UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

CREACIÓN DE SUPERFICIES DIGITALES CONTINÚAS BAJO DEMANDA A PARTIR DE DATOS PROVENIENTES DE SENSORES IOT.

TRABAJO FINAL DEL

Grado en Ingeniería Electrónica Industrial y Automática

REALIZADO POR

Alba Jiménez Coll

TUTORIZADO POR

Hassan Mohamed, Houcine

Martínez Llario, José Carlos

CURSO ACADÉMICO: 2019/2020

Agradecimientos.

A mis tutores por ayudarme y guiarme en este proyecto lleno de aprendizaje y sobre todo por el apoyo mostrado durante toda la investigación.

Así como a mis profesores de mi intercambio en Uruguay que me escucharon y aconsejaron durante toda mi estancia en la universidad ORT.

Finalmente agradecer a mis padres, que me han apoyado y han tenido mucha paciencia conmigo durante todo el proceso y estar a mi lado siempre que lo he necesitado.

Resumen

El objetivo es crear un sistema de visualización en tiempo real que muestre de forma continua e interoperable en formato imagen datos discretos (ej. datos de temperatura, calidad del aire, etc.), provenientes de sensores Arduino para facilitar la toma de decisiones y poder compartir y distribuir sobre internet dicha información de forma sencilla y compatible. Las tareas que se van a realizar son las siguientes:

- Búsqueda de un microcontrolador de bajo consumo, con conexión Wi-Fi, a base de baterías que capture datos discretos de ejemplo.

- Programación de dicho dispositivo para la captura y envío de peticiones HTTP de los datos observados para su adecuado registro.

- Configuración de una capa intermedia en el servidor utilizando el lenguaje de programación Python y librerías Flask, Django o similares para la captura de las peticiones HTTP y su traducción a lenguaje SQL Espacial (entidades de tipo puntual según el estándar ISO SQL/MM 13249-3) de los datos recibidos.

- Almacenamiento en un sistema relacional PostgreSQL y su extensión espacial PostGIS, previo diseño de un modelo de datos relacional adecuado. Todos los datos registrados de los sensores quedarán almacenados en este modelo de datos en función de la localización, el tiempo de captura y el dato. El modelo constará de los índices unidimensionales y bidimensionales (espaciales) necesarios para su máxima eficiencia.

- Configuración de un servidor de servlets (ej: tomcat) y un servidor especializado (Geoserver) que utilice el protocolo WMS que sea capaz de:

    - Calcular (programación mediante lenguaje SLD) las imágenes continúas a partir de los datos discretos de los sensores mediante un proceso de interpolación matemática (p. ej.: pesos inversos a la distancia).
    - Distribución de la información obtenida mediante el protocolo estándar. Para lograr la interoperabilidad se utilizará el protocolo WMS (ISO 19152) que asegura el compartir información georreferenciada continua de forma estándar entre el software para la toma de decisiones.

- Visualizar la información obtenida vía WMS con varios softwares especializados para la toma de decisiones como los SIG.

Palabras clave: Smart City, Arduino, Phyton, Geoserver, Postg

Resum

L'objectiu es crear un sistema de visualització en temps real que mostre de forma continua i interoperable, en format imatge, dades discretes (ex. dades de temperatura, calitat de l'aire, etc.). provinents de sensors Arduino per facilitar la presa de decisions i poder compartir i distribuir sobre internet aquesta informació de forma fàcil i compatible.
Les tasqyes que es van a realitzar son les següents:

- Recerca de un microcontrolador de baix consum, amb connexió Wi-Fi i a base de bateries que capturen dades discretes d'exemple.

- Programació d'aquest dispositiu per la captura, data i enviament de peticions HTTP de les dades observades per un correcte registre.

- Configuració d'una capa intermitja en el servidor utilitzant un llenguatge de programació Phyton i llibreries flask, django o similar per a la captura de les peticions HTTP i la seua traducció a llenguatge SQL Espacial (entitats de tipus puntual segons l'estàndard ISO SQL/MM 13249-3) de les dades rebudes.

- L'almaçenatge en un sistema relacional PostgreSQL i la seua extensió espacial PostGIS, previ disseny de un model de dades relacional adequat.Totes les dades registrades del sensors quedaran almacenades en aquest model de dades en funció de la localització, el temps de captura i les dades. El model constarà dels índex unidimensionals i bidimensionals (espacials) necessaris per a la seua màxima eficiència.

- Configuració d'un servidor servlets (ex. tomcat) i un servidor especialitzat (Geoserver) que utilitze el protocol WMS que siga capaç de:

    - Calcular (programació mitjançant llenguatge SLD) les imatges continues a partir de les dades discretes dels sensors mitjançant un procés de interpolació matemàtica (ex: pesos inversos a la distancia)
    - Distribució de la informació obtinguda mitjançant el protocol estàndard. Per aconseguir la interoperabilitat s'utilitzarà el protocol WMS (ISO 19152) que assegura el compartir informació georreferenciada continua de forma estàndard entre els software per la presa de decisions.

- Visualització la informació obtinguda via WMS amb diversos softwares especialitzats per la presa de decisions com els SIG.

Paraules clau: Smart City, Arduino, Phyton, GeoServer, PostgreSQL

Abstract

The objective is to create a real-time display system that displays discrete data continuously and interoperable in image format (eg temperature data, air quality, etc.). from Arduino sensors to facilitate decision making and to be able to share and distribute this information on the internet in a simple and compatible way. The task to be performed are the following:

- Search for a low-power microcontroller, Wi-Fi connection and a battery base that capture discrete sample data.

- Programming the device for capturing, dating and sending HTTP requests of the observed data to register it correctly.

- Configuration of an intermediate layer on the server using the Python programming language flask, django or similar libraries for capturing HTTP requests and their translation into Spatial SQL language (point type entities according to ISO SQL / MM 13249 standard -3) of the data received

- Storage in a PostgreSQL relational system and its PostGIS spatial extension, prior design of an appropriate relational data model. All recorded sensor data will be stored in this data model based on location, capture time and data. The constant model of the one-dimensional and two-dimensional (spatial) indexes necessary for maximum efficiency.

- Configuration of a servlet server (e.g., tomcat) and a specialized server (Geoserver) that uses the WMS protocol that is capable of:

    - Calculate (programming by means of SLD language) the continuous images from the discrete data of the sensors through a process of mathematical interpolation (eg: inverse weights at a distance).
    - Distribution of the information obtained through the standard protocol. To achieve interoperability, the WMS protocol (ISO 19152) will be used, which ensures continuous geo-referenced shared information as standard among decision-making software.

- Visualize the information obtained via WMS with several specialized software for decision making such as GIS.

Key words: Smart City, Arduino, Phyton, Geoserver, PostgreSQL

Índice

Índice de figuras

Listado de abreviaturas

ACID – Atomicity Consistency Isolation and Durability
CAD – Computer-aided design
CNIG - Centro Nacional de Información Geográfica
CRS - Sistema de Referencia de Coordenadas (Coordinate Reference System)
CSG - Consejo Superior Geográfico
EPSG - European Petroleum Survey Group
GTIDEE - Grupo de Trabajo IDEE
HTML - Lenguaje de Marcas Hipertexto (Hipertext Markup Language)
HTTP - Protocolo de Transferencia de Hipertexto (Hypertext Transfer Protocol)
JSON - JavaScript Object Notation
IDE- Infraestructura de Datos Espaciales
IEC – International Electrotechnical Commission
IOT - Internet of Things
IG - Información Geográfica
IGN – Instituto Geográfico Nacional
ISO - Organización Internacional de Estandarización (International Organization for Standardization)
IVA - Impuesto del Valor Añadido
OGC - Open Geospatial Consortium
OSGeo - Open Source Geospatial Foundation
OSM - OpenStreetMap
PNOA - Plan Nacional de Ortofotografía Aérea
SIG- Sistema de Información Geográfica
SGBD- Sistema de Gestión de Base de Datos
SLD - Styled Layer Descriptor
SQL - Structured Query Language
URL - Uniform Resource Locators
WMS - Web Map Service
WFS - Servicio Web de Fenómenos (Web Feature Service)
WI-FI - Wireless Fidelity

1. Introducción

1.1 Motivación

En el año 2007, las Naciones Unidas hicieron un estudio de los que estaba pasando en la población urbana, que sobrepasaba ya en ese momento a la población rural en el mundo. Se realizaban estudios de previsión y se predijo que en el año 2050 el 70% de la población mundial iba a ser urbana y con más de diez millones de habitantes en muchas de las ciudades. [1]
Las ciudades, son plataformas que presentan un gran **impacto en el desarrollo económico** y social de las naciones. En ellas viven y trabajan los ciudadanos, desarrollan sus actividades las empresas y son grandes centros de consumo de recursos. En la actualidad, las ciudades consumen el 75% de los recursos y de la energía mundial y generan el 80% de los gases responsables del efecto invernadero, ocupando tan sólo el 2% del territorio mundial. [1]

Existen muchas definiciones de *Smart City* (en castellano **Ciudad Inteligente**).
“Una Ciudad Inteligente y Sostenible es una ciudad innovadora que aprovecha las Tecnologías de la Información y la Comunicación (TIC) y otros medios para mejorar la calidad de vida, la competitividad, la eficiencia del funcionamiento y los servicios urbanos, al tiempo que se asegura de que responde a las necesidades de las generaciones presente y futuras en lo que respecta a los aspectos económicos, sociales, medioambientales y culturales.” UNE 178201:2016
Según el EIP-SC European Innovation Partnership on Smart Cities and Communities, las Ciudades y Comunidades Inteligentes se pueden definir de la siguiente manera: “Las Ciudades Inteligentes deben ser consideradas sistemas de personas que interactúan y usan flujos de energía, materiales, servicios y financiación para catalizar el desarrollo económico sostenible, la resiliencia, y una alta calidad de vida; estos flujos e interacciones se hacen “inteligentes” mediante el uso estratégico de infraestructuras y servicios de TICs en un proceso de planificación urbana y gestión transparentes que responda a las necesidades sociales y económicas de la sociedad” o según https://economipedia.com/definiciones/ciudad-inteligente-smart-city.html lo define como : “aquellas que utilizan el potencial de la tecnología y la innovación, junto al resto de recursos para hacer de ellos un uso más eficaz, promover un desarrollo sostenible y, en definitiva, mejorar la calidad de vida de sus ciudadanos.·

De todas las definiciones se puede comprobar que la aplicación de las TIC se hace imprescindible y creando ciudades inteligentes se mejora la calidad de vida de los ciudadanos. [2]

El uso de las smart cities presenta las siguientes características:

- **Incremento de la eficiencia y la calidad de los servicios.**
- **Reducción del gasto público.**
- Facilita la identificación de las necesidades de la ciudad.
- **Favorece el desarrollo social y la innovación**

- La **información es en tiempo real**

Dentro de las iniciativas que se enmarcan en las ciudades inteligentes tenemos las siguientes:

- **Movilidad urbana: en este apartado se desarrollan herramientas para gestionar el** tráfico en tiempo real, realizar la gestión de flotas, gestión de transporte de viajeros, gestión de aparcamientos, etc.
- **Gobierno participativo y e-Administración.**
- **Seguridad pública:** Herramientas para la gestión de servicios públicos de emergencia y protección civil.
- **La eficiencia energética:** gestión de parques y jardines públicos, medición de parámetros ambientales, recogida y tratamiento de residuos urbanos etc.
- **La gestión del mobiliario urbano y las infraestructuras de la ciudad:** gestión de infraestructuras públicas y equipamiento urbano, herramientas de geolocalización para el reporte de incidencias por parte de los ciudadanos, etc.
- **Salud:** gestión de los servicios sociales, servicios de sanidad pública, etc.
- **Educación y cultura:** e-learning, e-turismo, e-comercio, teletrabajo, etc.

En este trabajo final de grado, se pretende crear una imagen digital en la iniciativa de eficiencia energética para controlar el mapa térmico en tiempo real de una zona y poderse extrapolar a la extensión que se pudiera medir con los sensores apropiados para ello.

Para ello, se realizará la recolección de datos provenientes de un sensor conectado a un Arduino, georreferenciándose en puntos dentro de un modelo de datos relacional espacial (PostegreSQL+ PostGIS). La información cartográfica de referencia se obtendrá del servicio remoto de imágenes estáticas de Catastro, vía protocolo WMS, que nos mostrará parcelas, edificios y calles.

Algunos de los antecedentes de Smart cities en España son [3]:

- **Málaga**: El proyecto de Smart City de la cuidad de Málaga se basa en la **gestión de la energía.** Integrando fuentes renovables en la red eléctrica, instalando 17.000 contadores inteligentes e instalando en los edificios emblemáticos de la ciudad.
- **Barcelona**: El proyecto de Barcelona se basa en hacer que las TIC sean elementos básicos cuando se efectúan algunos de los servicios ciudadanos.
  Uno de los proyectos más claros es la "**Isla de energía de Endesa**" que se trata de un punto de recarga rápido de vehículos eléctricos, el primero en España.
- **Bilbao:** La ciudad de Bilbao está haciendo un cambio completo para reducir su impacto medioambiental e incrementar su eficiencia energética.
- **Santander:** En la ciudad de Santander se han instalado un gran cantidad de sensores que se encargan de recoger la información de cómo se encuentra la ciudad y lo usan para la funcionalidad de esta misma. Un ejemplo muy claro lo encontramos en el sistema de riego, que solo funciona si los sensores detectan que no está lloviendo, por tanto, hay un gasto muy inferior de agua.
- **Valencia:** Finalmente, Valencia en si también cuenta con un proyecto piloto que plantea una gestión mediante la base de una smart city en el Mercado de Russafa. El proyecto consiste en extraer datos de los sensores integrados en el edificio para que en la *Plataforma VLCi* se puedan visualizar en cuadros de texto. [4]

1.2 Objetivos

El objetivo principal de este trabajo es crear un sistema de visualización en tiempo real que muestre de forma continua e interoperable en formato imagen datos discretos (ej. datos de temperatura, calidad del aire, etc.), provenientes de sensores Arduino para facilitar la toma de decisiones y poder compartir y distribuir sobre Internet dicha información de forma sencilla y compatible.

En primer lugar, es necesario encontrar el cliente que se va a utilizar en el trabajo, Realizamos la búsqueda de un microcontrolador de bajo consumo, con conexión wifi y a base de baterías que capture datos discretos de ejemplo (temperatura, datos de calidad del aire, etc.) y realizaremos la programación de dicho dispositivo para la captura, fechado y envío de peticiones HTTP de los datos observados.

Y en segundo lugar el servidor, donde haremos la configuración de una capa intermedia en el servidor utilizando el lenguaje de programación Python y librerías Flask. para la captura de las peticiones HTTP y su traducción a lenguaje SQL Espacial (entidades de tipo puntual según el estándar ISO SQL/MM 13249-3) de los datos recibidos y almacenamiento en un sistema relacional PostgreSQL y su extensión espacial PostGIS, previo diseño de un modelo de datos relacional adecuado.

Todos los datos registrados de los sensores quedarán almacenados en este modelo de datos en función de la localización, el tiempo de captura y el dato. El modelo constará de los índices unidimensionales y bidimensionales (espaciales) necesarios para su máxima eficiencia y la configuración de un servidor de servlets (p. ej.: tomcat) y un servidor especializado (Geoserver) que utilice el protocolo WMS (ISO 19152) que sea capaz de calcular utilizando lenguaje SLD, las imágenes continúas a partir de los datos discretos de los sensores mediante un proceso de interpolación matemática (p. ej.: pesos inversos a la distancia) y Distribuir la información obtenida mediante dicho protocolo, que asegura el compartir información georreferenciada continua de forma estándar entre el software para la toma de decisiones.

Por lo tanto, como objetivos específicos tendremos los siguientes:

1. Programar el dispositivo Arduino para la captura de datos medioambientales como temperatura, humedad y calidad del aire. Este dispositivo mandará los datos periódicamente a un servidor propio (programado en Python) mediante conexión a WIFI realizando peticiones HTTP GET.

2. Generar un programa de servidor (Python) que capture las peticiones HTTP GET que realizarán los dispositivos Arduino con los valores de temperatura, humedad y calidad del aire para poder almacenar de forma estructurada, con registro temporal y componente espacial los datos provenientes de dispositivos IOT.

3. Crear un modelo de datos para almacenar los datos de los sensores, utilizando

integridad referencial, para ello se usará el Sistema de Gestión de Bases de Datos PostgreSQL, donde se crearán dos tablas, una de sensores y otra de datos relacionadas entre sí y donde quedarán registrados (con marca de tiempo) todos los datos de los sensores para posteriores consultas.

4. Realizar consultas (SQL) para crear informes, análisis, etc. de los datos, incluyendo análisis de históricos. Se pretende conseguir dos objetivos, por un lado, crear informes “temperaturas/humedad/calidad, medias/máximas/mínimas de la última hora/día/mes”, y por el otro relacionar las dos tablas de nuestro modelo de datos añadiendo la componente espacial de las localizaciones de las estaciones meteorológicas, para posteriormente poder ser visualizadas de forma gráfica en un sistema de información geográfica.

5. Distribuir la información espacial de forma estándar y en tiempo real (menos de 5 segundos de retardo), de forma que sea accesible desde cualquier plataforma tanto alfanumérica (cualquier cliente SQL de bases de datos tendrá acceso a los informes y a los datos numéricos almacenados en el servidor de PostgreSQL), como gráfica (utilizando estándares de distribución de la información espacial bajo demanda, como WMS) mediante un servidor.

6. Programar el Geoserver para que acceda a las tablas espaciales de nuestro modelo de datos, las convierta a datos gráficos geolocalizados, y a superficies continuas (mapas meteorológicos) bajo demanda, y las distribuya mediante una URL con el protocolo WMS para que cualquier programa (meteorológico, CAD, o SIG) puede cargar la información espacial de forma gráfica y geolocalizada.

1.3 Organización del documento

El documento está organizado en 6 secciones, en las cuales se detalla por partes todo el proceso que se ha llevado a cabo durante la realización de este trabajo.
Primeramente, la Sección 1 es una introducción donde se comentan los motivos que han llevado a realizar este estudio. También se exponen de forma clara cuales son los objetivos marcados en este proyecto y se comenta la distribución de los contenidos dentro del documento.
En la sección 2 se presentan las herramientas utilizadas en el desarrollo del trabajo.
En la sección 3 se muestran los diferentes programas y softwares que se han empleado para el desarrollo del TFG.
La Sección 4 es la correspondiente al cuerpo principal de este estudio, se detalla la programación utilizada en los sensores para poder obtener las imágenes continuas de los datos obtenidos por los sensores en un servidor de cartografía, así como la visualización de dichos datos mediante un sistema de información geográfica.
El trabajo finaliza con la Sección 5 en la que se presenta las conclusiones y el presupuesto del trabajo. Se termina este TFG con las referencias bibliográficas.

2. Descripción de herramientas y estándares

2.1 Máquina Virtual

Una máquina virtual se trata de un software que permite instalar en su interior otro sistema operativo simulando que tiene otro ordenador real, es decir emula un ordenador completo, tiene su propia memoria, disco duro, tarjeta gráfica, aunque sean todos ellos virtuales.
El acceso a la máquina virtual se obtiene desde el propio ordenador (*host*) accediendo a este como a una ventana más dentro de nuestro sistema operativo, pudiéndose ejecutar distintos sistemas operativos, dentro de la máquina virtual, sin estar directamente instalados en el *host.* [5]

Al crear la máquina virtual, se tiene que asignar los recursos que se van a necesitar, cuánta memoria RAM, cuanto espacio en el disco duro… Estos recursos se adquieren directamente de los propios del *host.*

La realización de este trabajo final de grado conlleva la instalación de varios softwares: el servidor web: Apache HTTP Server, el servidor de servlets Apache Tomcat, el sistema de información geográfica de escritorio QGIS, el servidor de mapas Geoserver, el sistema de gestión de base de datos relacional PostgreSQL, la extensión espacial PostGIS, bibliotecas de tratamiento ráster y vectorial: gdal, etc.

Uno de los principales usos de las máquinas virtuales es poner tener softwares diferentes en tu ordenador sin necesidad de instalarlos en este como tal y por tanto sin la problemática de que los procesos de instalación de dichos softwares que se van a utilizar puedan tener problemas con el software instalado en nuestros equipos o puedan desconfigurar o ralentizar el trabajo habitual del ordenador.[5]

En nuestro caso usaremos la máquina virtual VMWare Player (**Figura 1**) ya que se trata de una máquina de rápido rendimiento y de fácil uso.
Al utilizar la máquina virtual, todos los cambios que se realicen permanecerán en los discos duros virtuales (.vmdk). Se podrá copiar dicho directorio y seguir trabajando en cualquier otro ordenador, pudiéndose realizar copias de seguridad del ordenador virtual completamente.

Figura 1. Captura de la máquina virtual

2.2 Arduino

Arduino se trata de una plataforma “open-source”, es decir, de código abierto, que permite realizar proyectos de electrónica interactivos con un software y hardware de fácil uso para cualquier persona.
Tiene diferentes usos, tanto para desarrollar proyectos autónomos como para conectarse a otros dispositivos, usarlo con otros programas e interactuar con hardware, así como con software.

En nuestro caso, usaremos diferentes componentes para el proceso de montaje de Arduino.

- **La placa NodeMcu Amica:** Se trata de una placa basada en el módulo **WIFI ESP8266,** siendo una de las más interesantes dentro de este tipo ya que ofrece acceso a todos los pines ESP8226, entre ellos a los 4 pines que están destinados a la comunicación SPI, que son los **CS0**, **MISO**, **MOSI** y **SCL**, así como a un 1 convertidor A/D con antena integrada [6]. Que podemos ver en la **Figura 2:**

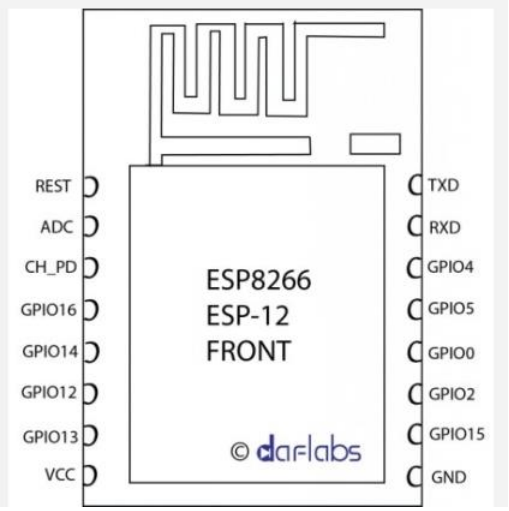

Figura 2. Esquema de pines tabla ESP8266

- **Sensor DHT11**: Se trata de un sensor de temperatura y humedad que nos proporciona estos datos de forma digital.
  Funciona con 3,3 y 5 V de alimentación, tiene un rango de temperatura de 0º a 50º y un rango de humedad de 20% a 80% ambos con un 5% de precisión.
  Nos proporciona una muestra por segundo y devuelve las medidas en ºC.
  Se trata de un sensor muy barato y de bajo consumo [7].

- **Sensor MQ135**: Se trata de un sensor de control de calidad que mide la contaminación en el medio ambiente detectando la concentración de gas.
  Proporciona una señal tanto analógica como digital.
  La señal digital puede ser procesada por un microcontrolador.
  La corriente de operación es 150mA y el voltaje de entrada es de 5V, utiliza una potencia de consumo de 800mW y un tiempo de precalentamiento de 20 segundos.
  Tiene una alta sensibilidad y un tiempo de respuesta rápido [8].

- **Conexión WIFI:** Como hemos hablado anteriormente, nuestra placa está basada en el módulo ESP8226. Este se basa en el chip ESP226, que incluye todo lo necesario para conectarse al WIFI y un procesador interno de 32 bits a 80MHz.
  Algunas de sus características más interesantes son que consta con 1 Mb de memoria flash, 80 K de DRAM y 35 K de IRAM. Así como la gestión completa del WIFI con amplificador incluido [9].

- **Envío de datos de los sensores mediante HTTP GET:** Se trata de un método que se usa para pedir recursos a un servidor. Las solicitudes que usan el método GET solo recuperan datos, para cambiar datos se deben usar otros métodos como PUT o DELETE.
  Las peticiones GET no pueden tener un cuerpo de mensaje, pero si pueden enviar datos al servidor usando parámetros URL.
  En el método GET, múltiples solicitudes GET que sean idénticas, tendrán el mismo efecto que si fuera una sola.
  Los buscadores usan el método GET para pedir una página del servidor [10].

2.3 Arquitectura cliente/servidor

Se trata de un sistema distribuido por el cual un cliente solicita un servicio o recurso a un servidor.
La comunicación entre estos se realiza, generalmente, a través de un entorno red y normalmente, están en hardware separado, aunque pueden estar en el mismo.
El intercambio de mensajes entre el cliente y el servidor se realiza siguiendo **un patrón de petición respuesta** (*request-response*), por el cual el cliente envía o solicita una petición y el servidor la atiende y genera una respuesta que se devuelve al cliente [11]. Se puede observar una representación gráfica de este proceso en la **Figura 3**

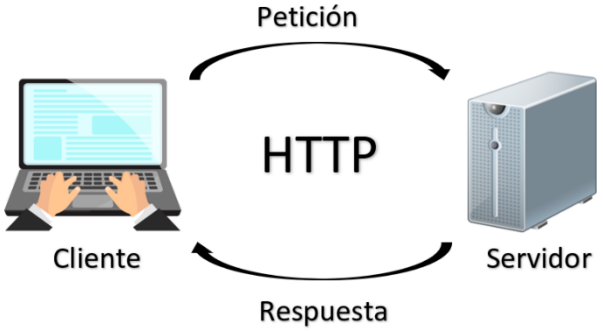

Figura 3. Diagrama cliente/servidor

Para poder realizar correctamente esta comunicación se necesita un lenguaje común, así como seguir unas reglas para que ambas partes sepan cómo actuar, esto queda definido en el **protocolo de comunicación.**

Para la correcta comunicación necesitamos conocer las **direcciones IP** tanto de origen como destino. La dirección IP se trata de la dirección que identifica los dispositivos de red de nuestro equipo.

También es necesario conocer **el puerto** de origen y destino. El puerto se trata de un valor numérico de 0 a 65535 que identifica la conexión cliente-servidor.

Y finalmente el nivel de aplicación, usaremos el más conocido HTTP (*Hyper Text Transfer Protocol)*

2.4 Open Geospacial Consortium

El Open Geospatias Consortium (OGC) se fundó en 1994 para definir estándares abiertos e interoperables dentro de los Sistemas de Información Geográfica (SIG) y hacerlos una parte integral de la infraestructura mundial de información [12].

Persigue que las diferentes empresas del sector creen acuerdos que posibiliten la interoperación de sus sistemas de geoprocesamiento y faciliten el intercambio de información geográfica. Esto es posible ya que los miembros desarrollan de forma colaborativa los estándares de interfaz y asociados y así pueden intercambiar IG fácilmente. Uno de los aspectos más importantes a resaltar que tiene el OGC es la gran función que realiza en materia de estandarización.

Las especificaciones con las que cuenta el OGC garantizan la interoperabilidad de los servicios de IG y de los contenidos que tiene disponibles, de una forma pública y sin costes en la propia página web (*http://www.opengeospatial.org*) (OGC, 2020)

Los estándares que vamos a usar son:

2.4.1 WMS: Se trata de un estándar para la implementación de servicios de acceso a cartografía. Permite definir operaciones para poder obtener mapas como imágenes, información sobre diferentes puntos del mapa y capacidades del servicio [13].

2.4.2 SQL: Se trata de un estándar de comunicación dentro de las bases de datos que nos permite el acceso global y la manipulación de datos. Otra característica es que se puede integrar a lenguajes de programación y combinar con bases de datos específicas, como MySQL o SQL Server [14].

2.4.3 SLD: Este estándar para la simbología, es un esquema XML que ayuda a definir propiedades visuales de los objetos geográficos que componen el mapa (color, tipo de línea groso, símbolo, etc.) [15].

2.4.4. WFS: Este estándar define como implementar los servicios de acceso a datos vectoriales en bruto, de esta forma, permite acceder y consultar todos los atributos de un fenómeno (*feature*) geográfico que se encuentre representado en modo vectorial.
Una de las cosas positivas del WFS frente al WMS es que no solo permite visualizar la información, sino que también permite consultarla libremente [16].

2.5 Base de datos espacial.

Definimos una base de datos como una colección de elementos o datos interrelacionados que pueden ser procesados por diferentes sistemas de aplicación [17]. Gracias a esto, se puede evitar tener una redundancia de datos significativa, haciendo que los datos aparezcan solo una vez en el sistema en forma de tablas y utilizando las relaciones para ello. También, tener los archivos indexados permite identificarlos antes y tener control sobre las consultas que se realizan.

Teniendo en cuenta que nos movemos en un mundo lleno de datos, es importante poder saber cuáles son los datos que nos interesan, sus características y como se relacionan entre sí para poder convertirlos en información. Es por ello, que un apartado importante en cualquier proyecto es definir el modelo de datos que se va a utilizar para almacenar la información en el sistema, en este caso se va a generar una Base de datos Espacial y para ello se utilizará el sistema de gestión de bases de datos relacionales PostgreSQL y su extensión espacial PostGIS.

2.5.1 PostgreSQL.

PostgreSQL (**Figura 4**), es un sistema de gestión de bases de datos de objeto abierto y muy avanzado. Se utiliza por muchas organizaciones y universidades para almacenar sus datos
Presenta las siguientes ventajas:

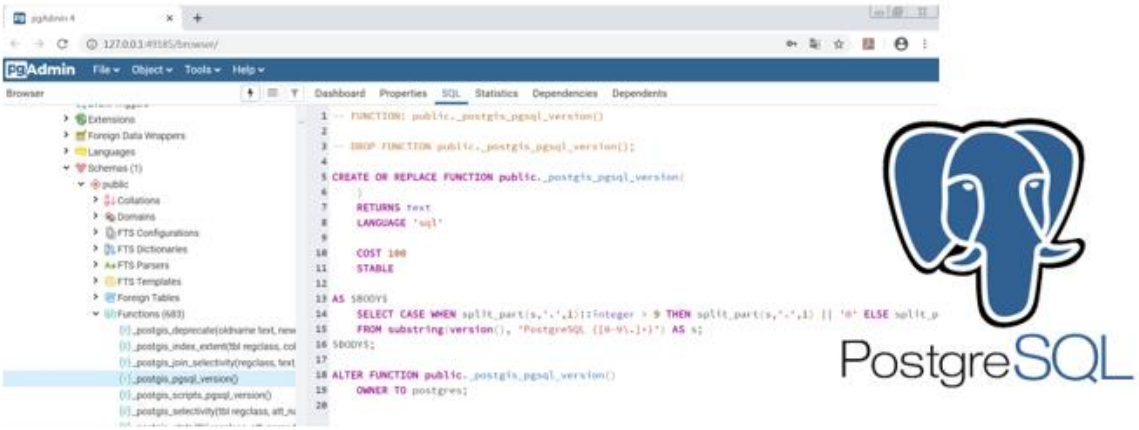

Figura 4. Captura de PostgreSQL

- **Instalación gratuita y en todos los equipos que se necesite**, independientemente de la arquitectura y la plataforma que se esté utilizando. Es un sistema multiplataforma, está disponible en diferentes sistemas operativos (Linux, Windows 32 y 64 bits, Unix.)
- Es **escalable**, es decir se puede configurar PostgreSQL en cada equipo según el hardware, por lo tanto se pueden realizar mayor cantidad de peticiones simultáneas a la base de datos de forma eficiente.

- **Estable y fiable**. Existe una comunidad de desarrolladores activos detrás del proyecto desde hace más de 20 años. PostgreSQL Tiene la capacidad de permitir realizar consultas a los clientes mientras los servidores están en espera o en modo recuperación y de esa forma no bloquear el sistema para consultas mientras se realizan estas tareas.
- **Usa la herramienta gráfica pgAdmin**, para hacer más fácil e intuitivo el trabajo de creación, gestión y administración de las bases de datos.
- **Es un estándar SQL**, ya que implementa las funcionalidades de la ISO/IEC 9075:2016, por lo tanto, permite poder incluir scripts de otros SGBD, así como realizar consultas a dichos sistemas.
- **Potente y robusta**: cumple con la característica ACID Compliant. Por lo tanto, provee atomicidad, consistencia, aislamiento y durabilidad para sus operaciones.
- **Extensible**: existen varias extensiones disponibles distribuidas por el grupo de desarrolladores del proyecto y es posible generar las extensiones que se necesiten utilizando diversos lenguajes de programación. [18].

Uno de los puntos más interesantes es su capacidad de trabajar y almacenar datos de tipo geométrico, utilizando la extensión PostGIS.

2.5.2 PostGIS

PostGIS se trata de una extensión espacial para PostgreSQL, un módulo de ampliación para PostgreSQL de gran utilidad a la hora de trabajar en proyectos de sistemas de información geográfica.

Gracias a PostGIS encontramos unas claras mejoras en PostgreSQL, como proveer funciones de transformación, análisis y consulta espaciales, dar soporte para archivos SIG ráster y vectoriales y herramientas de geocodificación 3D, topología, etc.

Se aumenta la velocidad de procesamiento gracias a índices espaciales.

En conclusión, convierte PostgreSQL en una base de datos espacial, que en general, en la práctica, funciona exactamente como un SIG ya que habilita el soporte para poder trabajar con objetos geográficos que se encuentren localizados en el espacio, lo único que se necesitaría sería un visualizador para ver las capas generadas en PostGIS [19].

2.6 SIG

La cartografía ha tenido desde el principio de su existencia como finalidad, aportar la información geográfica para poder tomar decisiones y con ello mejorar la capacidad de trabajar con los mapas. Con el paso de los años, ha aumentado la complejidad de estos sistemas y por tanto se necesita una variedad de información geográfica que procede de diferentes fuentes y con diferentes formatos.
Los SIG (Sistemas de información geográfica) se definen como una tecnología para poder manejar información geográfica y está compuesta por cuatro elementos fundamentalmente, que son el hardware, software, datos espaciales que permitan realizar análisis complejos y personal técnico que defina los criterios adecuados para ello (**Figura 5**) [20].

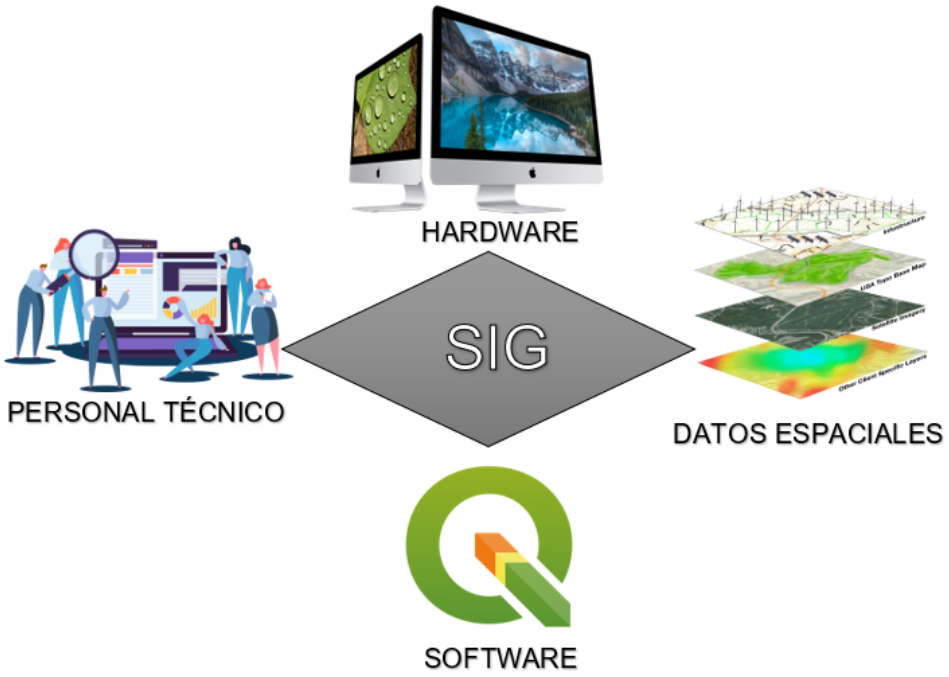

Figura 5. Componentes de un SIG

Otra definición de SIG podría ser, definirlo como un software específico con el cual los usuarios tienen la posibilidad de crear consultas y permite analizar y representar cualquier tipo de información geográfica referenciada que esté asociada a un territorio al conectar información gráfica con bases de datos.
Esto facilita la visualización de los datos que se han recogido y plasmado en un mapa con la finalidad de poder usarlos para relacionar diferentes fenómenos geográficos [21].

En este trabajo, se va a utilizar el software QGIS, que es un SIG de código abierto con licencia GNU(General Public License), producto oficial de la fundación de código abierto geoespacial (OSGeo), que soporta numerosos formatos y funcionalidades de datos vectoriales, ráster y bases de datos. (https://www.qgis.org/es/site/about/index.html)

2.7 Geoserver

GeoServer es un servidor de datos espaciales. Está escrito en java y es una aplicación web que permite a los usuarios compartir y editar datos cartográficos georreferenciados.

Está diseñado para garantizar la interoperabilidad, es decir, la capacidad que tienen los sistemas de información, de compartir datos y hacer posible el intercambio de información entre ellos sin necesidad de saber qué sistema están utilizando ni condicionando a un software y un sistema operativo determinado [22].

Es un proyecto vivo, en continua actualización y con una amplia comunidad muy activa, que tiene un interfaz muy amigable y soporta datos de múltiples formatos y es capaz de leer datos de bases de datos externas, compuestas de fuentes de datos diferentes y publicar datos de múltiples servicios de datos espaciales usando estándares abiertos.

Implementa los estándares de los protocolos OGC como el WMS y el WFS explicados anteriormente, así como soporta estilos CCS y tiene extensión al estándar SLD.

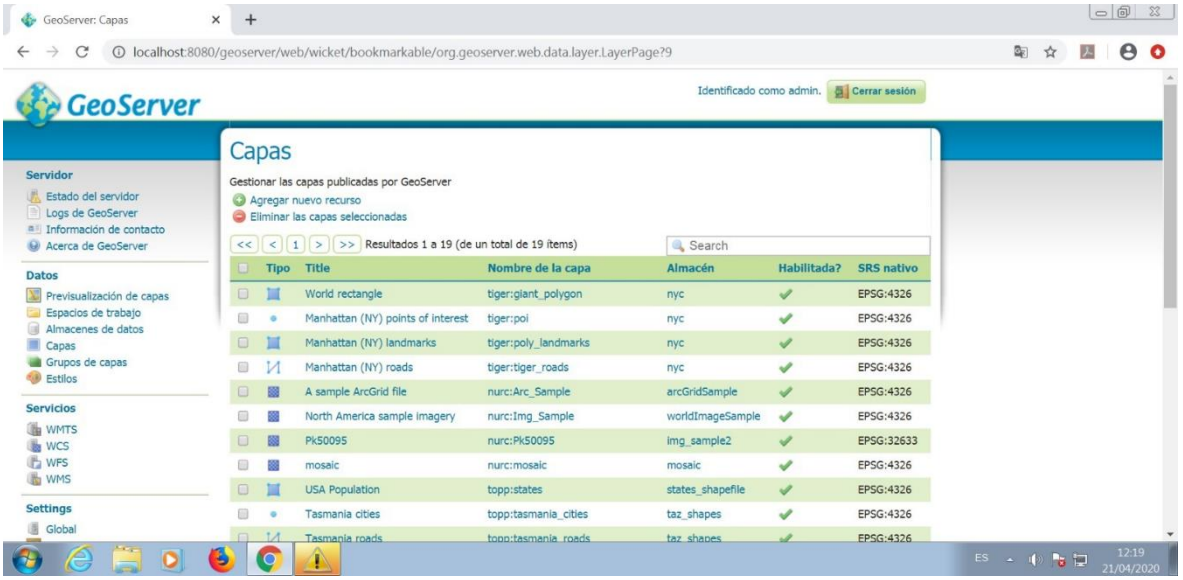

Figura 6. Captura de Geoserver

3. Software empleado

El Software que se utiliza en este proyecto es el siguiente:

Como se ha indicado anteriormente, la totalidad de este trabajo final de grado se ha realizado usando una máquina virtual. Para poder obtener esta máquina virtual, se descargará el software VMWare Workstation Player para MS Windows desde el sitio web oficial de VMWare.
https://www.vmware.com/es/products/workstation-player/workstation-player-evaluation.html
La versión de la máquina virtual que se va a utilizar se trata de **Workstation 15 Player para Windows.**

Una vez descargada, se obtienen dos tipos de ficheros, ficheros de extensión .vmdk que se corresponden con los discos duros virtuales y ficheros de extensión .vmx, fichero de texto con la configuración de la máquina virtual.

Seguidamente, se parametrizan los valores utilizando el fichero vmx que se destinaran a la máquina virtual desde el propio ordenador.

Para la programación de Arduino se realizará con la versión de **Arduino 1.8.10 para Windows** desde el sitio oficial de Arduino.
https://www.arduino.cc/en/main/software

En lo referente al sistema de gestión de bases de datos relacional para la gestión de los datos de los sensores se utilizará la versión de **PostgreSQL 12.1 Windows x86-64** que se descargará desde el sitio web oficial de PostgreSQL.
https://www.postgresql.org/download/windows/

Con la instalación de PostgreSQL se tendrá que instalar **pgadmin4 en su versión 4.15** que será el cliente SQL con el que se accederá al servidor de PostgreSQL. La descarga de este software se realizará también desde el sitio web oficial de pgAdmin.
https://www.pgadmin.org/download/

Para el almacenamiento de geometrías y se gestión espacial se utilizará la extensión espacial de PostgreSQL **PostGIS 3.0**, la descarga de esta extensión se realizará desde http://download.osgeo.org/postgis/windows/pg12/

El SIG que se utiliza para acceder a las superficies digitales continuas se trata de **QGIS 3.10.1** que se descargará desde el sitio web de QGIS.
https://www.qgis.org/es/site/forusers/download.html

Para la instalación de Geoserver, se utilizará la versión **Geoserver-2.16.0**, para poder realizar la instalación de Geoserver se necesita instalar un programa que actúe de servidor y tener la máquina virtual de Java (JRE) instalada en el sistema operativo.

El programa que se utiliza como servidor es Apache Tomcat, en la versión **Tomcat 9.0.27** y para la máquina virtual de Java la **JRE 8.**

4. Modelo o arquitectura para la adquisición y visualización de datos

En este apartado se procede a explicar las diferentes partes del proceso seguido para la realización del proyecto.

Se describen las configuraciones y los procesos en todos los softwares utilizados, así como los programas desarrollados en este proyecto para alcanzar el objetivo presentado anteriormente

4.1 Diagrama de flujo

Antes de describir uno a uno los diferentes procesos, se muestra en las siguientes figuras los diagramas de flujo utilizados.

En la **Figura 7**, se describe la estructura cliente-servidor utilizada para el desarrollo del TFG mostrando más detalladamente en la **Figura 8** y la **Figura 9** los pasos para la creación del cliente y servidor respectivamente.

Así mismo se procede a explicar con más detalle cada figura. En la **Figura 8**, se pueden observar dos ramas, esto es debido a que el proyecto, aunque su finalidad es la conexión de Arduino con varios sensores para la toma de datos, se procede también a realizar la simulación de varios sensores localizados en sitios diferentes para enriquecer este trabajo y obtener más datos.
En este proceso se conecta el Arduino a Wi-Fi, se capturan los datos y se estudian las peticiones que esto conlleva para seguidamente simular n sensores que serán los que envíen las peticiones al cliente.

En la **Figura 9**, se observa la configuración de Python que va a ser el lenguaje de programación que se va a utilizar, se configura el entorno de desarrollo integrado Eclipse, se instalan las librerías y se genera el módulo para que el servidor capture las peticiones de los sensores.

Teniendo en cuenta estas dos figuras, que serían la explicación del cliente y servidor de la **Figura 7**, se procede a explicar esta misma. Después de la conexión cliente-servidor, se genera el diagrama entidad relación para crear el modelo de datos en la base de datos relacional PostgreSQL, se generan tablas y vistas para relacionar consultas SQL, se visualizan los datos espaciales en el sistema de información geográfica QGIS, también se generan servicios de protocolo del OGC (WMS) y se configura el servidor de datos espaciales Geoserver. Se genera la simbología que se va a utilizar para la representación de la cartografía y se construyen capas superficiales a partir de puntos para obtener el servicio WMS que proporcionará la forma de distribuir las imágenes continuas de los datos de los sensores en tiempo real y de forma estándar, y que será el resultado final, de forma que el servicio que estará disponible para las peticiones de cualquier cliente (SmartCity, webmapping, SIG).

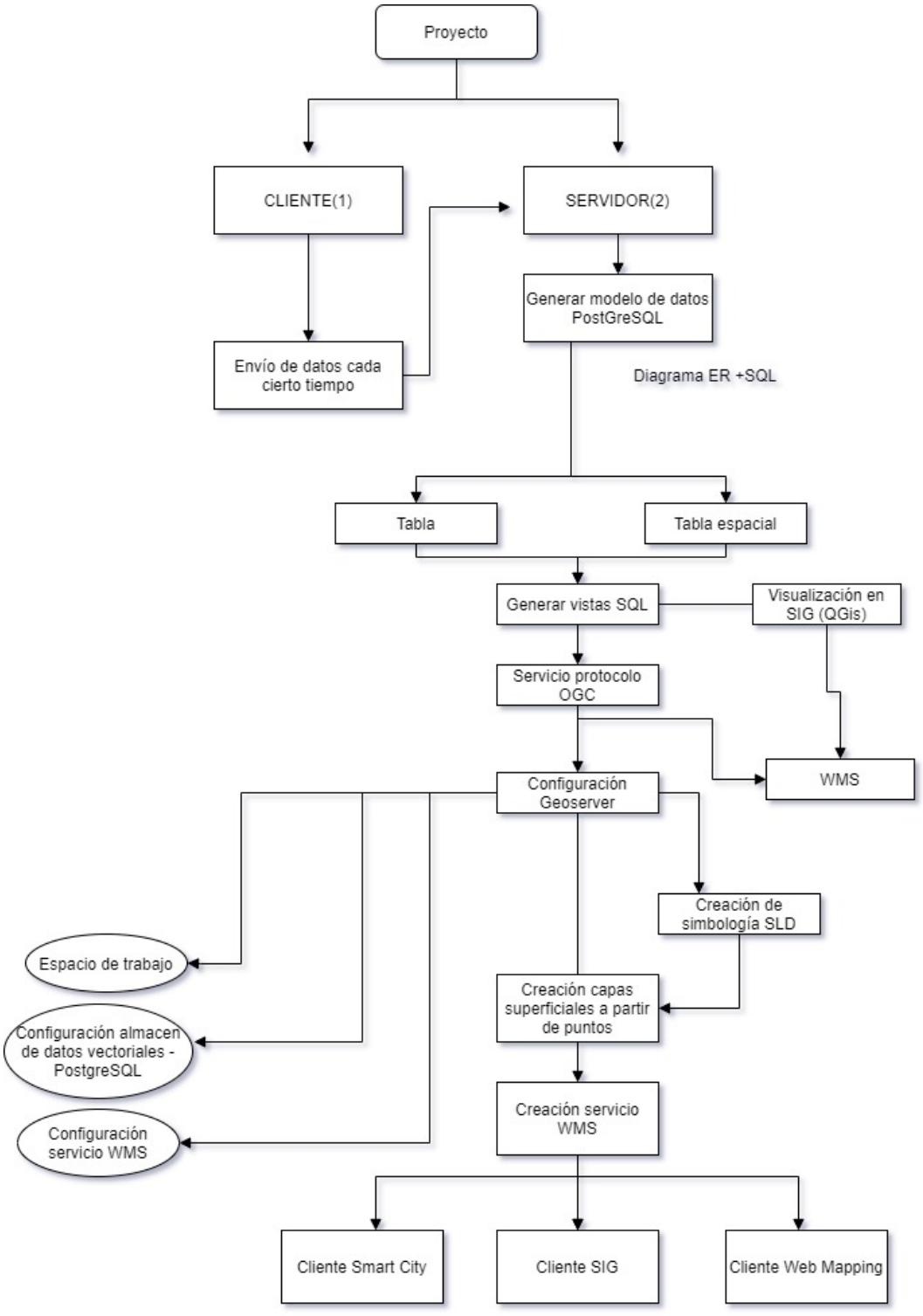

Figura 7. Diagrama de flujo general

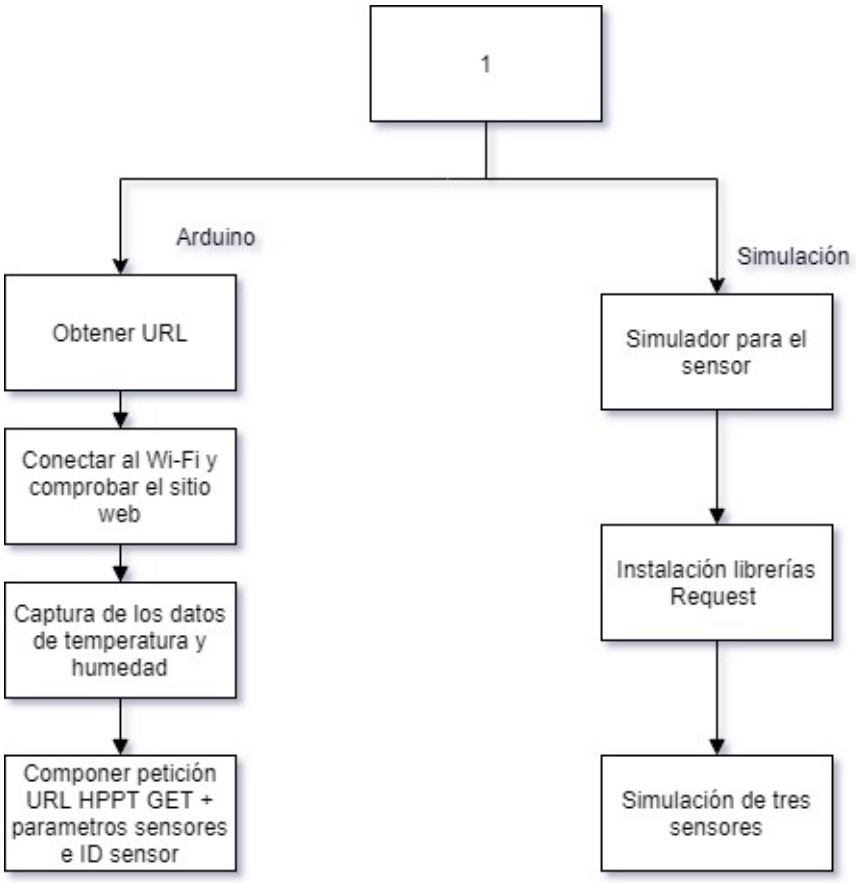

Figura 8. Diagrama de flujo del cliente

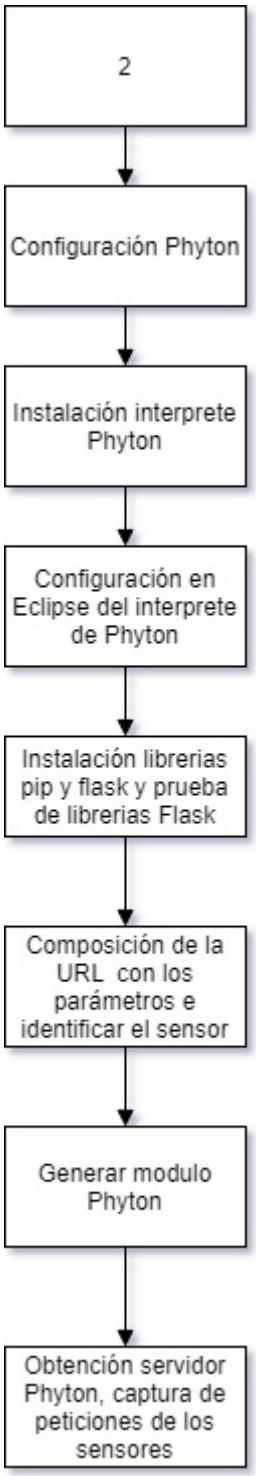

Figura 9. Diagrama de flujo del servidor

4.2 Programación Arduino para la captación de datos

Para empezar este trabajo de fin de grado se procederá a preparar la programación de los sensores y el Arduino para poder realizar correctamente la captación de datos de los sensores.

Para ello, primeramente, se realizará el montaje del circuito con el Arduino y sus respectivos sensores usando los pines correspondientes. Este montaje lo podemos observar en la **Figura 10.**

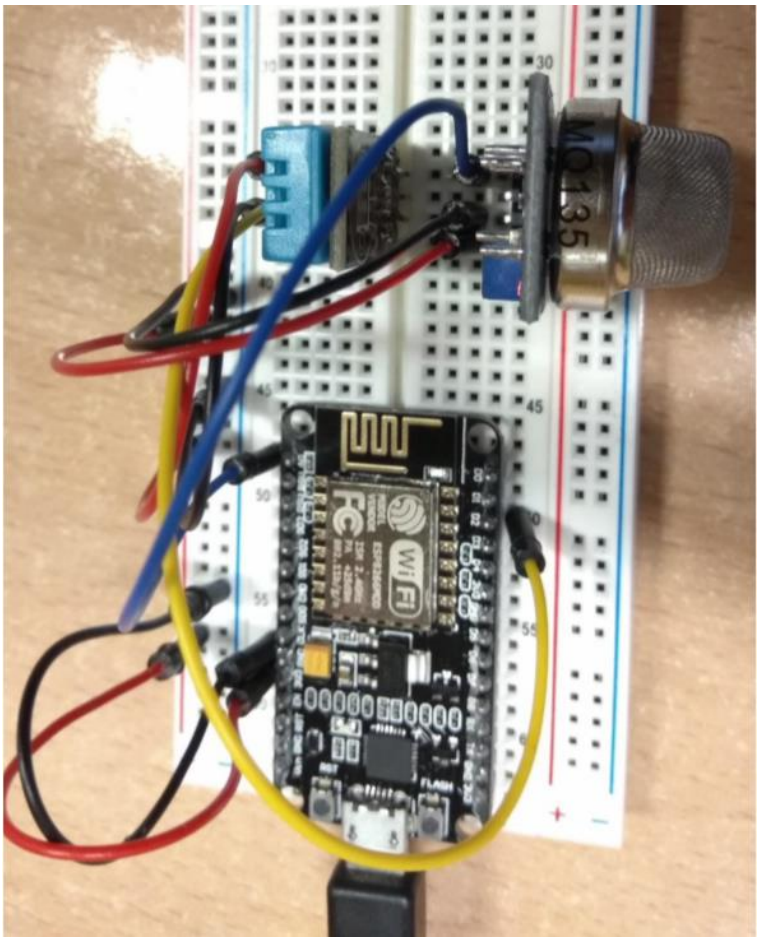

Figura 10. Esquema de montaje del Arduino con los sensores.

Una vez se ha montado el circuito, se comprobará el correcto funcionamiento de este. Se debe comprobar también, que el servicio de conexión Wi-Fi realiza correctamente su función.

Seguidamente, se entrará en una web para que cree una URL aleatoria que se usará posteriormente en el código de Arduino, la web que se usará para ello se trata de http://webhook.site, que facilitará una URL como se puede observar en la **Figura 11.**

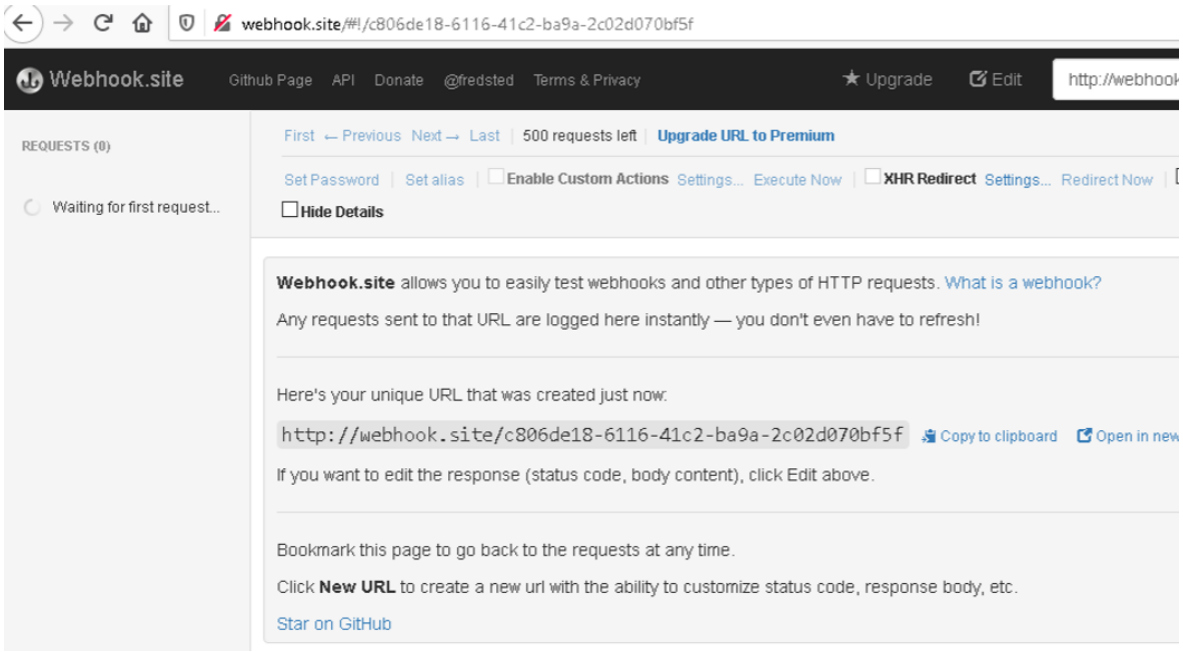

Figura 11. Captura de la página http://webhook.site

En este caso, la URL adquirida se trata de http://webhook.site/c806de18-6116-41c2-ba9a-2c02d070bf5f y será la URl que se pondrá en el código de Arduino para analizar las peticiones HTTP GET.

Una vez elegido este sitio web, se quedará a la espera de recibir las peticiones.

Uno de los objetivos de este trabajo es conseguir posteriormente poder sustituir este sitio web de prueba por el servidor que se va a programar en Python y que se encargará de recoger las peticiones HTTP GET de los sensores para almacenar los datos en el modelo de datos generado en el sistema de gestión de bases de datos PostgreSQL.

Para que reciba las peticiones, en primer lugar, se dará corriente a la placa de Arduino, esta deberá conectarse a la WIFI, configurando los valores de STASSID Y STAPSK con la red wifi y la contraseña respectivamente.

También se tendrá que comprobar el monitor serie de Arduino y el sitio web para comprobar que las peticiones llegan correctamente.

Cuando se inicia el Arduino, el sitio web no habrá recibido peticiones aún, como se puede observar en la **Figura 12.**

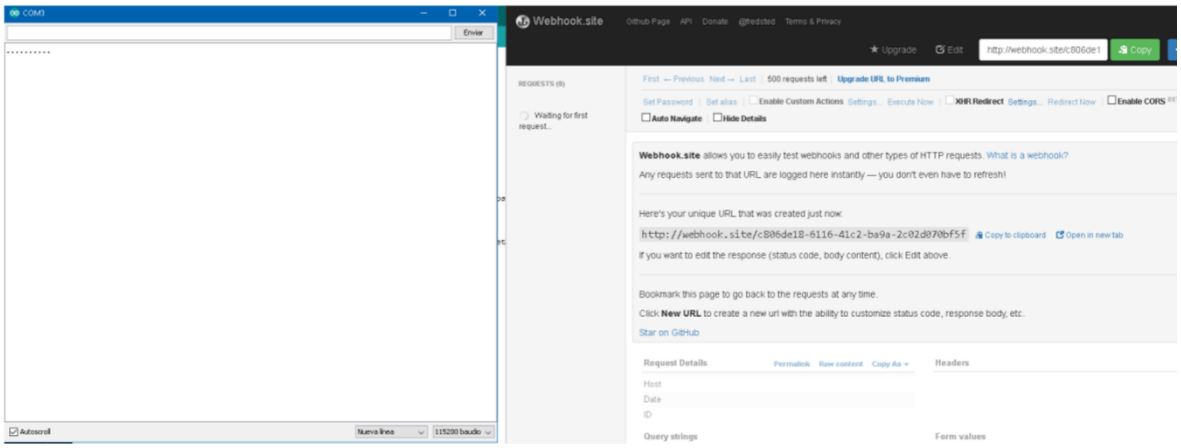

Figura 12. Programa a la espera de recibir peticiones

En la **Figura 13**, se puede observar cómo el Arduino se conecta a la Wi-Fi, coge los datos del sensor y envía la petición HTTP GET con los datos del sensor.

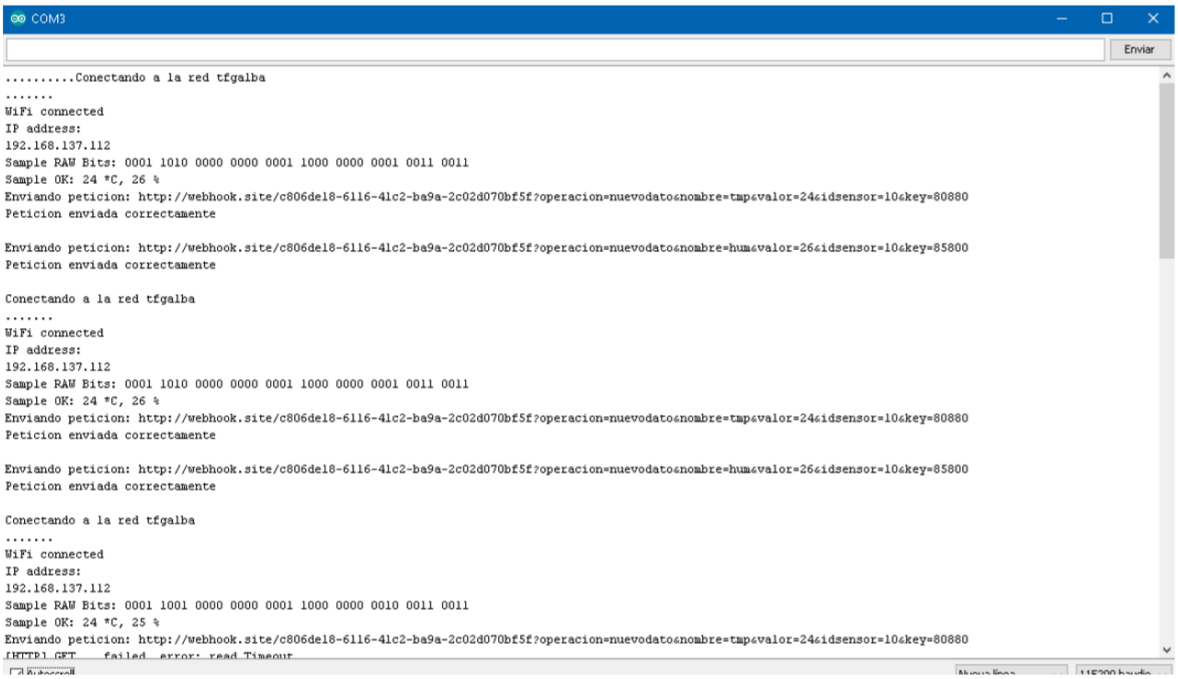

Figura 13. Conexión Wi-Fi, recogida de datos y envío de petición

El sitio web se actualiza con las peticiones, de esta forma, se comprueba que tanto la conexión Wi-Fi, como las peticiones HTTP GET en Arduino funcionan correctamente como se puede observar en la **Figura 14**.

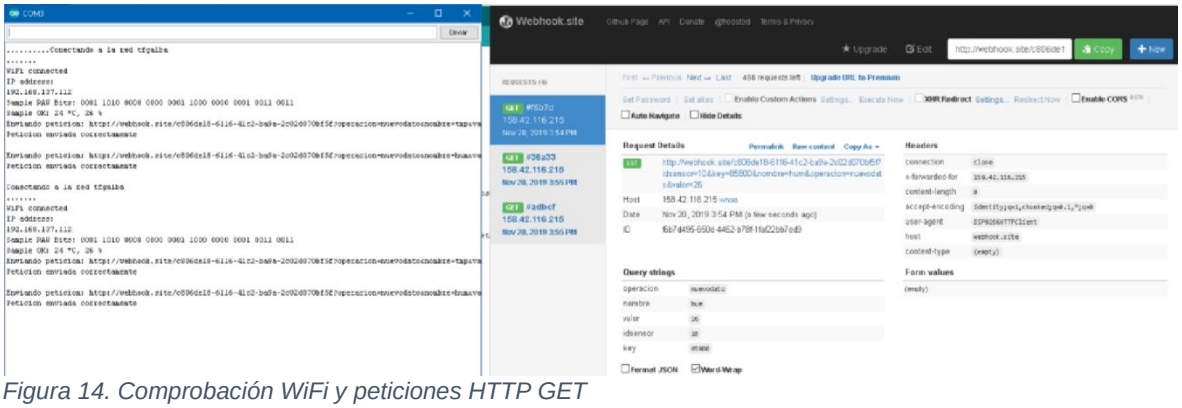

Figura 14. Comprobación WiFi y peticiones HTTP GET

Se ha realizado el siguiente programa para la captura de datos de temperatura, humedad y calidad del aire y posterior envío mediante peticiones HTTP GET, los datos al servidor.

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

#include <SimpleDHT.h> //Para la TMP y Humedad

/*
 * RED Wifi y contraseña
 * Mejora: Conectar la la red UPVIOT con WPA2 Enterprise
 */
#ifndef STASSID
#define STASSID "TFGalba"
#define STAPSK  "TFGalbaji"
#endif

const char* ssid     = STASSID;
const char* password = STAPSK;

/* ID del sensor, es almacenado en PostgreSQL con sus coordenadas Lat Lon de
localización */
const int id_sendor = 10;

/* Para comprobar las peticiones HTTP GET desde la WIFI del sensor utilizaremos
el sitio web:
 * https://webhook.site/
 * Este sitio web crea un end point y nos da la URL que ponemos en la variable
host
 */
String host = "http://webhook.site/c806de18-6116-41c2-ba9a-2c02d070bf5f";

/* Este host se sustituirá por la URL de nuestro servidor de python, tras comprobar
   que el código Arduino funciona correctamente */

/* Temperatura y humedad */
int pinDHT11 = 4;
int pinMQ135 = 0;
SimpleDHT11 dht11;
byte temperature = 0;
byte humidity = 0;
int mq135 = 0;
int idsensor = 10;

void setup() {
  Serial.begin(115200);
  conectaWifi ();
  Serial.println();
  Serial.println();
}
```

```
boolean conectaWifi () {
  Serial.print("Conectando a la red ");
  Serial.println(ssid);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  int nIntentos = 10;

  while (WiFi.status() != WL_CONNECTED && nIntentos > 0) {
    delay(500);
    nIntentos--;
    Serial.print(".");
  }

  if (nIntentos > 0) {
    Serial.println("");
    Serial.println("WiFi connected");
    Serial.print("IP address: ");
    Serial.println (WiFi.localIP());
    return true;
  }

  return false;
}

boolean cogeTmpHumedad ()
{
  byte data[40] = {0};
  if (dht11.read(pinDHT11, &temperature, &humidity, data)) {
    Serial.println("Read DHT11 failed");
    return false;
  }

  Serial.print("Sample RAW Bits: ");
  for (int i = 0; i < 40; i++) {
    Serial.print((int)data[i]);
    if (i > 0 && ((i + 1) % 4) == 0) {
      Serial.print(' ');
    }
  }
  Serial.println("");

  Serial.print("Sample OK: ");
  Serial.print((int)temperature); Serial.print(" *C, ");
  Serial.print((int)humidity); Serial.println(" %");

  return true;
}

boolean cogeMQ135() {
  mq135 = analogRead(pinMQ135);
  Serial.println("MQ135 analog output:  " + String(mq135));
```

```
  delay (500);

  return true;
}

int getKey (String nombre, int valor, int idsensor) {
   int key = 0;
   for (int i = 0; i < nombre.length(); i++) {
      key = key + nombre.charAt(i);
   }
   key = key * valor * idsensor;
   return key;
}

boolean enviaPeticionGet (String host, String nombre, int valor, int idsensor) {
    boolean exito = false;
    HTTPClient http;

    String request = host + "?operacion=nuevodato&nombre=" + nombre + "&valor="
+ valor +
                     "&idsensor=" + idsensor + "&key=" + getKey (nombre, valor,
idsensor);

    Serial.println("Enviando peticion: " + request);
    http.begin(request);  //Specify request destination
    int httpCode = http.GET();  //Send the request
    if (httpCode == HTTP_CODE_OK || httpCode == HTTP_CODE_MOVED_PERMANENTLY) {
        Serial.println("Peticion enviada correctamente");
        String payload = http.getString();   //Get the request response payload
        Serial.println(payload);             //Print the response payload

        exito = true;
    } else {
        Serial.printf("[HTTP]      GET...      failed,      error:      %s\n",
http.errorToString(httpCode).c_str());
    }

    http.end();   //Close connection
    return exito;
}

void loop() {
  int nIntentosGet;
  int nIntentosLecturaSensor;

  if (conectaWifi ()) {
    /*
     *  Lee temperatura y humedad y realiza las peticiones HTTP GET
     *  Al servidor de python para su gestion en la base de datos
     *  PostgreSQL
     */

    nIntentosLecturaSensor = 5;
```

```
    while (!cogeTmpHumedad () && nIntentosLecturaSensor > 0) {
      nIntentosLecturaSensor--;
      delay (500);
    }

    if (nIntentosLecturaSensor > 0) {
        //Manda temperatura
        nIntentosGet = 3;
        while   (!enviaPeticionGet   (host,   "tmp",temperature,idsensor)   &&
nIntentosGet > 0) {nIntentosGet--;}

        //Manda humedad
        nIntentosGet = 3;
        while   (!enviaPeticionGet   (host,   "hum",   humidity,   idsensor)   &&
nIntentosGet > 0) {nIntentosGet--;}
    }

    /*
     *  Lee calidad del aire y realiza las peticiones HTTP GET
     *  Al servidor de python para su gestion en la base de datos
     *  PostgreSQL
    */

    cogeMQ135();
    nIntentosGet = 3;
    while (!enviaPeticionGet (host, "mq135", mq135, idsensor) && nIntentosGet >
0) {nIntentosGet--;}

  }

  //Para ahorrar energía se conecta a la WIFI y envía las peticiones solo cada 60
segundos
  WiFi.disconnect();
  delay (60000);
}
```

4.3 Instalación de Python y entorno de desarrollo

Lo ideal sería poder disponer de sensores repartidos por varios lugares y poder obtener los datos en el servidor para representarlos posteriormente en el servidor de cartografía. En el apartado anterior se ha explicado el proceso que seguirían los sensores para un Arduino y en este apartado se va a realizar la simulación de n sensores que realizaran peticiones HTTP GET como lo hacía el Arduino.

El lenguaje utilizado para programar la simulación será Python y se utilizará el entorno de desarrollo integrado Eclipse+ con su respectivo plugin Pydev para poder soportar Phyton.

Por lo tanto, se instala eclipse desde https://www.eclipse.org/ide/ y en el menú help/eclipse market place se busca PyDev y se instala el intérprete de Python: https://www.python.org/downloads/release/python-380/ y se configura eclipse para usar este interprete tal y como se muestra en la **Figura 15.**

Figura 15. Instalación PyDev

La **Figura 16** muestra el proyecto para simular los sensores

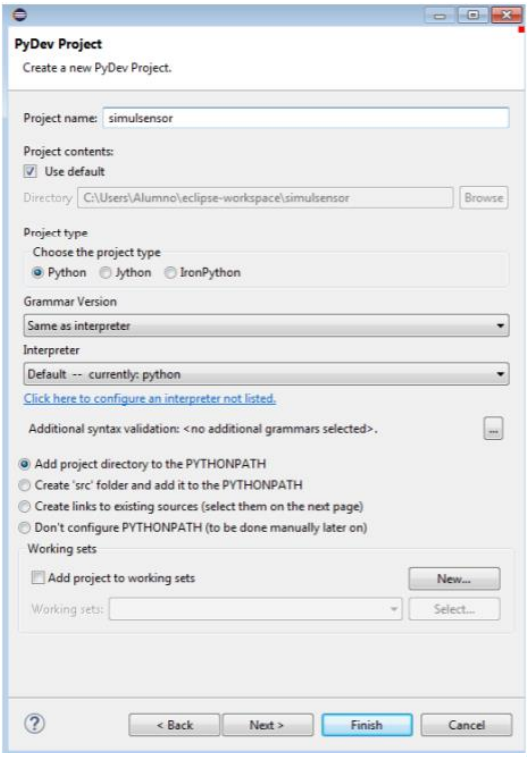

Figura 16. Proyecto simulación de sensores

Dentro del proyecto y utilizando la plantilla main se generará un nuevo módulo, *simulsensor* como se puede observar en la **Figura 17**

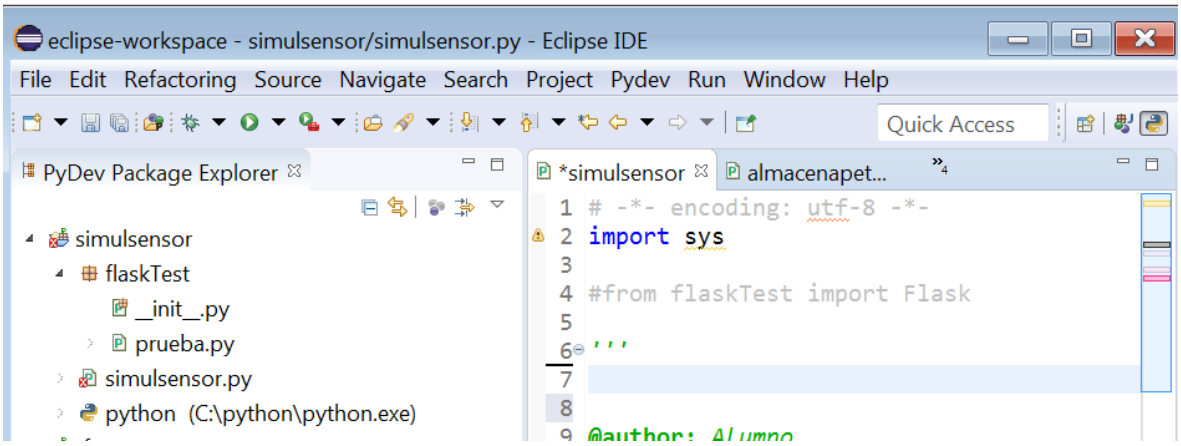

Figura 17. Creación del módulo s

El código completo del arranque del simulador de peticiones es el siguiente:

```
print (sys.version)
    print (sys.path)
    if __name__ == '__main__':

        print ("Ejemplo")
        print ("Número de sensores a simular")
        nsensores = input()
        print ("Tiempo entre peticiones (s)")
        nsecs = input();

        print ("Realizando simulación los siguientes parámetros")
        print ("nsensores: ", nsensores)
        print ("nsecs: ", nsecs
```

4.4 Instalación de Flask

Es necesario la instalación de la librería Flask para poder realizar peticiones HTTP GET en Python y simular el envío de datos de Arduino.

Para instalar y administrar paquetes de software o librerías de Python hay que utilizar el sistema de gestión de paquetes pip.

En https://www.liquidweb.com/kb/install-pip-windows/, Se descarga el get-pip.py y se instala dentro de c:\python\Scripts y desde este directorio, utilizando la herramienta pip se instala flask, tal y como se puede observar en la **Figura 18.**

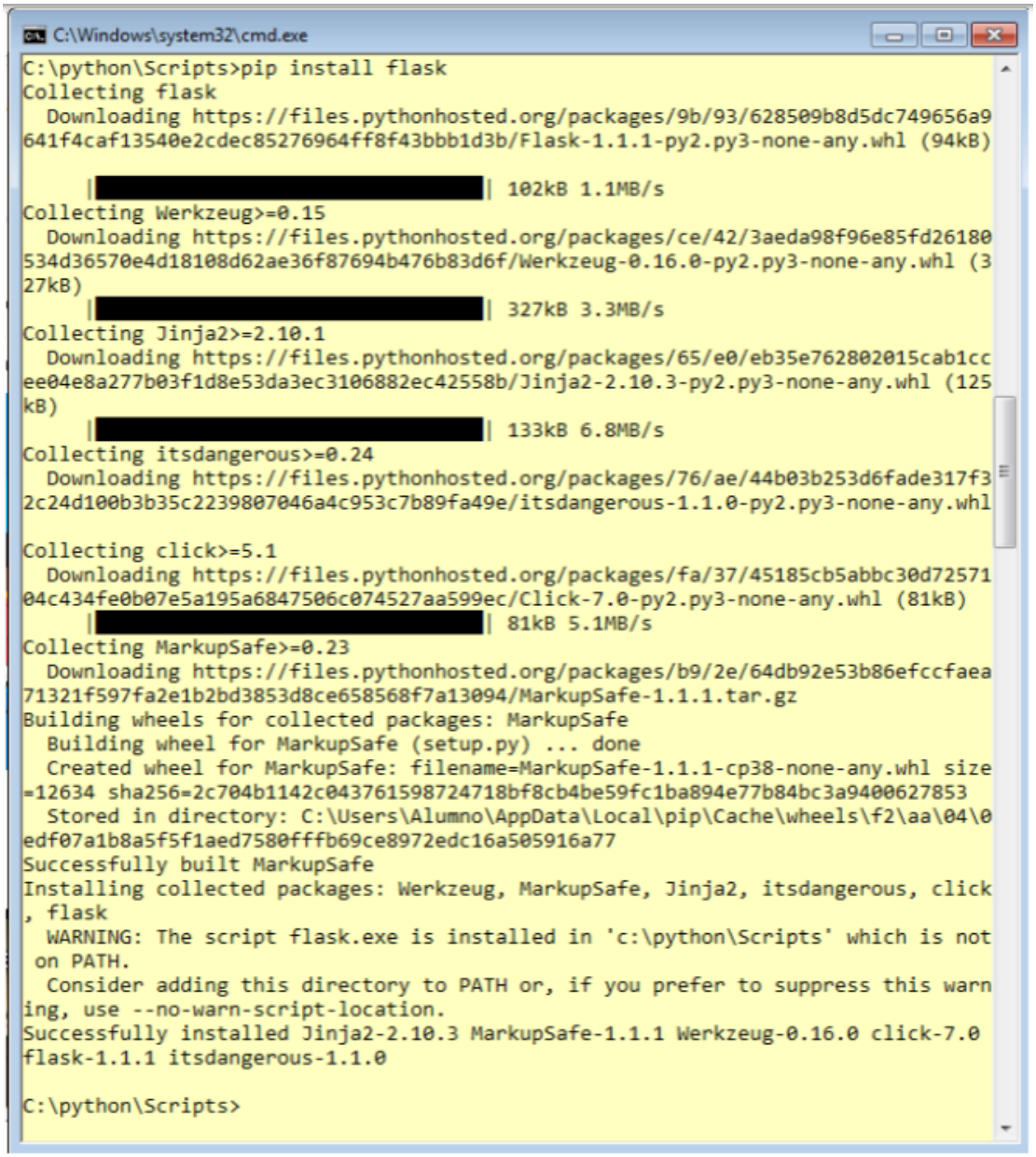

Figura 18. Instalación FLASK

La librería FLASK sirve tanto para realizar peticiones HTTP como para realizar un servidor que reciba las peticiones HTTP.

En el siguiente ejemplo, se probará a realizar un servidor que escuche peticiones. https://www.bogotobogo.com/python/python-REST-API-Http-Requests-for-Humans-withFlask.php, para ello se genera un nuevo paquete al que llamaremos flaskTest y un módulo nuevo al que llamaremos Prueba.

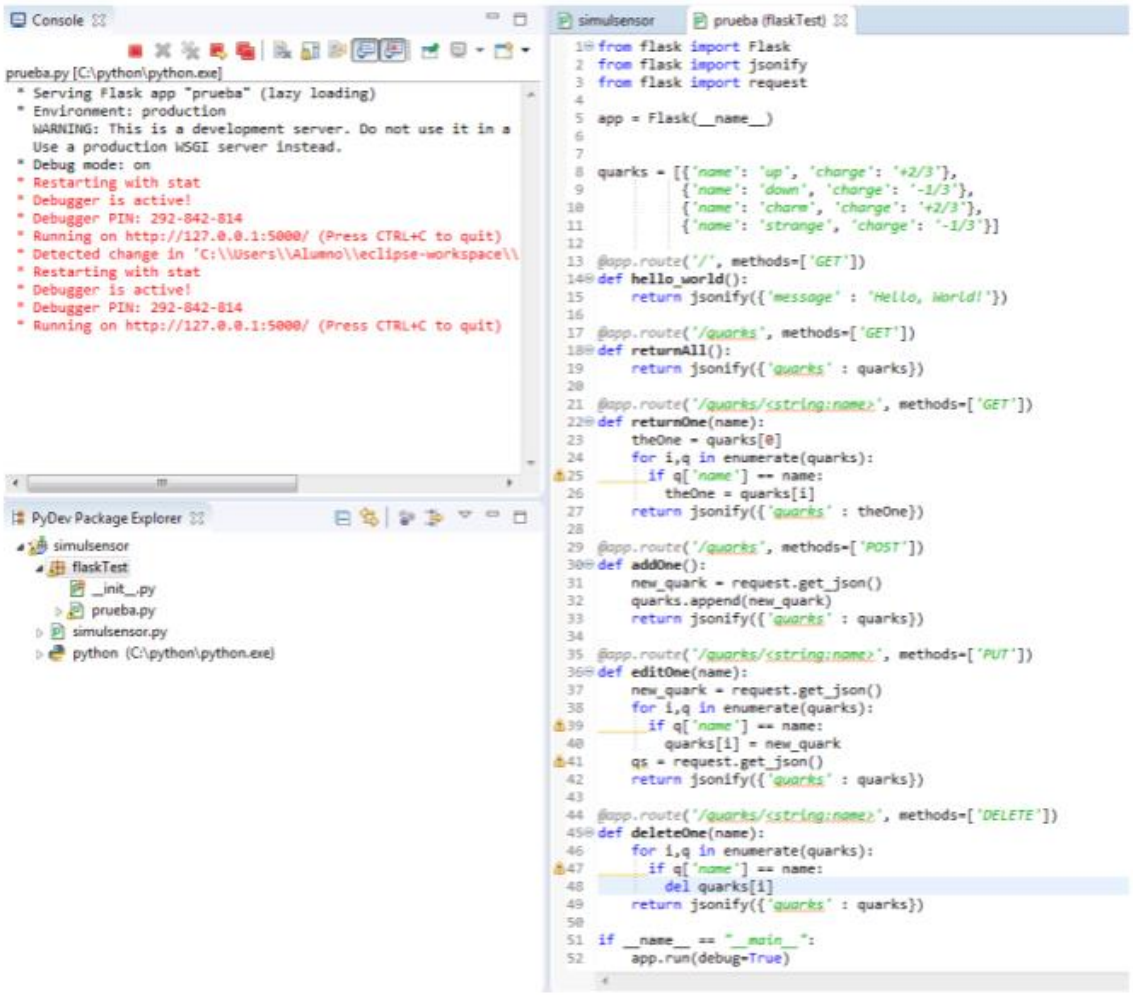

Figura 19. Ejemplo escucha de peticiones

4.5 Programación del lanzador de peticiones

En los apartados anteriores, se ha instalado y probado las librerías necesarias para poder realizar el trabajo. En este apartado se procederá a la realización del proyecto al que a partir de ahora llamaremos *TFG*.
Se crearán dos paquetes, un paquete al que llamaremos "*simulador*" y será el responsable de simular las peticiones realizadas por los sensores y un segundo paquete, denominado "*sensoserver*" que se encargará de escuchar las peticiones HTTP GET que han sido lanzadas por el Arduino.

La razón principal de la creación de estos paquetes, es porque normalmente estos programas se deben ejecutar desde ordenadores diferentes, el cliente, que realizará las peticiones y el servidor que las capturará y almacenará en la base de datos. Para la realización de este trabajo el cliente y el servidor están alojados en la misma máquina virtual y por lo tanto se utiliza localhost para realizar las peticiones tanto del cliente como del servidor. En el caso de que estuvieran en máquinas diferentes se debería cambiar ese parámetro por las direcciones IP del cliente y del servidor respectivamente.

Se instala la librería request para hacer peticiones HTTP GET como se puede observar en la **Figura 20**

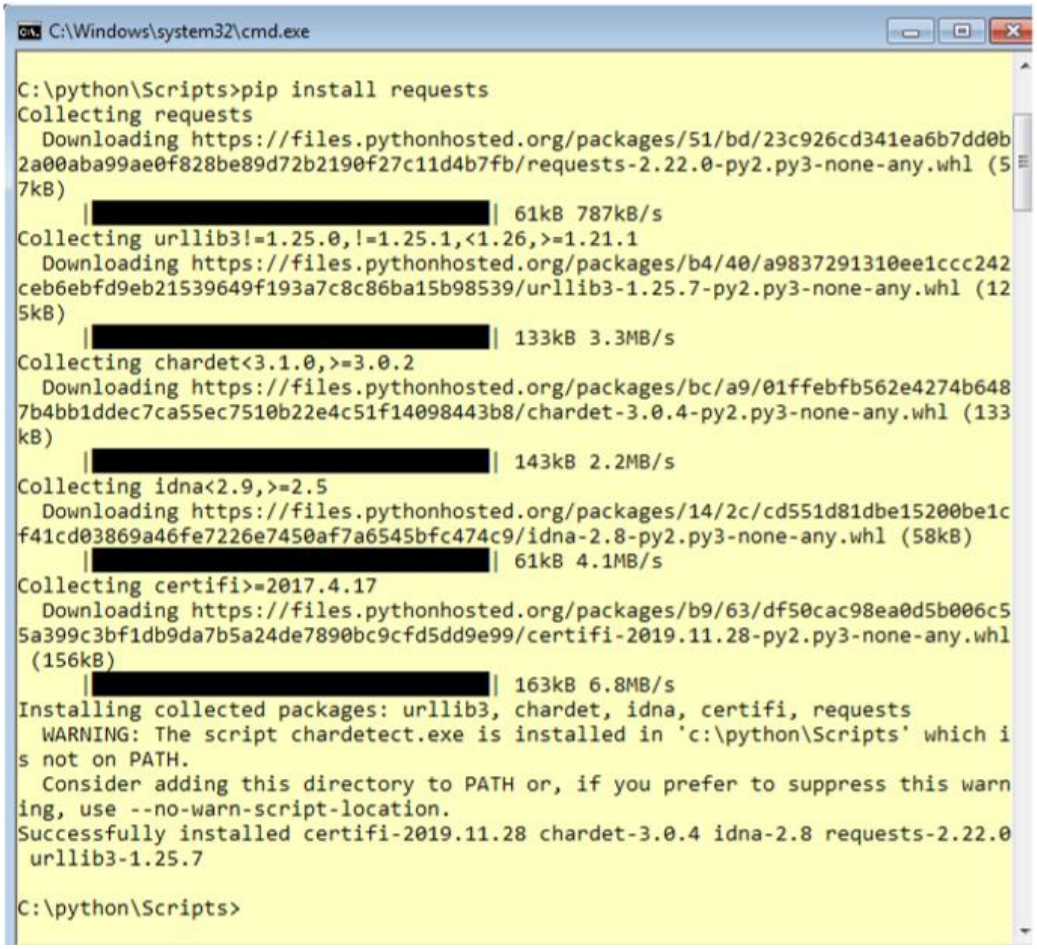

Figura 20. Instalación librería request

Y se simulan tres sensores con valores de temperatura, calidad de aire y humedad como se puede observar en las líneas 20 a 28 de la Figura 21, que en el caso de que se quisiera simular más sensores añadiríamos más líneas en este trozo del programa.

El lanzador envía una petición HTTP GET por cada línea de las referenciadas anteriormente y las repite utilizando los dos parámetros siguientes, que se preguntaran al inicio del programa: *nciclos:* número de veces que se repetirá el proceso y *nsec* el tiempo entre cada petición.

En nuestro ejemplo (**Figura 21**), se ejecuta cada 60 segundos entre peticiones y durante dos repeticiones. Durante el tiempo que duren las repeticiones los valores de los sensores se moverán entre *min y max.*

simulsensor | prueba (flaskTest) | simulador | sensorserver | *lanzapeticiones

```
1  # -*- encoding: utf-8 -*-
2
3  '''
4
5
6
7  '''
8
9  import requests
10 import time
11 import random
12
13 nciclos = -1
14 nsecs = -1
15
16 #valores minimos y maximos sobre los que
17 #se moverán las peticiones
18
19 sensores = [
20     {'id' : 1, 'nombre': 'tmp', 'min': 10.0, 'max': 15.0}, \
21     {'id' : 1, 'nombre': 'hum', 'min': 80.0, 'max': 90.0}, \
22     {'id' : 1, 'nombre': 'mq135', 'min': 40.0, 'max': 60.0}, \
23     {'id' : 2, 'nombre': 'tmp', 'min': 21.0, 'max': 24.0}, \
24     {'id' : 2, 'nombre': 'hum', 'min': 85.0, 'max': 90.0}, \
25     {'id' : 2, 'nombre': 'mq135', 'min': 45.0, 'max': 65.0}, \
26     {'id' : 3, 'nombre': 'tmp', 'min': 20.0, 'max': 28.0}, \
27     {'id' : 3, 'nombre': 'hum', 'min': 70.0, 'max': 80.0}, \
28     {'id' : 3, 'nombre': 'mq135', 'min': 30.0, 'max': 40.0}, \
29     {'id' : 4, 'nombre': 'tmp', 'min': 15.0, 'max': 25.0}, \
30     {'id' : 4, 'nombre': 'hum', 'min': 70.0, 'max': 80.0}, \
```

Figura 21. Código lanzador de peticiones.

El código completo del lanzador de peticiones es el siguiente:

```
import requests
import time
import random

nciclos = -1
nsecs = -1

#valores minimos y maximos sobre los que se moverán las peticiones

sensores = [
    {'id' : 1, 'nombre': 'tmp', 'min': 10.0, 'max': 15.0}, \
    {'id' : 1, 'nombre': 'hum', 'min': 80.0, 'max': 90.0}, \
    {'id' : 1, 'nombre': 'mq135', 'min': 40.0, 'max': 60.0}, \
```

```
    {'id' : 2, 'nombre': 'tmp', 'min': 21.0, 'max': 24.0}, \
    {'id' : 2, 'nombre': 'hum', 'min': 85.0, 'max': 90.0}, \
    {'id' : 2, 'nombre': 'mq135', 'min': 45.0, 'max': 65.0}, \
    {'id' : 3, 'nombre': 'tmp', 'min': 20.0, 'max': 28.0}, \
    {'id' : 3, 'nombre': 'hum', 'min': 70.0, 'max': 80.0}, \
    {'id' : 3, 'nombre': 'mq135', 'min': 30.0, 'max': 40.0}, \
    {'id' : 4, 'nombre': 'tmp', 'min': 15.0, 'max': 25.0}, \
    {'id' : 4, 'nombre': 'hum', 'min': 70.0, 'max': 80.0}, \
    {'id' : 4, 'nombre': 'mq135', 'min': 100.0, 'max': 500.0}, \
    {'id' : 5, 'nombre': 'tmp', 'min': 15.0, 'max': 20.0}, \
    {'id' : 5, 'nombre': 'hum', 'min': 75.0, 'max': 90.0}, \
    {'id' : 5, 'nombre': 'mq135', 'min': 100.0, 'max': 200.0}, \
    {'id' : 6, 'nombre': 'tmp', 'min': 20.0, 'max': 30.0}, \
    {'id' : 6, 'nombre': 'hum', 'min': 85.0, 'max': 90.0}, \
    {'id' : 6, 'nombre': 'mq135', 'min': 150.0, 'max': 200.0}, \
    {'id' : 7, 'nombre': 'tmp', 'min': 5.0, 'max': 20.0}, \
    {'id' : 7, 'nombre': 'hum', 'min': 70.0, 'max': 90.0}, \
    {'id' : 7, 'nombre': 'mq135', 'min': 100.0, 'max': 300.0}, \
    ]

#url base del servidor
#url = "http://webhook.site/4c354ec6-332c-4aae-a5c2-019a9a76e961"
url = "http://127.0.0.1:5000"

#cada nsecs se realizarán tres peticiones HTTP GET (tmp, humedad, calidad aire)
#a cada uno de los nsensores.

# Ejemplo peticion sensor humedad
# http://webhook.site/c806de18-6116-41c2-ba9a-2c02d070bf5f?
#   operacion=nuevodato&nombre=hum&valor=50&idsensor=10&key=165000

def cogeEntero(num):
    res = -1
    try:
        res = int(num)
        if res <= 0:
            res = -1
    except ValueError:
        pass
    if res == -1:
        print ("Introduce un número entero positivo")
    return res

def datosEntrada():
    global nsecs
    global nciclos

    while nsecs == -1:
        print ("Tiempo entre peticiones (s)")
        nsecs = cogeEntero(input())

    while nciclos == -1:
        print ("Número de repeticiones")
        nciclos = cogeEntero(input())

    print ("Realizando simulación los siguientes parámetros")
    print ("nsecs: ", nsecs)

def lanzaPeticion (nombre, valor, idsensor):
    global url
```

```
    #calcula key en funcion del nombre, valor e id
    key = 0
    for ch in nombre:
        key += ord(ch)

    key *= int (valor) * idsensor

    httpget = url + "?operacion=nuevodato&nombre=" + nombre + "&valor=" + '%.1f'
% valor + "&idsensor=" + str (idsensor) + "&key=" + str(key)
    print ("Peticion " + httpget)
    response = requests.get (httpget)
    print (response.status_code)

def esperaCiclo ():
    global nsecs
    rsecs = random.randint (0, 1000) / 1000.0

    time.sleep (nsecs + rsecs)

if __name__ == '__main__':

    datosEntrada()
```

Utilizando el webhook.site se observa el correcto funcionamiento de las peticiones.

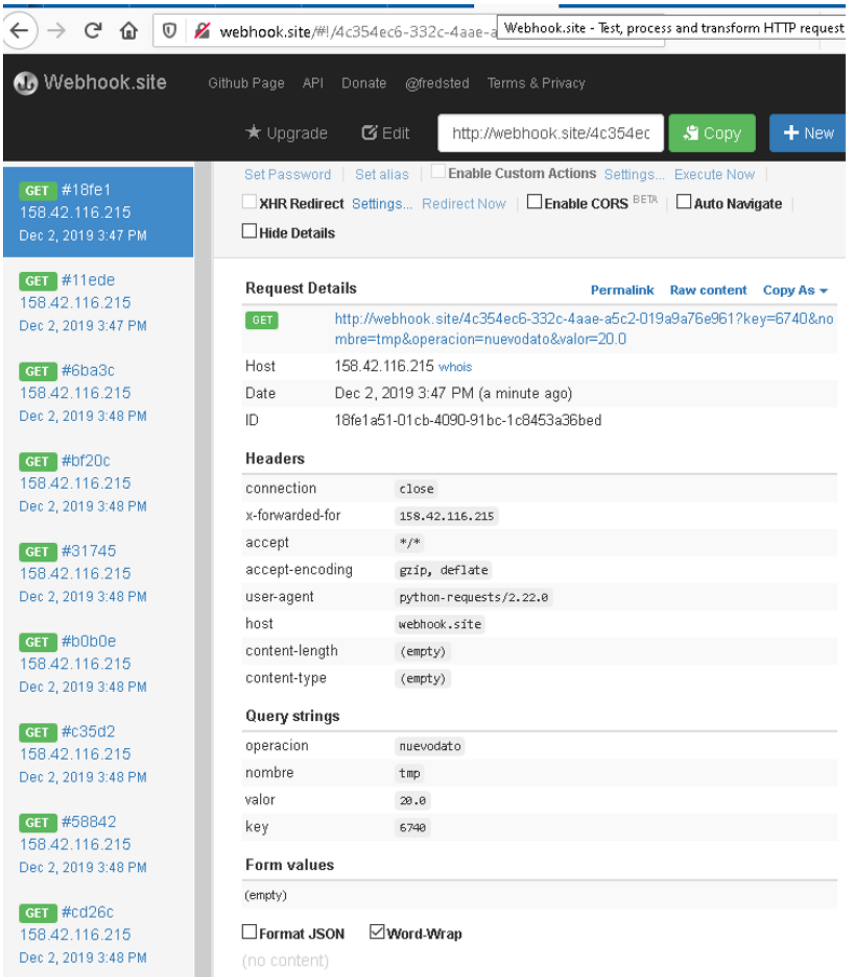

Figura 22. Peticiones que genera el lanzador

4.6 Creación del modelo de datos

En este apartado se va a implementar el servidor de Python, el cual recogerá las peticiones y las almacenará en el modelo de datos.

El modelo de datos de PostgreSQL, donde se almacenarán los datos capturados de los sensores, debe de crearse antes de empezar a programar el servidor. Para ello, se utiliza el cliente SQL pgadmin4, se crea la base de datos a la que llamaremos TFG que se convertirá a una base de datos espacial utilizando la extensión de PostGIS como se puede observar en la **Figura 23.**

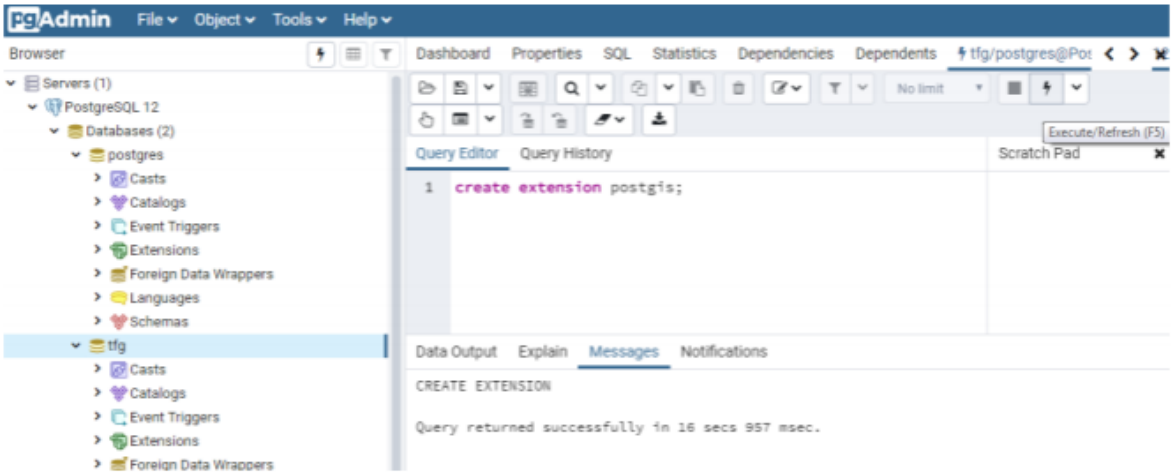

Figura 23. Creación modelo de datos.

Dentro de la BDE, se procede a generar el modelo de datos que constará de dos tablas, una tabla espacial de geometrías puntuales denominada **sensores**, con la siguiente estructura: un identificador entero como clave primaria, el nombre del sensor, la dirección url, que por defecto será http://www.upv.es, un campo booleano para saber si se encuentra activo o no y la geometría del punto para almacenar su localización.

La segunda tabla, una tabla alfanumérica de **datos** con la siguiente estructura, un gid serial como clave primaria, el identificador del sensor con clave ajena, el tipo de sensor del que se trata con una restricción (humedad, temperatura y mq135), los datos que recibimos y el tiempo en el que se recibe esta información.
La siguiente **Figura 24** muestra la creación de las tablas, con las restricciones y el indexado.

tfg/postgres@PostgreSQL 12

Query Editor | Query History

```
create table sensores (
  id integer primary key,
  nombre varchar not null,
  url varchar default 'http://www.upv.es',
  activo boolean not null,
  geom geometry (point, 4326) not null
);

create index idx_geom_sensores on sensores using gist(geom);


create table datos (
  gid serial primary key,
  idsensor integer not null,
  tiposensor varchar constraint r_sensor_tipo check (tiposensor in ('tmp','hum','mq135')),
  dato float not null,
  tiempo timestamp not null
);

create index idx_idsensor_datos on datos (idsensor);
create index idx_tiposensor_datos on datos (tiposensor);
create index idx_tiempo_datos on datos (tiempo);

alter table datos add constraint fk_datossensores foreign key (idsensor)
  references sensores (id) on update cascade on delete set null;
```

Figura 24. Creación de tablas

Se introducen valores en la tabla sensores de 7 sensores ficticios como se puede ver en la **Figura 25**.

```
insert into sensores (id,nombre,activo,geom)
  values (1, 'ETSIGCT', true, 'SRID=4326;POINT (-0.337769 39.481353)'::geometry);
insert into sensores (id,nombre,activo,geom)
  values (2, 'CPI', true, 'SRID=4326;POINT (-0.334330 39.477740)'::geometry);
insert into sensores (id,nombre,activo,geom)
  values (3, 'PISCINA', true, 'SRID=4326;POINT (-0.341124 39.480063)'::geometry);
insert into sensores (id,nombre,activo,geom)
  values (4, 'CFP', true, 'SRID=4326;POINT (-0.338630 39.479108)'::geometry);
insert into sensores (id,nombre,activo,geom)
  values (5, 'ASIC', true, 'SRID=4326;POINT (-0.338878 39.480587)'::geometry);
insert into sensores (id,nombre,activo,geom)
  values (6, 'CASA ALUMNO', true, 'SRID=4326;POINT (-0.341631 39.480746)'::geometry);
insert into sensores (id,nombre,activo,geom)
  values (7, 'TRINQUET EL GENOVES', true, 'SRID=4326;POINT (-0.336359 39.479877)'::geometry);
```

Figura 25. Introducción de datos de los sensores

Una vez creado el modelo de datos, el lanzador de peticiones enviará las peticiones de los 7 sensores ficticios que hemos creado para que se almacenen en la tabla **datos.**

Se ha definido una restricción de clave ajena en la tabla de datos, por lo tanto, se deberá tener en cuenta que, si el sensor no existe previamente en la tabla **sensores**, no se podrá almacenar nada en la tabla **datos**.

4.7 Programación del servidor

En este apartado se procede a programar un servidor que Python cuya función será capturar las peticiones que realizan los sensores de Arduino, en nuestro caso, el simulador de peticiones que realiza de forma ficticia este trabajo, y almacenarlas en el modelo de datos de PostgreSQL que se ha creado previamente.

Es importante instalar en Python la librería de acceso a PostgreSQL psycog2 para poder realizar este proceso correctamente.

Crearemos **almacenapeticiones.py**, este programa pondrá un servidor http en el puerto 5000 y será el encargado de escuchar las peticiones HTTP realizadas desde el simulador que se ha creado anteriormente en **lanzapeticiones.py** o en su defecto desde el Arduino.

Una vez creado, se procederá a cambiar la url de envío de peticiones que hasta el momento seguía siendo el servidor de prueba, por nuestro nuevo servidor como podemos observar en la **Figura 26.**

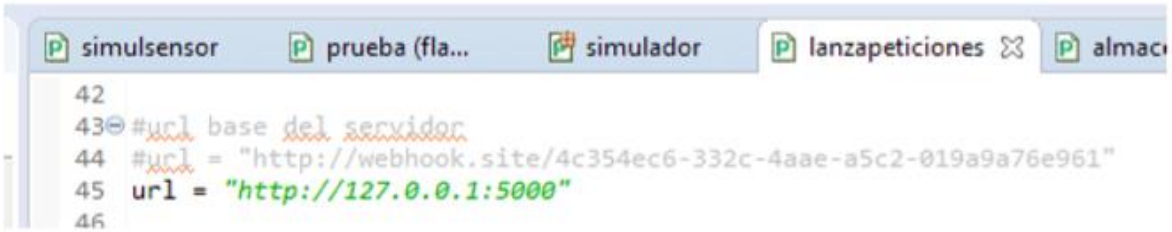

Figura 26. Cambio a la URL del servidor

El código completo del almacenador de peticiones es el siguiente:

```
from flask import Flask
from flask import request
from flask import Response
import psycopg2
import sys
from datetime import datetime

tipoSensores = ['tmp','hum','mq135']
conn = None
cur = None

app = Flask(__name__)

@app.route('/', methods=['GET'])
def cogeParametros():
    valor = 0.0
    idsensor = 0
    key = 0

    errortxt = None

    #Coge los parámetros de la peticion HTTP GET y
    #comprueba que son del tipo adecuado
    operacion = request.args.get('operacion')

    if operacion is None or operacion != 'nuevodato':
        errortxt = 'Error, operación desconocida'
        return Response (errortxt, mimetype='text/html', status=500)
```

```
idsensorstr = request.args.get('idsensor')
if idsensorstr is None:
    errortxt ='Error, id de sensor desconocido'
    return Response (errortxt, mimetype='text/html', status=500)
else:
    try:
        idsensor = int(idsensorstr)
    except ValueError:
        errortxt = 'Error, id de sensor no válido'
        return Response (errortxt, mimetype='text/html', status=500)

nombre = request.args.get('nombre')
if nombre is None or nombre not in tipoSensores:
    errortxt = 'Error, tipo de sensor desconocido ' + nombre
    return Response (errortxt, mimetype='text/html', status=500)

valorstr = request.args.get('valor')
if valorstr is None:
    errortxt = 'Error, valor desconocido'
    return Response (errortxt, mimetype='text/html', status=500)
else:
    try:
        valor = float(valorstr)
    except ValueError:
        errortxt = 'Error, valor no válido'
        return Response (errortxt, mimetype='text/html', status=500)

keystr = request.args.get('key')
if keystr is None:
    errortxt  = 'Error,  key no válida'
    return Response (errortxt, mimetype='text/html', status=500)
else:
    try:
        key = int(keystr)
    except ValueError:
        errortxt  = 'Error, key no válida'
        return Response (errortxt, mimetype='text/html', status=500)


#calcula la key y comprueba si coincide
keycal = 0
for ch in nombre:
    keycal += ord(ch)

keycal *= int (valor) * idsensor

if keycal != key:
    errortxt = 'Error, key no coincide'
    return Response (errortxt, mimetype='text/html', status=500)

#Ahora, imprimimos los valores como comprobación
print ("idsensor " + str(idsensor))
print ("nombre " + nombre)
print ("valor " + str(valor))

#La petición es correcta procede a almacenarla en la bdd
errortxt = almacena (idsensor, nombre, valor)

if errortxt is None:
```

```
        return Response ("Petición procesada correctamente",
mimetype='text/html', status=200)
    else:
        return Response (errortxt, mimetype='text/html', status=500)

#requiere una bdd con nombre golsa, usuario postgres y su contrasena
def conexionBDD ():
    global conn
    global cur

    conn = None
    try:
        conn=psycopg2.connect("dbname='TFG' user='postgres' password='pg'
port='5432'")
        cur = conn.cursor()
        conn.set_isolation_level(0)

    except psycopg2.Error as e:
        print (e)
        print ("No puedo crear la conexion a la BDD. ABORTANDO")
        sys.exit()

def almacena (idsensor, nombre, valor):
    global cur

    now = datetime.now()
    #timestamp = datetime.timestamp(now)

    try:
        cur.execute("INSERT INTO datos (idsensor, tiposensor, dato, tiempo)
VALUES (%s,%s, %s,%s)",
                    (idsensor, nombre, valor, now))
    except:
        return  "SQL Error:" + sys.exc_info()[0]

    return None


#Text con:
http://localhost:5000/?operacion=nuevodato&nombre=mq135&valor=30.0&idsensor=3&ke
y=33750
if __name__ == '__main__':
    conexionBDD ()
    #app.run(debug=True)
    app.run(debug=False, threaded=True)
```

4.8 Análisis de datos y visualización en un SIG

Una vez se obtienen las tablas con los diferentes datos registrados se puede proceder al análisis de los mismos.

En primer lugar, se procede a la consulta alfanumérica de los datos que hemos obtenido de los sensores. Un ejemplo de esto sería poder calcular las temperaturas mínimas, máximas o medias en el periodo de tiempo que determinemos.

Esta consulta se puede realizar tanto con los datos que se están almacenando en el momento, usando el comando now() como podemos ver en la **Figura 27.**

```
select d.idsensor, min(d.dato) as mintmp, max(d.dato) as maxtmp,
       avg(d.dato)::numeric (5,2) as avgtmp, count(*) as nmedidas,
       max(d.tiempo) as tiempo
from datos d
where d.tiposensor='tmp'
  and d.tiempo between (now() - '5 minute'::interval) and now()
  group by d.idsensor
```

Data Output | Explain | Messages | Notifications

	idsensor integer	mintmp double precision	maxtmp double precision	avgtmp numeric (5,2)	nmedidas bigint	tiempo timestamp without time zone
1	7	17.7	17.9	17.80	4	2019-12-03 19:26:26.796893
2	1	24.2	24.3	24.25	4	2019-12-03 19:26:22.459645
3	5	19.2	19.3	19.25	4	2019-12-03 19:26:25.310808
4	4	23.4	23.6	23.53	4	2019-12-03 19:26:24.435758
5	2	23.5	23.6	23.58	4	2019-12-03 19:26:23.086681
6	6	18.4	18.6	18.53	4	2019-12-03 19:26:25.988847
7	3	26.8	26.9	26.83	4	2019-12-03 19:26:24.084738

Figura 27. Almacenamiento de datos en el momento.

Sí el almacenamiento ya se ha parado, se puede realizar una consulta cambiando la variable now() por el tiempo del último registro y entonces obtendríamos la selección de los datos con los últimos 5 minutos desde la fecha más reciente en que se almacenó el último dato como se puede observar en la **Figura 28**.

```
with ultimoregistro as (
 select tiempo from datos where tiposensor = 'tmp' order by tiempo desc limit 1
)
select d.idsensor, min(d.dato) as mintmp, max(d.dato) as maxtmp,
       avg(d.dato)::numeric (5,2) as avgtmp, count(*) as nmedidas,
       max(d.tiempo) as tiempo
from datos d, ultimoregistro t
where d.tiposensor='tmp'
  and d.tiempo between (t.tiempo - '5 minute'::interval) and t.tiempo
  group by d.idsensor
```

Data Output | Explain | Messages | Notifications

	idsensor integer	mintmp double precision	maxtmp double precision	avgtmp numeric (5,2)	nmedidas bigint	tiempo timestamp without time zone
1	7	18	18.3	18.16	5	2019-12-03 19:31:54.602643
2	1	24.3	24.4	24.38	5	2019-12-03 19:31:49.614358
3	5	19.3	19.4	19.38	5	2019-12-03 19:31:53.33357
4	4	23.7	23.9	23.78	5	2019-12-03 19:31:52.365515
5	2	23.6	23.7	23.62	5	2019-12-03 19:31:50.973435
6	6	18.7	18.9	18.78	5	2019-12-03 19:31:53.795597

Figura 28. Almacenamiento de datos en un momento determinado.

Si realizamos consultas, se obtienen los datos por pantalla, pero si se necesita almacenar la información que se quiere obtener en tablas, cómo es nuestro caso, en lugar de realizar la consulta, vamos a trabajar con vistas, se puede observar esto en la **Figura 29**.

```
create view tmp_5min as
with ultimoregistro as (
 select tiempo from datos where tiposensor = 'tmp' order by tiempo desc limit 1
)
select d.idsensor, min(d.dato) as mintmp, max(d.dato) as maxtmp,
       avg(d.dato)::numeric (5,2) as avgtmp, count(*) as nmedidas,
       max(d.tiempo) as tiempo
from datos d, ultimoregistro t
where d.tiposensor='tmp'
  and d.tiempo between (t.tiempo - '5 minute'::interval) and t.tiempo
  group by d.idsensor
```

Figura 29. Creación de las vistas.

De esta forma, la vista almacena el resultado en un fichero y posteriormente se podrá visualizar en un sistema de información geográfica. En la **Figura 30** se pueden visualizar los datos obtenidos en los últimos 5 minutos y que se han almacenado en la vista SQL tmp_5min.

```
22  select * from tmp_5min
```

Data Output | Explain | Messages | Notifications

	idsensor integer	mintmp double precision	maxtmp double precision	avgtmp numeric (5,2)	nmedidas bigint	tiempo timestamp without time zone
1	7	18.2	18.5	18.32	5	2019-12-03 19:34:07.812262
2	1	24.4	24.5	24.44	5	2019-12-03 19:34:02.454956
3	5	19.4	19.5	19.44	5	2019-12-03 19:34:05.489129
4	4	23.8	24	23.88	5	2019-12-03 19:34:04.62908
5	2	23.6	23.7	23.66	5	2019-12-03 19:34:03.108993
6	6	18.8	19	18.88	5	2019-12-03 19:34:06.569191
7	3	27	27.2	27.12	5	2019-12-03 19:34:04.003044

Figura 30. Datos almacenados para tiempo 5 mins

Como se ha explicado en el apartado de modelo de datos, existe una tabla en la que se almacenan los datos y otra tabla espacial con información de los sensores. La siguiente vista que se va a crear es una vista concatenando ambas tablas (**Figura 31**) y consiguiendo de este modo poder utilizar el campo que nos marca la geometria (*geom*) para crear una **vista espacial** y de esta forma se visualizará graficamente utilizando algún software cliente de SIG que se pueda conectar a PostGIS como QGIS, gvSig o ArcGIS entre otros.
En este TFG, utilizamos QGIS como cliente SIG para visualizar los datos gráficos.

```
create view tmp_point_5min as
with ultimoregistro as (
 select tiempo from datos where tiposensor = 'tmp' order by tiempo desc limit 1
)
select d.idsensor, min(d.dato) as mintmp, max(d.dato) as maxtmp,
       avg(d.dato)::numeric (5,2) as avgtmp, count(*) as nmedidas,
       max(d.tiempo) as tiempo,
       max(s.geom)::geometry (point, 4326) as geom
from datos d, sensores s, ultimoregistro t
where d.idsensor = s.id and s.activo and d.tiposensor='tmp'
  and d.tiempo between (t.tiempo- '5 minute'::interval) and t.tiempo
  group by d.idsensor
```

```
select * from tmp_point_5min
```

Data Output | Explain | Messages | Notifications

	idsensor integer	mintmp double precision	maxtmp double precision	avgtmp numeric (5,2)	nmedidas bigint	tiempo timestamp without time zone	geom geometry
1	7	18.2	18.6	18.40	5	2019-12-03 19:35:14.21006	0101000020E6100...
2	1	24.4	24.5	24.46	5	2019-12-03 19:35:09.508791	0101000020E6100...
3	5	19.4	19.5	19.46	5	2019-12-03 19:35:13.075995	0101000020E6100...
4	4	23.8	24.1	23.94	5	2019-12-03 19:35:12.054936	0101000020E6100...
5	2	23.6	23.7	23.68	5	2019-12-03 19:35:10.18883	0101000020E6100...
6	6	18.8	19.1	18.94	5	2019-12-03 19:35:13.654028	0101000020E6100...
7	3	27.1	27.2	27.16	5	2019-12-03 19:35:10.881869	0101000020E6100...

Figura 31. Concatenación tablas

Para ello, se procederá a entrar en QGIS, se realizará una conexión a la base de datos espacial de PostGIS y de esta forma ya se puede acceder a las tablas que tiene el TFG y poder seleccionar la vista que hemos creado anteriormente. Se puede observar en la **Figura 32.**

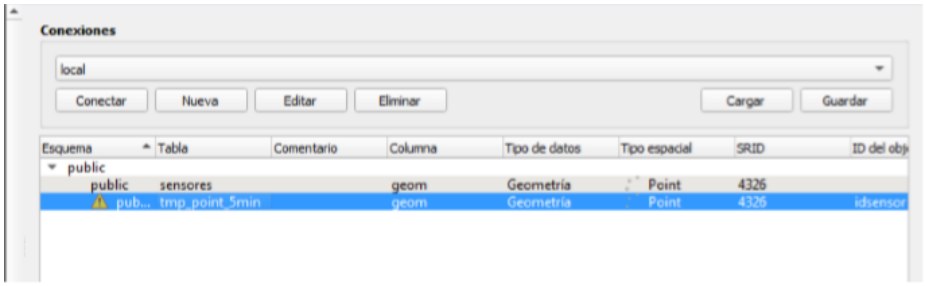

Figura 32. Selección de vistas

Cogemos la capa, abrimos las propiedades y elegimos como etiquetas de los puntos, etiquetas sencillas, queremos poder visualizar el sensor con su temperatura media que se ha calculado en la vista, para ello en expresión ponemos concat(idsensor, ':', avgtmp) y añadimos un buffer blanco alrededor de la etiqueta, tal y como se observa en la **Figura 33.**

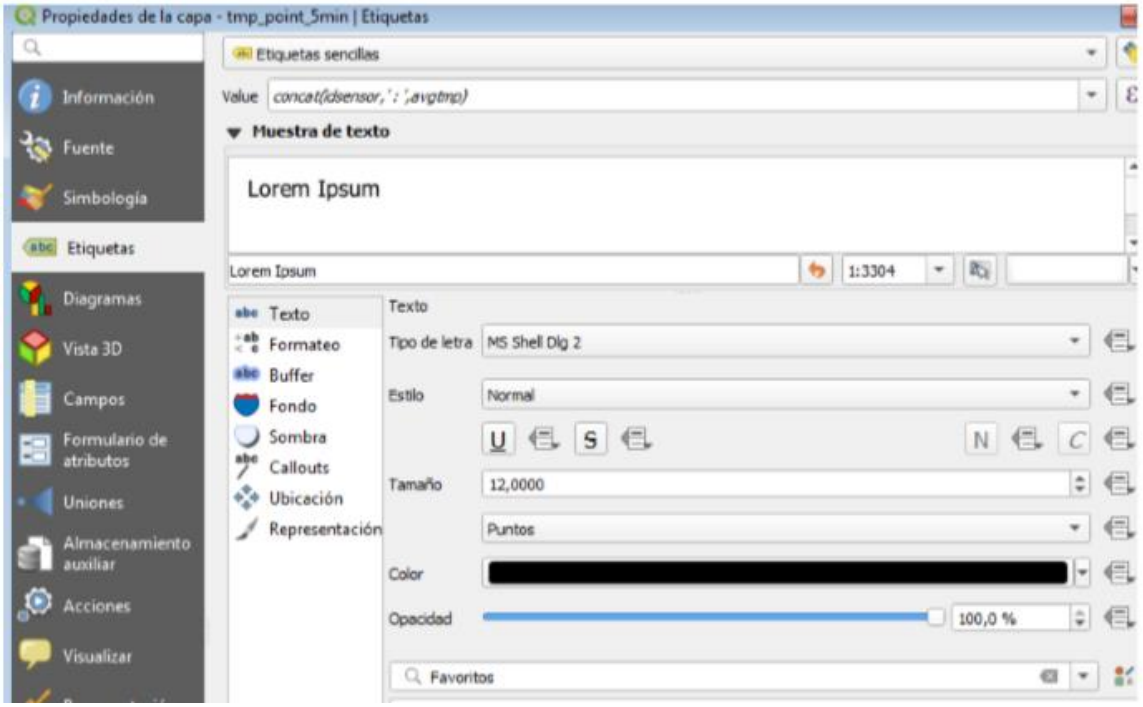

Figura 33. Edición propiedades de capa

Una vez realizado este proceso podemos observar en QGIS la capa con los textos de identificador y temperatura media que se han registrado en los últimos 5 minutos.

Para poder ver estos datos sobre la imagen de las respectivas zonas a las que hace referencia en la UPV tendremos que cargar su respectiva capa WMS que sacaremos del PNOA de la página web del IGN. http://www.ign.es/wms-inspire/pnoa-ma como podemos ver en la **Figura 34.**

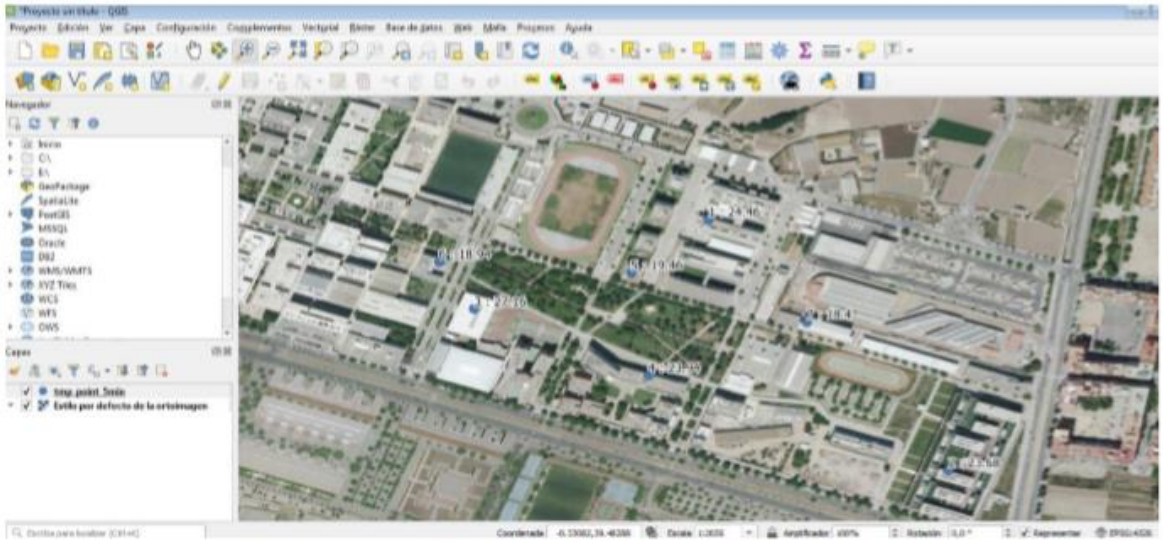

Figura 34. Capa WMS de PNOA

Esta capa se trata de una capa viva, lo que quiere decir es que, tras unos minutos los datos se irán actualizando automáticamente, también al moverse por el mapa, estos irán actualizándose con nuestro movimiento.

Uno de los objetivos de este trabajo de fin de grado es, al igual se ha podido obtener la foto anterior del PNOA mediante el estándar WMS, poner a disposición en Internet de forma estándar de WMS los datos gráficos de las estaciones meteorológicas para que cualquier persona, pueda cargar actualizados al instante utilizando cualquier software.

Para realizar esto, se utilizará el servidor de datos espaciales Geoserver, que utilizando capas de PostGIS, las emitirá a través de internet con WMS.

Antes de utilizar el servidor, se van a crear varias vistas con los datos recogidos en diferentes periodos de tiempo. Se crearán, para nuestro trabajo, vistas para 1 semana, 1 hora, 6 horas y 12 horas. En la Figura 35 se muestra un ejemplo para los datos recogidos en una hora y se calculan las medias de las temperaturas recogidas.

```
28  create or replace view mq135_1hour as
29  with ultimoregistro as (
30   select tiempo from datos where tiposensor = 'mq135' order by tiempo desc limit 1
31  )
32  select d.idsensor, min(d.dato) as mintmp, max(d.dato) as maxtmp,
33         avg(d.dato)::numeric (5,2) as avgtmp, count(*) as nmedidas,
34         max(d.tiempo) as tiempo,
35         max(s.geom)::geometry (point, 4326) as geom
36  from datos d, sensores s, ultimoregistro t
37  where d.idsensor = s.id and s.activo and d.tiposensor='tmp'
38    and d.tiempo between (t.tiempo- '1 hour'::interval) and t.tiempo
39    group by d.idsensor
41  select * from mq135_1hour
```

Data Output | Explain | Messages | Notifications

	idsensor integer	mintmp double precision	maxtmp double precision	avgtmp numeric (5,2)	nmedidas bigint	tiempo timestamp without time zone	geom geometry
1	7	14.1	18.6	16.33	55	2019-12-03 19:35:14.21006	0101000020E6100...
2	1	23	24.5	23.77	54	2019-12-03 19:35:09.508791	0101000020E6100...
3	5	18	19.5	18.77	54	2019-12-03 19:35:13.075995	0101000020E6100...
4	4	21.1	24.1	22.56	55	2019-12-03 19:35:12.054936	0101000020E6100...
5	2	22.8	23.7	23.28	53	2019-12-03 19:35:10.18883	0101000020E6100...
6	6	16.1	19.1	17.56	55	2019-12-03 19:35:13.654028	0101000020E6100...
7	3	24.8	27.2	26.05	52	2019-12-03 19:35:10.881869	0101000020E6100...

Figura 35. Datos recogidos en una hora.

4.9 Geoserver

Para poder instalar el servidor de datos espaciales Geoserver, es necesario instalar como servidor y contenedor de Geoserver: Apache tomcat y es necesario tener una máquina virtual de java instalada en el sistema operativo. El Apache Tomcat es un contenedor de *servlets* que son lo que utiliza Geoserver para servir la información a través de Internet.

Tomcat (**Figura 36**) se ejecuta como un servicio de Microsoft Windows. Para comprobar que funciona se introduce en el navegador http://localhost:8080/

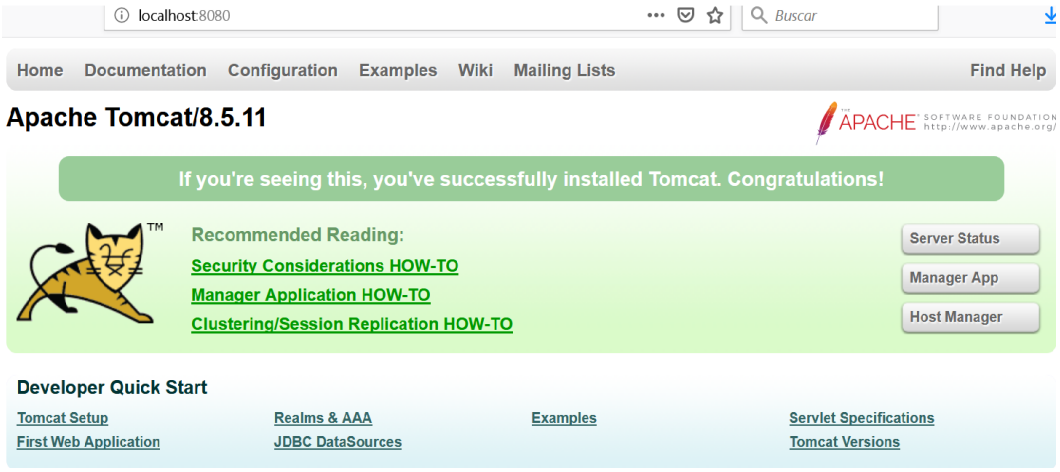

Figura 36. Captura Tomcat

Se añade un usuario a Tomcat en el fichero de configuraciones de usuarios y se cambia el tamaño de los archivos que permite desplegar, como se puede observar en la **Figura 37.**

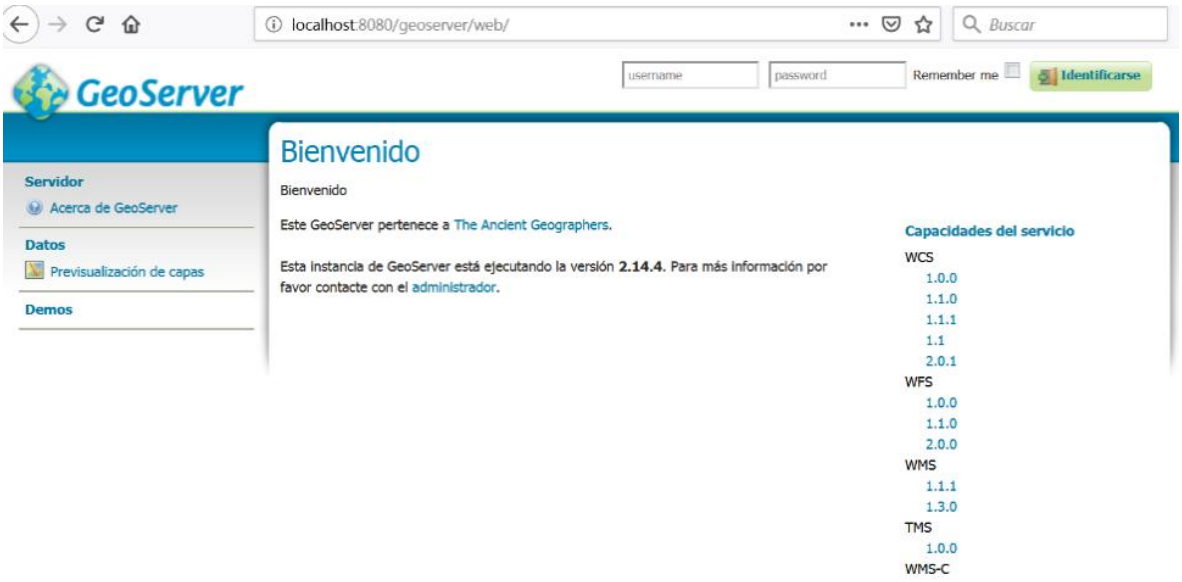

Figura 37. Configuración usuarios GeoServer

Una vez se instala el Geoserver se creará un nuevo espacio de trabajo, con el nombre TFG y se activará el servicio WMS como se puede observar en la **Figura 38.**

Figura 38. Edición espacio de trabajo

Seguidamente, se debe crear un almacén de datos vectoriales PostGIS en el anteriormente espacio de trabajo *TFG* añadiendo los parámetros de conexión que se detallan en la **Figura 39.**

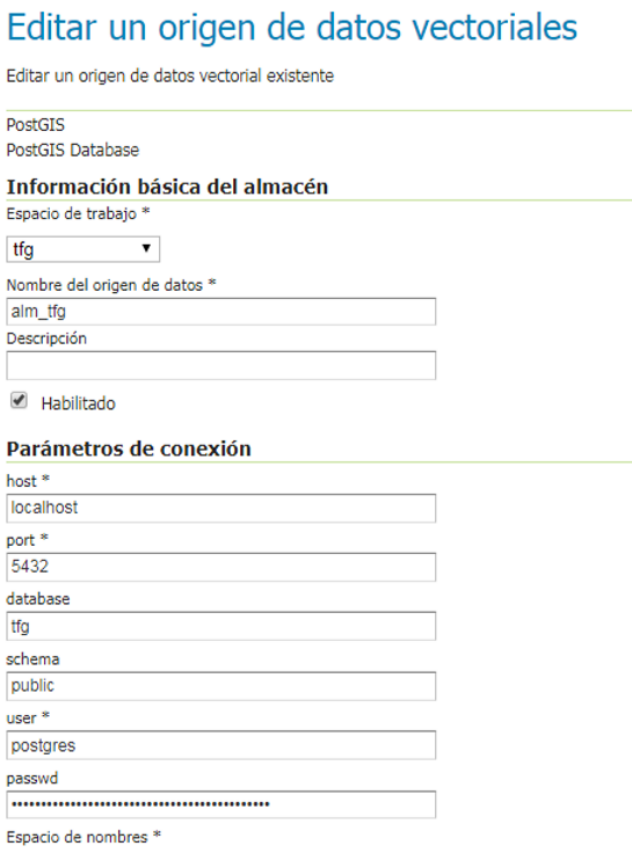

Figura 39. Editor origen datos vectoriales

Se crea una capa (**Figura 40**) con los datos que se han preparado en las vistas del apartado anterior.

Figura 40. Edición capa GeoServer

Se debe calcular el encuadre de datos y nativo. En el encuadre nativo se puede calcular la extensión de la capa y para hacerla más grande y que no solo tenga la superficie de los puntos calculados, se añade 1 segundo por lado.

Una vez terminado, se deberá rellenar también el encuadre lat/lon como podemos observar en la imagen XX y se guarda la capa como se puede observar en la **Figura 41.**

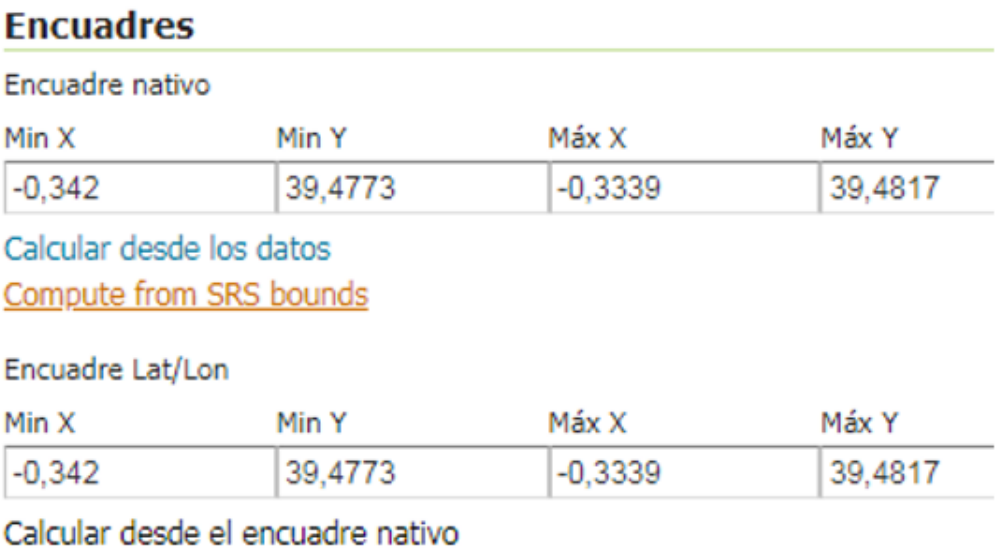

Figura 41. Encuadre de datos y nativo

Se necesita añadir la simbología. Para ello, se utiliza una simbología SLD, que es bastante parecida a la propia simbología SLD de QGIS.
Desde QGIS, se debe cambiar la simbología de etiquetas sencillas a avgtmp, ya que la simbología SLD no entiende la expresión utilizada anteriormente, concat(idsensor,' : ',avgtmp).
Se modifica la posición de los textos para que no tapen a los puntos y se guarda el estilo en formato SLD, como se muestra en la **Figura 42**.
El estilo se exporta desde QGIS y desde un editor de texto se modifica Single Symbol" a "Estaciones" para que esta aparezca posteriormente en la leyenda.

```
<?xml version="1.0" encoding="UTF-8"?>
<StyledLayerDescriptor xmlns="http://www.opengis.net/sld" xmlns:xlink="
http://www.w3.org/1999/xlink" version="1.1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
" xsi:schemaLocation="http://www.opengis.net/sld
http://schemas.opengis.net/sld/1.1.0/StyledLayerDescriptor.xsd" xmlns:ogc="
http://www.opengis.net/ogc" xmlns:se="http://www.opengis.net/se">
  <NamedLayer>
    <se:Name>tmp_point_5min</se:Name>
    <UserStyle>
      <se:Name>tmp_point_5min</se:Name>
      <se:FeatureTypeStyle>
        <se:Rule>
          <se:Name>Estaciones</se:Name>
          <se:PointSymbolizer>
            <se:Graphic>
              <se:Mark>
                <se:WellKnownName>circle</se:WellKnownName>
                <se:Fill>
                  <se:SvgParameter name="fill">#487bb6</se:SvgParameter>
                </se:Fill>
                <se:Stroke>
                  <se:SvgParameter name="stroke">#325780</se:SvgParameter>
                  <se:SvgParameter name="stroke-width">1</se:SvgParameter>
                </se:Stroke>
              </se:Mark>
              <se:Size>11</se:Size>
            </se:Graphic>
          </se:PointSymbolizer>
        </se:Rule>
        <se:Rule>
          <se:TextSymbolizer>
            <se:Label>
              <ogc:PropertyName>avgtmp</ogc:PropertyName>
            </se:Label>
            <se:Font>
              <se:SvgParameter name="font-family">MS Shell Dlg 2</se:SvgParameter>
              <se:SvgParameter name="font-size">15</se:SvgParameter>
            </se:Font>
            <se:LabelPlacement>
              <se:PointPlacement>
                <se:AnchorPoint>
                  <se:AnchorPointX>0</se:AnchorPointX>
                  <se:AnchorPointY>0.5</se:AnchorPointY>
                </se:AnchorPoint>
                <se:Displacement>
                  <se:DisplacementX>7</se:DisplacementX>
                  <se:DisplacementY>7</se:DisplacementY>
                </se:Displacement>
              </se:PointPlacement>
            </se:LabelPlacement>
            <se:Halo>
              <se:Radius>2</se:Radius>
              <se:Fill>
                <se:SvgParameter name="fill">#ffffff</se:SvgParameter>
              </se:Fill>
            </se:Halo>
            <se:Fill>
              <se:SvgParameter name="fill">#000000</se:SvgParameter>
            </se:Fill>
            <se:VendorOption name="maxDisplacement">20</se:VendorOption>
          </se:TextSymbolizer>
        </se:Rule>
      </se:FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
```

Figura 42. Cambio de simbología

Se crea en Geoserver un nuevo estilo tmp de tipo SLD en el espacio de trabajo TFG, como se muestra en la **Figura 43** y se sube el SLD de QGIS, seguidamente se valida y se aplica.

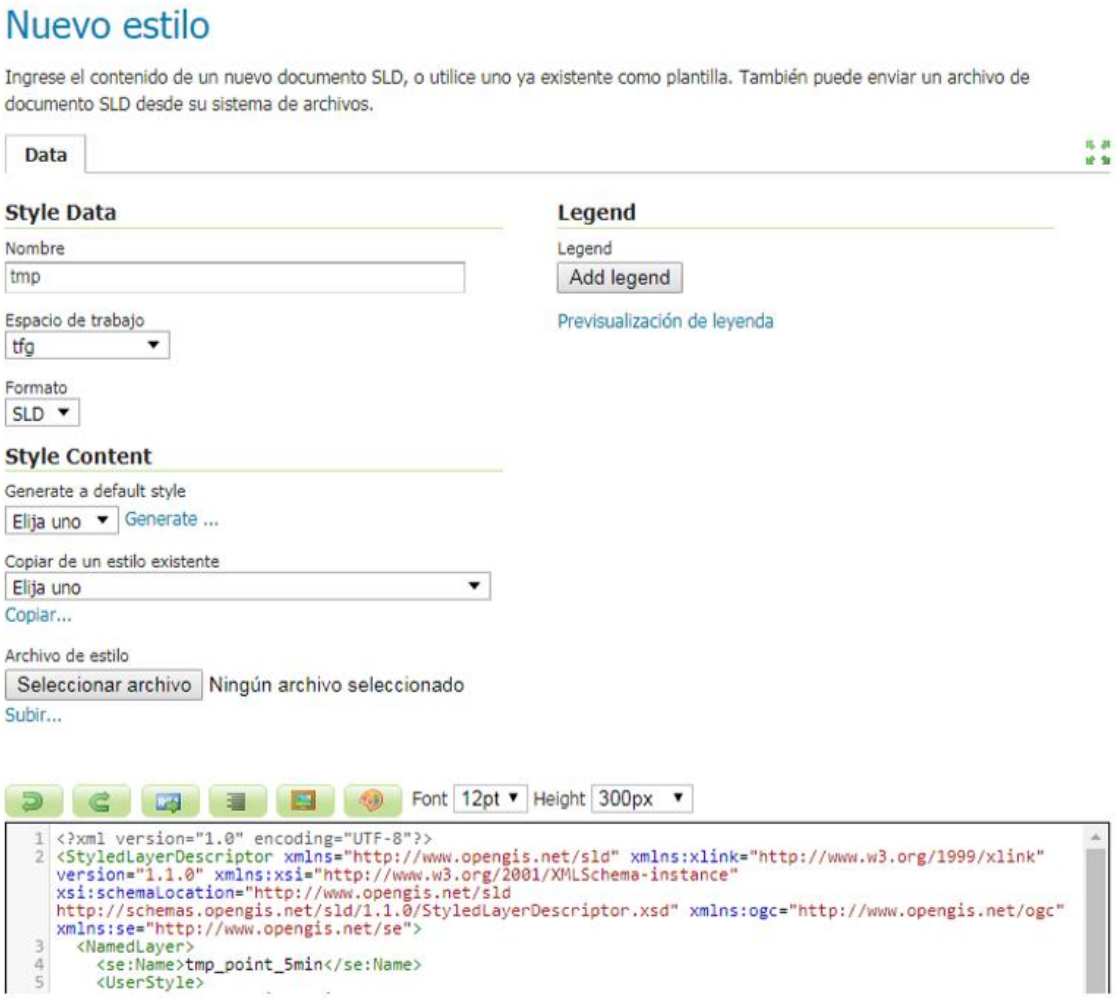

Figura 43. Creación de nuevo estilo

Por último, se marca la capa tmp_5min en las capas de Geoserver y se selecciona en la pestaña publicación el estilo tmp que se ha creado recientemente.

4.10 Servidor WMS

Una vez se ha creado el servidor WMS con la capa tmp_5min, se procede a probarlo con QGIS.
La URL del servidor será: ***http://localhost:8080/geoserver/TFG/wms*** y se ha conseguido que mediante la URL cualquier persona desde Internet tiene acceso a la información gráfica de las estaciones meteorológicas en tiempo real. Simplemente se debe cambiar localhost por la IP del servidor.

Con cualquier cliente de WMS (gvSIG, ArcGIS, QGIS o cualquier página web que ofrezca un cliente WMS) se puede tener acceso a la información.

En este TFG, se utiliza QGIS, pero se podría utilizar otro cliente.

Dentro de QGIS, se añade una capa WMS y un nuevo servidor como podemos observar en la **Figura 44** y **Figura 45** respectivamente.

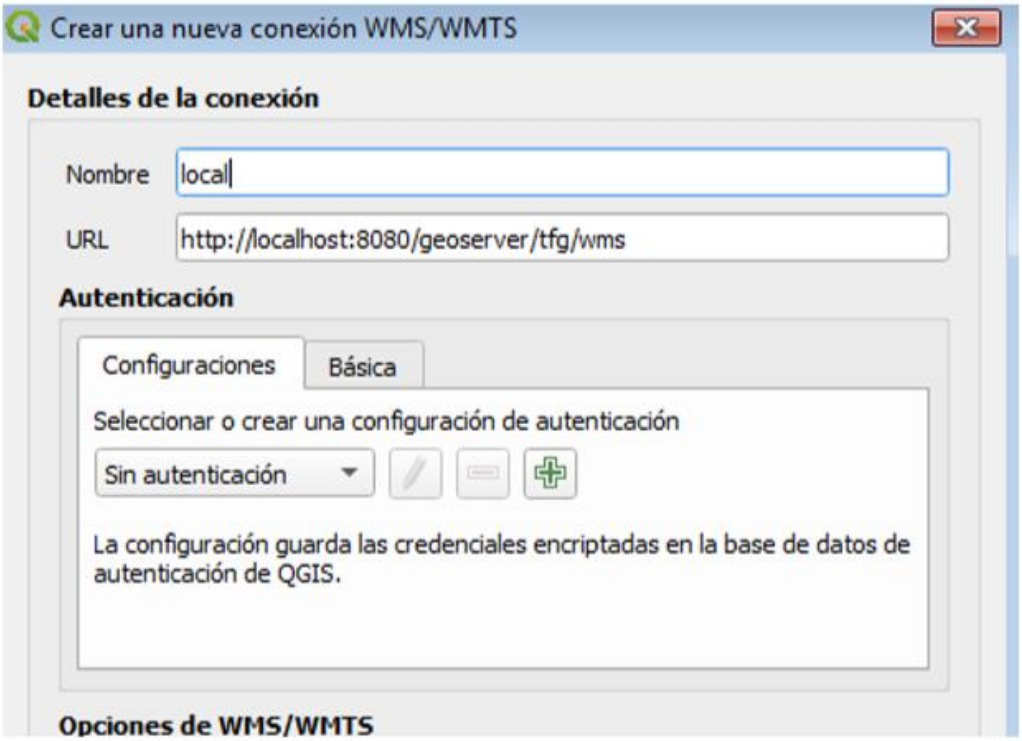

Figura 45. Nueva capa WMS

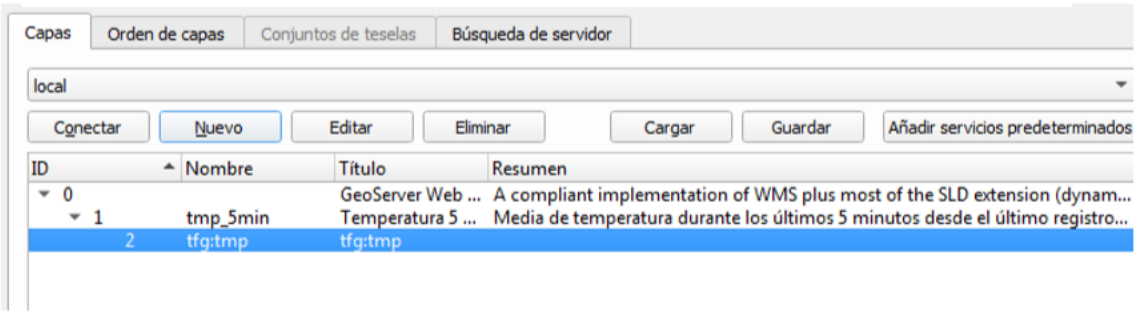

Figura 44. Creación nuevo servidor

De esta forma se tiene gráficamente el mapa de puntos, como se puede observar en la **Figura 46**, que al igual que se ha podido observar en el servidor de Python, simplemente al moverse, hacer zoom o refrescar la capa en QGIS la información en esta se actualizará.

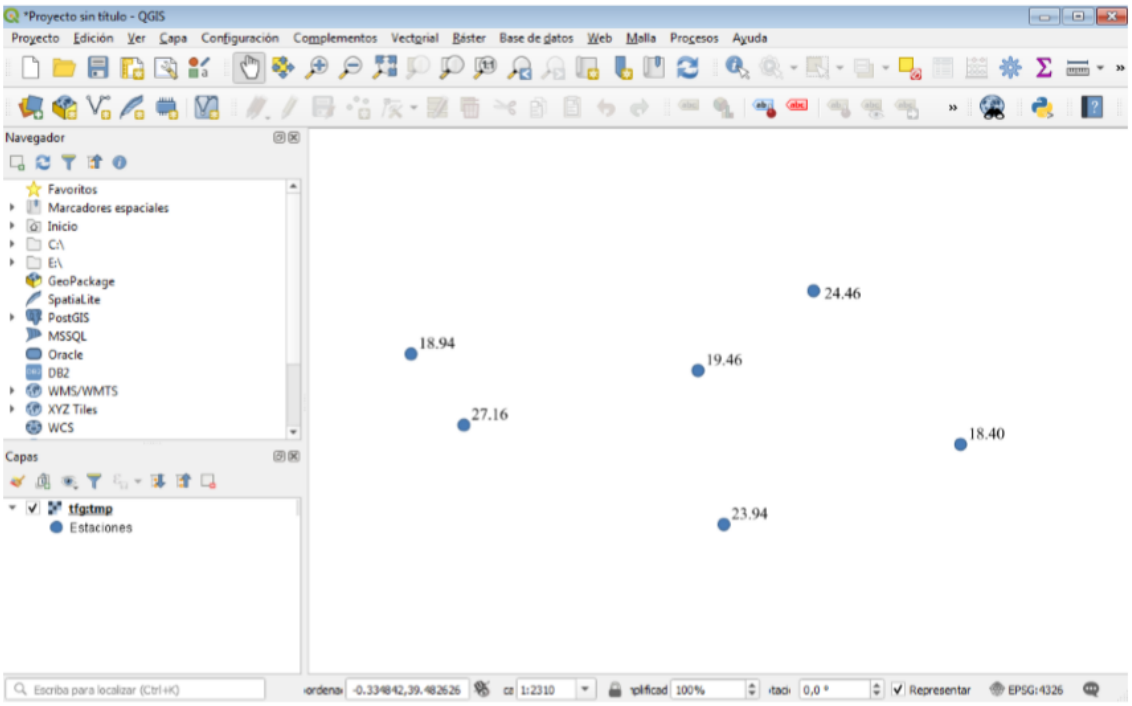

Figura 46. Representación gráfica de puntos

4.11 Creación de superficie continua

Para crear una superficie continua es necesario realizar un proceso de interpolación donde a cada píxel se le va a dar un valor en función de un modelo matemático o estadístico que tiene como entrada la información puntual de las estaciones.

Geoserver, mediante las transformaciones SLD puede realizar esta tarea en tiempo real, luego se puede obtener también una capa WMS que represente superficies continuas de temperatura, humedad, etc. que representen los datos en tiempo real de los sensores.

Para ello, se necesita instalar una extensión en Geoserver para que soporte el protocolo WPS (Procesamiento remoto en el servidor Geoserver), necesario para realizar estos cálculos, y también diseñar otro SLD.

Para poder realizar la instalación de la extensión WPS, primero que nada, se debe parar Tomcat.

tmp_max_mapa_5min.sld

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
 xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
 xmlns="http://www.opengis.net/sld"
 xmlns:ogc="http://www.opengis.net/ogc"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <NamedLayer>
    <Name>Barnes surface</Name>
    <UserStyle>
      <Title>Mapa Temp Máxima 5min</Title>
      <Abstract>Mapa de temperaturas máximas de los últimos 5 minutos</Abstract>
      <FeatureTypeStyle>
        <Transformation>
          <ogc:Function name="gs:BarnesSurface">
            <ogc:Function name="parameter">
              <ogc:Literal>data</ogc:Literal>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>valueAttr</ogc:Literal>
              <ogc:Literal>maxtmp</ogc:Literal>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>scale</ogc:Literal>
              <ogc:Literal>0.01</ogc:Literal>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>convergence</ogc:Literal>
              <ogc:Literal>0.2</ogc:Literal>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>passes</ogc:Literal>
              <ogc:Literal>4</ogc:Literal>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>minObservations</ogc:Literal>
              <ogc:Literal>1</ogc:Literal>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>maxObservationDistance</ogc:Literal>
              <ogc:Literal>0</ogc:Literal>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>pixelsPerCell</ogc:Literal>
              <ogc:Literal>10</ogc:Literal>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>queryBuffer</ogc:Literal>
              <ogc:Literal>0.01</ogc:Literal>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>outputBBOX</ogc:Literal>
              <ogc:Function name="env">
                <ogc:Literal>wms_bbox</ogc:Literal>
              </ogc:Function>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>outputWidth</ogc:Literal>
              <ogc:Function name="env">
                <ogc:Literal>wms_width</ogc:Literal>
              </ogc:Function>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>outputHeight</ogc:Literal>
              <ogc:Function name="env">
                <ogc:Literal>wms_height</ogc:Literal>
              </ogc:Function>
            </ogc:Function>
          </ogc:Function>
        </Transformation>
        <Rule>
          <RasterSymbolizer>
            <!-- specify geometry attribute of input to pass validation -->
            <Geometry><ogc:PropertyName>geom</ogc:PropertyName></Geometry>
            <Opacity>0.8</Opacity>
            <ColorMap type="ramp" >
              <ColorMapEntry color="#2E4AC9" quantity="-999" label="&lt;0" />
              <ColorMapEntry color="#41A0FC" quantity="4" label="4" />
              <ColorMapEntry color="#76F9FC" quantity="8" label="8" />
              <ColorMapEntry color="#2E6000" quantity="12" label="12" />
              <ColorMapEntry color="#9AF20C" quantity="16" label="16" />
              <ColorMapEntry color="#FAF833" quantity="18" label="18" />
              <ColorMapEntry color="#F19C33" quantity="22" label="22" />
              <ColorMapEntry color="#EA3F33" quantity="26" label="26" />
              <ColorMapEntry color="#BB3026" quantity="999" label=">26" />
            </ColorMap>
          </RasterSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
```

Figura 47. Cambio parametros.

De la misma manera se crea otro igual, pero que se empleará para las temperaturas máximas (solo se debe cambiar la línea 21 donde se añade el nombre del campo avgtmp en lugar de maxtmp: estilo tmp_media_mapa_5min.sld) (**Figura 47**).

Por último, se borra el estilo tmp del apartado anterior, y se cargan dos nuevos para los puntos de las estaciones, uno para las medias de las temperaturas tmp_media_estacion_5min.sld y otro para las temperaturas máximas tmp_max_estacion_5min.sld

Finalmente, en la capa de temperaturas en Geoserver (tmp_5min) se añade un estilo por defecto y los otros tres como extras como se puede observar en la **Figura 48** y también se puede ver el mapa con la capa de colores que marca la temperatura como se muestra en la **Figura 49.**

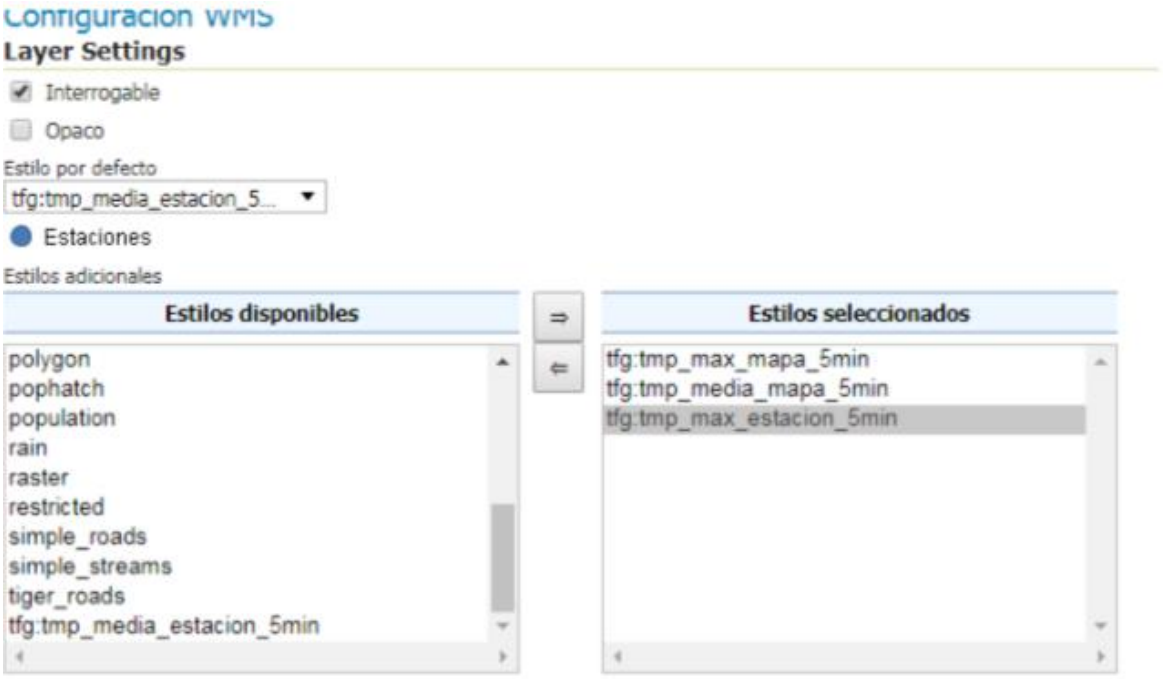

Figura 49. Edición capa temperatura.

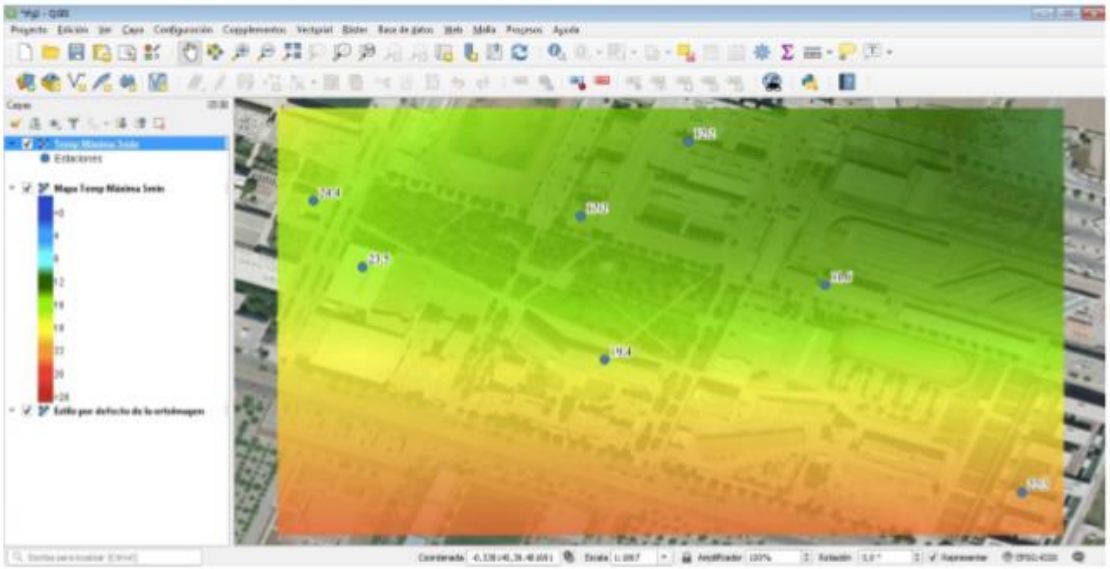

Figura 48. Mapa con capa de temperatura

5. Presupuesto

Se va a realizar este presupuesto, teniendo en cuenta que el trabajo es realizado por un experto en la materia y que por lo tanto en ningún caso las horas empleadas en la realización de este trabajo final de grado corresponden a las que se exponen aquí.

Se parte del supuesto de que se trabaja en régimen de autónomo y, por lo tanto, se necesita saber los gastos durante el tiempo de realización del proyecto. En este caso, se presupone que el tiempo de trabajo es de tres meses que implican las 140 horas dedicadas al proyecto, así como el tiempo de búsqueda de materiales, espera y otras operaciones que ocupan un tiempo suplementario al del propio trabajo.
Para ello se deben calcular los gastos mensuales, que se muestran en la siguiente tabla:

Gastos mensuales	
Alquiler	300.00 €
Luz	60.00 €
Agua	15.00 €
Red y telefono	65.99 €
Cuota autonomo	283.30 €
Colegio ingerieos	22.66 €
Gestoria	50.00 €
TOTAL	796.95 €

Una vez se han calculado los gastos mensuales se puede conocer el resultado del **gasto trimestral que es de 2390.85€.**

Gastos de inicio		
Sensores	50 x 2.3€	115.00 €
Placas Arduino	50 x 1.9€	95.00 €
Total		210.00 €

A esto, se le debe sumar el gasto inicial en componentes, es decir, el hardware y software utilizado. En este caso tanto el ordenador como el sistema operativo Windows no se toman en cuenta en el presupuesto, ya que se trata de una inversión anterior con la que ya se cuenta previamente. Los programas usados para la realización de este proyecto son open source (gratuitos) así que no implican un gasto en el proyecto. Estos cálculos dan como resultado unos **gastos totales de 2600,85€.**

Se procede al cálculo del salario empleado para el trabajador, como ya se comentando anteriormente se trata de un ingeniero experto en la materia por tanto se contará como precio por hora 50€ y el total de horas productivas de 140, teniendo en cuenta que no son el total de horas de trabajo sino aquellas que se pueden facturar, sin contar, la formación, la investigación de mejoras o las reuniones comerciales entre otros.
Por tanto, **el salario que recibe el ingeniero será de 7000€ total.**

Así, podemos concluir que **el presupuesto para la realización de este proyecto es de 9600€ más IVA.**

6. Conclusiones

Las conclusiones que se han obtenido en este trabajo final de grado son las siguientes:

En primer lugar, se ha conseguido de forma eficiente el objetivo principal de este trabajo, crear un sistema de visualización en tiempo real que muestre de forma continua e interoperable en formato imagen datos discretos provenientes de sensores Arduino y de esta forma se pueda facilitar la toma de decisiones y compartir y distribuir sobre Internet dicha información de forma sencilla e interoperable.

Para ello, se ha programado el dispositivo Arduino que desempeña correctamente su función captando los datos provenientes de los sensores.

Se ha generado el servidor de Python que captura de una forma muy eficiente los datos provenientes del Arduino. Se ha diseñado el modelo de datos capaz de almacenarlos y crear las tablas en las que estos han quedado registrados y donde se pueden realizar las consultas que se necesitan.

Mediante estas consultas se han generado informes de temperaturas/humedad teniendo en cuenta los datos conseguidos y mediante agregados se han realizado consultas estadísticas.

Si ha trabajado con tablas alfanuméricas y espaciales en el modelo de datos y con ello se han podido relacionar las tablas y mediante el componente espacial visualizarlas de forma gráfica posteriormente.

Una vez se ha conseguido almacenar esta información, se ha trabajado con protocolos estándar de interoperabilidad favoreciendo con ello, la posible distribución de forma estándar y en tiempo real de los datos calculados.
Los datos se pueden utilizar desde cualquier plataforma, cliente de SIG o usuario web.

Como objetivo final, se ha programado el servidor de cartografía Geoserver de tal manera que puede acceder a los datos y cargar la información espacial de forma gráfica y geolocalizada.

Uno de los puntos más importantes a tener en cuenta es el gran aprendizaje que se ha desarrollado al realizar este trabajo ya que han sido varios los programas que se ha aprendido a manejar. En primer lugar, la utilización de una maquina virtual, con todas las ventajas que esto conlleva, también el programa eclipse con el que se ha aprendido la programación en el lenguaje Python. También la creación de un modelo de datos, así como el uso de tablas interrelacionadas. Finalmente, el uso de servidores de cartografía es este caso Geoserver y como poder visualizar datos vectoriales.

Como trabajo futuro, se podría añadir algunas mejoras en la programación y hacer incluso más interesante este proyecto, como por ejemplo implementar que el servidor se pudiera desactivar o colgarse en un hosting remoto. También se podrían añadir más sensores a la tabla de sensores del modelo de datos para poder conseguir más datos y por tanto un mapa de temperaturas mucho más detallado o ampliar el tipo de datos que se capturan.
También sería interesante, comprobar el consumo energético de este y las posibles formas de ahorro para reducirlo.

Referencias bibliográficas.

[1] Smart Cities By GILLES BETIS, Member IEEE. Proceedings of the IEEE | Vol. 106, No. 4, April 2018

[2] Smart Cities: un primer paso hacia la internet de las cosas. Fundación Telefónica. [WWW Document]. URL https://www.telefonica.com/documents/341171/3261893/POLICY+PAPER_Smart+Cities_ES+La+Ciudad+como+plataforma+de+Transformaci%C3%B3n+Digital++Abril+2016.pdf/2c8ed5af-8690-44c2-aab0-4cbe3d1d89c2. (accedido 14.10.19)

[3] Fundación Endesa 2020. Smart Cities [WWW Document]. URL https://www.fundacionendesa.org/es/recursos/a201908-smart-city (accedido 14.10.19)

[4] Ajuntament de València. València ciudad inteligente. [WWW Document]. URL http://smartcity.valencia.es/vlci/piloto-sensores-mercado-de-russafa/ (accedido 14.10.19)

[5] Máquina virtual. [WWW Document]. URL https://www.vmware.com/es/products/workstation-pro.html (accedido 14.10.19)

[6] Prometec. Placa amica [WWW Document]. URL: https://www.prometec.net/nodemcu-arduino-ide/ (accedido 14.10.19)

[7] Prometec. Sensores de temperatura DHT11. [WWW Document]. URL https://www.prometec.net/sensores-dht11/ (accedido 14.10.19)

[8] CDMX Electronica (2016-2020) Modulo Detector de Calidad de Aire MQ-135. [WWW Document]. URL https://www.cdmxelectronica.com/producto/modulo-detector-de-calidad-de-aire-mq-135/ (accedido 14.10.19)

[9] Prometec. WIFI ESP8266 [WWW Document]. URL https://www.prometec.net/modelos-esp8266/ (accedido 14.10.19)

[10] HTTP GET method [WWW Document]. URL https://reqbin.com/Article/HttpGet (accedido 14.10.19)

[11] cliente servidor Apache HTTP Server, 2015. The Apache HTTP Server Project [WWW Document]. URL http://httpd.apache.org/ (accedido 14.10.19).

[12] Especificación OGC, 2019. OpenGIS® Catalogue Services Specification. [WWW Document]. URL https://www.ogc.org/ (accedido 14.10.19)

[13] OpenGIS Web Map Service (WMS) Implementation Specification version 1.3.0. [WWW Document]. URL https://www.ogc.org/standards/wms (accedido 14.10.19)

[14] OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option. [WWW Document]. URL https://www.ogc.org/standards/sfs/ (accedido 10.12.19)

[15] The OpenGIS® Styled Layer Descriptor (SLD) Profile of the OpenGIS® Web Map Service (WMS) Encoding Standard [WWW Document]. URL https://www.ogc.org/standards/sld (accedido 10.12.19)

[16] The OpenGIS® Web Map Service Interface Standard (WMS). [WWW Document]. URL https://www.ogc.org/standards/wms (accedido 10.12.19)

[17] PostgreSQL, 2020. PostgreSQL: Documentation: 12.2: PostgreSQL 12.2 Documentation [WWW Document]. URL https://www.postgresql.org/docs/12/index.html (accedido 10.12.19).

[18] Ventajas y Desventajas de PostgreSQL. [WWW Document]. URL https://todopostgresql.com/ventajas-y-desventajas-de-postgresql/.(accedido 10.12.20).

[19] Martínez Llario, José Carlos(2018). PostGIS Análisis Espacial Avanzado. Segunda edición. CreateSpace. 978-1727059359

PostGIS, 2020. PostGIS — Spatial and Geographic Objects for PostgreSQL [WWW Document]. URL http://postgis.net/ (accedido 10.12.19).

[20] QGIS, 2020. Proyecto QGIS [WWW Document]. URL https://qgis.org/en/site/ (accedido 22.4.20).

[21] ¿Qué es un SIG? [WWW Document]. URL http://sig.cea.es/SIG (accedido 22.4.20).

[22] GeoServer Documentation [WWW Document]. URL http://docs.geoserver.org/ (accedido 17.2.20).