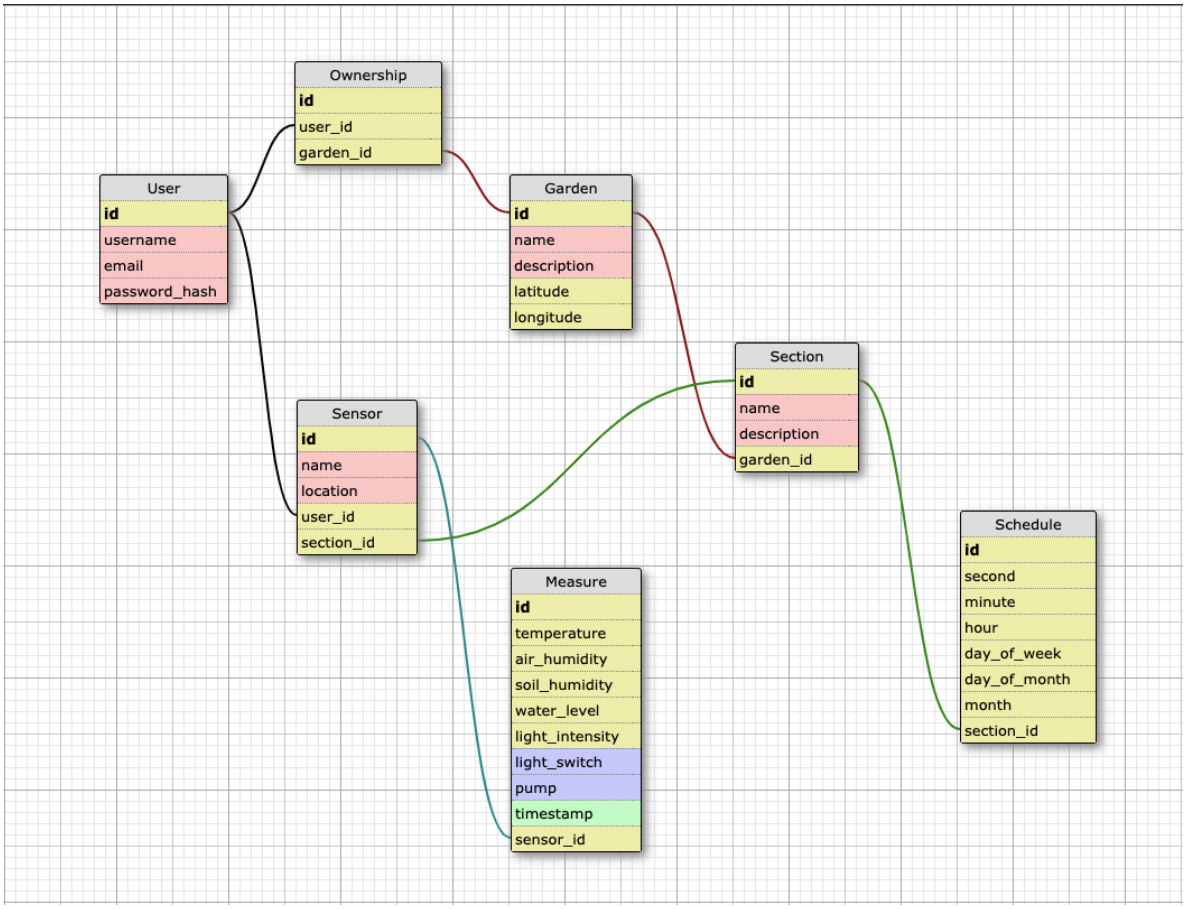


Figura 4.8 Esquema de la base de datos.

En esta base de datos, la tabla *User* representa al usuario que utiliza la aplicación, la tabla *Garden* representa a un único huerto y la tabla *Ownership* se utiliza de apoyo para la relación entre *User* y *Garden*. En cambio la tabla *Section* representa a una sección dentro de un mismo huerto, esta tabla representa a los controladores de sección dentro de cada huerto. En el interior de cada una de estas secciones están los sensores que son representados por la tabla *Sensor*, también de la tabla *Section* aparece la tabla *Schedule* que será utilizada en el futuro para periodizar las medidas que tomarán los sensores dentro de cada sección. Por último la tabla *Measure* representa una única medida tomada por un sensor y se utiliza para poder mostrar gráficas y otros datos de interés al usuario. Las tablas contienen campos fáciles de entender cuando el usuario sabe qué modela cada tabla.

En cuanto a las relaciones de cada tabla, se puede ver en la imagen que la única relación many to many es la de la tabla *User* con la tabla *Garden* utilizando la tabla de apoyo *Ownership* para este fin. Gracias a esta relación se consigue que un único huerto pueda ser gestionado por varios usuarios, una funcionalidad bastante útil para esos huertos compartidos. Las demás relaciones son todas one to many o one to one, en un único caso. De las relaciones se puede ver que un huerto representado por la tabla *Garden* puede tener muchas secciones, representadas por la tabla *Section*. A su vez una sección puede tener muchos sensores, representados por la tabla *Sensor*, aunque el sensor únicamente tendrá un dueño. La única relación one to one es la existente entre la tabla *Section* y *Schedule*, esto se debe a que una sección (la encargada de tomar los datos de todos los sensores conectados a ella) sólo tendrá una periodización a la hora de tomar los datos. Por último, un sensor tendrá muchas medidas, representadas por la tabla *Measure*.

Una vez creadas la base de datos y el sistema de autenticación por token y añadido el primer usuario se hizo la siguiente prueba:

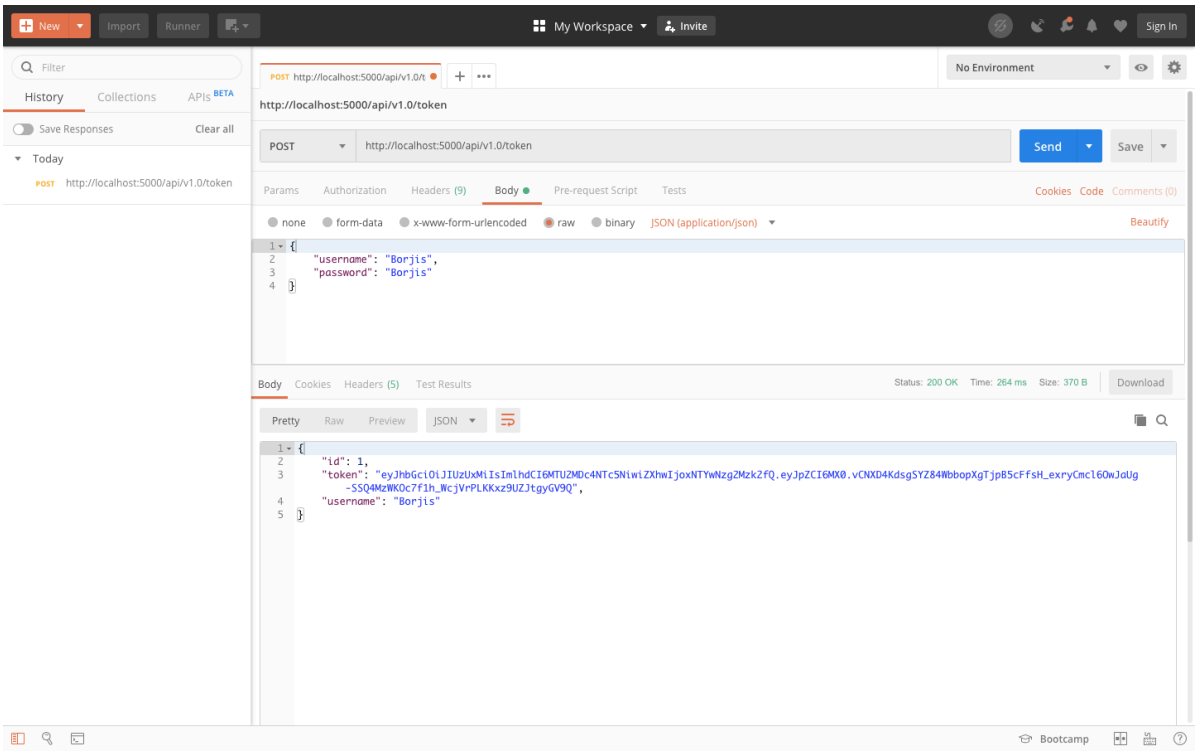


Figura 4.9 Petición de token con Postman.

En la figura 4.9 se muestra un ejemplo de uso de la aplicación Postman. En dicho ejemplo se realiza una petición HTTP POST al servidor alojado en http://localhost:5000, en la que adjuntando los credenciales del usuario en un JSON formado por los campos *username* y *password* el usuario obtiene de vuelta un JSON formado por sus datos de usuario y su token de autenticación.

El siguiente paso fue la creación de las rutas que darían funcionalidad al proyecto. Las primeras rutas consistían en un CRUD para todos los objetos utilizados en el proyecto excepto las medidas, dado que la gestión de las medidas la realizará el servidor internamente y ningún usuario tendrá acceso a esta función. Rutas HTTP para crear, editar, obtener y eliminar todos los objetos. Además de todas estas rutas básicas y alguna más para obtener datos específicos, se creó una ruta para que los controladores de sección puedan subir ficheros JSON con medidas realizadas por sus sensores, la ruta HTTP acepta ficheros JSON que son parseados por el servidor central para añadir las medidas de los sensores a la base de datos.

Todos los métodos utilizados por el CRUD creado devuelven el código HTTP 200 si el funcionamiento del método ha sido el esperado, HTTP 400 si la petición no contiene todos los datos necesarios, HTTP 401 si está desautorizado, HTTP 403 si no se tiene acceso a ese recurso en concreto, HTTP 404 si el recurso que se estaba buscando no se encuentra y HTTP 5xx para los errores producidos por el servidor, siendo x cualquier número.

Finalizado el CRUD y las demás rutas HTTP, además de testadas una a una con Postman para comprobar su correcto funcionamiento, se comenzó con el despliegue del servidor de WebSockets. Para esto se hizo uso de la librería Flask-SocketIO y el servidor proporcionado por eventlet. Las rutas HTTP pasaron a un archivo a parte dentro del servidor llamado *routes.py* mientras que toda la funcionalidad relacionada con WebSocket pasó al archivo *websocket.py*. En este archivo se crearon los métodos necesarios para la conexión del WebSocket con el usuario autenticado a través de la aplicación y con los diferentes controladores de sección. También se crearon los métodos de petición de datos en tiempo real. Aunque esta funcionalidad no se pudo testar hasta que la aplicación desarrollada en Angular y el código de conexión mediante WebSockets del controlador de sección estuvieran lo suficientemente avanzados.

4.5 Controlador de sección II

4.5.1 Comunicación Sección - Servidor central

Creada la API REST al completo y parte de la lógica utilizada por los WebSockets se retomó el desarrollo del controlador de sección. Además del archivo Python 3 que con los métodos de comunicación por ZigBee, se creó el programa principal que será alojado en el controlador de sección. Este programa hace uso de SocketIO, la librería de WebSockets para Python 3 y de APScheduler, librería que nos permitirá crear las tareas de recolección de datos.

Este programa se inicia con la conexión de un WebSocket al servidor central, una vez iniciada escucha los eventos creados en el servidor central de petición de datos en tiempo real. Se utilizó APScheduler para la gestión de tareas tipo Cron, y no Cron porque los programas tenían que ejecutarse en el mismo contexto. Al tratarse de un único XBee el que realiza las peticiones de medidas en tiempo real y la ejecución de la recolección de datos en segundo plano, no podemos realizar las dos tareas a la vez. Es por esto por lo que se ha de emplear un mecanismo que bloquee el acceso a este recurso limitado, por ello el método de comunicación con otros XBee hará uso de un Lock. Lock es un elemento del lenguaje Python dentro de la librería threading que permite bloquear la ejecución de un método mientras otro thread lo está ejecutando. Un thread es un contexto de ejecución de una aplicación, por motivos de optimización, algunas aplicaciones utilizan más de un thread para ejecutar tareas en paralelo y así reducir el tiempo de ejecución del programa principal.

En el programa principal del controlador de sección, la conexión con SocketIO se realiza en un thread principal que comprueba la conexión constantemente y permite la recepción de mensajes WebSocket sin bloquearse. La librería APScheduler también inicia un thread a parte cuando se inicia una tarea programada. Por estas dos razones, como las llamadas al método de comunicación con XBee se realizan en threads distintos se implementará un Lock en el método para bloquear una vez comience un thread a ejecutarlo y lo desbloqueará tras su utilización. Esta también es la razón por la que se hace uso de APScheduler y no Cron. Cron ejecutaría un script de Python en un contexto en el que el uso de Lock no tendría utilidad, puesto que al ejecutarse el programa que conecta mediante WebSocket en otro contexto, no sería el mismo Lock por lo tanto no surtiría efecto. Tras el desarrollo y prueba del programa que realiza la conexión con el servidor principal mediante WebSockets, se implementaron los métodos de recepción de mensajes y reenvío al correspondiente dispositivo final.

El reenvío de mensajes se realiza tras la recepción del mensaje mediante WebSockets, un evento definido con SocketIO en función del tipo de mensaje será activado en el controlador de sección, esto hace que se envíe un determinado mensaje por la red ZigBee a un dispositivo final específico. Los mensajes transmitidos por WebSockets para la activación de mecanismos o petición de datos en tiempo real son los mostrados en el apartado 3.4.1 Red ZigBee con el nombre del dispositivo final añadido para poder ser identificado por el controlador de sección.

En el inicio del programa creado para el controlador de sección, se invierten unos segundos de tiempo para que el coordinador ZigBee adquiera todos los dispositivos conectados a su red, estos dispositivos son almacenados en una lista en Python 3 para que a la hora de que un controlador de sección reciba un mensaje a retransmitir por la red ZigBee, sepa si ese dispositivo está conectado a la red o no.

El método encargado de ejecutarse periódicamente con APScheduler debe preguntar a cada uno de los dispositivos finales todos los datos de sus sensores y mecanismos en ese mismo momento. Puesto que no se quería abusar de la capacidad de Arduino UNO como dispositivo final, el controlador de sección tras enviar el mensaje *act* tipo 1 a un dispositivo final de la red almacenará la respuesta del dispositivo junto con *timestamp* del momento de la recepción en formato UTC. El formato es UTC para que, en última instancia sea la aplicación que muestra los datos la encargada de dar formato a la fecha en función de la configuración del sistema del usuario. La respuesta del dispositivo final es almacenada en un archivo llamado *measures.json* por el controlador de sección que vuelve a repetir este proceso por cada uno de los dispositivos finales conectados a su red ZigBee.

Una vez recopilada toda la información de la red, sacada de la lista adquirida en el inicio del programa, el controlador de sección trata de subir la información al servidor central mediante el uso de la API REST desarrollada en el apartado anterior. Si el controlador de sección tiene conexión con el servidor central tratará de subir el fichero y en caso de que reciba un código HTTP 200 garantizando que la comunicación ha ido bien, eliminará el fichero *measures.json.* En cambio si no tiene conexión o no recibe el código 200 tras tratar de enviar el fichero, lo mantendrá para más tarde, recuperada la conexión, tratar de subir el fichero.

4.5.2 WebRTC

Cuando el desarrollo de las comunicaciones con el servidor central para la recopilación de datos se terminó comenzó el desarrollo de las comunicaciones mediante WebRTC. Esta fue una de las partes con más cambios durante el desarrollo, se probaron librerías distintas y diferentes métodos antes de que se diera con el método que actualmente funciona.

En el primer momento se trató de usar la librería aiortc para Python 3, la primera prueba se realizó en un ordenador portátil y no en la propia Raspberry pi. El funcionamiento del programa basado en esta librería fue el esperado, pero al migrar el código del ordenador portátil al ordenador de placa reducida éste no funcionaba. La Raspberry pi no tiene la suficiente potencia de procesamiento para hacer uso de esta librería y pasados pocos segundos de ejecución de la transmisión por WebRTC la CPU alcanza niveles elevados de uso y la ejecución del programa en Python como del sistema operativo se finaliza. Se intentaron solucionar estos problemas contactando con el autor del código pero por falta de tiempo se optó por buscar otra solución.

Las opciones oficiales que ofrece el proyecto WebRTC se basan en la API de JavaScript utilizada en el contexto del navegador que se ejecuta el código, esta es una opción únicamente para navegadores web. También cabe la posibilidad de compilar el código en C que proporcionan para que los desarrolladores de navegadores web puedan incluir WebRTC de manera nativa. La compilación en C de este código, su comprensión y su correcto funcionamiento llevarían demasiado tiempo, por lo que esta segunda opción se descartó. Antes de optar por la solución final, se probaron más librerías para Nodejs, con poca fortuna debido a que no estaban siendo mantenidas desde hace años.

Finalmente se optó por diseñar una pequeña aplicación web, lo más reducida posible para poder hacer uso de la API de JavaScript y adapter.js, la cual había sido probada previamente realizando el tutorial del codeab de Google [30]. La desventaja que plantea esta opción además de la utilización de una aplicación web y de no poder unificar el desarrollo de toda la aplicación que se ejecuta en el controlador de sección en un mismo script de Python, es la necesaria utilización de dos WebSockets por cada controlador de sección, puesto que como la parte desarrollada en WebRTC utiliza WebSockets para la señalización, es necesario mantener un WebSocket con esta parte también (véase figura 4.10).

Esta solución se trata de un "workaround" para poder proveer de esta funcionalidad a nuestra aplicación. El futuro desarrollo de la aplicación consistirá en trabajar con las librerías en C del proyecto WebRTC y unificar todo el desarrollo en Python, debido a que Python tiene la posibilidad de ejecutar secciones del programa en C, al estar basado en este lenguaje.

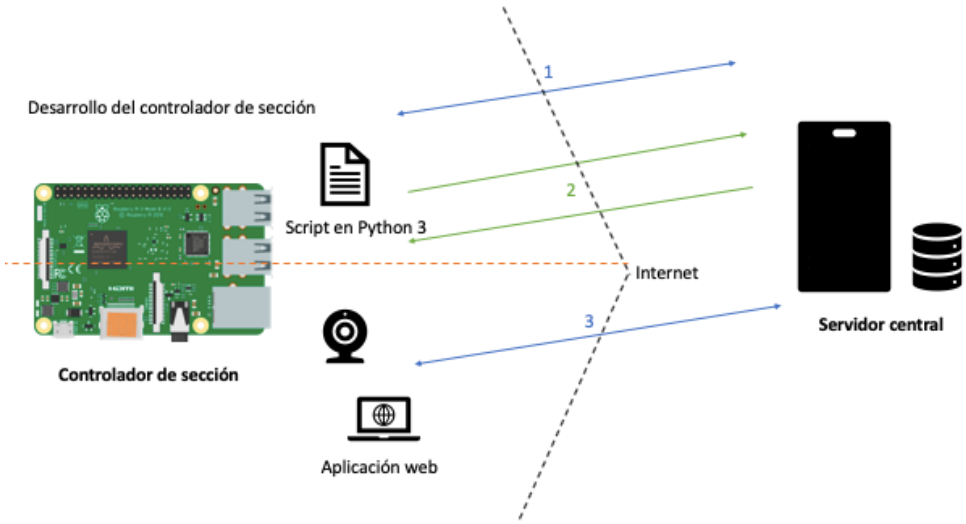


Figura 4.10 Desarrollo final de las comunicaciones Sección - Servidor central.

Como se puede ver en la imagen superior ahora el controlador de sección cuenta con dos WebSockets, numerados 1 y 3. Se ha dividido el controlador de sección para mostrar que hay dos aplicaciones coexistiendo, la primera el script en Python 3 que se encarga de la comunicación con los dispositivos finales, la gestión de las peticiones en tiempo real y la subida de los datos al servidor central usando un WebSocket y peticiones HTTP y por otra parte la aplicación desarrollada en Angular para la utilización de WebRTC ejecutándose en un navegador que utiliza otro WebSocket.

Este "workaround" dificulta la gestión de los usuarios conectados, puesto que ahora hay que mantener dos WebSockets abiertos por sección y hay que crear un nuevo punto de acceso en el servidor central para lo que se llamará *WebSocket alternativo* en el código desarrollado.

Comentado el diseño del "workaround" se procede con el desarrollo del código. Para el uso de WebSockets en el navegador se utiliza la librería de SocketIO para JavaScript. La aplicación web será desarrollada en Angular como la aplicación a la que accederán los usuarios desde el servidor central. Para la gestión de las comunicaciones mediante WebSockets se creará lo que se denomina servicio en la arquitectura de Angular. La aplicación consistirá en una única vista que se encargará de conectar con el servidor central por WebSocket y será incluida en una "sala" a la espera del evento de conexión generado por SocketIO para iniciar el intercambio de mensajes con WebRTC. Cuando el usuario inicie la comunicación, se compartirán los mensajes SDP que contienen toda la información acerca de la conexión y del códec y la tasa de transmisión a utilizar por codificadores y decodificadores de vídeo en cada uno de los elementos conectados. Una vez finalizada la comunicación, el usuario abandonará la "sala" generada por SocketIO y el controlador de sección eliminará los datos de la conexión esperando una comunicación nueva.

En este desarrollo, se ha pensado al usuario como el que siempre inicia la comunicación, por eso el papel del controlador de sección es el de recibir información, incluirla en los objetos necesarios para que WebRTC funcione y responder a esta petición de inicio de comunicación.

El primer proceso dentro de la API de WebRTC es permitir el acceso a la cámara mediante el método *navigator.mediaDevices.getUserMedia()*, con este método el usuario puede acceder a su cámara para así generar un flujo de vídeo que podrá ser enviado mediante WebRTC. Los siguientes pasos son generar el objeto *peerConnection* que contendrá la información de la conversación y al que, tras el proceso de comunicación mediante el servidor de señalización, se añadirá la información de red y de códec del otro punto con el que se está comunicando. Los únicos mensajes que deben ser compartidos mediante SocketIO, en el llamado proceso de señalización, son los generados por los eventos de creación de "oferta". Estos mensajes son los que contienen el SDP que informa al otro *peer* sobre nuestra red y códec utilizado.

Aunque teóricamente este proceso serviría para que múltiples usuarios se conectasen y vieran el vídeo de la cámara instalada en la Raspberry pi, debido al hecho de que mantener una comunicación entre dos *peers* eleva el uso de CPU a picos del 90% en la Raspberry pi, se cree que con el dispositivo y la implementación actual no se podrá aceptar más llamadas cuando un usuario ya está haciendo uso del streaming. Cabe señalar también que WebRTC está optimizado para comunicaciones entre dos usuarios, puesto que el uso de comunicación p2p (peer to peer) para conferencias multipunto no está recomendado puesto que cada usuario de la comunicación tendrá que crear un objeto por cada uno de los flujos de vídeo que recibe a la vez que codificar y enviar su vídeo a cada uno de los usuarios. Para el uso de conferencias multipunto con WebRTC se debería hacer uso de un servidor que reenvíe los flujos de vídeo a los usuarios, como se puede ver en la imagen inferior. Existen multitud de proyectos basados en WebRTC como Janus [31] que realizan este tipo de comunicación pero este proyecto quería realizar su propia implementación de WebRTC y no utilizar tecnologías de terceros más allá que las librerías utilizadas.

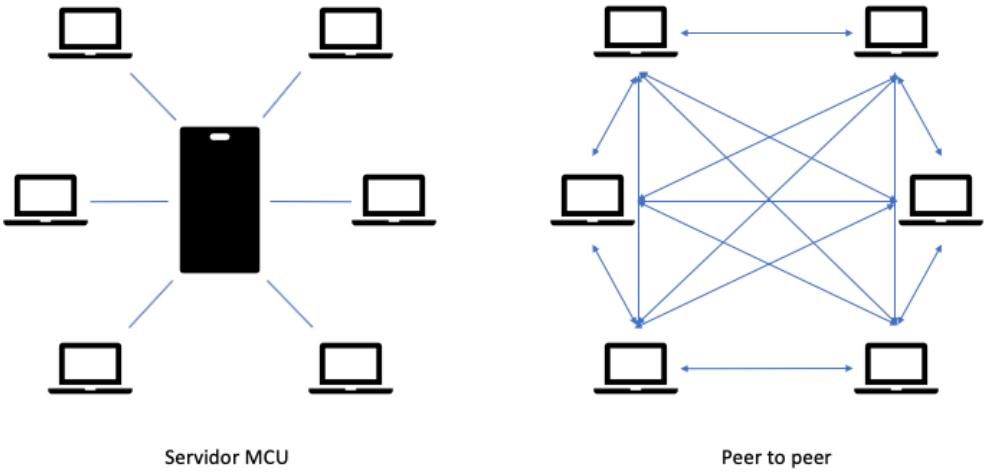


Figura 4.11 Comunicación a través de servidor vs p2p.

En la imagen superior se muestran las conexiones que tendría una conferencia con el uso de una infraestructura basada en un MCU (multipoint conferencing unit) frente a las conexiones creadas en una infraestructura peer to peer. Y los recursos consumidos por estas redes no se limitan sólo a las conexiones, para el entorno peer to peer una máquina tiene que codificar cada flujo de vídeo independientemente para cada una de las máquinas conectadas. Es por esto por lo que, en el estado actual del proyecto, sólo un usuario podrá ver el estado de la cámara de una sección particular simultáneamente.

4.5 Aplicación

El último objetivo en el desarrollo del proyecto fue poner una interfaz gráfica a todo lo realizado. Sin esta parte del proyecto, la funcionalidad existe pero es necesaria una interfaz sencilla para acceder a los servicios y visualizar los datos. El primer paso fue iniciar un nuevo proyecto en Angular, como el realizado para el uso de WebRTC en el controlador de sección. Esta aplicación contará con una vista de inicio para usuarios no identificados y una vista de acceso para iniciar sesión o registrar un nuevo usuario. Si el usuario accede con su contraseña podrá visualizar diferentes vistas que mostrarán sus huertos, las secciones de cada huerto, los dispositivos finales o sensores de cada sección y por último las medidas de cada sección.

Cada vista en Angular es creada con lo que Angular denomina componente. Un componente es una mezcla de HTML, CSS, TypeScript y un fichero de test unitario. La aplicación desarrollada tiene los siguientes componentes: *Home* que representa el inicio de la aplicación, *Login* que representa la vista de acceso o registro de un usuario, *Gardens* que muestra todos los huertos que el usuario posee, *Sections* que en función del huerto desde el que se ha accedido muestra las secciones correspondientes, *Sensors* que en función de la sección desde la que se ha accedido muestra los dispositivos finales correspondientes, *Measures* que muestra la información y da acceso a los servicios de un dispositivo final y otras vistas como *Maps* que muestra un componente con Google Maps incorporado donde salen las ubicaciones en el mapa de los huertos del usuario o *Analytics* donde se mostrarán gráficas de las últimas treinta medidas tomadas por cada uno de los dispositivos finales del usuario.

A continuación se muestran imágenes de algunas vistas de la aplicación realizada.

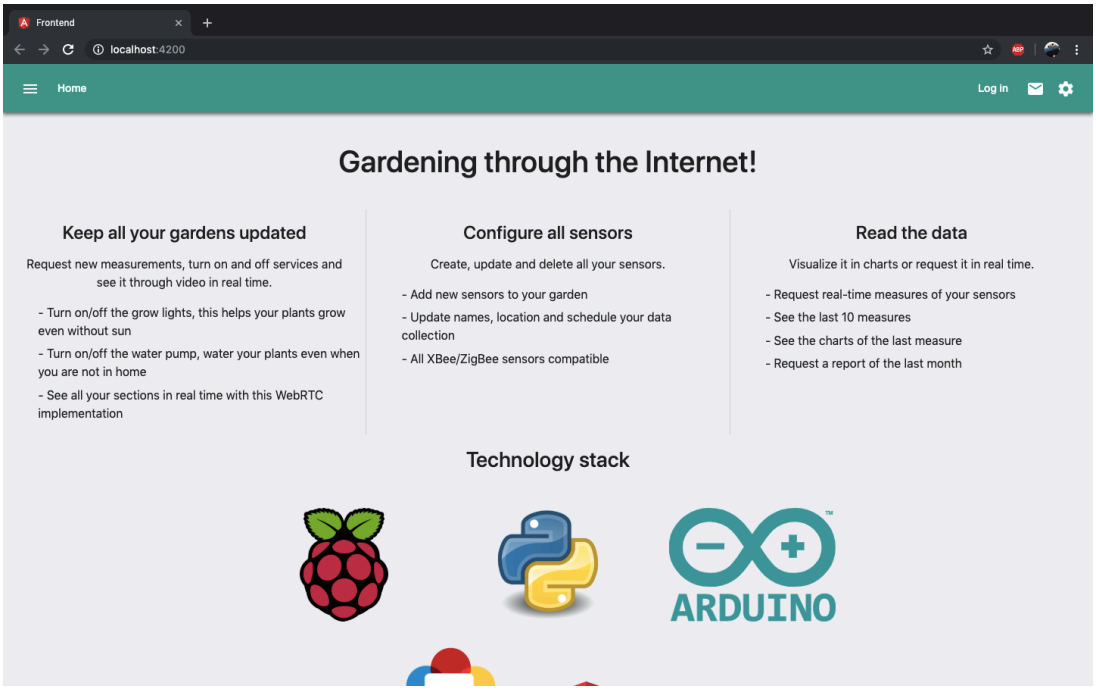


Figura 4.12 Home de la aplicación.

Esta vista es el punto de inicio de la aplicación en el navegador Chrome, cualquier usuario esté autenticado o no puede acceder a ella. En ella se muestran algunas de las características de este proyecto y también algunas de las tecnologías utilizadas. Haciendo click en *Log in* el usuario accederá a la siguiente pantalla (véase figura 4.13) donde podrá acceder a la aplicación o registrarse, rellenando un formulario como el siguiente:

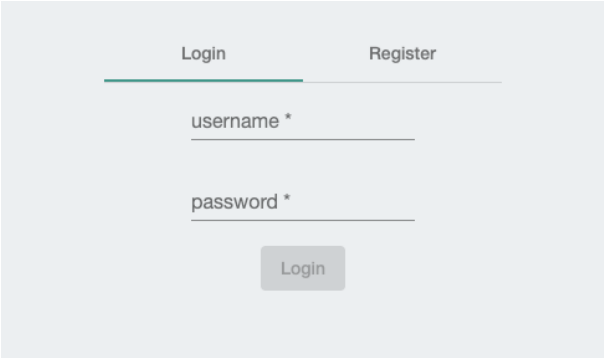


Figura 4.13 Formulario de Login.

Una vez introducido el nombre de usuario y la contraseña del usuario, este volverá a aparecer en el *Home* de la aplicación pero a diferencia de antes aparecerá su nombre de usuario en lugar de *Log in* indicando que ha accedido a la plataforma y habrá recibido un mensaje en el icono de la carta, indicando el id de WebSocket que está utilizando para la comunicación con el servidor central. Para acceder a las demás vistas, el usuario tendrá que hacer uso del menú lateral que aparece cuando haces click en el icono de las tres barras horizontales (véase figura 4.14). A continuación se muestra el menú lateral y algunas de las características antes comentadas.

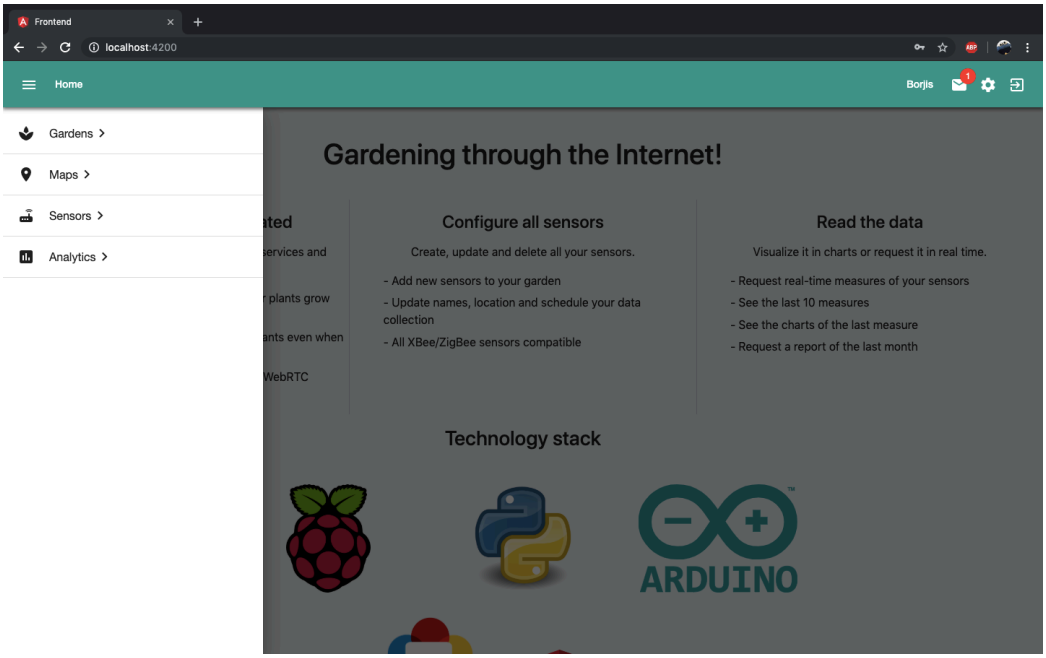


Figura 4.14 Menú lateral.

En la imagen superior se pueden ver algunas características comentadas como el nombre de usuario o el indicador de mensaje recibido, además del menú lateral desplegable que permite al acceso a las vistas *Gardens*, *Maps*, *Sensors* y *Analytics*.

El punto principal de la aplicación es la vista *Gardens*. En ella se pueden ver los huertos de un usuario, paginados de ocho en ocho. Aquí el usuario podrá acceder a las secciones de cada huerto haciendo click en el huerto, crear nuevos huertos y editar o borrar los existentes. En esta vista, los huertos aparecen con forma de tarjeta con una imagen descriptiva de la zona y una pequeña descripción de la ubicación o la finalidad del huerto (véase figura 4.15).

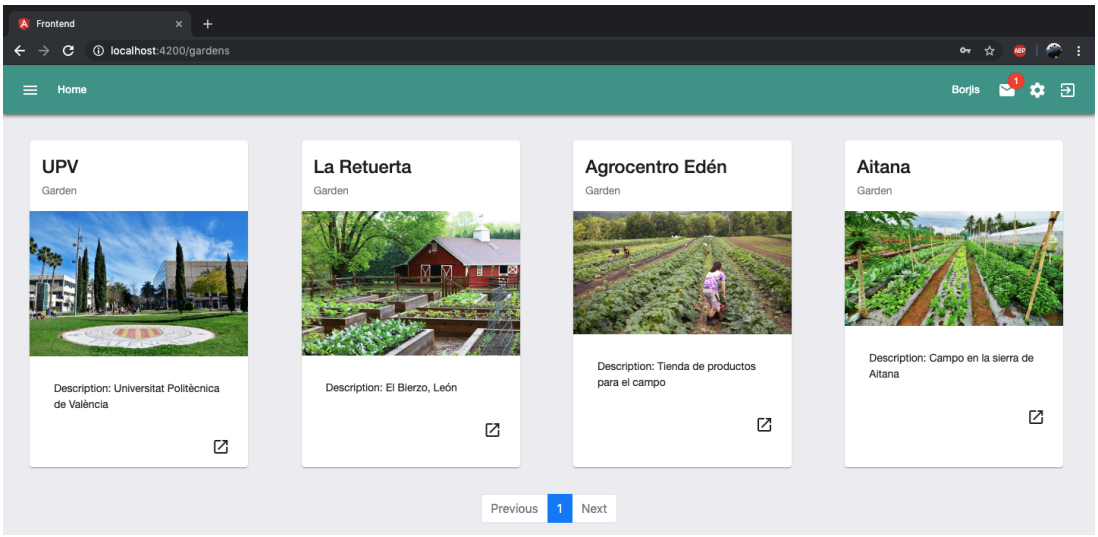


Figura 4.15 Vista Gardens.

Haciendo click en cualquiera de las tarjetas de los huertos de la imagen superior, se accede a la vista *Sections* correspondiente. Es decir, haciendo click en un huerto, se accede a las secciones de ese huerto. A continuación se muestra la figura 4.16, con la misma funcionalidad que la vista *Gardens*. Con el único objetivo de añadir una división más a cada huerto para una mejor gestión de ellos.

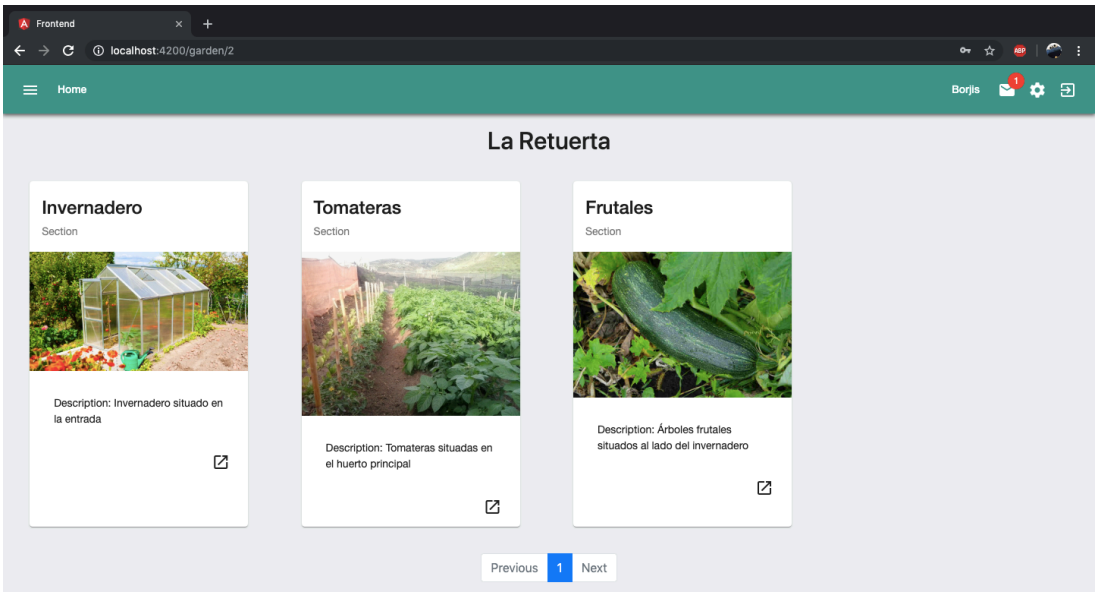


Figura 4.16 Vista Sections del huerto “La Retuerta".

En la imagen superior se pueden ver las secciones creadas para el huerto “La Retuerta”, como indica el título de la página. Este huerto consta de un invernadero, una zona con tomateras y una última zona destinada a árboles frutales. Haciendo click en cada una de las secciones mostradas se accede a la vista *Sensors*, que es la encargada de mostrar todos los dispositivos finales conectados a esa sección y también la encargada de mostrar el vídeo obtenido mediante la comunicación usando WebRTC (véase figura 4.17).

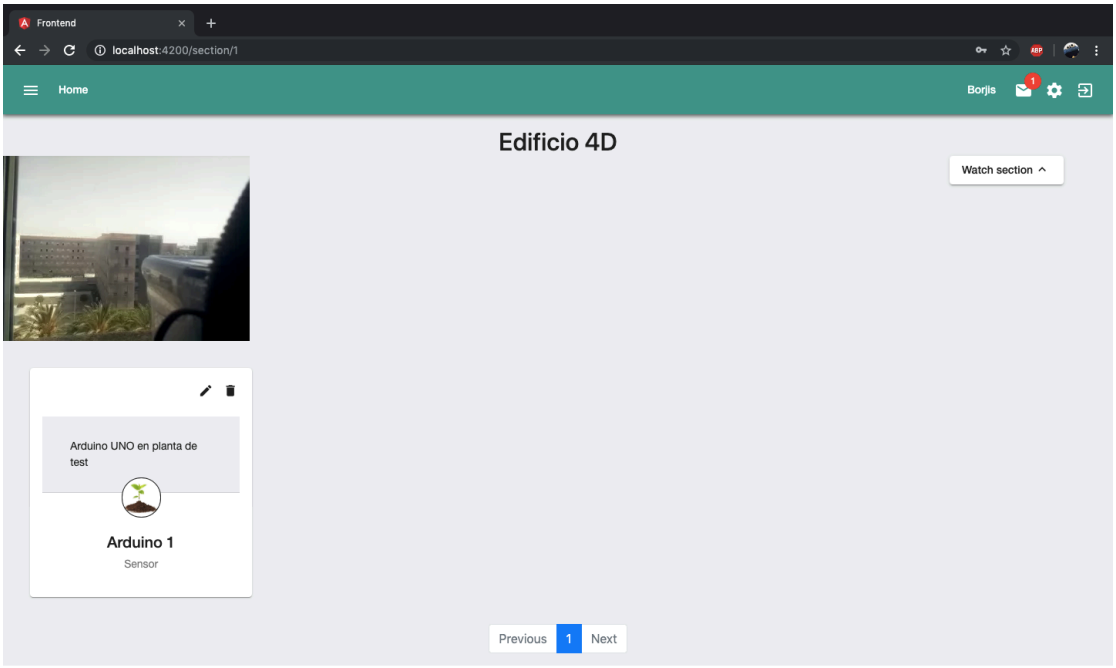


Figura 4.17 Vista Sensors de la sección “Edificio 4D”.

Arriba, en la figura 4.17, se puede ver la vista *Sensors*. En ella aparece un dispositivo final perteneciente a la sección “Edificio 4D” como dice el pie de foto. Justo encima de ella se puede ver el vídeo en directo de la cámara conectada a la sección “Edificio 4D”, en este caso no está en el lugar que debería (la UPV) está en la habitación donde se ha desarrollado gran parte de este Trabajo de Fin de Grado, enfocando a través de la ventana. En esta vista, a diferencia de las anteriores, se mostrarán doce dispositivos finales paginados y al hacer click en las tarjetas de los sensores, se accederá a los datos de las medidas del sensor en cuestión. Esta última vista dentro de este grupo de vistas será *Measures* (véase figura 4.18).

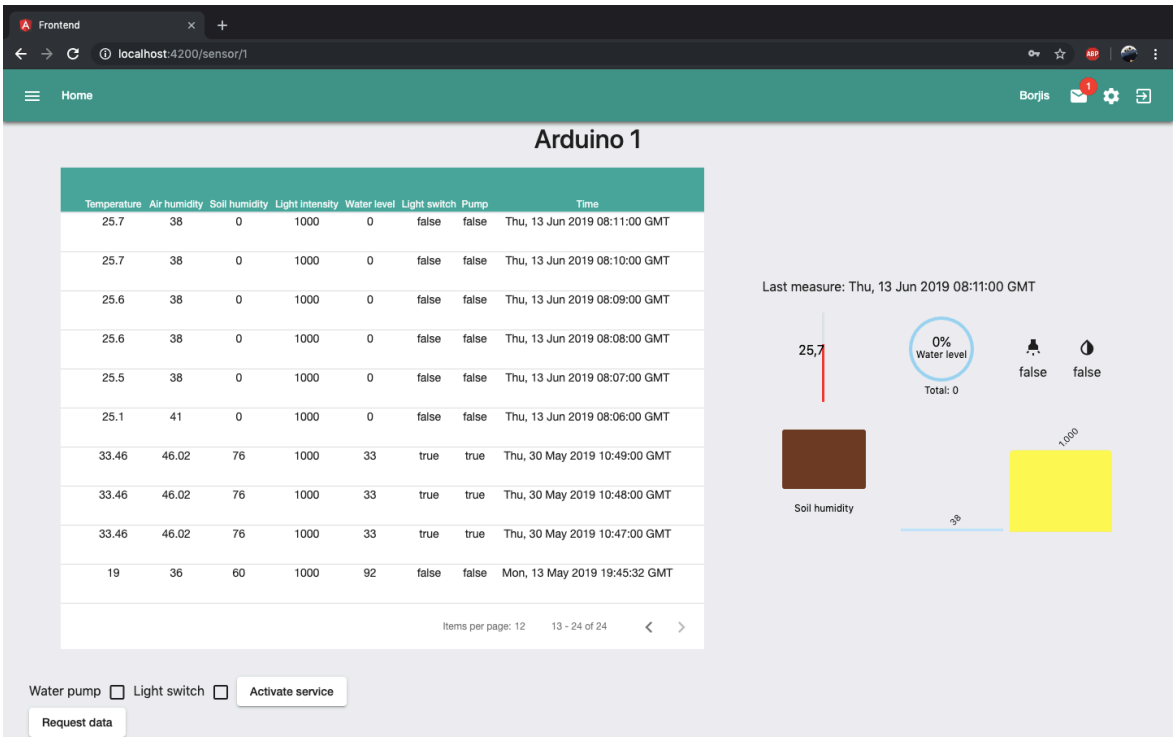


Figura 4.18 Vista Measures del dispositivo final “Arduino 1”.

En la figura de arriba, se muestra la vista *Measures*. En ella aparece una tabla con las últimas medidas tomadas por el dispositivo final y gráficos a la derecha de la última medida guardada.

En la parte baja de la pantalla aparece un cuadro de mandos donde se puede hacer uso de los servicios que proporciona ese dispositivo final. En este caso se pueden activar mecanismos como el bombeo de agua o la luz artificial, marcando las casillas del servicio a activar y apretando el botón *Activate service*. El botón mostrado abajo con el título *Request data*, permite pedir una medida en tiempo real de todos los sensores conectados al dispositivo final, para así, no tener que esperar a la siguiente medida tomada y registrada en la base de datos. Esta medida es mostrada en las líneas a continuación tras ser obtenida.

Además de estas funciones principales, se han creado diferentes vistas para añadir más funciones a la aplicación con vistas a mejorar poco a poco todo lo ofrecido por ésta. Vistas como *Maps* o *Analytics* serán comentadas a continuación.

La vista *Maps* (véase figura 4.19) proporciona un mapa creado con Google Maps para que el usuario pueda ubicar la posición de sus huertos. Esta vista en el futuro podría evolucionar en una visión conjunta de todos los huertos de cada uno de los usuarios, con vistas a que otros usuarios se relacionen y visiten los demás huertos, a modo de red social.

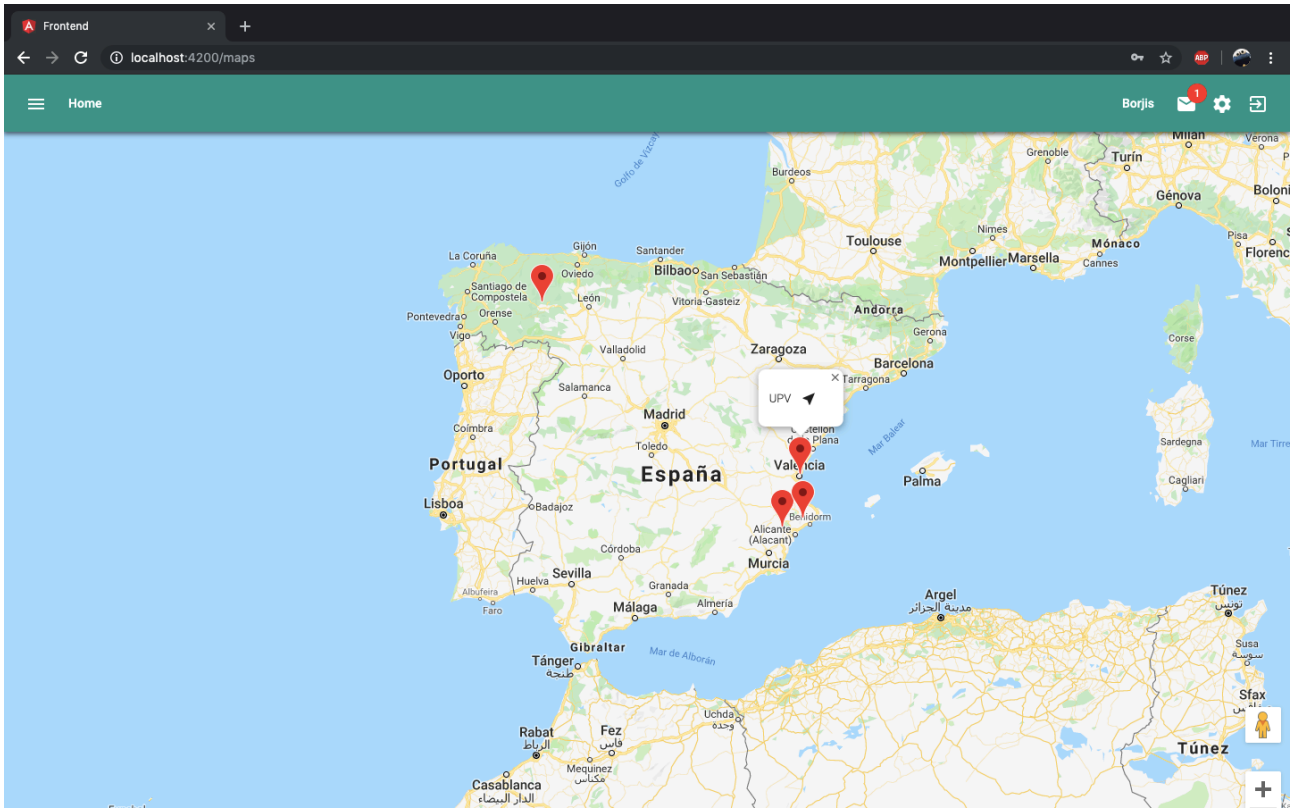


Figura 4.19 Vista Maps con los huertos de un usuario.

En esta figura se pueden observar los mismos cuatro huertos que en la figura 4.15, esta vez posicionados geográficamente. Al hacer click en cada uno de ellos, mostrarán el nombre del huerto en cuestión y haciendo click en el icono de la flecha, se navegará a la vista *Sections* de ese huerto.

Por otra parte, la vista *Analytics* trata de proporcionar una visión más clara de la evolución de cada cultivo mediante la muestra de las últimas treinta medidas tomadas por cada uno de los dispositivos finales del usuario.

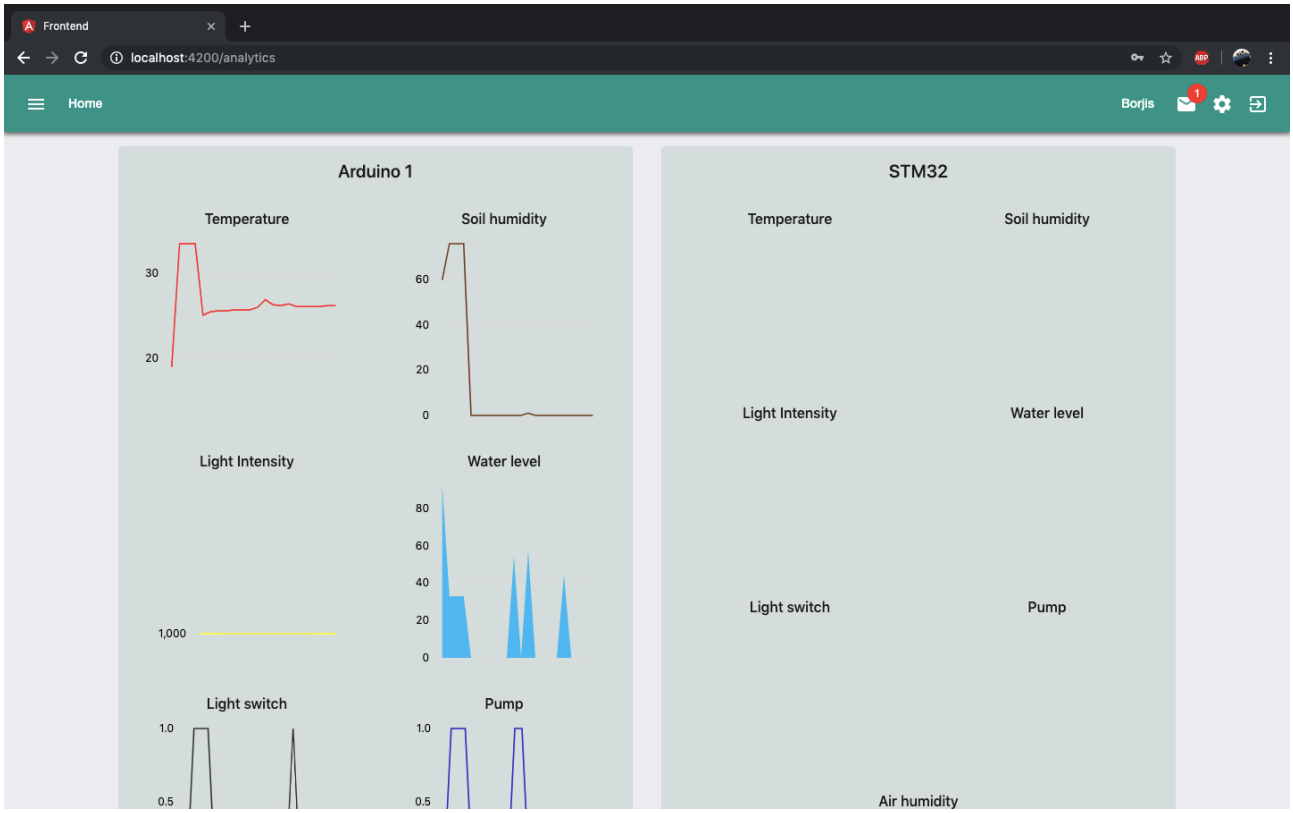


Figura 4.20 Vista Analytics con los dispositivos finales de un usuario.

En la figura 4.20 se puede ver la vista *Analytics*, en este caso compuesta por dos dispositivos finales del usuario autenticado, los dispositivos finales son “Arduino 1” y “STM32”, aunque el único utilizado ha sido “Arduino 1”, por eso aparecen en el las gráficas de las últimas treinta medidas.

Las gráficas utilizadas se han dispuesto de manera que se puedan comparar fácilmente y que se pueda ver si los resultados son los esperados. La primera gráfica es la de temperatura, seguidas de las gráficas bastante relacionadas como son la gráfica de la humedad del suelo y la gráfica de la intensidad de la luz. En este ejemplo las medidas de intensidad de luz son constantes porque se añadieron a mano. La intención es ver la relación entre el nivel de luz y la temperatura que experimentan las plantas, así como la temperatura a la que están las plantas y la evolución de la humedad del suelo. Por otra parte se puede ver la relación entre la intensidad de la luz y la evolución temporal de la activación de la luz artificial, la diferencia de estas dos gráficas nos daría la cantidad de luz que las plantas reciben directamente del sol. Junto a la humedad del suelo se han colocado las gráficas del nivel de agua en el tanque principal y la activación de la bomba, comparando estas gráficas se podría sacar cuanta agua se ha gastado cada vez que se activa la bomba y cada cuanto tiempo hemos de regar las plantas para mantener cierto nivel de humedad. Por último y aunque no se ve en este ejemplo, se puede ver una gráfica con las medidas de humedad del aire.

Además del desarrollo de todas las vistas, se desarrollaron también servicios para la gestión de las peticiones HTTP y el manejo de WebSocket. El servicio API es el encargado de gestionar todas las peticiones con la API REST diseñada con Flask. Desde ese servicio se definen todos los métodos que se utilizan en cada una de las vistas para obtener los datos. Por otro lado el servicio encargado de gestionar la conexión WebSocket tiene definidos diferentes eventos de SocketIO en los que la aplicación avisa al usuario y este lo muestra como una alerta en la aplicación o como un mensaje. También gestiona la señalización realizada con WebRTC. Junto con los servicios y los componentes se implementa un sistema de routing interno en la aplicación para navegar por las distintas vistas que esta tiene.

Otro aspecto importante del routing de la aplicación es la implementación de los interceptores y las guardas. Mediante el uso de guardas se previene que usuarios no autenticados en la aplicación, es decir, que no han accedido con su usuario, puedan acceder a vistas con autenticación requerida. Cuando una guarda ve que el usuario que intenta acceder no está autenticado, redirige al usuario a la vista *Login*. Mediante el uso de interceptores se facilita el uso de la inclusión del token recibido tras ser autenticado en cada una de las peticiones y mediante otro interceptor distinto se implementa el sistema de "desloggeo", esto es, que tras recibir el aviso de que el token está caducado por parte del servidor, la aplicación pida una nueva autenticación al usuario.

4.6 Diagramas de flujo

Este apartado muestra los diferentes diagramas de flujo de cada uno de los programas desarrollados en el Capítulo 4: Desarrollo. La intención es facilitar la comprensión al lector y visualizar de una manera más esquemática lo comentado anteriormente.

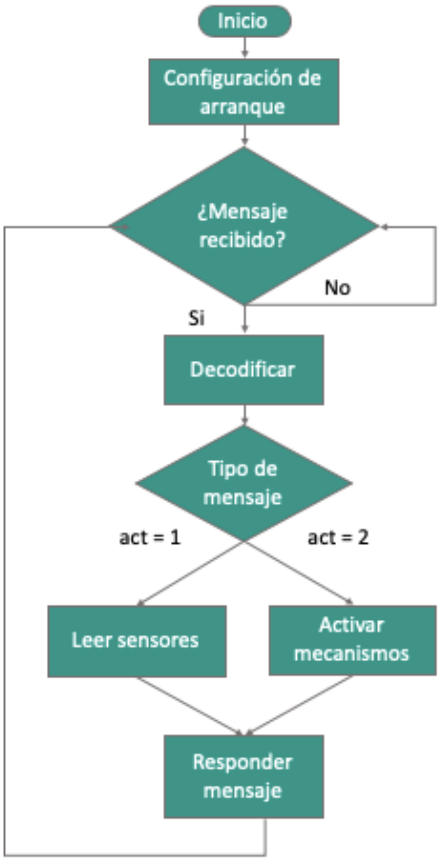


Figura 4.21 Diagrama de flujo dispositivo final.

El diagrama de la figura 4.21 muestra el funcionamiento del programa ejecutado en Arduino UNO. La configuración de arranque es el método *setup( )* mientras que el resto del diagrama es el interior del método *loop( )*. En el se encuentran métodos para envío de mensajes, codificación y decodificación de mensajes, un método para cada uno de los sensores conectados y otros métodos de utilidad para el desarrollo.

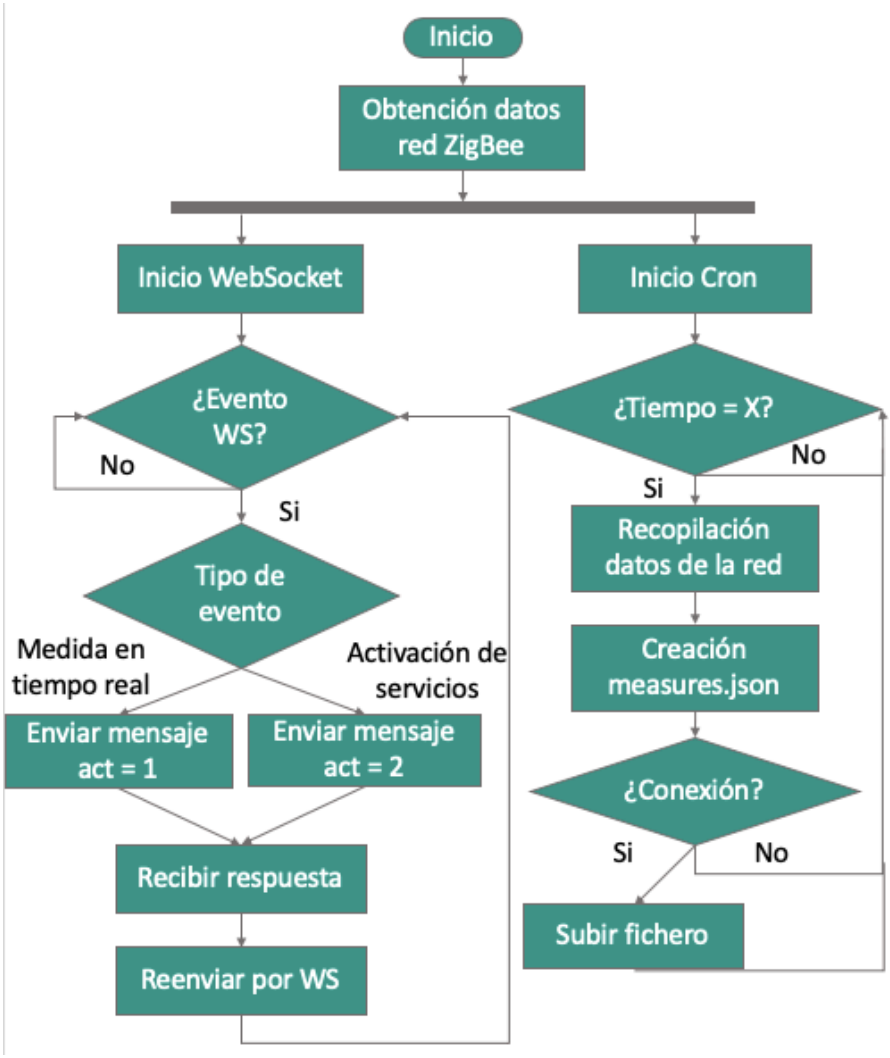


Figura 4.22 Primer diagrama de flujo controlador de sección.

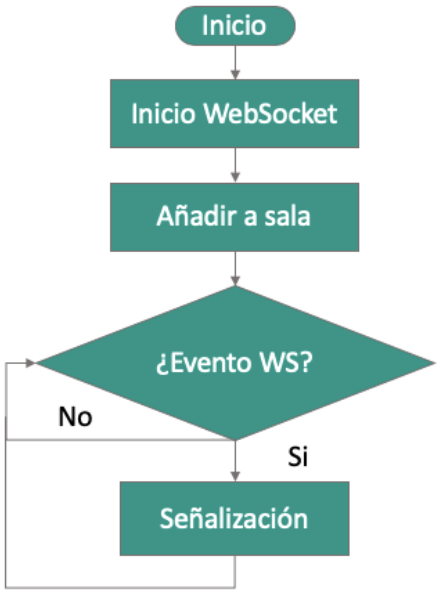


Figura 4.23 Segundo diagrama de flujo controlador de sección.

Los diagramas de flujo de las figuras 4.22 y 4.23 se han separado debido a que, aunque coexistan en el mismo dispositivo Raspberry pi, estos programas se ejecutan por separado. En el primer diagrama de flujo (véase figura 4.22) se ha definido la ejecución paralela de dos procesos mediante una línea de mayor grosos que acto seguido muestra el desarrollo de cada uno de estos procesos. Estos procesos son dos threads diferentes que ocurren en el mismo programa, uno para WebSocket y otro para la tarea tipo Cron. Este diagrama de flujo pertenece al programa en Python 3. En el cuadro de decisión que aparece *¿Tiempo = X?* se refiere a la periodización con la que los datos de la red son recopilados.

Por otra parte, el programa desarrollado con Angular se muestra en la figura 4.23. Este programa se encarga de señalizar e iniciar WebRTC en Raspberry pi, cuando la señalización termina, vuelve a esperar más conexiones.

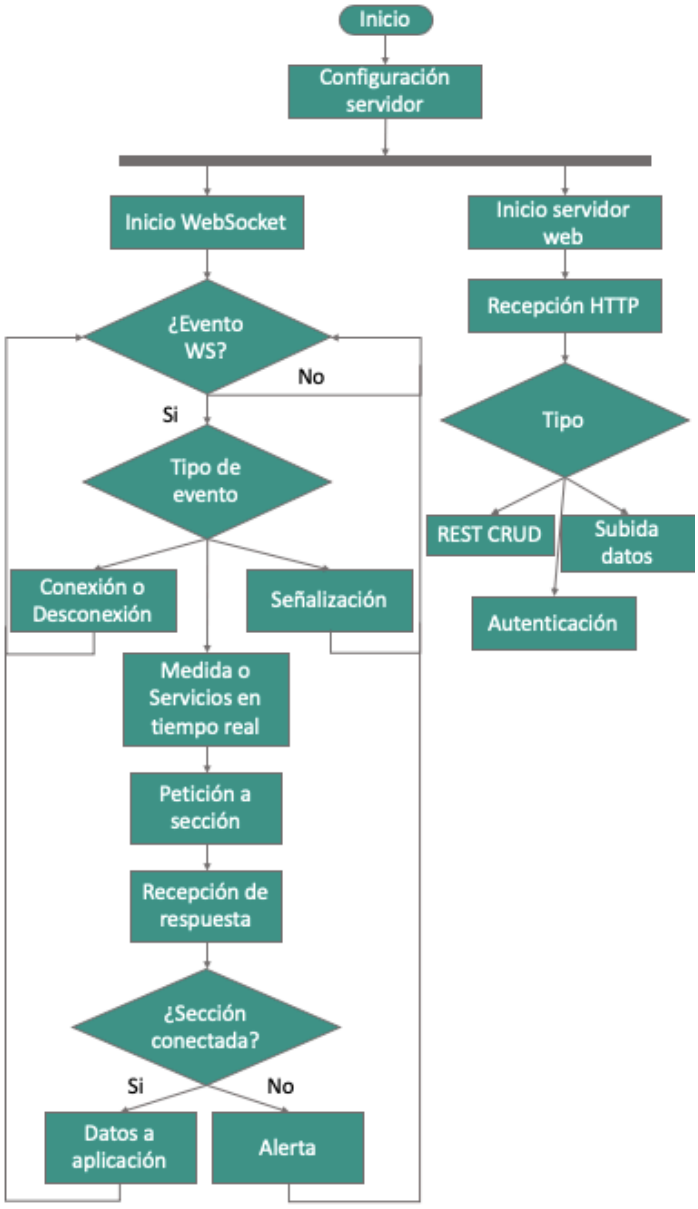


Figura 4.24 Diagrama de flujo servidor central.

El último diagrama de flujo es el del servidor central, mostrado en la figura 4.24. Se ha dividido la ejecución en dos threads, uno encargado de la gestión del del WebSocket, con sus conexiones, medidas en tiempo real y la señalización. Y por otro lado el servidor web encargado de proporcionar todos los datos y las rutas HTTP para el funcionamiento de la API REST.

Capítulo 5: Conclusiones

5.1 Conclusiones

En el presente Trabajo de Fin de Grado se ha desarrollado un sistema de monitorización y automatización de cultivos mediante redes de sensores utilizando el protocolo ZigBee. Una aplicación web desarrollada en Angular 7 permite acceder a los usuarios y gestionar sus huertos y los dispositivos conectados a ellos, haciendo uso de una API REST desarrollada con Flask en Python 3.

El objetivo del proyecto, el cuidado del cultivo ecológico, se obtiene motivando y facilitando el cuidado del huerto para los usuarios aficionados y también mejorando la gestión de recursos naturales y el cuidado de la producción para profesionales. El proyecto, diseñado para ser escalable permite hacer uso de la infraestructura a usuarios no profesionales y a empresas dedicadas a la agricultura. Los dispositivos conectados permiten automatizar y monitorizar el cuidado del huerto, así como vigilar el estado del huerto en tiempo real.

Un dispositivo final desarrollado con Arduino UNO permite automatizar la activación de mecanismos de iluminación y riego para cada huerto con el uso de relés. Con el uso de sensores, este dispositivo toma medidas de temperatura, humedad del aire, humedad del suelo, iluminación y nivel del agua en el tanque de riego. Gracias a la gestión de una Raspberry pi 3 model B conectada a Internet, el sistema es capaz de recopilar la información de cada huerto y enviarla al servidor central, para posteriormente ser mostradas por la aplicación en gráficas. Además de lo anterior se utiliza WebRTC para recibir el vídeo en directo de la cámara instalada en Raspberry pi en la aplicación.

El objetivo principal del proyecto es conseguido gracias a la implementación de la infraestructura IoT que facilita, junto con el uso de los dispositivos conectados, la gestión de los cultivos.

5.2 Líneas futuras de trabajo

El estado actual del proyecto permite la definición de nuevas líneas futuras de trabajo, algunas de ellas se plantean a continuación:

- Creación de una infraestructura con más de 100 dispositivos.
- Añadir más mecanismos de seguridad al sistema (HTTPS, encriptación en XBee…)
- Integración del circuito prototipo en una PCB y alimentación vía energía solar.
- Selección de nuevos sensores y mejora en la explotación de los datos.
- Creación de una cubierta hermética con materiales reciclados para los dispositivos.
- Mejora de la aplicación central añadiendo nuevas funcionalidades como comunicación con otros usuarios de la plataforma.
- Experimentación con la librería en C de WebRTC e implantación en un único código.

5.3 Presupuesto

A continuación se detalla el presupuesto necesario para replicar el proyecto, así como la posible vía de financiación del proyecto.

Concepto	Importe
Arduino UNO	20 €
Breadboard y cableado	5 €
DHT11	0,76 €
FC28	0,80 €
HW38	2 €
LDR	0,4 €
2 x Relé	1 €
Bomba de agua	15 €
XBee	20 €
Total	**64,96 €**

Tabla 5.1 Costes del dispositivo final.

Concepto	Importe
Raspberry pi 3 model B	35 €
Cámara raspberry V2	25 €
XBee explorer	20 €
XBee	20 €
Total	**100 €**

Tabla 5.2 Costes del controlador de sección.

Estos son los costes por cada uno de los dispositivos. La posible financiación del proyecto vendría ofreciendo una infraestructura por un pago reducido a los usuarios aficionados, de tal forma que ellos solo tuvieran que adquirir y montar los dispositivos a conectar a la red con una guía publica para hacer uso del proyecto. Y para las empresas, dado que la infraestructura se tendría que ofrecer para su gestión privada, se vendería el software para utilizar el producto y los dispositivos listos para ser utilizados.

Bibliografía

[1] That 'Internet of Things' Thing.
Disponible en: https://www.rfidjournal.com/articles/view?4986

[2] The Internet of Things: How the Next Evolution of the Internet Is Changing Everything.
Disponible en: https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

[3] The Internet of Fewer Things.
Disponible en: https://spectrum.ieee.org/telecom/internet/the-internet-of-fewer-things

[4] Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016.
Disponible en: https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016

[5] Organic farming statistics.
Disponible en: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Organic_farming_statistics#Total_organic_area

[6] PIB agricultura en España.
Disponible en: https://datos.bancomundial.org/indicador/nv.agr.totl.zs?name_desc=false

[7] Manual del huerto ecológico. Autor: Mariano Bueno.

[8] Proyecto Garduino.
Disponible en: https://www.instructables.com/id/Garduino-Gardening-Arduino/

[9] Capas del protocolo ZigBee.
Disponible en: https://www.digi.com/resources/documentation/Digidocs/90002002/Content/Reference/r_zb_stack.htm?TocPath=zigbee%20networks%7C_____3

[10] Angular framework homepage.
Disponible en: https://angular.io/

[11] Software XCTU.
Disponible en: https://www.digi.com/products/iot-platform/xctu

[12] API REST Tutorial de Miguel Grinberg.
Disponible en: https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask

[13] WebRTC project homepage.
Disponible en: https://webrtc.org/

[14] Cron y crontab explicados.
Disponible en: https://blog.desdelinux.net/cron-crontab-explicados/

[15] Flask microframework homepage.
Disponible en: http://flask.pocoo.org/

[16] Eventlet networking library homepage.
Disponible en: https://eventlet.net/

[17] Comparison of transparent and API modes.
Disponible en: https://www.digi.com/resources/documentation/Digidocs/90001942-13/concepts/c_xbee_comparing_at_api_modes.htm?TocPath=How%20XBee%20devices%20work%7CSerial%20communication%7C_____2

[18] Wireless mesh networking: ZigBee vs DigiMesh.
Disponible en: https://www.digi.com/pdf/wp_zigbeevsdigimesh.pdf

[19] Flask-SocketIO homepage:
Disponible en: https://flask-socketio.readthedocs.io/en/latest/

[20] SQLAlchemy toolkit homepage:
Disponible en: https://www.sqlalchemy.org/

[21] Design a Luxometer Using a Light Dependent Resistor.
Disponible en: https://www.allaboutcircuits.com/projects/design-a-luxmeter-using-a-light-dependent-resistor/

[22] XBee-Arduino library.
Disponible en: https://github.com/andrewrapp/xbee-arduino

[23] ArduinoJSON library.
Disponible en: https://arduinojson.org/

[24] Python-XBee library.
Disponible en: https://github.com/digidotcom/python-xbee

[25] Git homepage.
Disponible en: https://git-scm.com/

[26] Github homepage.
Disponible en: https://github.com/

[27] Flask-Migrate tool.
Disponible en: https://flask-migrate.readthedocs.io/en/latest/

[28] Postman homepage.
Disponible en: https://www.getpostman.com/

[29] Restful API with Python and Flask.
Disponible en: https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask

[30] Google codelab for WebRTC.
Disponible en: https://codelabs.developers.google.com/codelabs/webrtc-web/#0

[31] Janus server homepage.
Disponible en: https://janus.conf.meetecho.com/

Apéndice

Glosario

Adafruit
Compañía de hardware de código abierto.

C
Lenguaje de programación de alto nivel con posibilidad de programación a bajo nivel.

C++
Lenguaje de programación que extiende C con programación orientada a objetos.

CSMA/CA
Carrier sense multiple access with collision avoidance, protocolo de control de acceso al medio para redes inalámbricas.

CSS
Cascading style sheets, lenguaje de diseño gráfico utilizado en conjunto con HTML para dar diseño a sitios web, además de otras funciones.

Cisco IBSG
Cisco Internet Business Solutions Group, grupo de expertos del sector tecnológico que realiza análisis sobre los avances en la tecnología.

Cliente/servidor
Arquitectura de software en la que los servidores son los proveedores de recursos mientras que los clientes son los que los consumen.

Comunicación push
Tecnología que permite que el servidor envíe notificaciones a los clientes sin necesidad de una petición previa.

Cookie
Información que guardan cliente y servidor para mantener el estado entre peticiones HTTP.

Cron
Administrador temporizado de procesos en segundo plano para UNIX.

DIY
Do It Yourself, término que describe un movimiento basado en hacer las cosas uno mismo.

Debian
Proyecto y comunidad que desarrolla un sistema operativo GNU basado en software libre.

Dirección IP
Identificador utilizado por el protocolo IP. Las direcciones IP pueden ser públicas o privadas. Las privadas se utilizan para identificar a máquinas en redes internas, como puede ser detrás de un NAT (véase NAT).

Eventlet
Librería para Python para la gestión de operaciones concurrentes.

Firewall
: Programa informático de seguridad encargado de gestionar las conexiones de la computadora y los distintos elementos de la red.

Framework
: Esquema para el desarrollo de una aplicación. Suelen ser librerías con módulos para facilitar el proceso de desarrollo.

GNU/Linux
: Sistema operativo de software libre de tipo UNIX.

HAT
: *Hardware Attached on Top*, son módulos hardware para añadir en Raspberry pi.

HTML
: *HyperText Markup Language*, es un lenguaje de marcado para el desarrollo de páginas web.

HTTP
: *HyperText Transfer Protocol*, es el protocolo que permite la transferencia de información por la web.

IDE
: *Integrated Development Environment*, entorno de desarrollo integrado es un programa informático que facilita el desarrollo de aplicaciones.

IEEE
: *Institute of Electrical and Electronics Engineers*, asociación mundial de ingenieros encargada a la estandarización y el desarrollo en áreas técnicas.

IoT
: *Internet of Things*, es la interconexión de dispositivos a través de una red. Dispositivos como sensores o elementos cotidianos como un interruptor.

JavaScript
: Lenguaje de programación interpretado, desarrollado a partir del estándar ECMAScript. Utilizado mayormente para el desarrollo web.

LDR
: *Light Dependent Resistor*, es un elemento de circuitos electrónicos que varía su resistencia en función de la luz que recibe.

LPWAN
: *Low-Power Wide-Area Network*, es un tipo de red inalámbrica basada en WAN (véase WWAN) cuyo propósito es permitir las comunicaciones entre dispositivos de baja tasa de envío de datos y bajo consumo.

Lock
: En Python 3, un *Lock* es un objeto que permite bloquear más de un acceso simultáneo a un método. Mediante el uso de este objeto, se puede bloquear y desbloquear el acceso a un método hasta que la ejecución de este finalice. Es útil en programación con threads.

Many to many
: En un contexto de base de datos relacional, *many to many* o muchos a muchos, es un tipo de relación entre tablas que permite que una instancia de la tabla A tenga esté relacionada con muchas instancias de la tabla B y viceversa.

NAT

Network Access Translation, es un mecanismo utilizado en redes IP para utilizar la mismo dirección IP en redes distintas (lo que se suele llamar IP privada) y que no haya problemas de compatibilidad. Se utiliza para prevenir el problema de que las direcciones IP versión 4 es un recurso limitado.

One to many

En un contexto de base de datos relacional, *one to many* o uno a muchos, es un tipo de relación entre tablas en las que una instancia de la tabla A tiene relación con múltiples instancias de la tabla B y no al contrario.

One to one

En un contexto de base de datos relacional, *one to one* o uno a uno, es un tipo de relación entre una única instancia de la tabla A y una única instancia de la tabla B.

Parsear

Proviene de *parser* y básicamente quiere decir analizar un texto en busca de patrones.

Peer to peer

También conocida como *P2P*, es un tipo de conexión que permite la conexión directa entre máquinas, sin necesidad de un servidor como en la arquitectura cliente-servidor.

PEP

Python Enhancement Proposal, se trata de una comunidad encargada de gestionar las propuestas de mejoras para el lenguaje de programación Python.

Python 3

Lenguaje de programación interpretado en su versión 3, no compatible con su versión 2. Cuyo desarrollo puso énfasis en una sintaxis legible.

Raspbian

Sistema operativo GNU/Linux basado en Debian utilizado en Raspberry pi.

SDP

Session Description Protocol, es un protocolo para describir los parámetros de inicio de una sesión multimedia.

STUN

Session Traversal Utilities for NAT, se trata de un servidor encargado de devolver a los clientes detrás de un router NAT su dirección IP pública.

Shield

Se trata de una extensión del hardware de la placa, en este caso para Arduino. Equivalente a HAT (véase HAT) para Raspberry pi.

String

Se trata de un tipo de objeto incorporado en varios lenguajes de programación capaz de almacenar texto.

TCP

Transmission Control Protocol, se trata de un protocolo a nivel de transporte encargado de gestionar la correcta transmisión de los paquetes enviados por Internet. Es un elemento clave en Internet tal y como lo conocemos hoy en día.

TURN
: *Traversal Using Relays around NAT*, es un tipo de servidor encargado de retransmitir el tráfico multimedia cuando se utiliza WebRTC y la comunicación peer to peer (véase peer to peer) falla.

Thread
: Se trata de una de las ejecuciones simultáneas de un programa.

UDP
: *User Datagram Protocol*, es un protocolo a nivel de transporte que permite el envío de paquetes a través de Internet pero sin garantía de recepción.

UNIX
: Sistema operativo desarrollado en los laboratorios Bell del que surgen varias familias de sistemas operativos.

WLAN
: *Wireless Local Area Network*, es un tipo de red inalámbrica de alcance semi-corto, utilizada sobre todo en entornos de casas o edificios.

WMAN
: *Wireless Metropolitan Area Network*, es un tipo de red inalámbrica con alcance muy largo, cuya cobertura alcanza ciudades o pueblos completos.

WPAN
: *Wireless Personal Area Network*, es un tipo de red inalámbrica con alcance muy corto, utilizada por dispositivos conectados a muy poca distancia.

WSGI
: *Web Server Gateway Interface*, es un esquema que define la interacción entre servidores web para reenviar peticiones a aplicaciones web, definido por PEP (véase PEP).

WWAN
: *Wireless Wide-Area Network*, es un tipo de red inalámbrica con alcance largo cuyo alcance cubre varias redes WLAN.

WebRTC
: Proyecto de software libre desarrollado por Google cuyo objetivo es facilitar las comunicaciones peer to peer, tanto para envío de datos multimedia como para envío de mensajes.

XBee
: Tecnología propietaria de la empresa Digi tanto software como hardware para la interconexión de redes inalámbricas de baja tasa de transmisión.

ZigBee
: Conjunto de protocolos para la transmisión inalámbrica de bajo consumo.

Código

Todo el código desarrollado en el proyecto se puede consultar en Github.

Disponible en: https://github.com/Borjis131/TFG