UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Integración IoT en servicios Web

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Josep Benedito Fuster

Tutora: Sara Blanc Clavero

2017/2018

Agradecimientos

A Sara Blanc Clavero tutora de este proyecto; a José Luís Bayo Montón miembro del equipo de investigación SABIEN por sus indicaciones y directrices durante todo el proyecto.

A los profesores que han estado a mi lado durante mis estudios, haciéndome crecer tanto en el aspecto personal como en el profesional.

A la Universidad Politécnica de Valencia y a la Escuela Técnica Superior de Ingeniería Informática .

Gracias.

Resumen

En este trabajo esta orientado a Internet of Things, se ha desarrollado un sistema de coreografía en una RaspberryPI, para procesar los datos enviados por pequeños sensores y enviarlos a grandes grandes servicios como los servicios web o servicios en la nube. En este caso se implementará en un instituto de secundaria, con la finalidad de recibir los datos enviados por sensores de humedad instalados en el huerto de dicho instituto, para su posterior análisis y envío a distintos servicios. El sistema permite almacenar los datos recibidos en una base de datos local , la descarga de los datos a través de un navegador, la visualización por pantalla de dichos datos, la configuración del propio sistema para adaptarse al entorno, así como notificaciones al correo de los usuarios.

Palabras clave: RaspberryPI, IoT, .NET, REST, JSON, Mensajería de coreografía, SQLite.

Abstract

This project is oriented to the Internet of Things scope.It has been an implementation of a coreography system in a RaspberryPI, in order to collect the data sended by small sensors and send it to big platforms like web services or cloud services. In this case, it’s going to be deployed in a high school where there are several humidity sensors installed in this high school’s yard. The data will be procesed by the system and sended to different services. The systems allows to store the data in a local database, fetching the data using a web browser, data visualization, configuring the system in order to fit the local enviroment and sending notifications to the user’s email.

Keywords : RaspberryPI, IoT, .NET, REST, JSON, Service choreography , SQLite.

Tabla de contenidos

1. Introducción

Este proyecto, esta orientado a Internet of Things[1] de tal manera que se va conseguir llevar la información de pequeños sensores a grandes infraestructuras como puedan ser los servicios en nube.

Para ello se utilizará la Arquitectura orientada a servicios, SOA[2] en inglés, la cual permite definir piezas software que a través de mensajes pueden intercambiarse información. Existen dos roles dentro de SOA, productor y consumidor, aunque un servicio puede tener ambos roles.

Por lo tanto se va a utilizar una de las posibles implementaciones de SOA, en este caso será la coreografía de servicios[3]. Se desarrollarán diversos servicios los cuales se ejecutarán en paralelo por lo tanto serán autosuficientes aunque se intercambiarán mensajes debido a que en momentos puntuales, tienen una dependencia de datos de otro servicio.

Para la implementación de dichos servicios de coreografía, se va a utilizar el coreógrafo desarrollado por el equipo de investigación SABIEN[4], éste está diseñado para grandes entornos computacionales, por lo que es necesario adaptarlo para su uso en un sistema empotrado, en este caso en una RaspberryPI[5]. El coreógrafo es una serie de librerías que permiten crear un entorno de coreografía, añadirle servicios y que dichos servicios puedan enviar mensajes.

Los servicios van a procesar los datos enviados por sensores de humedad, para su análisis y posterior envío a un servicio web. La implantación de este sistema, se realizará en el huerto de en un instituto de educación secundaria, con tal de acercar la tecnología a los adolescentes y para facilitar el control del uso del agua.

Motivación

Una de las cosas que me hizo elegir este TFG, fue que implicaba trabajar con una RaspberryPI y además hacerlo utilizando .NET[6], jamás había trabajado con una RaspberryPI ni utilizado .NET, por lo tanto vi una buena oportunidad de aprender a utilizar estas tecnologías y así augmentar mis conocimientos.

Además, gracias a la asignatura Integración de Aplicaciones, mi interés por el concepto de Internet of Things ,aumento considerablemente, puesto que cada vez está más presente en el día a día. Es por esto, y por lo comentado en el párrafo anterior, que decidí realizar este TFG.

Objetivos

El objetivo principal es la implementación de un sistema de mensajería de coreografía que pueda ser soportado en un sistema empotrado como una RaspberryPI y que sea capaz de procesar datos de distintos sensores, analizarlos y enviarlos a servicios web o en nube.

Para ello, se pueden listar una serie de objetivos específicos:

- Adaptar el coreógrafo para su uso en la RaspberryPI.
- Probar si el core de la RaspberryPI puede ejecutar el coreógrafo.
- Diseñar los distintos servicios.
    - Definir las reglas sintácticas de los paquetes TCP[7].

    - Visualizar los datos por pantalla.
    - Almacenar los datos en la base de datos local.
    - Enviar los datos a ThingSpeak[8].
    - Enviar notificaciones a través de Microsoft Flow[9].
    - Descargar un archivo CSV con los datos de un día en concreto.
    - Modificación dinámica de los sensores activos.
- Establecer el flujo de los datos mediante los mensajes de coreografía.
- Configurar el coreógrafo antes de su ejecución.
- Implementar y testear los servicios
- Implantación en el instituto de secundaria.

También decir que un futuro objetivo, podría ser la creación de un entorno gráfico para la visualización de los datos.

Impacto Esperado

Una vez realizado el proyecto, el instituto va a ser capaz de leer y registrar los datos de humedad de los sensores de su huerto.

Esto les permitirá visualizar los datos además de recibir alertas cuando haya un comportamiento anómalo del sistema, como por ejemplo un sensor que ha dejado de funcionar, un valor de humedad demasiado alto o demasiado bajo.

Posteriormente a este trabajo se automatizará el riego por goteo, en función de los valores de humedad o del parte meteorológico, esto será de gran ayuda debido que en los periodos vacacionales el instituto se encuentra sin personal. Además se podrá hacer un riego óptimo, puesto que al regar solo en el caso de que la humedad de dicha zona baje hasta un valor determinado o no regar si va llover, optimizará el uso del agua.

También les permitirá hacer un análisis del consumo del agua en relación a la actividad que se desarrolle en el huerto, es decir si hay alguna relación entre el tipo de hortaliza plantada y el agua utilizada.

Se podrá realizar una interfaz para la visualización del huerto, donde los alumnos podrán ver que parcela del huerto corresponde a cada curso e intercalar los datos recibidos de los sensores con el registro de actividad del huerto, i.e. que variedad de hortaliza hay en cada parcela, el laboreo, la siembra, el abono...

Cabe decir que una vez terminado el trabajo, se obtendrá un producto mínimo viable que ayudará a los profesores y directora del centro a entender la solución al problema del riego automático para satisfacer sus necesidades actuales, pero se pueden utilizar sensores de cualquier tipo siempre que se ajusten a las reglas sintácticas definidas en este proyecto con una semántica JSON.

Comentar el potencial de este trabajo dentro del marco del proyecto de la Unión Europea Internet of Food and Farm 2020[10] en el que se busca utilizar las tecnologías de IoT para optimizar y facilitar la labor de los agricultores, así como minimizar los recursos utilizados.

Metodología

En primer lugar, se fija un objetivo principal acordado la tutora de este proyecto, para establecer las directrices del desarrollo del trabajo.

En segundo lugar, reunido con un miembro del equipo de investigación de SABIEN-ITACA, se obtiene la información sobre el funcionamiento del coreógrafo, ya que su equipo de investigación ha sido quien lo ha desarrollado, por lo tanto, era necesario con tal de comprender el funcionamiento , y así poder aprovechar al máximo su potencial.

Posteriormente, una vez verificado que las funciones básicas de coreografía pueden ser soportadas sobre una RaspberryPI, se empezó a desarrollar los distintos servicios que éste debía proporcionar y se verificaron utilizando un servicio de test para emular a los sensores con una carga sintética.

Durante el desarrollo, se realizan sesiones con la tutora para revisión y evaluación de objetivos obtenidos y posibles modificaciones.

Finalmente, junto con el alumno de intercambio Quentin Blazejewski se puso en producción el trabajo realizado para comprobar su correcto funcionamiento substituyendo la carga sintética por una carga real. Las pruebas se han realizado en el laboratorio, quedando pendiente, como trabajo futuro, las pruebas de integración en el instituto y su huerto escolar.

Estructura

En esta memoria se encuentra primero información del trabajo a realizar, el análisis y comparación de otros trabajos relacionados con la temática del proyecto haciendo ver las mejoras que este proyecto va a aportar.

Se realiza el análisis de las posibles soluciones del problema y se decide cuál es la que mejor se adapta a las necesidades del trabajo.

Posteriormente se diseña con detalle el flujo de datos, los servicios su interactuación y funcionamiento.

Se explica el desarrollo práctico de los distintos servicios utilizados para la consecución del problema planteado.

Se aplica el producto desarrollado en un entorno de prueba, para poder valorar el funcionamiento.

Se realiza una simulación, comprobando el correcto funcionamiento de todos los servicios implementados en el coreógrafo.

Enunciado de las conclusiones.

Proyecciones del trabajo.

Colaboraciones

En este trabajo se ha colaborado con el alumno de intercambio Quentin Blazejewski, quien ha desarrollado los sensores encargados de enviar los datos a la RaspberryPI.

2. Estado del arte

Debido al bajo coste de la RaspberryPI, no es difícil encontrar en internet diversos proyectos que permitan interactuar con sensores y por ejemplo automatizar el riego de las plantas[11].

También se han desarrollado trabajos relacionados con sistemas IoT y la monitorización, recolección de datos, o su envío a servicios web.

Por ejemplo, un proyecto titulado "Desarrollo de una estación meteorológica usando una RaspberryPI"[12] implementa un sistema que recoge datos de temperatura, humedad, intensidad lumínica y presión atmosférica y se muestran en un servidor web dónde se pueden ir viendo los valores recogidos. También se envían alertas en caso de que algún dato supere un valor crítico determinado por el usuario.

Otro proyecto relacionado es el llamado "Sistema portátil autónomo para la adquisición de datos conectado a la nube"[13] en este proyecto se utiliza una RaspberryPI que recoge el valor de pH de un medio líquido y lo envía al servicio de Microsoft Azure, también se envían alertas por Microsoft Flow al igual que se va a hacer en este proyecto.

Y finalmente el proyecto "IoT based agriculture monitoring and smart irrigation system using RaspberryPI"[14] en este proyecto se ha implementado un sistema que recoge datos de humedad y temperatura de un huerto doméstico y los envía a ThingSpeak y en el caso de que la humedad sea baja, se activa el sistema de riego.

Critica al estado del arte

Se ha podido observar que aunque hayan diversos trabajos, proyectos relacionados con la temática de este TFG, no se ha podido encontrar ninguno que utilice un sistema de coreografía. Por otra parte, algunos trabajos simplemente lo que hacen es de enlace entre el sensor y el servicio web, y los que hacen un procesado de los datos, es relativamente sencillo.

El desarrollo basado en coreografía de servicios tiene como ventaja principal evitar que se produzcan bloqueos entre servicios. Además, otras ventajas son la escalabilidad de la solución, su ampliación con nuevos servicios y su mantenimiento de forma más sencilla que si se desarrolla una solución monolítica basada en funciones. Esto es fundamental en el proyecto ya que se pretende continuar y ampliar en el futuro.

Propuesta

Por lo tanto, cabe destacar que este en este proyecto se hace una mejora consistente respecto a los trabajos relacionados anteriores puesto que la utilización de un coreógrafo, permite que cada servicio se desarrolle independientemente del resto, por lo que si alguno de ellos falla, no provocaría un fallo en cascada.

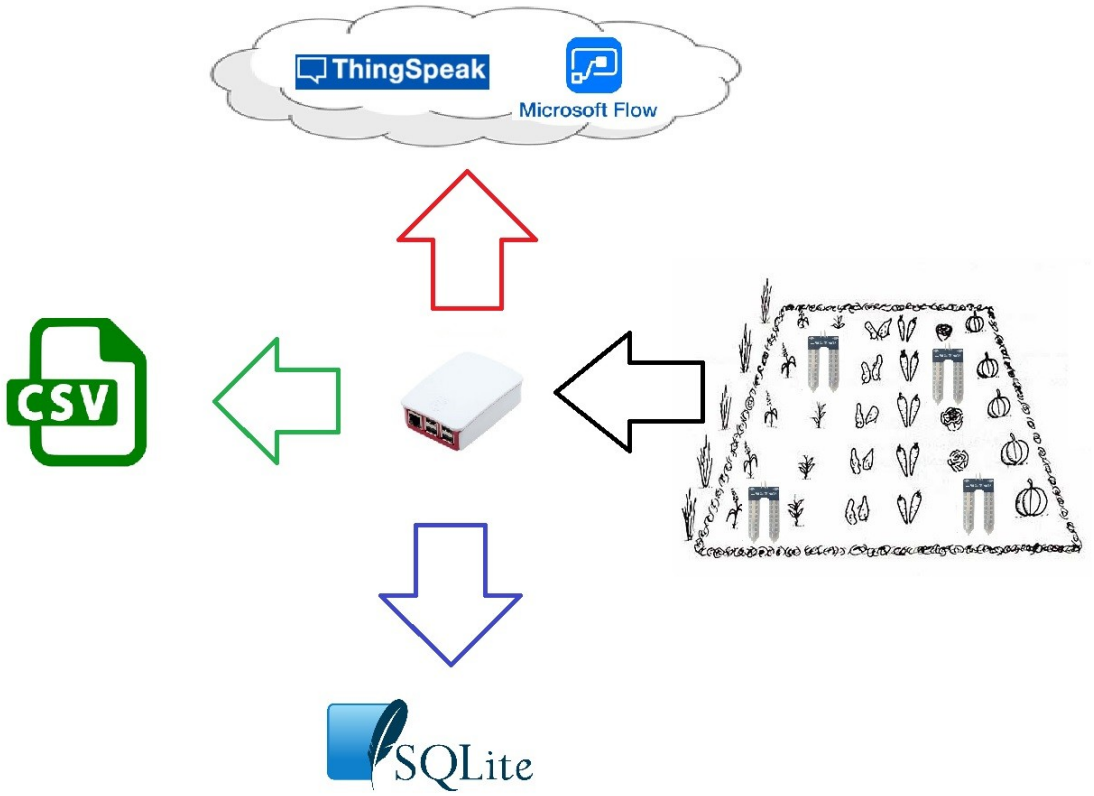
Además, la escalabilidad de este sistema se puede apreciar fácilmente, porque si un servicio se ve sobrecargado, se puede fácilmente desplegar un duplicado de dicho servicio para aligerar la carga de trabajo del servicio en concreto.

Finalmente decir, que otra virtud de este sistema es la posibilidad de añadir nuevos servicios/funcionalidades. Es posible ampliar con nuevos servicios sin tener que modificar el código ya existente, simplemente añadiendo un nuevo servicio al coreógrafo y etiquetando al

servicio con el nombre adecuado con tal de que encaje en el flujo de mensajería, sería suficiente.

3. Análisis del problema

En el dibujo 1 se muestra el concepto desarrollado en el trabajo. Los datos son recogidos en el huerto a través de sensores que envían la información a un gateway implementado sobre una RaspberryPI. Este gateway "inteligente" será capaz de procesar la información, extrayendo los datos de los mensajes recibidos, componiendo nuevos mensajes y enviándolos a diversos servicios propietario (ej. CSV) o públicos como ThingSpeak o MicrosoftFlow (a modo de ejemplo).



Dibujo 1: Concepto del proyecto

Análisis de la seguridad

En este proyecto, no se ha aplicado medidas de seguridad puesto que no se trata de ningún tipo de aplicación donde este aspecto sea crítico, como sería el caso de una aplicación bancaria o una aplicación de defensa.

Bien cierto es que, al no cifrar los envíos de los sensores, se podría ver el contenido de los mensajes. Una solución relativamente sencilla sería aplicar un protocolo de cifrado, como, por ejemplo, Diffie-Hellman[15] visto en la asignatura de Redes Corporativas.

Existe la posibilidad de un ataque de denegación de servicio, también conocido como DoS[16], donde el atacante podría ver a que puerto e IP están mandando los paquetes los sensores y empezar a enviar paquetes a esa dirección con tal de sobrecargar el puerto y que el listener TCP no sea capaz de seguir proporcionando el servicio.

Cabe recordar que este proyecto se va a implementar en un ámbito escolar, por lo que se supone que el riesgo de un ataque a la RaspberryPI es relativamente bajo.

Además, se han hecho pruebas de carga en el test de los servicios y los resultados muestran que el sistema es capaz de soportar la carga de trabajo esperada.

Por último comentar que sí se utiliza una clave privada al enviar los datos al ThingsSpeak, por lo que manipular la información que se almacena en la nube sería algo más complicado y que hace falta utilizar credenciales con tal de acceder por SSH[17] a la RaspberryPI.

Análisis del marco legal y ético

Análisis de la protección de datos: En este proyecto, se procesa y almacena información, pero, al tratarse de información no sensible, no se ha tomado ninguna medida en este aspecto.

Propiedad intelectual: No se utiliza ningún tipo de software privativo. El coreógrafo utilizado, ha sido desarrollado por el grupo de investigación SABIEN-ITACA de la UPV.

Análisis de riesgos

Los riesgo que presenta este proyecto son de carácter tecnológico.

Para empezar, existe la posibilidad que haya una falta de conectividad. Quiere decir que uno o varios de los sensores dejen de enviar datos a la RaspberryPI ya sea debido a un fallo del propio sensor, o que el sensor se ha colocado demasiado lejos y la señal no es capaz de llegar a la RaspberryPI. Para solucionar esto, el sistema esta preparado con un servicio que una vez pasado un determinado periodo de tiempo, comprueba que todos los sensores han conseguido realizar un envío; si alguno de ellos no ha enviado nada, se envía una notificación por correo a través de Microsoft Flow a una o varias direcciones de correo.

También es posible que la RaspberryPI se quede sin memoria, por lo que los datos enviados a partir de ese momento se perderían. Este caso se ha solucionado haciendo que el servicio que envía los datos en CSV al usuario borre dichos datos una vez enviados, de este modo se libera memoria en la RaspberryPI.

Por último, podría ser que el servicio de ThingSpeak no este disponible en algún momento, o que no haya conexión a Internet, con lo que los datos enviados a dicho servicio se perderían. Esto se ha solucionado guardando los datos también en una base de datos local.

Identificación análisis de soluciones posibles

Al empezar este proyecto, se plantearon dos soluciones. Utilizar o no utilizar un coreógrafo para gestionar la comunicación entre los distintos servicios que debe proporcionar la RaspberryPI.

Por una parte, el no utilizar el coreógrafo dota de más sencillez el funcionamiento, ya que la aplicación del coreógrafo es compleja . Uno de los problemas que presenta esta solución es que o todos los servicios leen los datos al mismo tiempo para procesarlos, cosa que ocasionaría una pérdida de tiempo y un desperdicio de recursos, o se ejecuta secuencialmente y, por lo tanto se ralentiza el proceso, además de que un fallo podría detener todo el funcionamiento.

La segunda solución posible, es utilizar la coreografía de servicios, esto nos aporta mucha flexibilidad puesto que una vez se tiene el funcionamiento básico(listener TCP y controlador de base de datos), cada servicio que se desee implementar simplemente se añade al entorno de coreografía como una pieza más transparente al resto de servicios ya operando, de este modo los cambios que se deben hacer al sistema ya implementado son nulos o mínimos.

Además nos proporciona una alta escalabilidad, puesto que duplicar un servicio cuando este se ve sobrecargado es sencillo duplicarlo, para así repartir la carga de trabajo. También comentar que el fallo de un servicio no es probable que ocasione un fallo global del sistema, ya que cada servicio se esta ejecutando independientemente del resto.

Solución propuesta

Una vez analizadas con detalle las dos soluciones posibles, se ha decidido que utilizar coreografía de servicios es la solución que mejor se adapta a las necesidades del proyecto. Se implementará el coreógrafo y los servicios necesarios sobre una RaspberryPI, con sus mensajes de coreografía y el flujo de trabajo específico para la solución propuesta. Además, se definirán las reglas sintácticas y semánticas del sistema para permitir el enlace de los datos desde los sensores a los servicios externos fuera del campo y se desarrollarán los componentes software necesarios para conseguir la interoperabilidad de la RaspberryPI (nuestra gateway inteligente) con los servicios externos. Se han elegido para el proyecto ThingSpeak y Microsoft Flow, además de desarrollar un servicio propietario CSV.

Plan de trabajo

	Descripción	Horas
1	Reuniones la tutora con tal de consultar dudas, debatir el esquema del proyecto	10
2	Instalar Windows IoT en la RaspberryPI	5
3	Documentarse sobre el funcionamiento de Windows 10 IoT	4
4	Realización de los ejercicios de ejemplo del coreógrafo	12

5	Implementación del coreógrafo y un servicio de echo para probarlo	10
6	Reuniones en ITACA para conocer en profundidad el funcionamiento del coreógrafo	10
7	Determinación de los servicios	4
8	Definición del flujo de trabajo como solución al caso concreto de huertos escolares	10
9	Implementación del Listener TCP	22
10	Implementación del servicio de visualización	10
11	Documentarse sobre el funcionamiento de SQLite en .NET	5
11	Implementación del servicio de control de base de datos	20
12	Documentarse sobre ThingSpeak	1
13	Creación y configuración de la cuenta de ThingSpeak	1
14	Implementación del servicio de ThingSpeak	20
15	Primeras pruebas con Quentin	5
16	Documentarse sobre Microsoft Flow	1
17	Creación y configuración de los flujos de alerta en Microsoft Flow	1
18	Implementación del servicio encargado de enviar alertas	25

19	Formulario para introducir los valores de configuración	3
20	Parametrizar el entorno de coreografía	5
21	Servicio encargado de devolver los datos en CSV	25
22	Creación del archivo HTML para descargar los datos en CSV	5
23	Servicio encargado de modificar dinámicamente los sensores activos	15
24	Pruebas y resolución de los problemas encontrados	30
25	Redacción de la memoria	60
26	Pruebas finales con Quentin	10
27	Implementación del sistema en el instituto	5
	Total	334

Presupuesto

Concepto	Precio
RaspberryPI	57,99€[18]
Desarrollo del producto Respecto a la referencia encontrada[19]	9€/hora → 9*334 = 3.006€
Desarrollo del producto Respecto mi valoración	30€/hora → 30*334 = 10.020€
Total Respecto a la referencia encontrada[19]	3.069,99€
Total Respecto mi valoración	10.077,99€

4. Diseño de la solución

Arquitectura del Sistema

En este proyecto, interactúan tres piezas, los sensores, la RaspberryPI y los servicios en la nube.

Figura 1: Capas de la arquitectura

En la primera capa se establece la relación entre los sensores, la RaspberryPI y los servicios en la nube.

Los sensores son los encargados de enviar la información a la RaspberryPI mediante TCP. Se ha elegido .JSON y se ha definido una sintaxis para que el servicio encargado de procesar los mensajes pueda procesarlos correctamente, por lo tanto si se desea añadir un nuevo sensor, este deberá seguir dicha sintaxis.

```
{"ID":INT,"Type":INT,"Flag":INT,"Data":String,"Checksum":INT}
```

El valor de *Type* indicará el tipo de sensor del que se trata, por ejemplo "0 – Temperatura", "1 – Humedad".

Dependiendo del valor del *Flag*, se realizarán unas acciones u otras. El valor 1 indica que el valor de datos es nulo, el 2 indica que hay múltiples valores en "Data" y el valor 4 que la acción viene dada manualmente. Esto último se utilizará cuando se automatice el riego, cosa que queda fuera de este trabajo.

El campo *Data* deberá ser los valores recogidos por el sensor separados por comas, por ejemplo "5,3,4". Hay sensores que generan varios valores, por ejemplo, ciertos sensores de humedad.

Cabe decir que existen estándares de facto como por ejemplo SensorML[20], pero en este caso se ha definido una sintaxis propia, similar al estándar de facto, pero simplificado al caso que nos ocupa.

Cuando los datos llegan a la RaspberryPI, es el servicio de TCP quien los procesará desagregando los campos .JSON y comprobando los flags y el checksum. El resultado se transporta con mensaje de coreografía al servicio de visualización, al servicio encargado de enviar alertas y al controlador de la base de datos. Los servicios de MicrosoftFlow y de ThingSpeak, enviarán los datos a los servicios en la nube. Para realizar este envío, se utilizará REST[21] más concretamente se utilizará el método estándar GET de HTTP, para construir una petición y enviarla al servicio determinado. Todos son servicios implementados sobre el gateway.

Finalmente, el servicio en la nube recibirá la petición enviada por el servicio de coreografía, y realizará las acciones pertinentes, guardar y mostrar los datos (ThingSpeak) o enviar un correo electrónico con un mensaje de alerta (Microsoft Flow).

En resumen, en esta capa se define la interoperabilidad de las tres piezas.

En la segunda capa se definen las reglas de comportamiento de los componentes del huerto. Por ejemplo, se define la frecuencia con la que los sensores deben enviar información. Si un sensor no envía información antes de cierto "tiempo máximo", el sistema enviará una notificación al móvil del personal encargado de que dicho sensor ha dejado de funcionar.

En la última capa se define la organización de los servicios para cumplir con la aplicación deseada. Los servicios implementados son:

El **servicio de visualización**, es el encargado de mostrar los datos por pantalla.

El **controlador de base de datos** guarda los datos recibidos consiguiendo así la persistencia de estos, además, este servicio les añade el time stamp que será de gran utilidad para los otros servicios. También mantiene una tabla con los IDs de los sensores activos.

El servicio encargado de enviar los datos al **ThingSpeak**, se ejecuta periódicamente en función del valor introducido al configurar el dispositivo y es el encargado de organizar la información de cada sensor. Esto lo hará enviando un mensaje de coreografía dirigido al controlador de la base de datos, que se encargará de recuperar el valor de los datos correspondientes a la consulta. Puesto que ThingSpeak limita el número de envíos que se pueden hacer en un determinado tiempo, hay que adecuar la frecuencia de envío ThingSpeak independiente de la frecuencia de recepción de los sensores enviando la media de los valores recibidos. Esta es la mejor forma de adaptar la aplicación al servicio ThingSpeak.

También estará ejecutándose cíclicamente el servicio encargado de **comprobar el correcto funcionamiento de los sensores**. El tiempo de ciclo, será en función del valor configurado, que al igual que en el servicio de ThingSpeak, se ha pasado como parámetro al crear este servicio junto con el número de sensores que van a estar enviando datos, puesto que es clave para ver cuales están fallando. Este servicio se encarga de enviar una notificación por correo electrónico al usuario o usuarios según su configuración en Microsoft Flow, indicando que sensor ha dejado de enviar datos para que así, el responsable este informado y pueda solucionar el posible problema que hay con ese sensor. Este servicio también recibe los datos del TCP listener y comprueba que ningún valor de humedad este dentro de los límites marcados, de no ser así se enviaría una notificación por correo electrónico.

El servicio encargado de **modificar los sensores** que se encuentran activos, recibirá un ID introducido por pantalla por el usuario y si se desea eliminar o añadir dicho sensor, y enviara mediante un mensaje de coreografía a los servicios que deben mantener un control de los sensores activos.

Finalmente, el servicio que facilita los datos en formato **CSV** esta esperando una petición GET que contiene la fecha de la cual se quieren obtener los datos. Esta petición se hace mediante un navegador web.

Diseño Detallado

Servicios

TCPListener
Este servicio es el encargado de procesar los datos enviados por los sensores y enviar un mensaje de coreografía a los servicios correspondientes, en función del contenido del mensaje.

DataBaseController
Este servicio se encarga de realizar las distintas operaciones sobre la base de datos, añadir, eliminar, actualizar y consultar datos.

ViewController
Este servicio mostrará por pantalla los datos recibidos por los sensores así como los sensores activos.

ThingSpeak
Este servicio se encargará de coger los datos enviados por los sensores, y enviar el valor medio de cada sensor a la plataforma de ThingSpeak.

MicrosoftFlowAlerts
Este servicio utilizando Microsoft Flow, enviará notificaciones al correo electrónico configurado en el flujo de Microsoft Flow cuando haya un valor anómalo de los datos o cuando un sensor deje de funcionar.

SocketCSV
Este servicio enviará los datos enviados por los sensores en CSV al usuario cuando éste lo solicite a través de una petición GET. Una vez enviados se borrarán de la base de datos.

SensorsController
Este servicio permite al usuario modificar dinámicamente los sensores que se encuentran activos. Cuando el usuario añade o elimina un sensor, este servicio enviará un mensaje de coreografía destinado a los servicios correspondientes.

Flujo de coreografía
Primero se explicará en más detalle el funcionamiento del coreógrafo y se verán ejemplos.

Los servicios envían un mensaje de coreografía al coreógrafo, éste esta compuesto principalmente por cuatro partes.

Sender
Es el valor del ID del servicio que va enviar el mensaje. Por defecto se utiliza el ID del servicio desde donde se crea el mensaje.

Receiver
Es el ID del servicio o servicios al cual va destinado el mensaje.

Contenido
Esta parte está formada por un SOAPMSG, y contiene el método del servicio que se debe ejecutar y los parámetros necesarios para ello. Los parámetros son un diccionario formado por pares de clave-valor.

Protocolo
Hay diversos protocolos, pero en este proyecto hacemos simplemente uso de dos.

Request
Es el que más se ha utilizado, cuando se indica este protocolo quiere decir que el Sender espera una respuesta, por lo tanto el servicio que ha sido llamado responderá a dicho servicio e intentará ejecutar el método de respuesta, el método que se debería de crear para poder procesar la respuesta sería "NombreDelMedotoSolicitadoResponse" y el valor devuelto será "NombreDelMedotoSolicitadoResult".

Eventmsg
Este es el más simple, indica que no se espera ningún tipo de respuesta, por lo tanto simplemente es un envío de información hacia diferentes servicios. Se podría utilizar el protocolo Request y seguiría funcionando igual, pero se generaría "ruido" en el entorno de coreografía al enviarse una respuesta vacía, por lo tanto si no se espera contestación por parte del servicio indicado, este es el protocolo más óptimo.

Respecto al ID de los servicios es una parte muy importante puesto que se **pueden utilizar máscaras para facilitar el envío de mensajes**. Esto quiere decir que se puede utilizar las siguiente IDs "Test.A.1","TEST.A.2","TEST.B.1", de tal modo que un mensaje dirigido a la ID "TEST.x.x" le llegaría a los tres servicios, con la ID "TEST.A.x" solo a los dos primeros primeros y con ID "TEST.A.1" solo al primero. Esto facilita el uso, puesto que no se deben enviar múltiples mensajes, sino más bien hacer un buen etiquetado de los mensajes.

A continuación diversos ejemplos:

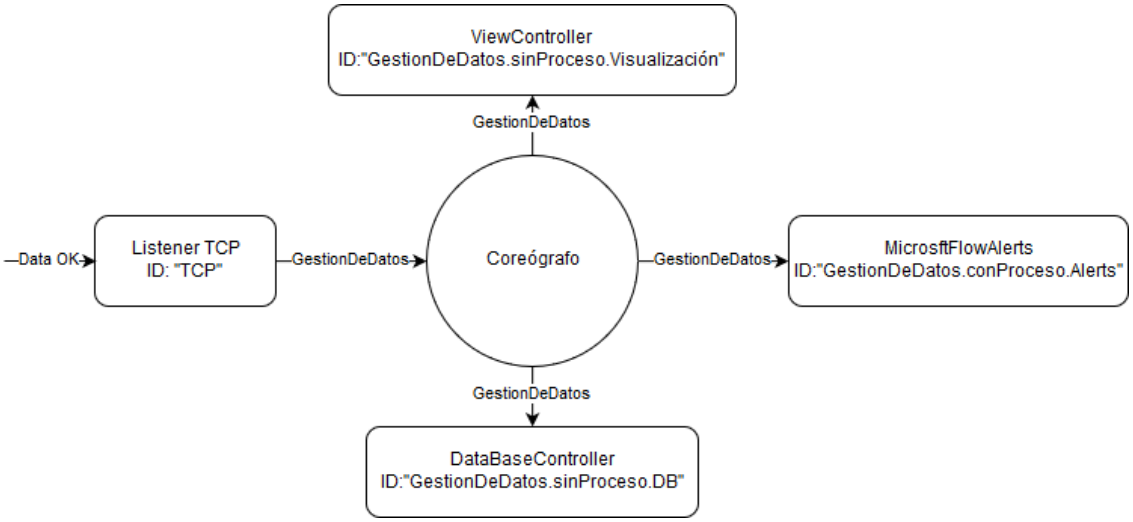


Figura 2: Ejemplo datos válidos

En este ejemplo el sensor ha enviado datos válidos y el listener TCP ha podido procesarlos y enviarlos el coreógrafo que se encargará de que los servicios correspondientes reciban el mensaje.

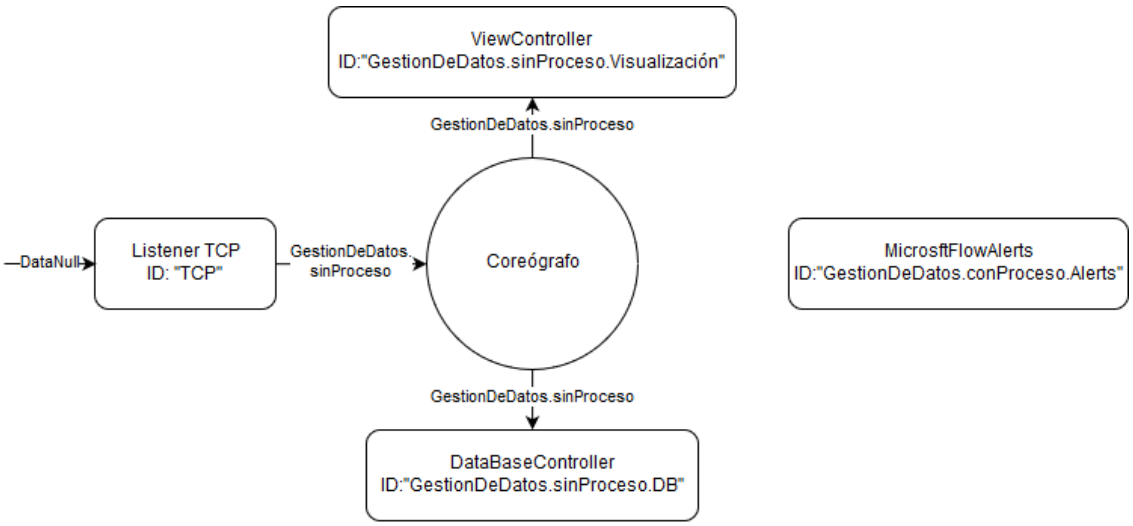


Figura 3: Ejemplo datos null

En este caso, el sensor no ha tenido tiempo de obtener los datos y envía un valor NULL. Al no tener datos, no hace falta que el servicio de MicrosoftFlow compruebe si se ha sobrepasado el limite superior o si esta por debajo del limite inferior. Pero si que se muestre por pantalla, y que se guarde en la base de datos, porque sino el servicio de alertas considerará erróneamente que el sensor ha fallado.

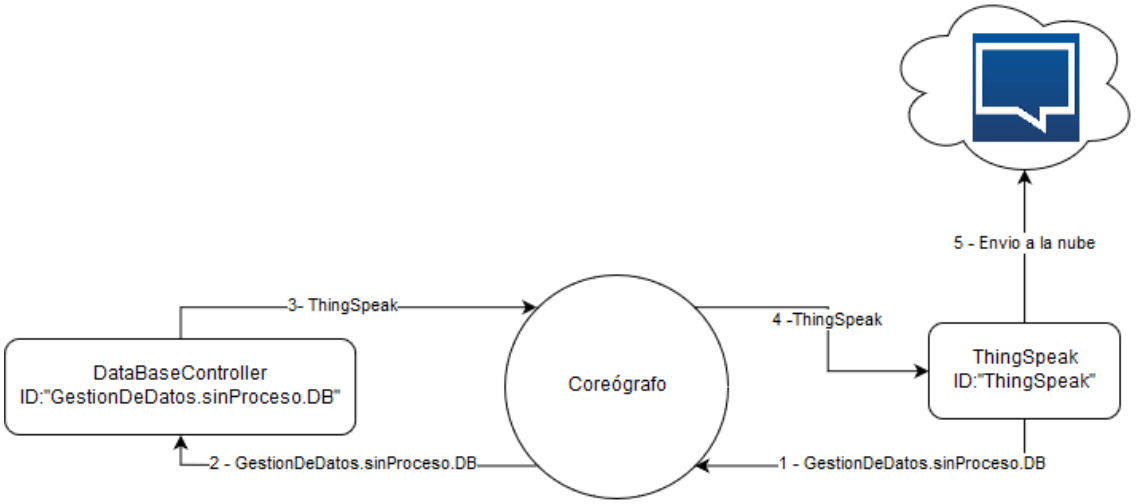


Figura 4: Ejemplo petición

Los ejemplos anteriores eran mensajes de evento, mientras que este se trata de una petición a un servicio. El servicio solicitado devolverá el resultado al servicio correspondiente utilizando la ID del solicitante enviada en el mensaje de petición.

Visto en detalle unos pocos ejemplos, ahora se verá el funcionamiento global.

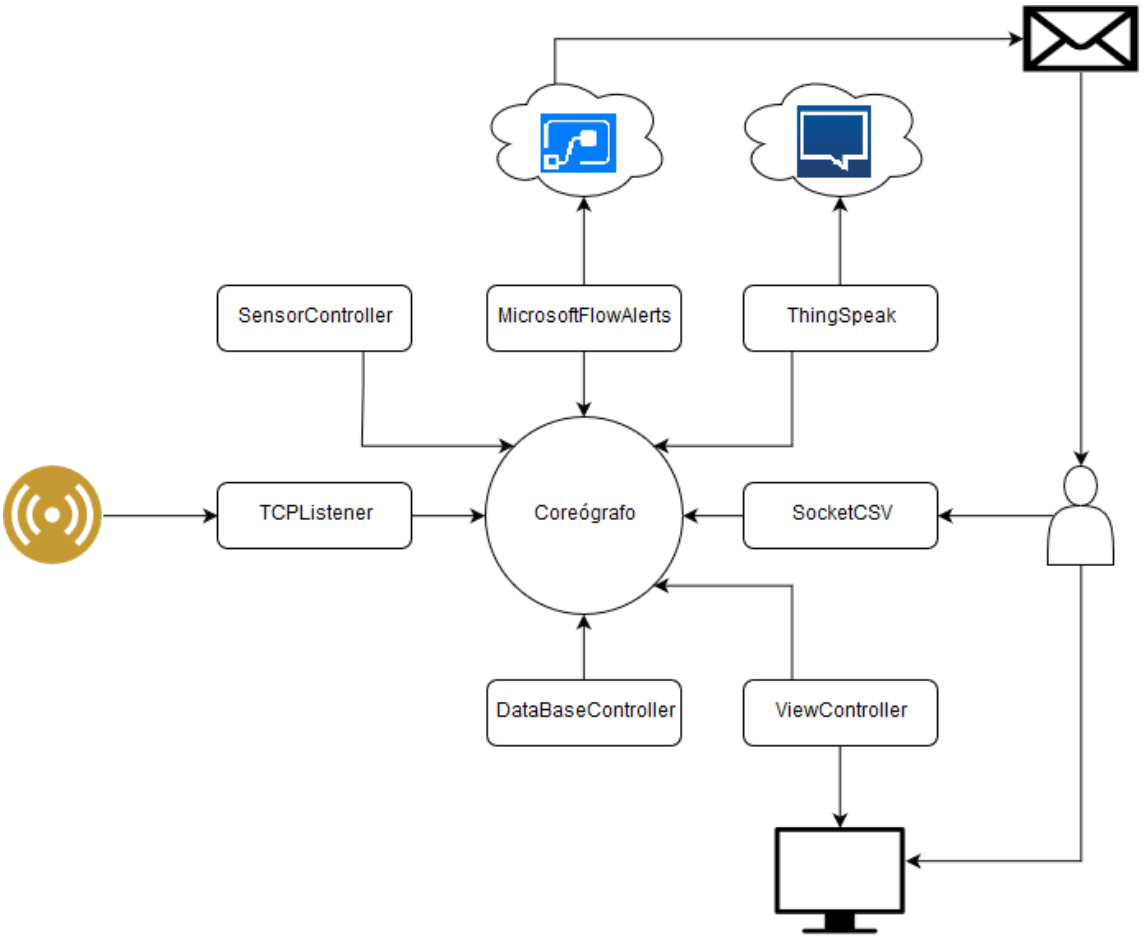


Figura 5: Ejemplo detallado del sistema

Cada servicio interactúa con el sistema de forma diferente. A continuación se muestran distintas tablas donde se puede observar con que IDs va a enviar cada servicio los mensajes y que IDs llevarán los mensajes que éstos van a recibir.

TCPListener, ID: "TCP"

Envia	**Recibe**
GestionDeDatos.X.X	
GestionDeDatos.SinProceso.X	

DataBaseController, ID: "GestionDeDatos.SinProceso.DB"

Envia	Recibe
GestionDeDatos.conProceso.Alerts	GestionDeDatos.X.X
ThingSpeak	GestionDeDatos.SinProceso.X
CSV	GestionDeDatos.SinProceso.DB

ViewController, ID: "GestionDeDatos.SinProceso.Visualización"

Envia	Recibe
	GestionDeDatos.X.X
	GestionDeDatos.SinProceso.X

ThingSpeak, ID: "ThingSpeak"

Envia	Recibe
GestionDeDatos.SinProceso.DB	ThingSpeak

MicrosoftFlowAlerts, ID: "GestionDeDatos.conProceso.Alerts"

Envia	Recibe
GestionDeDatos.SinProceso.DB	GestionDeDatos.X.X
	GestionDeDatos.conProceso.Alerts

SocketCSV, ID: "CSV"

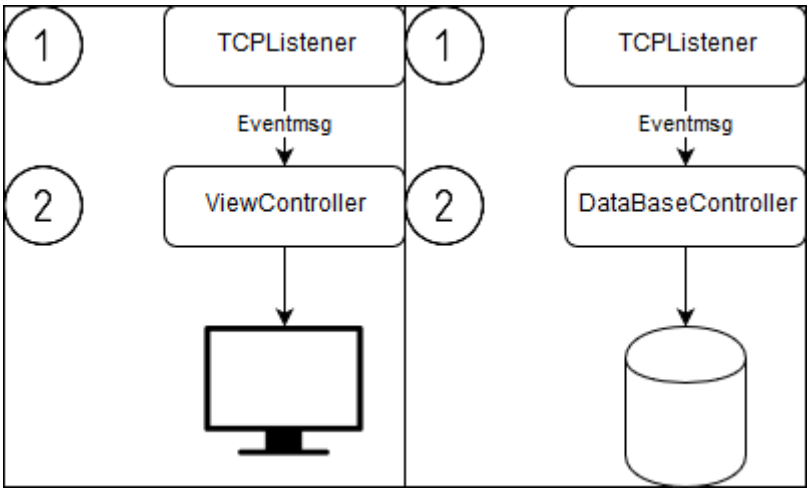
Envia	Recibe
GestionDeDatos.SinProceso.DB	CSV

SensorsController, ID: "Sensors"

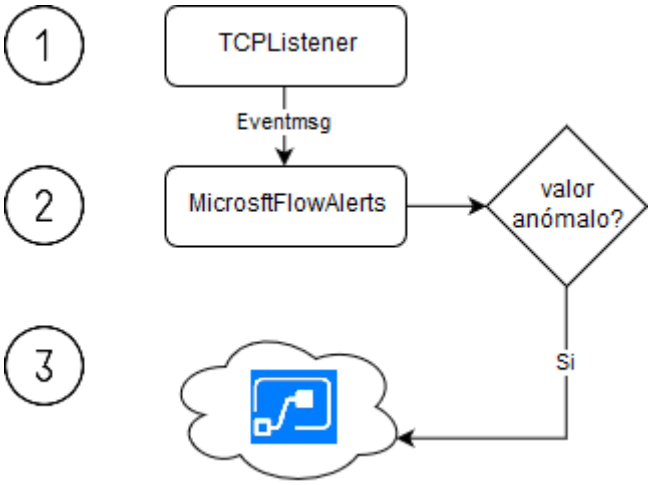
Envia	Recibe
GestionDeDatos.X.X	

Flujo de trabajo

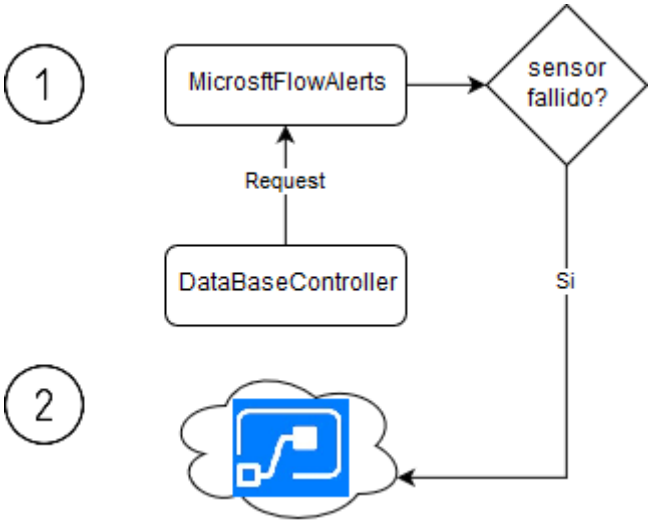
Del flujo de coreografía definido en el apartado anterior a través de la identificación de los mensajes se ha obtenido diversos flujos de trabajo.



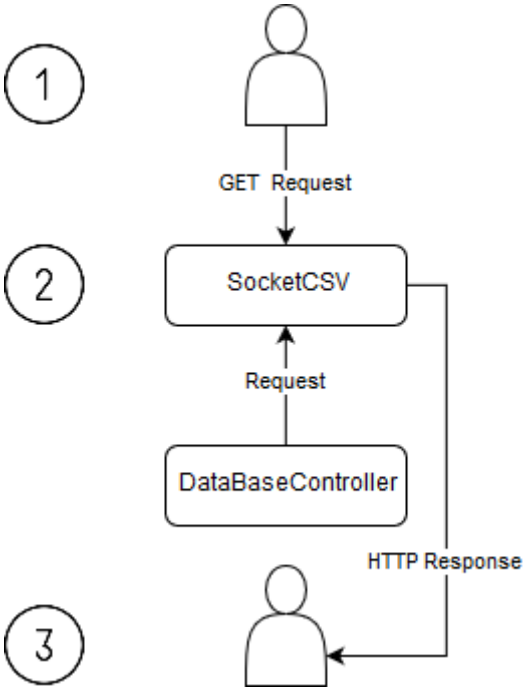
Estos dos flujos de trabajo se ejecutarán cuando el TCPListener, reciba un mensaje del sensor independientemente de que los datos sean válidos, no válidos o nulos.



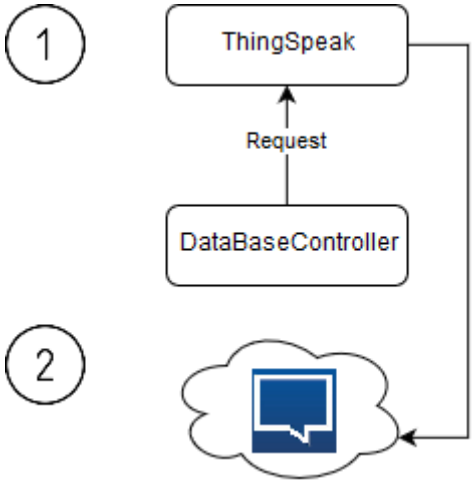
Este flujo se ejecuta cuando los datos que recibe el TCPLister son válidos.



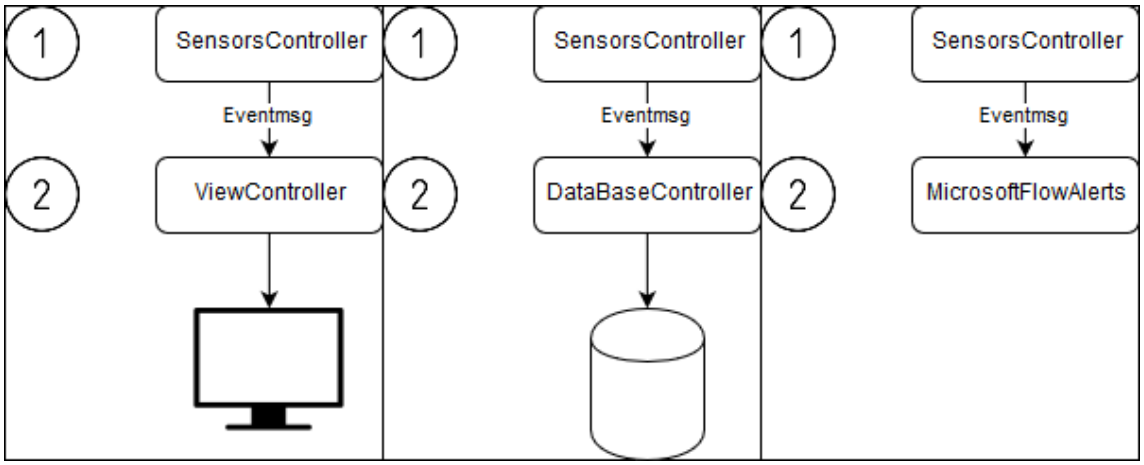
Este flujo se ejecutará periódicamente. El servicio de MicrosoftFlowAlerts pide los datos al controlador de la base de datos, y si algún sensor ha fallado, enviara una alerta a través de MicrosoftFlow.



Este flujo ocurre cuando el usuario envía una petición GET al SocketCSV; este pedirá los datos al controlador de la base de datos, y responderá a la petición del usuario.



Este servicio, que se ejecuta cíclicamente, envía los valores de los sensores a la plataforma de ThingSpeak.



Finalmente, estos tres flujos se ejecutan cuando hay una modificación en la configuración de los senores.

Tecnología Utilizada

En este proyecto,se han utilizado diversas tecnologías.

- Windows 10 IoT
- .NET
- C#
- JavaScript
    - jQuery

Windows 10 IoT

Es un sistema operativo desarrollado por Microsoft; esta orientado a ser utilizado como su propio nombre indica con aplicaciones del ámbito de IoT.

Hay dos versiones, el Core y la Enterprise, esta última ofrece una versión mas completa, puesto que puede implementar aplicaciones tanto de Universal Windows Platform como de Win32, mientras que la versión llamada Core simplemente puede implementar aplicaciones UWP. No obstante se ha utilizado la versión Core, puesto que es la única que soporta la arquitectura de CPU que tiene la RaspberryPI es decir la de ARM.

.NET

Es una plataforma de código abierto utilizada para desarrollar diferentes tipos de aplicaciones, se pueden usar distintos lenguajes, editores y librerías para desarrollar aplicaciones para Web, escritorio y en este caso IoT.

Los lenguajes que soporta .NET son C#, F#, o Visual Basic. En este proyecto, se va a utilizar C#.

C#

Es un lenguaje de tipado fuerte orientado a objetos que facilita el desarrollo de una gran variedad de aplicaciones que utilizan el Framework .NET. Se puede utilizar tanto para crear aplicaciones de escritorio para Windows, Servicios Web, aplicaciones cliente-servidor, y mucho más.

Su sintaxis es fácil de entender y aprender, sobretodo si el desarrollador tiene familiaridad con otros lenguajes como puedan ser C, C++ o Java. En el caso de tener conocimiento previo se puede empezar a desarrollar aplicaciones con C# en muy poco tiempo.

JavaScript

Es un lenguaje scripting del lado cliente. Principalmente es utilizado para mejorar la iteración de un usuario con una página web, es decir, una página web puede ser mucho más interactiva gracias al uso de JavaScript. También se está utilizando ampliamente en el desarrollo de juegos y de aplicaciones para smartphones.

Pese a su nombre no tiene nada que ver con Java, algunas de sus principales diferencias, son que JavaScript presenta un tipado débil y es de carácter asíncrono mientras que Java tiene un tipado fuerte y es secuencial.

jQuery

Es una librería ampliamente utilizada de JavaScript. Permite simplificar la manipulación y el manejo de eventos de un documento HTML, y gracias a Ajax es mucho más fácil el intercambio de datos con un servidor web sin la necesidad de recargar una página.

5. Desarrollo de la solución propuesta

Esta parte se centrará más en el desarrollo del coreógrafo y de cada servicio, puesto que los su función ya han sido explicados en el apartado anterior.

Base de datos

Para empezar, la base de datos que se ha utilizado es SQLite[22] debido a que es una base de datos pequeña y puede ser soportada por la RaspberryPI.

Tabla DATA

Es la tabla que se utiliza para almacenar la información recibida, por lo tanto un fallo podría ocasionar la pérdida de datos, cosa que hemos de evitar a toda costa. Se ha creado siguiendo este esquema:

```
CREATE TABLE IF NOT EXISTS DATA

(Primary_Key INTEGER PRIMARY KEY AUTOINCREMENT,

SENSOR_ID NUMBER NOT NULL,

TYPE VARCHAR2(15) NOT NULL,

VALUE_LIST VARCHAR(30) NOT NULL,

TIME_STAMP VARCHAR2(20) NOT NULL)
```

Tabla SENSORS

Se guardan los sensores que están activos.

```
CREATE TABLE IF NOT EXISTS Sensors

(SENSOR_ID NUMBER PRIMARY KEY AUTOINCREMENT)
```

Coreógrafo

MainPage

Es la clase que se ejecuta al arrancar el proyecto.

Public MainPage()

En este método, se crea el entorno de coreografía, se crean los distintos servicios a los cuales se les van a asignar un ID con la finalidad de que al enviar mensajes de coreografía, el coreógrafo sea capaz de hacer llegar el mensaje al servicio adecuado, una vez creados y configurados se asignan al entorno de coreografía.

Private void startCoreographer(object sender,RoutedEventArgs e)

Este es un método que se dispara tras un evento, en este caso, al pulsar sobre el botón Start del apartado de configuración.

Es aquí dónde primero se inicializa la base de datos (se crea las tabla si no está ya creada) , y a continuación se crean y añaden al coreógrafo los servicios parametrizables, pues para poder crear estos servicios hay que tener los valores de configuración.

Finalmente, se pone en marcha el entorno de coreografía.

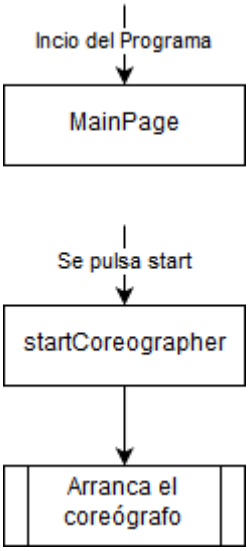


Figura 6: Flujo MainPage

Cabe destacar que una vez arrancado el coreógrafo, también se pueden añadir servicios.

```
DataBaseController db = new DataBaseController(sensores.Text);
db.setIdentificadorAccion("GestionDeDatos.SinProceso.DB");
env.AddIService("database", db, db.getIdentificadorAccion());
```

Texto 1: Ejemplo de creación y configuración de un servicio

Servicios

Antes de empezar con los servicios, se explica los métodos más importantes de la interfaz IServicio que vamos a utilizar, para tener una mejor visión sobre el funcionamiento.

IServicio

Esta interfaz pertenece al paquete Sabien.Portable.Choreography.

Bool Notificar(XFIPAMSG msg)

Este método es donde llegarán los mensajes de coreografía, una vez en este método se llamará al método indicado según el contenido del mensaje.

Void Start()
Este método se ejecuta cuando se pone en marcha el entorno de coreografía.

Void Stop()
Cuando el coreógrafo se para, este método se ejecuta.

TCPListener : IServicio
Es el encargado de procesar los datos enviados por los sensores.

Public TCPListener()
Se instancia un nuevo ModemTCP, este objeto es parte de las librerías de Sabien, que nos permitirá recibir los mensajes enviados por los sensores.

Public Start()
Se configura el ModemTCP para que sea capaz de recibir los mensaje y tratarlos correctamente.

Public Stop()
Se para el listener.

Private void Com_OnTCPMensajeRecibido(MemoryStream msg)
Es el método ejecutado al recibir un mensaje por TCP, este mensaje llamará al método encargado de buscar los delimitadores.

Private void startDelimiter(String msg)
Se busca el delimitador de inicio con tal de saber donde empieza el mensaje. Y se llama al método encargado de buscar el delimitador de fin.

Private void endDelimiter(String msg)
Se realiza la búsqueda del delimitador de fin para poder tener el mensaje completo. A continuación se envía el mensaje completo al método correspondiente.

Private void sendData(String msg)
Recibe el mensaje completo, y lo envía mediante un mensaje de coreografía a los servicios que necesitan esta información. Si se trata de un valor no válido (Checksum erróneo) o si es un valor NULL, llegará al servicio de visualización y al controlador de la base de datos y si es un valor válido llegará a los dos anteriores y al servicio de alertas.

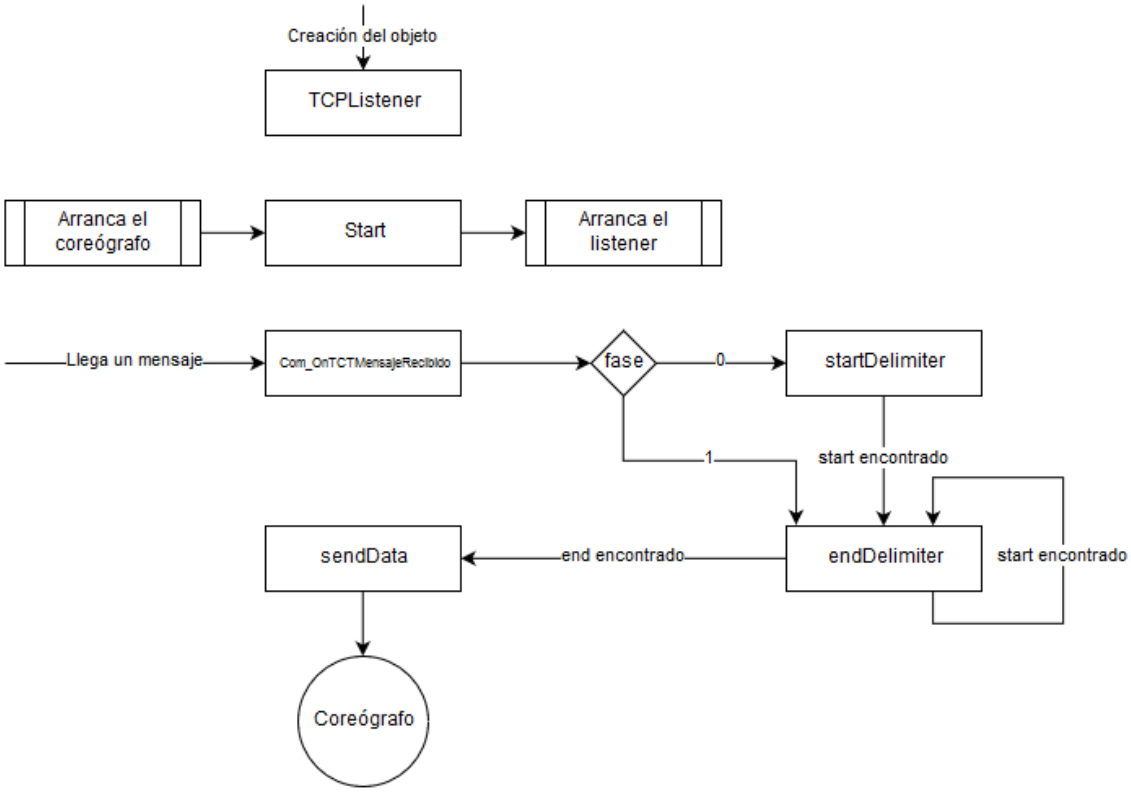


Figura 7: Flujo TCPListener

Cuando llega un nuevo mensaje, éste es procesado por el método responsable de ello y lo envía a buscar el delimitador correspondiente, esto lo hace dependiendo del valor de una variable global a la que hemos denominado *fase*, cuyo valor indicia si se esta buscando el delimitador de inicio "0" o el delimitador de fin "1".

Una vez encontrado el delimitador de inicio, se cambia la variable global a "1" y se llama al método que busca el delimitador de fin, enviando como parámetro el mensaje recibido desde el delimitador de inicio hasta el fin del mensaje.

A continuación, se encuentra el delimitador de fin, por lo tanto ya se tiene el mensaje completo, se actualiza el valor de la *fase* a "0" y se envía al método encargado de enviar el mensaje de coreografía el mensaje completo como parámetro.

Finalmente, una vez el mensaje completo llega al último método, se procesa para poder obtener los valores enviados por el sensor y se envía a través del coreógrafo.

```
XFIPAMSG msgC = new XFIPAMSG(
this.getIdentificadorAccion(),
"GestionDeDatos",
XFIPAMSGTipoProtocolo.eventmsg
);

Dictionary<string, object> parametros = new Dictionary<string, object>();
parametros.Add("sensor", sensor);
parametros.Add("type", types[type]);
parametros.Add("values", values);

msgC.Contenido = new SOAPMSG("ReceiveData", parametros);

OnPropagateCommand(this, msgC);
```

Texto 2: Ejemplo de creación y envío de un mensaje de coreografía

Cabe la posibilidad de que un mensaje no llegue completo en un paquete TCP y que el resto del mensaje llegue en el siguiente paquete. Para solucionar este problema, se definen dos variables globales, *partialMSG* donde se guardará la primera parte del mensaje en caso de que llegue cortado y *partial*, que indica si hay o no un mensaje cortado.

Para ello, en el método encargado de buscar el delimitador de fin, si se llega al final del mensaje y no se ha encontrado, actualizará el valor de la variable *partial* a TRUE y guardará la parte del mensaje en *partialMSG*. Por lo tanto, al llegar un nuevo paquete TCP, como el valor de la *fase* es "1" se llamará directamente a buscar el delimitador fin, que una vez encontrado se comprobará la variable *partial* y en caso de ser afirmativa, se concatenará la variable *partialMSG* con el mensaje recibido hasta el delimitador de fin y se llamará al método encargado de enviar el mensaje de coreografía.

En el caso de que la parte final se pierda, también se comprueba dentro del método donde se busca el delimitador fin, si es un mensaje *partial* y contiene el delimitador de inicio, en cuyo caso se actualiza el valor de *partial* a FALSE, y hace una llamada a si mismo con el mensaje recibido desde el delimitador de inicio hasta el fin de este.

DataBaseController : BasicService

Es aquí donde se manipulará la base de datos. Este servicio, no implementa la interfaz IServicio, sino que extiende la clase BasicService la cual implementa la interfaz.

Public override Start()

Se inicializa la base de datos.

Public override bool Notificar(XFIPAMSG msg)

Se gestionan los mensajes de coreografía recibidos.

Public void ReceiveData(XFIPAMSG msg)
Recibe un mensaje de coreografía enviado por el listener, que contiene la información enviada por un sensor. Una vez procesado el contenido, se inserta en la base de datos.

Public SqliteDataReader GetData(String query)
Este método se encargará de ejecutar la consulta a la base de datos y devolver los datos.

Public void DeleteData(String query)
Este método se encarga de eliminar los datos de la base de datos.

Public SensorChange(XFIPAMSG)
Recibe un mensaje de coreografía enviado por el servicio de control de sensores, dependiendo del valor de la variable *action*, llamará a newSensor o deleteSensor.

Public newSensor(int id)
Añade un sensor a la tabla de SENSORS.

Public deleteSensor(int id)
Elimina un sensor de la tabla de SENSORS.

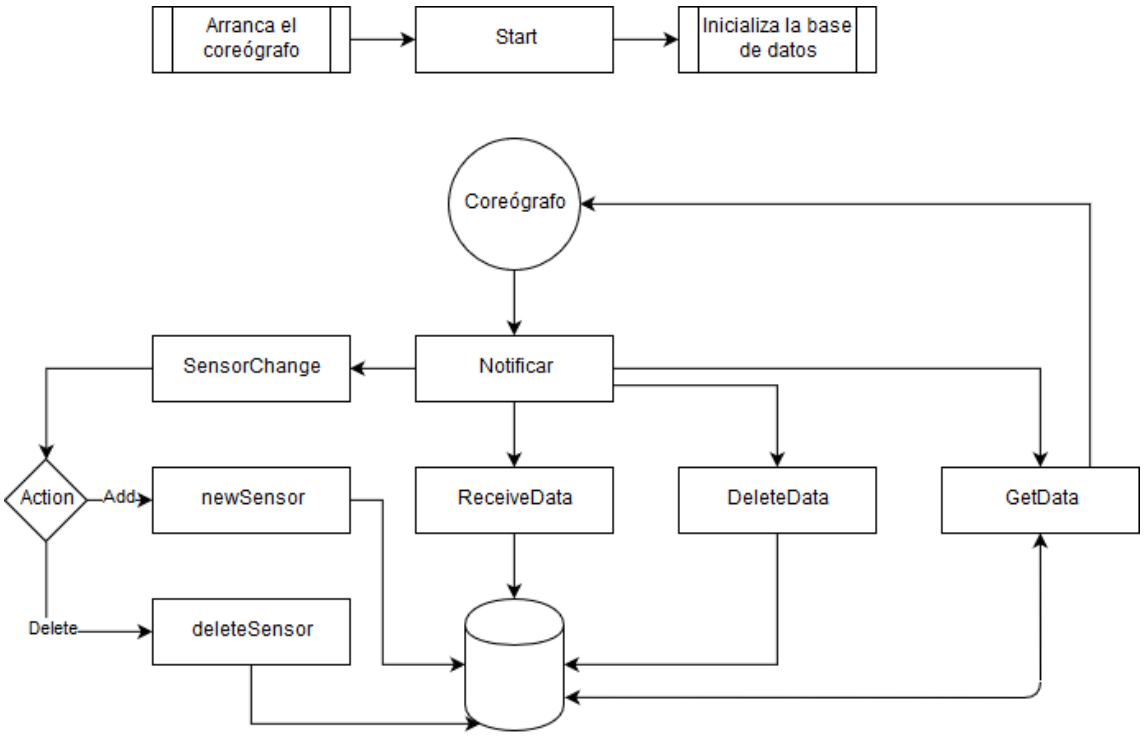


Figura 8: Flujo DataBaseController

Primero se tratan los mensajes de coreografiá para saber que métodos se han de ejecutar.

```
public override bool Notificar(XFIPAMSG msg)
{
    if (msg.protocolo == XFIPAMSGTipoProtocolo.eventmsg)
    {
        try
        {
            switch (msg.Contenido.metodo)
            {
                case "ReceiveData":
                    ReceiveData(msg);
                    break;
                case "SensorChange":
                    SensorChange(msg);
                    break;
                default:
                    return base.Notificar(msg);
            }

        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            return base.Notificar(msg);
        }

        return true;
    }
    else
    {
        return base.Notificar(msg);
    }
}
```

Texto 3: Ejemplo del tratamiento de los mensajes de coreografía

Para empezar, se comprueba el tipo de protocolo del mensaje, puesto que al tratarse de una clase que hereda de BasicService, los mensajes con protocolo Request son gestionados automáticamente y lo que hará será buscar los métodos que tengan el mismo nombre que el que contiene el mensaje de coreografía y esté etiquetado con el valor [PublicMethod].

A continuación se obtiene cual es el método que se debe ejecutar, y se hace una llamada a éste.

```
[PublicMethod]
public SqliteDataReader GetData(String query)
```

Texto 4: Ejemplo de un método publico del servicio

Si solo se implementa la interfaz IServicio, se deben tratar todos los mensajes.

Siguiendo con el funcionamiento del servicio, cuando se trata de insertar los datos que han sido enviados por el listener, simplemente se procesa el mensaje de coreografía para obtener los valores y se insertan en la base de datos. Por el contrario si se trata de eliminar datos, ejecuta la instrucción recibida.

Otra función que implementa este servicio, es realizar las consultas a la base de datos, y devolver los resultados al servicio solicitante. La consulta será proporcionada por dicho servicio.

Finalmente, si hay alguna modificación de los sensores activos, el servicio SensorsController enviará un mensaje de coreografía indicando si se añade o si se elimina y el ID del sensor, el controlador lo modificará en la base de datos.

ViewController : IServicio

Se muestra la información por pantalla siguiendo el diseño definido en ViewController.xaml.

Public ViewController()

Inicializa el controlador gráfico.

Public Start()

Se inicia la visualización.

Public Stop()

El controlador gráfico, se detiene.

Public bool Notificar(XFIPAMSG msg)

Trata correctamente los mensajes de coreografía recibidos.

Public void ReceiveData(XFIPAMSG msg)

Este método recibe la información de un sensor gracias a un mensaje de coreografía enviado por el listener. Se obtiene el contenido de dicho mensaje y se muestra por pantalla.

Public SensorChange(XFIPAMSG)

Recibe un mensaje de coreografía enviado por el servicio de control de sensores, dependiendo del valor de la variable *action*, llamará a printSensor o removeSensorScreen.

Public printSensor(int id)

Añade un sensor a la lista de sensores que se muestra por pantalla

Public removeSensorScreen(int id)
Elimina un sensor de la lista de sensores que se muestra por pantalla.

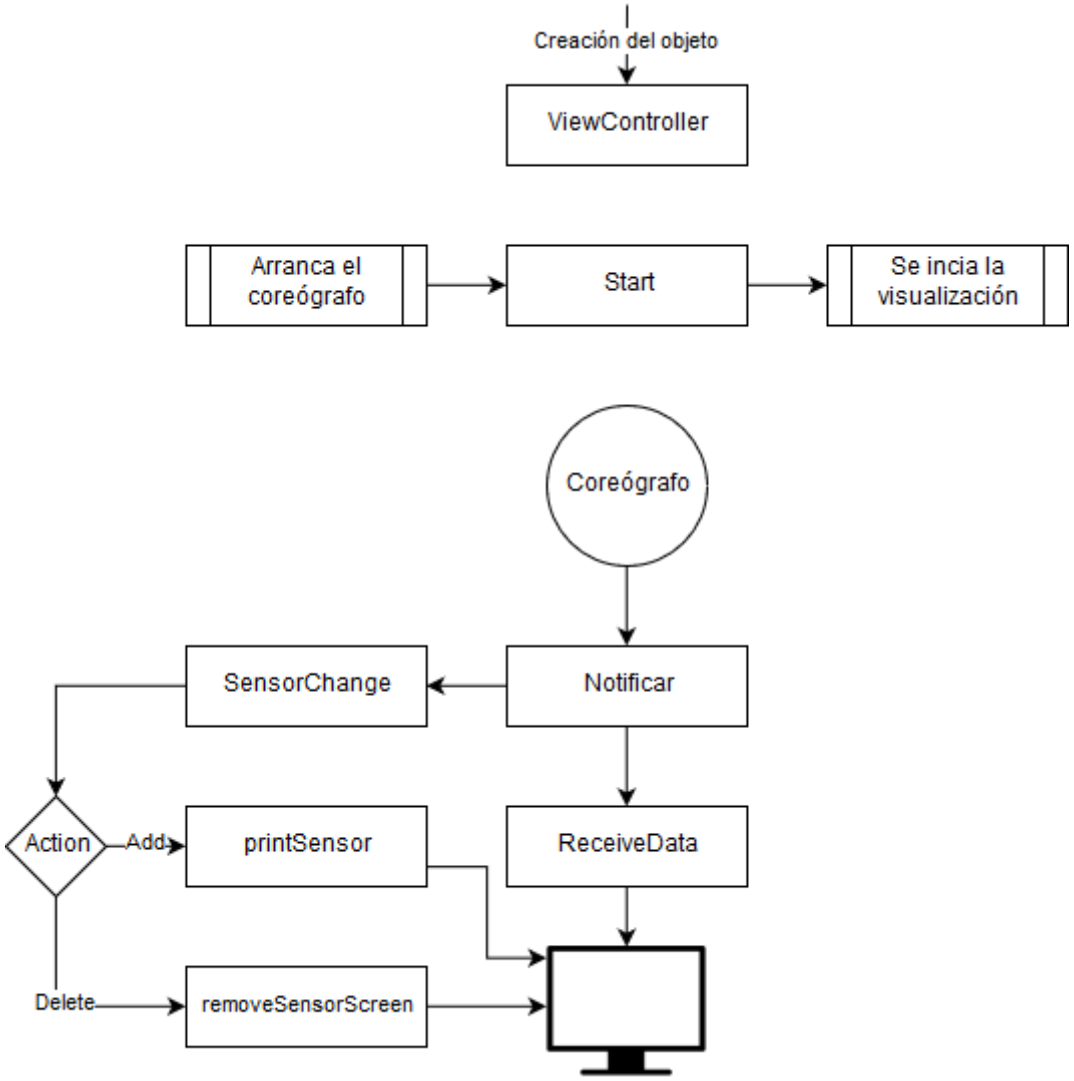


Figura 9: Flujo ViewController

Este es el servicio más sencillo que se encuentra en este proyecto, simplemente muestra por pantalla los valores recibidos y los sensores activos. Los valores vienen en un mensaje de coreografía que ha enviado el listener TCP, los sensores activos son inicializados al principio y se van modificando a medida que el SensorsController va enviando mensajes.

ThingSpeak : IServicio
Este servicio es el encargado de recoger la información y enviarla a la plataforma de ThingSpeak.

Public ThingSpeak(Int TS)
Se crea un cliente HTTP para enviar los datos al servidor de ThingSpeak. Y se da valor a la variable que indica la frecuencia de ejecución.

Public Start()
Iniciamos periódicamente el método encargado de enviar los datos.

Public Stop()
Se deja de enviar datos a ThingSpeak.

Public bool Notificar(XFIPAMSG msg)
Hace una gestión de los mensajes de coreografía recibidos.

Public void RetrieveData()
Envía un mensaje de coreografía al controlador de la base de datos, con tal de obtener el valor medio de cada zona y sensor de los datos enviados desde la última vez que se consultó.

Public void GetDataResponse(SqliteDataReader GetDataResult)
Obtiene el resultado del mensaje enviado en el método anterior y lo envía a ThingSpeak.

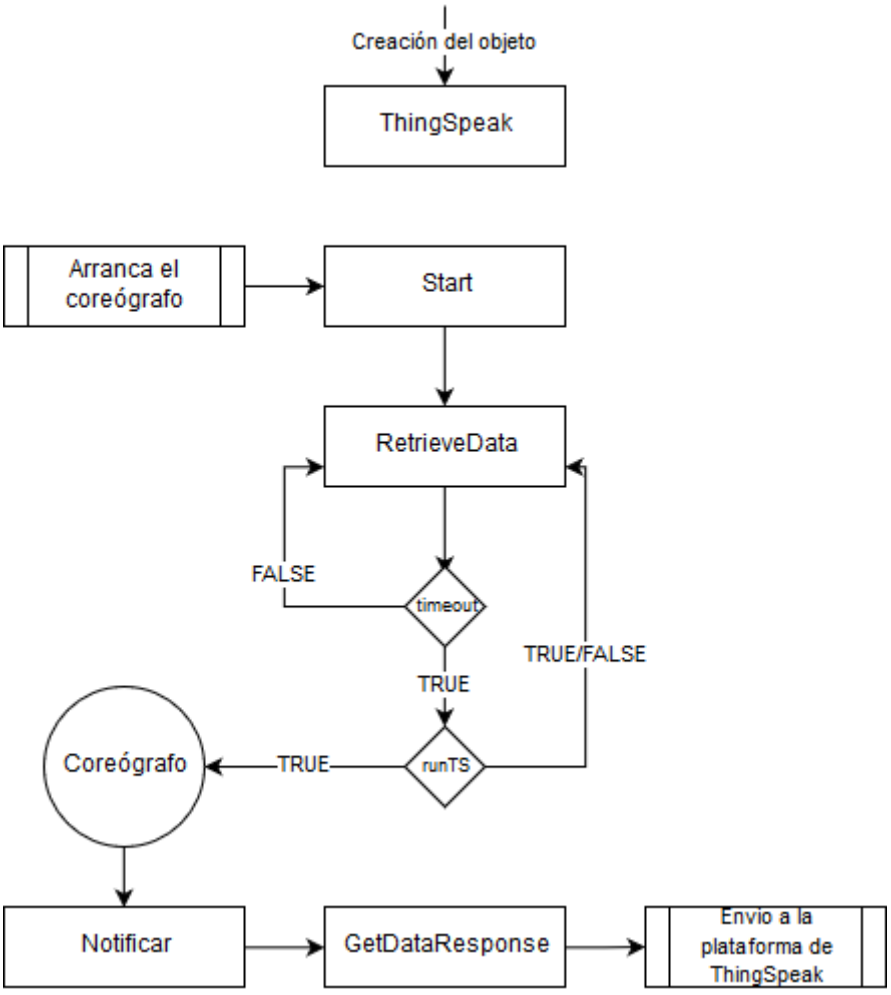


Figura 10: Flujo ThingSpeak

Los datos se van a enviar periódicamente, el intervalo de tiempo viene definido por el valor que ha introducido el usuario en el apartado de configuración. Este valor es enviado al crear el servicio.

Cabe destacar que no todos los valores son enviados a ThingSpeak, debido a que la versión gratuita solo permite enviar un cierto número de datos al día. Es por ello que se obtiene los valores de cada sensor enviando un mensaje de coreografía al controlador de la base de datos, que devolverá el resultado de la consulta que contenía el mensaje. Una vez recibidos los datos enviados por los sensores, se crea un diccionario donde se guarda el ID del sensor y un objeto que se ha llamado *dataReceived,* este objeto contiene dos variables *sum* y *dataReaded*. Puesto que los valores de cada sensor vienen en un string separados por comas, se obtienen los valores de dicho string, se añaden a *sum* y se le suma a la variable *dataReaded* el número de valores que se han añadido a la variable *sum* , al final se envía la media de cada sensor.

Para no obtener valores duplicados, se crea una variable llamada *timeStamp* que se inicializa al arrancar el servicio con el valor actual de la fecha del sistema, y se obtienen solamente los valores posteriores a ese momento, una vez obtenidos los valores, se actualiza la variable para la próxima vez.

MicrosoftFlowAlerts : IServicio

Es aquí donde se enviarán las alertas al usuario en caso de que algún sensor falle o haya algún valor fueras de los límites marcados.

Public MicrosoftFlowAlerts(Int numSensors, Int MF)

Configura el cliente HTTP que será utilizado posteriormente para el envío de las alertas. Además, se da valor a los valores parametrizables.

Public Start()

Se inicia cíclicamente el método encargado de enviar los datos, según el valor enviado al crear el servicio.

Public Stop()

Deja de comprobar el estado de los sensores.

Public bool Notificar(XFIPAMSG msg)

Gestiona los mensajes de coreografía recibidos.

Public void ReceiveData(XFIPAMSG msg)

Recibe los datos de los sensores gracias al mensaje enviado por el TCP listener si se trata de un sensor de humedad, comprueba que no haya valores ni demasiado elevados ni demasiado bajos, de ser así, envía una notificación.

Public void requestSensors()

Mediante un mensaje de coreografía enviado al controlador de la base de datos, pide los sensores que están activos desde un determinado time stamp.

Public void GetDataResponse(SqliteDataReader GetDataResult)

Recibe la respuesta al mensaje enviado por el método anterior, procesa el mensaje y obtiene los sensores que no están activos y llama al método encargado de enviar la alerta.

Public void sendAlert(String msg)

Crea un mensaje de petición HTTP con un mensaje de alerta determinado y lo envía utilizando el cliente anteriormente configurado.

Public SensorChange(XFIPAMSG)

Recibe un mensaje de coreografía enviado por el servicio de control de sensores, dependiendo del valor de la variable *action*, llamará a newSensor o deleteSensor.

Public newSensor(int id)

Añade un sensor a la lista de sensores configurados.

Public deleteSensor(int id)

Elimina un sensor de la lista de sensores configurados,

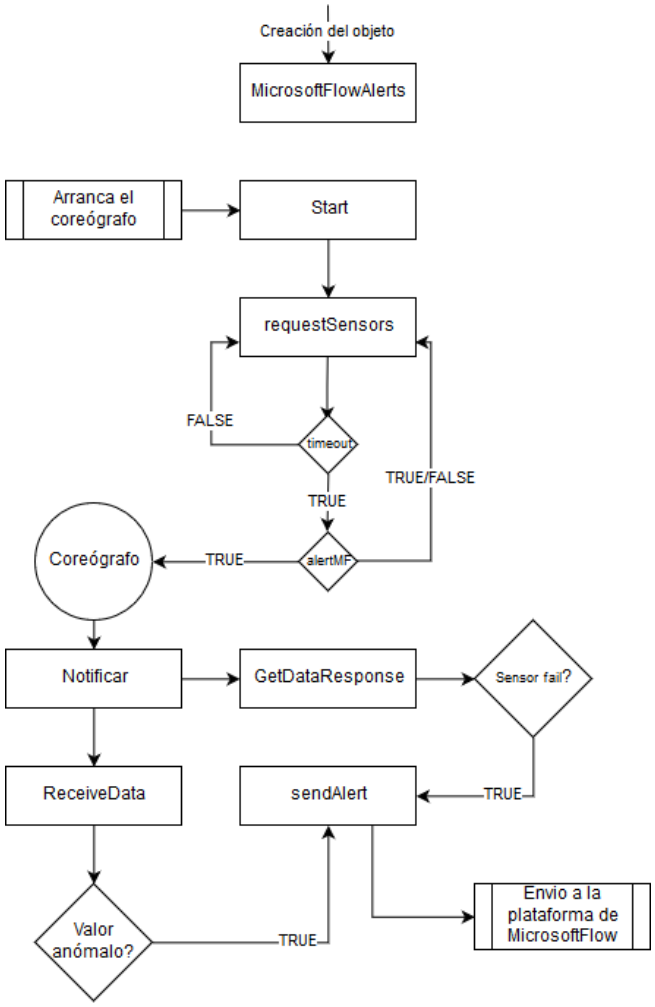


Figura 11: Flujo MicrosoftFlowAlerts

Este servicio notifica al usuario cuando hay un comportamiento anómalo en el huerto.

El primer tipo de alerta que envía es cuando la humedad es demasiado alta o demasiado baja. Por lo tanto el mensaje que es enviado por el listener TCP, también llega a este servicio que procesa el mensaje para obtener los valores y ver si alguno esta por encima del máximo recomendable o en caso contrario, por debajo del mínimo. Si algún valor cumple alguna de estas condiciones, se notifica por correo electrónico al usuario o usuarios configurados en Microsoft Flow.

La segunda alerta que puede enviar es cuando alguno de los sensores ha dejado de funcionar. Esta comprobación al igual que en el servicio de ThingSpeak, se ejecuta cíclicamente según el valor introducido en la configuración. Este valor se pasa como parámetro al inicio del servicio junto con el número de sensores que van a estar enviando datos.

Para obtener que sensores han dejado de funcionar, al iniciar el servicio una vez obtenido el valor referente a los sensores que fue introducido en la configuración, se crea una lista de enteros que contiene el ID de cada sensor, también se crea una variable llamada *timeStamp* que se inicializa a la fecha actual del sistema; esta variable se actualiza cada vez que se comprueban los sensores.

El método que comprueba los sensores, primero envía un mensaje de coreografía dirigido al controlador de la base de datos, este devolverá el resultado de la consulta, en este caso los distintos sensores que han enviado desde un time stamp determinado. El ID de estos sensores se guarda en una lista. Seguidamente, se crea un Array de los sensores que han fallado, siendo estos los sensores que están en la lista de sensores creada al inicio pero no en la lista de sensores activos. Finalmente, para cada sensor que ha dejado de emitir, se envía una alerta por correo electrónico al usuario o usuarios configurados en el flujo de Microsoft Flow.

Tanto para el primero como para el segundo tipo de alerta, el servicio envía mediante HTTP una petición GET a la URL del flujo de Microsoft Flow, este flujo espera un objeto JSON en el cuerpo de la petición, en este caso solo hay un valor, el mensaje que se ha de enviar. Una vez recibida la petición, el flujo envía el contenido del mensaje por correo electrónico.

SocketCSV

Este archivo, contiene dos clases

Public class SocketCSV : BasicService

Es la clase del servicio.

Public override Start()

Se llama a Run() para que inicie el servidor.

Public async Task Run()

Configura el servidor de HTTP, con el puerto en el que va estar escuchando y con el contralor que será el encargado de recibir las peticiones. En este caso, la clase ParameterController, una vez configurado, se iniciará.

Public class ParameterController

Se encarga de gestionar las peticiones que llegan al servidor.

Public IGetResponse GetWithSimpleParameters(String day,String month, String year)

Gestiona las peticiones de tipo GET, que son las que se utilizan. Y devuelve los datos del día deseado en formato CSV.

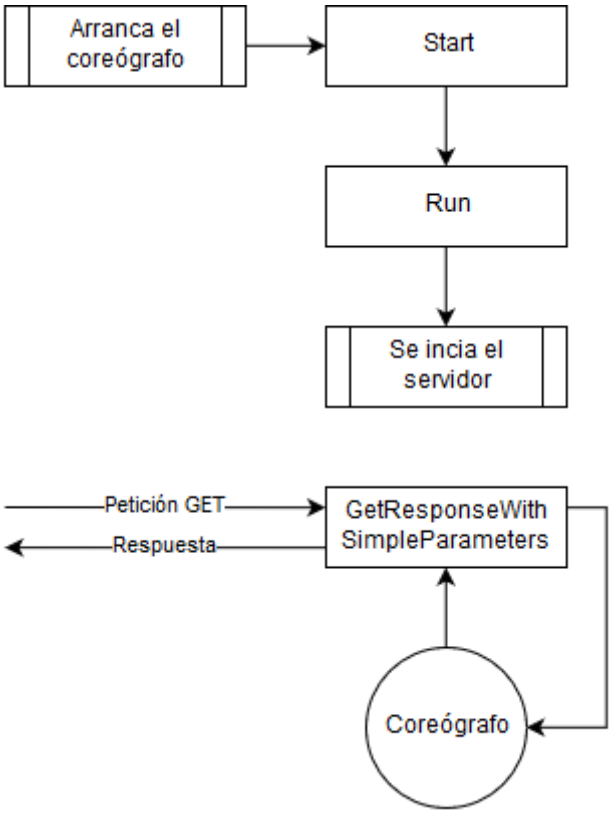


Figura 12: Flujo SocketCSV

Es el servicio encargado de devolver los datos en CSV. Este servicio implementa un servidor HTTP, que gestiona las peticiones de tipo GET que contienen la fecha de la cual se quieren obtener los datos, y responde con estos en formato CSV. Aquí podemos ver un el esquema de la URL a la que se llama:

```
http://IP:PUERTO/api/date/{dia}/{mes}/{año}
```

Para facilitar el uso de este servicio, se ha creado un archivo HTML con un formulario donde el usuario debe rellenar la IP, el puerto y la fecha, una vez enviado el formulario mediante JavaScript se genera la URL y se gestiona la respuesta de la petición GET con tal de que el usuario se descargue el archivo CSV.

Una peculiaridad de este servicio es que como debe obtener los valores de la base de datos para cada petición, no puede lanzar un mensaje de coreografía y esperar la respuesta en el método de

respuesta(“GetDataResponse”) sino que debe realizarse todo en el mismo momento. Es por ello que se optó por utilizar el servicio Proxy que ofrece el coreógrafo de modo que se envía un mensaje de coreografiá y el método se queda esperando la respuesta.

```
ChoreographyServicesProxy ProxyManager = ChoreographyEnvironment.getCurrentEnvironment<StandardChoreographyEnvironment>().ServicesProxy;
IDataBaseController t = ProxyManager.getSingleProxy<IDataBaseController>();
SqliteDataReader reader = t.GetData(query);
```

Texto 5: Ejemplo del uso del proxy

Para el uso del Proxy, hace falta la creación de una interfaz que se utilizará para llamar al método deseado. Una vez creada la interfaz, la clase del servicio al que llamamos debe implementarla.

Al utilizar el Proxy, el servicio se detiene, esto puede impedir que otras peticiones que lleguen en ese momento no fueran atendidas, pero al tratarse de un momento puntual que no ocasionará ningún fallo grave como pudiera ser la pérdida de datos, se puede quedar detenido por unos instantes.

SensorsController : UserControl, IServicio

Es el encargado de notificar a los servicios si ha habido algún cambio en los sensores que están en ejecución.

Public SensorsController()

Inicializa el controlador gráfico.

Public Start()

Se inicia la visualización.

Public Stop()

El controlador gráfico, se detiene.

Public void addSensor(object sender,RoutedEventArgs e)

Este método se ejecuta después de que el usuario haya pulsado el botón “Añadir”. Envía un mensaje de coreografía a los servicios que han de llevar un control de los sensores y llama al método printSensor.

Public void startCoreographer(object sender,RoutedEventArgs e)

Cuando el usuario pulsa el botón “Eliminar”, se ejecuta este método.Envía un mensaje de coreografía a los servicios que han de llevar un control de los sensores y llama al método removeSensorScreen.

Public printSensor(int id)

Añade un sensor a la lista de sensores que se muestra por pantalla

Public removeSensorScreen(int id)

Elimina un sensor de la lista de sensores que se muestra por pantalla.

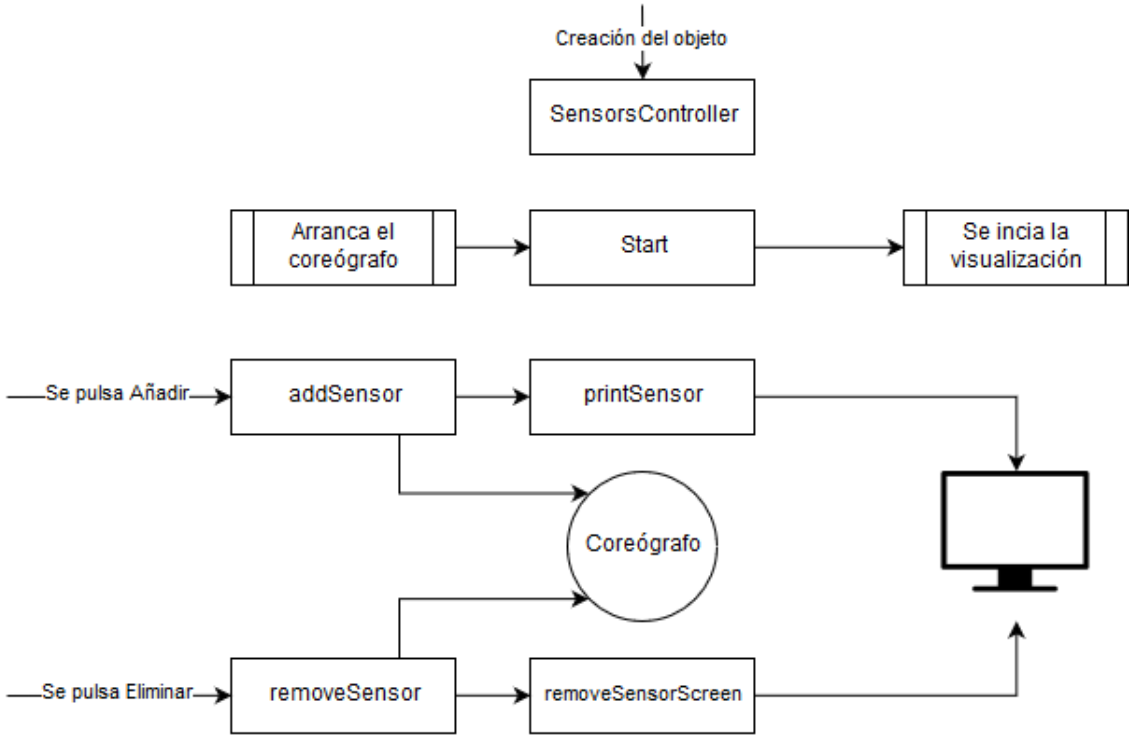


Figura 13: Flujo SensorsController

Cuando el usuario introduce un ID y pulsa el botón de “Añadir” o “Eliminar”, se modifica en pantalla los sensores que están activos, y se notifica a los servicios que deben llevar un control de los sensores activos.

6. Implantación

Tanto durante el desarrollo del proyecto, como para su puesta final en producción, se ha desplegado el programa en una RaspberryPI.

Para empezar, se necesita una RaspberryPI, en este caso se ha optado por el modelo “Raspberry PI Model 3”, ya que tiene la posibilidad de utilizar Wi-Fi, puesto que los sensores enviarán los datos mediante este método y no por Ethernet. La RaspberryPI ha sido proporcionada por la tutora. Una vez en posesión de ésta, se instala la versión de Windows para los sistemas IoT, Windows 10 IoT. Se opta por esta opción puesto que el coreógrafo está implementado utilizando .NET, y por lo tanto se reducen las posibilidades a este sistema operativo, que aunque a priori se pudiera pensar que esto puede limitar el funcionamiento de la aplicación, no se ha tenido ningún problema de este tipo.

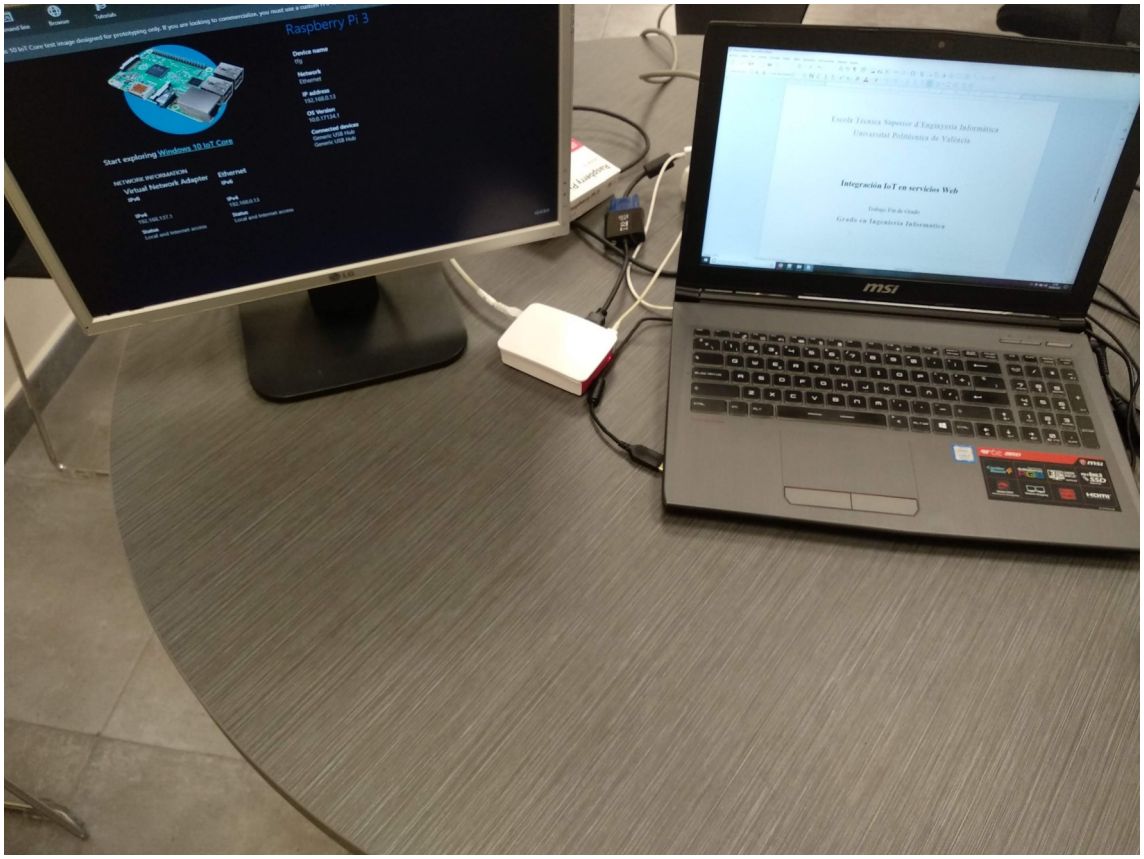
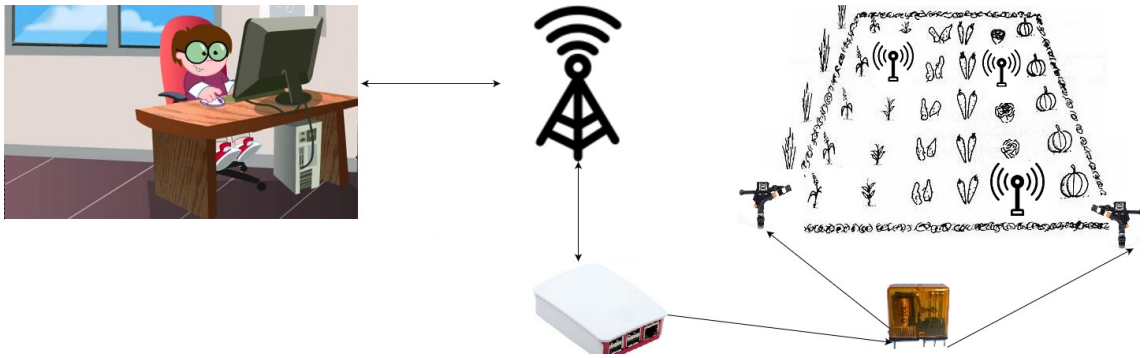


Foto 1: Implantación en local

Mediante esta configuración y utilizando un servicio de test detallado en el apartado de Pruebas, ha sido posible ir probando las distintas funcionalidades durante el desarrollo del proyecto.

También se desplegó la aplicación junto con los sensores, en este caso los sensores que se han utilizado son los de Texas Instruments CC3220[23].

Finalmente, se implantará todo en el instituto de secundaria.

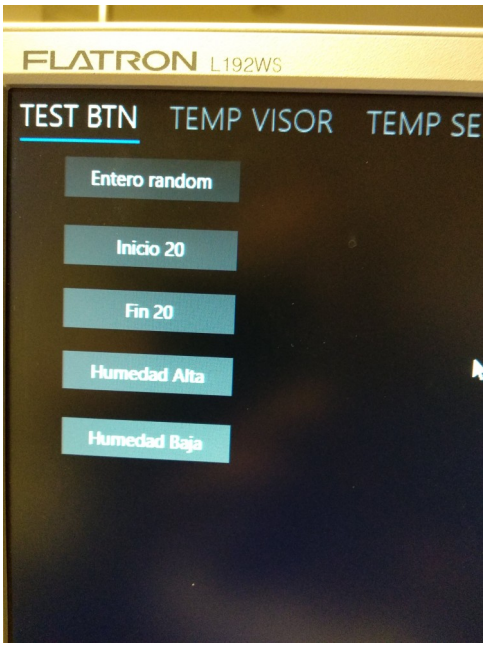


Dibujo 2: Ejemplo de la futura implantación en el instituto

7. Pruebas

Para probar el sistema sin la necesidad de desplazarse a la UPV, se añadió un servicio al coreógrafo llamado “Test” el cual al pulsar los botones lanzan un mensaje de TCP, emulando así a los sensores.

- Entero random: envía un mensaje completo con valores aleatorios.
- Inicio 20: Es la primera parte de un mensaje.
- Fin 20: Es la parte final del mensaje, completando el mensaje enviado por “Inicio 20”.
- Humedad Alta: Envía un mensaje con un valor de humedad superior al límite superior establecido.
- Humedad Baja: Envía un mensaje con un valor de humedad inferior al límite inferior establecido.

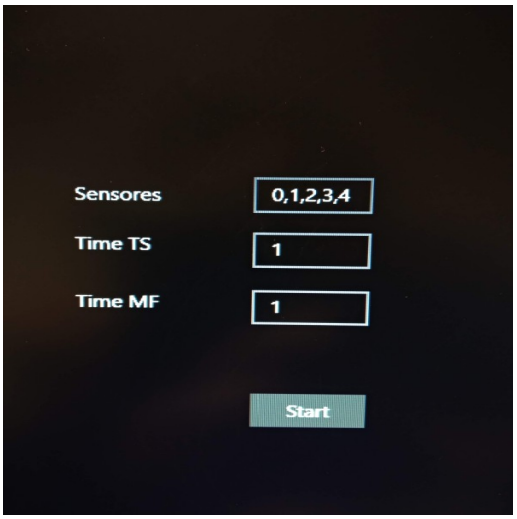


Estas pruebas están orientadas a comprobar el funcionamiento del sistema en función de los objetivos propuestos al inicio de este proyecto.

Se muestra a continuación el resultado.

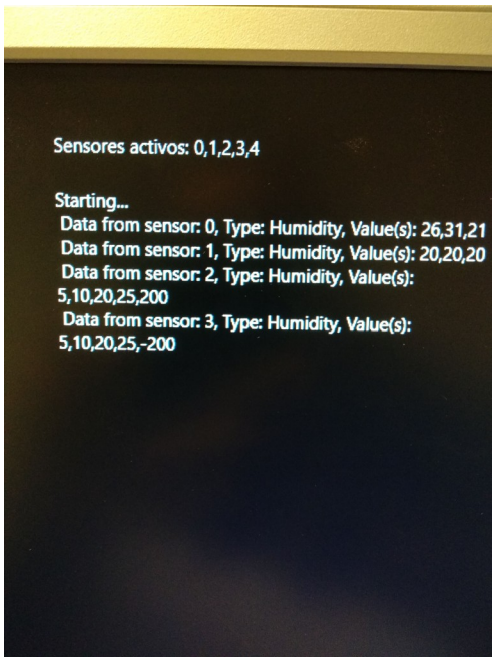
Configurar el coreógrafo antes de su ejecución.

Es en esta parte, donde se introducen los valores para la configuración del sistema, el número de sensores y la frecuencia con la que se ejecutarán los servicios de ThingSpeak y MicrosftFlowAlerts de este último simplemente la comprobación de los sensores activos. Los valores de los tiempos son en minutos. Al pulsar el botón Start, se inicia el coreógrafo.



Visualizar los datos por pantalla.

Se observa como los datos recibidos del listener TCP se muestran correctamente en pantalla.

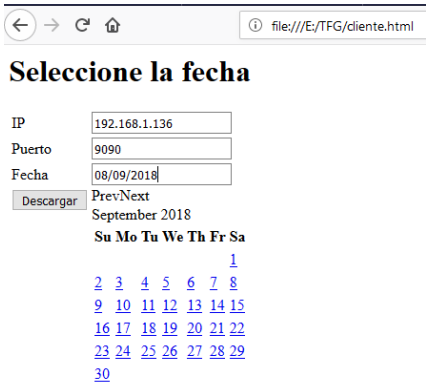


Definir las reglas sintácticas de los paquetes TCP

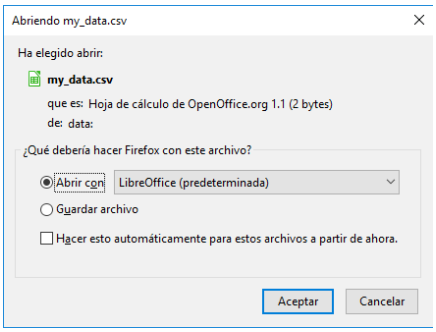
Al haber sido mostrados los datos correctamente en el servicio de visualización, se ha demostrado implícitamente que el servicio de TCP ha sido capaz de procesar los datos recibidos y enviarlos a los servicios correspondientes.

Descargar un archivo CSV con los datos de un día en concreto

Para ello mediante el archivo HTML que hemos creado, se introducen los valores de puerto, IP y la fecha.



Al pulsar descargar, se obtiene el archivo.



En el documento descargado, los cuatro valores corresponden con los enviados en el momento de realizar las pruebas.

	A	B	C	D	E	F	G	H
1	Sensor	Type	Time stamp	Values				
2	2	Humidity	08/09/2018 21:07:39	5	10	20	25	200
3	0	Humidity	08/09/2018 21:07:44	27	32	22		
4	3	Humidity	08/09/2018 21:07:44	5	10	20	25	-200
5	3	Humidity	08/09/2018 21:07:45	5	10	20	25	-200
6	3	Humidity	08/09/2018 21:07:47	5	10	20	25	-200
7	3	Humidity	08/09/2018 21:07:47	5	10	20	25	-200
8	0	Humidity	08/09/2018 21:09:28	26	31	21		
9	1	Humidity	08/09/2018 21:09:41	20	20	20		
10	2	Humidity	08/09/2018 21:09:44	5	10	20	25	200
11	3	Humidity	08/09/2018 21:09:47	5	10	20	25	-200
12	0	Humidity	08/09/2018 21:10:56	22	27	17		
13	1	Humidity	08/09/2018 21:10:59	20	20	20		
14	2	Humidity	08/09/2018 21:11:00	5	10	20	25	200
15	3	Humidity	08/09/2018 21:11:02	5	10	20	25	-200

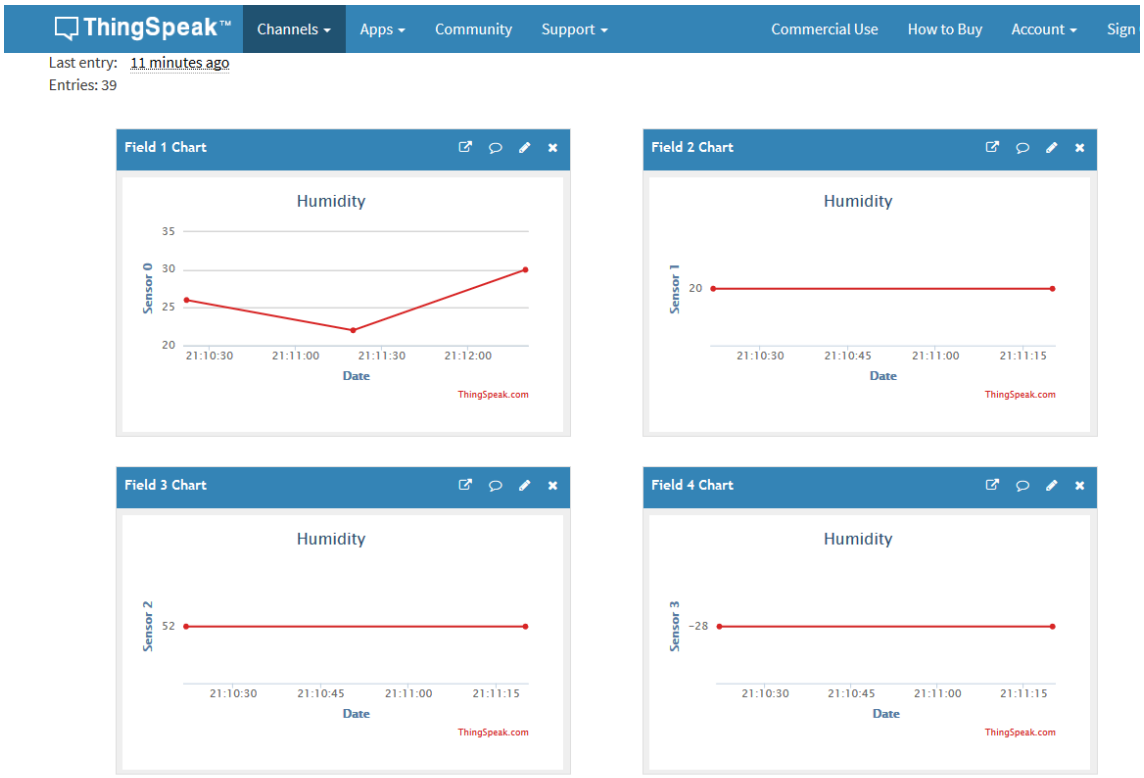
Almacenar los datos en la base de datos local

El servicio de CSV obtiene los valores de la base de datos local, almacenados por el controlador de la misma.

La tabla mostrada anteriormente contiene los valores enviados en esta sesión de pruebas por tanto se puede afirmar que el controlador de la base de datos ha almacenado correctamente los valores en la base de datos local.

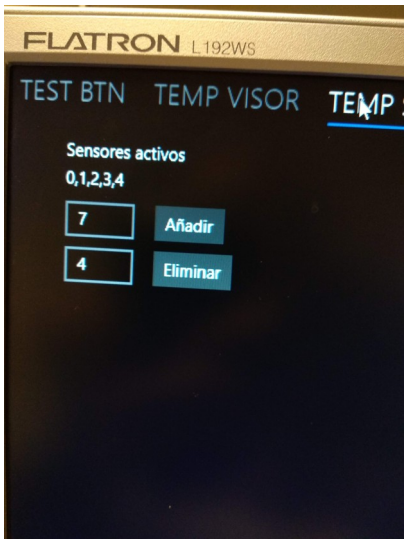
Enviar los datos a ThingSpeak

Se puede ver como los datos han llegado correctamente a ThingSpeak.

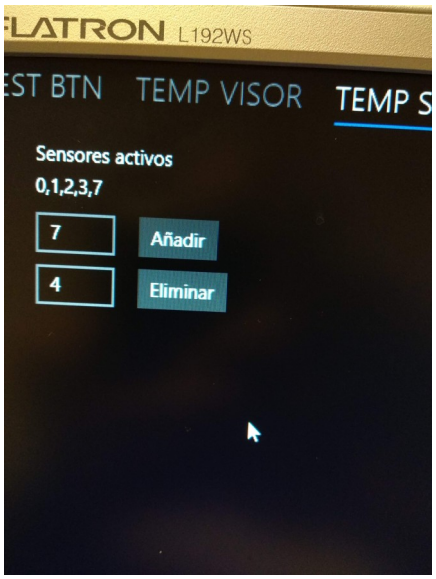


Modificación dinámica de los sensores activos

El usuario puede añadir y eliminar los sensores que se encuentran activos en el sistema, esto se hará a través de una interfaz gráfica.

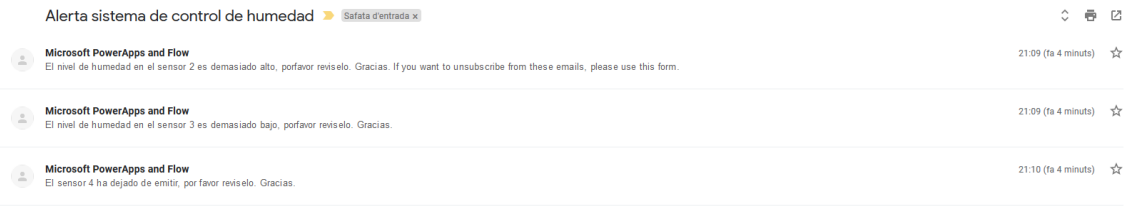


En este caso se va a eliminar el sensor 4 y se va a añadir el 7, a continuación se observa el resultado.



Enviar notificaciones a través de Microsoft Flow

En este caso deberían llegar tres notificaciones, una por el sensor número dos, otra por el sensor número tres puesto que ambos sensores han enviado valores anómalos de humedad y otra respecto al sensor cuatro puesto que se ha configurado como activo y no se ha recibido ningún dato.



También se ha realizado la prueba después de modificar los sensores, y en este caso se ha seguido recibiendo la alerta por los sensores dos y tres, pero ya no se ha recibido ninguna alerta por el sensor cuatro que ha sido eliminado, pero sí por el sensor siete.

Por lo que respecta al resto de objetivos, “Adaptar el coreógrafo para su uso en la RaspberryPI”, “Probar si el core de la RaspberryPI puede ejecutar el coreógrafo”, “Establecer el flujo de los datos mediante los mensajes de coreografía” han quedado más que demostrados en las pruebas anteriores.

8. Conclusiones

Como se ha podido ver en apartado de pruebas, no solo se ha demostrado que la RaspberryPI puede soportar el funcionamiento de un coreógrafo, sino que además es capaz de ejecutar todos aquellos servicios que se habían planteado al inicio de forma satisfactoria.

En este trabajo se han implementado distintas tecnologías, la de mayor peso es sin duda .NET, esta tecnología no se ha visto durante el transcurso del grado, y por lo tanto he aprendido su funcionamiento, cosa que considero será de gran utilidad en mi futuro laboral.

Comentar que una de las dificultades de este proyecto, fue la utilización del coreógrafo, puesto que al haber sido desarrollado por el equipo de investigación SABIEN-ITACA, aún habiéndome facilitado ejemplos del uso del coreógrafo, algunas veces no era posible obtener la información necesaria para el desarrollo del proyecto mediante ingeniería inversa de los ejemplos. Por lo tanto, debía recurrir a un miembro del equipo para poder resolverlo.

Relación del trabajo desarrollado con los estudios cursado

Dejando de un lado los aspectos básicos vistos en asignaturas de programación a lo largo del grado como por ejemplo, la utilización de interfaces, el uso de bucles, la utilización de Arrays, de listas…

Se puede relacionar este proyecto directamente con diversas asignaturas cursadas.

Integración de Aplicaciones

En esta asignatura de cuarto curso perteneciente a la rama de "Tecnologías de la información", se aprendió las distintas herramientas para el intercambio de datos entre distintos sistemas, entre ellas colas de mensajería utilizando RabbitMQ[24]. El símil que se puede hacer con respecto al proyecto, sería que los distintos sistemas en nuestro caso son los servicios y que la herramienta en lugar de ser RabbitMQ, sería el propio coreógrafo.

Redes

En este apartado se englobaría tanto la asignatura anual de segundo "Redes de Computadores", como las asignaturas de rama "Diseño y configuración de redes de área local " y "Redes corporativas", ya que sin ellas no habría sido posible tener el conocimiento necesario para entender el funcionamiento de TCP.

Bases de datos y sistemas de la información

Es en esta asignatura de tercer curso, donde se empezó a utilizar las bases de datos SQL algo nunca visto hasta ese momento en el grado, se aprendió principalmente los distintos tipos de consultas, como por ejemplo las subconsultas, las consultas con predicados existenciales entre otros, así como la creación de bases de datos. Es por ello que ha sido posible la implementación de una base de datos en este proyecto y la posterior consulta e inserción de datos en ella, todo y que solo se utiliza una tabla y las consultas e inserciones que se realizan en ella son más bien triviales, no hubieran sido posible sin haber cursado esta asignatura.

Desarrollo Web

Esta asignatura de tercer curso de la rama de "Tecnologías de la información", es donde se aprendió sobre las tecnologías web, por ejemplo se crearon distintas aplicaciones web con JavaScript y JQuery entre otros, y también se aprendió sobre REST y los distintos tipos de llamada, GET, POST, PUT, DELETE. Estos conocimientos han sido necesarios a la hora de realizar las distintas peticiones GET que hay en el proyecto, y como no, al crear el archivo HTML que utiliza JavaScript y JQuery para proporcionar los datos en un archivo CSV.

9. Trabajos futuros

Uno de los aspectos que no se ha podido cubrir en en este proyecto y que hubiera sido bastante interesante, es el desarrollo de un cliente donde desde su propio ordenador, el usuario pudiera visualizar los datos de una manera más gráfica, como por ejemplo los últimos datos enviados por cada sensor, además del valor medio del día actual entre otros. También se podría incorporar a este cliente la funcionalidad de descargar los archivos CSV.

Este proyecto al haber demostrado con éxito que una RaspberryPI es capaz de implementar un coreógrafo, abre las puertas a que cualquiera que desee implementar un sistema que procese, almacene, monitorice datos provenientes de cualquier sensor IoT, como por ejemplo, temperatura, polución… pueda hacerlo y añadir los servicios que quiera.

La vida de este proyecto no acaba aquí, puesto que una vez implando en el instituto se pueden ir añadiendo servicios tal y como vayan surgiendo las necesidades, por ejemplo un riego inteligente, cuando la humedad sea baja en vez de enviar un correo para que el responsable vaya y riegue, activar el sistema de riego automáticamente durante un periodo determinado. Otro servicio posible seria la comprobación de la predicción del tiempo y si va a llover, enviar una notificación al usuario para que se evite regar y así no desperdiciar agua, o si el riego está automatizado pues directamente que no se riegue.

10. Referencias

1. IoT

   https://www.domodesk.com/221-a-fondo-que-es-iot-el-internet-de-las-cosas.html

2. SOA

   https://www.bmc.com/blogs/service-oriented-architecture-overview/

3. Coreografía

   https://flylib.com/books/en/2.365.1/choreography.html

4. Sabien

   http://www.sabien.upv.es/

5. RaspberryPI

   http://www.mdpi.com/2079-9292/6/3/51/htm

6. .Net

   https://docs.microsoft.com/es-es/dotnet/

7. TCP – Kurose 6th Edition [3.5 Connection-Oriented Transport: TCP]

   https://www.bau.edu.jo/UserPortal/UserProfile/PostsAttach/10617_1870_1.pdf

8. ThingSpeak

   https://thingspeak.com/

9. MicrosoftFlow

   https://emea.flow.microsoft.com/es-es/

10. Internet of Food and Farm

    https://www.iof2020.eu/latest/news/2018/08/could-iot-help-farmers-in-this-dry-period

11. Riego automatizado de plantas

    https://ecocosas.com/agroecologia/riego-automatico-simple-escalable-raspberry-pi/

12. Estación meteorológica

    https://riunet.upv.es/bitstream/handle/10251/74585/MOYA%20-%20DESARROLLO%20DE%20UNA%20ESTACI%C3%93N%20METEOROL%C3%93GICA%20USANDO%20UNA%20RASBERRI%20PI.pdf?sequence=4&isAllowed=y

13. Control del pH

    https://riunet.upv.es/bitstream/handle/10251/91764/S%c3%81NCHEZ%20-%20Sistema%20port%c3%a1til%20aut%c3%b3nomo%20para%20la%20adquisici%c3%b3n%20de%20datos%20conectado%20a%20la%20nube.pdf?sequence=1&isAllowed=y

14. Obtención de datos de humedad y temperatura

    https://irjet.net/archives/V5/i1/IRJET-V5I1307.pdf

15. Diffie-Hellman – Kurose 6th Edition [8.2.2 Public Key Encryption]

https://www.bau.edu.jo/UserPortal/UserProfile/PostsAttach/10617_1870_1.pdf

16. DoS - Kurose 6th Edition [1.6 Networks Under Attack]

    https://www.bau.edu.jo/UserPortal/UserProfile/PostsAttach/10617_1870_1.pdf

17. SSH

    https://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-ssh.html

18. Precio RaspberryPI

    https://www.amazon.es/dp/B01CI58722/ref=asc_df_B01CI5872255090219/?tag=googshopes-21&creative=24514&creativeASIN=B01CI58722&linkCode=df0&hvdev=c&hvnetw=g&hvqmt=

19. Salario bruto anual ingeniero junior

    https://www.xataka.com/tecnologiazen/la-realidad-del-perfil-de-informatico-junior-en-espana-segun-los-informes

20. SensorML

    http://docs.opengeospatial.org/bp/17-011r2/17-011r2.html

21. REST

    https://www.w3.org/2001/sw/wiki/REST

22. SQLite

    https://www.sqlite.org/index.html

23. Texas Instruments CC3220

    http://www.ti.com/lit/ds/symlink/cc3220.pdf

24. RabbitMQ

    https://www.rabbitmq.com/