TRABAJO FIN DE GRADO

UCAM
UNIVERSIDAD CATÓLICA
DE MURCIA

ESCUELA UNIVERSITARIA POLITECNICA

Departamento de Ciencias Politécnicas
Grado en Ingeniería Informática

Sistema de monitorización de cultivos

Autor: D. Sebastián Martínez Pérez

Directora: Dra. Dña. Raquel Martínez España

Murcia, mayo de 2020

TRABAJO FIN DE GRADO

UCAM
UNIVERSIDAD CATÓLICA
DE MURCIA

ESCUELA UNIVERSITARIA POLITECNICA

Departamento de Ciencias Politécnicas
Grado en Ingeniería Informática

Sistema de monitorización de cultivos

Autor: D. Sebastián Martínez Pérez

Directora: Dra. Dña. Raquel Martínez España

Murcia, mayo de 2020

Agradecimientos

En primer lugar, agradecer a mis padres por su sacrificio, por haberme permitido estudiar y llegar hasta donde estoy hoy. A mi hermana por tener la paciencia que tiene conmigo y darme cariño.

A Dra. Dña. Raquel Martínez España, por permitirme desarrollar este Trabajo Fin de Grado.

A D. Francisco Carbonell Ferrando, por los conocimientos que adquirí cuando fui contratado en mi primer trabajo en Primaflor, me dio la oportunidad de adquirir muchos conocimientos del funcionamiento de una empresa agrícola.

A D. Víctor Cervero Polo por los años compartidos en S.A.T. 9989 Peregrín, años muy duros y en los que me ensañaron más conceptos, para mejorar esta empresa, aplicando la tecnología. Siendo pioneros y consiguiendo unos resultados extraordinarios.

A Dña. Antonia Martínez Guevara, por los conocimientos más técnicos de cultivos para comenzar este Trabajo Fin de Grado.

También a Yolanda por aguantarme y tener paciencia durante todo este tiempo.

Por último, a mi otra familia, que siempre me han ayudado Manolo y Brígida, que con el confinamiento los echo de menos.

Índice de ilustraciones

Índice de tablas

1. Resumen

El uso correcto de las tecnologías de la información permite a las empresas obtener una mejora de beneficios, así como optimización de recursos. Además de mejorar los sistemas de producción.

El desarrollo de este trabajo fin de grado se enfoca en el desarrollo de un sistema de monitorización de cultivos que mediante el uso de la tecnología IoT pueda capturar datos, además del uso del Cloud Computing que permita almacenar los datos en la nube. Así como un software cliente que pueda visualizar los datos capturados por los dispositivos IoT.

Se ha utilizado una metodología ágil para el desarrollo de las historias de usuario ya que permite más flexibilidad, adaptando la solución según las necesidades, con entregas periódicas para ir comprobando el producto.

Palabras clave: IoT, Cloud Computing, Cultivos.

2. Abstract

The correct use of information technologies allows companies to obtain improved profits, as well as optimization of resources. In addition to improving production systems.

The development of this end-of-degree project focuses on the development of a crop monitoring system that using IoT technology can capture data, in addition to the use of Cloud Computing that allows data to be stored in the cloud. As well as a client software that can visualize the data captured by the IoT devices.

An agile methodology has been used for the development of user stories since it allows more flexibility, adapting the solution according to needs, with periodic deliveries to check the product.

Keywords: IoT, Cloud Computing, Crops.

3. Introducción

3.1. Motivación

Hoy en día el correcto uso de las tecnologías de la información (TI) permite a las empresas obtener mejora de beneficios, optimización de riesgos y recursos, y sobre todo para las organizaciones agrarias que, aunque han mejorado sus sistemas de producción, no están utilizando la tecnología para mejorar sus beneficios y productos para sus clientes. Principalmente la mejora de recursos como es el agua que es un recurso escaso y de elevado coste en el sureste español. En la agricultura las condiciones meteorológicas pueden afectar al desarrollo de las plantas, pero estas no pueden ser controladas actualmente.

Tras más de 20 años trabajando para empresas agrícolas hortícolas en el desarrollo de aplicaciones para el control de trazabilidad, aplicaciones fitosanitarias, control de plagas, control de costos y planificación de los cultivos, he llegado a la conclusión de que los parámetros relacionados con las condiciones de cultivo no se están analizando con los medios adecuados. El análisis de esta información permitiría a las empresas agrícolas mejorar sus planificaciones y de esta manera optimizar sus recursos y ser más competitivas en los mercados.

La principal motivación del desarrollo del presente trabajo fin de grado (TFG) es ofrecer a las empresas agrícolas un sistema de medida de variables continúa relacionadas con las condiciones de cultivo que les permita tomar las mejores decisiones en las diferentes etapas.

El objetivo principal de este TFG es dotar al personal técnico de las empresas agrícolas de información sobre las condiciones de los cultivos in-situ. Además de poder obtener datos históricos para su estudio posterior por parte de dichos técnicos para mejorar la toma de decisiones en planificaciones futuras.

3.2. Definición

Este TFG contempla el análisis, diseño, desarrollo y despliegue de una solución hardware y software, para la toma de decisiones agronómicas, monitorizando variables ambientales durante el ciclo de cultivo.

Se pretende fabricar un dispositivo hardware utilizando componentes hardware Open Source, que permite medir mediante sensores parámetros como la temperatura y humedad, tanto del suelo como del ambiente, la presión atmosférica y luminosidad. Además, de conocer la posición de dicho dispositivo en la zona de cultivo a monitorizar mediante un sistema de geolocalización.

En la parte de comunicaciones se tiene que implementar una solución que permita capturar los datos enviados por dichos dispositivos a un servidor Back-End para su almacenamiento.

En la parte de software se pretende desarrollar un servidor Back-End para la recogida de datos mediante una capa API REST para la captura de los datos proporcionados por los dispositivos que los almacenará en una base de datos.

Además, se desarrollará una solución Front-End que consiste en una aplicación web y una aplicación móvil que permita consultar los datos almacenados, obteniendo dichos datos a través del servidor Back-End.

La aplicación cliente debe de proporcionar datos para conocer el estado de cultivo en su situación actual, para ver si se cumplen las condiciones adecuadas. Además de permitir consultar los datos históricos para analizar y poder tomar decisiones en futuras planificaciones de los cultivos y así poder optimizar la producción.

3.3. Objetivos propuestos

Para este TFG se pretenden alcanzar los siguientes objetivos:

- OG-01: Desarrollar un hardware (que denominaremos en adelante Gadget) para poder medir unas magnitudes, como son la humedad, temperatura y luminosidad. Para alcanzar este objetivo general se tiene que desarrollar los siguientes objetivos específicos:
    - OE-01: Estudiar los diferentes sistemas Open Source de comunicación que mejor se pueden adaptar a la solución.
    - OE-02: Estudiar los diferentes dispositivos hardware a utilizar y la cantidad de entradas analógicas, digitales y buses de comunicación que permite.
    - OE-03: Estudio de los diferentes sensores hardware que permitan medir las magnitudes. Medición analógica, digital y sistemas de comunicación entre la placa principal y los sensores.
    - OE-04: Herramientas de desarrollo a utilizar para el desarrollo del software necesario para el hardware específico, como realizar la lectura de los sensores y chip para capturar las medidas.
    - OE-05: Crear el dispositivo que permita la lectura de los parámetros indicados y envío de los datos al Gateway.
    - OE-06: Estudiar las diferentes formas de obtener el posicionamiento de los Gadget.
    - OE-07: Estudio de tipos de baterías a utilizar por el hardware y la durabilidad.
    - OE-08: Probar la estabilidad del dispositivo.
- OG-02: Implementar un Gateway que permita la recogida de datos de los Gadget y enviar los datos a un servidor Back-End para ser tratados. Para alcanzar este objetivo general se tiene que desarrollar los siguientes objetivos específicos:

    - OE-09: Estudiar los diferentes tipos de hardware para realizar la comunicación entre los Gadget y el servidor Back-End para la captura de datos.
    - OE-10: Estudiar el tipo de solución o plataforma a usar para el desarrollo de envío de datos al servidor Back-End.
    - OE-11: Optimizar el envío de datos al servidor Back-End dependiendo de las limitaciones del hardware utilizado.
- OG-03: Implementar un servidor Back-End para la recogida de datos de los Gadget. Para alcanzar este objetivo general se tiene que desarrollar los siguientes objetivos específicos:
    - OE-12: Implementar el servidor Back-End mediante una capa API REST para la recogida de datos de los Gateways y almacenar los datos en un servidor de base de datos.
    - OE-13: Guardar los datos en la base de datos para su correcta lectura, unidades de medida, tipo de sensor, etc..
    - OE-14: Desarrollar la parte del servidor Back-End para implementar una capa API REST para la consulta de datos por parte de las aplicaciones Front-End a la base de datos.
    - OE-15: Implementar servidor Back-End en Cloud Computing.
- OG-04: Generar una aplicación Front-End para poder consultar los datos capturados por los Gadget, mediante un portal web y aplicación móvil. Para alcanzar este objetivo general se tiene que desarrollar los siguientes objetivos específicos:
    - OE-16: Estudiar el tipo de herramientas a utilizar para el desarrollo del portal web y de la aplicación móvil.
    - OE-17: Visualizar los datos capturados por los dispositivos.
    - OE-18: Implementar la visualización de localización de los Gadget.
    - OE-19: Implementar servidor web en Cloud Computing.

4. Estudio de mercado

En este apartado se analizan las herramientas que están utilizando empresas agrícolas para realizar las mediciones de las condiciones del cultivo, además, he realizado una búsqueda de herramientas similares a mi TFG. El objetivo es tener una comparativa con las ventajas y desventajas de cada una de estas.

Las empresas agrícolas consultadas son, Melón del Mediterráneo, S.L. y S.A.T. 9989 Peregrín para ver el tipo de herramientas que están utilizando en la monitorización de cultivos.

La empresa Melón del Mediterráneo, S.L. es la que más apuesta por el uso de tecnológicas porque considera que es muy importante para la reducción de costes en la producción, poniendo a mi disposición información para la realización de este TFG.

4.1. Conceptos relevantes del dominio de aplicación

En este apartado se definen algunos conceptos que forma parte del sistema de monitorización de los cultivos para el desarrollo del TFG.

4.1.1. Back-End

El Back-End es la capa de acceso a datos de un software o cualquier dispositivo, que no es directamente accesible por los usuarios, además contiene la lógica de la aplicación que maneja dichos datos. El Back-End también accede al servidor, que es una aplicación especializada que entiende la forma como el navegador solicita cosas (maldeadora, 2017).

4.1.2. Front-End

El Front-End es la parte de un programa o dispositivo a la que un usuario puede acceder directamente. Son todas las tecnologías de diseño y desarrollo

web que corren en el navegador y que se encargan de la interactividad con los usuarios (maldeadora, 2017).

4.1.3. El internet de las cosas

El internet de las cosas (IoT, por sus siglas en inglés) es un sistema de dispositivos de computación interrelacionados, máquinas mecánicas y digitales, objetos, animales o personas que tienen identificadores únicos y la capacidad de transferir datos a través de una red, sin requerir de interacciones humano a humano o humano a computadora (techtarget.com, 2017).

4.1.4. Gateway

El Gateway (puerta de enlace), es un dispositivo que permite interconectar redes con protocolos y arquitecturas diferentes a todos los niveles de comunicación. Su propósito es traducir la información del protocolo utilizado en una red al protocolo usado en la red de destino (ecured.cu, 2019).

4.1.5. JavaScript Object Notation

JavaScript Object Notation (JSON) es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript, Standard ECMA-262. JSON es un formato de texto que es completamente independiente del lenguaje, pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos (json.org, 2019).

4.1.6. Representational State Transfer

Representational State Transfer (REST) es una interfaz para conectar varios sistemas basados en el protocolo HTTP y nos sirve para obtener y generar datos y operaciones, devolviendo esos datos en formatos muy específicos, como

XML y JSON. REST se apoya en HTTP, los verbos que utiliza son exactamente los mismos, con ellos se puede hacer GET, POST, PUT y DELETE (Moncayo, 2018).

4.1.7. Application Programming Interfaces

Application Programming Interfaces (API) es un conjunto de reglas (código) y especificaciones que las aplicaciones pueden seguir para comunicarse entre ellas: sirviendo de interfaz entre programas diferentes de la misma manera en que la interfaz de usuario facilita la interacción humano-software (ticbeat.com, 2017).

4.1.8. Cloud computing

Cloud computing es un conjunto de principios y enfoques que permite proporcionar infraestructura informática, servicios, plataformas y aplicaciones (que provienen de la nube) a los usuarios, según las soliciten y a través de una red. Las nubes son grupos de recursos virtuales (por ejemplo, el potencial de procesamiento en bruto, el almacenamiento o las aplicaciones basadas en la nube) que se coordinan mediante un software de gestión y automatización, para que los usuarios puedan acceder a ellos según lo soliciten, a través de los portales de autoservicio a los que dan soporte el escalado automático y la asignación dinámica de recursos. El cloud computing permite que los departamentos de TI no pierdan tiempo ampliando las implementaciones personalizadas al darle a las unidades empresariales el poder para solicitar e implementar sus propios recursos (redhat, 2019).

4.1.9. Cultivo

El cultivo es el conjunto de operaciones destinadas a la obtención de productos vegetales mediante la siembra de estos. Las formas de cultivo según la cantidad de agua aportada son el cultivo de secano, en el que no se aporta riego desarrollándose exclusivamente con agua que se puede obtener por sus propios medios, lluvia o aguas extraída del suelo y el cultivo de regadío en el que

se aportan el agua necesaria para el desarrollo del cultivo de forma artificial median el riego.

El desarrollo y crecimiento de los cultivos depende de factores como la nutrición, riego o enfermedades que pueden ser manejados para la mejora de la producción agrícola y otros como los climáticos que solo pueden ser manejados parcialmente.

Los principales factores ambientales que determinan el crecimiento y desarrollo de los cultivos son:

- Temperatura: afecta la tasa de desarrollo de la planta. Todos los procesos fisiológicos de la planta ocurren más rápidamente a temperatura optima. Un buen manejo del cultivo puede contrarrestar los efectos negativos de las altas y bajas temperaturas. La temperatura también influye en la absorción de nutrientes, desarrollo microbiano y agua.
- Humedad: el agua es un componente principal de la fotosíntesis porque ayuda a los tejidos a permanecer firmes y a mover nutrientes a través de la planta. Los cultivos crecen más rápidamente siempre que tengan agua suficiente.
- Radiación solar: el desarrollo del cultivo está influenciado por la cantidad de radiación solar que puede interceptar y usar durante su desarrollo, ya que esta determina la actividad fotosintética.

4.2. Relación con proyectos con la misma funcionalidad

Actualmente las empresas agrícolas consultadas no disponen de un sistema de monitorización automatizado de los cultivos, que proporcione los datos necesarios para el seguimiento y optimización del cultivo.

Las empresas agrícolas consultadas obtienen estos datos de estaciones meteorológicas estatales o contratan algún servicio privado como ofrece la empresa WIDHOC (WIDHOC, 2019) que proporcionan datos a nivel local,

instalando un dispositivo, como una pequeña estación que permite medir algunas variables climáticas. También existe la posibilidad de adquirir una estación meteorológica propia, pero el coste es muy elevado y proporciona información extra no relevante.

También están utilizando dispositivos de captura de datos en el campo, pero los datos tienen que ser recogidos manualmente. Hay varios fabricantes como, por ejemplo, PCE Ibérica (PCE Instruments, 2019) y Hanna instruments (HANNA Instruments, 2019) que proporcionan estos sensores.

Entre los productos disponibles en el mercado que abordan este problema están los ofrecidos por Sensoterra (Sensoterra, 2019), que se centran exclusivamente en la medida de la humedad del suelo, para la mejora de regadío. Ofrece la instalación de varios dispositivos que permiten obtener datos de diferentes zonas del mismo cultivo. En la zona del norte de la provincia de Almería, la orografía es muy variable y una misma explotación puede tener diferentes condiciones, por consiguiente, es determinante la instalación de un número suficiente de sensores, para monitorizar el cultivo correctamente.

Tras realizar una búsqueda de las alternativas disponibles en el mercado, el producto más parecido al propuesto en este TFG sería el ofertado por la empresa LESS industries (LESS Industries, 2019) con diferentes sedes en Argentina, Australia y Chile. Esta empresa proporciona sensores para la medición de Temperatura, humedad y salinidad de suelo, temperatura y humedad ambiental, permitiendo consultar los datos online.

Además, existen otras soluciones que pueden cubrir partes de este proyecto, como las que ofrece The Things Network. Este framework proporciona un conjunto de herramientas abiertas y una red global abierta para construir una aplicación IoT con la máxima seguridad y lista para escalar. Construyendo una red segura y colaborativa de Internet de las cosas que se extiende por muchos países de todo el mundo (The Things Network, 2019).

Mientas ThingsBoard es otra solución que permite la conectividad del dispositivo a través de protocolos de IoT estándar. Permite aprovisionar, supervisar y controlar los dispositivos IoT de forma segura utilizando API enriquecidas del lado del servidor. Además, permite recompilar y almacenar datos de telemetría y visualizarlos con widgets integrados o personalizados y paneles flexibles (ThingsBoard, 2017).

4.3. Estudio de viabilidad

El estudio de la viabilidad permitirá evaluar la eficacia del producto final del proyecto basándonos en una serie de información empírica o visible. El resultado de este estudio va a permitir definir si el proyecto es viable o necesita modificaciones para cumplir con los objetivos propuestos.

4.3.1. Alcance del proyecto

El principal objetivo del sistema es la captura de datos que se consideran actualmente relevantes para poder tomar decisiones agronómicas que permita desarrollar el cultivo de una manera óptima y eficiente, además de utilizar estos datos para una mejor planificación.

Los datos que queremos monitorizar en cultivo seria:

- Humedad del suelo.
- Temperatura del suelo.
- Humedad ambiental.
- Temperatura ambiental.
- Presión atmosférica.
- Rayos UVI.
- Luminosidad lúmenes.
- Geolocalización.

Para la captura de datos utilizaremos un dispositivo IoT que dispondrá de diferentes sensores analógicos y digitales que recogerá los datos a monitorizar.

Este dispositivo será colocado en la finca con comunicación a una Gateway y enviará paquetes de datos capturados al Gateway para su procesamiento. Este dispositivo lo denominamos Gadget.

El Gateway recogerá los datos de los diferentes Gadget que estarán posicionados alrededor de éste y procesará los paquetes recibidos. Para ello el Gateway encapsulará en paquetes JSON que enviada a través de internet al servidor Back-End que estará alojado en la nube.

El servidor Back-End implementa una Web API REST que permite la comunicación entre el Gateway y éste, permitiendo recibir los paquetes JSON. Una vez que los paquetes JSON son recibidos, este servidor procederá a traducirlos para guardarlos en la base de datos, identificando el dispositivo Gadget que ha enviado los datos y los datos de los diferentes sensores que ha capturado.

Los usuarios podrán consultar los datos almacenados mediante dos aplicaciones Front-End, que consistirá en una aplicación Web y una aplicación para dispositivos móviles, compatible para iOS y Android. Estas permitirán consultar los datos enviados por los Gadget. La comunicación de las aplicaciones Front-End se realizará a través del servidor Back-End usando la Web API REST para acceder a los datos almacenados en el servidor, permitiendo la aplicación Web descargar los datos históricos en un archivo Excel para que los técnicos puedan analizar los datos.

Gracias a esta herramienta los técnicos podrán obtener los datos para procesarlos en sus posteriores estudios de planificación además de poder mejorar la toma de decisiones en el cultivo según su situación.

4.3.2. Estudio de la situación actual

En la actualidad la empresa Melón del Mediterráneo, S.L. está utilizando una sonda para la lectura de la humedad del suelo para controlar la cantidad de agua que hay que utilizar en el cultivo, en esta zona el agua es un bien escaso

y de elevado coste, por lo que su uso óptimo permite mejorar el rendimiento económico.

Esta empresa coloca sondas distribuidas por la finca en función de su superficie. A medida que se van colocando las sondas son asociadas a la parcela y sector de riego, además de marcar la profundidad de instalación. Esta información la van registrando en papel, que luego es escrita en un archivo en formato Excel.

Un operario de la finca cada cierto tiempo, va tomando la lectura de los sensores manualmente mediante un dispositivo que conecta a la sonda y anota los resultados en papel. Mas tarde entrega los datos al técnico para que este los registre en formato Excel como se puede ver en la ilustración 1 y de esta manera determinar las necesidades de riego del cultivo.

	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP	AQ	AR	AS	AT	AU	AV	AW
1	D	SONDAS				D				1 2	Variar regula + - potenciar abajo		D	SONDAS						D				SONDAS			D		
2	24-feb	27-feb	28-feb	01-mar	02-mar	03-mar	04-mar	05-mar	06-mar	07-mar	08-mar	09-mar	10-mar	11-mar	12-mar	13-mar	14-mar	15-mar	16-mar	17-mar	18-mar	19-mar	20-mar	21-mar	22-mar	23-mar	24-mar	25-mar	26-mar
3					.5		1,75				0,75	x	x	x	x	0,75	x	x	x	x	0,75	x	x	x	x	x	0,75	x	x
4				3	12		5	0	0	0	3	0	0	0	2	3	0	0	0	2	3	0	0	0	0	3	3	0	0
5																													
6				13	13		12	7	8	10	12	8	10	11	11	12	10	11	11	12	12	10	11	11	12	12	12	11	11
7																													
8				12	12		11	2	1	4	6	2	2	3	6	9	1	4	5	6	7	3	3	4	5	8	9	4	4
9	24-feb	27-feb	28-feb	01-mar	02-mar	03-mar	04-mar	05-mar	06-mar	07-mar	08-mar	09-mar	10-mar	11-mar	12-mar	13-mar	14-mar	15-mar	16-mar	17-mar	18-mar	19-mar	20-mar	21-mar	22-mar	23-mar	24-mar	25-mar	26-mar
10			0,50		1,00		1,50	x	x	0,50	x	0,50	x	0,50	x	0,50	x	0,50	0,50	x	0,50	x	x	0,50	x	x	x	x	0,50
11				22	23		6	0	1	10	6	10	6	8	5	8	6	10	5	3	6	4	8	11	7	9	11	13	13
12				17	22		8	7	4	7	2	5	2	4	2	4	3	7	4	2	9	3	6	11	5	6	10	12	13
13				30	31		15	1	8	12	9	13	6	11	6	11	7	11	6	2	10	3	8	12	7	9	12	13	15
14				31	32		6	6	11	14	8	13	6	12	6	12	13	13	7	8	12	7	11	14	9	10	12	15	18
15	24-feb	27-feb	28-feb	01-mar	02-mar	03-mar	04-mar	05-mar	06-mar	07-mar	08-mar	09-mar	10-mar	11-mar	12-mar	13-mar	14-mar	15-mar	16-mar	17-mar	18-mar	19-mar	20-mar	21-mar	22-mar	23-mar	24-mar	25-mar	26-mar
16		1,00			0,50		0,50	x	0,50	x	0,50	x	0,50	x	0,50	x	0,50	x	0,50	x	0,50	x	0,50	x	0,50	x	x	0,75	x

Ilustración 1 - Ejemplo de archivo de captura de datos

En el caso de S.A.T. 9989 Peregrín está utilizando datos de las estaciones meteorológicas estatales para conocer los datos necesarios para el desarrollo de sus cultivos, como se puede ver en la ilustración 2.

Ilustración 2 - Datos climáticos. Imagen obtenida de eportal.mapama.gob.es

Actualmente en pruebas con la aplicación de WIDHOC (WIDHOC, 2019), con tres sensores puestos en tres fincas, como se puede ver en la ilustración 3.

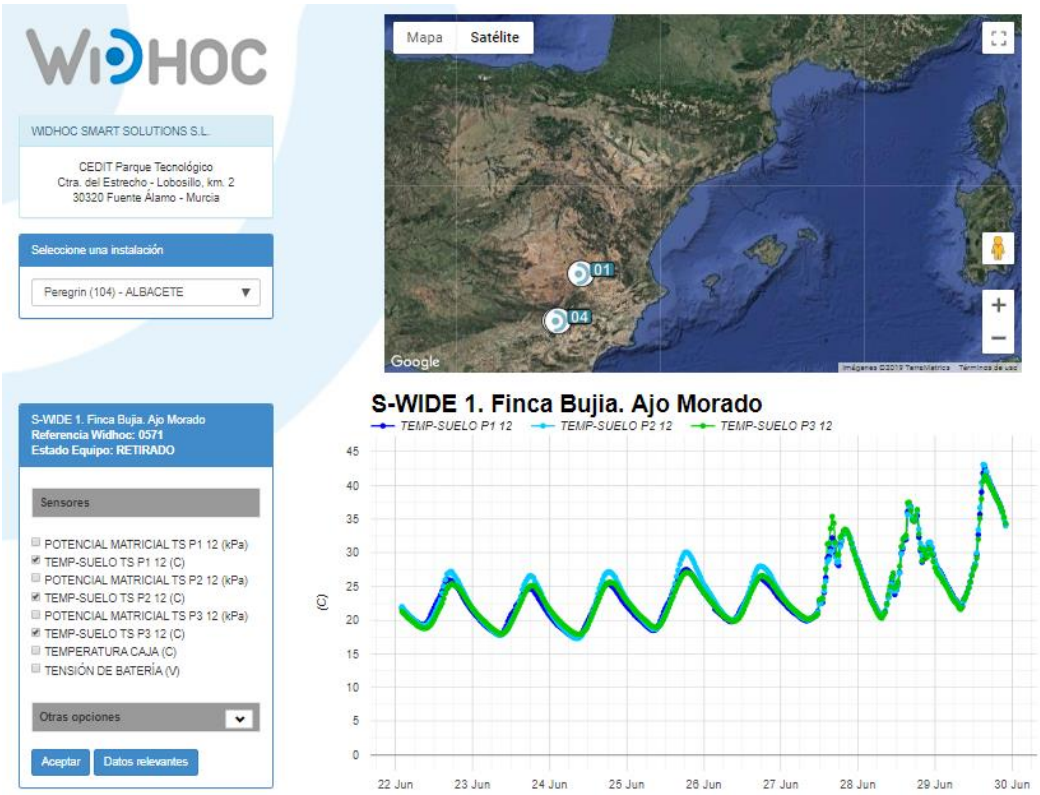

Ilustración 3 - Ejemplo de la aplicación WIDHOC. Imagen capturada de http://fcloud1.upct.es

Como se ha mencionado anteriormente existen varias soluciones para la medición de los factores que afectan al cultivo, ya sea mediante sistemas manuales o sistema automatizados. Las opciones que se están utilizando en

estos momentos no ofrecen una solución completa y automatizada como la que se propone en este proyecto.

El desarrollo de este TFG es una alternativa que permite al personal técnico la consulta de los datos in-situ de la situación de los cultivos en las diferentes zonas donde se haya instalado los Gadget, consultándolos mediante un dispositivo móvil o mediante un navegador web.

4.3.3. Estudio y valoración de las alternativas de solución

En función de las necesidades que se han mencionado y de la situación actual de las empresas agrícolas consultadas, la toma de datos mediante sensores manuales conlleva un elevado coste de personal para registrarlos, se plantea una mejora en el proceso de automatización respecto a la toma de datos manual y se elimina el factor humano, mediante el desarrollo de este TFG.

En comparación a las soluciones utilizadas por las empresas agrícolas consultadas y otras soluciones encontradas he realizado la tabla 1, donde se puede ver una comparativa de las principales características de las alternativas de la solución.

	Sensores manuales	WIDHOC	Sensoterra	Less Industries
Captura de datos	Manual	Automática	Automática	Automática
Temperatura ambiente	Depende del sensor	No	No	Sí
Humedad ambiente	Depende del sensor	No	No	Sí
Temperatura suelo	Depende del sensor	Sí (Varias profundidades)	No	Sí
Humedad suelo	Depende del sensor	Sí (Varias profundidades)	Sí	Sí
Luminosidad	Depende del sensor	No	No	No
Presión atmosférica	Depende del sensor	No	No	No
Consulta de datos	Papel/Archivo	Web	Web/App	Web/App
Geolocalización	No	Sí	Sí	Sí

Tabla 1 - Comparativa de soluciones actuales

Como alternativa y la mejora a las soluciones utilizadas por las empresas agrícolas consultadas y a las necesidades de Melón del Mediterráneo, S.L. se ha optado por desarrollar este TFG que se adapte a los requerimientos por parte de los técnicos de cultivo.

La elección de las tecnologías se basa en la funcionalidad que se desea desarrollar, dependiendo del hardware IoT que se vaya a utilizar para el desarrollo del Gadget tendré que estudiar los siguientes puntos:

- Sistemas de comunicación a utilizar para envío de datos.
- Herramientas y tecnología para el desarrollo de la aplicación del Gadget.
- Uso de hardware chips TMP36 o hardware Open Source para los sensores que captura las diferentes magnitudes como: VEML6070, GY-BME280, BH1750FVI, etc..

Mientras para el desarrollo de servidor Back-End y Front-End la elección de las tecnologías se basa en la funcionalidad que se desea desarrollar, se podría emplear algunas de las siguientes opciones:

- Uso de cloud computing para alojamiento del servidor Back-End y alojamiento del servidor cliente web o uso de plataformas como The Things Network o ThingsBoard.
- Persistencia de datos mediante motores de base de datos, almacenados en cloud computing.
- Aplicación de dispositivos móviles que soporte iOS y Android.

4.3.4. Selección de la solución

Tras valorar las soluciones comerciales disponibles, se ha optado por seleccionar la solución analizando el proyecto en dos partes.

En primer lugar, en el desarrollo del Gadget no hay ninguna solución que cubra las necesidades de las empresas agrícolas consultadas, pues en la comparación de las soluciones falta la medición de algunos factores como la iluminación o que permita la personalización a algún otro factor que se considere relevante.

En segundo lugar, se considera que la mejor opción para las aplicaciones de Back-End y Front-End es desarrollar una solución a medida, pues nos permitirá responder a las necesidades particulares que puedan surgir ya que se pueden tratar y personalizar los datos.

Además, el desarrollo a medida va a permitir controlar el código durante todo el ciclo de vida del producto.

En la ilustración 4 podemos observar un esquema general de la infraestructura que presentará el TFG.

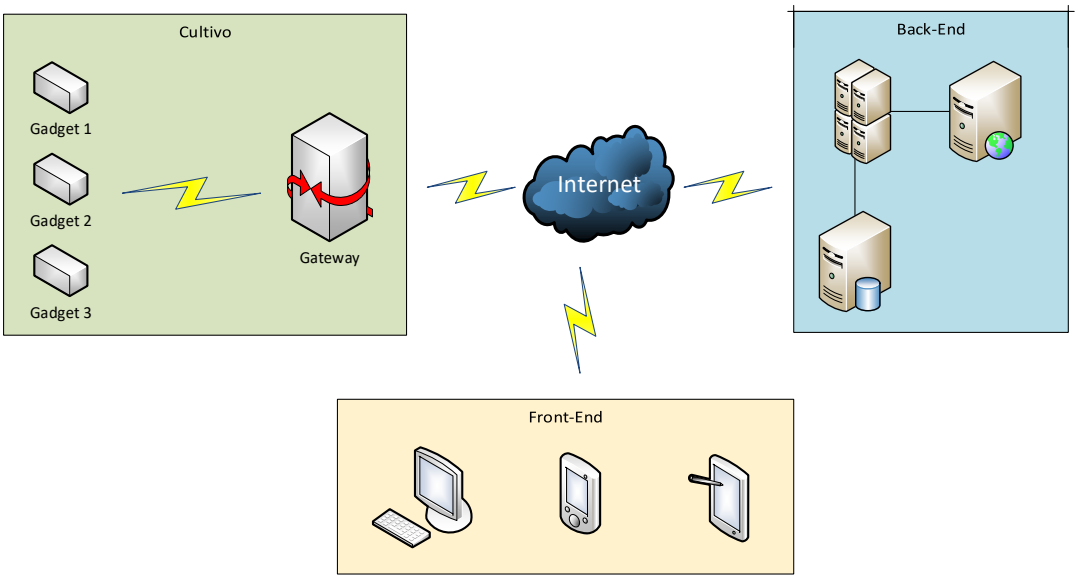

Ilustración 4 - Infraestructura TFG

4.3.5. Visión futura

La solución tiene que ser versátil, de manera que permita incorporar nuevos sensores necesarios para la captura de datos, así como, tras el registro continuado de los datos que obtendremos y el análisis de estos por técnicos especializados se podrá determinar que otros datos puedan ser más relevantes y adaptar el sistema para la mejora de este, cubriendo las necesidades demandadas por los potenciales clientes.

5. Metodologías usadas

Las metodologías de desarrollo del software nos permiten gestionar un proyecto ofreciéndonos un conjunto de procedimientos que nos ayudan a la planificación, actuación, definición de objetivos y revisión continua de los mismos. Permitiendo la consecución del objetivo final minimizando los riesgos en la medida de lo posible.

“Una metodología de software es un enfoque, una manera de interpretar la realidad o la disciplina en cuestión, que en este caso particular correspondería a la Ingeniería de Software. De hecho, la metodología destinada al desarrollo de software se considera como una estructura utilizada para planificar y controlar el procedimiento de creación de un sistema de información especializada” (Gomez, 2017).

Encontramos diferentes metodologías a la hora de desarrollar un proyecto, dependiendo del tipo de contexto en que se encuentre el proyecto seleccionaremos la metodología más apropiada.

Un proyecto se puede encontrar con unos de los siguientes contextos:

- Simple.
- Complicado.
- Complejo.
- Caótico.
- Desordenado.

5.1. Metodologías tradicionales

En los modelos de planificación de gestión de proyectos informáticos tradicionales lo que se pretende es descomponer el proyecto en fases como podrían ser la siguiente:

- Analizar

- Diseñar
- Desarrollar
- Probar
- Entregar

Este sería un ciclo de vida de proyecto **predictivo** o **en cascada** (Domínguez, 2017) y funciona bien para contextos simples, donde existen pocas modificaciones en el proyecto, los acontecimientos son muy previsibles y el trabajo tiene un alcance limitado.

El problema lo encontramos cuando se produce una modificación de los requisitos del proyecto, conlleva a tener que redefinir la planificación en cada una de las fases, lo que supone un elevado coste adicional.

Por ello surgieron metodologías alternativas que permiten flexibilizar la rigidez. A continuación, describimos a modo de ejemplo algunas de ellas:

- **Proceso relacional unificado** (RUP), se siguen creando fases que se pueden aplicar de forma interactiva, pudiendo solaparse y repetirse cíclicamente cada fase (Metodoss, 2019).
- **Desarrollo rápido de aplicaciones** (RAD), este método destaca por la participación del usuario, en la que se desarrollan prototipos para probar las técnicas y perfeccionar los requisitos (Kissflow, 2018).
- **Modelo en espiral**, es una mezcla entre el modelo en cascada y RAD, permitiendo reducir los factores de riesgo e incorporar nuevas funcionalidades en cada iteración (Asp Gems, 2019).

La metodología tradicional está muy centrada para el uso exclusivo de la documentación durante todo el ciclo y en cumplir con el plan de cada una de las fases establecidas del proyecto.

5.2. Metodologías ágiles

Para solucionar los problemas de las metodologías tradicionales, aparecen las metodologías ágiles permitiendo utilizar diferentes técnicas para resolver proyectos con contextos más complicados y complejos.

El uso de estas metodologías en el desarrollo del software es menos riguroso que las metodologías tradicionales, pues a la hora de desarrollar software es más interactivo e incremental. Se pretende dividir el software en partes más pequeñas y manejables para una entrega rápida al cliente, así el cliente puede comprobar las entregas y el equipo puede beneficiarse de la retroalimentación para mejorar el producto en cada ciclo.

Para promover el uso de las metodologías ágiles se crea el manifiesto por el desarrollo ágil del software:

“Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:

Individuos e interacciones sobre procesos y herramientas Software funcionando sobre documentación extensiva Colaboración con el cliente sobre negociación contractual Respuesta ante el cambio sobre seguir un plan.

Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda.” (agilemanifesto.org, 2001).

El manifiesto por el desarrollo ágil de software define doce principios que pueden ser consultados en el siguiente enlace web (agilemanifesto.org, 2019).

Toda metodología ágil se puede resumir en lo siguiente:

- Mejorar la satisfacción del cliente, pues este forma parte del proyecto viendo los avances en cada una de las etapas de desarrollo del proyecto.

- Se consigue motivar e implicar al equipo de desarrollo del proyecto, pues conocen el estado en cada momento y todos los integrantes son tenidos en cuenta.
- Se aceptan que los requisitos cambien para proporcionar venta competitiva al cliente.
- Ahorro del tiempo y de costes, pues se trata de manera mucho más eficaz sin olvidar el presupuesto acordado y los tiempos de entrega definidos.
- Se realizan entregas parciales que permite detectar errores lo antes posible y eliminar características innecesarias, porque lo que se consigue más eficiencia y mayor velocidad.
- Permite rentabilidad la inversión de manera más rápida.

Entre algunos ejemplos de estas metodologías tenemos las siguientes:

- **Extreme Programming** (XP) es exitosa porque enfatiza la satisfacción del cliente, porque se va entregando lo que va necesitando, en lugar de entregar un producto final a una fecha lejana. Permite adaptarse a los requisitos cambiantes de los clientes. Para esta metodología las pruebas son la base de la construcción, pues propone que los desarrolladores sean los que escriban las pruebas a medida que van construyendo el código y se vayan ejecutando lo más frecuente posible y automáticamente. Define una serie de roles y anti-roles que puede ser combinados en una misma persona, aunque hay algunos roles que no. Se realizan reuniones para la toma de decisiones y resolver los problemas que surjan en el desarrollo. La estructura de esta metodología seria la siguiente: Planificación, Diseño, Codificación y Pruebas. Para más información consultar el Anexo 10.3.1 (Extreme Programing, 2019).
- **SCRUM** es un marco dentro del cual las personas pueden abordar problemas de contexto complejo, al tiempo que entregan productos productivos y creativos del mayor valor posible. Esta metodología establece un método para trabajar en equipo a partir de Product Backlog del proyecto, siendo el Product Owner el que establece el orden de prioridad. El equipo desarrolla alguna de las funcionalidades en un Sprint que se comprometen a realizar. Tras la finalización del Sprint se obtiene

un incremento del producto utilizable que será revisado para su validación. Esta metodología se basa en los principios de: Autoorganización y colaboración, Priorización, Inspección y adaptación y Mantener un latido. También se definen unos roles que representan una responsabilidad en el proceso del desarrollo. Además, define los siguientes artefactos: Prouct Backlog, Sprint Backlog y Burndown Chart. Para más información consultar el Anexo 10.3.2 (scrum.org, 2019).

- **Kanban** nació para aplicarse a los procesos de fabricación y con el tiempo ha empezado a ser reconocido por las entidades empresariales de diferentes ámbitos. Permite realizar proyecto que tiene una naturaleza cambiante de los requisitos, permitiendo adaptarse a estas. En lugar de realizar un gran trabajo, este se descompone en trabajos intangibles que se visualizan para conocer en cada momento su progreso hasta la entrega al cliente. Presenta seis prácticas centrales que deben estar presentes que son: Visualizar el flujo de trabajo, Eliminar las interrupciones, Gestiona el flujo, Hacer las políticas explicitas, Circuitos de retroalimentación y Mejorar colaborando. Para más información consultar el Anexo 10.3.3 (kanbanize.com, 2019).
- **Pragmatic Programming** proporciona una serie de mejores prácticas de programación (Aguilera, 2017).
- **Feature Driven Development (FDD)** no cubre todo el proceso de desarrollo de un producto, sino que se centra en las fases de diseño y construcción (Garzas, 2012).
- **Dynamic System Development Method (DSDM)** propone un método de trabajo para el ciclo de vida de un proyecto en fases, donde las primeras se realizan una sola vez, mientas la últimas se realizan de forma iterativa e incremental (Aguile Bussines Consortium, 2019).
- **Scrumbam** combina las mejoras características de las metodologías SCRUM y Kanban, permitiendo moverse hacia un desarrollo ágil y a la mejora constate de sus procesos (Kanban tool, 2019).

5.3. Comparación de metodologías tradicionales y agiles

En la tabla 2 se puede observar la diferencia entre metodologías ágiles y tradicionales (López Gil, 2018).

	Ágiles	Tradicionales
Enfoque	Adaptación	Predictivo
Éxito de medición	Valor del negocio	Conformación de planificar
Tamaño del proyecto	Pequeño	Grande
Perspectiva para el cambio	Cambio y adaptabilidad	Cambio y sostenibilidad
Cultura	Liderazgo – Colaboración	Comandos de control
Documentación	Bajo	Pesado
Cliente	Parte del equipo	Interactúa mediante reuniones
Énfasis	Muchos	Limitado
Planificación por adelanto	Mínimo	Exhaustivo
Retorno de la inversión	A principios del proyecto	Fin del proyecto
Tamaño de equipo	Pequeños	Grandes

Tabla 2 - Diferencias entre metodologías ágiles y tradicionales

Al realizar el estudio de ambas metodologías, se ha optado por el uso de la metodología ágil, ya que en la parte de hardware la definimos en un contexto complejo y la de software en un contexto complicado.

Las principales ventajas del uso de la metodología ágil son que permiten el cambio y adaptación del proyecto. Además de la intervención del cliente como parte del equipo de desarrollo del proyecto, permitiendo ver resultados del proyecto con entregas parciales y así tener un feedback por parte del cliente.

Además, se ha realizado una comparación de algunas de las metodologías ágiles para seleccionar cual dé ellas se adapta mejor a este proyecto. En la tabla 3 se puede ver la comparación de estas.

	XP	Scrum	Kanban
Tipo de iteraciones	Iteraciones de plazo fijo	Iteraciones de plazo variable	Iteraciones plazo fijo o variable
Roles	Sí	Sí	No
Limitación de tareas	Limitación por iteración	Limitación por iteración	Limitación por estado
Incorporación de tareas	No es posible hasta finalizar el Srpint	No es posible hasta terminar la iteración	Es posible, en tanto exista capacidad
Seguimiento de tareas	Gráfico Burn-down	Velocity	Tablero Kanban
Estimación	Obligatoria	Obligatoria	Opcional

Tabla 3 - Tabla de comparación de metodologías agiles

5.4. Metodología elegida para la realización del TFG

Tras analizar algunas de las metodologías agiles que más se adapta a el desarrollo del TFG y teniendo en cuenta la limitación que supone que el desarrollo de este sea realizado por una única persona, en lugar de un equipo, a priori una buena opción sería basarnos en la metodología Kanban.

La metodología XP no es apropiada porque se fundamenta en una programación en parejas, además no es posible que una única persona adopte todos los roles por incompatibilidad. Aunque se considera útil el uso de las pruebas unitarias.

La metodología SCRUM, implica que el Product Owner y el Scrum Master sea la misma persona, lo cual no es compatible. No obstante, se considera útil el uso de Sprints a la hora de agrupar las tareas a realizar en un periodo de tiempo definido.

Por consiguiente, la metodología seleccionada para la realización de este TFG seria la llamada Scrumban.

Tomando como base la metodología Kanban se establecerá un tablero Kanban que nos permita visualizar en todo momento el flujo de trabajo, pudiendo identificar los cuellos de botella que se produzcan.

Además, se limitará y se ajustará el WIP para solo tener las tareas abiertas que se puede asumir y de esta manera medir y optimizar los tiempos de entrega de cada ciclo.

Además, se usarán iteraciones (basado en la metodología SCRUM) que ayuden a controlar las fechas del proyecto y sus funcionalidades, se tendrán ordenados los Product Backlogs con sus correspondientes Sprint Backlogs con las distintas historias de usuario para las distintas funcionalidades del proyecto.

Para controlar el desarrollo del proyecto se usará la herramienta Microsoft Azure DevOps, que permite tener un tablero Kanban, registrar los Backlog, registrar sus tareas y asignarlas a los Sprint, como se puede ver en la ilustración 5.

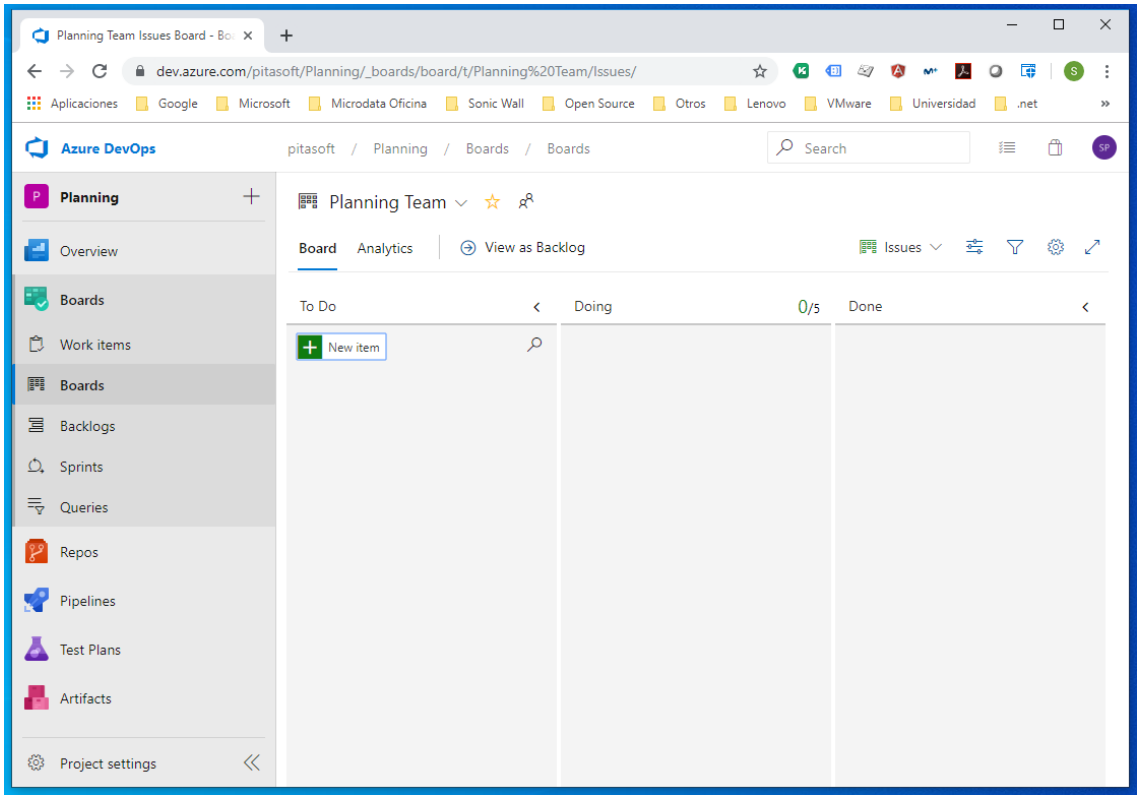

Ilustración 5 - Tablero Kanban - Imagen obtenida de Microsoft Dev Ops https://dev.azure.com

6. Tecnologías y herramientas utilizadas en el proyecto

En los siguientes apartados se enumeran y explican brevemente las tecnologías, leguajes de programación y herramientas para el desarrollo de este TFG.

6.1. Tecnologías

En este apartado hago una pequeña descripción de las tecnologías que se utilizan para desarrollar del TFG.

6.1.1. MVC

Modelo-vista-controlador (MVC) es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado, define componentes para la representación de la información y la interacción del usuario como se puede ver en la ilustración 6. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento (Microsoft, 2019).

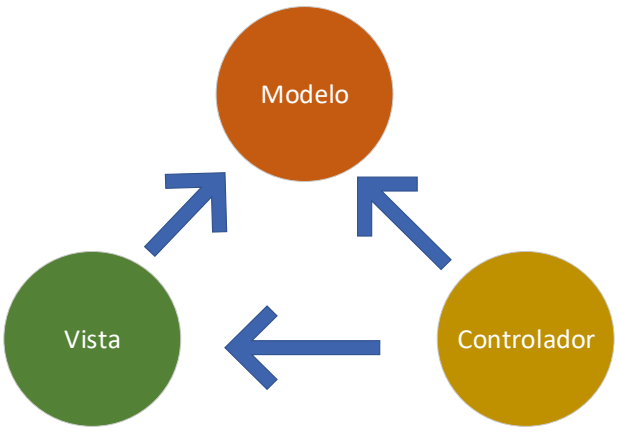

Ilustración 6- Patrón MVC

Esta tecnología se va a utilizar en el servidor Back-End con el uso de ASP.Net Core Web Api.

6.1.2. MVVM

El patrón modelo–vista–modelo de vista (model–view–viewmodel, MVVM) es un patrón de arquitectura de software que se caracteriza por tratar de desacoplar lo máximo posible la interfaz de usuario de la lógica de la aplicación, como se puede ver en la ilustración 7 (Microsoft, 2017).

Ilustración 7 - Patrón MVVM

Esta tecnología se utilizará en el desarrollo de la aplicación móvil con Xamarin.

6.1.1. Patrón de inyección de dependencias

El patrón de inyección de dependencias consiste en hacer que nuestras piezas de software sean independientes comunicándose únicamente a través de una interfaz. Así en nuestro código fuente se implementarán las interfaces y se elimina la instanciación de objetos mediante la instrucción new o la necesidad de un modo de configuración que indique que clases se instanciarán en el caso de solicitarlo (Jorge Rubira, 2011).

En este proyecto se va a utilizar la inyección de dependencias para la definición del API en el servidor Back-End (Microsoft, 2019).

6.1.2. Entity Framework Core

Entity Framework (EF) Core permite el acceso a datos mediante un modelo, donde el modelo está formado por clases de entidad que permiten consultar y guardar los datos en una base de datos (Microsoft, 2016). EF Core se utilizará en este proyecto para la persistencia de datos en el servidor Back-End.

6.1.3. Syncfusion

Syncfusion son librerías que proporciona más de 1.100 componentes de interfaz de usuario para todos los marcos .NET, así como múltiples temas, máscaras y opciones de personalización para el desarrollo web, multiplataforma y de escritorio. Esta tecnología se utilizará para el desarrollo de las aplicaciones Front-End (Syncfusion Inc., 2019).

6.1.4. Google Maps Platform

Google Maps Platform permite crear experiencias dinámicas y personalizadas para acercar el mundo real a los usuarios a través de mapas estáticos y dinámicos (Google Cloud, 2019). Esta tecnología se utilizará para la visualización de los mapas en aplicaciones Front-End.

6.1.5. LoRa

LoRa es una tecnología inalámbrica, al igual que WIFI, Bluetooth, LTE, SigFox o Zigbee, que emplea un tipo de modulación en radiofrecuencia patentado por Semtech. Esta tecnología de modulación se denomina Chirp Spread Spectrum o CSS y empleada en comunicaciones militares y espaciales desde hace décadas.

Actualmente la tecnología LoRa está administrada por la LoRa Alliance, quien certifica a los fabricantes de hardware que desee trabajar con ella (LoRa Alliance, 2019).

6.1.6. LoRaWAN

LoRaWAN es el protocolo de red que usa la tecnología LoRa, y es un estándar de red que apunta a requerimientos característicos de IoT, tales como conexiones bidireccionales seguras, bajo consumo de energía, largo alcance de comunicación, bajas velocidades de datos, baja frecuencia de transmisión, movilidad y servicios de localización. Permite la interconexión entre objetos

inteligentes sin la necesidad de instalaciones locales complejas y además otorga amplia libertad de uso al usuario final.

La arquitectura de red típica es una red en estrella de estrellas, de forma que la primera estrella está formada por los dispositivos finales y las puertas de enlace, y la segunda estrella está formada por las puertas de enlace y un servidor de red central. Las puertas de enlace son un puente transparente entre los dispositivos finales y el servidor central, como se puede ver en la ilustración 8 (MultiTech Developer Resources, 2019).

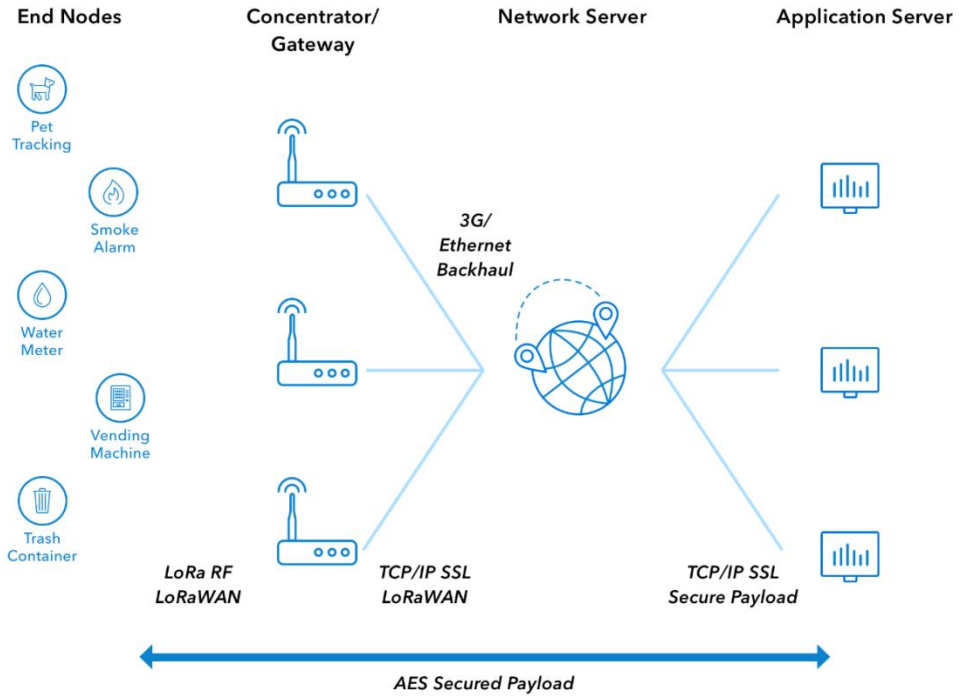

Ilustración 8 - Ejemplo de red LoRaWAN - Imagen obtenida de https://www.thethingsnetwork.org

Esta tecnología se utilizará en el desarrollo del Gadget y Gateway para realizar las comunicaciones entre ellos y el Back-End.

6.1.1. Hardware de Fuentes Abiertas.

El hardware de Fuentes abiertas (Hardware Open Source, OSHW en Ingles) es aquel hardware cuyo diseño se hace disponible públicamente para que cualquier persona lo pueda estudiar, modificar, distribuir, materializar y vender, tanto el original como otros objetos basado en este diseño, así da libertar

de controlar la tecnología y al mismo tiempo compartir conocimientos y estimular la comercialización por medio del intercambio abierto de diseños (Open Source Hardware Association, 2019).

6.1.2. Protocolo I2C

El protocolo I2C (Circuito Inter-Integrado, en Ingles Inter-Integrated Circuit) es un protocolo de comunicación serial desarrollado por Philips Semiconductors en la década de los 80 y permite la comunicación de múltiples chips con dos vías de comunicación como la hace el protocolo UART, permitiendo comunicar varios maestros entre multitud de chip esclavos (Telsaben.com, 2017).

Esta tecnología se utilizará en el desarrollo del Gadget, para comunicar los chips de los sensores con el módulo maestro.

6.1.3. MultiTech

MultiTech es una empresa que se centra en el desarrollo de software y hardware Open Source para desarrollar dispositivos IoT.

6.1.3.1. MultiConnect Conduit Platform

MultiConnect Conduit es una puerta de enlace de comunicaciones celular escalable y configurable para aplicaciones industriales de IoT. Conduit permite a los usuarios conectar diferentes tipos de accesorios dependiendo de las necesidades de comunicación, como se puede ver en la ilustración 9.

Ilustración 9 - MultiTech Conduit - Imagen obtenida de https://www.multitech.com

Además, Conduit permite utilizar Node-RED de IBM o mLinux Open Embedded para el desarrollo de aplicaciones para monitorear y controlar los dispositivos IoT.

Esta tecnología se utilizará como Gateway para recoger los paquetes de los Gadgets y enviarlos al servidor Back-End.

6.1.3.2. MultiConnect mDot

El MultiConnect mDot (ver ilustración 10) es un módulo de red de área amplia de bajo consumo (LPWAN) preparado para LoRaWAN, permitiendo una comunicación de larga distancias. Este es compatible con el procesador ARM Cortex M4 y al usar la tecnología LoRa, mDot simplifica la implementación y la conectividad local para las aplicaciones de IoT, además de estar certificado por la CE.

Ilustración 10 - MultiTech mDot - Imagen obtenida de https://www.multitech.com

El módulo mDot se empleará para el desarrollo de los Gadget, enviando los paquetes de los datos capturados por los sensores tanto analógicos como digitales al Gateway.

6.1.4. Plataforma NET

NET es un marco de software de computadora administrado gratuito y de código abierto para los sistemas operativos Windows, Linux y macOS. El proyecto está principalmente desarrollado por Microsoft y publicado bajo la licencia MIT.

NET es fundamentalmente modular en su diseño y arquitectura. Los componentes del compilador, el tiempo de ejecución y la biblioteca son entidades independientes que se comunican a través de interfaces adecuadamente diseñadas. Esto permite que se incorporen o quiten componentes según las necesidades concretas. Las propias bibliotecas son modulares y se distribuyen mediante NuGet.

Permite el desarrollo de aplicaciones Web, aplicaciones de escritorio y aplicaciones móviles, en varias plataformas. Esta tecnología se utilizará en el desarrollo del servidor Back-End y aplicaciones Front-End (Microsoft, 2019).

6.1.5. Blazor

Blazor forma parte de la plataforma .NET permitiendo la creación de interfaces de usuario web interactivas utilizando C# en lugar de JavaScript. Las aplicaciones Blazor se componen de componentes de IU web reutilizables implementados usando C#, HTML y CSS. Además, puede ejecutar su código C# del lado del cliente directamente en el navegador utilizando WebAssembly (Microsoft, 2019). Esta tecnología se utilizará para el desarrollo de la aplicación web cliente.

6.1.6. Microsoft Azure SQL Server

Microsoft Azure SQL Server es un sistema de gestión de base de datos relacional de Cloud Computing, desarrollado por la empresa Microsoft que permite utilizar como lenguaje de desarrollo Transact-SQL (T-SQL), tiene una implementación del estándar SQL ANSI, utilizado para manipular y recuperar datos (DML) y para crear tablas y definir relaciones entre ellas (DDL) (Microsoft Azure, 2019).

Esta tecnología se utilizará como gestor de base de datos para almacenar los datos permanentes capturados por los Gadget.

6.1.7. Node-RED

Node-RED es una herramienta de programación para conectar dispositivos de hardware, API y servicios en línea de formas nuevas e interesantes.

Proporciona un editor basado en navegador que facilita la conexión de flujos mediante la amplia gama de nodos de la paleta que se pueden implementar en su tiempo de ejecución con un solo clic. (Node-RED, 2019)

Esta herramienta se utilizará en el desarrollo de la aplicación del Gateway.

6.1.8. MBED

MBED es una plataforma y sistema operativo para dispositivos conectados a Internet basados en microcontroladores ARM Cortex-M de 32 bits. Tales dispositivos también se conocen como dispositivos de IoT. El proyecto es desarrollado en colaboración por Arm y sus socios tecnológicos (Arm Mbed, 2019).

Proporciona la plataforma de software MBED OS con las herramientas para crear el firmware de los microcontroladores que se ejecuta en los dispositivos IoT, como compiladores, scripts de prueba y depuración. Además,

cuenta con un conjunto de bibliotecas que facilitan el desarrollo de dicho firmware, como comunicaciones, gestión de E/S, etc., ver ilustración 11 (Mbed OS, 2019).

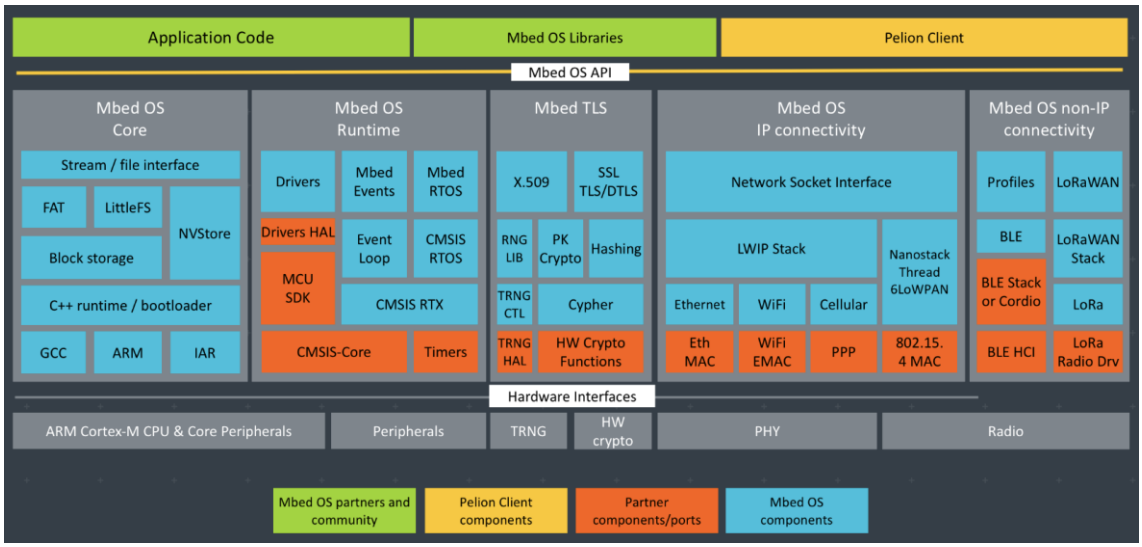

Ilustración 11 – The basic architecture of an Mbed board. Imagen obtenida de https://os.mbed.com/docs/mbed-os/v5.14/introduction/index.html.

Esta tecnología se utilizará en el desarrollo del Gadget.

6.2. Lenguajes de programación

6.2.1. C++

C++ es un lenguaje de programación diseñado en 1979 por Bjarne Stroustrup. Las características que considero más relevantes de este lenguaje son la eficiencia con el hardware, programación orientada a objetos, lenguaje fuertemente tipado, control de excepciones y sintaxis heredada del lenguaje C (cpluscplus, 2019).

Este lenguaje de programación será utilizado en el desarrollo del Gadget, pues las herramientas de desarrollo de la plataforma MBED utilizan este lenguaje.

6.2.2. C#

C# es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET. Su sintaxis básica deriva de C++ y utiliza el modelo de objetos de la plataforma .NET. Las características que considero más relevantes de este lenguaje son la programación orientada a objetos, lenguaje fuertemente tipado y control de excepciones.

Además, ofrece una cantidad de librerías que facilitan el desarrollo del software, como son Entity Framework Core para la gestión de datos, librería de Newtonsoft para el manejo de JSON, etc. (Microsoft, 2019).

Este lenguaje se utilizará en el desarrollo del Back-End y Front-End.

6.2.3. JavaScript

JavaScript (JS) es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Este tiene una sintaxis similar a C.

Se utiliza principalmente en el lado del cliente web permitiendo mejorar en la interfaz de usuario y páginas web dinámicas (MDN, 2019).

Este lenguaje se utilizará en el desarrollo de la aplicación del Gateway, porque Node-RED está hecho en JavaScript. Además de la aplicación cliente web, para la mejora de la interfaz.

6.3. Herramientas

6.3.1. mBed-Cli

Estas herramientas las utilizare para el desarrollo del Gadget, pues contiene los compiladores necesarios para generar el firmware del

microcontrolador, además de las herramientas para la depuración de dicho código (Arm Mbed, 2019).

6.3.2. Git

Git es un software de control de versiones, su propósito es llevar registro de los cambios del código fuente de las aplicaciones a desarrollar. Esta herramienta va a permitir gestionar el control de versión en todo el código del proyecto (Atlassian, 2019).

6.3.3. Microsoft Visual Studio Code

Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código en varios lenguajes como: C++, C#, Python, HTML, SQL, etc. (Microsoft, 2019). Esta herramienta se utilizará para escribir el código en C++ para el Gadget y poder modificar otros archivos de otras partes del proyecto.

6.3.4. Microsoft Visual Studio

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE) para Windows y macOS. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos, refactorización de código y uso de pruebas unitarias, compatible con múltiples lenguajes de programación, como C++, C#, Visual Basic .NET, F#, Java, Python, Ruby y PHP, etc. (Microsoft, 2019).

Permite a los desarrolladores crear aplicaciones web, servicios web, aplicaciones de escritorio y aplicaciones para dispositivos móviles. Esta herramienta se utilizará para el desarrollo del Back-End y Front-End.

6.3.5. Microsoft Azure DevOps

Microsoft Azure DevOps es un producto de Microsoft que proporciona administración de código fuente, informes, administración de requisitos, gestión de proyectos utilizando metodologías agiles, compilaciones automatizadas, pruebas y gestión de lanzamientos cubriendo todo el ciclo de vida del desarrollo de un proyecto (Microsoft, 2019).

Esta herramienta se utilizará para la aplicación de la metodología ágil.

6.3.6. Putty

Putty es un cliente SSH, Telnet, rlogin, y TCP raw Open Source y licenciado bajo la Licencia MIT (Putty.org, 2019). Esta herramienta se utilizará como consola entre el ordenador y el Gadget.

6.3.7. Microsoft SQL Management Studio

Microsoft SQL Server Management Studio (SSMS) es una aplicación de software que se utiliza para configurar, administrar todos los componentes dentro de Microsoft SQL Server. Una característica central de SSMS es el Explorador de objetos, que permite al usuario navegar, seleccionar y actuar sobre cualquiera de los objetos dentro del servidor (Microsoft, 2019).

Esta herramienta se utilizará para la conexión con Microsoft Azure SQL para realizar consultar a la base de datos o realizar cambios de la estructura.

6.3.8. Microsoft Azure Data Studio

Microsoft Azure Data Studio es una herramienta para tareas de administración, diseño y consulta de bases de Microsoft SQL Server, facilitando la realización de consultas SQL, el diseño de la estructura de las bases de datos, la administración de servidor y la comparación y conversión de bases de datos. Permite su ejecución en Windows, Linux y MacOS (Microsoft, 2019).

Esta herramienta se utilizará para la conexión con Microsoft Azure SQL para la administración de la base de datos.

6.3.9. Navegador web

Google Chrome es un navegador web, desarrollado por Google, derivado de proyectos de código abierto (como el motor de renderizado Blink). Este navegador se utilizará para el acceso de herramientas web y uso de la aplicación cliente para su correcto funcionamiento y depuración.

6.3.10. Postman

Esta herramienta nos permite construir y gestionar de forma cómoda nuestras peticiones a servicios REST (POST, GET, PUT, etc.) (Postman, 2019). Esta herramienta se utilizará para realizar pruebas con el servidor Back-End.

6.3.11. Xamarin

Xamarin es una herramienta que forma parte de la plataforma .NET, que permite utilizar el lenguaje de programación C#, para escribir aplicaciones móviles nativas para Android, iOS y Windows, y compartir código a través de múltiples plataformas, incluyendo Windows y macOS (Microsoft, 2019). Esta herramienta se utilizará para el desarrollo de la aplicación móvil.

6.3.12. EasyEDA

EasyEDA es una herramienta gratuita basada en la nueve que permite diseñar diagramas, circuitos y PCB a los ingenieros electrónicos (EasyEDA, 2019). Esta herramienta se utilizará para los esquemáticos de electrónica.

6.3.1. Adobe illustrator

Adobe illustrator es una aplicación que permite crear imágenes vectoriales, para luego generar imágenes binarias (Adobe, 2020). Esta aplicación se utilizará para la creación de los iconos de la aplicación.

6.3.2. Protoboard

La protoboard es una placa de pruebas en los que se pueden insertar elementos electrónicos y cables con los que se pueden armar circuitos electrónicos sin necesidad de soltar ningún componente como se puede ver en la ilustración 12 (330ohms, 2016).

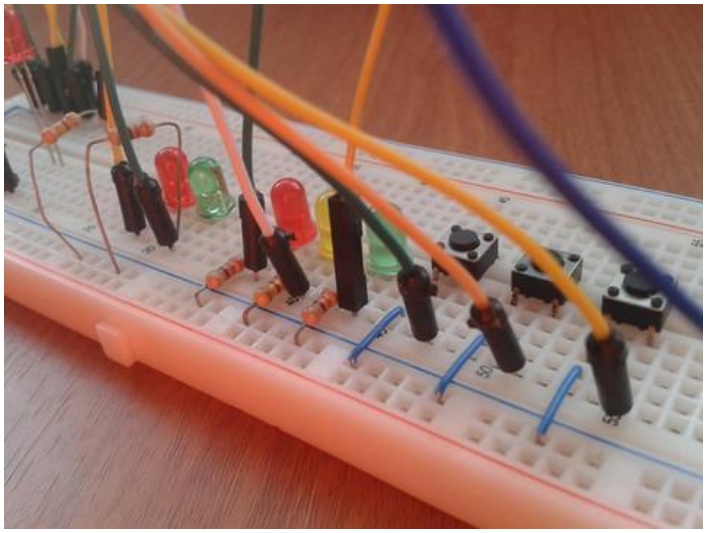

Ilustración 12 - Protoboard, imagen obtenida de blog.330ohms.com

La protoboard se utilizará para el desarrollo del Gadget.

6.3.3. MultiTech mDot Developer Kit

El MultiTech mDot Developer Kit permite conectar un módulo MultiTech mDot para hacer testeo, programación y evaluación, en la ilustración 13 se puede ver dicha placa (MultiTech, 2019). Esta placa se utilizará para el desarrollo del Firmware que se implementará en el Gadget.

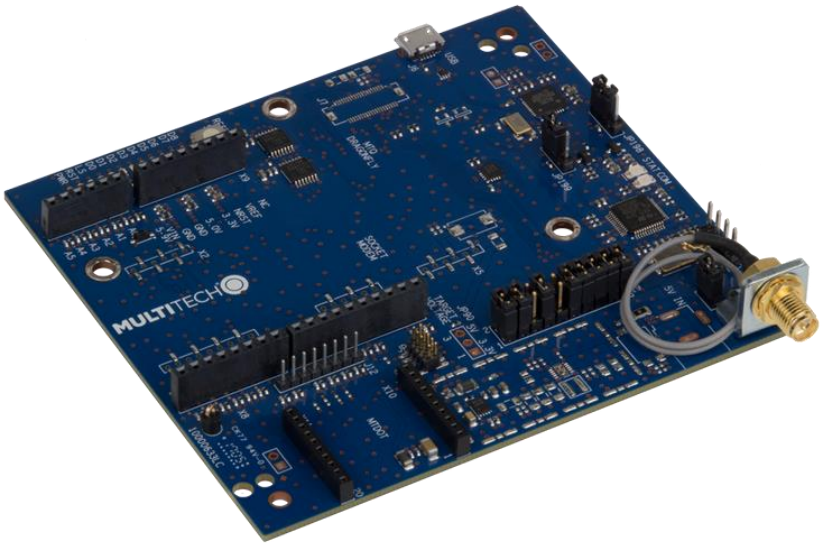

Ilustración 13 - MultiTech mDot Developer Kit, imagen obtenida de www.multitech.com

6.4. Resumen

A continuación, se realiza un resumen especificando las tecnologías utilizadas para el desarrollo de este TFG, en las diferentes partes de este.

Para el desarrollo del Gadget y la comunicación con el servidor Back-End, se opta por la tecnología que oferta la empresa MultiTech (MultiTech, 2019), que presenta un portafolio de productos y herramientas como las descritas a continuación:

- Hardware IoT mediante la plataforma ARM MBed OS, que dispone de herramientas para el desarrollo de la aplicación mediante compiladores C y C++.
- Diferentes Gateway que permiten la comunicación LoRa para la recepción de los datos de los dispositivos IoT, con una comunicación de topología en estrella con una distancia de hasta 15Km, que lleva integrado un sistema operativo Linux. Estos se pueden programar a alto nivel mediante la plataforma Node-RED, que se basa en JavaScript.

Para la parte del desarrollo del servidor Back-End y aplicaciones Front-End, se ha optado por el uso de la plataforma .NET de Microsoft y sus herramientas. Se ha seleccionado esta plataforma por su versatilidad a la hora

de desarrollar cualquier tipo de aplicación y todos los servicios que Microsoft Azure ofrece en Cloud Computing para el desarrollo del TFG, usando las siguientes tecnologías:

- Alojamiento de la persistencia de datos mediante Microsoft Azure SQL Server, en la nube. El lenguaje de programación usado sería: SQL y T-SQL.
- El servidor Back-End uso la tecnología ASP.NET Core Web API alojado en Microsoft Azure, a que permite el uso de sistemas Linux o Windows, ofreciendo una gran estabilidad y monitorización del sistema. El lenguaje de programación utilizado sería C#.
- Para la aplicación web cliente se utilizará Blazor con la que se puede crear una aplicación Web usando tecnología de la plataforma .NET, gracias a WebAssembly (una nueva tecnología soportada por casi todos los navegadores importantes como Chrome, Firefox, Edge, etc.). Se alojará en Microsoft Azure, en sistemas Windows o Linux. El lenguaje de programación a utilizar seria C#, HTML y JavaScript.
- La aplicación para dispositivos móviles se utilizará Microsoft Xamarin que ofrece una herramienta de programación de aplicaciones con C# para iOS y Android nativo, utilizando la plataforma .NET.

En la ilustración 14 podemos observar un esquema general de la infraestructura que presentará el TFG incluyendo las tecnologías propuestas.

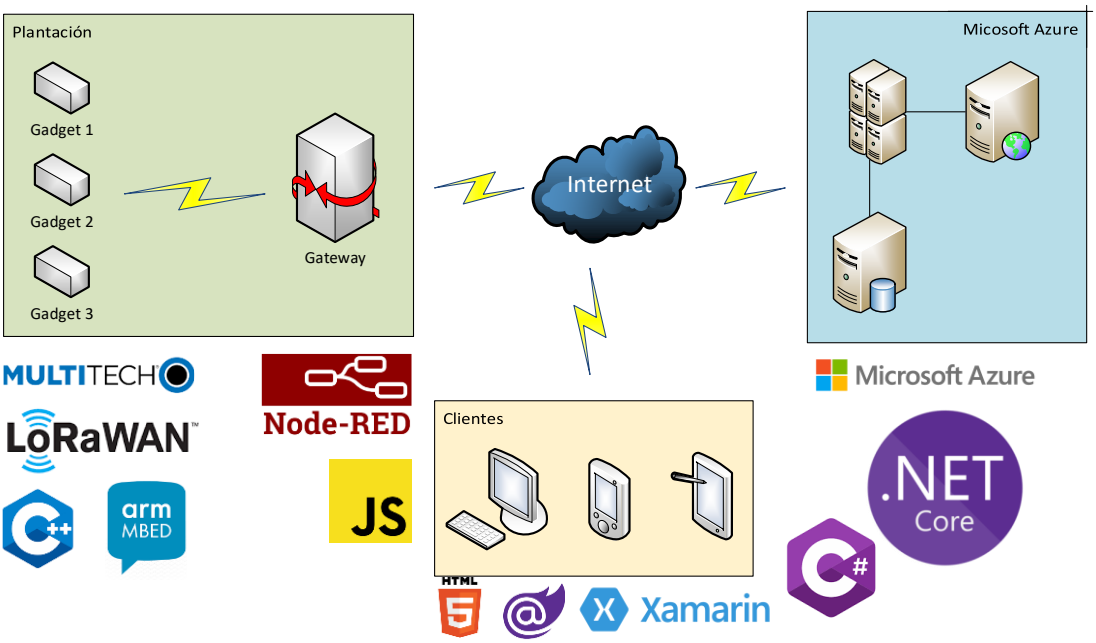

Ilustración 14 - Infraestructura con las tecnologías utilizadas.

7. Estimación de recursos y planificación

En este apartado se realizará una estimación de la duración y la planificación de los Sprint necesarios para la elaboración, además de un coste económico para el desarrollo del TFG.

Como para el desarrollo del proyecto, estamos siguiendo una metodología Scrumban, definiremos un Product Backlog que detallaremos en épicas, cuestiones y tareas a realizar para conseguir la realización del proyecto.

Se va a utilizar la herramienta Microsoft Azure DevOps que va a ayudar con la gestión de la metodología. Esta herramienta permite registrar el Product Backlog, tiene un tablero Kanban donde ir colocando las tareas que están pendientes por hacer, las que se están haciendo y las finalizadas. Además, permite asignar las tareas a los Sprints para calcular la velocidad del proyecto.

7.1. Indicar las estimaciones utilizadas

Se han identificado las historias de usuario para preparar el Product Backlog como se puede ver en la tabla 4, donde se han especificado las épicas, las cuestiones y sus tareas, donde para cada tarea se le establecerá una prioridad y su estimación.

Para dar prioridad a las tareas se utilizará la técnica MoSCoW que se basa en la segmentación y agrupación de los elementos del Product Backlog para poder asignar las tareas en orden de importancia, donde la escala de prioridades es la siguiente:

- **Must** (M): El producto debe tenerlo.
- **Should** (S): El producto debería tenerlo.
- **Could** (C): El producto puede tenerlo, pero no es necesario.
- **Won’t** (W): Requerimientos descartados de momento pero que en un futuro podrían contemplar.

Como las tareas son distintas en alcance, complejidad y dificultad las mediremos en puntos de historia (Story Points) unas frente a otras. Se utilizará la técnica de Planning Pocker o Estimación Pocker, en la que se usaran unas cartas marcadas con números basadas en la serie de Fibonacci, con los valores: 0, ½, 1, 2, 3, 5, 8, 13, 20, 40 y 100, así cada miembro del equipo votará usando su criterio (Carmen Lasa Gómez, 2018).

Cabe destacar los siguientes valores especiales, que indicamos a continuación:

- 0, para estimar historia que ya están hechas o que son prácticamente inmediatas, por lo que tareas que impliquen menos de una 1 hora serán estimadas con un 0.
- 100, se reservará para tareas abordables, pero muy grandes que habrá que descomponer en tareas más simples.

También hay que descartar que como vamos a utilizar una metodología Scrumban vamos a definir el WIP con un máximo de 4 tareas abiertas.

Para la realización del TFG se dispondrá de unas 544 horas, como la disponibilidad de tiempo semanal que se puede dedicar a este proyecto son de unas 34 horas, por lo que se estiman unas 16 semanas para la realización del proyecto.

Velocidad, trabajo y tiempo son las tres magnitudes para medir la gestión del proyecto, estas tres magnitudes se relacionan con la siguiente formula: Velocidad = (Puntos de historia / Tiempo).

Para la realización de este proyecto se obtiene una velocidad de 0,444 puntos de historia por hora, que consiste en la división de 241,5 puntos de historia entre 544 horas.

Para tener diferentes versiones de estimación para el TFG se realizará una estimación: optimista, realista y pesimista. El margen para estas nuevas

estimaciones será de más o menos un 20 por ciento de velocidad, obteniendo como resultado la tabla 4.

Visión	Velocidad	Duración (horas)
Optimista + 20%	0,533	453
Realista	0,444	544
Pesimista - 20%	0,355	680

Tabla 4 - Visión de estimaciones

Se define que cada Sprint va a tener una duración de 2 semanas, así la cantidad de horas por Sprint será de 68 horas, a partir de estas estimaciones se calcula el número de Sprint para cada una de las estimaciones como se puede ver en la tabla 5.

Visión	Velocidad	Duración horas	Sprint	Puntos por Srpint	Horas	Semanas
Optimista +20%	0,533	453	7	37	68	14
Realista	0,444	544	8	31	68	16
Pesimista - 20%	0,355	680	11	25	68	22

Tabla 5 - Calculo de Sprint por visión de estimaciones

Analizando la estimación realista, la duración total del proyecto está fijada en 544 horas con una velocidad de 0,444 puntos de historia por hora. Por lo que para el desarrollo del proyecto se necesitaran sobre unos 8 Sprint, pudiendo conseguir un máximo de 31 puntos de historia por Sprint.

Mientras la visión optimista, la duración del proyecto seria sobre unas 453 horas, con una velocidad de 0,533 puntos de historia por hora que se podría realizar en 7 Sprints.

Por último, la visión pesimista la duración del proyecto seria sobre unas 680 horas con una velocidad de 0,355 puntos de historia por hora, realizando el TFG en 11 Sprints con un máximo de 25 puntos de historia por Sprint.

7.2. Realizar una planificación temporal del proyecto

En la tabla 7 se puede ver el Product Backlog donde hemos agrupado las tareas en épicas y cuestiones. A cada tarea se le ha asignado su priorización, sus puntos de historia y su correspondiente Sprint.

Se han identificado cada una de las épicas, cuestiones y tareas con un identificador (Id) que han sido obtenidos de la aplicación Microsoft Azure DevOps al registrar cada una de ellas.

En la tabla 6 se puede consultar un resumen de los Sprint con la cantidad de puntos de historia que van a resolver en cada uno de ellos, donde se puede ver que en el Sprint 3 sobrepasa la cantidad de historia establecidas por Sprint y el ultimo Sprint se queda con un número menor de historias, por si surge alguna tarea que no estaba contemplada en la planificación inicial.

Sprint	Puntos de historia
1	30,0
2	31,0
3	32,0
4	31,0
5	30,5
6	31,0
7	31,0
8	25,0

Tabla 6 - Resumen de puntos de historia por Srpint.

Product Backlog									
Épicas		Cuestiones		Tareas		Prioridad	Story Points	Realizado	Sprint
Id	Titulo	Id	Titulo	Id	Titulo				
148	Gadget	151	Monitorización de factores	160	Estudiar herramientas de desarrollo del Firmware	M	8	N	1
				161	Comunicación puerto serie para interacción	M	2	N	1
				162	Lectura analógica	M	8	N	1
				163	Lectura/Escritura digital	M	2	N	1
				211	Estudiar comunicación bus I2C	M	5	N	1
				164	Comunicación sensores con bus I2C	M	2	N	1
				165	Búsqueda de dispositivos I2C	S	3	N	1
				168	Gestor de sensores	S	5	N	2
				218	Estudiar tipo de sensores del mercado	M	5	N	4
				217	Implementar sensores	M	13	N	4
		153	Enviar datos al Gateway	167	Codificación de datos	M	5	N	2
				166	Envió de datos al Gateway	M	5	N	2
		208	Energía	209	Estudio de suministro de energía	M	8	N	4
				210	Implementación de la energía	M	3	N	4
				216	Medición nivel de la energía	S	2	N	4
				220	Ahora de energía	C	3	N	7
		152	Localización	169	Estudio de sistemas GPS hardware	C	13	N	7
				170	Enviar datos de localización al Gateway	C	5	N	7
147	Gateway	154	Recepción datos Gadget	171	Estudiar entorno de desarrollo Node-RED	M	8	N	2

				172	Recepción de paquetes del Gadget	M	3	N	2
		155	Envió de datos al Back-End	174	Transformar paquetes Gadget a JSON	M	5	N	2
				175	Enviar datos al servidor Back-End	M	8	N	3
		179	Estado del Gateway	180	Obtener datos del estado del Gateway	S	5	N	3
				182	Enviar estado Gateway en formato JSON al Back-End	S	2	N	3
		173	Localización	176	Lectura de posicionamiento del Gateway	C	3	N	3
				177	Envió de datos de posicionamiento al Back-End	C	2	N	3
148	Back-End	156	Recepción datos Gateway y persistencia de datos	202	Crear base de datos en Azure	M	0,5	N	3
				182	Modelo de datos	M	5	N	3
				203	Implementar servidor de aplicación en Azure	M	0,5	N	3
				183	Controlador recepción datos Gadget	M	2	N	3
				184	Controlador recepción estado Gateway	C	2	N	3
				205	Controlador recepción localización Gateway	S	2	N	3
		159	Acceso aplicaciones Front-End	185	Controlador acceso datos Gadgets	M	3	N	5
				186	Controlador acceso datos Gateways	S	3	N	5
				219	Controlador manteniendo de gadgets	C	5	N	8
149	Front-End	157	Aplicación Web	207	Crear base aplicación IU	M	8	N	5
				213	Cuadro de mandos	M	5	N	5
				204	Implementar servidor de aplicación en Azure	M	0,5	N	5
				189	Consultar datos Gadgets	M	5	N	5
				191	Consultar históricos Gadgets	M	3	N	5
				198	Consultar datos Gateways	S	5	N	7
				190	Consultar localización Gadgets	C	5	N	8
				199	Consultar localización Gateways	C	5	N	7
				192	Exportar datos a Excel	M	3	N	5

				215	Mantenimiento de Gadgets	C	5	N	8
		158	Aplicación móvil	206	Crear base aplicación IU	M	8	N	6
				212	Cuadro de mandos	M	8	N	6
				194	Consultar datos Gadgets	M	5	N	6
				196	Consultar históricos de los Gadgets	M	5	N	6
				200	Consultar datos Gateways	S	5	N	6
				195	Consultar localización de los Gadgets	C	5	N	8
				201	Consultar localización de los Gateways	C	5	N	8
Suma de puntos de historias…							**241,5**		

Tabla 7 - Product Backlog.

7.3. Realizar una valoración de la dedicación y el coste económico

Para el desarrollo del proyecto se necesita un graduado en informática que, aplicando el convenio de dependencia mercantil, el bruto anual es de 15.988,08 EUR con 14 pagas, más la seguridad social de la empresa que corresponde sobre un 30%, además de aplicar un gasto de estructura del 15% para cubrir los gastos, como las herramientas de desarrollo (Suscripción Visual Studio), ordenadores y otros gastos de la empresa. Ver tabla 8.

	%	Importe	EUR
Sueldo Bruto anual			15.988,08 €
Seguridad social empresa	30	4.796,42 €	20.784,50 €
Gastos de estructura	15	3.117,68 €	23.902,18 €

Tabla 8 - Costo de un graduado de informática anual

Tras el cálculo, teniendo en cuenta que el contrato sería de 8 horas al día y un mes tiene una media de 21 día laborables el coste de la hora del graduado de informática seria de 10,16 EUR/Hora. Para el desarrollo del proyecto se han establecido unas 544 horas, el coste total en mano de obra sería de unos 5.527,04 EUR.

Además, debemos tener en cuenta los costes del hardware y otros servicios para el desarrollo de los Gadgets, el Gateway y el Backend, en la tabla 9 se puede ver el coste más impuestos.

Producto	Importe
Kit de desarrollo MultiTech	1.051,00 €
MultiTech mDot	476,10 €
Batería LG HG2 18650 3.000 mAh 20 A	192,00 €
Cargador baterías Efest Lush Q4 baterías Litio 18650	15,38 €
Conversores Step-Up/Step-Down	71,40 €
Varios componentes de electrónica	1.092,65 €
Compra de dominios	14,95 €
Compra certificado SSL Backend	23,90 €
Compra certificado SSL Aplicación web	23,90 €
Total…	2.951,28 €

Tabla 9 - Coste de materiales

Hay que tener en cuenta los servicios de Cloud Computing que contaran con 2 servidores de aplicaciones, uno para el Backend y otro para la aplicación web, además del servidor de base de datos. También una línea de móvil para la conexión a internet del Gateway, donde se puede ver el coste anual más impuestos en la tabla 10.

Producto	Unidades	Precio	Importe	Anual
Azure App Service/Mes	2	11,08 €	22,16 €	265,92 €
Azure SQL Server/Mes	1	12,65 €	12,65 €	151,80 €
Línea móvil 2Gb/Mes	1	12,40 €	12,40 €	148,76 €
		Total…	47,21 €	566,48 €

Tabla 10 - Coste servicios de pago por uso

El total de coste del proyecto sería de 9.044,80 EUR, como se puede ver en la tabla 11.

Tipo	EUR
Materiales	2.951,28 €
Servicios pago por uso	566,48 €
Mano de obra	5.527,04 €
Total…	**9.044,80 €**

Tabla 11 - Coste total del proyecto

Respecto a los ingresos que se pretende obtener para este proyecto, habría que estudiar las diferentes formas de alquilar los equipos y estudiar las tarifas a aplicar a los clientes, así como un alquiler por conexión por zona del Gateway, más una cantidad por Gadgets alquilados, con diferentes tarifas según la cantidad de equipos alquilados.

8. Desarrollo del contenido del proyecto

Según la planificación realizada del proyecto se llevará a cabo en 8 Sprints y para cada uno de ellos se describirán las metas, resultados y revisiones.

8.1. Sprint 1

Para este primer Sprint se va a comenzar a desarrollar el Gadget, por lo que se tendrá que empezar a conocer las herramientas que ofrece MDEB para el desarrollo del firmware. Una vez conocida las herramientas de desarrollo, se comenzará a interactuar con el hardware a través del puerto serie, como la visualización y entrada de datos por pantalla, lectura analógica, escritura digital y comunicación con el bus I2C.

8.1.1. Planificación

Según la planificación realizada para el Sprint 1 tendremos que acometer 30 puntos de historia, donde en la tabla 12 se han identificado las tareas a realizar en este Sprint, además de su prioridad y la estimación. Las tareas se realizarán en el orden indicado.

Tareas		Prioridad	Story Points	Realizado	Sprint
Id	Titulo				
160	Estudiar herramientas de desarrollo del Firmware	M	8	N	1
161	Comunicación puerto serie para interacción	M	2	N	1
162	Lectura analógica	M	8	N	1
163	Lectura/Escritura digital	M	2	N	1
211	Estudiar comunicación bus I2C	M	5	N	1
164	Comunicación sensores con bus I2C	M	2	N	1
165	Búsqueda de dispositivos I2C	S	3	N	1
	Suma de puntos de historia		30		

Tabla 12 - Sprint Backlog 1

8.1.2. Metas

Tras la planificación, unas de las metas de este Sprint es generar un firmware que sea capaz de poder interactuar con el mediante una herramienta terminal como es Putty, poder leer valores de una entrada analogía, interpretar dichos valores, interactuar con entradas/salidas digitales y realizar la comunicación con Hardware Open Source a través del bus de comunicaciones I2C.

Para realizar el firmware es necesario adquirir el conocimiento de las herramientas que ofrece Mbed, así como entender el funcionamiento del bus I2C, como forma de comunicar y limitaciones que este ofrece. Estas herramientas y sistemas son totalmente desconocidos.

Como resultado final se debería obtener una pequeña aplicación con la capacidad de visualizar datos de la lectura de un puerto analógico, escribir en un puerto digital, permitir la comunicación con un Hardware Open Source mediante el bus I2C, además de localizar si está conectado a nuestro hardware.

8.1.3. Resultados

Durante el desarrollo de este Sprint, se ha realizado todas las tareas marcadas relacionadas con la Monitorización de factores del Gadget.

La primera tarea identificada con id 160, ha sido estudiar las diferentes herramientas que ofrece Mbed para el desarrollo del firmware para el dispositivo MutiTech mDot, pudiendo utilizar una herramienta en línea que ofrece Mbed o Arm Mbed Cli para trabajar en local.

De las pruebas realizadas se ha optado por trabajar con la herramienta Arm Mbed Cli, en el anexo 10.1.1 se incluye los procedimientos de instalación de dicha herramienta.

Se ha tenido que estudiar la librería que ofrece Mbed-OS (Arm Mbed, 2019) para el desarrollo del firmware, dicha librería permite la interacción con el

hardware, como permitir diferentes sistemas de comunicaciones, E/S de ficheros, E/S digitales, E/S analogías, etc..

Además, como se ha utilizado la librería que ofrece MultiTech para facilitar la gestión del hardware mDot a la hora de las comunicaciones LoRaWAN, ofreciendo la librería libmDot-mbed5 (Multitech/libmDot-mbed5, 2019), con diferentes versiones disponibles. Para este proyecto se ha optado por la última versión disponible 3.2.1, que es compatible con la librería Mbed-OS versión 5.13.1, que no es la última, consultando la documentación facilitada por MultiTech. Para que se pueda compilar el firmware correctamente se tienen que seguir las compatibilidades de versiones, si no este no funcionara. En el anexo 10.1.2 se incluye el procedimiento de creación de un proyecto y su compilación.

Es muy importante elegir las versiones de las librerías adecuadas recomendadas por el fabricante del hardware, para poder generar el firmware correctamente. Para conseguir el firmware se ha de seleccionar un perfil de compilación adecuado al hardware, este perfil es llamado Toolchain y seleccionar el tipo de binario a generar por el compilador adecuado para el hardware que tiene que ser compatible con dichas librerías, esto es llamado target.

En la tarea id 161, se ha realizado una pequeña aplicación donde se visualizan datos en pantalla pudiendo interactuar con el teclado, esta tarea no ha tenido ninguna complejidad una vez se han seleccionado las librerías correctas. Se ha utilizado el editor Visual Studio Code para la escritura de código, una ventana terminal del sistema para la compilación y una herramienta terminal como Putty con la que se ha podido ver los resultados.

En la tarea id 162, que consiste en la Lectura de sensor analógico, se ha procedido a estudiar las clases que proporciona Mbed-OS para la lectura de un puerto analógico, se ha realizado una aplicación que hace una lectura de dicho puerto analógico, con la ayuda de una Protoboard se ha realizado un pequeño circuito como el de la ilustración 16 y se ha visualizado el dato leído. El principal problema de la lectura de un puerto analógico ha sido la interpretación de los

datos, pues mDot tiene la capacidad de poder leer valores comprendidos entre 0V a 3V, que son representados con un valor tipo float (0.0 a 1.0) o tipo unsigned int (0x0 a 0xFFFF).

En la tarea id 163, Lectura/Escritura de un sensor digital, se añadido a la placa de prototipo un diodo y resistencia conectado al mDot como se puede ver en la ilustración 16 y al pulsar una tecla sobre el terminal Telnet se enciende o pagar sucesivamente. En la realización de esta tarea no ha habido problemas, una vez encontrada la clase para la escritura del puerto digital.

A continuación, se ha procedido a estudiar el bus I2C, tarea id 211, para ver su funcionamiento, pues como vamos a utilizar Hardware Open Source, hay multitud de hardware que utilizan dicho bus para la lectura o escritura de datos ofrecidos por dichos hardware.

Una vez estudiado el bus I2C, se ha procedido a probar un sensor BME280 como el que se puede ver en la ilustración 15, que permite la lectura de temperatura, humedad y presión atmosférica mediante el bus I2C, que consiste en la tarea id 164. Para ello se ha añadido la librería del sensor BME280 que está disponible en la web Mbed-OS (HereLab, 2019). Además, en la plataforma Mbed-OS se puede encontrar multitud de librerías de sensores que se pueden utilizar en los proyectos y permiten muy fácilmente interactuar con dichos sensores. En la ilustración 16 hay un esquemático de la electrónica empleada para el desarrollo de este Firmware.

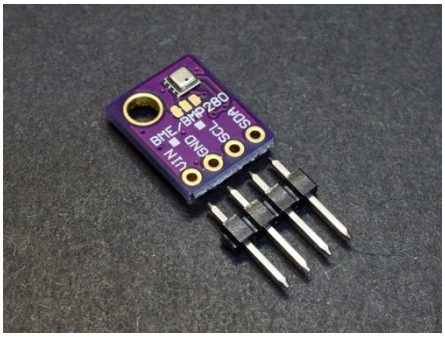

Ilustración 15 - Sensor BMP280, imagen obtenida de protosupplies.com

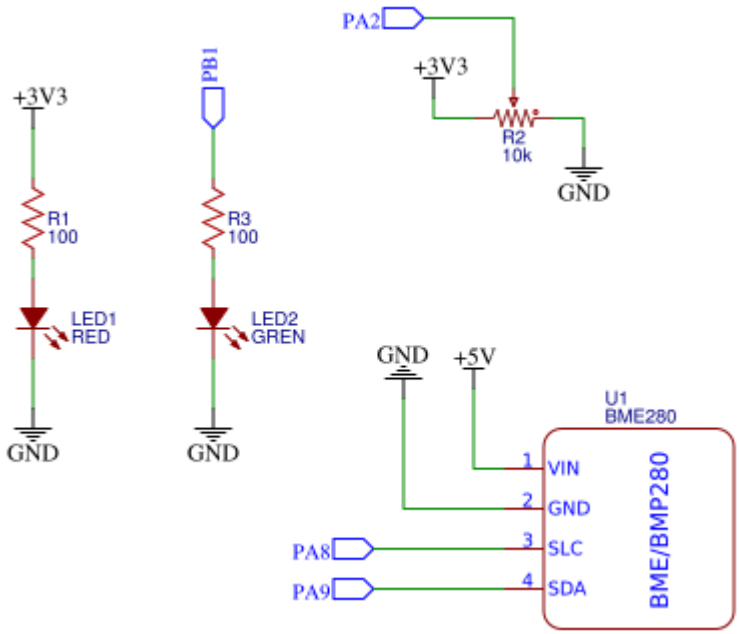

Ilustración 16 – Esquemático electrónica Sprint 1

En la ilustración 16, se puede observar las conexiones PA2, PB1, PA8 y PA9 son los Pins de conexión del MultiTech mDot, que en el Anexo 10.3 se puede ver un diagrama de Pins correspondientes.

Por último, se ha realizado una pequeña aplicación que permite escanear los sensores conectados al bus I2C para determinar que sensores y la cantidad que hay conectados al bus del mDot, que corresponde con la tarea id 165.

En esta tarea de comunicación con el bus I2C no ha tenido mayor dificultad una vez encontrada la clase adecuadas y entender el funcionamiento de dicho bus.

Como resultado en la ilustración 17 se puede ver la electrónica desarrollada, con la Protoboard y los componentes utilizados.

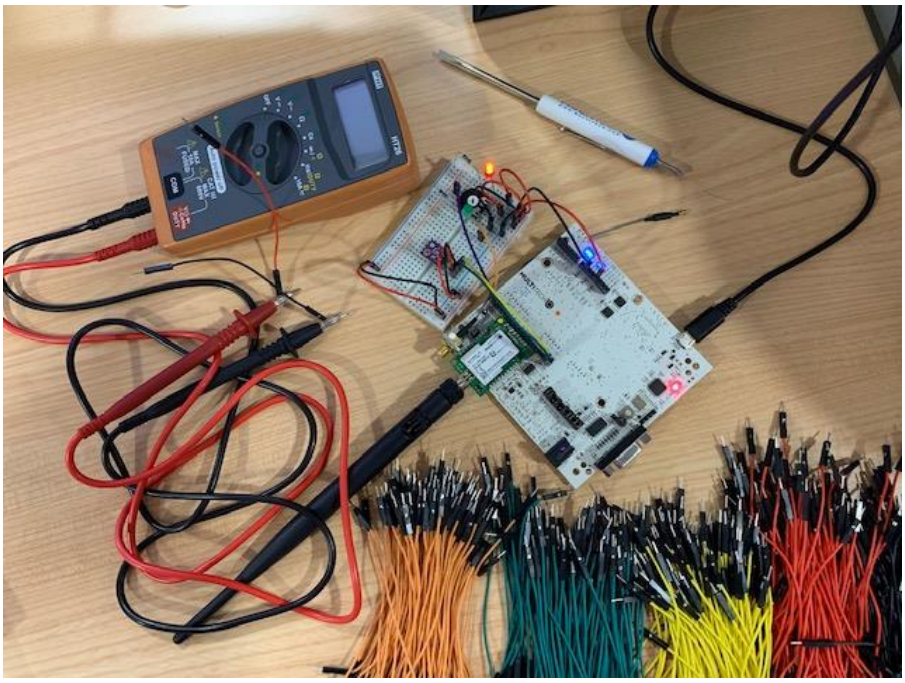

Ilustración 17 - Foto del hardware desarrollado en el Sprint 1

En la ilustración 18 se puede ver una salida del terminal, la cantidad de dispositivos conectados al bus I2C, que en este caso es 1, también se puede observar el valor de del potenciómetro que ha capturado el puerto analógico, el Led de la salida digital si está encendido o apagado, además de la temperatura y presión que se ha obtenido del sensor BME260.

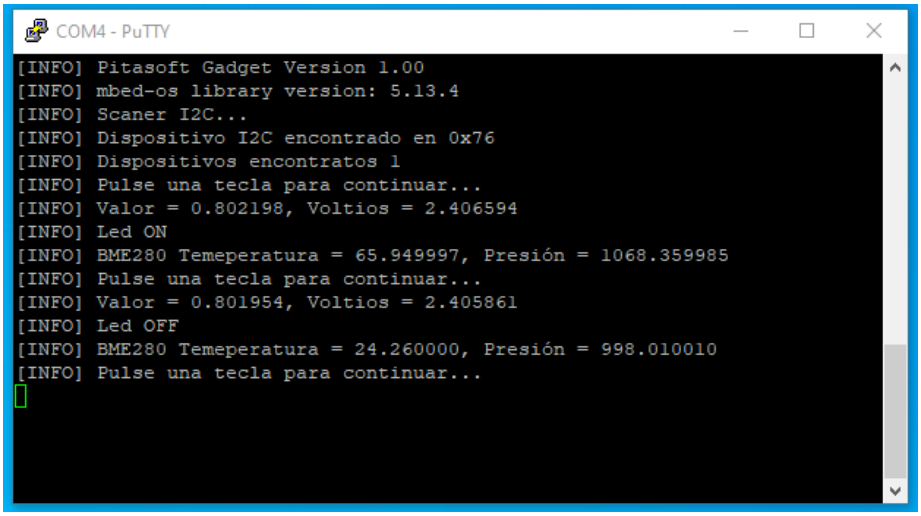

Ilustración 18 - Salida del terminal Sprint 1

8.1.4. Revisión

Al tener claro los resultados que se quieren obtener en este Sprint, no se han producido ningún cambio, pues principalmente se pretendía interactuar con

el hardware y sus herramientas. No ha quedado ninguna tarea pendiente para hacer en el siguiente Sprint.

8.1.5. Retrospectiva

En la tabla 13 se puede ver la comparativa entre los puntos de historia estimados con los reales para la realización del Sprint.

Tareas		Prioridad	Story Points	Realizado	Sprint	Story Points Reales
Id	Titulo					
160	Estudiar herramientas de desarrollo del Firmware	M	8	S	1	8
161	Comunicación puerto serie para interacción	M	2	S	1	2
162	Lectura analógica	M	8	S	1	9
163	Lectura/Escritura digital	M	2	S	1	1
211	Estudiar comunicación bus I2C	M	5	S	1	5
164	Comunicación sensores con bus I2C	M	2	S	1	3
165	Búsqueda de dispositivos I2C	S	3	S	1	3
	Suma de puntos de historia		30			31

Tabla 13 - Comparación de los puntos de historia estimados y reales del Sprint 1

Se han producido diferencias respecto a los puntos de historia en la tarea id 162, en la lectura de puerto analógico se ha tenido que realizar una consulta al servicio técnico de MultiTech, para aclarar el funcionamiento de este y se ha tenido que esperar respuesta por parte de este.

Mientras en la tarea id 163 una vez conocido el entorno y la biblioteca de clases se ha empleado menos tiempo.

En la tarea id 164, en la comunicación en el bus I2C con un sensor SHT-10 se han producido problemas, pues este no ha podido establecer la comunicación mediante este bus, pero con otro sensor ha funcionado correctamente, por lo que se ha perdido tiempo en las pruebas.

En la ilustración 19 se puede observar una gráfica Brundown donde se puede ver la evolución del desarrollo del Sprint 1.

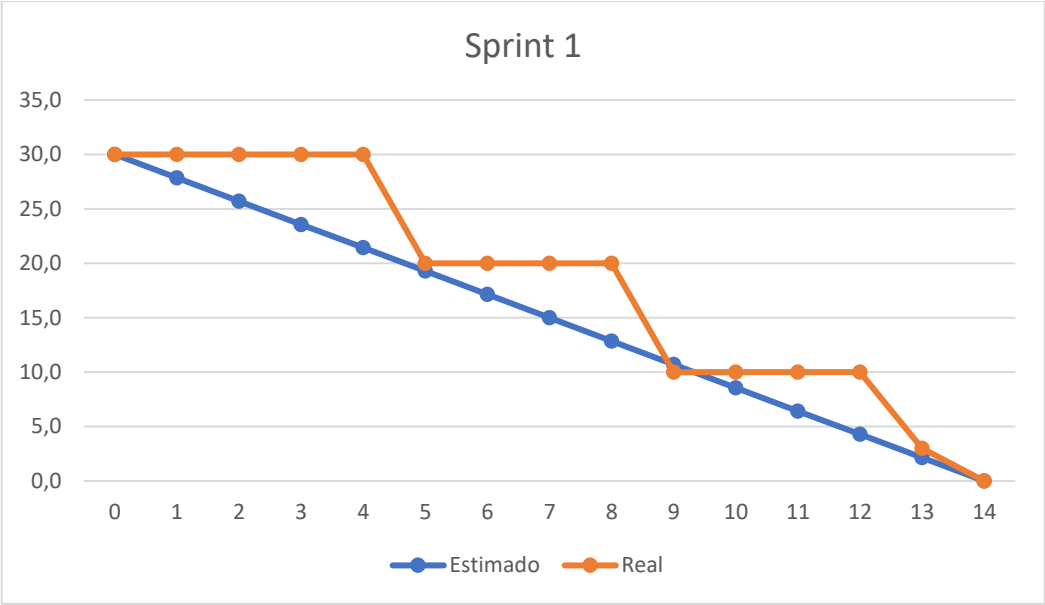

Ilustración 19 - Gráfico Brundown Sprint 1

8.2. Sprint 2

8.2.1. Planificación

En este segundo Sprint se tendrá que acometer 31 puntos de historia, donde en la tabla 14 se han identificado las tareas a realizar en este Sprint, además de su prioridad y la estimación. Estas tareas se van a realizar en el orden indicado en dicha tabla.

Tareas		Prioridad	Story Points	Realizado	Sprint
Id	Titulo				
168	Gestor de sensores	S	5	N	2
167	Codificación de datos	M	5	N	2
166	Envió de datos al Gateway	M	5	N	2
171	Estudiar entorno de desarrollo Node-RED	M	8	N	2
172	Recepción de paquetes del Gadget	M	3	N	2
174	Transformar paquetes Gadget a JSON	M	5	N	2
	Suma de puntos de historia		31		

Tabla 14 - Planificación Sprint 2

8.2.2. Metas

Las metas de este segundo Sprint consisten en crear una clase llamada Gadget que pretende gestionar los sensores y enviar datos al Gateway. Esta clase ha de poder registrar los diferentes sensores que se utilizaran, además se ha de estudiar la forma de enviar los paquetes al Gateway y como registrar dichos Gadgets al Gateway para permitir su recepción.

Una vez que se han enviado los datos al Gateway, se ha de verificar que son recibidos y transformarlos en JSON para su posterior envió al servidor Back-End.

Se ha de adquirir conocimiento del funcionamiento del Gateway. Como para este proyecto se va a utilizar el dispositivo MultiTech Coundit, que dispone de un sistema Linux embebido y una interfaz gráfica que permite una fácil configuración. Además de disponer de varias herramientas para poder

programarlo, en este proyecto se va a utilizar Node-RED por lo que habrá que adquirir los conocimientos de dicha herramienta, para aprender a recibir los paquetes enviados por los Gadgets y prepararlos para su posterior envió.

Como resultado final, se debería tener el Gadget más evolucionado, capaz de poder enviar paquetes al Gateway y este poder recibir dichos paquetes para generar un JSON para su posterior envió al servidor Back-End.

8.2.3. Resultados

En el desarrollo de este Sprint, se han realizado todas las tareas indicadas relacionadas con la cuestión de Monitorización de factores y Enviar datos al Gateway del Gadget, además de las cuestiones de Recepción de datos del Gadget y Envió de datos al Back-End del Gateway.

La primera tarea, identificada con el número id 168, se ha procedido a crear un Gestor de sensores, que ha sido la creación de varias clases que describiremos a continuación para el control del Gadget:

- Clase abstracta **StepBase,** define un método virtual llamado **run** donde se le pasa un contador de Step, que va a tener un rango de 1 a 100. Esta clase nos define un Step, que es un paso de ejecución o ciclo de ejecución del Gadget donde se puede programar las funciones personalizables que se desea que ejecute el Gadget en cada paso.
- Clase abstracta **GadgetBase**, es la encargada del control de flujo del Gadget como se puede ver en la ilustración 20, definiendo los siguientes pasos:
    - Método virtual **initialize**, donde se programará la configuración inicial.
    - Método virtual **firstTime,** que se ejecutara solo una vez en el ciclo del Step.
    - Método virtual **beforeStep**, donde se puede ejecutar código antes que se ejecute el Step.

- Método virtual **afterStep**, donde se ejecutará el código después de ejecutar el Step.
- Método virtual **sleep**, que se programará el código necesario para que el Gadget paralice su actividad, ahorro de energía.
- Método virtual **afterDyning**, donde se programarán las acciones necesarias después de despertarse el Gadget.

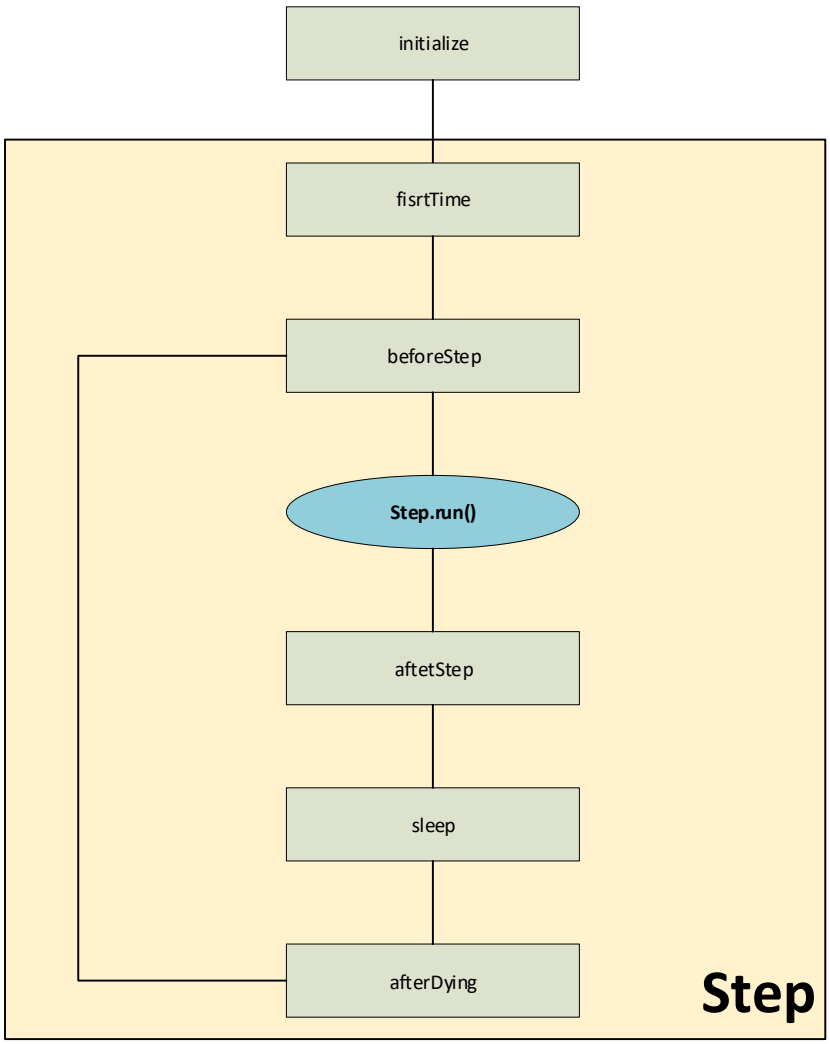

Ilustración 20 - Control de flujo del Gadget

- Clase **SampleBuffer**, que hereda de **vector<uint8_t>** que se utiliza para almacenar los datos que se leerán de los sensores, proporciona los siguientes métodos:
  - Método **pustData,** para almacenar datos de los sensores.
  - Método **clearBuffer,** vaciar el buffer de datos.
  - Método **isEmptyBuffer,** para saber si el buffer está vacío.
  - Método **bufferSize,** conocer el tamaño del buffer.

  - Método **byteAt,** para acceder a un byte del buffer.
- Clase abstracta **SensorBase,** permite definir las acciones que se pueden realizar con un sensor, este será identificado con un identificador y nombre, además de definir los siguientes métodos para el control del sensor:
  - Método virtual **initialize,** se utilizará para inicializar la configuración del sensor.
  - Método virtual **read,** es para la lectura de los datos del sensor, pasando como parámetro un **SampleBuffer** donde depositar los datos.
  - Método virtual **sleep.** para poner el sensor en un estado de ahorro de energía.
  - Método virtual **wakeup**, para poner el sensor en funcionamiento.
- Clase abstracta **SensorI2CBase,** que hereda de la clase **SensorBase** para la gestión de dispositivos que utilizan el bus I2C para el acceso a los datos, donde se indicaría la dirección del dispositivo y la clase I2C para acceder al bus I2C, además se han definido los siguientes métodos:
  - Método **getI2C,** para obtener la instancia al objeto I2C utilizado para la comunicación del bus.
  - Método **getSlaveAddress,** para obtener la dirección que utiliza el sensor esclavo.
  - Método virtual **readI2C,** permite leer datos del dispositivo esclavo conectado al bus I2C.
  - Método virtual **writeI2C,** permite enviar datos al dispositivo esclavo conectado al bus I2C.
- Clase **Sensors,** hereda de **list<SensorBase>** para la gestión de los dispositivos he implementa los siguientes métodos:
  - Método **addSensor,** permite añadir un sensor.
  - Método **initializeSensors,** inicializa los sensores registrados.
  - Método **readSensors,** lee todos los sensores registrados y almacena los datos en el buffer.
  - Método **sleepSensors,** pone los sensores en modo ahorro de energía.
  - Método **wakeUpSensors,** pone los sensores en marcha.

- Estructura **GadgetConfig,** tiene como parámetro el tiempo que el Gadget va a estar dormido.
- Clase **Gadget,** hereda de las clases **GadgetBase**, **Sensors** y **SampleBuffer.** Esta clase a la hora de crearla se le pasa la estructura **GadgetConfig** para especificar el tiempo de dormir el dispositivo, además sobrescribe y define nuevos métodos que comentaremos a continuación, para añadir nuevas funcionalidades y la gestión de los sensores:
    - Método **addSensor,** para añadir sensores al Gadget.
    - Método **getConfig,** para obtener la configuración del Gadget.
    - Se ha sobrescrito el método **firstTime** para la inicialización de los sensores.
    - Se ha sobrescrito el método **beforeStep** que leer los sensores y una vez leídos los pone en modo ahorro de energía.
    - Se ha sobrescrito el método **afterStep** para visualizar los datos leídos.
    - Se ha sobrescrito el método **afterDying** para despertar los sensores.

Una vez desarrolladas las clases anteriores se ha procedido a escribir el programa que se hizo en el Sprint 1 para que realice la misma funcionalidad, pero utilizando las siguientes clases:

- Clase **LedSensor** que hereda de la clase **SensorBase,** activa y desactiva una salida digital, sobrescribiendo los siguientes métodos:
    - Método **initialize,** que activa la salida digital.
    - Método **read,** que hace una parada del sistema de un segundo.
    - Método **sleep,** que desactiva la salida digital.
    - Método **wakeUp,** que activa la salida digital.
- Clase **TMP3X** que hereda de la clase **SensorBase,** permite leer una entrada analogía, añadiendo la funcionalidad del sensor de temperatura analógica de un chip TMP36 sobrescribiendo los siguientes métodos:
    - Método **initialize,** que es sobrescrito para poder utilizar la clase.
    - Método **read** que visualiza la temperatura.

- Clase **BME280** que hereda de la clase **SensorI2CBase**, cogiendo la librería obtenida en el Sprint 1, se ha reescrito para el funcionamiento con el Gadget, para ello se han sobrescrito los métodos necesarios y se han añadidos otros:
    - Se ha sobrescrito el método **initialize** para inicializar el sensor BMP280.
    - Método **getTemperature,** obtiene la temperatura del sensor en grados centígrados.
    - Método **getPressure,** obtiene la presión del sensor en pascales.
    - Método **getHumidity,** obtiene el porcentaje de humedad.
    - Método **getAltitude,** permite calcular la altura a nivel del mar según una presión.
    - Se ha sobrescrito el método **read** para la lectura de los factores del sensor, es decir la temperatura, presión y humedad.

En la tarea id 167 se han estudiado diferentes sistemas de codificación de los datos que recogen los sensores para luego enviar al Gateway. Una primera opción sería codificarlos en forma de cadena, identificando el sensor con su identificador y a continuación enviar los datos leídos por este. Para indicar el fin de la cadena utilitaria el carácter asterisco, una representación podría ser como la siguiente:

```
Sensor1,valor1-1,valor1-2,…,valor1-n;sensor2,valor2-1,valor2-2,…,valor2-n;…;*
```

La longitud de la cadena puede ser muy grande y hay que tener en cuenta que la comunicación por LoRaWAN está limitada a la cantidad de bytes que se pueden enviar en Europa, desde 51 bytes a 242 bytes por paquete, dependiendo de la señal. Si en el peor de los casos fueran 51 bytes, habría que romper la cadena en varios bloques.

Un ejemplo de unos valores leídos por el sensor bme280 serian, temperatura 21,290001, humedad 44,201172 y presión 992.769919 y para la lectura del sensor TMP3X con valor de temperatura de -22.087912 tendríamos la siguiente codificación:

```
118,21.290001,44.201172,992.760010;-2,-22.014650;*
```

La codificación tiene un tamaño de unos 50 bytes, que si añadimos más sensores ocuparía más. Como solo se utilizan caracteres de 0 al 9, además de los símbolos asterisco, punto y coma, coma, signo menos y punto, se podrían comprimir los datos, porque para representar estos caracteres solo harían falta 4 bits, por cada dos caracteres se podrían representar con un solo byte. Por ejemplo, la cadena “11” que en bytes es representada por los bytes 49,49 se representaría por un solo byte con valor 17. La cadena anterior se quedaría en un tamaño de 25 bytes, ocupando la mitad, también se podría tener en cuenta reducir la precisión de los datos, reduciendo la cantidad de decimales para así hacer la cadena de datos más pequeña, por ejemplo, en la temperatura poner solo 2 decimales.

Habría que tener en cuenta que la cadena debería tener una longitud par, si no es así se tendría que añadir un 0 al principio de la cadena o un asterisco al final de dicha cadena.

La otra codificación y por la que he optado es una codificación binaria como se puede ver en la tabla 15. Se debe tener claro cómo van a ser configurados los bytes, no es tan versátil como el sistema anterior, el servidor Backend tendrá que conocer dicha codificación para su interpretación y tiene que ser fija, la ventaja es que permite tener más precisión en los datos.

Por lo tanto, cuando el servidor Backend reciba el paquete interpretara que ha recibido datos del sensor bme280 y vendrá acompañado de tres números en punto flotante, con la representación de la temperatura, humedad y presión.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
bmp280		temperatura				humedad				presión				tmp3x		temperatura			
0xfe	0xff	0x01	0x1e	0xb0	0xc1	0x76	0x00	*0xec*	0x51	0xaa	0x41	0x00	0xce	0x30	0x42	0xa4	0x30	0x78	0x44
short		float				float				float				short		float			

Tabla 15 - Paquete de datos binario

En las clases creadas en la tarea anterior se han modificado los sensores para almacenar los datos en el buffer según la codificación binaria.

Para la tarea id 166, se ha procedido a enviar el buffer de datos al Gateway, para ellos se han implementado las siguientes clases:

- La estructura **GadgetDotConfig,** contiene los datos necesarios para la configuración de la conexión LoRaWAN y otros parámetros de funcionamiento.
- La clase **Dot,** está basada en la librería llamada dot_util obtenida de ejemplos de aplicaciones de MultTech (Arm MBED, 2019). A continuación, describimos algunos de los métodos implementados en esta clase:
    - Método **getConfigDot,** obtiene los parámetros de configuración.
    - Método **getDot,** obtiene la instancia del objeto mDot que corresponde con el dispositivo MultiTech mDot o xDot.
    - Método **initialize,** inicializa las configuraciones del dispositivo MultiTech mDot o xDot para establecer las comunicaciones con el Gateway y de qué forma las va a realizar, según se ha parametrizado la estructura GadgetDotConfig.
    - Método **sendSamples,** enviar los datos del buffer especificado al Gateway.
    - Método **sleep,** pone el dispositivo MultiTech mDot o xDot en ahorro de energía.
- La clase **GadgetDot** deriva de la clase Gadget, en esta clase se ha implementado las funcionalidades de la clase Dot y se han sobrescrito los siguientes métodos para implementar dichas funcionalidades:
    - Método **sleep,** permite poner el dispositivo en ahorro de energía.
    - Método **afterSteep,** se ha sobrescrito para enviar los datos almacenados en el buffer al Gateway.

Ahora tendríamos una aplicación que está recogiendo datos de los sensores y enviándolos al Gateway, por lo que se procede a la configuración del Gateway para la recepción de los paquetes enviados por el Gadget y la

programación de la aplicación en la parte del Gateway, en la ilustración 21 tenemos la salida del terminal de la aplicación Gadget.

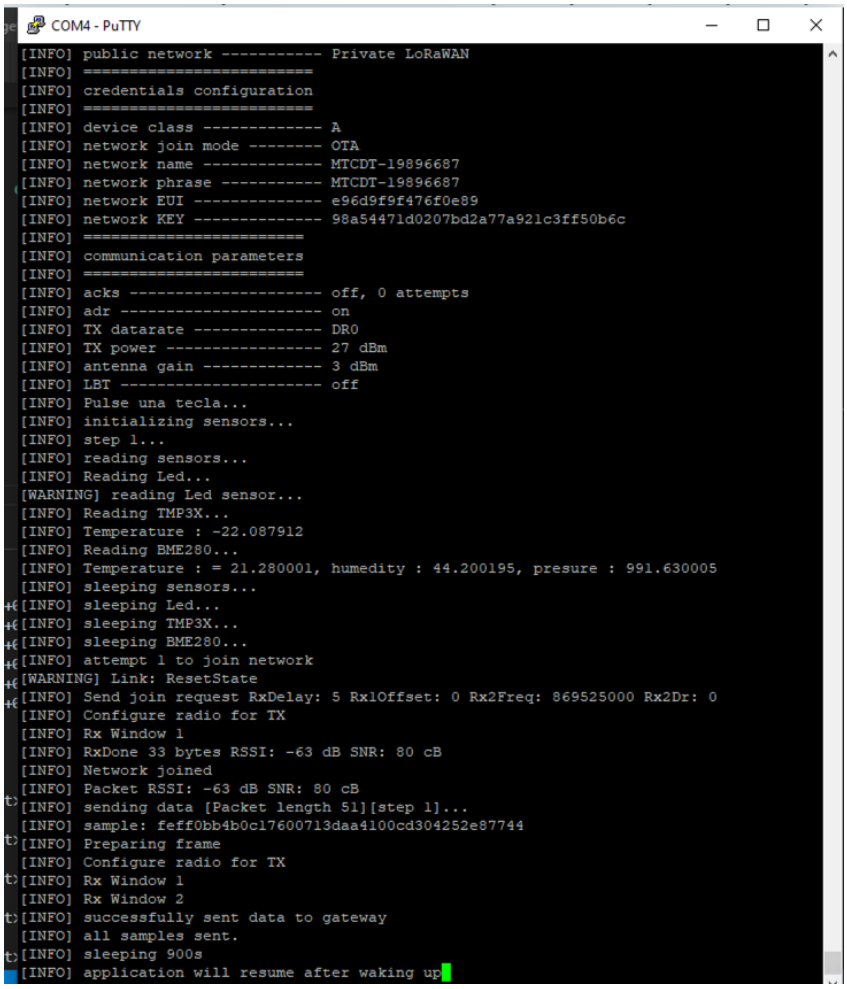

Ilustración 21 - Salida del Gadget

En la tarea id 17 se ha estudiado el entorno de desarrollo de Node-RED, estudiando la forma de escribir el código como entender el control de flujo de una aplicación, como crear aplicaciones de ejemplo. En la ilustración 22 se puede apreciar el entorno de desarrollo del Node-RED.

La siguiente tarea realizada ha sido la id 172, donde se ha tenido que configurar el Gateway para la recepción de los paquetes enviados por el Gadget y ver los datos recibidos.

Por último, para este Sprint se ha realizado la tarea id 174 que ha consistido en transformar los paquetes recibidos por el Gadget en formato JSON para su posterior envió al servidor Back-End, donde serán tratados. En la

ilustración 22 se puede ver la aplicación realizada donde puede verse como se han obtenidos los datos del Gadget y se han trasformado en JSON para su posterior envió al servidor Back-End.

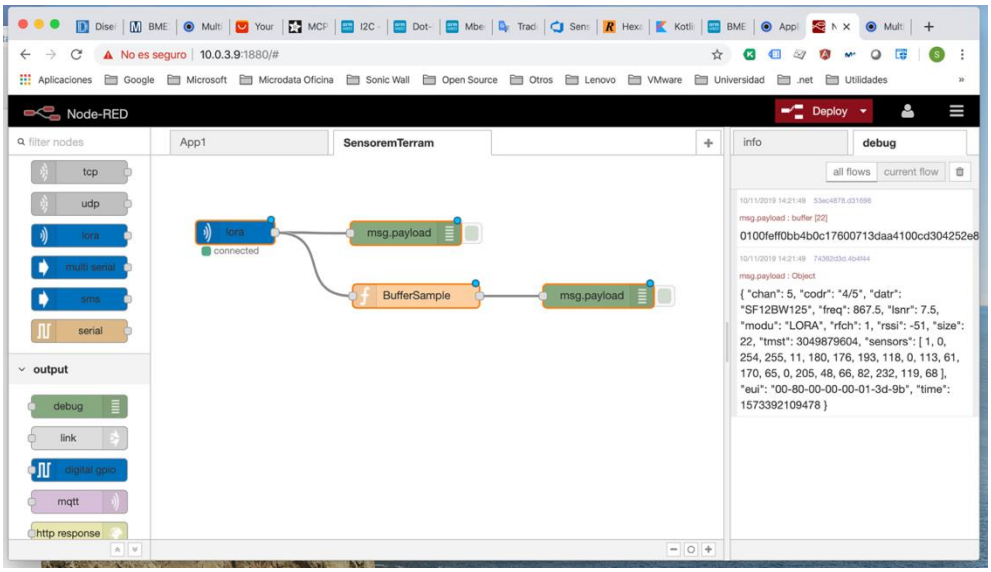

Ilustración 22 - Aplicación Node-RED con recepción de paquete

8.2.4. Revisión

Los resultados de este Sprint han sido los esperados, pues lo que se pretendía era enviar paquetes de los sensores al Gateway. Se han recibido los paquetes al Gateway una vez configurado este y transformado los datos a un formato JSON. No se ha quedado ninguna tarea pendiente para hacer en el siguiente Sprint.

8.2.5. Retrospectiva

En la tabla 16 se puede ver la comparativa entre los puntos de historia estimados con los reales para la realización del Sprint.

Tareas		Prioridad	Story Points	Realizado	Sprint	Story Points Reales
Id	Titulo					
168	Gestor de sensores	S	5	S	2	7
167	Codificación de datos	M	5	S	2	4
166	Envió de datos al Gateway	M	5	S	2	5
171	Estudiar entorno de desarrollo Node-RED	M	8	S	2	8
172	Recepción de paquetes del Gadget	M	3	S	2	3
174	Transformar paquetes Gadget a JSON	M	5	S	2	5
	Suma de puntos de historia		31			32

Tabla 16 - Comparación de los puntos de historia estimados y reales del Sprint 2

Se han producido diferencias respecto a los puntos de historia en la tarea id 168, pues se ha necesitado más horas para la programación, pruebas y testeo de dicha tarea.

Mientas en la tarea id 167 se han necesitados menos horas para el desarrollo que las indicadas en la estimación. Para el resto de las tareas se han realizado en la estimación definida.

En la ilustración 23 se puede observar una gráfica Brundown donde se puede ver la evolución del desarrollo del Sprint 2.

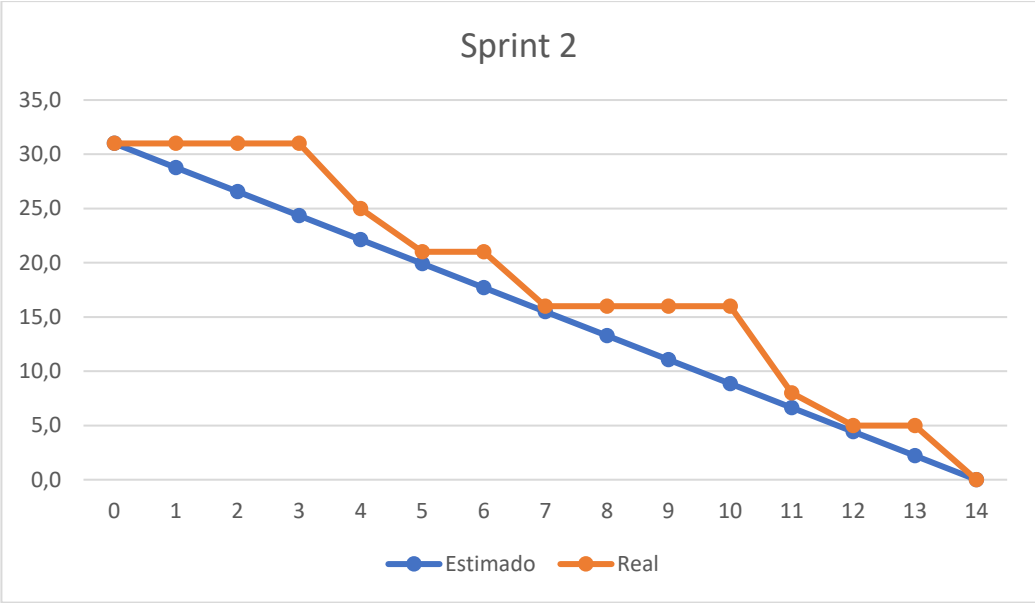

Ilustración 23 - Gráfico Brundown Sprint 2

8.3. Sprint 3

8.3.1. Planificación

En este tercer Sprint se tendrá que acometer 32 puntos de historia donde en la tabla 17 se han identificado las tareas a realizar en este Sprint, además de su prioridad y la estimación. Estas tareas se realizarán en el orden indicado como se puede observar en dicha tabla.

Tareas		Prioridad	Story Points	Realizado	Sprint
Id	Titulo				
203	Implementar servidor de aplicación en Azure	M	0,5	N	3
183	Controlador recepción datos Gadget	M	2	N	3
175	Enviar datos al servidor Back-End	M	8	N	3
180	Obtener datos del estado del Gateway	S	5	N	3
176	Lectura de posicionamiento del Gateway	C	3	N	3
184	Controlador recepción estado Gateway	C	2	N	3
182	Enviar estado Gateway en formato JSON al Back-End	S	2	N	3
205	Controlador recepción localización Gateway	S	2	N	3
177	Envió de datos de posicionamiento al Back-End	C	2	N	3
202	Crear base de datos en Azure	M	0,5	N	3
182	Modelo de datos	M	5	N	3
	Suma de puntos de historia		32		

Tabla 17 - Planificación Sprint 3

8.3.2. Metas

Una de las metas de este tercer Sprint es comenzar el desarrollo del servidor Back-End, por lo que se empezará a implementar parte de este en Azure, se tendrán que desarrollar el controlador para recibir los datos de los Gadget como en el Gateway enviar los JSON al servidor Back-End.

Se tendrá que estudiar la posibilidad de obtener información de estado y localización del Gateway para su posterior envió al Back-End, como implementar en el servidor Back-End los controladores para la recepción de dichos datos en formato JSON.

Por último, se tendrá que implementar la persistencia de los datos mediante una base de datos Microsoft Azure SQL, que es donde el servidor Back-End almacenará los datos, como ver la estructura del modelo de datos que se va a utilizar dicho servidor.

Como resultado final, se debería tener gran parte de la implementación del servidor Back-End, donde este ya almacenaría los datos recibidos por los Gadgets y la información de los Gateways como su localización y estado almacenado de manera persistente para su posterior consulta por las aplicaciones Front-End.

8.3.3. Resultados

En el desarrollo de este Sprint, se han realizado todas las tareas indicadas relacionadas con las cuestiones de Envió de datos al Back-End, Estado del Gateway, Localización del Gateway y Recepción de datos del Gateway y persistencia de datos.

En la primera tarea, con id 203, se ha procedido a crear una Aplicación web ASP.Net Core con Visual Studio 2019, con la plantilla API, para la creación del Back-End con un servicio RESTful HTTP que es llamado **SensoremTerram.Backend**. Para la creación de un proyecto en Visual Studio 2019 se puede revisar el Anexo 10.1.3.

Una vez creada la aplicación se ha procedido a crear una App Services en Azure (Microsoft, 2019) para alojar nuestro servidor Back-End, y se ha publicado nuestro proyecto creado y se ha procedido a comprobar que funciona correctamente como se puede observar en la ilustración 24, que se obtiene un JSON con datos de ejemplo.

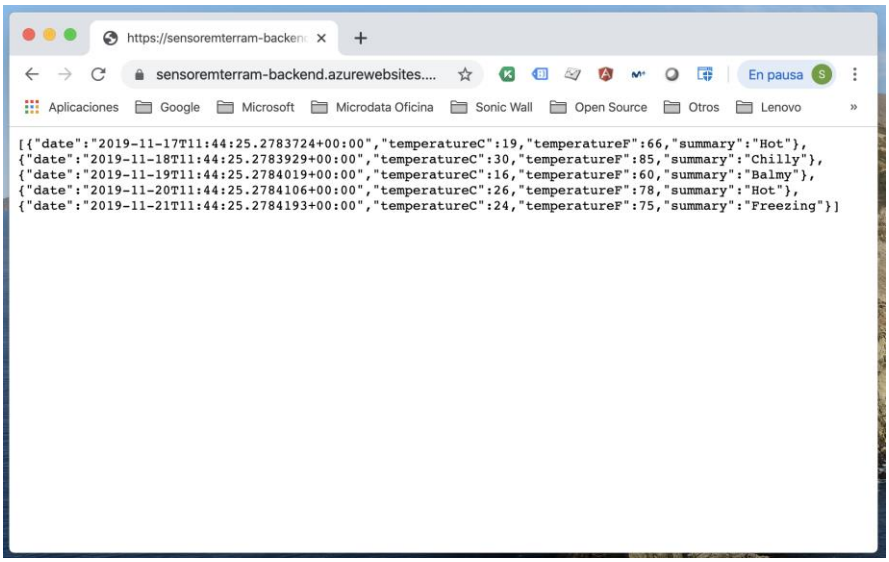

Ilustración 24 - Salida controlador ejemplo de nuestro Back-End

Una vez terminada la tarea anterior se ha procedido a desarrollar las tareas id 183 e id 175, pues esas tareas son complementarias. Se ha añadido nuevos proyectos a nuestro proyecto inicial, estos nuevos proyectos los enumeramos a continuación:

- **SensoremTerram.Models**, donde se definirán las clases de modelo de datos
- **SensoremTerram.DataAccess**, donde se definirán las interfaces para definir las funciones API.
- **SensoremTerram.DataAccess.Memory**, donde se implementarán las API para almacenar los datos en memoria volátil.

En nuestro proyecto vamos a utilizar inyección de dependencias definiendo las interfaces con las funciones API para la comunicación entre el Gateway y el servidor Back-End, después se implementan con su funcionalidad y podemos cambiar su comportamiento definiendo otras nuevas funciones. Ahora vamos a recibir los paquetes de los Gadget que los guardaremos en memoria, pero más adelante se tendrá que definir las entidades de la base de datos para almacenar de forma persistente los datos, permitiendo poder implementar los datos en el sistema de almacenamiento permanente que se desee utilizar.

También se ha utilizado un método de extensión (Microsoft, 2015) para vincular las interfaces con su implementación, mediante la clase estática **DataAccessExtensions**, que implementa el método de extensión **AddDataAccess**, donde se indican las implementaciones de dichas interfaces.

En el proyecto **SensoremTerram.Models** se ha creado la clase **GadgetPacket** con todas las propiedades que nos enviara el Gateway, que fueron definidas en el Sprint anterior.

En el proyecto **SensoremTerram.DataAccess** se ha creado una interfaz llamada **IGadgetPacketRepository** donde se han definidos dos métodos: el método **Process** para almacenar el paquete del Gadget y el método **GetPackets** para obtener todos los métodos recibidos.

A continuación, se ha procedido a crear la clase **GadgetPacketRepository** que hereda de **IGadgetPacketRepositiory** e implementar los métodos anteriores definidos.

En el proyecto **SensoremTerram.Backend** se ha creado un nuevo controlador llamada **GadgetController** y se definen dos métodos para la función POST y GET del REST API para añadir los paquetes de los Gadgets y poder consultarlos respectivamente, implementando la interfaz **IGadgetPacketRepository**. Para ver el correcto funcionamiento se ha utilizado Postman enviando datos al servidor Back-End con el método POST y después se ha realizado un GET para consultar los datos enviados. En la ilustración 25 se puede la prueba de datos enviados.

A continuación, se ha procedido a implementar el envío de Paquetes de los Gadgets en el Gateway y se procedido a ver que se envían correctamente dichos datos, pudiendo comprobar con Postman que se reciben correctamente.

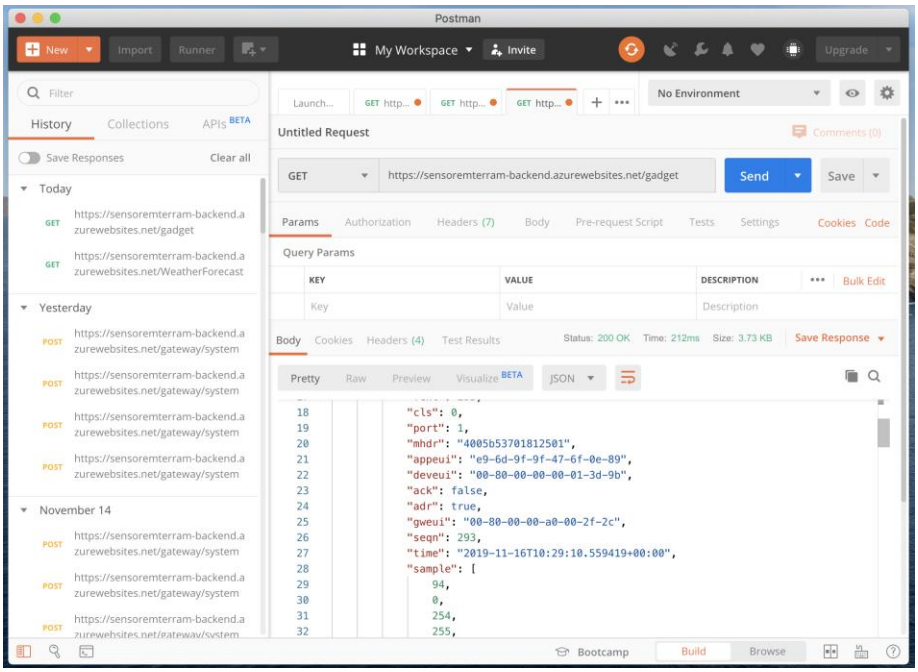

Ilustración 25 – Método GET del controlador de los Gadget

Para la comunicación del API de nuestro Back-End, se está utilizando la librería **Pitasoft.Result** que se puede obtener de NuGet, que implementa varias clases que utilizamos para obtener los resultados obtenidos de nuestro API del Back-End. La clase base **ResultBase** se definen unas propiedades con la información del resultado de la operación, que son las siguientes propiedades:

- Propiedad **status**, indica si ha tenido éxito la operación, si es correcto devuelve 1.
- Propiedad **code**, indica un código de error para aclarar el error que se ha producido.

En la tarea Id 180 se ha procedido a ver si había posibilidad de obtener información del sistema del Gateway, como un identificador para identificar al Gateway como identificar los datos que son enviados por este. Mediante la documentación de MultiTech he podido encontrar documentación del Conduit API Reference (MultiTech Developer Resources, 2019) que me permite interactuar tanto para obtener información como para poder configurarlo. Con la realización de esta tarea también se ha completado la tarea Id 176, pues también es posible obtener la localización del Gateway.

En Node-RED se ha programado la llamada del API del Conduit, empaquetando la información en JSON y se envía al Back-End, además en la información obtenida del sistema se ha capturado un campo llamado **gatewayId** que es utilizado para identificar los datos enviados desde cada Gateway, pues se puede dar la posibilidad que varios Gateway puedan capturar los datos del mismo Gadget, de esta manera se identifica que Gateway ha enviado la información, además como los paquetes del Gadget son identificados en la estructura de datos por el StepId, podemos identificar los paquetes duplicados para un periodo de tiempo que son recibidos en el Back-End. Las tareas Id 176 e Id 177 también quedan realizadas. Además, se ha modificado la estructura del JSON creado en el Sprint anterior para añadir el identificador del Gateway y así como en los modelos de datos se ha añadido dicha información para su posterior tratamiento.

En la tarea Id 184, se han creado nuevas clases en un nuevo proyecto llamado **SensoremTerram.Models** añadiendo las siguientes clases:

- Clase **GatewaySystem**, esta clase devuelve información del Gateway, como que puertos tiene disponibles, sistema de comunicación, etc..
- Clase **AccessoryCard**, clase que contiene información sobre tarjetas que se puede conectar al Conduit.
- Clase **Memory**, clase que contiene información sobre los recursos de memoria del Conduit.
- Clase **Mem**, clase que contiene propiedades de memoria.
- Clase **Swap**, clase que contiene información de memoria de intercambio.
- Clase **Radio**, clase que contiene información sobre el modem.

En el proyecto **SensoremTerram.DataAccess** se ha añadido una nueva interfaz llamada **IGatewaySystemRepository** para la recepción de datos del Gateway mediante el método **System** y la consulta de dichos datos mediante el método **GetSystem**.

En el proyecto **SensotemTerram.DataAccess.Memory** se ha implementado la clase **GatewaySystemRepository** que implementa la interfaz **IGatewaySystemRepository** para almacenar los datos en memoria.

En el proyecto **SensotemTerram.Backend** se ha implementado el controlador **GatewayController** que implementa la interfaz **IGatewaySystemRepository** permitiendo capturar los datos enviados por el Gateway sobre la información de este, además de poder consultarlos.

Mediante Postman se ha realizado la llamada al servidor Back-End para enviar datos de información del sistema, como consultar los datos que han sido enviados por el Gateway, como se puede ver en la ilustración 26.

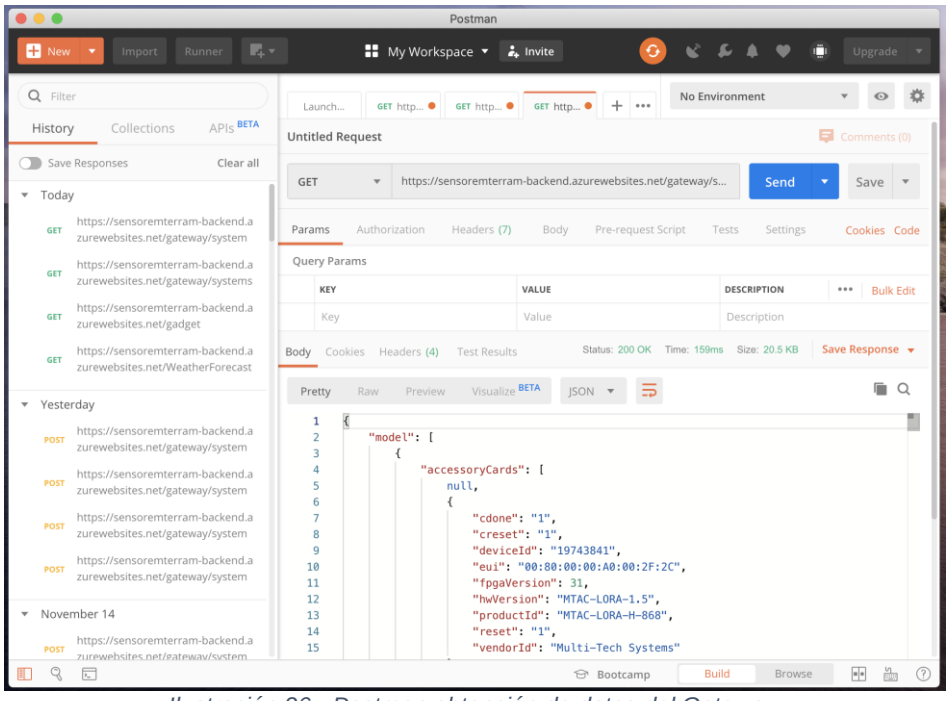

Ilustración 26 - Postman obtención de datos del Gateway.

En la tarea Id 205, en el proyecto **SensoremTerram.Models** se ha añadido la clase **GatewayStatsGPS**, clase que contiene la información de la localización del Gateway, ya que el dispositivo Conduit que se dispone para este proyecto tiene un módulo GPS.

En el proyecto **SensoremTerram.DataAccess** se ha añadido la interfaz **IGatewayStatsGPS** que contiene la definición del método **GPS** para la recepción

de los datos de la localización y el método **GetGPS** para obtener los datos almacenados.

En el proyecto **SensoremTerram.DataAccess.Memory** se ha creado la clase **GatewayStatsGPS** que hereda la interfaz **IGatewayStatsGPS** que almacena las localizaciones del Gateway y que se puedan consultar los datos enviados.

Mientras en el proyecto **SensoremTerram.Backend** en el controlador **GatewayController** se ha implementado la interfaz **IGatewayStatsGPS** para la captura de datos y la consulta, en la ilustración 27 se puede ver cómo es posible obtener las localizaciones de los Gateways.

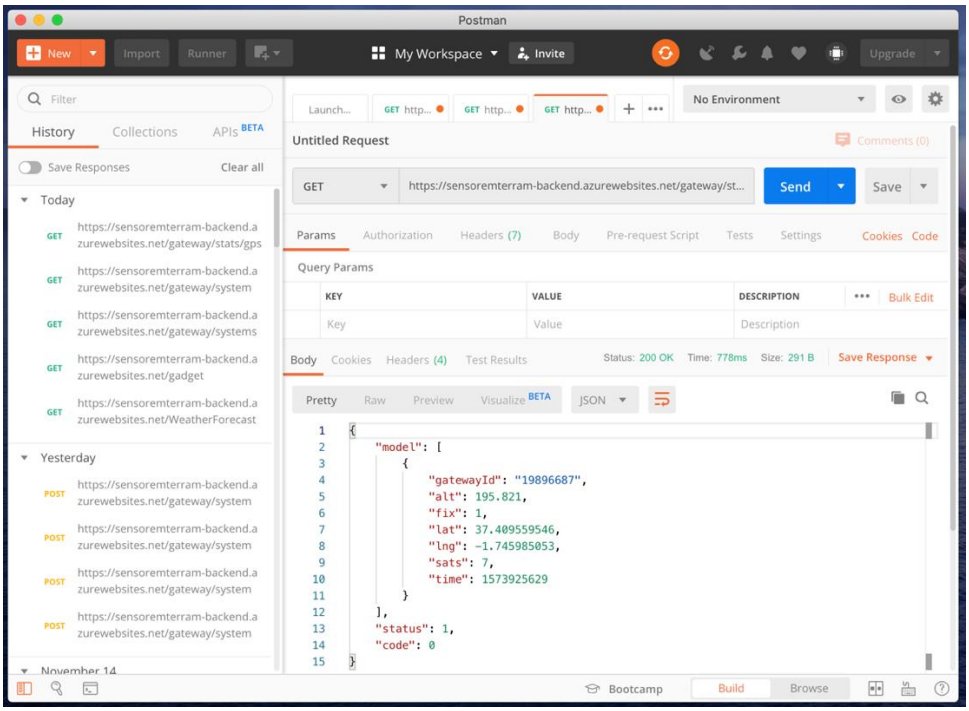

Ilustración 27 – Postman obtención de datos de localización del Gateway

A continuación, se procede a la creación de las tareas Id 182 e Id 202 para hacer los datos persistentes. Para la tarea Id 202, se ha creado la base de datos en Microsoft Azure SQL Database, en el siguiente enlace se puede ver la creación de una base de datos (Microsoft Azure, 2019).

Para la tarea Id 182 se ha creado el modelo de la base de datos, para ello se ha utilizado EF Core, se ha definido las siguientes clases que representaran

el modelo de nuestra base de datos dentro de un nuevo proyecto llamado **SensoremTerram.Entities**:

- Clase **DataLog**, almacena los datos recibidos por el Gateway en formato JSON.
- Clase **Unit**, define las unidades de medida.
- Clase **Gateway**, almacenar las propiedades necesarias para el Gateway.
- Clase **Cluster**, para agrupar los Gadget en grupos para calcular medias de Gadgets.
- Clase **Sensor**, define un sensor que puede utilizar un Gadget.
- Clase **SensorConfig**, almacenar la configuración de la estructura de datos recibida por el sensor.
- Clase **Hardware**, almacenar la configuración hardware de un Gadget, para definir un tipo de configuración Hardware.
- Clase **HardwarePost**, permite almacenar comentarios sobre el Hardware definido para poder hacer un seguimiento del funcionamiento de este.
- Clase **Sample**, define una muestra de dato con su unidad de medida.
- Clase **HardwareConfig**, almacena la configuración de los sensores definidos para un Hardware definido.
- Clase **Gadget**, almacenar las propiedades del Gadget, como el tipo de Hardware, identificador, nombre, localización y grupo.
- Clase **GadgetSample**, almacena las muestras recibidas por los Gadgets según la configuración del Hardware definido.
- Clase **SampleGroup**, donde se almacenarán los cálculos de las muestras recibidas por cada Gadget y Clúster, según el Hardware definido, para acelerar la consulta de los valores de los diferentes sensores definidos.

En el modelado de los datos se ha tenido que probar diferentes formas de almacenar los datos que son recibidos por el Gateway, pero al final se ha decidido guardar los datos recibidos por esté en formado JSON dentro de la base de datos, pues se cree que va a ser más flexible en el futuro, además de que Microsoft SQL Server permite trabajar con JSON directamente para el tratamiento de datos.

Se ha creado un nuevo proyecto llamado **SensoremTerram.DataAccess.SQLServer** para implementar las interfaces definidas anteriormente en el proyecto **SensoremTerram.DataAccess** para almacenar los datos en memoria persistente, que en nuestro caso es un servidor SQL Server, además de la definición del método de extensión **AddDataAccess** para la configuración de la llamada a las nuevas interfaces. También se ha creado la clase llamada **SensotemTerramContext** donde se define el Modelado de la base de datos relacional, como la relación entre clases de modelos, claves primarias, índices y tipos de datos de las columnas de las tablas, haciendo que los datos almacenados en la base de datos cumplan los requisitos establecidos. Como se ha comentado anteriormente, gracias a la inyección de dependencias se podría implementar la persistencia de datos en cualquier base de datos, como por ejemplo SQLite, Oracle Database, etc..

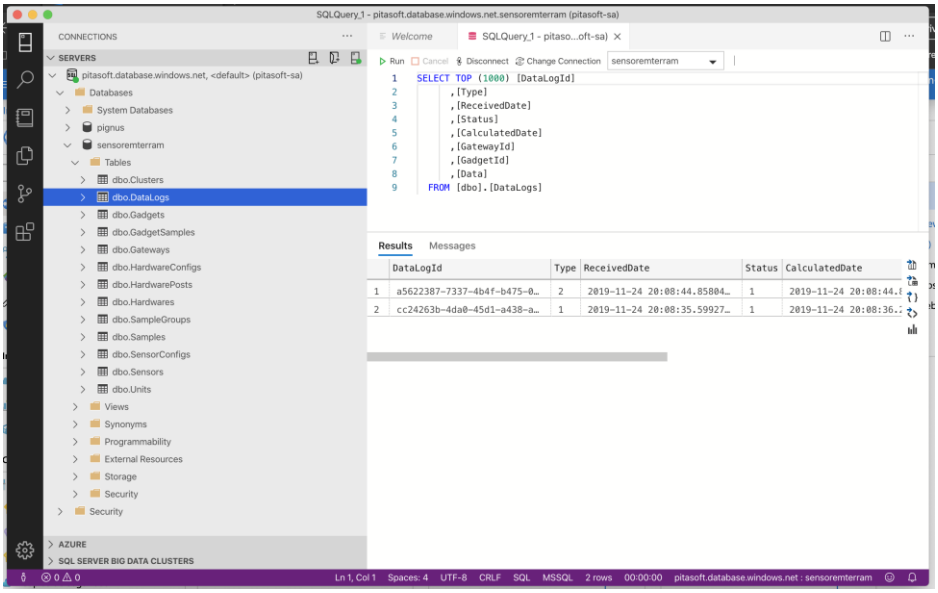

Ilustración 28 - Visualización de datos con Azure Data Studio

Como se puede ver en la ilustración 28, se puede ver la estructura de la base de datos y la consulta de la tabla **DataLogs** con los datos almacenados recibidos del Gateway.

8.3.4. Revisión

Los resultados de este Sprint han sido los establecidos inicialmente, que ha sido enviar datos desde el Gateway, tanto de los Gadgets como información

del Gateway al servidor Back-End y almacenar dichos datos de manera persistente, definiendo un modelado de la base de datos. No se ha quedado ninguna tarea pendiente por hacer, pero una vez terminado el Sprint y con el modelado de datos que se ha definido, hay que definir una nueva tarea que no se definió en la planificación inicial, dentro de la épica Back-End se define la siguiente tarea como se puede ver en la tabla 18.

Cuestiones		Tareas		Prioridad	Story Points	Realizado	Sprint
Id	Titulo	Id	Titulo				
211	Procesado de datos	222	Programador de tareas	M	3	N	

Tabla 18 - Nuevas tareas definidas en el Sprint 3 Revisión.

8.3.5. Retrospectiva

En la tabla 19 se puede ver la comparativa entre los puntos de historia estimados con los reales para la realización del Sprint.

Tareas		Prioridad	Story Points	Realizado	Sprint	Story Points Reales
Id	Titulo					
203	Implementar servidor de aplicación en Azure	M	0,5	S	3	0,5
183	Controlador recepción datos Gadget	M	2	S	3	2
175	Enviar datos al servidor Back-End	M	8	S	3	8
180	Obtener datos del estado del Gateway	S	5	S	3	5
176	Lectura de posicionamiento del Gateway	C	3	S	3	0
184	Controlador recepción estado Gateway	C	2	S	3	2
182	Enviar estado Gateway en formato JSON al Back-End	S	2	S	3	1
205	Controlador recepción localización Gateway	S	2	S	3	2
177	Envió de datos de posicionamiento al Back-End	C	2	S	3	1
202	Crear base de datos en Azure	M	0,5	S	3	0,5
182	Modelo de datos	M	5	S	3	10
	Suma de puntos de historia		32			32

Tabla 19 - Comparación de los puntos de historia estimados y reales del Sprint 3

Como se puede observar se han producido bastantes diferencias respecto a la estimación inicial con la real.

Cuando se realizó la tarea Id 180, se encontró documentación para obtener datos del Gateway, por lo que la tarea Id 176 también quedo solucionada, al mismo tiempo, eliminando puntos de historias.

También en las tareas Id 182 e Id 177 se han realizado en menos puntos de historia, pues se ha replicado la tarea id 175 y con la experiencia obtenida se ha conseguido realizar con menos puntos de historia.

Mientras en la tarea Id 182, se han necesitado más puntos de historia a la hora de crear el modelado de datos pues esta tarea ha sido más compleja que lo estimado inicialmente, se han probado diferentes modelados de datos para que la aplicación sea lo más flexible posible, por lo que se han tenido que dedicar más horas de las estimadas.

En la ilustración 29 se puede observar una gráfica Brundown donde se puede ver la evolución del desarrollo del Sprint 3.

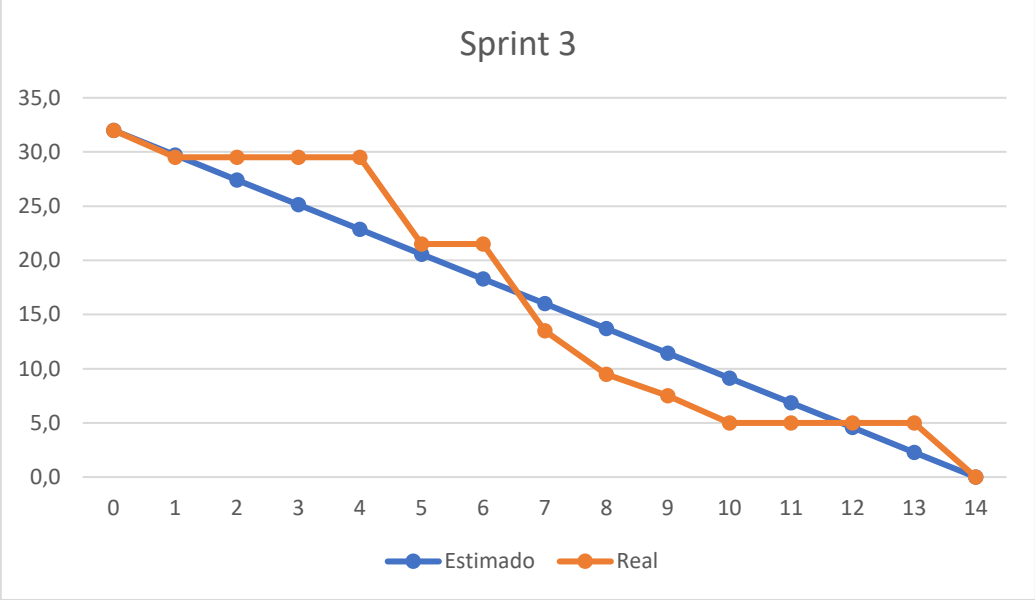

Ilustración 29 - Gráfico Brundown Sprint 3

8.4. Sprint 4

8.4.1. Planificación

En este cuarto Sprint se tendrá que acometer 36 puntos de historia donde en la tabla 30 se han identificado las tareas a realizar en este Sprint, además de su prioridad y la estimación, se ha acordado en la planificación añadir la tarea surgida en el Sprint 3, pues una vez estudiados los sensores e implementados habría que procesar dichos datos, esta tarea no estaba planificada, pero se ha acordado dedicar más puntos de historia a este Sprint para no alargar el proyecto. Estas tareas se van a realizar en el orden indicado en dicha tabla.

Tareas		Prioridad	Story Points	Realizado	Sprint
Id	Titulo				
218	Estudiar tipo de sensores del mercado	M	5	N	4
217	Implementar sensores	M	13	N	4
209	Estudio de suministro de energía	M	8	N	4
210	Implementación de la energía	M	3	N	4
216	Medición nivel de la energía	S	2	N	4
223	Procesar datos de los Gadgets	M	5	N	
	Suma de puntos de historia		36		

Ilustración 30 - Planificación Sprint 4

8.4.2. Metas

Las metas de este cuarto Sprint, es estudiar los sensores de Hardware Open Source que hay disponible en el mercado para después implementarlo en el Gadget.

También se comenzará a ver los diferentes tipos de batería que se pueden encontrar en el mercado y que se utilizará por el Gadget, además de poder medir el nivel de batería para conocer el consumo y ver la durabilidad.

Por último, una vez se implementen los sensores en el Gadget se procederá a procesar los datos recibidos en el servidor Back-End para más adelante poder consultarlos por las aplicaciones Front-End.

8.4.3. Resultados

En el desarrollo de este Sprint, se han realizado todas las tareas indicadas relacionadas con las cuestiones de Monitorización de factores, Energía y Procesado de datos.

En la tarea id 218, se ha procedido a estudiar diferentes sensores de Hardware Open Source disponibles en el mercado, probando algunos de ellos y comprobando su funcionamiento. Para este proyecto, en el desarrollo del Gadget se ha optado por utilizar los sensores que detallamos a continuación:

- BME280, este sensor permite medir la temperatura, humedad y presión atmosférica mediante el bus I2C (Mouser, 2019). Este sensor ha sido utilizado para medir los factores ambientales. Ver ilustración 31.

Ilustración 31 - Sensor BME280. Imagen obtenida de https://electronics.semaf.at

- MCP9808, este sensor permite medir la temperatura mediante el bus I2C (Adafruit, 2019). Este sensor se ha utilizado para medir la temperatura del dispositivo. Ver ilustración 32.

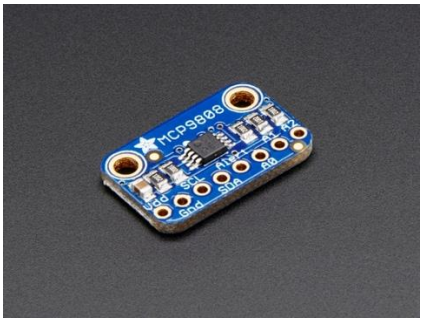

Ilustración 32 - Sensor MCP9808. Imagen obtenida de https://www.adafruit.com

- SHT20, este sensor permite medir la temperatura y humedad (Mouser, 2019). Este sensor se ha utilizado para medir la temperatura y humedad del suelo. Ver ilustración 33.

Ilustración 33 - Sensor FS200-SHT20. Imagen obtenida de https://www.amazon.es

- VEML6070, este es un sensor que permite medir la luz ultravioleta (UV) mediante el bus I2C (Mouser, 2019). Este sensor se ha utilizado para medir la luz ultravioleta. Ver ilustración 34.

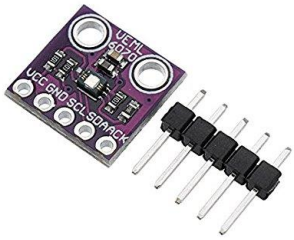

Ilustración 34 - Sensor GY-VEML6070. Imagen obtenida de https://www.faranux.com

- BH1750FVI, este sensor permite medir la luz ambiental con un rango de medición de 0 a 100.000 luz mediante el bus I2C (Mouser, 2019). Este sensor se ha utilizado para medir la luz ambiental. Ver ilustración 35.

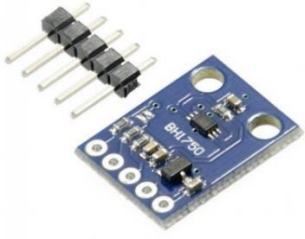

Ilustración 35 - Sensor BH1750FVI. Imagen obtenida de https://core-electronics.com.au

- HD-38, este es un sensor de humedad del suelo que permite la lectura analógica (Mactronica, 2019). Este sensor también se ha utilizado para la lectura de la humedad del suelo. Ver ilustración 36.

Ilustración 36 - Sensor HD-38. Imagen obtenida de https://www.mactronica.com.co

A continuación, se ha procedido a realizar la tarea Id 217, que consiste en implementar el código necesario para el proyecto del Gadget, se detalla a continuación:

- Para el sensor BME280 fue implementado en el Sprint 2.
- Para el sensor MCP9808, se creado el código del sensor basándose en el código obtenido del siguiente enlace (M, 2017). Se ha procedido a crear los archivos llamados MCP9808.h y MCP9808.cpp, que implementa la clase MCP9808 que hereda la clase base SensorI2CBase para poder medir la temperatura del Gadget, se han creado los siguientes métodos:
  - Método **getTemperature**, que permite obtener la temperatura.
  - Método **sleep** para poner en ahorro de energía el dispositivo.
  - Método **wakeUp** para poner en funcionamiento el dispositivo.
- Para el sensor SHT20, se ha creado su código basándolo en las librerías consultadas de los enlaces: (MtM+, 2017) y (DFRobot, 2019). Se han creado los ficheros SHT2X.h y SHT2X.cpp que implementa la clase SHT2X que hereda la clase base SensorI2CBase para poder medir la temperatura y humedad del suelo. En el código se ha implementado los siguientes métodos:
  - Método **getTemperature** para obtener la temperatura.
  - Método **getHumidity** para obtener la humedad.
  - Método **setPrecisión** para especificar la precisión del sensor.
  - Método **reset** para hacer un reinicio por software del sensor.

- Para el sensor VEML6070, a la hora de crear el código se han utilizado la librería del enlace (Roberts, 2017), que está basada en la librería del producto 2899 de Adafruit. Se ha procedido a crear los ficheros VEML6070.h y VEML6070.cpp, creando la clase VEML6070 que hereda la clase SensorI2CBase y nos va a permitir medir la radiación ultravioleta en el Gadget. Se han implementado los siguientes métodos:
    - Método **getUV** permite obtener la medición de UV.
    - Método **sleep** para poner en modo ahorro de energía el sensor.
    - Método **wakeUp** para poner en funcionamiento el sensor.
    - Método **configure** permite configurar la sensibilidad del sensor.
    - Método **setInterrupt** permite indicar los modos de interrupción.
- Para el sensor BH1750, hay bastantes librerías disponibles, pero a la hora de escribir el código se han utilizado los siguientes enlaces: (Arai, 2015), (Stehlik, 2014) y (Grzywacz, 2017). Basándonos en dichas librerías se han escrito los archivos BH1750.h y BH1750.cpp que implementa la clase BH1750 que hereda de SensorI2CBase e implementa la funcionalidad para medir los Lux para el Gadget. En esta clase se han implementado los siguientes métodos: **getLux** para obtener los Lux, **sleep** para poner el dispositivo en ahorro de energía, **wakeUp** para poner el dispositivo en modo funcionamiento, configure permite configurar el sensor con diferentes funciones y **setMeasurementTime** permite configurar el tiempo de muestreo.
- Para el sensor HD-38, he creado dos nuevos archivos llamados HD38.h y HD38.cpp que heredan de la clase base **SensorBase**. Este sensor hay que conectarlo a una entrada analógica para leer su valor en voltios y después transformar dicho valor a su correspondiente representación. Se ha implementado el método: **getHumidity** que permite obtener el porcentaje de humedad.

En la tarea Id 209, se ha procedido a estudiar los diferentes tipos de batería para implementar en el Gadget, aunque en el mercado hay variedad de tipo de baterías, como baterías de níquel hierro (NI-FE), baterías de níquel cadmio (NI-CD), baterías níquel hidruro (Ni-MH), baterías de iones de litio (LI-ION) y baterías de polímero de litio (LIPo). En este proyecto se ha optado por

utilizar baterías LI-ION de LG tipo 18650 de 3.6V de 3.000 mAh (Ver ilustración 37). Este tipo de batería hay que tener cuidado porque son muy potentes y no poseen un circuito de protección electrónico en el embalaje de la celda, existe el peligro de sobrecalentamiento, cortocircuito o descarga excesiva (Fenix, 2019).

Ilustración 37 - Batería LG INR18650-HG2 18650 3000maH 3.6V LI-ION Unprotected. Imagen obtenida de https://www.batterijeshop.com

Para solucionar este problema se ha utilizado un circuito de protección de BMS (Battery Management System) para las baterías LI-ION 18650, ver ilustración 38, que proporciona una electrónica de seguridad para evitar accidentes con el uso de las baterías de litio, siendo sus funciones esenciales (Energy EV, 2014):

- Desconectar o pagar la carga cuando la tensión de una celda de la batería cae por debajo de 2.5V.
- Apagar el proceso de carga cuando la tensión de una celda de la batería sube por encima de 4.2V.
- Apagar el sistema cada vez que la temperatura de una celda exceda los 50 grados centígrados.

Ilustración 38 - Módulo protector de baterías BMS. Imagen obtenida de https://laelectronica.com.gt

El dispositivo MultiTech mDot y la mayoría de la electrónica Hardware Open Source trabaja con un voltaje entre 3.3V y 5V, pero también hay electrónica que solo funciona a 3.3V o 5V. Además, como las baterías de litio tienen un voltaje 3,6V, pero el rango de funcionamiento de estas esta entre 4.25V cuando la batería está completamente cargada y 2.54V cuando esta descargada, es decir voltaje mínimo de descarga por seguridad.

Se ha optado por utilizar un voltaje de 3.3V, para ello se ha intentado regular la tensión de la batería con un regular de tensión fijo LM1117T-3.3 TC-220, ver ilustración 39, siguiendo las instrucciones del fabricante (Mouser, 2019), pero dependiendo del consumo y la resistencia del circuito, solo se ha alcanzado un voltaje de 3.1V, pero cuando la batería comienza a disminuir su voltaje, también comienza a disminuir la salida del voltaje, no obteniendo los resultados esperados, pues la batería no está en el consumo mínimo cuando el circuito deja de funcionar.

Ilustración 39 - Regulador Fijo LM1117T-3.3 3.3V 1A TO-220. Imagen obtenida de https://www.cetronic.es

Para resolver el problema se ha utilizado un regulador de tensión V7V8F3 del fabricante Pololu referencia 2122 (Pololu, 2019). Este es un regulador que permite disminuir o aumentar la tensión fijando su salida a 3.3V entre los voltajes comprendidos entre 2.7V y 11.8V, pudiendo suministrar entre 500mA a 1A. Así, se consigue que la electrónica funcione más tiempo, pues se puede aprovechar el uso de las baterías de litio que suministran un voltaje inferior a 3.3V, ver ilustración 40.

Ilustración 40 - Pololu 3.3V Step-Up/Step-Down Voltage Regulator S7V8F3. Imagen obtenida de https://www.pololu.com

Además, se ha utilizado una porta baterías para cuatro baterías de Litio 18650 conectadas en paralelo, como se puede ver en la ilustración 41, para la alimentación del Gadget.

Ilustración 41 - Porta baterías para 4 baterías en paralelo, foto obtenida del https://mercadolibre.com.mx

Con todo esto también se ha resuelto la tarea Id 210, que la implementación de la batería, en nuestro Gadget, como se puede ver en la ilustración 42.

Ilustración 42 Gadget autónomo conectado a batería

Para la Medición del nivel de la batería, que consiste en la tarea Id 216, lo realizamos mediante un divisor de tensión, utilizando resistencias de 100k ohmios, como la batería suministra un voltaje superior a 3V y la entrada analógica del MultiTech mDot solo puede leer voltajes de hasta 3V, con el divisor de tensión resolvemos el problema, así se puede leer el voltaje suministrado por la batería. Se ha implementado una nueva clase llamada **Battery** que deriva de **SensorBase**, creando los siguientes archivos Battery.h y Battery.cpp. Esta clase va a permitir leer el voltaje de la batería conectando una entrada analogía a la salida del divisor de tensión. Se ha creado el método **getVolts** que devuelve los voltios leídos.

En la ilustración 43 se puede observar el diagrama electrónico utilizado para desarrolla el Gadget hasta la tarea Id 216.

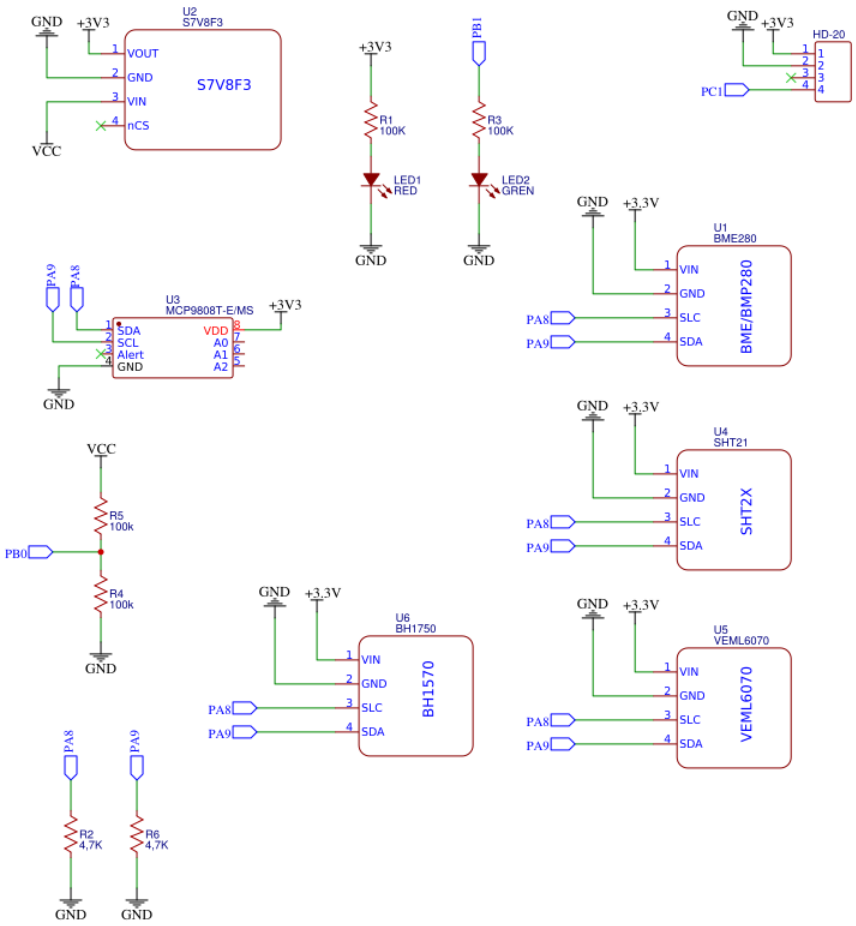

Ilustración 43 - Diagrama electrónico del Sprint 4

Por último, se ha realizado la tarea Id 223, que es el procesamiento de los datos paquetes recibidos por los gadgets. Para esta tarea se ha creado en el servidor Back-End un proceso en segundo plano que se ejecuta cada 5 minutos, que lo que hace es procesar los paquetes recibidos por los Gadgets. Se ha implementado un servicio con ámbito en una tarea en segundo plano como se puede ver en el enlace siguiente (Microsoft, 2019), se ha creado la interfaz **IScopeProcessingService** para implementar la ejecución de las tareas en segundo plano y la clase **ScopeProcessingService** que implementa dicha interfaz y los correspondientes métodos para ejecutar la tarea de procesamiento de los DataLog de los Gadgets. También se ha creado la clase **ConsumeScopeServiceHostedService** que es la que se encarga de ejecutar la tarea en segundo plano.

En la clase **SensoremTerramContext** se ha añadido un método llamado Seed, que hace la inicialización de datos necesarios para poder procesar los paquetes, se ha definido un Hardware y todos los sensores. Además de los Samples que queremos controlar en los Gadgets.

Para procesar los paquetes se han creado las siguientes clases, se detallan a continuación:

- Clase **GadgetData**, esta clase permite cargar la configuración del Hardware que tiene definido, para procesar los DataLog recibidos según su configuración de Hardware, además almacena los DataLog y comprueba que estos tienen la secuencia correcta. Define los siguientes métodos:
    - Constructor **GadgetData** crear la instancia de la clase cogiendo los datos necesarios para procesar los DataLog.
    - Método **LoadHardwareAsync**, que carga la estructura con los datos que va a procesar y que va a recibir el paquete.
    - Método **AddDataLog**, guarda en memoria un DataLog para procesar.
    - Método **GetSensor**, permite obtener la configuración de un sensor.

- Clase **SensorData** que define la configuración de un sensor con sus factores como temperatura, humedad, etc.. Estos son utilizados para cargar los valores de los paquetes recibidos. Define los siguientes métodos:
    - Constructor **SensorData** que crea la instancia con el identificador del sensor.
    - Método **LoadSensorAsync** que se encarga de cargar los factores en memoria para más adelante procesarlos.
    - Método **AddFactor** identificar el factor que se quiere guardar en el sistema.
    - Método **Read** permite transformar un paquete del Gadget recibido en los datos, guardando los valores de los factores en la clase **SampleData**.
- Clase **FactorData** define los factores de un sensor, contiene las propiedades del orden del factor, el tipo de dato que se tiene que procesar, un factor para aplicar al dato, una constante a aplicar al dato cargado, la unidad de media del factor y el significado del factor, es decir el Sample. Define un constructor para cargar los datos necesarios.
- La clase **StepData** es la clase que agrupa los DataLog de un Step que envía el Gadget. Cuando el paquete que tiene que enviar el Gadget al Gateway es demasiado grande para su envió, este paquete es roto en varias partes por el límite de datos que se pueden enviar por LoRaWAN, en esta clase se almacenan todas las partes de paquete y cuando esta recibe todos los paquetes si son correctos permite procesar los datos. Define los siguientes métodos:
    - Constructor **StepData** para crear la estructura de datos necesarios para la clase.
    - Método **AddDataLog**, que almacena los DataLog para procesarlos.
    - Método **Read** carga los valores de los sensores en la estructura de datos SampleData, que se utilizara para guardar en la base de datos.
- Clase **SampleData** permite guardar los valores de los factores leídos de un sensor.

Con la definición de estas clases se procesan los paquetes recibidos de los Gadget cada 5 minutos en un proceso en segundo plano, para que la respuesta del servidor Back-End sea lo más rápida posible a la hora de procesar los datos recibidos por el Gateways. También se puede dar el caso que un mismo paquete de un Gadget, puede ser recogido por varios Gateways, también se ha implementado que cuando se reciban los datos por el servidor Back-End este elimina los paquetes duplicados recibidos por varios Gateways.

En la ilustración 44 se puede ver la consulta de datos procesados por el servidor Back-End.

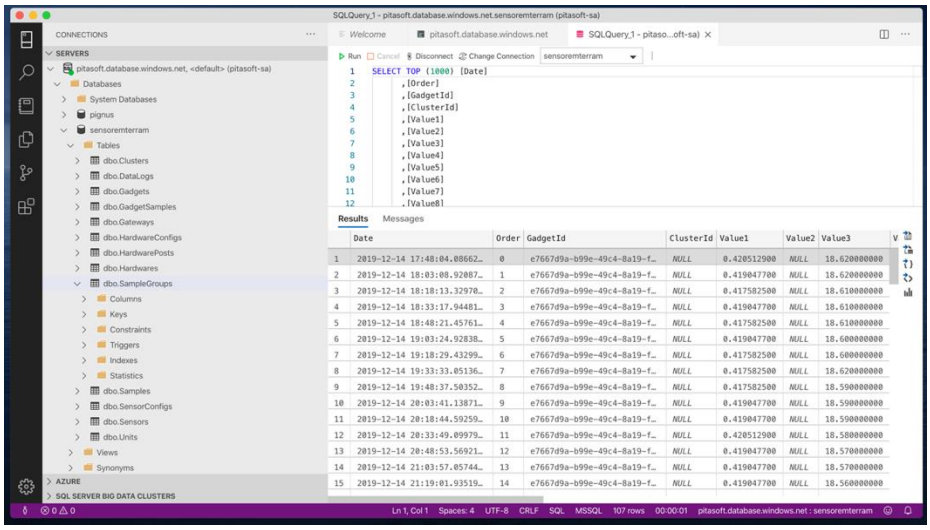

Ilustración 44 - Consulta de datos procesados por el servidor Back-End

8.4.4. Revisión

Los resultados de este Sprint han sido los esperados, pues se pretendía estudiar diferentes sensores para implementar en el Gadget, estudiando varios tipos de sensores y eligiendo algunos de ellos. Se ha implementado las baterías que suministran la energía al Gadget, además de poder medir el nivel de la batería. También se ha implementado una tarea en segundo plano que se ejecuta en el servidor Back-End que procesa los paquetes recibido, guardando los datos en una estructura más fácil para trabajar con dichos datos. No se ha quedado ninguna tarea pendiente para hacer en el siguiente Sprint.

8.4.5. Retrospectiva

En la tabla 20 se puede ver la comparativa de los puntos de historia estimados con los reales para la realización del Sprint.

Tareas		Prioridad	Story Points	Realizado	Sprint	Story Points Reales
Id	Titulo					
218	Estudiar tipo de sensores del mercado	M	5	S	4	4
217	Implementar sensores	M	13	S	4	13
209	Estudio de suministro de energía	M	8	S	4	7
210	Implementación de la energía	M	3	S	4	4
216	Medición nivel de la energía	S	2	S	4	2
223	Procesar datos de los Gadgets	M	5	S		10
	Suma de puntos de historia		36			40

Tabla 20 - Comparación de los puntos de historia estimados y reales del Sprint 4

Como se puede observar se han producido alguna diferencia respecto a la estimación inicial con la real, además de dedicar más horas para la realización de este Sprint y esperar a recibir el hardware necesario para la realización del Gadget, por lo que se han realizado varias tareas a la vez.

Respecto a las tareas Id 218, 209 y 216 se han realizado en menos tiempo que el estimado, se han visto diferentes sensores de Hardware Open Source, una vez que se ha tenido claro el tipo de batería y como medir su nivel, se ha pedido el material a la espera de poder realizar las siguientes tareas.

La tarea Id 223 ha necesitado más tiempo del previsto, pues la parte de software a desarrollar ha sido más compleja que la esperada. Además, se ha tenido que inicializar algunos datos de la base de datos para poder procesar dichos datos.

En la tarea Id 210, que ha sido la implementación de la energía, como se ha decidido utilizar un voltaje de 3,3V, se ha intentado mantener constante el voltaje estable con un regulador de tensión, pero este no ha funcionado bien pues cuando se tenía el voltaje estabilizado a 3,3V y la batería bajaba el voltaje

aunque la batería tenía un voltaje superior a 3.3V, el regulador de tensión daba un voltaje inferior y la electrónica dejaba de funcionar. Se ha tenido que dedicar más tiempo para resolver el problema hasta que se ha encontrado el Hardware adecuado, que proporcionase la potencia adecuada.

En la ilustración 45 se puede observar una gráfica Brundown donde se puede ver la evolución del desarrollo del Sprint 4.

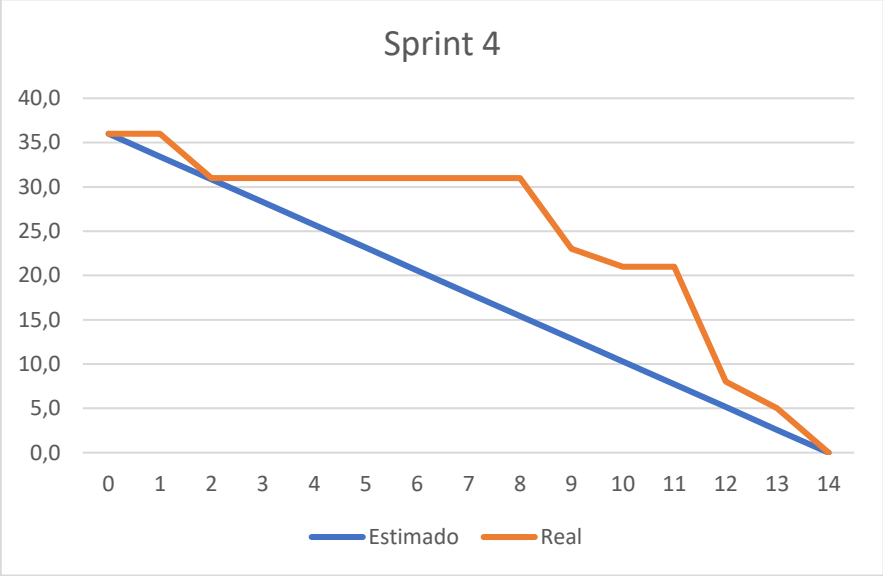

Ilustración 45 - Gráfico Brundown Sprint 4

8.5. Sprint 5

8.5.1. Planificación

En este quinto Sprint se tendrá que acometer 30,5 puntos de historia donde en la tabla 21 se han identificado las tareas a realizar en este Sprint, además de su prioridad y la estimación.

Tareas		Prioridad	Story Points	Realizado	Sprint
Id	Titulo				
185	Controlador acceso datos Gadgets	M	3	N	5
186	Controlador acceso datos Gateways	S	3	N	5
207	Crear base aplicación IU	M	8	N	5
213	Cuadro de mandos	M	5	N	5
204	Implementar servidor de aplicación en Azure	M	0,5	N	5
189	Consultar datos Gadgets	M	5	N	5
191	Consultar históricos Gadgets	M	3	N	5
192	Exportar datos a Excel	M	3	N	5
	Suma de puntos de historia		30,5		

Tabla 21 - Planificación Sprint 5

8.5.2. Metas

Las metas de este quinto Sprint, pretende crear la aplicación Front-End que permita mediante un navegador web, poder consultar los últimos datos obtenidos de los Gadgets, el histórico de datos de los Gadgets y poder exportar dichos datos a un archivo Excel.

También se tendrá que implementar en el servidor Back-End los controladores para que la aplicación Front-End pueda obtener dichos datos.

8.5.3. Resultados

En el desarrollo de este Sprint, se han realizado todas las tareas indicadas relacionadas con las cuestiones del Front-End para el navegador Web y la parte de comunicación con el servidor Back-End.

Para la tarea Id 207, se ha creado el proyecto para trabajar con Blazor una tecnología de Microsoft que está en fase preliminar que permite crear aplicaciones para navegadores web utilizando la tecnología WebAssembly, Wasm abreviado (WebAssembly, 2019) permitiendo poder escribir código C# en lugar de JavaScript en la parte cliente. Para la creación de esta tarea se han probado las diferentes formas de implementar un proyecto Blazor:

- Blazor en el lado del servidor, que está en versión reléase, donde todo el código se ejecuta en el servidor y el cliente no tiene carga, este sistema es más compatible con navegadores menos potentes y antiguos, pero tiene que utilizar Microsoft SignalR (Microsoft, 2019) para la comunicación entre el cliente y el servidor, esto también implica que hay que implementar en la nube un servidor de SignalR que puede ser gratuito para pocas conexiones o con un coste de unos 45 EUR/mes que permite un mayor número de conexiones. Este tipo de proyecto tiene las siguientes desventajas:
    - La interacción del cliente se realiza en el servidor.
    - Se necesita una comunicación continua.
- Blazor del lado del cliente, este proyecto genera el código binario Wasm que es descargado en el equipo local y es ejecutado en este. Este es apropiado para aplicaciones que no necesitan comunicación y además está limitado a que este tipo de proyecto no permite el Intercambio de recursos de origen cruzado o control de acceso HTTP (CORS) (MDN web docs, 2019), es decir, no se puede tener acceso a recursos con una URL diferente entre el cliente y el servidor, además los navegadores no permiten este tipo de comunicación por razones de seguridad. También actualmente este tipo de proyecto no permite el uso de CORS en la versión preliminar de Blazor.
- Blazor hospedado, utiliza dos proyectos, un cliente con tecnología Wasm, y un servidor Back-End que permite la comunicación con la aplicación cliente pues se encuentra en la misma dirección el cliente y el servidor.

Para el desarrollo de este proyecto se ha decidido utilizar el tipo Blazor hospedado por lo que en el proyecto actual tenemos dos nuevos proyectos:

- **SensoremTerram.Server,** este proyecto contiene el servidor Back-End y sustituye al proyecto **SensoremTerram.Backend** y en este proyecto se han implementado las funciones programadas en el Back-End.
- **SensoremTerram.Client**, este proyecto programará toda la parte del cliente. Se han incluido todas las librerías necesarias para la creación del proyecto. En este proyecto vamos a utilizar las librerías de *Syncfusion* (Syncfusion Inc., 2019) que permiten hacer una mejor experiencia con la interfaz de usuario, como visualizar gráficos, tablas de datos, etc.. Además, proporciona herramientas, como *Theme Studio for Essential JS2*, que permite la personalización de la visualización de los controles de la librería de Syncfusion (Syncfusion Inc., 2019).

Para la tarea Id 204 una vez creados los proyectos y configurados se ha procedido a crear una App Services en Azure (Microsoft, 2019) para alojar nuestro nuevo servidor Back-End y aplicación Front-End web en la nube. Se ha procedido a comprobar que funciona correctamente como se puede observar en la ilustración 46.

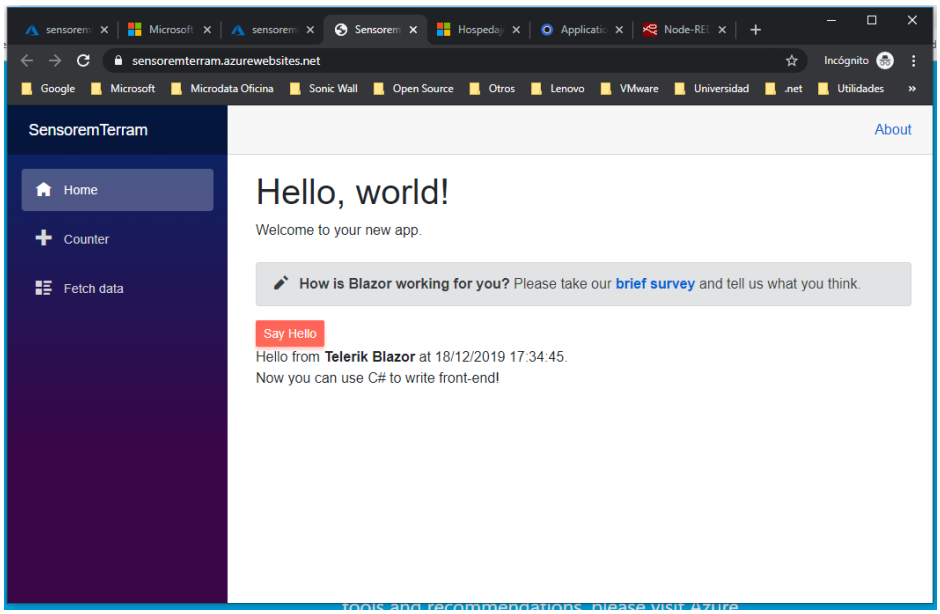

Ilustración 46 - Aplicación Front-End web

Para la realización de la tarea Id 185, Controlador acceso a datos Gadgets, se ha creado una nueva interfaz llamada **IGadgetRepository** en el proyecto **SensoremTerram.DataAccess**, donde se han implementado los

siguientes métodos para definirlas en su implementación para el acceso a la base de datos:

- **GetAsync**, para obtener la información de los Gadgets, permitiendo realizar consultas por el estado de los Gadgets o devuelve todos los Gadgets.
- **GetLasDataAsync**, para obtener información de los últimos datos disponibles por cada uno de los Gadgets.
- **GetRawDataAsync**, para obtener los datos en bruto de un Gadget en un intervalo de fecha.
- **GetSummaryAsync**, para obtener resumen diario de un Gadget y un Factor o Sample para un intervalo de fechas seleccionadas.

Después se ha procedido en el proyecto **SensoremTerram.DataAccess.SQLServer** la implementación de dicha interfaz, creando una nueva clase llamada **GadgetRepository** donde se implementan los métodos.

Y en el proyecto **SensoremTerram.Server** en el controlador **GadgetController** se han implementado las llamadas del API REST para los métodos anteriores.

También se han creado en el proyecto **SensoremTerram.Models**, las siguientes clases para el modelado de los datos:

- Clase **Gadget**, donde se ha implementado la clase que contiene las propiedades con la información del Gadget como su identificador, nombre, localización, estado y otros datos.
- Clase **GadgetCollection**, que devuelve una colección de los Gadget.
- Clase **RowSample**, define para una fecha los valores de los diferentes valores de los factores o samples medidos por los Gadgets.
- Clase **GadgetSample** que hereda de la clase **RowSample**, definiendo para una Gadget los valores de los factores.

- Clase **Header**, define las propiedades con los factores devueltos en un **RowSample**, con el identificador, nombre y formato del valor.
- Clase **GadgetLatestData**, que devuelve los últimos datos procesados de cada uno de los Gadgets, devolviendo una colección de **GadgetSample** con los valores de los factores medidos.
- Clase **DataAnalysis**, permite calcular datos estadísticos como la media, mediana, mínimo, máximo, rango, varianza, desviación estándar y coeficiente de variación, a partir de una colección de datos.
- Clase **GadgetRawData**, que devuelve los datos recibidos de un Gadget para un intervalo de fechas, devolviendo una colección de **RowSamples** con los factores del Gadget capturados por el intervalo de fechas, además de los identificadores de los factores recibidos y un conjunto de colecciones de datos con datos estadísticos como la media, mediana, mínimo, máximo, rango, desviación estándar y coeficiente de variación.
- Clase **RowSummary**, dispone de propiedades con valores estadísticos para una fecha, como mínimo, máximo, media, mediana, rango, varianza, desviación estándar y coeficiente de variación.
- Clase **GadgetSummary**, devuelve para un Gadget y Factor de medición una colección de **RowSummary** para un intervalo de fechas.

Para la tarea Id 186, Controlador acceso datos Gateway se ha creado una nueva interfaz llamada **IGatewayRepository** en el proyecto **SensoremTerram.DataAccess**, donde se ha definido el método **GetAsync**, que permite devolver todos los Gateways o según el estado actual de estos.

En el proyecto **SensoremTerram.DataAccess.SQLServer** se ha implementado dicha interfaz creando la clase **GatewayRepository**. Además, en el proyecto **SensoremTerram.Server** se ha definido en el controlador **GatewayController** la llamada del API REST para permitir realizar la consulta de estos datos.

También en el proyecto **SensoremTerram.Models** se han definido las siguientes clases para el modelado de datos devuelto por el método anterior, que son las siguientes:

- Clase **Gateway**, que devuelve la información de un Gateway, como su identificador, nombre, estado, localización y otros datos.
- Clase **GatewayCollection**, que devuelve una colección de **Gateway** con sus datos.

Mediante Postman se ha podido comprobar el funcionamiento de las llamadas a las funciones definidas en las tareas Id 185 y 186, donde se han implementado los controladores para la consulta de datos de los Gadgets y Gateways, como se puede ver en la ilustración 47.

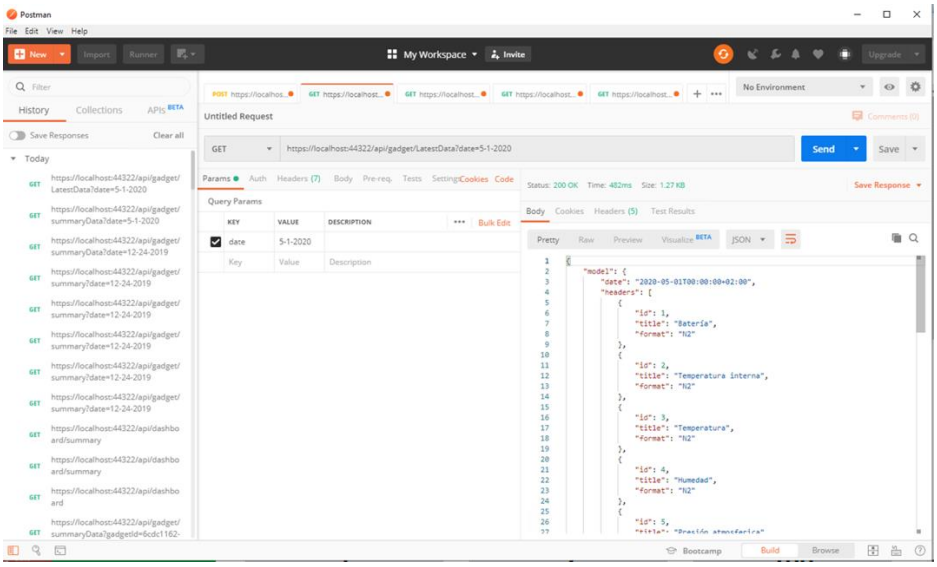

Ilustración 47 - Resultado de consulta de los controladores

En la tarea Id 213, lo que se pretende es disponer de un cuadro de mandos donde poder visualizar en un resumen el estado del sistema y ver que está funcionando correctamente. Para conseguir dicha tarea se ha creado una nueva interfaz llamada **IDashboardRepository** en el proyecto **SensoremTerram.DataAccess** donde se han definido los siguientes métodos:

- Método **GetSummary** devuelve la información con el estado del sistema.
- Método **GetDataLogSummaryAsync** devuelve la información resumen del estado de los paquetes recibidos por los Gateways.

Esta interfaz ha sido implementada en el proyecto **SensoremTerram.DataAccess.SQLServer** con la clase **DashboardRepository**. Para realizar la llamada al API REST para obtener los

datos se ha creado en el proyecto **SensoremTerram.Server** el controlador llamado **DashboardController** que permite realizar las llamadas a los dos métodos definidos anteriormente.

En el proyecto **SensoremTerram.Models** se han implementado las siguientes clases:

- Clase **DonutChartData** define las propiedades de visualización de un dato para un gráfico tipo tarta.
- Clase **BarCharData** define las propiedades para la visualización de un gráfico tipo barras.
- Clase **Dashboard** que contiene los datos de estado del sistema, como el número de Gadgets con diferentes estados, las conexiones de los Gadgets, el estado de los Gateways y las conexiones de estos.
- Clase **DataLogSummary** contiene los datos resumen de los paquetes recibidos de los Gateways, como el número de paquetes recibidos, paquetes procesados, paquetes pendientes de procesar y paquetes con errores.

A continuación, se ha implementado en el proyecto **SensoremTerram.Client**, la página **Index.razor** donde se ha implementado el cuadro de mandos y al ejecutar la aplicación obtenemos el resultado como se puede observar en la ilustración 48.

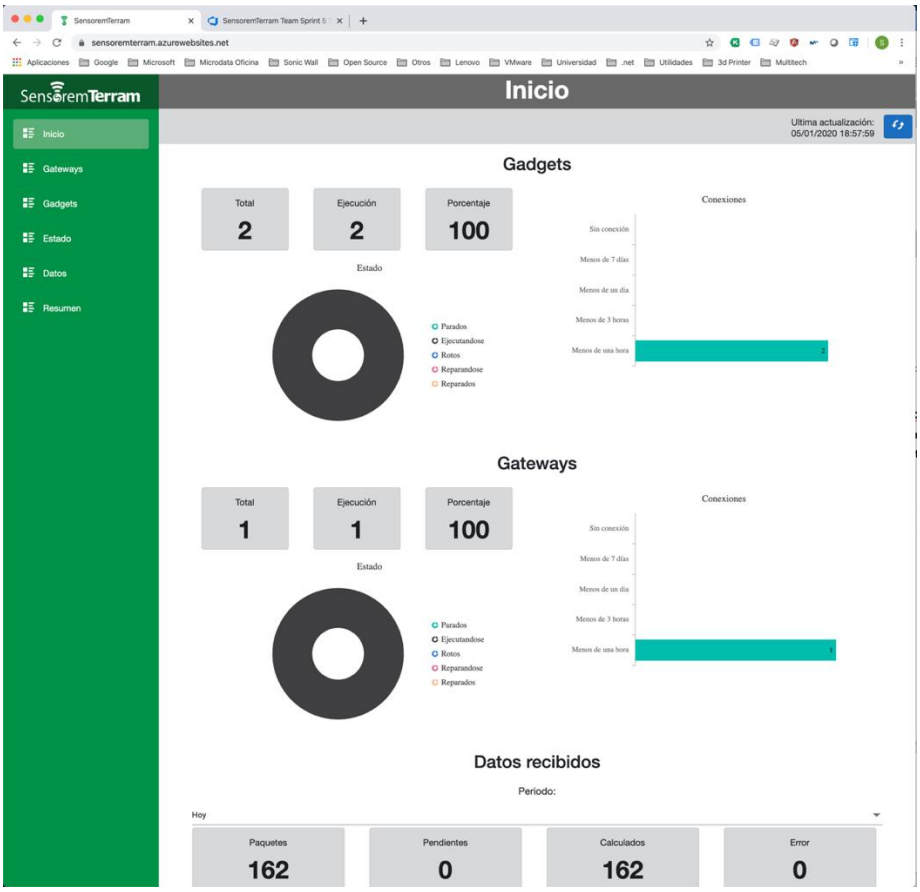

Ilustración 48 - Panel de mandos aplicación web.

Para la tarea Id 189, Consultar datos Gadgets, se ha creado en el proyecto **SensoremTerram.Client** una nueva página llamada **Gadgets.razor** la cual hace una llamada al controlador API REST para obtener los datos de los Gadgets disponibles que fue implementado en la tarea Id 185 y visualizar los datos de estos, como se puede ver en la ilustración 49.

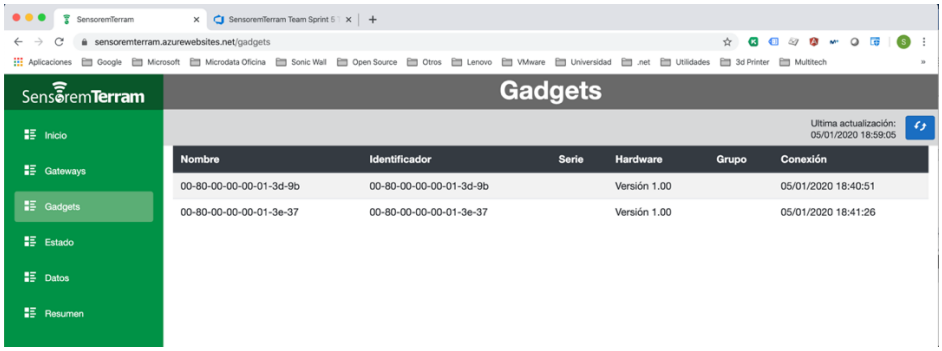

Ilustración 49 - Consulta de Gadgets en web

Para la tarea Id 191, Consulta de históricos de los Gadgets, se han creado en el proyecto **SensoremTerram.Client** tres nuevas páginas que describimos a continuación:

- **LatestData.razor**, se ha implementado la consulta de los últimos datos de los Gadgets, para ello se realiza una consulta al controlador implementado en la tarea Id 185, como se puede ver en la ilustración 50.

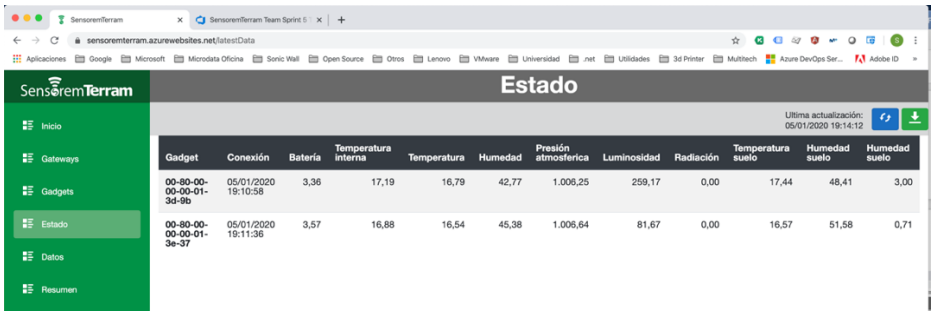

Ilustración 50 - Consulta del estado de los Gadgets en web

- **RawData.razor**, se ha implementado la consulta de datos en bruto de un Gadget, pudiendo seleccionar un Gadget mediante un desplegable y para un intervalo de fechas, obteniendo los datos de los factores con sus datos estadísticos, realizando la consulta al API REST implementado en la tarea Id 185, para la obtención de dichos datos, como se puede ver en la ilustración 51. Además, para la visualización del gráfico, en el proyecto **SensotemTerram.Models** se ha implementado la clase **LineChatData** para poder representar gráficos lineales.

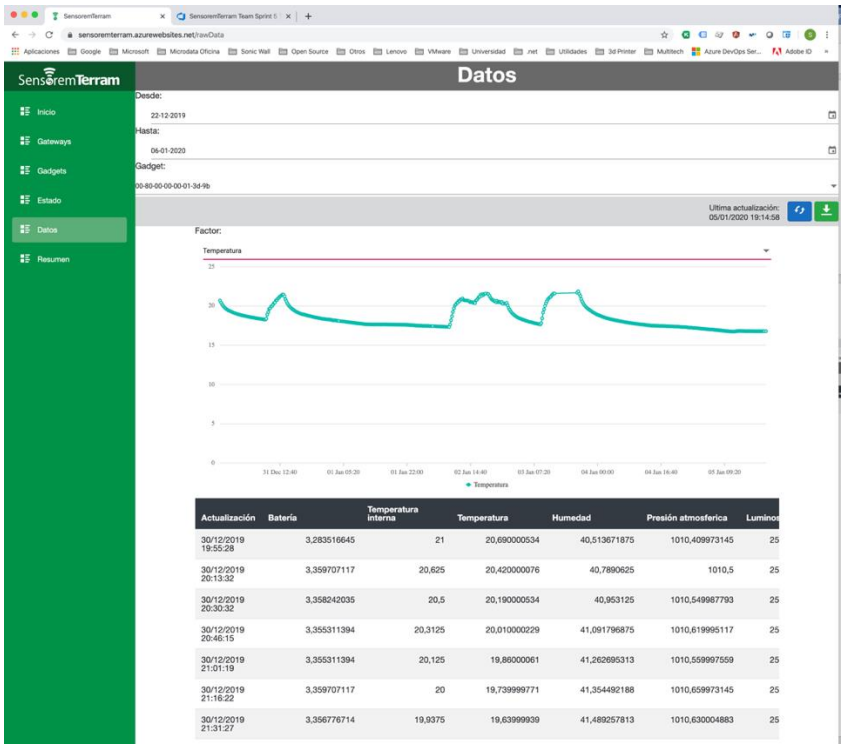

Ilustración 51- Consulta de datos en bruto de un Gadget en web

- **SummaryData.razor**, se ha implementado la consulta de datos para un Gadget y un factor para un intervalo de fechas, donde se obtiene por día los mínimos, máximos, media, mediana, rango, desviación estándar y coeficiente de variación, para obtener los datos se realiza una llamada a la API REST implementado en la tarea Id 185 como se puede ver en la ilustración 52.

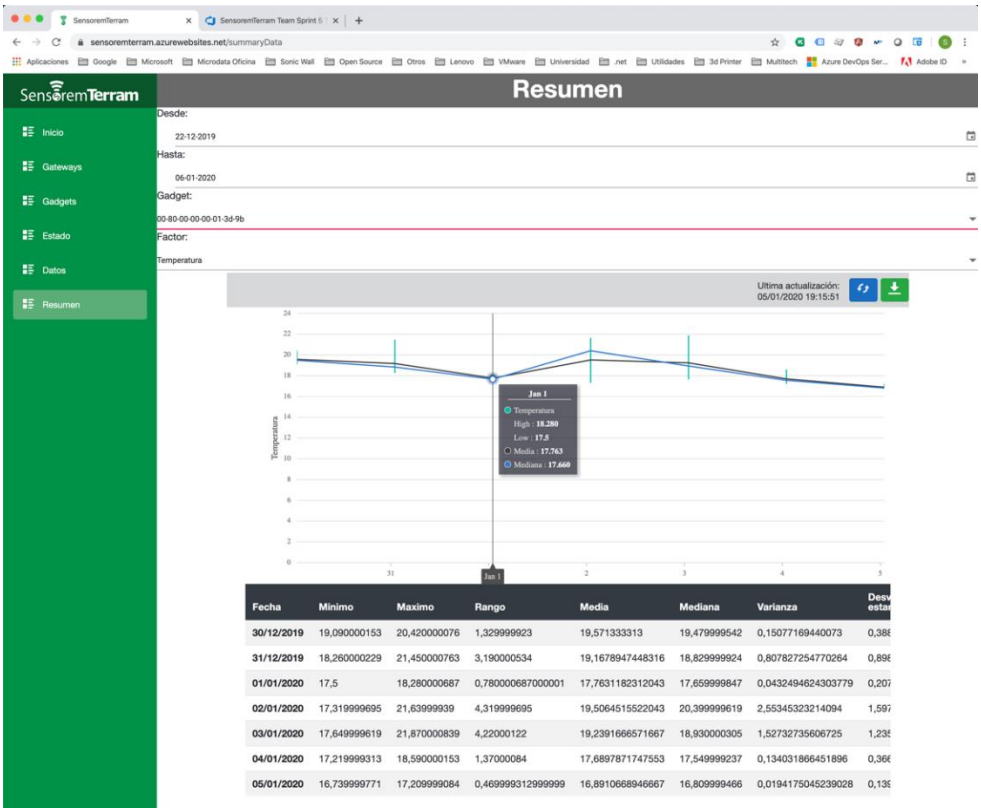

Ilustración 52 - Consulta de un Gadget y factor en web

Por último, para la tarea Id 192, Exportar datos a Excel, se ha implementado en el proyecto **SensoremTerram.Client** una clase método de extensión llamada **ModelUtils** donde se han implementado tres métodos, ambos llamados **ToExcel**, permitiendo generar los archivos Excel con los datos generados de las tres páginas de la consulta de los históricos de los Gadgets. Para generar los archivos Excel se han utilizado las librerías de Syncfusion XlsIO facilitando la generación de estos (Syncfusion Inc., 2019).

8.5.4. Revisión

Los resultados de este Sprint han sido los esperados, pues se pretendía crear la aplicación web utilizando Blazor donde:

- Poder consultar el estado del sistema mediante un panel de control, pudiendo con un solo vistazo poder visualizar dicho estado.
- Consultar los Gadget disponibles.
- Consultar los datos históricos de los Gadgets.
- Exportar los datos históricos de los Gadgets a un archivo Excel.

Para ello se ha implementado los controladores necesarios para que la aplicación cliente web puede obtener los datos mediante una API REST. No se ha quedado ninguna tarea pendiente para hacer en el siguiente Sprint, pero una ver revisado el Sprint se he decidido crear una nueva tarea para mejorar la interfaz de usuario, para hacerla más adaptable a diferentes pantallas y más intuitiva para visualizar como se puede ver en la tabla 22.

Cuestiones		Tareas		Prioridad	Story Points	Realizado	Sprint
Id	Titulo	Id	Titulo				
157	Aplicación Web	231	Mejorar interfaz de usuario	M	8	N	

Tabla 22 - tareas definidas en el Sprint 5 Revisión.

8.5.5. Retrospectiva

En la tabla 23 se puede ver la comparativa de los puntos de historia estimados con los reales para la realización del Sprint.

Tareas		Prioridad	Story Points	Realizado	Sprint	Story Points Reales
Id	Titulo					
185	Controlador acceso datos Gadgets	M	3	S	5	4
186	Controlador acceso datos Gateways	S	3	S	5	1
207	Crear base aplicación IU	M	8	S	5	10
213	Cuadro de mandos	M	5	S	5	7
204	Implementar servidor de aplicación en Azure	M	0,5	S	5	1
189	Consultar datos Gadgets	M	5	S	5	2
191	Consultar históricos Gadgets	M	3	S	5	6
192	Exportar datos a Excel	M	3	S	5	3
	Suma de puntos de historia		30,5			34

Tabla 23 - Comparación de los puntos de historia estimados y reales del Sprint 5

Como se puede observar se han producido diferencias respecto a la estimación inicial con la real por lo que se han necesitado más horas para poder finalizar dicho Sprint.

En la tarea Id 185, se han necesitado más horas para implementar el código, mientras que en la tarea Id 186 ha sido más sencillo la implementación del código.

Respecto a la tarea Id 207 se ha necesitado más tiempo pues se han tenido que estudiar los diferentes modos de implementación de Blazor y ver cual se adapta más a las necesidades de este proyecto.

Para la tarea Id 213 se han necesitado más horas pues se ha tenido que probar las librerías de Syncfusion para ver el funcionamiento y ver como implementarlas. Se han tenido problemas de configuración para poder utilizarlas, como problemas en la compilación del proyecto con Blazor, pues al ser una

tecnología preliminar, las herramientas no están optimizadas y producen comportamientos no esperados.

La tarea Id 204 se han necesitados más horas, pues al ser una tecnología nueva se ha tenido que probar en los servidores de Azure varios tipos de implementación, como un servidor web estático o un servidor de aplicaciones para ver cual funciona mejor.

En cambio, en la tarea Id 189 se ha necesitado menos tiempo pues ha sido muy fácil su implementación. Pero para la tarea Id 191 han sido necesarios más hora de programación para su implementación, con los problemas que nos ha surgido con la herramienta.

En la ilustración 53 se puede observar una gráfica Brundown donde se puede ver la evolución del desarrollo del Sprint 5.

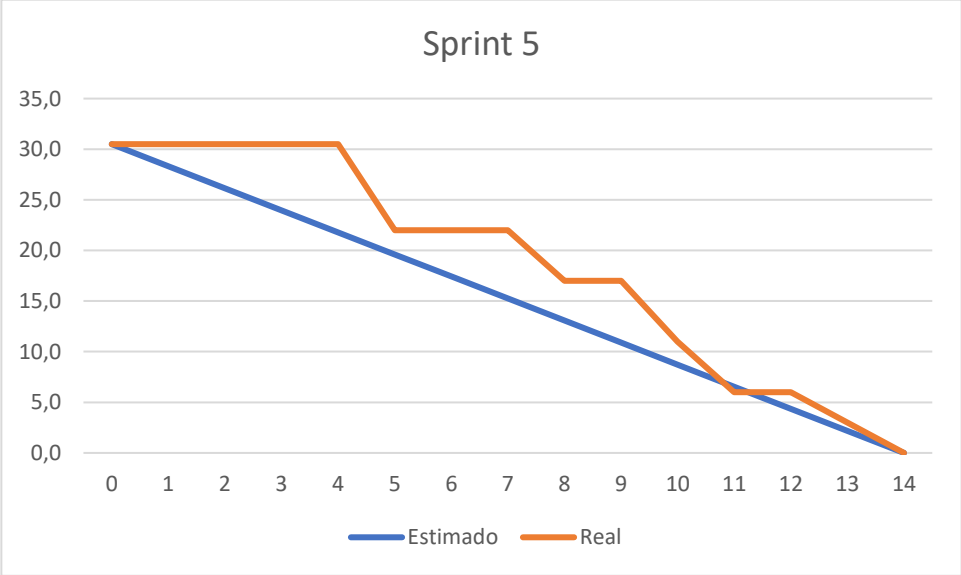

Ilustración 53 - Gráfico Brundown Sprint 5

8.6. Sprint 6

8.6.1. Planificación

En el sexto Sprint se tendrá que acometer 31 puntos de historia donde en la tabla 24 se han identificado las tareas a realizar en este Sprint, además de su prioridad y la estimación.

Tareas		Prioridad	Story Points	Realizado	Sprint
Id	Titulo				
206	Crear base aplicación IU	M	8	N	6
212	Cuadro de mandos	M	8	N	6
194	Consultar datos Gadgets	M	5	N	6
196	Consultar históricos de los Gadgets	M	5	N	6
200	Consultar datos Gateways	S	5	N	6
	Suma de puntos de historia		31		

Tabla 24 - Planificación Sprint 6

8.6.2. Metas

Las metas de este sexto Sprint pretenden crear la aplicación Front-End para dispositivos móviles como Android o iOS en los que se pueda consultar el cuadro de mando donde poder consultar el estado de los Gadgets y Gateways como el estado de los paquetes recibidos por los Gateways, además de consultar los datos de los Gadgets disponibles, como consultar los datos históricos de los Gadgets y consultar los datos de los Gateways.

8.6.3. Resultados

En el desarrollo del Sprint 6 se ha desarrollado parte de la aplicación Fron-End correspondiente a los clientes móviles utilizando la herramienta Microsoft Xamarin (Microsoft, 2019), que nos permite crear aplicaciones para dispositivos Android, iOS y Windows. Mediante la comunicación con el servidor Back-End mediante REST API se pretende visualizar los datos en dichos dispositivos.

Para la tarea Id 206, se ha creado el proyecto para el desarrollo de la aplicación móvil, se ha utilizado la tecnología Xamarin Forms, que nos permite desarrollar aplicaciones móviles para diferentes plataformas mediante Visual Studio se ha utilizado el asistente para la creación del proyecto de Xamarin Forms, en el que se han creado los siguientes proyectos:

- **SensoremTerram.App**, donde se desarrolla la parte común utilizada en los proyectos para las diferentes plataformas.
- **SensoremTerram.App.Android**, es el proyecto dedicado para la aplicación para la plataforma Android, aquí se desarrollará la parte destinada a dicha plataforma.
- **SensoremTerram.App.iOS**, es el proyecto que contiene el código para el desarrollo de la plataforma iOS, aquí se escribirá el código específico para dicha plataforma.
- **SensoremTerram.App.UWP**, es el proyecto para la plataforma Windows, utilizando la tecnología Universal Windows Platform que permite desarrollar aplicaciones para diferentes plataformas de Windows como móviles, escritorio, tabletas, etc. (Microsoft, 2018), donde escribiremos el código específico para Windows.

También se han utilizado diferentes librerías obtenidas de NuGet como son las siguientes librerías:

- **Autofac**, esta librería proporciona un contenedor que permite inyectar las dependencias de los módulos desarrollados (Autofac Contributors, 2020).
- **Arc.UserDiaglogs**, esta librería nos permite mostrar cuadros de diálogos en las diferentes plataformas (Aritchie, 2020).
- **Syncfusion**, proporciona un conjunto de librerías que nos permite visualizar gráficos, tablas y una gran variedad de controles (Syncfusion Inc., 2019).

Además, se han utilizado varias librerías que pertenecen a Pitasoft que están disponibles en NuGet, para el desarrollo de aplicaciones, que nos permite desarrollar utilizando el patrón Model-View-ViewModel (Microsoft, 2017),

además de proporcionar un conjunto de clases para facilitar la creación de aplicaciones con Xamarin Forms.

Dentro del proyecto **SensoremTerram.App** se han creado varias capetas donde se han implementado diferentes interfaces y clases para el desarrollo del proyecto que se puede ver en la ilustración 54 y que detallaremos a continuación:

- La carpeta **Behaviors** va a contener clases que permitan cambiar el comportamiento de controles para adaptarlos a las necesidades de la aplicación.
- La carpeta **Converters** va a contener las clases que permitan transformar datos que pueda necesitar la aplicación para poder ser utilizados.
- La carpeta **Models** va a contener la estructura de datos que puede necesitar la aplicación.
- La carpeta **Services** va a contener los servicios que son necesarios para el correcto funcionamiento de la aplicación.
- La carpeta **Style** va a contener los estilos de apariencia de la aplicación, como el color y forma de las tablas que se utilizan.
- La carpeta **ViewModels** es donde se va a implementar toda la lógica de las pantallas de las aplicaciones. Todos los archivos que representan la lógica terminan con el postfijo ViewModel.
- La carpeta **Views** se va a implementar el diseño de las pantallas. Dentro de esta carpeta hay una carpeta llamada **Controls** donde se depositarán todos los controles personalizados necesarios para la aplicación. Todos los archivos que representan vistas terminan con el postfijo View.

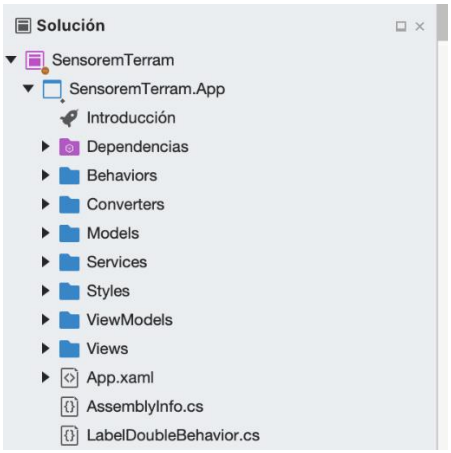

Ilustración 54 – Estructura de directorios del proyecto SensoremTerram.App.

En la capeta **Services**, se han creado interfaces y clases, que ofrecen diferentes servicios que se necesitan para el desarrollo de la aplicación. A continuación, se realiza una descripción detallada de cada una de las estructuras de datos:

- **DialogService**, esta clase implementa la interfaz **Pitasoft.Shell.Xamarin.Services.IDialogService** que nos va a permitir visualizar cuados de información para informar al usuario sobre alguna notificación.
- **IoCService**, esta clase implementa la interfaz **Pitasoft.Shell.Services. IIoCService** donde se especifican las Interfaces y sus correspondientes implementaciones, para cuando la aplicación necesite el uso de dicha interfaz se puede obtener su implementación mediante la inyección de dependencias. Además, en esta clase se tienen que registrar todos los modelos de vista para poder utilizarlos, sino se registra, la aplicación no puede generar la lógica para las vistas, produciendo excepciones en la ejecución.
- **ISensoremTerramRestService**, esta interfaz que hereda la interfaz **Pitasoft.Shell.Services.IRestService** es donde se define el cliente para poder realizar las llamadas a los servicios REST, demás de heredar las interfaces **IDashboardRepository**, **IGadgetRepository**, **IGatewayRepository** y **ISampleRepository** para implementar los métodos para obtener los datos del servidor Back-End.

- **ISensoremTerramSettingsService**, esta interfaz que hereda de la interfaz **Pitasoft.Shell.Xamarin.Services.ISettingsService** permitirá definir los parámetros de personalización de la configuración de la aplicación.
- **MenuService** es una clase que hereda de la clase abstracta **Pitasoft.Shell.Menu.MenuServiceBase** donde se definirá el menú de la aplicación con sus correspondientes opciones.
- **NavigationService** es una clase que implementa la interfaz **Pitasoft.Shell.Xamarin.Services.INavigationService** donde se han definido unos métodos que permite poder gestionar el sistema de navegación entre las diferentes pantallas de la aplicación.
- **SensoremTerramRestService** clase que hereda la clase abstracta P**itasoft.Shell.Services.RestServiceBase** que implementa métodos para poder realizar llamadas al servidor Back-End mediante comandos GET, POST, PUT y DELETE e implementa la interfaz **ISensoremTerramRestService** donde se implementa los métodos para realizar las llamadas para obtener los datos necesarios del servidor Back-End.
- **SensoremTerramSettingService** clase que hereda la clase abstracta **Pitasoft.Shell.Xamarin.Services.SettingServiceBase** que ofrece métodos para poder guardar y leer parámetros de configuración de la aplicación e implementa la interfaz **ISensoremTerramSettingsService** donde se implementa los parámetros de configuración.

La carpeta **ViewModels** es donde se van a crear todos los modelos de las vistas que se van a utilizar en la aplicación, inicialmente se han creado para dicha tarea los siguientes modelos vista:

- Clase **AboutViewModel** que hereda de la clase abstracta **Pitasoft.Shell.Xamarin.ViewModels.PageBase** que implementa la funcionalidad del patrón MVVM, incluyendo los métodos necesarios para dicha implementación. En esta clase se ha implementado la vista modelo de la pantalla referente a la información de la aplicación.

- Clase **MainViewModel** que hereda de la clase abstracta **Pitasoft.Shell.Xamarin.ViewModels.MainViewModelBase** que implementa la funcionalidad del patrón MVVM, donde se ha definido la funcionalidad de la pantalla principal.
- Clase **MenuViewModel** que hereda de la clase abstracta **Pitasoft.Shell.Xamarin.ViewModels.ManuViewModelBase** que implementa la funcionalidad del patrón MVVM, donde implementa la funcionalidad del menú principal de la aplicación.

En la carpeta **Views** se definen todas las pantallas que se van a utilizar en la aplicación permitiendo desacoplar la interfaz de usuario con la lógica de la aplicación. Para esta tarea se han definido las siguientes vistas:

- Vista **AboutView** visualiza la pantalla de la información de la aplicación como se puede ver en la ilustración 55.

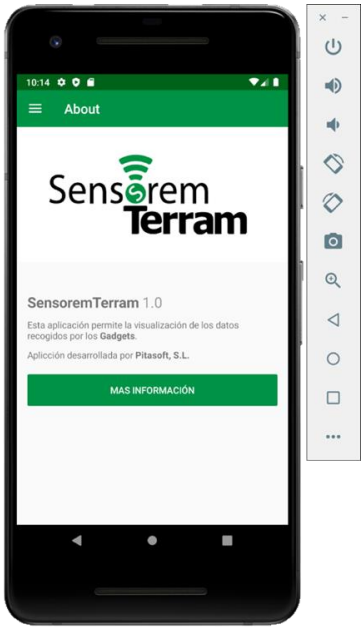

Ilustración 55 - Pantalla con información sobre Sensorem Terrram

- Vista **MainView** visualiza la pantalla principal de la aplicación, que consiste en un contendor de páginas que implementa la clase **MasterDetailPage**, que permite tener pantallas para la navegación entre las diferentes pantallas además de disponer de una pantalla deslizable

para visualizar una pantalla detalle que se ha utilizado para visualizar el menú,

- Vista **MenuView** visualiza el menú de la aplicación como se puede ver en la ilustración 56.

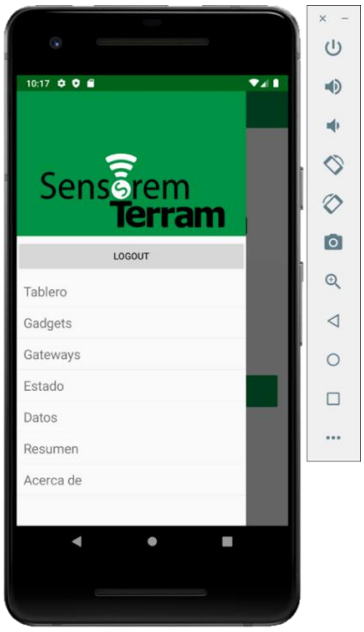

Ilustración 56 - Menú principal de la aplicación

- Vista **StartView** es una pantalla en blanco, necesaria en iOS que es utilizada por la vista **MainView** utilizándola inicialmente en la parte maestra, pues sino la aplicación producirá un error.

Además, dentro del proyecto **SensoremTerram.App** está el archivo de inicio de la aplicación **App.xaml** que se ha modificado para adaptar a las necesidades de la llamada de la aplicación, como configurar la dirección de los servicios REST, se han definido los colores que se van a utilizar en la aplicación y otras configuraciones necesarias para el correcto funcionamiento de esta.

En el desarrollo de la tarea Id 212, que consiste en el desarrollo de un cuadro de mandos para visualizar los datos referentes a los Gadget y Gateways, para saber su estado de funcionamiento, así como los paquetes recibidos para conocer el estado del sistema se ha procedido a crear una nueva librería llamada **SensoremTerram.Shared** donde se ha implementado la clase **DataSummaryConnections** que contiene los datos referentes a la información

de las conexiones, con la información necesaria para la visualización de los datos.

Dentro de la clase **SensoremTerramRestService** se ha implementado la interfaz **IDashboardRepository** que contiene las llamadas al servidor Back-End para la obtención de los datos referente al estado del sistema.

Para la implementación de la lógica de la aplicación se ha creado la clase **DashboardViewModel** que hereda de la clase abstracta **PageBase** implementando la funcionalidad MVVM, donde se ha desarrollado la recepción de datos del servidor Back-End, además de procesar los datos para su posterior visualización.

Por último, para esta tarea de ha desarrollado la vista llamada **DashboardView** que permite visualizar los datos de estado del sistema como se puede ver en la ilustración 57. Para ello se ha creado dentro de la carpeta **Controls** los controles personalizados que se utilizarán en las vistas.

Se han creado los siguientes controles, **Scoreboard** que permite la visualización de un valor numérico, el control **SummaryConnections** para visualizar mediante gráficos el estado de las conexiones de los Gadget y los Gateways. Además del control **SummaryDataLogs** que permite conocer el estado de los paquetes recibidos de los Gadgets como de los Gateways, pudiendo consultar datos del día actual, como de los últimos 3 días, los últimos 7 días, así como los paquetes procesados en los últimos 30 días.

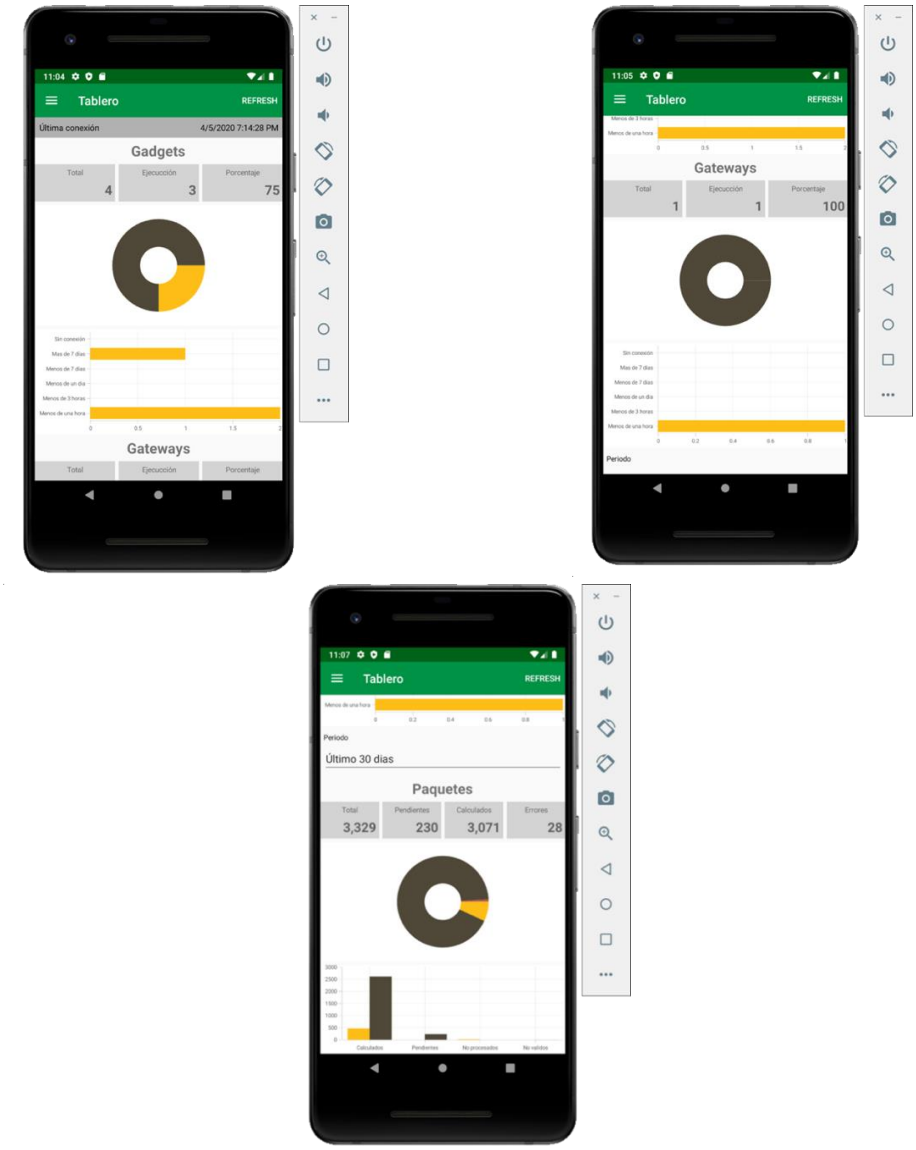

Ilustración 57 - Pantalla de cuadro de mandos

Para la tarea Id 194, se ha añadido a la interfaz **ISensoremTerramRestService** la interfaz **IGadgetRepositiry**, con lo que se ha tenido que implementar en la clase **SensoremTerramRestService** las correspondientes llamadas al servidor Back-End para obtener los datos necesarios para la aplicación. Se han creado dos pantallas, una consiste en la pantalla que permite consultar datos de los Gadgets como se puede ver en la ilustración 58, para realizar dicha pantalla se ha creado la lógica **GadgetsViewModel** que permite obtener los datos del servidor Back-End, además de su correspondiente vista **GadgetsDataView** que contiene la visualización de los últimos datos de los Gadgets, como es la versión del hardware, su identificador, nombre y fecha de su última conexión.

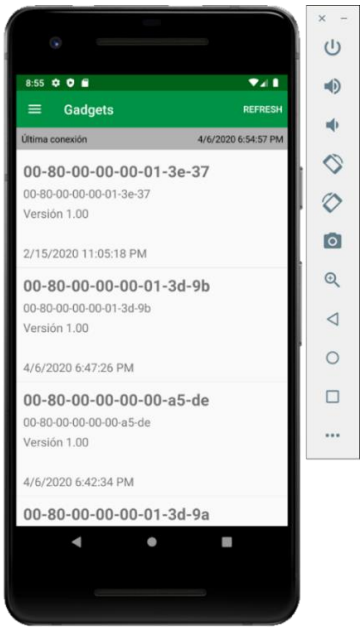

Ilustración 58 - Consulta de Gadgets

Además, se ha creado otra pantalla que permite consultar los últimos datos recibidos de cada uno de los Gadgets. Se ha creado la lógica del modelo **GadgetLatestDataViewModel**, que permite obtener los datos del servidor Back-End y su correspondiente vista **GadgetLatestDataView** como se puede ver en la ilustración 59. Mediante esta pantalla se ha dibujado una tabla utilizando el control **SfDataGrid** de Syncfusion, permitiendo visualizar el estado de los Gadgets con sus últimos datos recogidos.

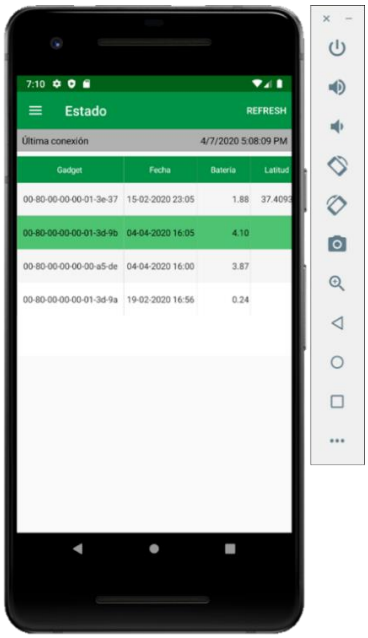

Ilustración 59 - Últimos datos de los Gadgets

Para la realización de la tabla se ha creado un nuevo proyecto llamado **SensoremTerram.Table** que se va a utilizar para la representación de los datos en formato tabla donde se han definido las siguientes interfaces y clases para dicha función:

- Interfaz **ICells** implementa las funcionalidades mínimas de las celdas para poder consultar si tiene valor, en una determinada celda de un registro.
- Clase **Cells**, implementa la interfaz **ICells** y representa las celdas de un registro, pudiendo acceder a dichas celdas para obtener valores y asignarlos.
- Clase **FixedCells** implementa la interfaz **ICells** y representa las celdas de un registro cuyo contenido es fijo.
- Clase **Columns** representa las columnas de la tabla, que como propiedades tiene un identificador, titulo y formato de la representación de los datos de las columnas.
- Clase **Row** representa el registro de una tabla y este contiene **FixedCells** y **Cells**.
- Clase **Table** es la representación de la tabla que contiene columnas y registros, pudiendo acceder a los datos con sus correspondientes índices.

Una vez que se puede hacer la representación de la tabla se ha creado un control llamado **TableView** que se encuentra dentro de la carpeta **Controls** dentro de la carpeta **Views** y este permite visualizar la estructura de datos Table en la pantalla. Con dicha implementación se pretende simplificar la visualización de datos en las próximas pantallas.

En la tarea Id 196 se han creado dos pantallas que nos permite consultar los datos de los Gadget de diferentes formas, una de ellas una consulta detallada para un intervalo de fecha y otra pantalla que se puede consultar los datos estadísticos por día.

Primero se ha heredado la interfaz **ISempleRepository** en la interfaz **ISensoremTerramRestService**, por lo que en la clase

SensoremTerramRestService se ha implementado los métodos necesarios, para obtener la lista de factores disponibles.

Para la primera pantalla se han creado dos pantallas, una pantalla para realizar la consulta de los datos que está formada por la clase **GadgetRawDataQueryViewModel** donde se ha implementado toda la lógica de la pantalla, además de una nueva clase llamada **GadgetRawDataQuery** donde se almacenan los parámetros de la consulta que ha sido creada en la carpeta **Models** dentro del proyecto **SensoremTerram.App.** Esta clase contiene como propiedades el intervalo de fecha y el identificador de Gadget para realizar la consulta al servidor Back-End. Para la pantalla se ha creado la clase **GadgetRawDataQueryView** que en la ilustración 60 se puede ver su representación y funcionamiento.

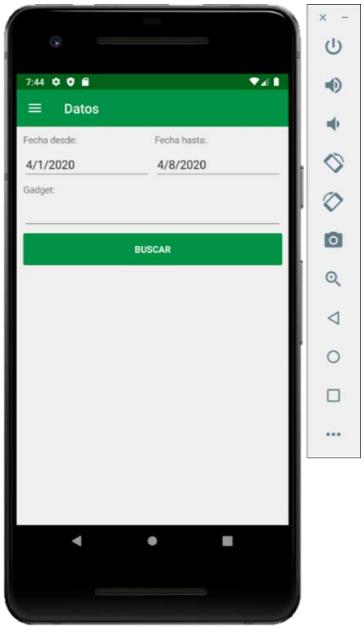

Ilustración 60 - Pantalla de consulta de datos de Gadgets

En dicha pantalla se puede seleccionar un intervalo de fecha y seleccionar un dispositivo Gadget mediante un desplegable y una vez seleccionados los parámetros deseados se puede consultar los datos en una nueva pantalla. Esta nueva pantalla se ha implementado mediante la clase **GadgetRawDataViewModel** donde se ha implementado la lógica de la pantalla, creando los datos necesarios para la visualización. Se ha creado dos estructuras de tabla una que contiene un resumen estadístico con los datos para el intervalo

de fecha solicitado, una estructura de datos para visualizar un gráfico y por último otra tabla con el detalle de los datos.

Para la visualización se ha creado la clase **GadgetRawDataView** y el control **RawDataCharView** que permite visualizar la gráfica para un factor seleccionado, que para la representación de los gráficos se ha utilizado un control de Syncfusion, como se puede ver en la ilustración 61.

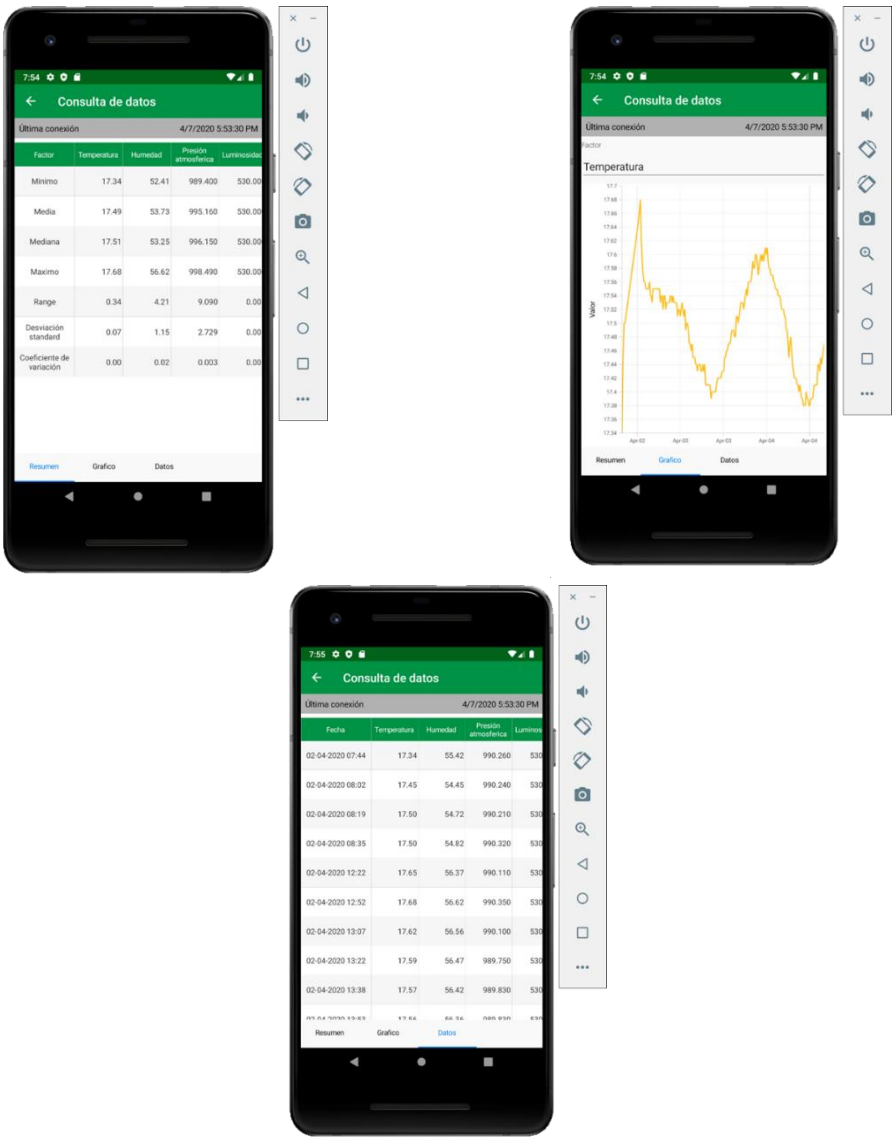

Ilustración 61 - Pantalla de consulta de datos de Gadget

También como se ha comentado se ha creado otra pantalla que permite consultar datos resumen por día de los Gadgets, para ello se han creado dos pantallas: una pantalla para realizar la consulta y otra para visualizar los datos.

Para la pantalla de consulta se ha creado la clase **GadgetSummaryDataQueryViewModel** que contiene la lógica para la pantalla solicitada, esta necesita la clase **GadgetSummaryDataQuery** que contiene las propiedades como son el intervalo de fecha, el identificador del Gadget y el factor a consultar, que es pasada a la pantalla de visualización de datos que se comentada a continuación. Para la pantalla se ha creado la clase **GadgetSummaryDataQueryView** como se puede ver en la ilustración 62.

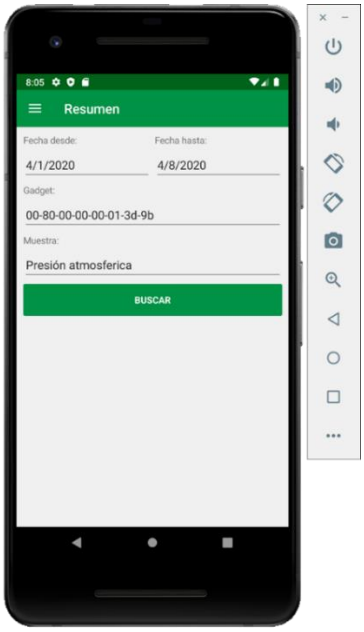

Ilustración 62 - Consulta resumen de Gadget

Una vez ejecutada la consulta se procede a visualizar los datos con la nueva pantalla. Para esta nueva pantalla se ha implementado la clase **GadgetSummaryDataViewModel** donde está la lógica de la pantalla, que en este caso crea una clase Table donde se puede ver los datos estadísticos por día del factor consultado, además de la estructura de datos necesarios para la visualización de datos, que para ello se ha creado una nueva clase llamada **DataSummary** donde contiene el mínimo, máximo, media y mediana del factor del Gadget consultado, en el proyecto llamado **SensoremTerram.Shared**. Además, se ha creado la clase **GadgetSummaryDataView** que representa la visualización de dicha pantalla como se puede ver en la ilustración 63.

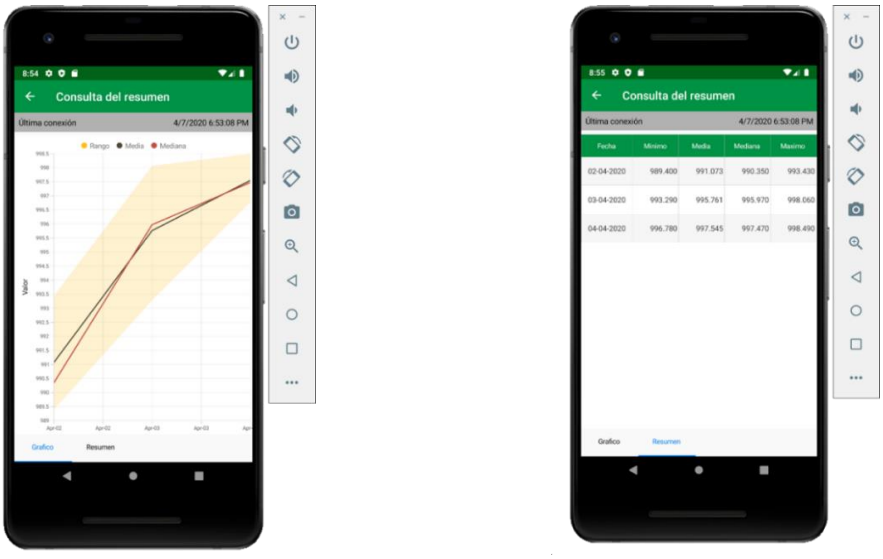

Ilustración 63 - Pantalla resumen datos de un Gadget

Como se puede apreciar en la ilustración 63, se puede ver una gráfica con la evolución de un factor en el tiempo, dibujando en esta el valor medio, la mediana y el rango de valores. Además, se puede consultar en una tabla resumen los datos estadísticos por día, es decir, la media, mediana, varianza, rango, etc., que en la ilustración anterior es la visualización de la presión atmosférica.

Para la última tarea de este Sprint, se ha realizado la tarea Id 200, que consiste en consultar los datos de los Gateways. Para la realización de dicha tarea se ha procedido a crear la clase **GatewaysViewModel** que contiene la lógica de la pantalla. Para poder obtener los datos de servidor Back-End se ha heredado la interfaz **IGatewayRepository** en la interfaz **ISensoremTerramRestService**, así se ha tenido que implementar los nuevos métodos en la clase **SensoremTerramRestService**. Su correspondiente pantalla se ha creado mediante la clase **GatewaysView** donde en la ilustración 64 se puede ver visualizar donde se dispone una vista de los Gadgets disponibles con la última fecha de actualización.

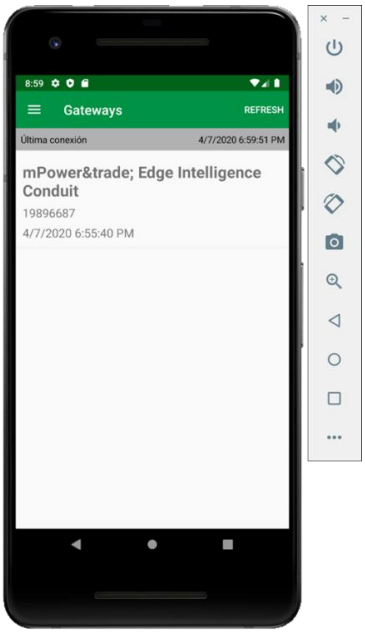

Ilustración 64 - Consulta de Gateways

8.6.4. Revisión

Los resultados de este Sprint han sido los deseados ya que se ha conseguido una aplicación móvil multiplataforma que funciona en Android, iOS y Windows donde:

- Poder consultar el estado del sistema mediante un panel de control, pudiendo con un solo vistazo poder visualizar dicho estado.
- Consultar los Gadget disponibles.
- Consultar los datos históricos de los Gadgets.
- Consultar los Gateways disponibles.

Utilizando la infraestructura creada en Sprint anteriores se ha creado la aplicación móvil utilizando una implementación parecida a la creada en la aplicación cliente web con tecnología Blazor. No se ha quedado ninguna tarea pendiente para hacer en el siguiente Sprint, pero una ver revisado el Sprint se he decidido crear una nueva tarea para mejorar alguna visualización de la aplicación cliente como tener una pantalla inicial mientras se carga la aplicación como una personalización de los iconos de la aplicación.

Además, se ha observado que el servidor Back-End se produce fallo en el procesamiento de los datos recibidos por los Gateway de los Gadgets, y deja de funcionar el proceso automático de procesamiento de paquetes. Como se dispone de bastantes paquetes con información, se ha de depurar y mejorar el algoritmo de procesado de paquetes de los Gadgets. Así que también se ha decidido crear un sistema de eventos en donde reflejar la información sobre el procesamiento de los datos recibidos de dichos Gadgets, dejando información sobre errores críticos, errores, advertencias e información sobre los paquetes recibidos y poder observarlos para ver el correcto funcionamiento del sistema.

A continuación, exponemos nuevas tareas que están definidas en la tabla 25, donde se pretende realizar una mejorar del proyecto, con errores que se han detectado para obtener un mejor comportamiento y hacer que este tenga una apariencia más atractiva.

Cuestiones		Tareas		Prioridad	Story Points	Realizado	Sprint
Id	**Titulo**	**Id**	**Titulo**				
221	Procesado de datos	288	Revisión de procesado de datos	M	5	N	
		289	Registro de eventos del sistema	M	2	N	
157	Aplicación Web	290	Consulta de eventos del sistema	M	3	N	
158	Aplicación móvil	287	Mejora de iconos, pantalla inicial	C	5	N	
		291	Consulta de eventos del sistema	M	3	N	
Suma de puntos de historias…					**18,0**		

Tabla 25 - Nuevas tareas definidas en el Sprint 6 Revisión.

8.6.5. Retrospectiva

En la tabla 26 se puede ver la comparativa de los puntos de historia estimados con los reales para la realización del Sprint.

Tareas		Prioridad	Story Points	Realizado	Sprint	Story Points Reales
Id	Titulo					
206	Crear base aplicación IU	M	8	N	6	10
212	Cuadro de mandos	M	8	N	6	8
194	Consultar datos Gadgets	M	5	N	6	10
196	Consultar históricos de los Gadgets	M	5	N	6	10
200	Consultar datos Gateways	S	5	N	6	2
	Suma de puntos de historia		31			40

Tabla 26 - Comparación de los puntos de historia estimados y reales del Sprint 6

Como se puede observar se han producido diferencias respecto a la estimación inicial con la real por lo que se han necesitado más horas para poder dar por finalizado este Sprint.

En la tarea Id 206, se han necesitado más horas para implementar la base de aplicación de Xamarin, pues se ha tenido que experimentar con las librerías y ver que la aplicación tiene un comportamiento apropiado.

En la tarea Id 212 se ha procedido a desarrollar un sistema muy parecido al desarrollado en la aplicación Web con Blazor y con la experiencia aprendida con el uso de gráficos de la libraría Syncfusion ha sido muy parecido y se ha realizado en el tiempo estimado.

Respecto a la tarea Id 194 se ha necesitado más tiempo pues se ha tenido que probar con diferentes controles y ver cual se adaptaba más a los requerimientos, así como se ha tenido que dedicar tiempo a diferentes implementaciones de datos para representar los datos en formato tabla.

Para la tarea Id 196 también se han necesitado bastante más horas pues con el problema del tamaño de pantalla se ha tenido que buscar diferentes

implementaciones para ver cuál era la mejor solución para la visualización de los datos histórico. Se ha tenido que probar con diferentes controles y ver como se comportaban.

Por último, para la tarea Id 200 con lo aprendido en las tareas anteriores se ha realizado consumiendo menos tiempo que lo esperado. Por lo que este Sprint ha sido más complicado necesitando bastante más tiempo que el estimado inicialmente.

En la ilustración 65 se puede observar una gráfica Brundown donde se puede ver la evolución del desarrollo del Sprint 6.

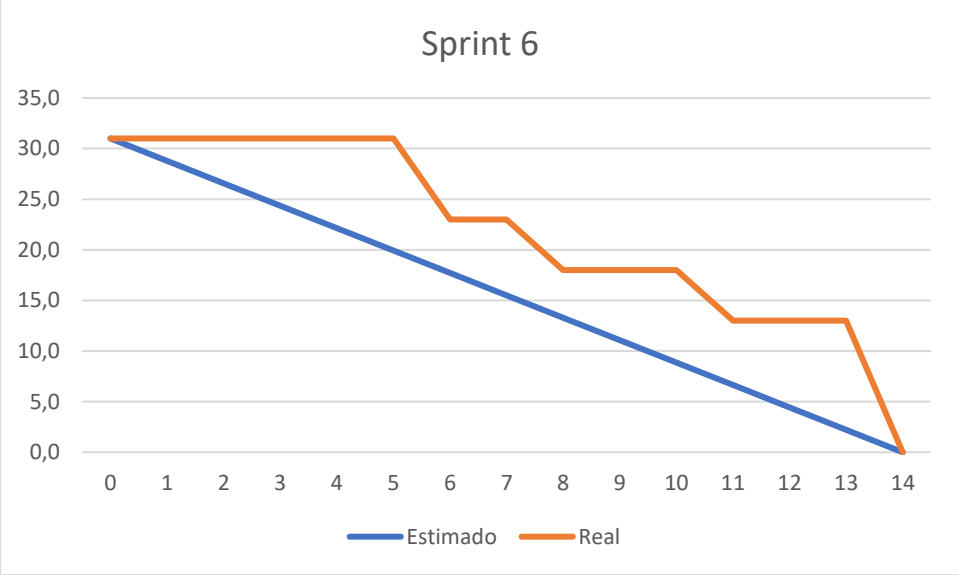

Ilustración 65 - Gráfico Brundown Sprint 6

8.7. Sprint 7

8.7.1. Planificación

En el séptimo Sprint se tendrá que acometer 31 puntos de historia donde en la tabla 27 se han identificado las tareas a realizar en este Sprint, además de su prioridad y la estimación.

Tareas		Prioridad	Story Points	Realizado	Sprint
Id	Titulo				
220	Ahorro de energía	C	3	N	7
169	Estudio de sistemas GPS hardware	C	13	N	7
170	Enviar datos de localización al Gateway	C	5	N	7
198	Consultar datos Gateway	S	5	N	7
199	Consultar localización Gateways	C	5	N	7
	Suma de puntos de historia		31		

Tabla 27 - Planificación Sprint 7

8.7.2. Metas

Entre las metas de este sprint están estudiar la manera de localizar geográficamente los Gadgets para ello se tiene que estudiar si se puede hacer mediante algún dispositivo hardware o mediante la aplicación móvil, registrando la localización del Gadget. Si se puede obtener la localización del Gadget mediante hardware se ha de enviar la localización al Gateway para que sea procesada por el servidor Back-End.

Además, se tiene que estudiar el consumo de energía del Gadget e intentar que este consuma menos energía.

Por último, en la aplicación cliente web se ha poder consultar la información de los Gateways, además de conocer su localización que será dibujada en un mapa.

8.7.3. Resultados

En el desarrollo del séptimo sprint se han realizado todas las tareas relacionadas con la geolocalización del Gadget y consumo de energía, además de las tareas del Front-End para el navegador Web la consulta de datos del Gateway y consultar su localización.

Se ha comenzado por la tarea Id 169, en la que se ha precedido a estudiar que dispositivos de hardware se disponen en el mercado para utilizar en el Gadget y como el dispositivo mDot de MultiTech puede comunicarse. Como principalmente en el desarrollo del Gadget se está utilizando el protocolo I2C, se han buscado dispositivos GPS que pueden comunicarse por dicho bus. La otra posibilidad es el uso de protocolo UART (Estrada-Marmolejo, 2017).

Entre los dispositivos de hardware de GPS que permite comunicar con el protocolo I2C o UART se ha utilizado el Hardware Open Source de SparkFun producto GPS-14414 que se denomina GPS Breakout – XA110 (Qwiic) (SparkFun, 2020). Este dispositivo está compuesto por un chip Titan X1 GPS el cual permite una comunicación con ambos buses, además posee una antena integrada, admite hasta 210 canales PRN con 99 canales de búsqueda y 33 canales de seguimientos simultáneos. Tiene soporte de GPS, GLONASS, QZSS, SBAS y etc.. Dispone de una batería RTC incorporada que permite una funcionalidad de arranque en caliente. En la ilustración 66 se puede ver el dispositivo.

Ilustración 66 - SparkFun GPS Breakout - XA1110

Una vez montado el dispositivo en la protoboard, se ha procedido a realizar la comunicación con el protocolo I2C y realizar pruebas de comunicación con el dispositivo.

Realizadas las pruebas se ha procedido a realizar la tarea Id 170, para integrarlo en el Gadget para proceder a enviar los datos de posicionamiento del Gateway. Para realizar la comunicación entre el hardware mDot y el módulo de GPS se ha utilizado una librería llamada TinyGPS++ (Hart, 2014) para Arduino que permite analizar flujos de datos NMEA (National Marine Electronis Association, 2019) proporcionados por los módulos GPS. Esta librería se ha integrado en nuestro código adaptándola a el código, se han creado varios archivos que enumeramos a continuación:

- Archivo **constants.h** que define constantes y macros.
- Archivo **type.h** define tipos utilizados en la librería TynyGPS++.
- Archivo **TinyGPSplus.h** define las estructuras de datos.
- Archivo **TinyGPSplus.cpp** donde está el código que permite analizar los datos NMEA.

Estos archivos se han integrado en el proyecto del Gadget, adaptándolos para que puedan ser compilados por el compilador de Mbed.

Utilizando las librerías que proporciona SparkFun para controlar módulos GPS basados en MT3333 y MT3339 sobre I2C que están en el enlace (SparkFun, 2019) se han creado la clase **I2CGPS** que permite controlar dicho hardware, esta clase hereda de **SensorI2CBase** y define los siguientes métodos:

- Método **getGPS** que obtiene la estructura de datos con la información del GPS.
- Método **check** permite verificar si hay datos en buffer pendientes.
- Método **availabel** devuelve el número de bytes que tiene que procesar.
- Método **readNext** lee el siguiente byte disponible del buffer.

- Método **sentMTKpacket** permite enviar al módulo GPS un comando para la configuración de este.
- Método **createMTKpacket** permite crear un comando MTK para poder enviarlo al módulo GPS.
- Método **calcCRCforMTK** crea el digito de control que debe tener el paquete MTK al final de su secuencia.

Ahora el Gadget puede leer los datos suministrados por el hardware GPS y enviar dichos datos al Gateway como si fuera un sensor más y el servidor Back-End solo tiene que procesar dichos datos.

Esta clase lo que hace es leer los paquetes con una longitud de 32 bytes que se almacenan en un buffer que se ha definido con un tamaño de 512 bytes. Una vez que se han leído todos los paquetes del GPS o el buffer se ha llenado, lo que se procede es procesar dichos paquetes con la librería TinyGPS++. Esta librería lee los datos en formato NMEA y va actualizando los datos que ha recibido el GPS, como la fecha, hora, longitud, latitud, altitud, número de satélites que tiene localizados y otros parámetros.

Mediante él envió de paquetes MTK al módulo GPS se permite la configuración de este. Estos paquetes tienen una estructura bien definida, además de incluir un digito de control que tiene que ser calculado correctamente para que el paquete MTK sea procesado. Se puede configurar el modo de funcionamiento de GPS, como la frecuencia de actualización y modos de ahorro de energía, etc. (GlobalTop Tech Inc., 2015).

Una vez implementada la librería y probada se ha procedido a añadir la llamada de la clase I2CGPS para que el Gadget comience a leer los datos del GPS y enviarlos al Gateway.

Para la tarea Id 220, ahorro de energía se ha procedido a estudiar el consumo de energía del Gadget, para intentar minimizar dicho consumo. Para realizar dicha tarea se ha utilizado un amperímetro conectado a la entrada de la batería y se ha medido el consumo, como se puede ver en la ilustración 67.

Inicialmente se ha medido el consumo del Gadget sin el GPS, que se estudiara más adelante, el consumo inicial es de 4,149 mA aproximadamente cuando este encendido. A continuación, se han configurado los chips para conseguir ahorrar energía poniendo estos en modo ahorro de energía, consiguiente un consumo de 3,750 mA en modo reposo.

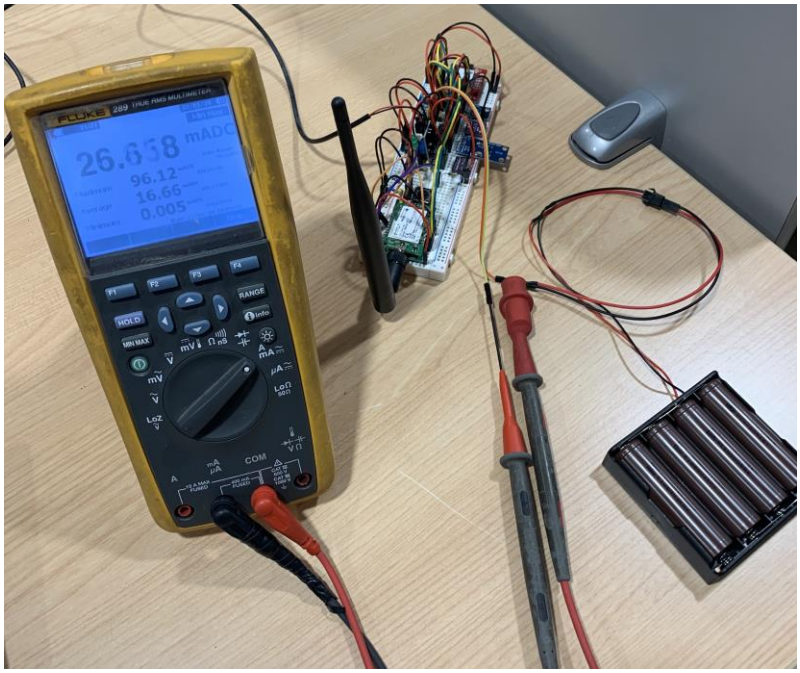

Ilustración 67 - Medición del consumo eléctrico.

Además, se ha modificado el circuito eléctrico para minimizar aún más el consumo eléctrico, pues se ha utilizado un transistor 2N3904 para utilizarlo como interruptor cuando es saturado, con esto desactivamos la parte del circuito analógico, con lo que se consigue disminuir el consumo de energía en modo reposo a 2,700 mA aproximadamente, en la ilustración 68 se puede ver como se ha montado el transistor.

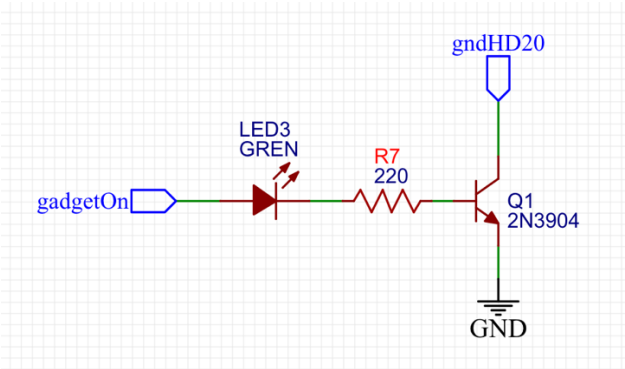

Ilustración 68 - Transistor 2N3904.

También se ha eliminado el led de encendido del Gadget disminuyendo el consumo a 1,301 mA aproximadamente. Con esta modificación ya no se sabe si esta está funcionando o esta apagado, por lo que se ha añadido al circuito un pulsador conectado a una toma del MultiTech mDot para que despierte el Gadget y ver que este está funcionando, mediante el led de funcionamiento, la electrónica añadida se puede observar en la ilustración 69.

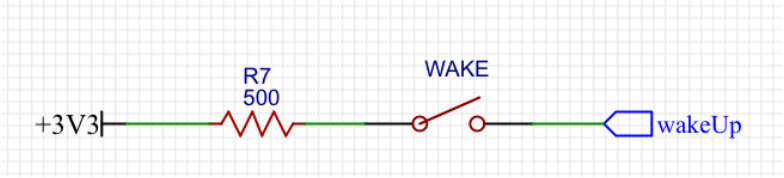

Ilustración 69 - Electrónica para despertar al Gadget.

Se ha añadido al circuito otro pulsador para poder hacer reset al MultiTech mDot y otras partes del hardware, así poder poner el hardware en un estado normal si este se quedara en un estado inconsistente, aunque hasta ahora el sistema no ha producido un comportamiento inesperado, ya que el sistema lleva más de un mes funcionando sin ningún tipo de problemas. En la ilustración 70 se puede ver el esquema de hardware implementado.

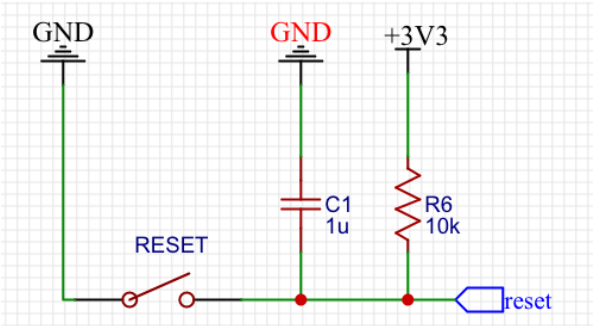

Ilustración 70 - Electrónica para hacer reset del Gadget.

El GPS tiene un consumo muy elevado de energía, pues cuando está conectado tiene un consumo aproximado de unos 39,203 mA. Mediante el uso de comandos MTK se han realizado diferentes configuraciones, pues el dispositivo tiene varios modos de funcionamiento para el consumo de energía, estos modos son:

- Consumo continuo de energía que tiene mejor comportamiento para GNSS, pero incrementa el consumo en casi 40 mA.

- Modo de ahorro de energía, que optimiza el consumo de energía, cuando el dispositivo esta en este modo el consumo total del circuito es de unos 3,344 mA, por lo que es mucho mejor, este puede tener varios modos de funcionamiento:
    - Modo Standby, este modo se ha probado, pero tiene el problema que cuando el dispositivo está dormido, solo se puede despertar por una comunicación serie o haciendo un reset del dispositivo. Esta opción se ha tenido que descartar, pues mediante el bus I2C no se puede despertar y el reset puede hacer que se tenga que volver a realizar toda la inicialización del GPS, como comenzar a detectar satélites y otros parámetros.
    - Modo Periódico, también se ha probado, siendo el problema la durabilidad del consumo de la batería que ha sido muy baja, pues los ciclos de activación y parada tiene un máximo aproximado de un 1 día, todos los días supone que tiene que actualizar datos y hay que dejar que el dispositivo calcule los datos de los satélites y luego leerlos, por lo que hay que dejar el dispositivo despierto en un periodo de tiempo para que podamos hacer la lectura a través del bus I2C.
    - Modo AlwaysLocate(TM), este es un modo automático de ahorro de energía, va automáticamente determinando si consume más energía o menos, pues si tiene desplazamiento el dispositivo, que no es nuestro caso, pues el Gadget no tiene por qué moverse durante un periodo de tiempo largo. Pero este modo tiene el problema que cuando el dispositivo va a realizar la lectura de los datos y este no está despierto no se puede hacer. El consumo de energía se ha optimizado durando bastante, pero no se pueden leer los datos del GPS por el bus I2C cuando se necesitan leer.
- Modo Backup, es un modo que tiene un consumo muy bajo cuando es activado, lo que hace es guardar en una memoria los datos del último estado y hay que hacer un wake up en el pin del dispositivo activándolo a 3.3v este pasa a estar activo y dejando un tiempo prudente podemos leerlo mediante el bus I2C.

Entre las configuraciones que se han probado, finalmente se ha decidido usar el modo Backup, pero esto implica realizar modificaciones básicas del Gadget, además de modificar la electrónica.

Para la tarea 198, que consiste en Consultar datos de los Gateways en la aplicación Front-End web. Se ha procedido a crear una nueva página llamada **Gateways.razor** en el proyecto **SensoremTerram.Client** que permite obtener los datos del servidor Back-End llamando al controlador API REST para obtener los datos de los Gateways disponibles en el sistema que fue implementado en el Sprint 5 y visualizar en pantalla la identificación de estos, como la última conexión que realizo el Gateway como se puede observar en la ilustración 71.

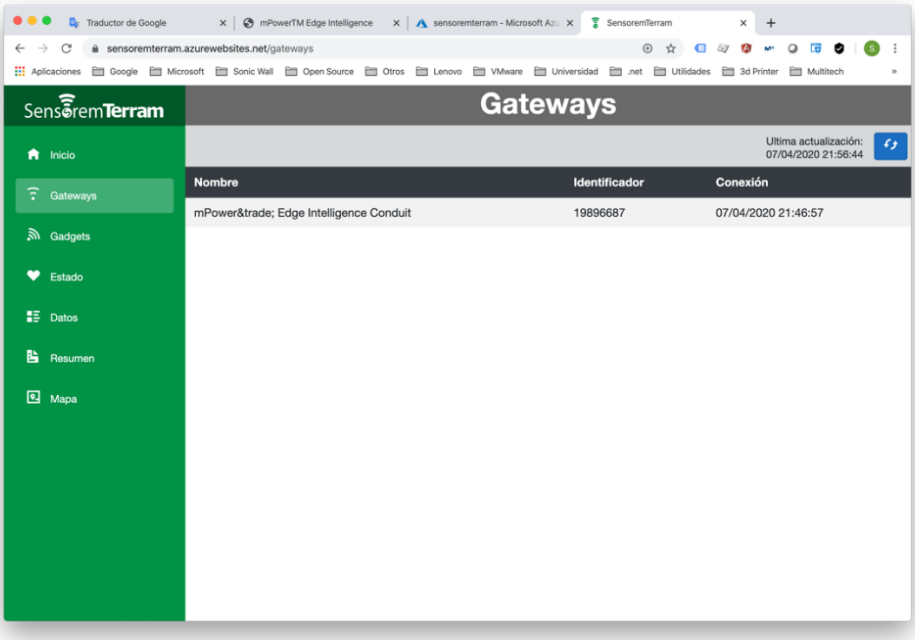

Ilustración 71 - Consulta de Gateways en la aplicación web.

La última tarea Id 199 consiste en poder consultar la localización de los Gateways en un mapa y para poder realizar esta tarea se ha tenido que crear una nueva página llamada **Map.razor** obteniendo el resultado que se puede ver en la ilustración 72. Para la visualización del mapa se ha utilizado **Google Maps Platform** (Google Cloud, 2019) y como dicha plataforma implementa el **Maps JavaScript API** el cual permite interactuar y personalizar los mapas. Así, se ha creado una nueva librería en la solución llamada **SensoremTerram.Google**

donde implementamos las clases necesarias para utilizar dicha API, que a continuación se detallará.

Una aplicación de Blazor puede invocar funciones de JavaScript desde métodos de .NET y viceversa. Estos escenarios se denominan interoperabilidad de JavaScript o interoperabilidad de JS (Microsoft Docs, 2020). Mediante el uso de la interoperabilidad se han construido en el proyecto **SensoremTerram.Google** una serie de clases que nos permite el uso de la Api de Google.

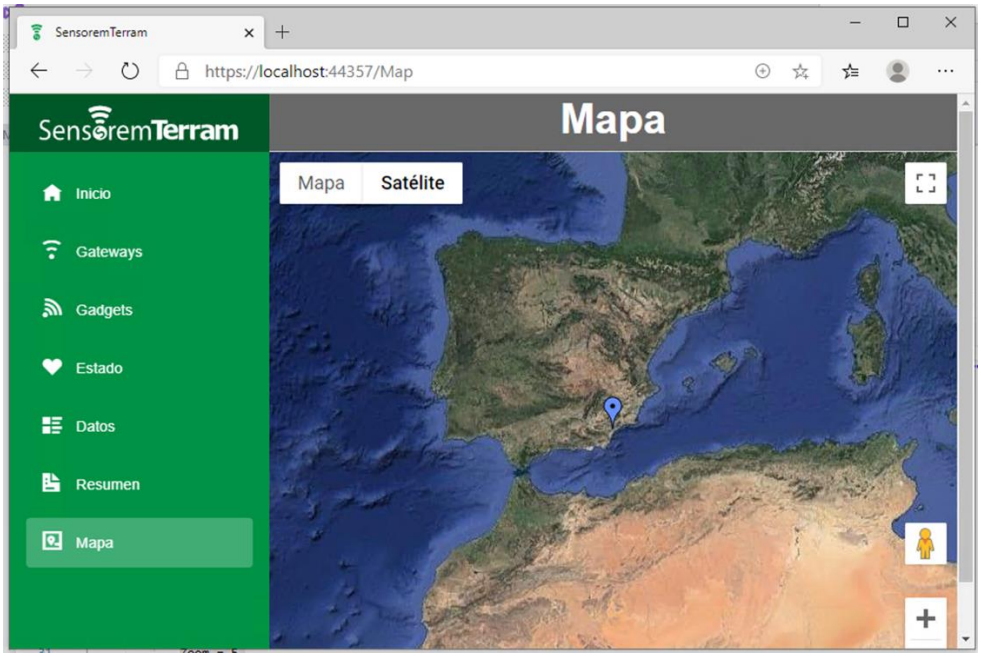

Ilustración 72 - Consultar visualización de la localización de los Gateways en la aplicación web.

El proyecto es bastante amplio, detallaremos las clases más importantes, detallando el funcionamiento por carpetas de la solución:

- Capeta Raíz del proyecto, encontramos las clases más básicas utilizadas en la librería, que detallamos a continuación:
  - La interfaz **IJSObject** representa un objeto JavaScript que contiene una propiedad con un identificador y hereda la interfaz **IDisposable**.
  - La clase abstracta **DisposableObjectBase** implementa la interfaz **IDisposable**, que es utilizada por otras clases para liberar memoria.

    - La clase **JSObject** hereda de la clase **DisposableObjectBase** y la interfaz **IJSObject** y representa un objeto de JavaScript, posee métodos para añadir eventos a los objetos JavaScript.
- Carpeta wwwroot contiene todo el código JavaScript necesario para la librería y el código .NET que va a utilizar, no encontramos el archivo **googleInteop.js**.
- La carpeta Maps encontramos los objetos de Google Maps con los que vamos a interactuar con la API de Google:
    - La clase abstracta **MapCompomentBase** implementa el componente para la visualización del Mapa.
    - La componente **GoogleMap.razor** que hereda la clase MapComponentBase permite dibujar un mapa permitiendo configurar el comportamiento y posee las propiedades necesarias para interactuar con el mapa.
    - La clase **InfoWindow** hereda de la clase JSObject permitiendo dibujar un globo mostrando información, esta clase representa la Clase InfoWindow de JavaScript de Google Map, donde se han implementado las propiedades, métodos y eventos necesarios para el proyecto.
    - La clase **Map** hereda de la clase JSObject y esta representa un objeto Map de la librería JavaScript de Google Map, donde se han implementado las propiedades, métodos y eventos necesarios para el proyecto.
    - La clase **Marker** hereda de la clase JSObject y representa el objecto Marker de la clase JavaScript de Google Map, donde se han implementado las propiedades, métodos y eventos necesarios para el proyecto.

Dentro de la carpeta **Maps** también hay dos capetas que detallamos a continuación

- La carpeta **Events** define clases necesarias para recibir información de los eventos como son las clases **MouseEvent** y **IconMouseEvent**.

- La carpeta **Models** define el conjunto de clases y enumeradores necesarios para las clases **InfoWindow**, **Map** y **Marker**, como ejemplos importantes de las clases son: **MapOptions**, **MarkerOptions** y **InfoWindowOptions**. Aunque hay una multitud de clases que han sido creadas basándose en la documentación de Google (Google Developers, 2020).

De la librería de Google se ha implementado lo necesario para nuestro proyecto.

8.7.4. Revisión

Los resultados del Sprint han sido los deseados pue se ha realizado el estudio de sistema GPS hardware y enviar datos de la localización de los Gadgets al servidor Back-End.

También se ha estudiado como ahorrar energía en el dispositivo Gadgets para que tenga un mejor consumo de la batería, pero también se ha observado con las pruebas realizadas que se tendrán que realizar modificaciones en el hardware de Gadget y modificar el software para poder despertar el Hardware del GPS. Por lo que se proponen dos nuevas tareas para desarrollar para mejorar el sistema de energía como se puede ver en la tabla 28.

Además, como el uso de la tecnología Blazor está en preview, Microsoft ha sacado una nueva versión con bastantes modificaciones, por lo que es recomendable adaptar el proyecto con el nuevo funcionamiento, pues la nueva versión en principio es más estable que las anteriores acercándose más a la versión final.

Cuestiones		Tareas		Prioridad	Story Points	Realizado	Sprint
Id	Titulo	Id	Titulo				
208	Energía	292	Mejorar el hardware para despertar GPS	W	2	N	
		293	Mejorar el software STEP y despertar GPS	W	5	N	
157	Aplicación Web	294	Adaptar proyecto a la nueva versión Blazor	M	1	N	
			Suma de puntos de historias		**8,0**		

Tabla 28 - Nuevas tareas definidas en el Sprint 7 Revisión.

Para el resto de las tareas que ha consistido en la consulta de los datos de los Gateways y su localización mediante Google Maps. No quedado ninguna tarea pendiente para hacer en el siguiente Sprint.

8.7.5. Retrospectiva

En la tabla 29 se puede ver la comparativa entre los puntos de historia estimados con los reales para la realización del Sprint.

Tareas		Prioridad	Story Points	Realizado	Sprint	Story Points Reales
Id	Titulo					
220	Ahora de energía	C	3	N	7	4
169	Estudio de sistemas GPS hardware	C	13	N	7	10
170	Enviar datos de localización al Gateway	C	5	N	7	5
198	Consultar datos Gateway	S	5	N	7	2
199	Consultar localización Gateways	C	5	N	7	12
	Suma de puntos de historias		31			33

Tabla 29 - Comparación de los puntos de historia estimados y reales del Sprint 7.

Como se puede observar se han producido pequeñas diferencias de tiempo respecto a la estimación inicial con la real para finalizar este Sprint.

Se ha producido diferencia en la tarea Id 169 necesitando menos tiempo para llevarla a cabo. Los sistemas de GPS, pues una vez consultado los diferentes sistemas de hardware y seleccionado uno, se ha procedido a consultar la documentación y hacer pruebas para la lectura de las coordenadas facilitadas

por el GPS, escribiendo el código necesario para la configuración del hardware y la lectura de los datos. Para la tarea Id 170 se ha procedido a crear el código necesario para enviar los datos al servidor Back-End según las especificaciones utilizadas anteriormente, consumiendo el tiempo estimado.

Una vez realizado él envió e las coordenadas GPS por el Gadget al servidor Back-End se ha procedido a mejorar el consumo de energía del Gadget realizando la tarea Id 220, consiguiendo mejorar el consumo de energía y viendo el comportamiento del hardware se ha decidido crear nuevas tareas para mejorar el sistema.

Respecto a la tarea Id 198 se ha consumido menos tiempo de lo estimado, pues con la experiencia adquirida en anteriores tareas se ha realizado más eficientemente.

Por último para la tarea Id 220 ha sido necesario un consumo de tiempo mayor pues esta tarea ha sido más complicada que lo estimado inicialmente, se ha tenido que estudiar el funcionamiento de la interoperabilidad de la plataforma .NET que utiliza Blazor para la comunicación del JavaScript, pues la API de Google Map esta implementada en JavaScript siendo necesario consultar la documentación del Google, así como la de Microsoft que aunque hay bastante documentación de Blazor, todavía hay bastantes carencias en el sistema y gracias a los foros de desarrollo de GitHub y las consultas que otros programadores han tenido parecidas al desarrollo de esta tarea, se ha podido realizar la tarea planificada inicialmente, con un consumo de tiempo mayor.

En la ilustración 73 se puede observar una gráfica Brundown donde se puede ver la evolución del desarrollo del Sprint 7.

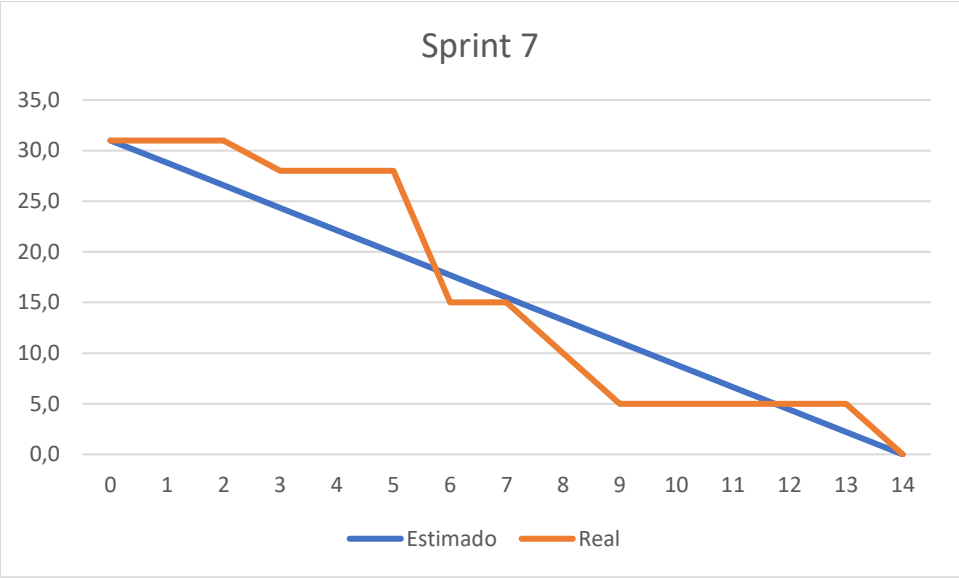

Ilustración 73 - Gráfico Brundown Sprint 7

8.8. Sprint 8

8.8.1. Planificación

Para el octavo Sprint se realiza una nueva planificación en la que se tendrán que acometer 33 puntos de historia donde en la tabla 30 se han identificado las tareas a realizar en este Sprint, además de su prioridad y la estimación.

Tareas		Prioridad	Story Points	Realizado	Sprint
Id	Titulo				
294	Adaptar proyecto a la nueva versión Blazor	M	1	N	8
288	Revisión de procesado de datos de los Gadgets	M	5	N	8
289	Registro de eventos del sistema	M	2	N	8
290	Consulta de eventos del sistema	M	3	N	8
190	Consultar localización Gadgets	C	5	N	8
195	Consultar localización de los Gadgets	C	5	N	8
201	Consultar localización de los Gateways	C	5	N	8
287	Mejora de iconos, pantalla inicial	C	4	N	8
291	Consulta de eventos del sistema	M	3	N	8
	Suma de puntos de historias		33		

Tabla 30 - Planificación Sprint 8.

8.8.2. Metas

Las metas propuestas en este Sprint consisten en adaptar el proyecto de Blazor, la nueva versión que hay disponible que actualmente es la versión 3.2.0 preview 2, que en principio debe ser más estable que las versiones anteriores.

También se ha de mejorar el procesado de los paquetes recibidos por los Gateway pues se ha observado que el sistema está dejando de procesar los paquetes ya que se está produciendo un error interno del sistema. Se ha de generar un sistema de registro de eventos para registrar los errores que se producen en el sistema, así como la información que pueda ser interesante para analizar en el procesado de datos. Se ha de poder visualizar los registros de los

eventos para poder hacer un seguimiento de estos, mediante las aplicaciones Front-End, tanto en la aplicación web como en la aplicación móvil y para poder analizar los fallos producidos por el sistema.

Además, se ha de poder visualizar la localización de los Gadget en la aplicación Front-End web y en la aplicación Front-End móvil para poder consultar la localización de los Gateways como de los Gadgets.

Por último, seria mejorar la presentación de la aplicación móvil, como poner una pantalla de inicio de aplicación y configurar los iconos de la aplicación.

8.8.3. Resultados

Para el desarrollo del octavo sprint se ha actualizado el proyecto Blazor a la última versión disponible, se ha mejorado el procesado de paquetes que recibe el Back-End y se registran los eventos producidos por este para poder ver los tipos de problemas que se producen, además de poder visualizarlos en las aplicaciones clientes. También se han añadido a las aplicaciones móviles la consulta de la geolocalización de los Gateways y los Gadgets.

Este Sprint se ha iniciado con la tarea Id 294, que ha consistido en actualizar el proyecto Blazor a una nueva versión, por lo que se ha tenido que volver a crear el proyecto de nuevo, creando de nuevo los proyectos **SensoremTerram.Server** y **SensoremTerram.Client**, en donde el proyecto **Client** se ha modificado la configuración de inicio de aplicación y adaptado a la nueva configuración. También se han actualizado las librerías de Syncfusion para Blazor a la nueva versión 18, que tienen un comportamiento más estable con Blazor que la versión anterior.

Seguidamente se ha procedido a realizar la tarea Id 289, que consiste en registrar los eventos del sistema, para ello se ha creado una nueva clase llamada **Event** en el proyecto **SensoremTerram.Entities** que contiene las siguientes propiedades:

- **EventId,** que identifica el evento.

- **Date,** que permite identificar la hora y fecha en que se ha producido el evento.
- **Type,** que utiliza el enumerador **EventType** que permite identificar el grado de importancia del evento. Los estados definidos son:
    - **Notification,** para eventos informativos.
    - **Warning,** para identificar eventos de advertencia en los procesos.
    - **Error,** identifica errores graves en los procesos.
    - **Critical,** identifica errores críticos del sistema como excepciones no controladas en el sistema.
- **Message,** donde se registra la información del mensaje.
- **DataLogId,** indica el paquete recibido al que tiene referencia el evento, no es obligatorio.

En el proyecto **SensotemTerram.DataAccess.SQLServer** se ha realizado la modificación de la estructura de la base de datos para registra la nueva tabla llamada **Events** y se ha creado las relaciones entre las diferentes tablas.

Una vez modificada la estructura de la base de datos para poder registrar los eventos en el sistema se procede con la tarea Id 288, donde se hace una revisión del procesado de los paquetes recibidos al servidor Back-End tanto de los Gateways como de los Gadgets. Al procesar los paquetes recibidos también se pretende registrar los procesos eventos del procesamiento del sistema. Con la experiencia adquirida en el desarrollo del proyecto se ha procedido a crear un nuevo sistema de procesamiento de los paquetes.

Para obtener la independencia de la base de datos como se ha hecho con el proyecto de la gestión de la base de datos se ha procedido a crear un nuevo proyecto llamado **SensoremTerram.Engine** donde se ha implementado la interfaz **IEngine** que define el método **RunAsync** que es el encargado en procesar los paquetes recibidos en el sistema.

Para la implementación del motor procesador de paquetes se ha creado un nuevo proyecto llamado **SensoremTerram.Engine.SQLServer** donde se

implementa la funcionalidad para el procesamiento de los paquetes recibidos en el servidor Back-End, para ello se han creado las siguientes clases como se puede ver en el diagrama de clases de la ilustración 74.

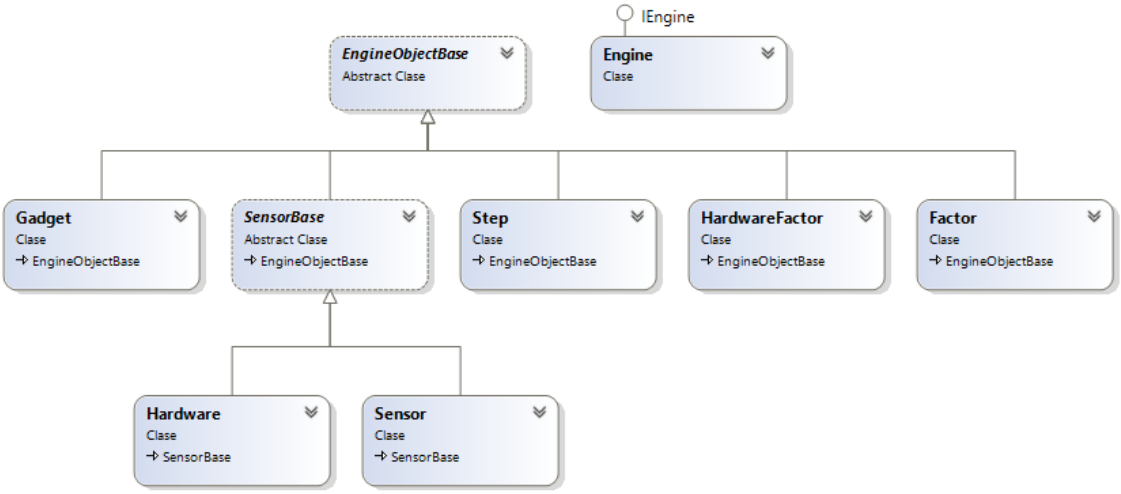

Ilustración 74 - Diagrama de clases de Engine.

A continuación, se realiza una explicación más detallada de la estructura del diagrama de clases:

- La clase **Engine,** que implementa la interfaz **IEngine**, se encarga de procesar los paquetes recibidos pendientes, inicialmente carga en memoria las configuraciones de Hardware y procesa los paquetes pendientes según la configuración establecida en cada Gadget.
- La clase abstracta **EngineObjectBase,** heredaran todos los objetos utilizados por la clase **Engine**.
- La clase **Factor,** hereda de la clase **EngineObjectBase** y que sustituye a la clase **FactorData** en el anterior procesado de los datos y esta clase contiene información sobre cada factor que tienen los sensores. Es decir, tiene propiedades como el orden en que aparece los datos en el paquete, el formato del dato, el factor de corrección, la unidad de medida y valor leído.
- La clase **Gadget,** hereda de la clase **EngineObjectBase** y sustituye a la clase **GadgetData**, en esta clase se ha cambiado bastante la estructura, pues ahora se almacenan todos los **Step** para analizar, además una nueva clase **Hardware** contiene la configuración del Hardware para

procesar los **Step**. Esta nueva clase almacena los **Step** que forman los paquetes pendientes de procesar además controla mejor la recepción de los paquetes comprobando que estos contienen la estructura correcta y si no son correctos son notificados por el nuevo sistema de eventos indicando los errores encontrados.

- La clase abstracta **SensorBase,** hereda de la clase **EngineObjectBase** que contiene varias propiedades que sirven para identificar los sensores y hardware, así como la longitud de los paquetes y si estos son válidos.
- La clase **Hardware,** hereda de la clase **SensorBase** y esta sustituye a la clase **HardwareData**, ahora esta nueva clase tiene más importancia en el procesado de los datos, pues se encarga de procesar los **Step** de los Gadget que tiene dicho **Hardware**, además ahora genera eventos notificando el proceso.
- La clase **Sensor,** hereda de la clase **SensorBase** y sustituye a la clase **SensorData**. Esta clase contiene los factores que dispone cada sensor. Además, permite cargar en memoria dichos factores para que la clase **Hardware** los utilice para procesar los **Step**.
- La clase **Step,** hereda de la clase **EngineObjectBase**, esta clase sustituye a la clase **StepData** y contiene todos los paquetes que forman parte del **Step**, permite cargar los paquetes en memoria que forma parte del **Step**, además de notificar con eventos los problemas que se producen a la hora de cargar los paquetes pendientes de procesar.

El sistema anterior en ocasiones no podía procesar paquetes con una estructura incorrecta, cuando el sistema tenía una excepción grave este dejaba de procesar los paquetes y si se dejaban varios días sin procesar, agrupándose paquetes en **Step** que no correspondían, produciendo que no se procesaran más paquetes. Con la nueva modificación también se ha optimizado el uso de la memoria, pues en lugar de que cada **GadgetData** tenía una clase **HardwareData**, ahora los Gadget comparten la clase Hardware que tiene más importancia a la hora de procesar los paquetes, pues es esta la encargada de procesar los datos recibidos de los Gadgets. Se han eliminado todas las clases sustituidas y aquellas que ya no hacen falta para el proceso de los paquetes.

También se ha modificado la estructura de la clase **DataLog** en el proyecto **SensoremTerram.Entities**, añadiendo una nueva propiedad llamada **DataLogGroupId**, donde los paquetes que pertenecen a un mismo **Step** quedan agrupados. Esto ha producido un cambio de la estructura de la base de datos.

La clase **ScopedProcessingService** se ha modificado simplificándola, ahora se encarga de llamar a la interfaz **IEngine** que se encargara de procesar los paquetes pendientes de procesar. Con el uso de la clase **Engine** se puede realizar una llamada cuando se quiera para procesar los paquetes, pudiendo lanzar el proceso desde la aplicación clientes cuando se considerada oportuno.

Como prueba del correcto funcionamiento del nuevo sistema de procesado de paquetes se han procesado todos los paquetes que se tienen almacenados en el sistema para comprobar lo estable que es, teniendo un mejor comportamiento, donde antes se producían excepciones y dejando en el sistema los eventos información sobre el procesamiento de los paquetes recibidos como se puede ver en la ilustración 75.

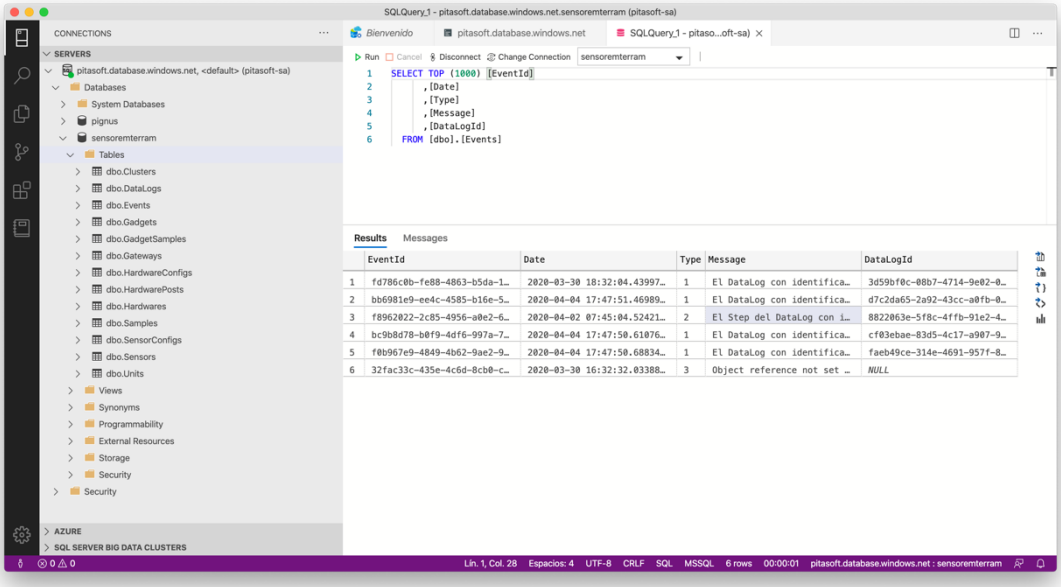

Ilustración 75 - Consulta de la tabla Event en SQL Server.

También en esta tarea se ha considerado añadir nuevas propiedades en la clase **Gadget** correspondiente al proyecto **SensoremTerram.Entities**, modificando la estructura de la base de datos, las propiedades añadidas son:

- Propiedad **Temperature,** que representa la temperatura interna del Gadget.
- Propiedad **Battery,** que representa el nivel de la batería registrada por el Gadget, que su valor es un enumerador llamado **LevelBattery** que indica diferentes estados de la batería como puede ser: Full, High, Midium, Low y Unknown.

Además de ha modificado la clase **Hardware** de la base de datos añadiendo las siguientes propiedades:

- Propiedad **MaximumBattery,** para el valor máximo de la batería.
- Propiedad **MinimumBattery,** para el valor mínimo de funcionamiento de la batería.

Se ha realizado un estudio de los datos recogidos por los Gadgets, se ha establecido como valor mínimo de la batería a 1.80 y máximo a 4.10, con estas propiedades determinamos el nivel de la batería de los Gadgets.

La tarea Id 290 que consiste, en poder consultar los eventos del sistema que se han producido en la aplicación cliente web. Para poder realizar la consulta en la base de datos, se ha procedido a definir la interfaz **IEventRepository** en el proyecto **SensoremTerram.DataAccess** para obtener los eventos del sistema.

Para realizar la consulta en la base de datos de los eventos, se ha procedido a implementar la interfaz **IEventRepository** en el proyecto **SensoremTerram.DataAccess.SQLServer**, donde se ha implementado la clase **EventRepository** que permite obtener los eventos producidos en el sistema. Así, en el servidor Back-End se ha implementado un nuevo controlador llamado **EventController** para que las aplicaciones cliente puedan acceder a los datos de los eventos.

Se ha realizado la página web en el proyecto **SensoremTerram.Client**, llamada **Events.razor** que permite consultar los datos de los eventos como se puede ver en la ilustración 76.

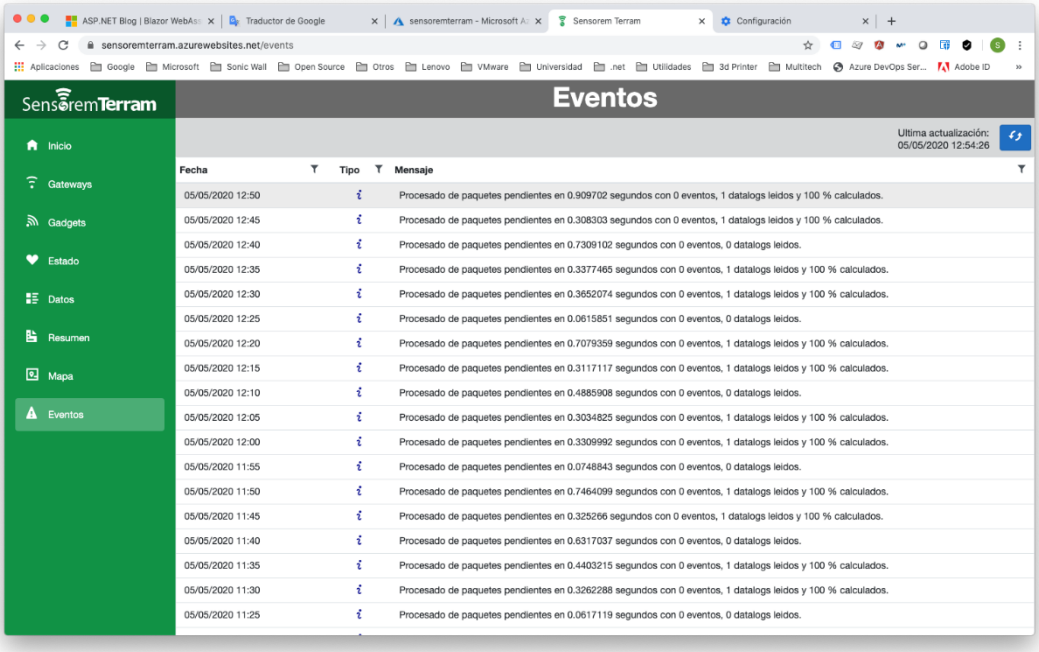

Ilustración 76 - Visualización de eventos en aplicación web.

Para la tarea Id 190, consultar la localización de los Gadgets se ha modificado la página **map.razor** que en el Sprint anterior solo mostraba los Gateways, para que también muestre los Gadget. Se ha creado una nueva interfaz llamada **IMapRepository** en el proyecto **SensoremTerram.DataAccess** para implementar las funciones necesarias para obtener la información para la visualización en el mapa. Concretamente se ha implementado el método **GetIconMapAsync** para obtener todos los iconos a visualizar. Este método devuelve una clase llamada **IconMapCollection** implementada en el proyecto **SensoremTerram.Models**. Esta clase devuelve una colección de la clase **IconMap** que contiene la información necesaria para la visualización de los datos en el mapa.

En el proyecto **SensoremTerram.DataAccess.SQLServer** se ha implementado la interfaz con la clase **MapRepository**, devolviendo los datos a visualizar. Además de implementar el controlador en el servidor Back-End para

que este devuelva los datos solicitados por las aplicaciones clientes, ya sean la aplicación web o la aplicación móvil.

Para la visualización de los datos en pantalla utilizando las librerías **SensoremTerram.Google** creada en el Sprint anterior, se ha modificado el componente de visualización llamado **Map.razor** del proyecto **SensoremTerram.Client**, para visualizar los elementos, además si se pulsa sobre el icono que representa el elemento muestra una ventana de información como se puede ver en la ilustración 77, mostrando los iconos con diferente color dependiendo si es una Gadget o un Gateway.

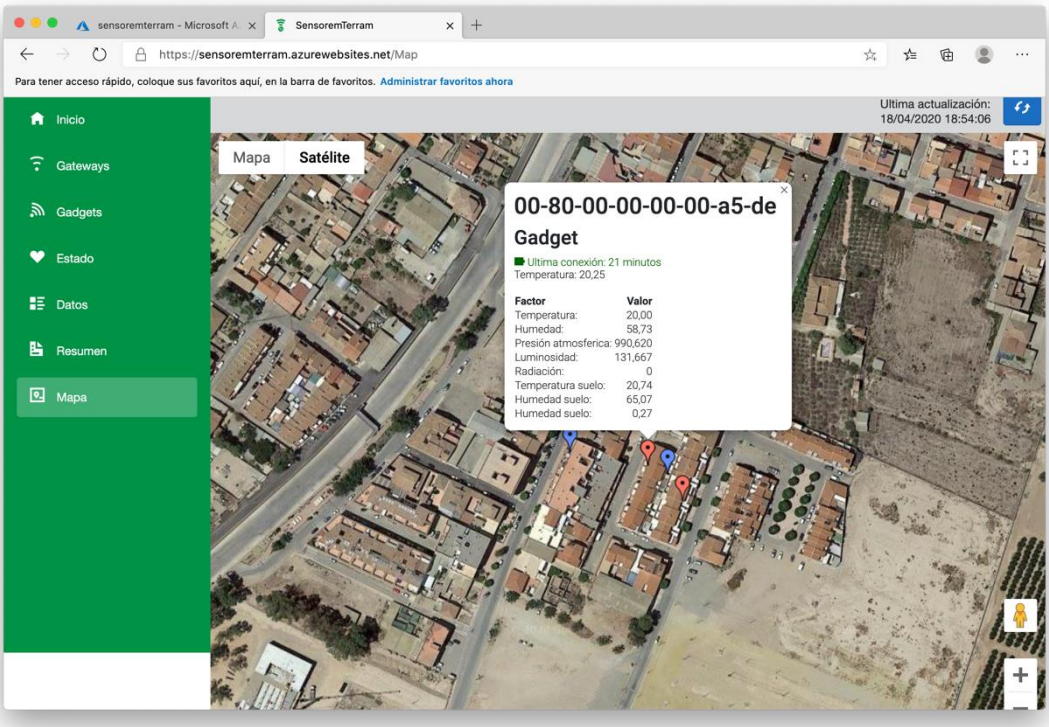

Ilustración 77 - Visualización de Gadgets en mapa en aplicación web.

Las siguientes tareas realizadas han sido la tarea 195 y 201, que se han realizado juntas, pues se ha implementado el controlador del servidor Back-End devolviendo los datos, tanto de los Gadget como de los Gateways. Para la obtención de datos del servidor se ha implementado la interfaz **IMapService** en la clase **SensoremTerramRestService**, así se pueden obtener los datos del servidor.

Para la implementación de mapa en las aplicaciones clientes se ha realizado una personalización del control del Map ofrecido por Xamarin, basándose en la documentación ofrecida por Xamarin para renderizar un control mapa (Microsoft Docs, 2019). Para la personalización del control, se ha creado una nueva clase llamada **IconMapView** en el proyecto **SensoremTerram.App** en la carpeta **Controls** dentro de **Views**, donde se han implementado las propiedades y métodos necesarios para la visualización de los Gadgets y Gateways en el mapa.

También se han tenido que implementar las clases en cada uno de los sistemas, es decir, en Android, iOS y Windows para poder visualizar los puntos de interés como las ventanas de visualización de información, que detallamos a continuación:

- Para Android, se han creado los siguientes objetos:
    - Clase **IconMapViewRenderer**, que implementa la funcionalidad del control personalizado de Xamarin para la visualización del mapa en Android. Se ha modificado el comportamiento para cambiar los tipos de iconos además de añadir la funcionalidad para visualizar las viñetas con la información de los Gadgets y Gateways.
    - Layout **GadgetMapInfoWindows,** que muestra la información de los Gadgets.
    - Layout **GatwayMapInfoWindows,** muestra la información de los Gateways.
- Para iOS, se han creado los siguientes objetos:
    - Clase **IconMapViewRenderer,** que es la implementación del control personalizado de Xamarin para la visualización del mapa en iOS.
    - Interfaz **ILoadIconMap**, implementa un método llamado **LoadIconMap** que permite pasar los datos a las vistas para visualizar los datos correctos.
    - Vista **GadgetDetailMapView,** que representa la visualización de la información de un Gadget.

  - Clase **GadgetDetailMapView**, contiene la lógica de la vista **GadgetDetailMapView** e implementa la interfaz **ILoadIconMap**.
  - Vista **GatewayDetailMapView,** que representa la visualización de la información de un Gateway.
  - Clase **GatewayDetailMapView**, contiene la lógica de la vista **GatewayDetalMapView** e implementa la interfaz **ILoadIconMap**.
  - Clase **IconMapMKAnnotationView,** que hereda de la clase **MKAnotationView**, que permite la gestión de los iconos de un mapa de Apple y con esta clase se modifica el comportamiento de visualización y muestra las viñetas con la información de los Gadgets y Gateways.
- Para Windows, se han creado los siguientes objetos:
  - Clase **IconMapViewRenderer,** representa la implementación del control personalizado de Xamarin para la visualización del mapa en una aplicación UWP. Se ha implementado la visualización de los iconos con la representación de los Gadgets y Gateways además de activar la viñeta con la información de estos cuando son pulsados.
  - Vista **GadgetDetailMapView**, que permite visualizar la información de un Gadget.
  - Vista **GatewayDetailMapView,** sirve para visualizar la información de un Gateway.

Como resultado se ha obtenido la ilustración 78 donde se puede ver la visualización del mapa con la localización de los Gadget y Gateways en Android y iOS.

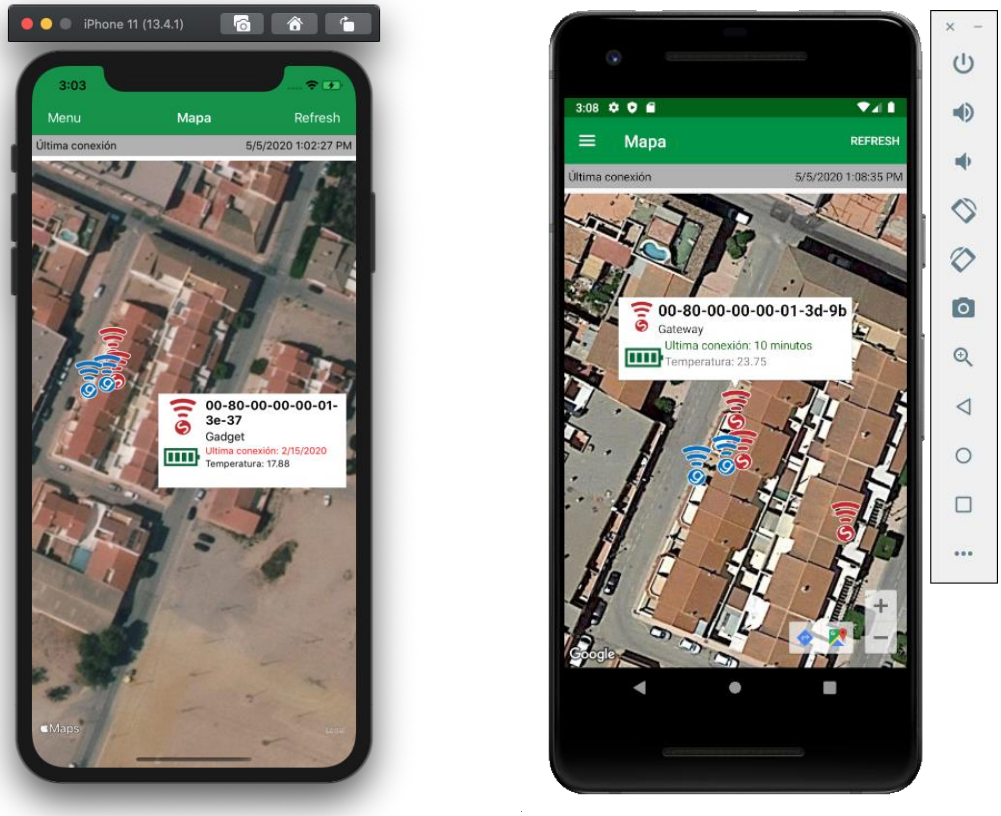

Ilustración 78 - Visualización de los elementos aplicación móvil. Imagen de la izquierda iOS. Imagen de la derecha Android.

Además, se ha creado una nueva vista llamada **GadgetView** y su vista modelo correspondiente llamada **GadgetViewModel**, que permite visualizar la información detallada de los últimos datos recibidos de un Gadget cuando se pulsa sobre la viñeta de información de un Gadget en un mapa, obteniendo la siguiente la ilustración 79.

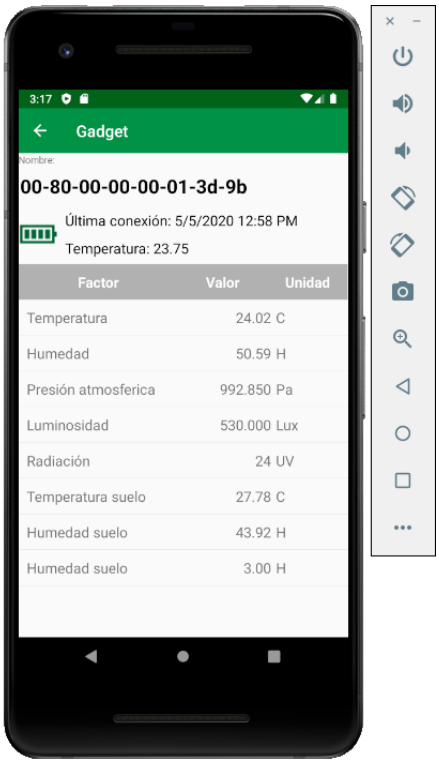

Ilustración 79 - Visualización de detalle de Gadget.

Para la tarea Id 291, consulta de los eventos del sistema en la aplicación en dispositivos móvil, se ha implementado una nueva vista llamada **EventsView** y su correspondiente vista modelo **EventViewModel** donde se ha implementado toda la lógica para la visualización de los eventos. Para la obtención de los datos se ha implementado la interfaz **IEventRepository** en la clase **SensoremTerramRestService**, así se dispone de la comunicación con el servidor Back-End para obtener los datos del servidor.

Además, para la visualización de los iconos se necesitado crear la clase **TypeEventConverter** que implementa la interfaz **IValueConverter** de Xamarin, que permite para un tipo de evento obtener una imagen indicando el icono del error, en la ilustración 80 se puede ver una muestra la pantalla donde se visualizan los eventos del sistema.

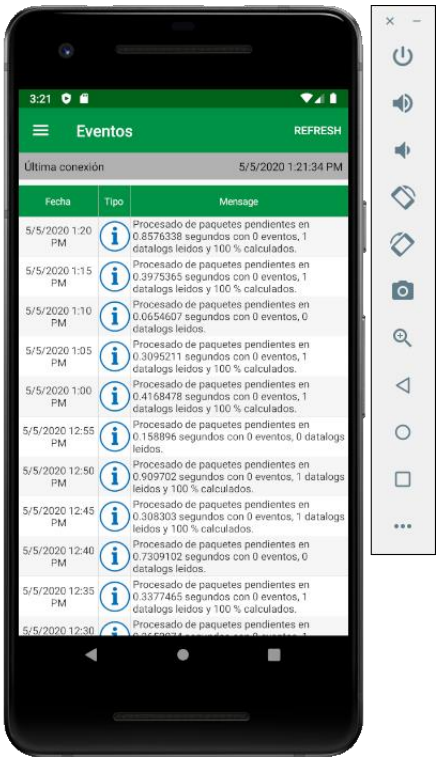

Ilustración 80 - Visualización de eventos en aplicación móvil.

La última tarea de este sprint es la tarea Id 287 que consiste en poner los iconos de la aplicación y poner una pantalla inicial al arrancar la aplicación y mejorar el funcionamiento del tamaño de pantalla.

Primero se han creado diferentes iconos para cada plataforma, como el icono de la aplicación, el icono de los Gadget, el icono de Gateways, así como diferentes iconos de baterías con Adobe Ilustrador. También, para el icono de aplicación, se han creado los tamaños para cada una de las diferentes plataformas, como se puede ver en la ilustración 81.

Ilustración 81 - Icono de la aplicación. A la izquierda en iOS. A la derecha Android.

A continuación, se ha creado la pantalla de inicio de aplicación como se puede ver en la ilustración 82, para ello se ha tenido que configurar diferentes métodos dependiendo de la plataforma, que detallamos a continuación:

- Para Android, se ha creado una layout llamada **splash_screen.xml**, donde se ha diseñado la pantalla inicial, además de añadirla a la actividad principal para su visualización.
- Para iOS, se ha modificado el archivo **LaunchScreen.storyboard** para adaptar a nuestro diseño deseado, además de adaptarlo a diferentes tipos de dispositivos.
- En UWP, solo se ha tenido que configurar el icono en el manifiesto.

Ilustración 82 - Pantalla de inicio de aplicación

8.8.4. Revisión

Los resultados de este Sprint han sido los esperados, se pretendía actualizar el proyecto del cliente Web desarrollado por Blazor a una versión más moderna. Se ha actualizado el proceso que se encarga de procesar los paquetes recibidos por los Gadget, emitiendo eventos para conocer si el sistema está procesando los paquetes recibidos y si se están produciendo incidencias.

Además, en las aplicaciones clientes, tanto en la aplicación web como en las aplicaciones móviles, se puede consultar los eventos, para hacer el seguimiento.

La aplicación web se ha modificado para poder consultar la localización de los Gadgets y poder visualizar información detallada de estos. También en la aplicación móvil se ha añadido la posibilidad de consultar la localización de los Gadgets, así como de los Gateways, además de poder consultar los detalles de los Gadgets.

Por último, se ha mejorado la estética de la aplicación, añadiendo iconos y mejorando la apariencia de la aplicación.

También se ha detectado que cuando un sensor se estropea el Gadget envía datos incorrectos al sistema pues por lo que se ha decidió añadir una nueva tarea para mejorar el comportamiento ante esta situación, que podemos ver en la tabla 31.

Cuestiones		Tareas		Prioridad	Story Points	Realizado	Sprint
Id	Titulo	Id	Titulo				
208	Energía	296	Detectar funcionamiento de los sensores	M	3	N	

Tabla 31 - Nuevas tareas definidas en el Sprint 8 Revisión.

No ha quedado ninguna tarea pendiente para hacer en el siguiente Sprint.

8.8.5. Retrospectiva

En la tabla 32 se puede ver la comparativa entre los puntos de historia estimados con los reales para la realización del octavo Sprint.

Tareas		Prioridad	Story Points	Realizado	Sprint	Story Points Reales
Id	Titulo					
294	Adaptar proyecto a la nueva versión Blazor	M	1	N	8	1
288	Revisión de procesado de datos de los Gadgets	M	5	N	8	8
289	Registro de eventos del sistema	M	2	N	8	2
290	Consulta de eventos del sistema	M	3	N	8	3
190	Consultar localización Gadgets	C	5	N	8	6
195	Consultar localización de los Gadgets	C	5	N	8	5
201	Consultar localización de los Gateways	C	5	N	8	5
287	Mejora de iconos, pantalla inicial	C	4	N	8	2
291	Consulta de eventos del sistema	M	3	N	8	3
	Suma de puntos de historias		33			35

Tabla 32 - Comparación de los puntos de historia estimados y reales del Sprint 8

Como se puede observar se han producido alguna diferencia respecto a la estimación inicial con la real para concluir el Sprint.

Para la realización de la tarea Id 294, no se han producido diferencias, pero la nueva versión de Blazor no funciona correctamente hasta la espera de que saque una nueva versión, pues se ha tenido que desactivar alguna función que no limita la aplicación, que el equipo de desarrollo de Microsoft ya tiene constancia y en una futura versión estará solucionado.

Para la tarea Id 289, se ha empleado el tiempo establecido, pues lo único era crear la nueva estructura de datos para guardar los eventos. Mientras que para la tarea Id 288, se ha consumido más tiempo, con el nuevo desarrollo, se han producido errores a la hora de guardar las transacciones en la base de datos, no generando los eventos en la base de datos, por lo que se ha establecido diferentes contextos para evitar el problema.

Para la tarea Id 290, se ha realizado la página web que permite consultar los eventos del sistema, consumiendo el tiempo establecido.

Las tareas Id 190 y 195 se han agrupado para realizarlas juntas, como al final del controlador se obtiene los datos de Gadgets y Gateways. Se ha consumo más tiempo, pues se ha tenido que implementar el controlador del Mapa en las diferentes plataformas y la parte de desarrollo de iOS ha sido más complicada por falta de documentación.

Para la tarea Id 291 se ha creado la pantalla para los dispositivos móviles que permite consultar los eventos del sistema.

Por último, para la tarea Id 287, se han creado los iconos de la aplicación, logotipo de la aplicación, iconos de baterías, iconos para la visualización de los mapas, así como las pantallas iniciales de la aplicación.

En la ilustración 83 se puede observar una gráfica Brundown donde se puede ver la evolución del desarrollo del Sprint 8.

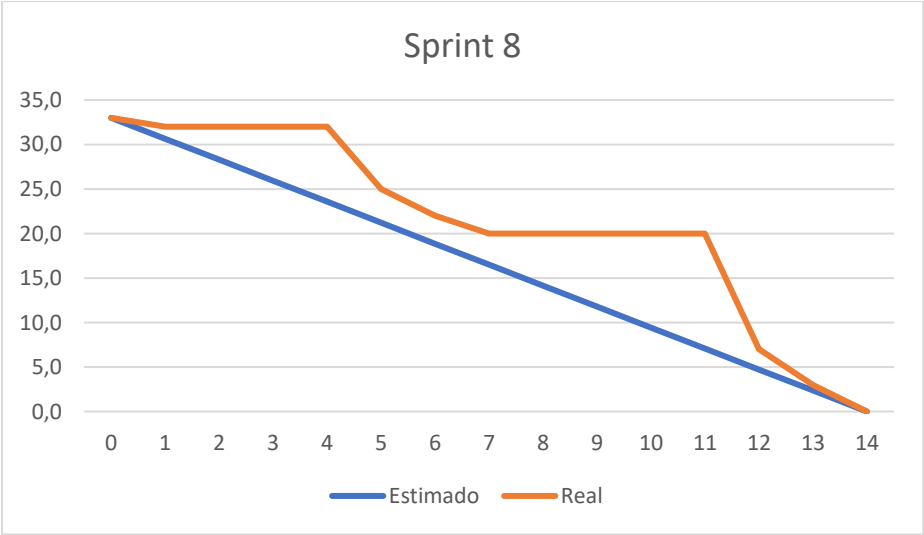

Ilustración 83 - Gráfico Brundown Sprint 8

8.9. Sprint 9

8.9.1. Planificación

Este es el último Sprint del TFG en el que se tendrán que acometer 28 puntos de historia, en la tabla 33 se han identificado las tareas a realizar en este Sprint, además de su prioridad y estimación. Como se puede observar, se ha añadido una nueva tarea en la planificación, Microsoft ha publicado la versión RC1 de Blazor, se ha decidido añadir esta una nueva tarea para la actualización del producto a una versión más estable que las preview anteriores.

Tareas		Prioridad	Story Points	Realizado	Sprint
Id	Titulo				
295	Actualizar proyecto Blazor RC1	M	1	N	9
231	Mejora de interfaz de usuario	M	8	N	9
292	Mejorar el hardware para despertar GPS	W	2	N	9
293	Mejorar el software STEP y despertar GPS	W	5	N	9
296	Detectar funcionamiento de los sensores	M	2	N	9
219	Controlador mantenimiento de Gadgets	C	5	N	9
215	Mantenimiento de Gadgets	C	5	N	9
	Suma de puntos de historias		28		

Tabla 33 - Planificación Sprint 9.

8.9.2. Metas

Como metas para este último Sprint se tiene que actualizar el proyecto a la nueva versión que ha publicado Microsoft de Blazor a la versión RC1, que, aunque la última versión está funcionando correctamente y hubo algún error que otro esta versión ha de ser más estable pues se encuentra cerca de la versión final.

También se tiene que mejorar la interfaz de usuario de la aplicación web, pues en las revisiones anteriores se comentó la mejora para que la aplicación tenga un mejor comportamiento responsivo, además de añadir las unidades de medida de los factores.

Como última meta a cometer en el proyecto se ha de mejorar el hardware para poder despertar el módulo GPS mediante una señal de WakeUp, además de modificar el software del Gadget para que este puede despertar el módulo GPS y poderlo poner en modo ahorro energía ya que este tiene un consumo elevado de energía cuando está en pleno funcionamiento.

8.9.3. Resultados

Para el desarrollo de este último Sprint se ha procedido actualizar el proyecto a la última versión publicada por Microsoft de Blazor, que es la versión RC1, para solucionar problemas que se han tenido en el Sprint anterior, esta es la tarea Id 295.

Esta tarea ha consistido en volver a crear de nuevo los proyectos de **SensoremTerram.Sever** y **SensoremTerram.Client**, quitando los proyectos anteriores y copiar los archivos necesarios para que estos proyectos funcionen correctamente. En el uso de esta nueva versión no se ha detectado ningún problema en la implementación, además ahora la nueva versión consigue mejor estabilidad en la publicación del servidor, pudiendo instalar la aplicación web como si fuera una aplicación de escritorio. En versiones anteriores fue introducida, pero se tuvo que eliminar la característica por problemas de implantación en el servidor.

La siguiente tarea realizada Id 292, ha consistido en añadir una salida digital del módulo MultiTech mDot hacia el módulo GPS, en la entrada WakeUp, para que esté pueda activarla y así poder despertar el GPS. En la ilustración 84, se puede ver la modificación que se ha producido en el hardware.

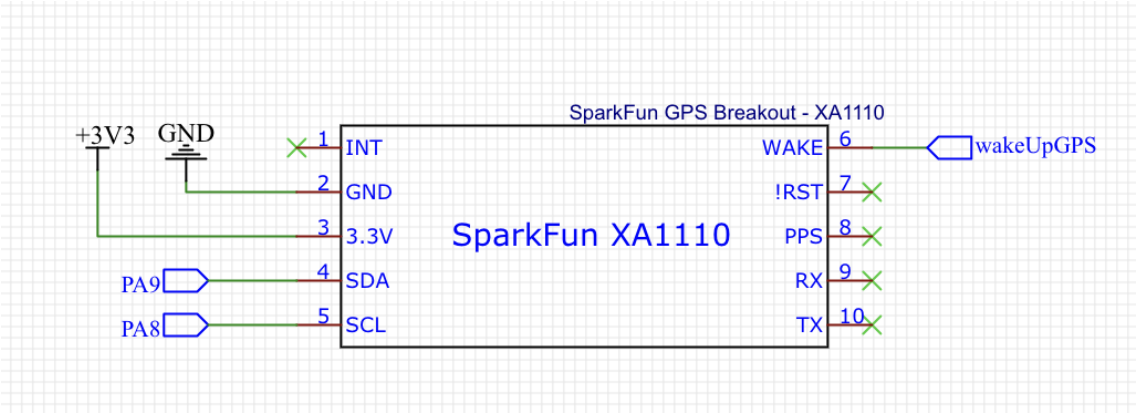

Ilustración 84 - Modificación conexión módulo GPS.

En la tarea Id 296, se ha procedido a modificar la aplicación del Gadget, para implementar la activación de la señal para poder despertar el GPS, para que este pueda coger datos del posicionamiento y poder leerlos antes de volver a ponerlo en modo ahorro energía para que la batería dure más.

Para ello, se ha procedido a modificar el valor máximo del Step, que ha pasado a ser en lugar de 100 a 200 pasos, creando una constante llamada CYCLE en el archivo **GadgetBase.h** que permita cambiar el número de Step en el proyecto sin tener que modificar el código, además se ha modificado la clase abstracta **GadgetBase** para usar dicha constante en el proceso de ejecución del Gadget. Así se disponen más pasos, que sirven para aumentar el tiempo de estado suspendido del GPS. También se ha añadido un nuevo método llamado **getBuffer** para obtener el buffer donde se almacenan los datos acumulados en cada paso, así que la clase **GadgetBase** dispone de un buffer, que inicialmente no disponía.

Se ha modificado la clase abstracta **StepBase** donde ahora el método run recibe dos parámetros, en lugar de uno, ahora también recibe un **SampleBuffer**, para que en cada paso se puede añadir datos en el buffer y no tiene que ser un sensor.

También se ha modificado la clase abstracta **SensorBase**, en la que se han añadido nuevos métodos que detallamos a continuación:

- Método **enabled** para determinar si el sensor este activo.

- Método **isI2C** para determinar si el sensor está conectado al bus I2C.

Además, se han modificado los siguientes métodos:

- Método **read**, ahora en lugar de pasar un **SampleBuffer**, se le pasa un **GadgetBase**.
- Método **sleep** ahora recibe como parámetro un **GadgetBase**, para así tener información de las condiciones del Gadget.
- Método **wakeUp**, también se le pasa como parámetro una clase **GadgetBase**, para tener también información del Gadget.

Se ha creado una nueva clase abstracta llamada I2CBase, donde se ha implementado los métodos que permitan comunicar con el bus I2C, definiendo los siguientes métodos:

- Método **exists**, permite indicar si para una dirección del bus I2C existe el dispositivo.
- Método **getSlaveAddress**, permite obtener la dirección del módulo hardware que está conectado al bus I2C.
- Método **readI2C** que permite la lectura de una cantidad de bytes del bus I2C, este tiene tres métodos sobrecargados con diferentes formas de leer datos del bus I2C.
- Método **writeI2C** con el que se puede mandar datos al bus I2C, que también tiene tres métodos sobrecargados con diferentes formas de enviar datos al bus I2C.

También se ha modificado la clase **SensorI2CBase**, que ahora hereda de la clase **I2CBase**. Además, a la clase **Sensors** se le ha añadido el método **I2CScaner**, que permite analizar para los sensores declarados en el Gadget hacer un barrido del bus I2C para determinar cuáles de estos están activos, y los que no dejarlos deshabilitados.

Para la tarea Id 293 se ha seguido modificando el software del Gadget, para ello se ha modificado la clase **Sensor**, donde se han modificado los

métodos: **readSensors**, **sleepSensors** y **wakeUpSensor**. Ahora se le pasa como parámetro la clase **GadgetBase**.

Para mejorar el ahorro de energía, se ha modificado la clase I2CGPS, donde se ha sobrescrito el método **wakeUp** donde para un Step determinado se despierta el sensor, para que este capture los datos del sensor sobre un Step y así poder leer los datos del GPS y en la ejecución del siguiente Step, este será puesto en modo ahorro energía.

Para despertar el módulo GPS, lo que se ha hecho es habilitar una salida del módulo MultiTech mDot, que llamando al nuevo método llamado **enableWakeUpPin**, activa una salida digital sobre unos 2 segundos para despertar el módulo GPS.

En la tarea Id 231, se ha modificado la aplicación cliente web, en la que se ha actualizado las librerías Syncfusion, además del archivo CSS, para mejorar la adaptabilidad de la aplicación web a diferentes pantallas. También se han añadido en las pantallas las unidades de medida, para conocer en que se están midiendo los factores y las tablas dibujadas de todas las paginas se han mejorado, pues ahora al utilizar el control Grid de las librerías Syncfusion en lugar de utilizar la meta etiqueta Table de html, da mejor comportamiento la aplicación además de poder ordenar y utilizar filtros para realizar búsquedas, mejorando la usabilidad de la aplicación, como se puede apreciar en la ilustración 85.

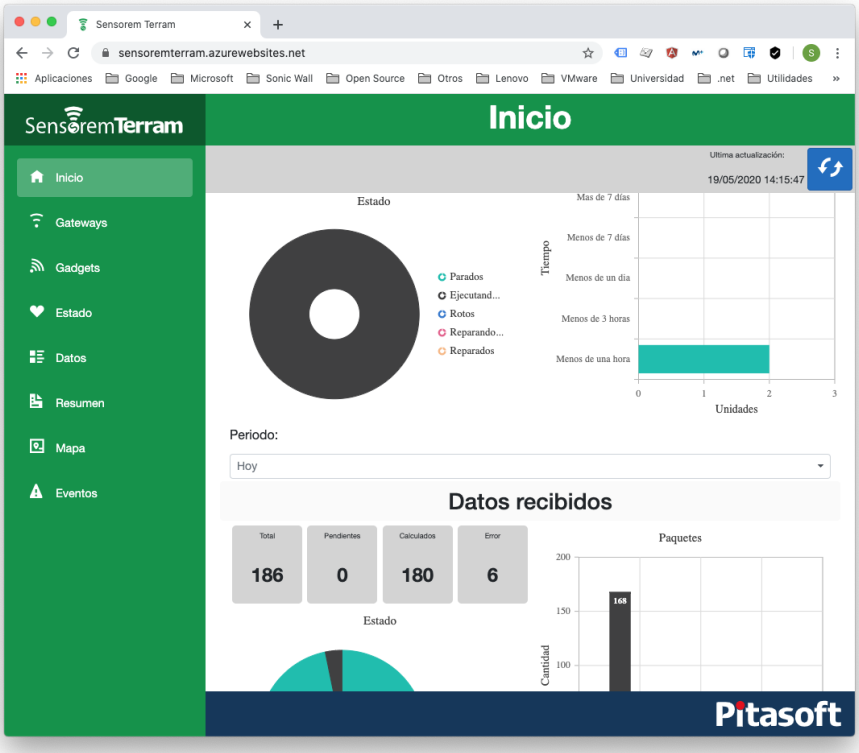

Ilustración 85 - Mejora responsive de aplicación web.

Para la tarea Id 219, se ha modificado la interfaz **IGadgetRepository** del proyecto **SensoremTerram.DataAccess**, añadiendo a la definición de los siguientes métodos:

- Método **PutGadgetAsync** para permitir modificar información de los Gadget.
- Método **ChangeGadgetStateAsync** para cambiar el estado del Gadget.

Así que se ha tenido que definir los métodos en las clases que implementen dicha interfaz. También se ha modificado el servidor Back-End para añadir la funcionalidad de mantenimiento de Gadget, se ha modificado en la clase **GadgetRepository** implementado los dos métodos en el proyecto **SensoremTerram.DataAccess.SQLServer**. Además de modificar el controlador **GadgetContoller** del proyecto **SensoremTerram.Server** añadiendo dos métodos llamados **Put** y **PutStatus** que implementa la acción PUT en la interfaz API REST para actualizar los datos del Gadget.

Por último, se ha realizado la tarea Id 215, en la que se ha añadido a la aplicación web la opción de poder modificar información de los Gadget, así como cambiarlos de estado, es decir, ponerlos activo o en modo reposo. Para realizar dicho cambio se ha modificado el archivo **Gadget.razor** implementando la funcionalidad y llamando al servidor Back-End para actualizar los datos. En la ilustración 86 se puede ver el cuadro de dialogo para modificar datos de los Gadgets.

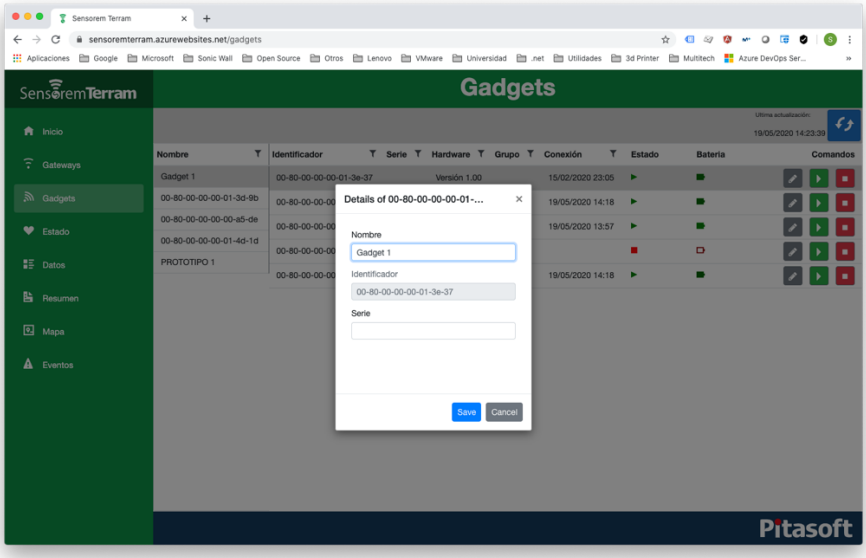

Ilustración 86 - Pantalla de modificación de Gadget.

8.9.4. Revisión

Los resultados de este Sprint han sido los esperados, se pretendía mejorar el consumo de energía del Gadget, ahora solo se leen los datos de los Gadget cada 200 Step, que sería más o menos sobre unas 50 horas, como el Gadget una vez puesto en un cultivo no se debería mover mucho, es un tiempo adecuado para conocer su posición.

Además, se ha mejorado el funcionamiento de la lectura de datos de los sensores, pues cuando un sensor deja de funcionar, el sistema sigue capturando los datos de dicho sensor siendo estos datos incorrectos. Ahora en la inicialización del Gadget comprueba la existencia de los sensores y desactiva los que no están presentes, así el Gadget solo lee datos de sensores que están activos y envía solo estos datos.

Por último, se ha mejorado la adaptabilidad de la aplicación web a diferentes tipos de pantallas. Por lo que no se ha quedado ninguna tarea pendiente y se da por concluido el TFG, aunque siempre habrá cosas para mejorar y el uso de este sistema analizando los datos va a permitir mejorar la aplicación.

8.9.5. Retrospectiva

En la tabla 34 se puede ver la comparativa entre los puntos de historia estimados con los reales para la realización del octavo Sprint.

Tareas		Prioridad	Story Points	Realizado	Sprint	Story Points Reales
Id	Titulo					
295	Actualizar proyecto Blazor RC1	M	1	S	9	1
231	Mejora de interfaz de usuario	M	8	S	9	8
292	Mejorar el hardware para despertar GPS	W	2	S	9	2
293	Mejorar el software STEP y despertar GPS	W	5	S	9	2
296	Detectar funcionamiento de los sensores	M	2	S	9	4
219	Controlador mantenimiento de Gadgets	C	5	S	9	1
215	Mantenimiento de Gadgets	C	5	S	9	4
Suma de puntos de historia			28			22

Tabla 34 - Comparación de los puntos de historia estimados y reales del Sprint 9

Se han producido diferentes en este último Sprint respecto a lo estimado, se procede a analizar a continuación.

Para la tarea Id 295, se ha consumido el tiempo estimado, se ha creado el proyecto, se han copiado los archivos necesarios y todo ha funcionado correctamente.

Para la tarea Id 292 se ha modificado el hardware para añadir la salida del MultiTech mDot que conecta con el pin WakeUp del módulo GPS, en los tres modulo Gadget fabricados.

Para la tarea Id 296, se ha necesitado más tiempo para la modificación de todas las clases para adaptar a los nuevos requisitos establecidos, consumiendo más tiempo para verificar el correcto funcionamiento del Gadget, además se ha actualizado las librerías MultiTech instalando la última versión y modificando algo de código para adaptar el software a estas.

Para la tarea Id 293, se ha necesitado menos tiempo, pues con las modificaciones de la tarea Id 296 la modificación ha sido menor.

En la tarea Id 231, se ha necesitado más tiempo para mejorar el comportamiento de la aplicación web respecto a la adaptabilidad de diferentes pantallas, así como poner las unidades de medida para cada uno de los factores de los datos capturados por los sensores.

La tarea Id 219, se ha necesitado menos tiempo que el estimado, pues solo se ha definido el método para modificar los datos en la base de datos.

Por último, se ha realizado la tarea Id 215, que también ha necesitado menos tiempo para su realización, en la que se permite modificar algunos datos del Gadget, así como poder activarlo o desactivarlo.

En la ilustración 87 se puede observar una gráfica Brundown donde se puede ver la evolución del desarrollo del Sprint 8.

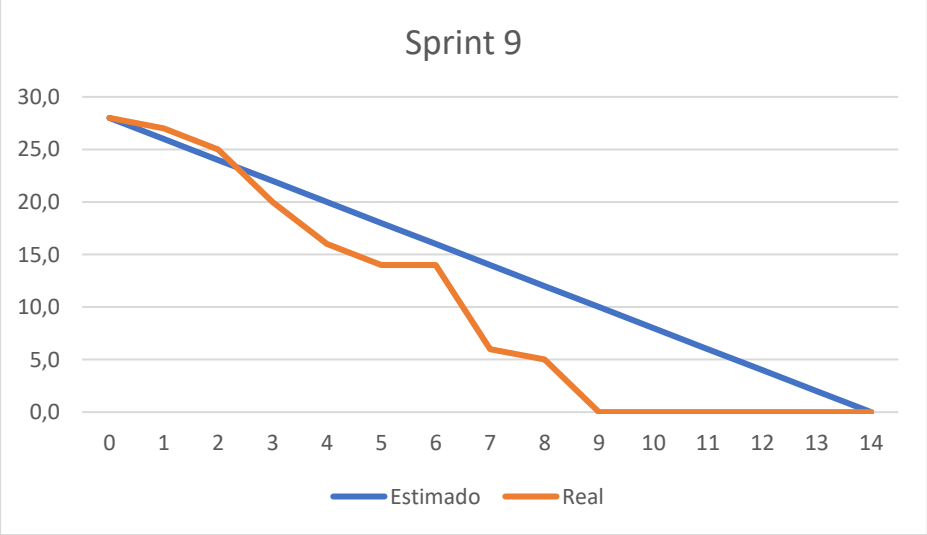

Ilustración 87 - Ilustración 73 - Gráfico Brundown Sprint 9

9. Despliegue y prueba de la solución

9.1. Plan de pruebas

Las pruebas se han ido realizando en el desarrollo de cada uno de los Sprint, observando que los resultados de las tareas especificadas han tenido los resultados adecuados a los establecido en la especificación de cada una de las tareas definidas en el Backlog.

Ha medida que se ha ido desarrollando cada tarea se ha verificado que se obtenido los resultados deseados, como se ha podido apreciar en cada tarea, y añadiendo en el Backlog nuevas tareas para mejorar los resultados cuando esto no han sido satisfactorios.

9.2. Despliegue de la solución

Para el despliegue de la solución se ha utilizado diferentes plataformas como se ha comentado anteriormente, que analizamos a continuación.

Para la implementación del Gadget, se ha de compilar la aplicación modificado en el archivo **main.c**, se ha de añadir la directiva de compilador **#define NDEBUG**, para eliminar todos los controles de depuración establecidos en el código. Una vez compilado el software, solo tiene que copiar en el MultiTech mDot el archivo obtenido de la compilación llamado **Gadget.bin** que se encuentra en la carpeta BUILD. Una vez conectado a la protoboard este comienza a capturar los datos de los sensores conectados.

Para el despliegue del Gateway se ha utilizado la plataforma ofrecida por MultiTech DeviceHQ (MultiTech, 2020). Esta ofrece un conjunto de herramientas basadas en la nube, que permite gestionar los dispositivos de MultiTech además de poder desplegar aplicaciones desarrolladas en Node-RED o incluso aplicaciones de terceros, como se puede ver en la ilustración 88. En nuestro caso, se ha subido la aplicación desarrollada el MultiTech Conduit a dicha plataforma y así poder implementarla en otros Gateways.

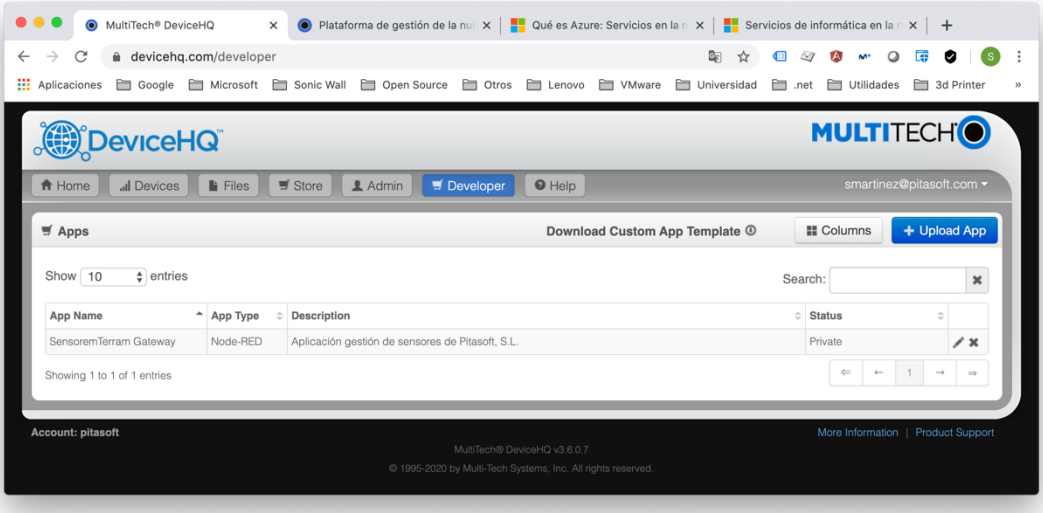

Ilustración 88 - Implementación de aplicaciones Node-RED.

Seleccionado un Gateway se puede implementar la aplicación con un solo clic, como se puede ver en la ilustración 89.

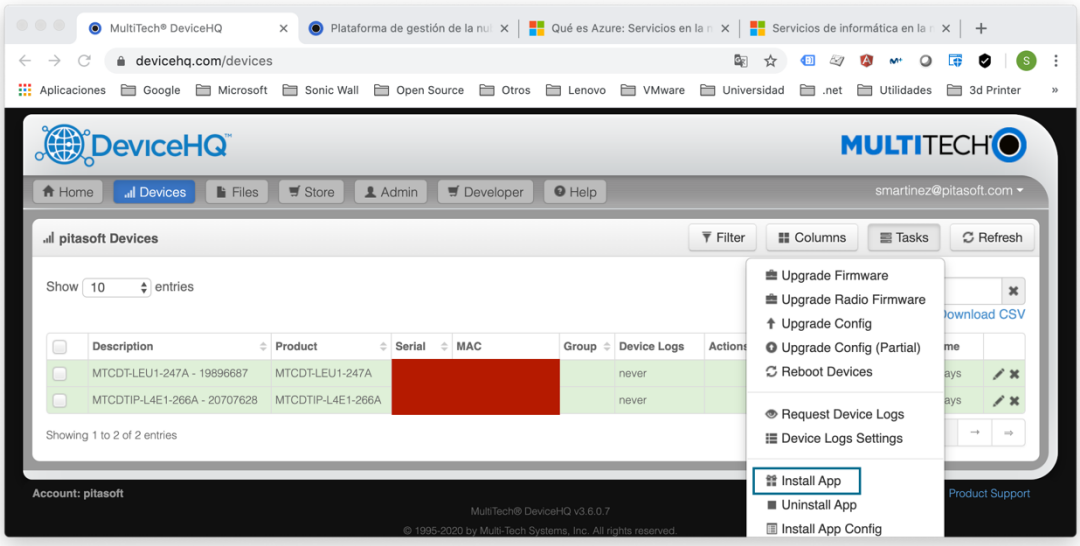

Ilustración 89 - Implementación aplicación en le Gateway.

También se tienen que configurar los parámetros de conexión del protocolo LoRaWAN para que los Gadget pueden enviar los paquetes a los Gateway. Se debe configurar la red con el canal adecuado, ya que al estar en España se utiliza el plan europeo. Además de configurar el modo de funcionamiento de la red, público o privado, como el identificador de red y la clave de acceso de esta. En el siguiente enlace hay un video explicativo de cómo realizar la configuración (MultiTech Developer Resources, 2020).

Ora forma de implementación de la aplicación en el hardware Conduit seria la importación del archivo que contiene el código de la aplicación Node-RED mediante la plataforma web que dispone el propio hardware y al compilarlo este se pone en marcha automáticamente.

Para el servidor Back-End y cliente web, como se ha comentado anteriormente se ha utilizado la plataforma Microsoft Azure (Microsoft, 2020), en ellas se ha creado una base de datos SQL y una App Service, como se explicó en el Sprint 3. Una vez creados los recursos en la plataforma Microsoft Azure se deben configurar tres parámetros para el correcto despliegue, estos son:

- La cadena de conexión a la base de datos que se encuentra en el archivo **appsettings.json** dentro del proyecto **SensoremTerram.Server.**
- Configurar la licencia de Syncfusion en el archivo **Program.cs** en el proyecto **SensoremTerram.Client**.
- Configurar la licencia del Api de Google Map en el archivo index.html dentro de la carpeta **wwwroot** del proyecto **SensoremTerram.Client**.

Una vez configurados los parámetros se ha procedido a publicar la aplicación mediante el comando de Visual Studio Publish como se puede ver en la ilustración 90 y este se encarga de publicar dicha aplicación, o copiar los archivos binarios mediante FTP al servidor de Azure.

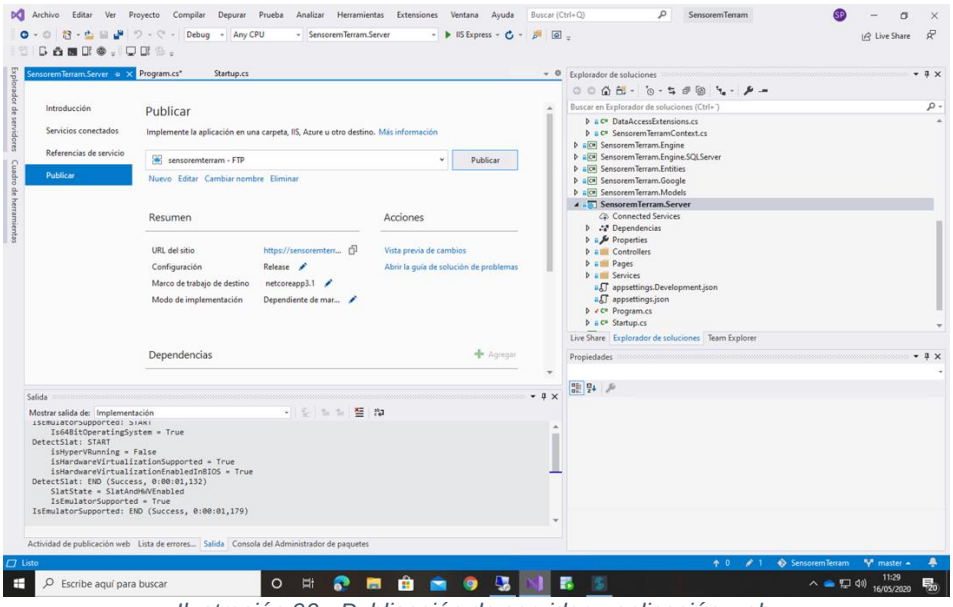

Ilustración 90 - Publicación de servidor y aplicación web.

Para acceder a la aplicación web, se accede en el siguiente enlace: https://sensoremterram.azurewebsites.net/. Al iniciarse por primera vez la aplicación, esta crea la base de datos, si no ha sido implementada en Azure y hace las configuraciones necesarias para el correcto funcionamiento de la misma, los dispositivos como los Gadgets y los Gateways una vez puestos en funcionamiento se configuran automáticamente en el sistema, por lo que irán apareciendo automáticamente.

Para la publicación de la aplicación móvil, tanto en Android como en iOS, Visual Studio también nos permite publicar las aplicaciones conectando dichos dispositivos, pero primero se ha de configurar la licencia de Syncfusion que se encuentra en el archivo **App.xaml.cs** en el proyecto **SensoremTerram.App**. Para el caso de Android, conectando el dispositivo a un ordenador mediante un cable USB e implementado la aplicación esta queda instalada en el dispositivo. En cambio, para la instalación en un dispositivo con iOS, se necesita un ordenador de Apple con macOS y mediante Visual Studio para Mac se permite la publicación pero se tiene que disponer de una cuenta de desarrollo de Apple que se puede conseguir en el enlace (Apple Developer, 2020), en el siguiente enlace hay una explicación de cómo realizar la implementación en un dispositivo iOS (Microsoft Developer, 2018).

10. Conclusiones

10.1. Objetivos alcanzados

Al comenzar el desarrollo del TFG se tenía como objetivo el desarrollo de una solución de hardware y software para la monitorización de variables ambientales para el desarrollo en el ciclo de cultivo y se pretendían alcanzar varios objetivos generales que comentamos a continuación:

- OG-01: Se ha desarrollado un prototipo de hardware llamado Gadget que utiliza componentes hardware Open Source que permite la captura de variables o factores ambientales para la toma de decisiones agronómicas, además de conocer su localización. Se ha alcanzado los siguientes objetivos específicos:
    - OE-01: Se han estudiado los diferentes sistemas Open Source de comunicación y finalmente se decidió utilizar el protocolo de comunicación LoRaWAN.
    - OE-02: Se han estudiado diferentes dispositivos hardware y entre ellos se aposto por el uso de los productos facilitados por el fabricante MultiTech pues dispone del hardware mDot que ha permitido desarrollar el Gadget.
    - OE-03: Se han utilizado diferentes sensores de hardware que permiten medir magnitudes, en el mercado existe una gran variedad de hardware que permite la medición de diferentes magnitudes.
    - OE-04: Se ha utilizado las herramientas de desarrollo adecuadas para implementar el firmware mediante un compilador de C++ específico para la plataforma y adaptado al hardware utilizado.
    - OE-05: Las magnitudes monitorizadas se han enviado al Gateway, pues el MultiTech mDot que dispone diferentes sistemas de comunicación con otros hardware está capturando los datos recogidos y enviándolos al Gateway gracias al protocolo de comunicación LoRaWAN.

    - OE-06: Se ha utilizado un hardware especifico de localización por GPS que ha permitido la localización del Gadget.
    - OE-07: Se han estudiado diferentes tipos de baterías, además se ha intentado minimizar el consumo de energía para que el dispositivo tenga más durabilidad en el tiempo.
    - OE-08: Se ha probado la estabilidad del dispositivo, comprobando que este tiene un comportamiento estable, no dejando de responder por problemas de inestabilidad del software.
- OG-02: Se ha implementado un Gateway que permite recoger los datos de los Gadgets y enviar dichos datos a un servidor Back-End. Se ha alcanzado este objetivo general, gracias a conseguir los siguientes objetivos específicos:
    - OE-09: Se han estudiado diferentes tipos de hardware y se aposto por los productos facilitados por MultiTech, que ha permitido la comunicación entre los Gadget y el Gateway, además de la comunicación entre el Gateway y el servidor Back-End. Se ha utilizado el MultiTech Conduit con un hardware Gateway que permite la comunicación en el protocolo LoRaWAN además de estar conectado a internet mediante diferentes formas de conexión, desde una conexión Wifi, conexión Ethernet o una conexión 3G.
    - OE-10: La plataforma MultiTech Conduit permite entre otras utilizar la plataforma de desarrollo Node-RED que ha permitido fácilmente desarrollar una aplicación de alto nivel con JavaScript que ha permitido la comunicación con el servidor Back-End.
    - OE-11: También se ha optimizado él envió de datos desde el Gadget al Gateway, ya que hay una limitación en la longitud de datos que se puede enviar establecida en el protocolo LoRaWAN, se ha intentado comprimir él envió de datos.
- OG-03: Se ha desarrollado un servidor Back-End que permite la recogida de datos de los Gadget, que son capturados por los Gateways y enviados al servidor. Para alcanzar este objetivo general se ha desarrollado los siguientes objetivos específicos:

  - OE-12: Se ha desarrollado un servidor Back-End utilizando la plataforma ASP.NET Core con C#, donde se ha implementado un API REST que ha permitido enviar datos de los Gateways.
  - OE-13: Los datos recibidos de los Gadget se han almacenados en una base de datos SQL, además de procesar los datos para poder consultarlos.
  - OE-14: Se ha desarrollado en el servidor Back-End la API REST que permite a las aplicaciones Front-End accede a los datos procesados para poder visualizarlos.
  - OE-15: El servidor Back-End ha sido publicado en Cloud Computing gracias a la plataforma Microsoft Azure, en un servidor Linux.
- OG-04: Se ha desarrollado una aplicación Front-End que permite visualizar los datos de los Gadgets, mediante una aplicación web o aplicación móvil. Para alcanzar este objetivo general se han tenido que alcanzar los siguientes objetivos específicos:
  - OE-16: Se decidió utilizar la plataforma Open Source ofrecida por Microsoft .NET, que permite poder desarrollar una aplicación móvil gracias a Xamarin, como una aplicación Web con WebAssembly con Blazor.
  - OE-17: Se puede consultar los datos en diferentes plataformas, ya sea en iOS, Android y Windows o incluso en otras plataformas que disponga un navegador web compatible con WebAssembly. Las aplicaciones Front-End, realizan la comunicación con el servidor Back-End mediante el API REST, estas permiten mostrar los datos de los cultivos, así como poder visualizar los datos actuales como poder consultar datos históricos y poder exportar dichos datos a un archivo Excel para el tratamiento de los datos por parte del usuario.
  - OE-18: Las aplicaciones clientes, permite visualizar mediante un mapa la localización de los Gadgets como de los Gateways.
  - OE-19: La aplicación cliente web ha sido publicada en Cloud Computing, pues está integrada con el servidor Back-End.

Por último, se han alcanzado los objetivos que se pretendía alcanza en este TFG, pero se han producido desviaciones de tiempo en el desarrollo del proyecto respecto a lo estimado. Este ha sido un proyecto ambicioso y en la tabla 35 se puede ver el Product Backlog con las estimaciones, las nuevas tareas creadas para completar los objetivos, así como se puede ver que se ha completado todas las tareas que se han definido, tanto en la planificación como las revisiones realizada en cada uno de los Sprint. También se puede ver la diferencia que se ha producido entre lo estimado y lo realizado que ha sido de unos 57,5 puntos de historia que a la velocidad de desarrollo de 0,444 puntos de historia por hora supone que se han necesitado sobre unas 674 horas aproximadamente para llevar a cabo el proyecto. Esto ha supuesto un incremento de coste en mano de obra de 1315,96 EUR, lo que supone un sobrecoste del 24% más. Esto supone que en las próximas estimaciones habría que tener en cuenta este sobrecoste.

La metodología ágil ha permitido y adaptado las necesidades del proyecto según las necesidades que han ido surgiendo en el desarrollo de cada Sprint.

Además, solo se ha necesitado el alojamiento de un servidor, como está integrado el servidor Back-End y el servidor cliente web, disminuyendo el coste en unos 132,96 EUR anuales.

Product Backlog												
Épicas		Cuestiones		Tareas		Prioridad	Story Points inicial	Realizado	Sprint inicial	Story Points nuevas	Srpint Real	Story Points Reales
Id	Titulo	Id	Titulo	Id	Titulo							
148	Gadget	151	Monitorización de factores	160	Estudiar herramientas de desarrollo del Firmware	M	8,0	S	1		1	8,0
				161	Comunicación puerto serie para interacción	M	2,0	S	1		1	2,0
				162	Lectura analógica	M	8,0	S	1		1	9,0
				163	Lectura/Escritura digital	M	2,0	S	1		1	1,0
				211	Estudiar comunicación bus I2C	M	5,0	S	1		1	5,0
				164	Comunicación sensores con bus I2C	M	2,0	S	1		1	3,0
				165	Búsqueda de dispositivos I2C	S	3,0	S	1		1	3,0
				168	Gestor de sensores	S	5,0	S	2		2	7,0
				218	Estudiar tipo de sensores del mercado	M	5,0	S	4		4	4,0
				217	Implementar sensores	M	13,0	S	4		4	13,0
		153	Enviar datos al Gateway	167	Codificación de datos	M	5,0	S	2		2	4,0
				166	Envió de datos al Gateway	M	5,0	S	2		2	5,0
		208	Energía	209	Estudio de suministro de energía	M	8,0	S	4		4	7,0
				210	Implementación de la energía	M	3,0	S	4		4	4,0
				216	Medición nivel de la energía	S	2,0	S	4		4	2,0
				220	Ahora de energía	C	3,0	S	7		7	4,0
				292	Mejorar el hardware para despertar GPS	W		S		2,0	9	2,0
				293	Mejorar el software STEP y despertar GPS	W		S		5,0	9	2,0
				296	Detectar funcionamiento de los sensores	M		S		2,0	9	4,0

		152	Localización	169	Estudio de sistemas GPS hardware	C	13,0	S	7		7	10,0
				170	Enviar datos de localización al Gateway	C	5,0	S	7		7	5,0
147	Gateway	154	Recepción datos Gadget	171	Estudiar entorno de desarrollo Node-RED	M	8,0	S	2		2	8,0
				172	Recepción de paquetes del Gadget	M	3,0	S	2		2	3,0
		155	Envió de datos al Back-End	174	Transformar paquetes Gadget a JSON	M	5,0	S	2		2	5,0
				175	Enviar datos al servidor Back-End	M	8,0	S	3		3	8,0
		179	Estado del Gateway	180	Obtener datos del estado del Gateway	S	5,0	S	3		3	5,0
				182	Enviar estado Gateway en formato JSON al Back-End	S	2,0	S	3		3	0,0
		173	Localización	176	Lectura de posicionamiento del Gateway	C	3,0	S	3		3	0,0
				177	Envió de datos de posicionamiento al Back-End	C	2,0	S	3		3	1,0
148	Back-End	156	Recepción datos Gateway y persistencia de datos	202	Crear base de datos en Azure	M	0,5	S	3		3	0,5
				182	Modelo de datos	M	5,0	S	3		3	10,0
				203	Implementar servidor de aplicación en Azure	M	0,5	S	3		3	0,5
				183	Controlador recepción datos Gadget	M	2,0	S	3		3	2,0
				184	Controlador recepción estado Gateway	C	2,0	S	3		3	2,0
				205	Controlador recepción localización Gateway	S	2,0	S	3		3	2,0
		159	Acceso aplicaciones Front-End	185	Controlador acceso datos Gadgets	M	3,0	S	5		5	4,0
				186	Controlador acceso datos Gateways	S	3,0	S	5		5	1,0
				219	Controlador mantenimiento de Gadgets	C	5,0	S	8		9	1,0
		221	Procesado de datos	223	Procesar datos de los Gadgets	M		S		5,0	4	10,0
				288	Revisión de procesado de datos de los Gadgets	M		S		5,0	8	8,0
				289	Registro de eventos del sistema	M		S		2,0	8	2,0
149	Front-End	157	Aplicación Web	207	Crear base aplicación IU	M	8,0	S	5		5	10,0
				213	Cuadro de mandos	M	5,0	S	5		5	7,0
				204	Implementar servidor de aplicación en Azure	M	0,5	S	5		5	1,0

				189	Consultar datos Gadgets	M	5,0	S	5		5	2,0
				191	Consultar históricos Gadgets	M	3,0	S	5		5	6,0
				198	Consultar datos Gateway	S	5,0	S	7		7	2,0
				190	Consultar localización Gadgets	C	5,0	S	8		8	6,0
				199	Consultar localización Gateways	C	5,0	S	7		7	12,0
				192	Exportar datos a Excel	M	3,0	S	5		5	3,0
				215	Mantenimiento de Gadgets	C	5,0	S	8		9	4,0
				231	Mejora de interfaz de usuario	M		S		8,0	9	8,0
				290	Consulta de eventos del sistema	M		S		3,0	8	3,0
				294	Adaptar proyecto a la nueva versión Blazor	M		S		1,0	8	1,0
				295	Actualizar proyecto Blazor RC1	M		S		1,0	9	1,0
		158	Aplicación móvil	206	Crear base aplicación IU	M	8,0	S	6		6	10,0
				212	Cuadro de mandos	M	8,0	S	6		6	8,0
				194	Consultar datos Gadgets	M	5,0	S	6		6	10,0
				196	Consultar históricos de los Gadgets	M	5,0	S	6		6	10,0
				200	Consultar datos Gateways	S	5,0	S	6		6	2,0
				195	Consultar localización de los Gadgets	C	5,0	S	8		8	5,0
				201	Consultar localización de los Gateways	C	5,0	S	8		8	5,0
				287	Mejora de iconos, pantalla inicial	C		S		5,0	8	2,0
				291	Consulta de eventos del sistema	M		S		3,0	8	3,0
Suma de puntos de historias…						**241,5**				**42,0**		**299,0**

Tabla 35 - Product Backlog resultado.

10.2. Conclusiones del trabajo y personales

Cuanto terminé mis estudios de ingeniería técnica de informática de sistema en el año 1997 y tras acumular experiencia en: nuevos lenguajes, herramientas de desarrollo y funcionamiento de empresas, llegó un momento en que decidí seguir con mis estudios para mejorar profesionalmente siendo todo un reto para mi volver a estudiar después de tanto tiempo.

Hace años cuando se comenzó a hablar del IoT, pensé en llevar a cabo este proyecto, no pudiendo hacerlo por falta de tiempo. Por lo que pensé llevarlo a cabo con este TFG.

Con la excusa del desarrollo del TFG me propuse comenzar este proyecto, que en el inicio me costó bastante, he tenido que adaptar mi mente para escribir en los requerimientos. Además de tener que formarme en el aprendizaje de las metodologías agiles, pues cuando termine la carrera todas estas estaban apareciendo y aunque había odio hablar de ellas me sonaba como algo muy complicado. Pero al final, estas metodologías son muy parecidas a las metodologías que utilizo a la hora de desarrollar un nuevo proyecto, descomponiendo un gran problema en otros problemas más pequeños y sencillos, además de ir mostrando a los usuarios finales los resultados y adaptarlos a las nuevas necesidades.

El desarrollo de este TFG me ha servido para conocer más estas metodologías agiles más en profundidad y mejorar mi metodología de desarrollo a nivel profesional, controlando más la evolución del proyecto mediante los Sprint.

Además, este proyecto también me ha servido para conocer más sobre el mundo IoT, como diferentes sistemas de comunicaciones, pues a la hora de comenzar el proyecto tuve que estudiar diferentes sistemas que hay en el mercado y tomar la decisión por uno de ellos para realizar este TFG.

He tenido que recuperar conocimientos de electrónica para implementar en el desarrollo del Gadget, que para mí ha sido muy motivador el desarrollo de este y ver cómo funciona. Así como buscar hardware Open Source y ver cómo funciona. Toda una nueva experiencia para mí, pues nunca había llevado a cabo un proyecto de desarrollo de hardware y he adquirido bastantes conocimientos para seguir mejorando el producto y seguir investigando.

Como conocer el porfolio de la marca MultiTech, así como el aprendizaje de la infraestructura LoRaWAN y aprender la configuración de dichos dispositivos, plataformas que ofrece.

El aprendizaje de la plataforma Node-RED que permite de una manera gráfica y fácil el desarrollo de una aplicación que permite interactuar con un hardware a muy alto nivel, sin tener que profundizar.

Por otra parte, el uso de nuevas tecnologías como Blazor me ha aportado bastantes conocimientos sobre esta nueva tecnología que parece que va a portar un buen futuro y que parte de los ingenieros de Microsoft está desarrollando con la ayuda de otros desarrolladores, desarrollando un proyecto Open Source, pretendiendo llevar esta tecnología al desarrollo de aplicaciones de escritorio, como móviles.

También me ha servido el desarrollo de este TFG para mejorar mis conocimientos sobre Xamarin.

Gracias al aprendizaje adquirido en la adaptación al grado de informática, en la asignatura de desarrollo a aplicaciones móviles me ha servido para comprender más aun las plataformas de Android y iOS, poniendo en práctica los conocimientos aportados y mejorando el desarrollo de aplicaciones móviles en dichas plataformas.

Para el desarrollo del servidor Back-End he aplicado mis conocimientos que he adquirido durante estos años en el desarrollo de aplicaciones, intentando crear la independencia de datos con servidores específicos de base de datos, el

uso de inyección de dependencias, pudiendo desarrollar dicho proyecto con cualquier tipo de base de datos.

En definitiva, he salido de mi zona de confort para poder desarrollar un proyecto que tenía en mente y que tenía mucha ilusión para llevar a cabo.

10.3. Vías futuras

Sensorem Terram que es el nombre que le he dado al proyecto, puede seguir evolucionando para mejorar, por lo que a continuación propongo una serie de mejoras del proyecto:

- Implementar opción de mantenimiento de usuarios, para registrar usuario y roles.
- Implementar sistema de autentificación que solo permita acceso a usuarios registrados.
- Implementar sistema de autorización, para que los usuarios solo puedan acceder a su información.
- Implementar en el servidor Back-End seguridad mediante el uso de JWT, tanto para la conexión de los Gateways como de las aplicaciones móviles.
- Implementar sistema de mantenimiento de Gadgets y Gateways, para llevar el control de los Gadget, para registrar incidencias, mantenimientos y reparaciones.
- Mantenimiento de Hardware, para poder crear, modificar y eliminar hardware y sensores.
- Diseño y fabricación de la PCB.
- Diseño y fabricación de la caja para proteger la electrónica.
- Implementar sistema de facturación a los usuarios de los Gadgets alquilados.

Una vez puesto en marcha el proyecto en real se tiene que analizar los datos que se obtienen y ver las mejoras que se tienen que seguir realizando, para ofrecer a los clientes más información que les pueda servir para mejorar los

cultivos, así como el estudio de nuevos sensores que pueda servir para capturar otra información importante para analizar.

El uso de Inteligencia Artificial en los datos capturados puede ofrecer información a los usuarios sobre patrones que permitan mejorar los cultivos.

Finalmente, este es un proyecto muy interesante que ya es una realidad para comenzar su implantación en real en poco tiempo.

11. Bibliografía

330ohms. (2 de Marzo de 2016). *330ohms*. Obtenido de ¿Que es una Protobard?: https://blog.330ohms.com/2016/03/02/protoboards/

Adafruit. (Noviembre de 2019). *MCP9808 High Accuracy I2C Temperature Sensor Breakout Board*. Obtenido de Adafruit.com.

Adobe. (Mayo de 2020). *Software de diseño de gráficos vectoriales.* Obtenido de Adobe: https://www.adobe.com/es/products/illustrator.html?gclid=Cj0KCQjwncT1BRDhARIsAOQF9LkpQvlCNXg5gpzfWgPCZvhBSGh-hCi1ZrGtv5QdipN5luO3_S9X1xwaAkftEALw_wcB&sdid=8DN85NTQ&mv=search&ef_id=Cj0KCQjwncT1BRDhARIsAOQF9LkpQvlCNXg5gpzfWgPCZvhBSGh-hCi1ZrGtv5QdipN5luO3_S9

agilemanifesto.org. (2001). *Manifiesto por el Desarrollo Ágil de Software*. Obtenido de Manifiesto por el Desarrollo Ágil de Software: https://agilemanifesto.org/iso/es/manifesto.html

agilemanifesto.org. (Octubre de 2019). *Principios del Manifiesto Ágil*. Obtenido de agilemanifesto.org: https://agilemanifesto.org/iso/es/principles.html

Aguile Bussines Consortium. (Octubre de 2019). *What is DSDM?* Obtenido de aguilebussines.com: https://www.agilebusiness.org/page/whatisdsdm

Aguilera, D. (28 de Marzo de 2017). *El programador pragmático – Consejos para ser mejor desarrollador WordPress*. Obtenido de Nelio Software: https://neliosoftware.com/es/blog/programador-pragmatico-consejos-desarrollador-wordpress/

Apple Developer. (16 de Mayo de 2020). *Apple Developer*. Obtenido de Apple Developer: https://developer.apple.com

Arai, K. (7 de Mayo de 2015). *Digital 16bit Serial Output Type Ambient Light Sensor IC by ROHM, Ambient light sensor (Illuminance to digital converter)*. Obtenido de os.mbed.com: https://os.mbed.com/users/kenjiArai/code/BH1750/docs/tip/classBH1750.html

Aritchie. (Febrero de 2020). *Arc.UserDialogs*. Obtenido de NuGet: https://www.nuget.org/packages/Acr.UserDialogs/

Arm Mbed. (Octubre de 2019). *Developing: Mbed CLI - Tools*. Obtenido de Mbed OS 5 Documentaion: https://os.mbed.com/docs/mbed-os/v5.14/tools/developing-mbed-cli.html

Arm MBED. (Noviembre de 2019). *Dot-Examples*. Obtenido de Arm MBED: https://os.mbed.com/teams/MultiTech/code/Dot-Examples/

Arm Mbed. (Octubre de 2019). *Home*. Obtenido de Mbed: https://www.mbed.com/en/

Arm Mbed. (Octubre de 2019). *mbed-os*. Obtenido de Github: https://github.com/ARMmbed/mbed-os/tree/mbed-os-5.14.1

Asp Gems. (5 de Abril de 2019). *El modelo de desarrollo en espiral como mezcla de cascada e iterativo*. Obtenido de aspgems.com: https://aspgems.com/metodologia-de-desarrollo-de-software-iii-modelo-en-espiral/

Atlassian. (Octubre de 2019). *What is Git*. Obtenido de Atlassian: https://www.atlassian.com/git/tutorials/what-is-git

Autofac Contributors. (Febrero de 2020). *Home*. Obtenido de Autofact: https://autofac.org/

Carmen Lasa Gómez, A. Á. (2018). *Métodos Ágiles. Scrum, Kanban, Lean (Manual Imprescindible).* Anaya Multimedia.

cpluscplus. (Octubre de 2019). *cplusplus.com*. Obtenido de The C++ Resource Network: http://www.cplusplus.com/

DFRobot. (Noviembre de 2019). *SHT20_I2C_Temperature & Humidity*. Obtenido de DFRobot: https://wiki.dfrobot.com/SHT20_I2C_Temperature_&_Humidity_Sensor__Waterproof_Probe__SKU__SEN0227

Domínguez, P. (30 de Octubre de 2017). *En qué consiste el modelo en cascada - Gestiona tu proyecto de desarrollo*. Obtenido de OpenClassrooms: https://openclassrooms.com/en/courses/4309151-gestiona-tu-proyecto-de-desarrollo/4538221-en-que-consiste-el-modelo-en-cascada

EasyEDA. (Octubre de 2019). *Simulador de circuitos y diseño de circuitos impresos online*. Obtenido de EasyEDA: https://easyeda.com/es

ecured.cu. (Octubre de 2019). *Gateway*. Obtenido de Gateway: https://www.ecured.cu/Gateway

Energy EV. (29 de Agosto de 2014). *LAS BATERÍAS DE LITIO Y BMS, ¿COMO FUNCIONAN?* Obtenido de Energyecv.com: https://energyev.com/las-baterias-de-litio-y-bms-como-funcionan/

Estrada-Marmolejo, D. R. (Otubre de 2017). *Puerto Serial – protocolo y su teoría*. Obtenido de hetpro: https://hetpro-store.com/TUTORIALES/puerto-serial/

Extreme Programing. (Octubre de 2019). *A gentle introduction*. Obtenido de Extreme Programing: http://www.extremeprogramming.org

Fenix. (Noviembre de 2019). *What is the Difference Between "Protected" and "Unprotected" 18650 Batteries*. Obtenido de Fenix-store.com: https://www.fenix-store.com/blog/what-is-the-difference-between-protected-and-unprotected-18650-batteries/

Garzas, J. (4 de Septiembre de 2012). *Un resumen de la metodología ágil FDD*. Obtenido de javiergarzas.com: https://www.javiergarzas.com/2012/09/metodologia-gil-fdd-1.html

GlobalTop Tech Inc. (2015). *PMTK Packet User Manual .* Obtenido de sparkfun.com: https://cdn.sparkfun.com/assets/parts/1/2/2/8/0/PMTK_Packet_User_Manual.pdf

Gomez, K. (27 de Julio de 2017). *Top 5 Metodologías de Desarrollo de Software*. Obtenido de Top 5 Metodologías de Desarrollo de Software: https://www.megapractical.com/blog-de-arquitectura-soa-y-desarrollo-de-software/metodologias-de-desarrollo-de-software

Google Cloud. (Diciembre de 2019). *Mapas personalizados, Google Maps Platform*. Obtenido de Google Colud: https://cloud.google.com/maps-platform/maps?hl=es

Google Developers. (20 de Febrero de 2020). *Google Maps JavaScript API V3 Reference*. Obtenido de Google Developers: https://developers.google.com/maps/documentation/javascript/reference?hl=es

Grzywacz, M. (22 de Agosto de 2017). *Library for BH1750 I2C light sensor. Supports autoranging! True to datasheet. (beware: calls are blocking at the moment)*. Obtenido de os.mbed.com: https://os.mbed.com/users/amateusz/code/BH1750/

HANNA Instruments. (Octubre de 2019). *Instrumentos de medida y análisis*. Obtenido de HANNA Instrumets: https://www.hannainst.es/

Hart, M. (28 de Septiembre de 2014). *TinyGPS++*. Obtenido de Arduiniana: http://arduiniana.org/libraries/tinygpsplus/

HereLab. (Octubre de 2019). *HereLab / BME280*. Obtenido de arm MBED: https://os.mbed.com/teams/HereLab/code/BME280/

Jorge Rubira. (21 de Abril de 2011). *¿Qué es la inyección de dependencias?* Obtenido de Genbeta: https://www.genbeta.com/desarrollo/que-es-la-inyeccion-de-dependencias

json.org. (Octubre de 2019). *Introducción a JSON*. Obtenido de Introducción a JSON: https://www.json.org/json-es.html

Kamalittipong, R. (Abril de 2020). *BlazorGoogleMaps: Blazor interop for GoogleMap library*. Obtenido de GitHub: https://github.com/rungwiroon/BlazorGoogleMaps

Kanban tool. (Octubre de 2019). *Scrumban - Lo mejor de Kaban y Scrum*. Obtenido de Kanban tool: https://kanbantool.com/es/scrumban-scrum-y-kanban

kanbanize.com. (Octubre de 2019). *Qué es Kanban: Fundamentos*. Obtenido de Qué es Kanban: Fundamentos: https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-kanban/

Kissflow. (31 de Octubre de 2018). *Rapid Application Development: Definition, Steps, Advantages and Case Study*. Obtenido de Kissflow: https://kissflow.com/rad/rapid-application-development/

LESS Industries. (Octubre de 2019). *Monitoreo inteligente en agricultura e industria*. Obtenido de Monitoreo inteligente en agricultura e industria: http://www.lessindustries.com/index-es.html

LoRa Alliance. (Octubre de 2019). *Home page*. Obtenido de LoRa Alliance: https://lora-alliance.org/

López Gil, A. (Septiembre de 2018). *Estudio comparativo de metodologías tradicionales y ágiles para proyectos de Desarrollo de Software.* Obtenido

de Estudio comparativo de metodologías tradicionales y ágiles para proyectos de Desarrollo de Software: http://uvadoc.uva.es/bitstream/handle/10324/32875/TFG-I-1015.pdf;jsessionid=B4AE4A26DA84BEAF35BCF1CB20DDC2DB?sequence=1

M, J. (9 de Enero de 2017). *MCP9808 Digital temperature sensor*. Obtenido de os.mbed.com: https://os.mbed.com/users/Jmfox24/code/MCP9808/

Mactronica. (Noviembre de 2019). *SENSOR DE HUMEDAD DEL SUELO HD-38*. Obtenido de Mactronica: https://www.mactronica.com.co/sensor-de-humedad-del-suelo-hd38-997242974xJM

maldeadora. (2017). *Qué es Frontend y Backend*. Obtenido de Platzi: https://platzi.com/blog/que-es-frontend-y-backend/

Mbed OS. (Octubre de 2019). *Introduction - Mbed OS 5*. Obtenido de Mbed OS 5 Documentation: https://os.mbed.com/docs/mbed-os/v5.14/introduction/index.html

MDN. (Octubre de 2019). *What is JavaScript?* . Obtenido de MDN web docs: https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/Qu%C3%A9_es_JavaScript

MDN web docs. (6 de Mayo de 2019). *Control de acceso HTTP (CORS) - HTTP | MDN*. Obtenido de developer.mozilla.org: https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS

Metodoss. (Octubre de 2019). *Metodología RUP*. Obtenido de Metodoss.com: https://metodoss.com/metodologia-rup/

Microsoft. (20 de Julio de 2015). *Métodos de extensión: Guia de programación de C#*. Obtenido de Microsoft Docs: https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/extension-methods

Microsoft. (27 de Noviembre de 2016). *Entity Framework Core*. Obtenido de Microsoft Docs: https://docs.microsoft.com/es-es/ef/core/

Microsoft. (7 de 8 de 2017). *Patrón Model-View-ViewModel*. Obtenido de Microsoft Docs: https://docs.microsoft.com/es-es/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

Microsoft. (7 de Mayo de 2018). *Microsoft Docs*. Obtenido de ¿Qué es una aplicación para la Plataforma universal de Windows (UWP)?: https://docs.microsoft.com/es-es/windows/uwp/get-started/universal-application-platform-guide

Microsoft. (Octubre de 2019). *Blazor Build client web apps with C#*. Obtenido de .NET: https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor

Microsoft. (Noviembre de 2019). *Crear una aplicación web*. Obtenido de Microsoft Azure: https://azure.microsoft.com/es-es/get-started/web-app/

Microsoft. (Octubre de 2019). *Descarga de SQL Server Management Studio (SSMS)*. Obtenido de Microsoft Docs: https://docs.microsoft.com/es-es/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-2017

Microsoft. (Octubre de 2019). *Documentación de .NET*. Obtenido de Microsoft Docs: https://docs.microsoft.com/es-es/dotnet/

Microsoft. (30 de 1 de 2019). *Guía de C#*. Obtenido de Microsoft Docs: https://docs.microsoft.com/es-es/dotnet/csharp/

Microsoft. (1 de 8 de 2019). *Información general de ASP.NET Core MVC*. Obtenido de Microsoft Docs: https://docs.microsoft.com/es-es/aspnet/core/mvc/overview?view=aspnetcore-3.0

Microsoft. (5 de Noviembre de 2019). *Inserción de dependencias en ASP.NET Core*. Obtenido de Microsoft Docs: https://docs.microsoft.com/es-es/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-3.0

Microsoft. (27 de 11 de 2019). *Introducción a ASP.NET Core SignalR*. Obtenido de Microsoft Docs: https://docs.microsoft.com/es-es/aspnet/core/signalr/introduction?view=aspnetcore-3.1

Microsoft. (Octubre de 2019). *Microsoft DevOps Services*. Obtenido de Microsoft Azure: https://azure.microsoft.com/es-es/services/devops/

Microsoft. (14 de 5 de 2019). *¿Qué es Azure Data Studio?* Obtenido de Microsoft Docs: https://docs.microsoft.com/es-es/sql/azure-data-studio/what-is?view=sql-server-2017

Microsoft. (19 de Noviembre de 2019). *Tareas en segundo plano con servicios hospedados en ASP.NET Core*. Obtenido de Microsoft Docs: https://docs.microsoft.com/es-es/aspnet/core/fundamentals/host/hosted-services?view=aspnetcore-3.1&tabs=visual-studio

Microsoft. (Octubre de 2019). *Visual Studio 2019*. Obtenido de Visual Studio: https://visualstudio.microsoft.com/es/vs/

Microsoft. (Octubre de 2019). *Visual Studio Code - Code Editing. Redefined*. Obtenido de Visual Studio: https://code.visualstudio.com/

Microsoft. (Octubre de 2019). *Xamarin | Open-source mobile app platform for .NET*. Obtenido de .NET: https://dotnet.microsoft.com/apps/xamarin

Microsoft. (Mayo de 2020). *Servicios de informática en la nube*. Obtenido de Microsoft Azure: https://azure.microsoft.com/

Microsoft Azure. (9 de Septiembre de 2019). *Crear una base de datos única*. Obtenido de Microsoft Docs: https://docs.microsoft.com/es-es/azure/sql-database/sql-database-single-database-get-started?tabs=azure-portal

Microsoft Azure. (Octubre de 2019). *SQL Database - Base de datos en la nube como servicio | Microsoft Azure*. Obtenido de Microsoft Azure: https://azure.microsoft.com/es-es/services/sql-database/

Microsoft Developer. (13 de Diciembre de 2018). *Debug to iOS Devices Over Wi-Fi | The Xamarin Show*. Obtenido de YouTube: https://www.youtube.com/watch?v=GoGgIBDXyTc

Microsoft Docs. (11 de 06 de 2019). *Customizing a Map Pin*. Obtenido de Microsoft Docs: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/custom-renderer/map-pin

Microsoft Docs. (19 de Febrero de 2020). *Microsoft Docs*. Obtenido de Llamada a funciones de JavaScript con métodos de .NET en Blazor de ASP.NET Core: https://docs.microsoft.com/es-es/aspnet/core/blazor/call-javascript-from-dotnet?view=aspnetcore-3.1

Moncayo, J. M. (17 de Mayo de 2018). *¿Qué es REST? Conoce su potencia*. Obtenido de ¿Qué es REST? Conoce su potencia: https://openwebinars.net/blog/que-es-rest-conoce-su-potencia/

Mouser. (Noviembre de 2019). *Ambient Light Sensor ICs - ROHM*. Obtenido de Mouser España: https://www.mouser.es/new/rohm-semiconductor/rohm-ambient-light-sensor-ics/

Mouser. (Noviembre de 2019). *BME280 Humidity and Pressure Sensor - Bosch*. Obtenido de Mouser España: https://www.mouser.es/new/bosch/bosch-bme280/

Mouser. (Noviembre de 2019). *Digital Humidity and Temperature Sensors - Sensirion*. Obtenido de Mouser España: https://www.mouser.es/new/sensirion/sensirion-digital-humidity-temp-sensors/

Mouser. (Noviembre de 2019). *LM1117 800mA Low-Dropout Linear Regulators - TI*. Obtenido de Mouser España: https://www.mouser.es/new/texas-instruments/nationalLM1117/

Mouser. (Noviembre de 2019). *Sensor de luz UV VEML6070 - Vishay*. Obtenido de Mouser España: https://www.mouser.es/new/vishay/vishay-veml6070-sensor/

MtM+. (2 de Junio de 2017). *SHT21 & SHT25 humidity and temperature sensors*. Obtenido de os.mbed.com: https://os.mbed.com/teams/MtM/code/SHT2X/file/f9e3348b41a2/SHT2X.h/

MultiTech. (Octubre de 2019). *IoT Devices + Hardware | Wireless Solutions Company | MultiTech*. Obtenido de Multitech: https://www.multitech.com/

MultiTech. (Octubre de 2019). *mDot LoRa Module Developer Kit*. Obtenido de MultiTech: https://www.multitech.com/brands/mdot-devkit

MultiTech. (Octubre de 2019). *MultiTech*. Obtenido de MultiTech: https://www.multitech.com/

MultiTech. (Octubre de 2019). *MultiTech*. Obtenido de mDot LoRa Module Developer Kit: https://www.multitech.com/brands/mdot-devkit

MultiTech. (Abril de 2020). *IoT Cloud Management Platform | M2M Remote Device Management*. Obtenido de MultiTech: https://www.multitech.com/brands/devicehq

MultiTech Developer Resources. (Octubre de 2019). *Introduction to LoRa*. Obtenido de Introduction to LoRa: http://www.multitech.net/developer/software/lora/introduction-to-lora/

MultiTech Developer Resources. (Noviembre de 2019). *MTR and Conduit API Reference*. Obtenido de MultiTech Developer Resources:

http://www.multitech.net/developer/software/mtr-software/mtr-api-reference/

MultiTech Developer Resources. (Mayo de 2020). *MultiTech Developer Resources*. Obtenido de LoRa Configuration: http://www.multitech.net/developer/software/lora/getting-started-with-lora-conduit-aep/

Multitech/libmDot-mbed5. (Octubre de 2019). *libmDot-mbed5*. Obtenido de MultiTech: https://os.mbed.com/teams/MultiTech/code/libmDot-mbed5/

National Marine Electronis Association. (2019). *Asociación Nacional de Electrónica Marina de NMEA*. Obtenido de nmea.org: https://www.nmea.org/

Node-RED. (Octubre de 2019). *Node-RED*. Obtenido de Node-RED: https://nodered.org/

Open Source Hardware Association. (Octubre de 2019). *Spanish*. Obtenido de Open Source Hardware Association: https://www.oshwa.org/definition/spanish/

PCE Instruments. (Octubre de 2019). *Fabricante y distribuidor de equipos de medición.* Obtenido de Fabricante y distribuidor de equipos de medición.: https://www.pce-iberica.es

Pololu. (Noviembre de 2019). *Pololu 3.3V Step-Up/Step-Down Voltage Regulator S7V8F3*. Obtenido de Pololu.com: https://www.pololu.com/product/2122

Postman. (Octubre de 2019). *The Collaboration Platform for API Development*. Obtenido de Postman: https://www.getpostman.com/

Putty.org. (Octubre de 2019). *Download PuTTY - a free SSH and telnet client for Windows*. Obtenido de Putty.org: https://www.putty.org/

redhat. (Octubre de 2019). *El concepto de cloud computing*. Obtenido de El concepto de cloud computing: https://www.redhat.com/es/topics/cloud

Roberts, S. (19 de Junio de 2017). *Driver for Vishay VEML6070 UV-A Sensor*. Obtenido de os.mbed.com: https://os.mbed.com/users/smatthew/code/VEML6070//file/14568f6891b3/VEML6070.h/

scrum.org. (Octubre de 2019). *¿Què es scrum?* Obtenido de ¿Què es scrum?: https://www.scrum.org/resources/what-is-scrum?gclid=CjwKCAjwq4fsBRBnEiwANTahcAhYDVXe3wLSfxkLtM5P2fNHsUEDA-g5m3Rz8ykfVEQa-MClTi1rxBoC7O8QAvD_BwE

Sensoterra. (Octubre de 2019). *Homepage*. Obtenido de Sensoterra: https://www.sensoterra.com/

SparkFun. (19 de Diciembre de 2019). *SparkFun_I2C_GPS_Arduino_Library*. Obtenido de GitHub: https://github.com/sparkfun/SparkFun_I2C_GPS_Arduino_Library

SparkFun. (Febrero de 2020). *SparkFun GPS Breakout - XA1110 (Qwiic)*. Obtenido de sparkfun.com: https://www.sparkfun.com/products/14414

Stehlik, M. (18 de Agosto de 2014). *Library for digital light sensor BH1750 (GY-30).* Obtenido de os.mbed.com: https://os.mbed.com/users/vrabec/code/BH1750/

Syncfusion Inc. (Diciembre de 2019). *.NET, Xamarin, JavaScript, componentes de IU angular*. Obtenido de Syncfusion: https://www.syncfusion.com/

Syncfusion Inc. (Diciembre de 2019). *Componentes Blazor | Más de 65 controles de IU y DataViz para Web*. Obtenido de Syncfusion: https://www.syncfusion.com/blazor-components

Syncfusion Inc. (Diciembre de 2019). *Overview of Syncfusion Excel (XlsIO) library*. Obtenido de Syncfusion Documentation: https://help.syncfusion.com/file-formats/xlsio/overview

Syncfusion Inc. (Diciembre de 2019). *Theme Studio | JavaScript | React | Vue | Angular | Syncfusion*. Obtenido de ej2.syncfusion.com: https://ej2.syncfusion.com/themestudio/?theme=material

techtarget.com. (Enero de 2017). *¿Qué es Internet de las cosas (IoT)? - Definición en WhatIs.com*. Obtenido de ¿Qué es Internet de las cosas (IoT)? - Definición en WhatIs.com: https://searchdatacenter.techtarget.com/es/definicion/Internet-de-las-cosas-IoT

techtarget.com. (s.f.). *¿Qué es Internet de las cosas (IoT)? - Definición en WhatIs.com*. Obtenido de ¿Qué es Internet de las cosas (IoT)? - Definición en WhatIs.com: https://searchdatacenter.techtarget.com/es/definicion/Internet-de-las-cosas-IoT

Telsaben.com. (4 de Febrero de 2017). *Protcolo I2C - Fundamentos del Protocolo I2C - Aprende*. Obtenido de Teslabem.com: https://teslabem.com/nivel-intermedio/fundamentos-del-protocolo-i2c-aprende/

The Things Network. (Octubre de 2019). *Homepage*. Obtenido de The Things Network: https://www.thethingsnetwork.org/

ThingsBoard. (Octubre de 2017). *Open-Source IoT Platform*. Obtenido de ThinsBoard: https://thingsboard.io/

ticbeat.com. (12 de Julio de 2017). *¿Qué es una API y para qué sirve?* Obtenido de ¿Qué es una API y para qué sirve?: https://www.ticbeat.com/tecnologias/que-es-una-api-para-que-sirve/

WebAssembly. (Diciembre de 2019). *WebAseembly*. Obtenido de WebAssembly: https://webassembly.org/

WIDHOC. (Octubre de 2019). *Homesite - Empresa de Base Tecnologica*. Obtenido de Homesite - Empresa de Base Tecnologica: http://widhoc.com/

12. Anexos

12.1. Manuales de instalación

12.1.1. Instalación del Arm Mbed Cli

Para la instalación de un equipo en Windows hay dos métodos de instalación disponibles. El primero es con el instalado preconstruido que se puede descargar e instalar, que es el utilizado. El segundo método de instalación es más flexible y permite una mayor personalización para adaptarse mejor a las necesidades de cada sistema.

Para el primer método que ha sido el empleado me he descargado el instalado y lo he ejecutado del siguiente enlace, https://github.com/ARMmbed/mbed-cli-windows-installer/releases/tag/v0.4.10, que para el desarrollo de este proyecto se ha utilizado la versión 0.4.10, descargando el archivo Mbed_installer_v0.4.10.exe.

Una vez descargado el archivo se ha procedido con la instalación utilizando el asistente como se ve en la ilustración 91, aceptando licencias para proceder a su instalación.

Creación de un proyecto y compilación

Ilustración 91 - Asistente de instalación Mbed CLI para Windows

Una vez instalado en una ventana del símbolo del sistema se puede ejecutar el comando `mbed --version` para verificar la correcta instalación, como se puede ver en la ilustración 92.

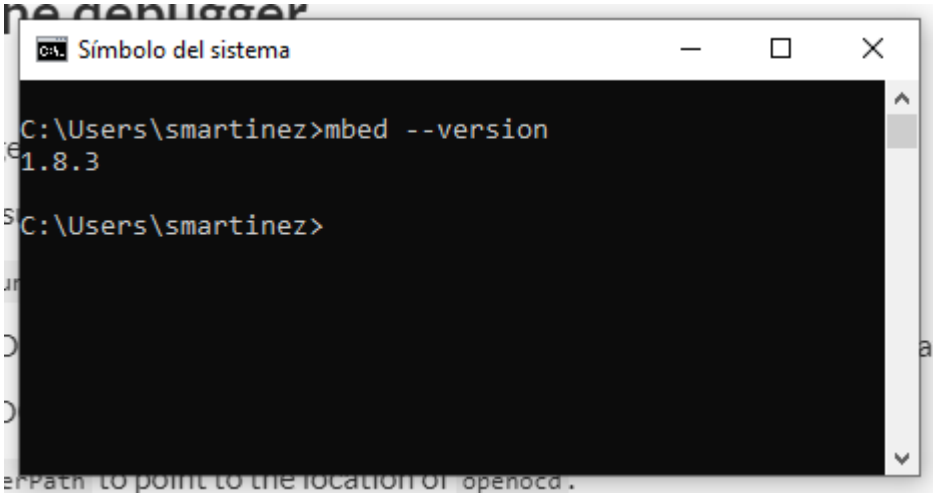

Ilustración 92 - Comprobar versión de instalación Mbed CLI

12.1.2. Creación de un proyecto y compilación con Mbed CLI

Para la creación de un proyecto con la herramienta Mbed CLI se ha de ejecutar el siguiente comando:

```
$ mbed new <nombre_del_proyecto>
```

Una vez creado el proyecto se puede añadir librerías necesarias con el siguiente comando:

```
$ mbed add <url_libreria>
```

Antes de compilar la solución hay que seleccionar la herramienta de desarrollo con el comando:

```
$ mbed toolcahin <destino>
```

A continuación, el dispositivo destino que va destinado el firmware, que para ello se puede hacer de dos maneras, que lo detecte Mbed CLI o indicárselo ejecutando los siguientes comandos:

```
$mbed target <target>   // targer = auto -> Detección automática
```

Por último, para compilar el proyecto se ejecuta el siguiente comando:

```
$ mbed compile
```

Obteniendo como resultado un binario con el firmware si no se producen errores de compilación.

12.2. Manuales de usuario

12.2.1. Manual aplicación web

12.2.1.1. Introducción

Para poder acceder a consultar los datos de los Gadget instalados en los cultivos se tiene que acceder desde el siguiente enlace web https://sensoremterram.azurewebsites.net/ y le aparecerá una pantalla como la que se puede ver en la ilustración 93.

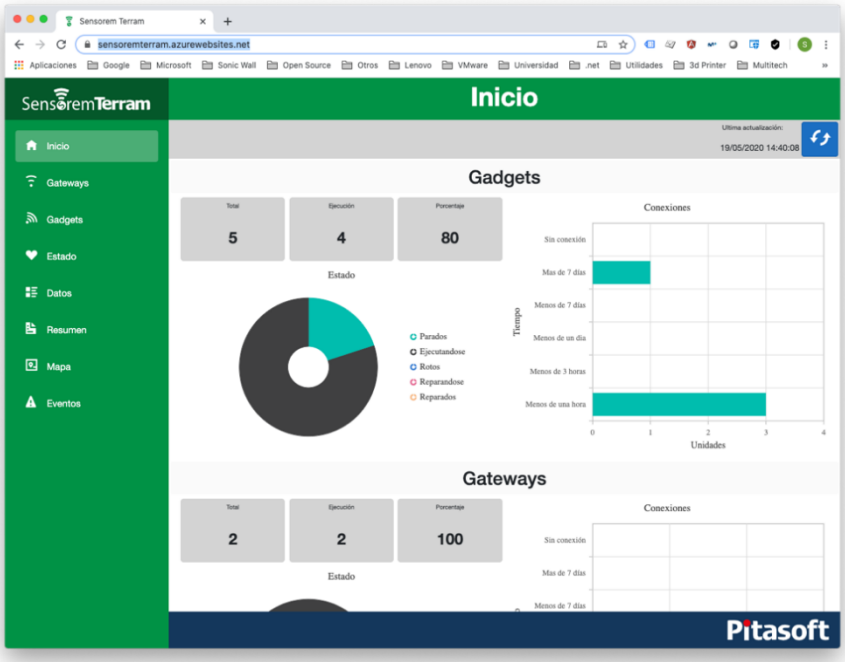

Ilustración 93 - Pantalla inicial aplicación Sensorem Terram.

Como se puede observar en la ilustración 94, la aplicación tiene dos zonas de trabajo. La parte izquierda que contiene un menú con las diferentes opciones disponibles que detallaremos una por una en las siguientes secciones y que se ha remarcado con un recuadro de color rojo. Mientras la parte derecha, que denominamos zona principal, contiene la información para consultar en cada una de las opciones y esta ha sido remarcada con un recuadro azul.

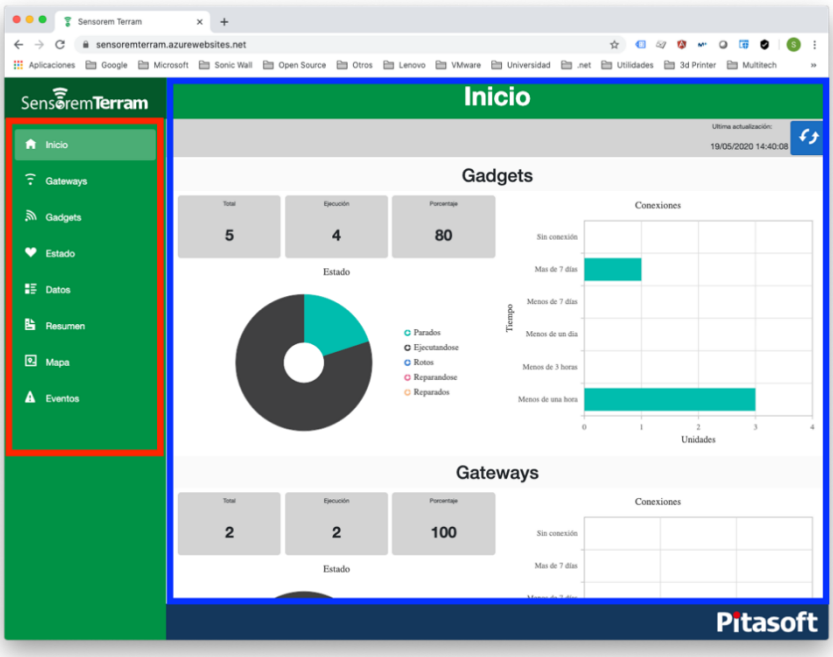

Ilustración 94 - División de zonas de pantalla aplicación.

La zona de contenido se divide en las siguientes zonas, como se puede ver en la ilustración 95 y que detallamos a continuación:

- Título, marcado con un recuadro en color azul, en ella se indica el nombre de la opción en la que nos encontramos.
- Barra de parámetros, encuadrada en color rojo, esta zona normalmente es utilizada para indicar los parámetros de búsqueda de datos en la aplicación.
- Barra de herramientas, embarcada en color verde, nos muestra la hora de consulta de datos además puede disponer de iconos que nos permite hacer las funciones de actualizar, buscar o exportar datos.
- Contenido, enmarcado en color naranja, es la zona principal donde se muestra la información.

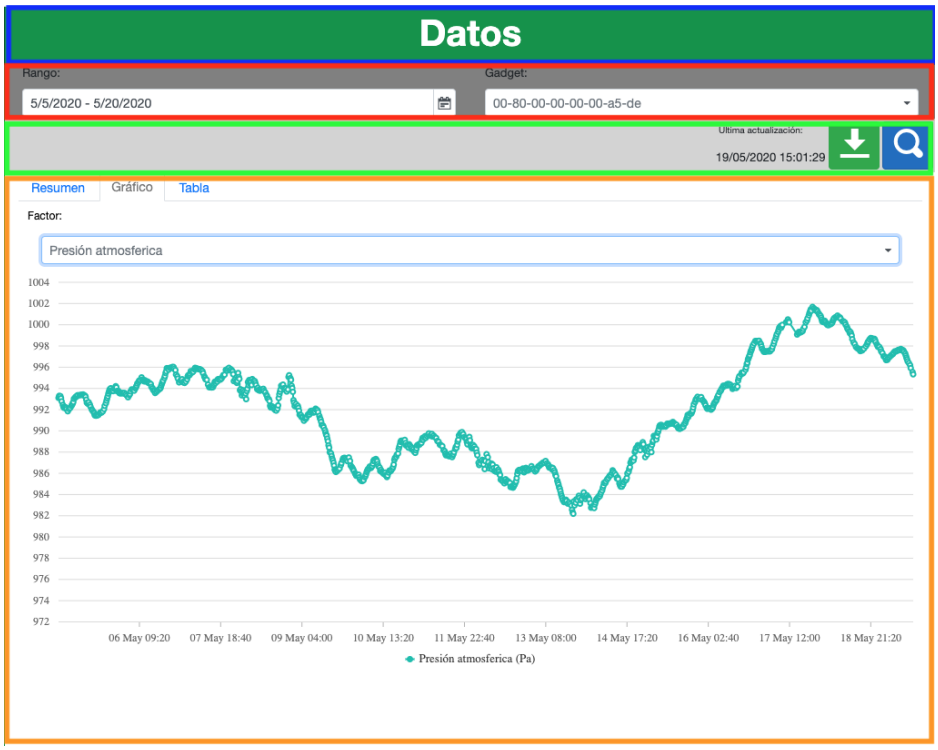

Ilustración 95 - Detalle de la zona principal.

12.2.1.2. Opción inicio

En la opción de inicio se puede visualizar un resumen del sistema, donde se puede visualizar la información de estado de los Gadget y Gateways. Conocer el número de Gadgets y Gateways que están funcionando y el estado de conexión de dichos dispositivos.

En la ilustración 96, se puede ver información sobre los Gadgets, donde se puede ver que hay 5 Gadgets funcionando, 4 en ejecución que suponen el 80% de los sistemas funcionando, esto es representado en el grafico circular.

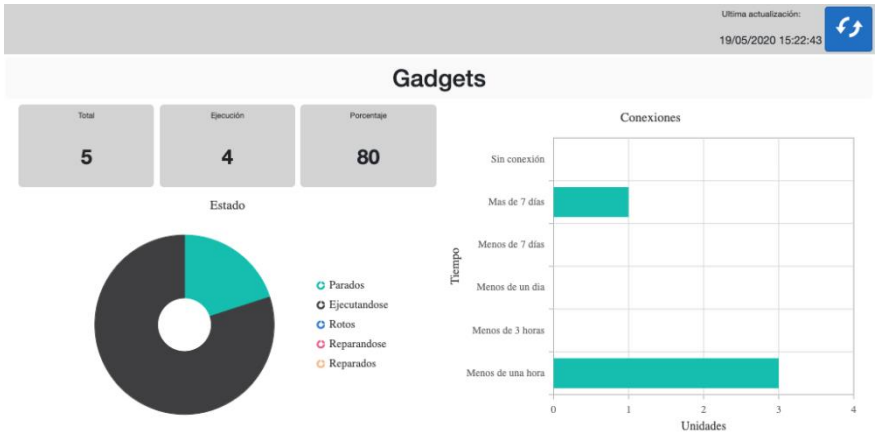

Ilustración 96 - Información de Gadgets

En la ilustración 96, se puede observar que en la barra de herramientas se dispone de un botón azul, que permite actualizar los datos de la pantalla, como la hora de actualización de los datos.

Mientas en el gráfico de barras se puede apreciar las conexiones de los sistemas, indicando cuanto tiempo hace que se han conectado los sistemas y la cantidad.

Para los Gateways tenemos la misma información que los Gadgets como se puede ver en la ilustración 97, cuantos sistemas están disponibles, que porcentaje está en ejecución y el tiempo de su última conexión.

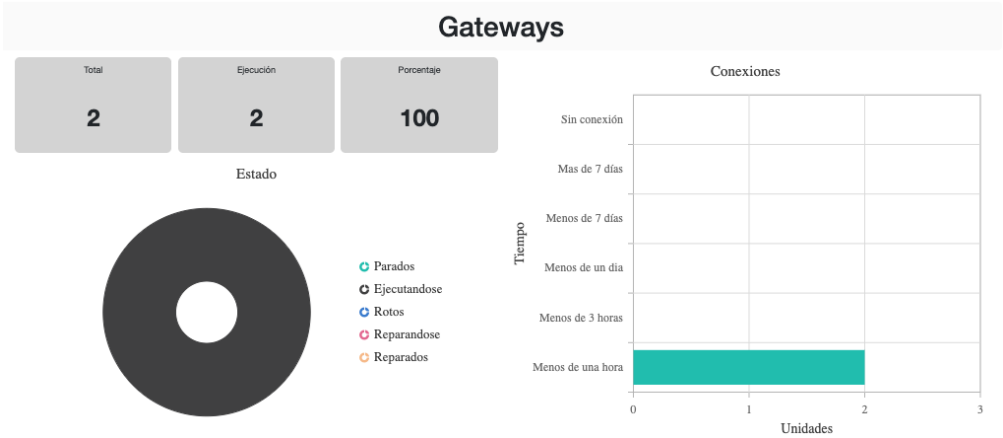

Ilustración 97 - Información de los Gateways

En la parte más inferior se encuentra la información sobre los paquetes recibidos del sistema como se puede ver en la ilustración 98. Hay un desplegable donde se puede seleccionar información a consultar entre los paquetes recibidos hoy, últimos 3 días, últimos 7 días y últimos 30 días.

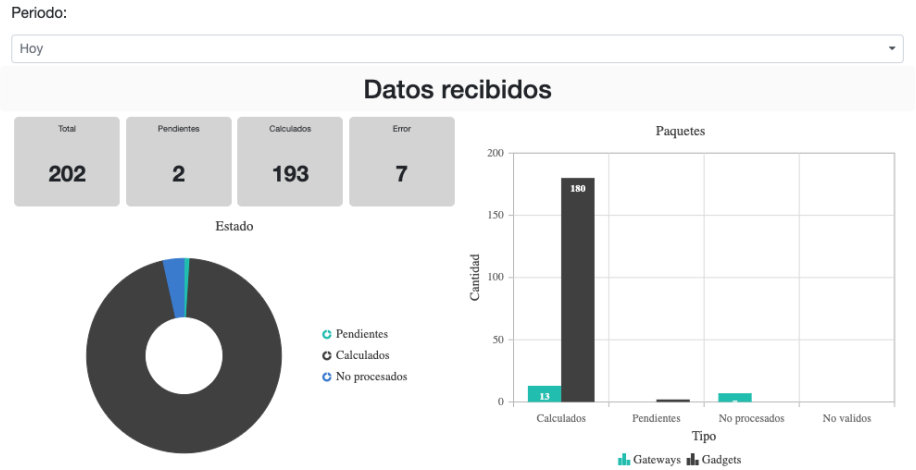

Ilustración 98 - Información de paquetes recibidos.

En la ilustración 98, se puede ver la cantidad de paquetes recibidos, el número de paquetes pendientes de procesar por el sistema y el número de paquetes que no son correctos. En el grafico circular se puede ver la representación de dichos datos y en el gráfico de barras se puede ver los diferentes tipos estado de paquetes, así como saber la cantidad de paquetes por Gadgets y Gateways.

12.2.1.3. Gateways

En la opción **Gateways** permite tener información sobre los Gateways, como se puede ver en la ilustración 99 pudiendo ver el nombre de dichos dispositivos, el identificador del dispositivo y la información de la última conexión.

Ilustración 99 - Pantalla de información de los Gateways.

En esta pantalla se pueden ordenar los datos por columnas, además de poder hacer filtros pulsando sobre el icono de filtro.

12.2.1.4. Gadgets

En la opción **Gadgets** se permite ver información sobre los Gadgets, como se puede ver en la ilustración 100. Aquí hay información de cada uno de los Gadgets registrado, con información del nombre, identificador, número de serie, versión del hardware que utiliza, ultima conexión, estado (encendido, pagado, en reparación) y nivel de batería.

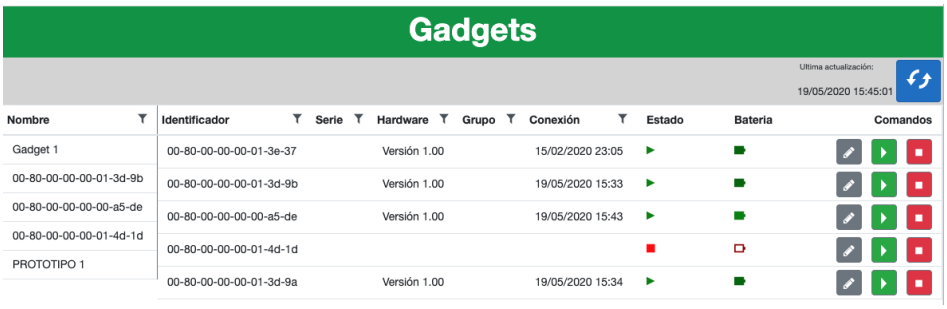

Ilustración 100 - Información de los Gadgets.

En esta pantalla se dispone de varias opciones, el botón con el icono de un lápiz de color gris permite modificar información del Gadget. Ver ilustración 101.

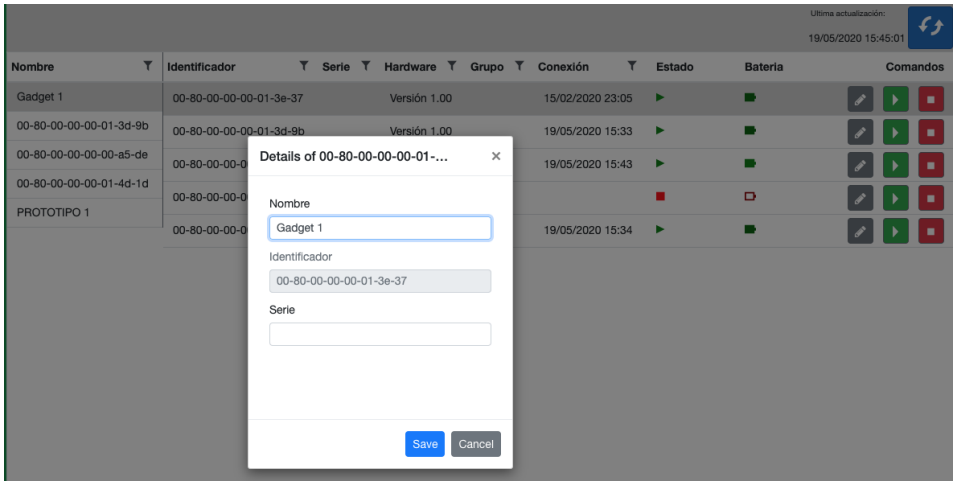

Ilustración 101 - Modificación de la información del Gadget.

Se puede cambiar el nombre del Gadget y el número de serie. Mediante el botón verde se puede poner el Gadget en modo funcionamiento, mientas que, con el botón rojo para el Gadget, es para que el sistema no procese los paquetes que envía.

12.2.1.5. Estado

La opción **Estado** permite consultar los últimos datos capturados por los Gadgets como se puede ver en la ilustración 102. En esta pantalla se puede observar para un Gadget, el nivel de la batería, la última localización, así como los datos de los últimos factores.

Nombre	Conexión	Batería	Latitud	Longitud	Altitude	Temperatura ...	Temperatura
Gadget 1	15/02/2020 23:05:18	1,88 V	37,409390 Y	-1,745609 X	208,940 m	17,88 C	17,26 C
00-80-00-00-00-01-3d-9b	19/05/2020 16:03:59	4,09 V	Y	X	m	23,69 C	23,85 C
00-80-00-00-00-00-a5-de	19/05/2020 15:58:52	3,38 V	Y	X	m	23,69 C	23,81 C
PROTOTIPO 1	19/05/2020 16:04:40	4,06 V	Y	X	m	24,88 C	24,04 C

Ilustración 102 - Información de estado de los Gadgets

12.2.1.6. Datos

La opción **Datos** puede consultar para un intervalo de fecha y un Gadget, la información capturada. En la ilustración 103, se puede ver la consulta de un Gadget para el intervalo de fecha que comprende entre el 5 de mayo del 2020 al 20 de mayo del 2020. Para realizar la consulta se tiene que pulsar una vez seleccionando los parámetros adecuados el botón azul con el icono de lupa, para realizar la búsqueda de datos en el sistema. Una vez obtenidos los datos aparecerá un nuevo botón verde, que permite descargar los datos en un archivo formato Excel.

	Temperatura	Humedad	Presión atmosferica	Luminosidad	Radiación	Temperatura suelo	Humedad suelo	Humedad suelo
Media	23,90 C	46,76 H	990,982 Pa	507,121 Lux	24 UV	27,05 C	41,81 H	3,00 H
Mediana	23,98 C	47,10 H	991,620 Pa	530,000 Lux	24 UV	27,19 C	41,79 H	3,00 H
Mínimo	22,91 C	37,79 H	982,540 Pa	446,667 Lux	24 UV	23,68 C	33,31 H	3,00 H
Máximo	24,75 C	55,77 H	999,150 Pa	530,000 Lux	24 UV	28,29 C	52,61 H	3,00 H
Rango	1,84 C	17,98 H	16,610 Pa	83,333 Lux	0 UV	4,61 C	19,30 H	0,00 H
Desviación estandar	0,42 C	4,33 H	4,314 Pa	36,688 Lux	0 UV	0,77 C	4,05 H	0,00 H
Coeficiente de variación	0,02 C	0,09 H	0,004 Pa	0,072 Lux	0 UV	0,03 C	0,10 H	0,00 H

Ilustración 103 - Información de un Gadget.

Una vez realizada la búsqueda, aparece una pantalla como la mostrada en la ilustración 103, donde se disponen de tres opciones: Resumen, Gráfico y Tabla.

En la pantalla de resumen aparece un resumen estadístico de los datos para el intervalo de tiempo seleccionado, como se puede ver en la ilustración 103. Mientras para la opción Gráfico, se permite visualizar un gráfico, para cada uno de los factores que este monitorizando el Gadget permitiendo visualizar la evolución de dicho parámetro. Ver ilustración 104. Esta pantalla tiene un desplegable que permite seleccionar el factor a visualizar.

Ilustración 104 – Gráfico evolución de un factor.

Por último, también está disponible la pestaña Tabla donde se puede visualizar los datos en bruto monitorizados por el Gadget como se puede ver en la ilustración 105.

Resumen Gráfico Tabla

Actualización	Temperatura	Humedad	Presión atmosferica	Luminosidad	Radiación	Temperatura suelo	Humedad suelo	Humedad suelo
05/05/2020 0:06:43	23,56 C	48,65 H	993,410 Pa	530,000 Lux	24 UV	27,10 C	42,80 H	3,00 H
05/05/2020 0:21:53	23,57 C	48,59 H	993,610 Pa	530,000 Lux	24 UV	27,12 C	42,63 H	3,00 H
05/05/2020 0:37:03	23,57 C	48,55 H	993,440 Pa	530,000 Lux	24 UV	27,23 C	42,34 H	3,00 H
05/05/2020 0:52:08	23,55 C	48,53 H	993,750 Pa	530,000 Lux	24 UV	27,11 C	42,60 H	3,00 H
05/05/2020 1:07:16	23,56 C	48,46 H	993,470 Pa	530,000 Lux	24 UV	27,14 C	42,44 H	3,00 H
05/05/2020 1:22:25	23,55 C	48,39 H	993,220 Pa	530,000 Lux	24 UV	27,23 C	42,17 H	3,00 H
05/05/2020 1:37:33	23,53 C	48,33 H	993,190 Pa	530,000 Lux	24 UV	27,12 C	42,37 H	3,00 H
05/05/2020 1:52:41	23,52 C	48,29 H	992,850 Pa	530,000 Lux	24 UV	27,06 C	42,49 H	3,00 H
05/05/2020 2:07:49	23,52 C	48,33 H	992,830 Pa	530,000 Lux	24 UV	27,04 C	42,52 H	3,00 H
05/05/2020 2:22:59	23,52 C	48,24 H	992,650 Pa	530,000 Lux	24 UV	27,05 C	42,41 H	3,00 H
05/05/2020 2:38:05	23,50 C	48,24 H	992,600 Pa	530,000 Lux	24 UV	27,06 C	42,35 H	3,00 H
05/05/2020 2:53:15	23,50 C	48,22 H	992,540 Pa	530,000 Lux	24 UV	27,02 C	42,40 H	3,00 H
05/05/2020 3:08:23	23,50 C	48,18 H	992,480 Pa	530,000 Lux	24 UV	27,11 C	42,15 H	3,00 H
05/05/2020 3:23:30	23,49 C	48,16 H	992,400 Pa	530,000 Lux	24 UV	27,04 C	42,27 H	3,00 H
05/05/2020 3:38:40	23,48 C	48,09 H	992,400 Pa	530,000 Lux	24 UV	27,09 C	42,11 H	3,00 H

Ilustración 105 - Tabla de factores de un Gadget.

12.2.1.7. Resumen

En la opción **Resumen**, permite para un intervalo de fecha, un Gadget y un factor, un resumen. En la ilustración 106 se puede ver un ejemplo de consulta de parámetros.

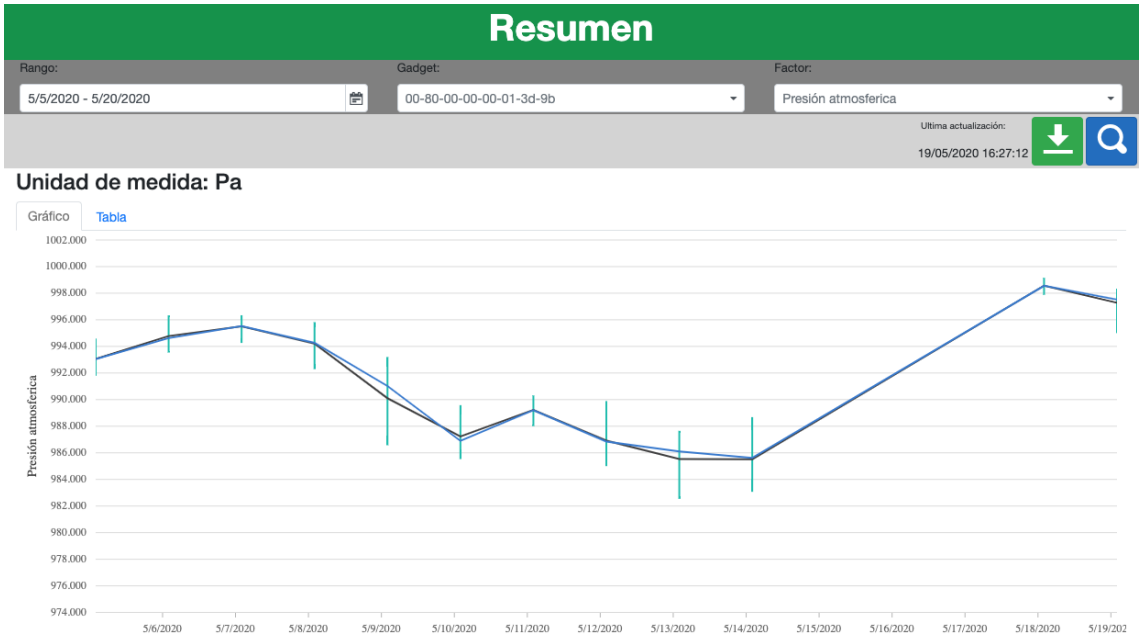

Ilustración 106 - Resumen de un Gadget y factor.

En la ilustración 106, se puede ver para el intervalo de fecha del 5 de mayo del 2020 al 20 de mayo del 2020, para un Gadget en concreto y el factor Presión atmosférica la gráfica con la evolución que ha desarrollado. En donde se pude ver una línea de color azul que representa la mediana y la de color negro que representa la media, además de ver el rango por día, es decir el valor máximo y mínimo por día.

También hay disponible la pestaña Tabla que permite visualizar los datos estadísticos por día como se puede ver en la ilustración 107. Además, si pulsamos sobre el botón verde permite descargar en un archivo Excel la información consultada.

Unidad de medida: Pa

Gráfico | Tabla

Fecha	Mínimo	Máximo	Rango	Media	Mediana	Varianza	Desviación estandard	Coeficiente de variación
05/05/2020	991,790	994,590	2,800	993,066	993,040	0,616	0,785	0,001
06/05/2020	993,540	996,330	2,790	994,775	994,620	0,637	0,798	0,001
07/05/2020	994,260	996,340	2,080	995,502	995,520	0,302	0,550	0,001
08/05/2020	992,290	995,820	3,530	994,209	994,270	0,808	0,899	0,001
09/05/2020	986,570	993,200	6,630	990,095	991,020	4,793	2,189	0,002
10/05/2020	985,530	989,580	4,050	987,223	986,900	1,308	1,143	0,001
11/05/2020	988,010	990,310	2,300	989,216	989,210	0,443	0,666	0,001
12/05/2020	985,000	989,890	4,890	986,911	986,840	1,467	1,211	0,001
13/05/2020	982,540	987,650	5,110	985,540	986,100	2,582	1,607	0,002
14/05/2020	983,070	988,680	5,610	985,507	985,620	1,876	1,370	0,001
18/05/2020	997,880	999,150	1,270	998,564	998,560	0,181	0,426	0,000
19/05/2020	995,000	998,340	3,340	997,280	997,520	0,755	0,869	0,001

Ilustración 107 - Tabla resumen de un Gadget y factor.

12.2.1.8. Mapa

En la opción **Mapa** se puede consultar la localización de los Gadget y Gateways como se puede ver en la ilustración 108.

Ilustración 108 - Localización geográfica.

Se puede apreciar que existen dos tipos de dispositivos, los iconos de color rojo representan los Gadget y los de color azul son los Gateways. Sobre estos iconos se puede pulsar para obtener información de estado de cada uno de ellos como se puede ver en la ilustración 109, donde la ilustración A es el estado de un Gadget y la ilustración B es el estado de un Gateway.

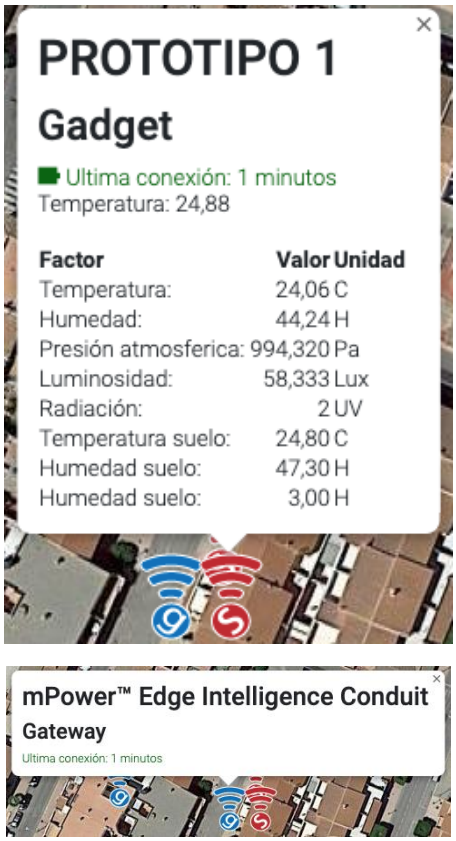

Ilustración 109 - Estado de un sistema.

12.2.1.9. Eventos

En la opción de **Eventos**, de puede visualizar los eventos del sistema que se están registrando como se puede ver en la ilustración 110.

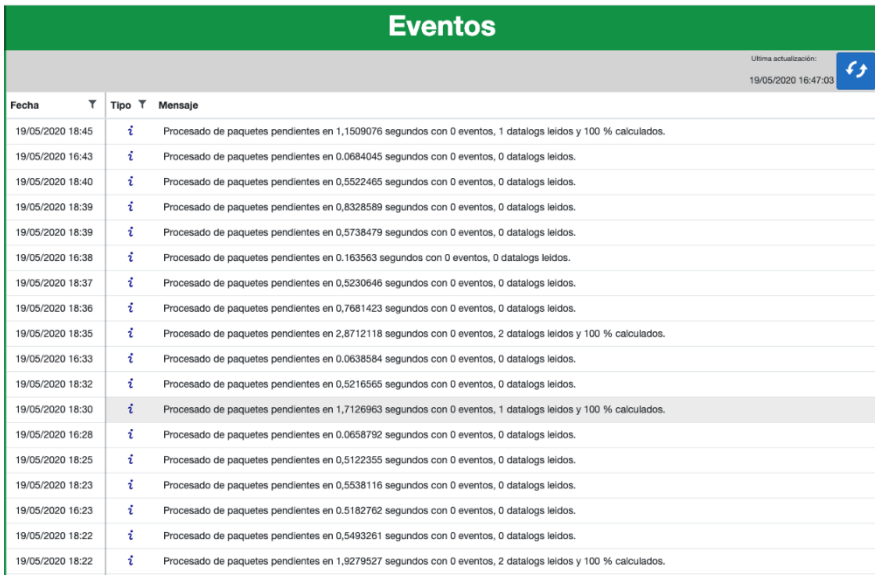

Ilustración 110 - Información de eventos.

En esta pantalla se pueden ver los eventos producidos en el sistema, como el tipo de evento que se ha producido que puede ser de tres tipos: Notificación, Advertencia, Error o Critico. Así como la información sobre el evento producido.

12.2.2. Manual aplicación móvil

La aplicación es compatible con sistema Android y iOS, proporcionando acceso al sistema Sensorem Terram.

12.2.2.1. Introducción

Para poder acceder a consultar los datos de los Gadget instalados en los cultivos se tiene que acceder desde el icono de la aplicación, que dependiendo del sistema operativo de su dispositivo puede ser diferente, como se muestra en la ilustración 111.

Ilustración 111 - Iconos de aplicación Sensorem Terram.

Una vez pulsado sobre dicho icono la aplicación arrancará y aparecerá una pantalla como la que se puede observar en la ilustración 112.

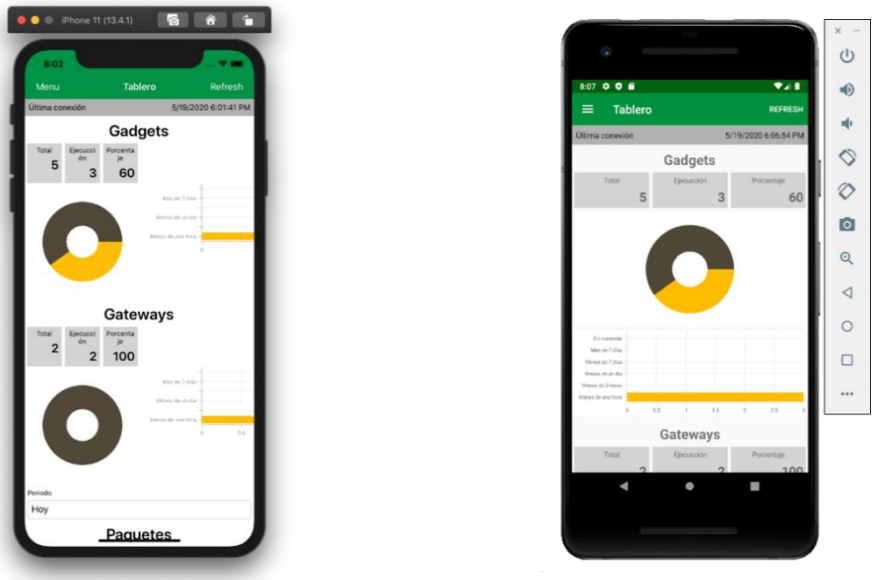

Ilustración 112 - Pantalla inicial de Sensorem Terram

Como se puede observar en la ilustración 112, la aplicación dispone toda la pantalla donde se puede consultar la información, esta zona la denominamos zona de principal, donde en cada una de las opciones disponibles del menú se mostrará la información solicitada.

En la parte superior la aplicación siempre dispone de la opción menú que mostrara las opciones disponibles como se puede ver en la ilustración 113.

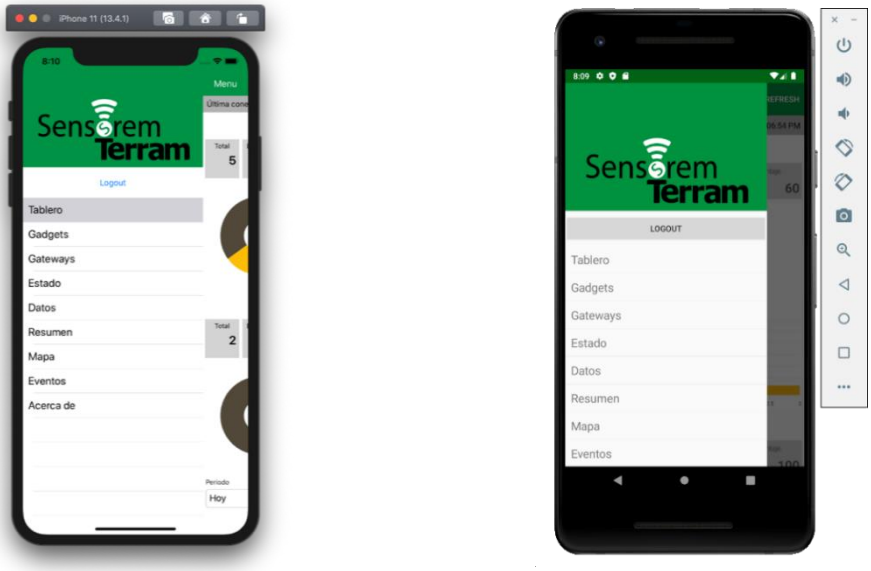

Ilustración 113 - Menú aplicación Sensorem Terram.

Dependiendo de la opción seleccionada, en la zona principal aparecerá la pantalla para poder consultar la información deseada.

12.2.2.2. Opción inicio

En la opción de inicio se puede visualizar un resumen del sistema, donde se puede visualizar la información de estado de los Gadget y Gateways. Conocer el número de Gadgets y Gateways que están funcionando y el estado de conexión de dichos dispositivos.

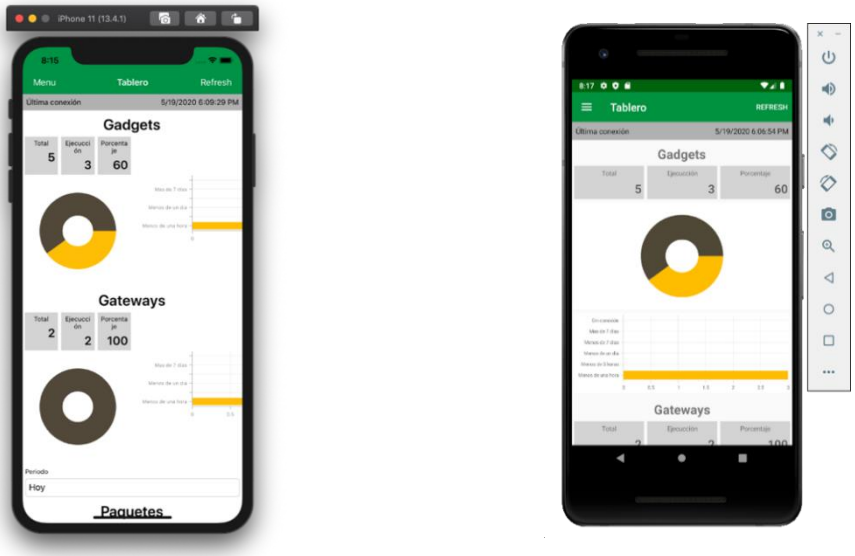

Ilustración 114 - Tablero de mandos.

En la ilustración 114, se puede ver información sobre los Gadgets, donde se puede ver que hay 5 Gadgets funcionando, 3 en ejecución que suponen el 60% de los sistemas funcionando, esto es representado en el grafico circular. Además, también se puede observar que en la barra de herramientas se dispone de un botón llamado **Refresh** que permite actualizar los datos de la pantalla, como la hora de actualización de los datos.

Mientas en el gráfico de barras se puede apreciar las conexiones de los sistemas, indicando cuanto tiempo hace que se han conectado y la cantidad.

Para los Gateways tenemos la misma información que los Gadgets como se puede ver en la ilustración 114, cuantos sistemas están disponibles, que porcentaje está en ejecución y el tiempo de su última conexión.

Desplazando la pantalla hacia arriba se encuentra la información sobre los paquetes recibidos del sistema como se puede ver en la ilustración 98. Hay un desplegable donde se puede seleccionar información a consultar entre los paquetes recibidos hoy, últimos 3 días, últimos 7 días y últimos 30 días.

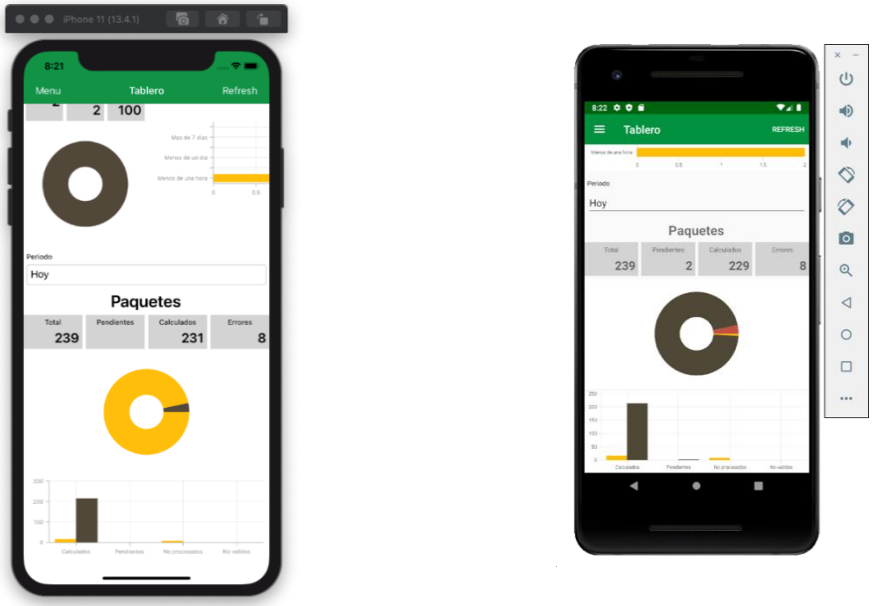

Ilustración 115 - Información de los paquetes recibidos.

En la ilustración 115, se puede ver la cantidad de paquetes recibidos, el número de paquetes pendientes de procesar por el sistema y el número de paquetes que no son correctos. En el grafico circular se puede ver la representación de dichos datos y en el gráfico de barras se puede ver los diferentes tipos estado de paquetes, así como saber la cantidad de paquetes por Gadgets y Gateways.

12.2.2.3. Gateways

En la opción **Gateways** permite tener información sobre los Gateways, como se puede ver en la ilustración 116 pudiendo ver el nombre de dichos dispositivos, el identificador del dispositivo y la información de la última conexión.

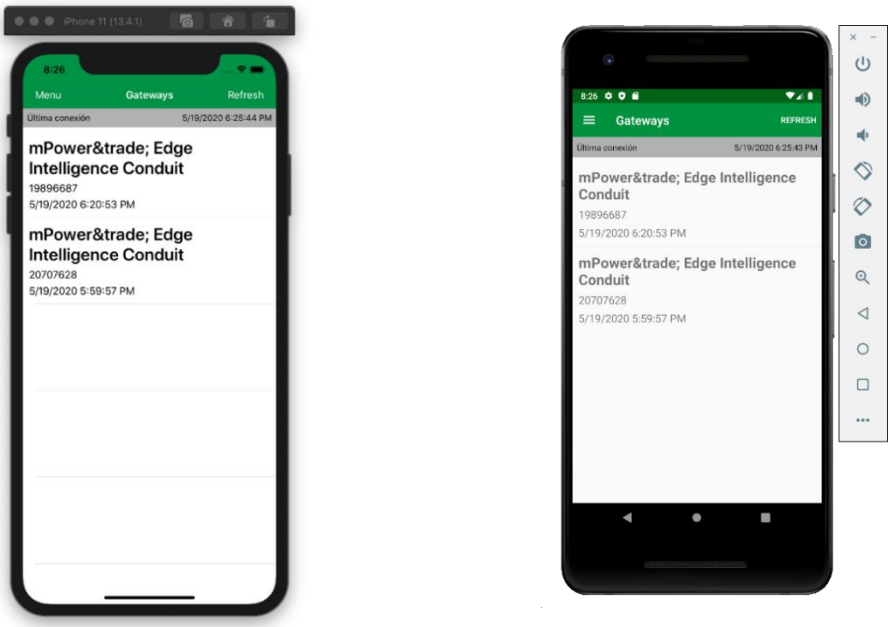

Ilustración 116 - Opción Gateways.

12.2.2.4. Gadgets

En la opción **Gadgets** se permite ver información sobre los Gadgets, como se puede ver en la ilustración 117. Aquí hay información de cada uno de los Gadgets registrado, con información del nombre, identificador, número de serie, versión del hardware que utiliza, ultima conexión.

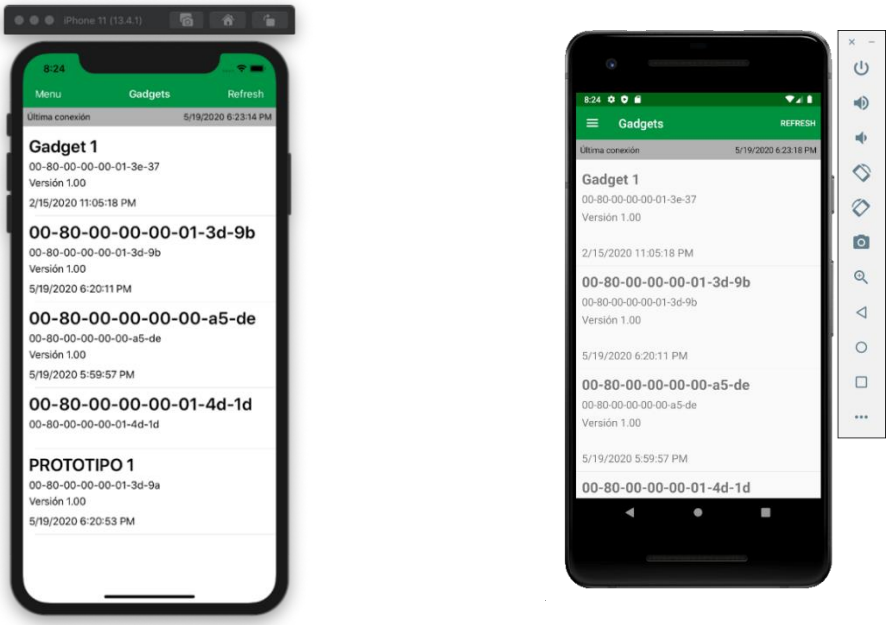

Ilustración 117 - Opción Gadgets

12.2.2.5. Estado

La opción **Estado** permite consultar los últimos datos capturados por los Gadgets como se puede ver en la ilustración 118. En esta pantalla se puede observar para un Gadget, el nivel de la batería, la última localización, así como los datos de los últimos factores.

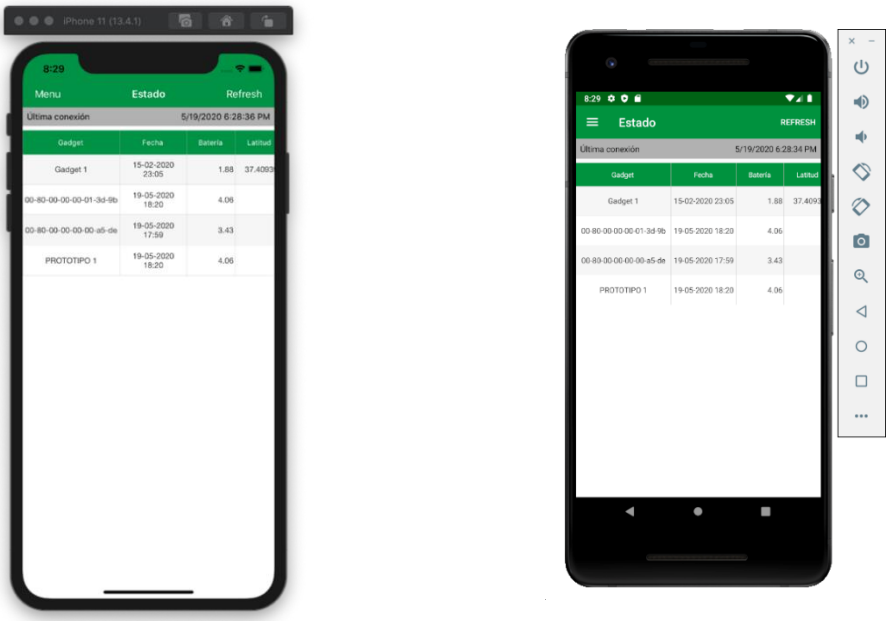

Ilustración 118 - Opción estado de los Gadgets

12.2.2.6. Datos

La opción **Datos** se puede consultar para un intervalo de fecha y un Gadget, la información capturada. Inicialmente aparece una pantalla como la que se puede ver en la ilustración 119, donde se piden los parámetros que se quieren consultar.

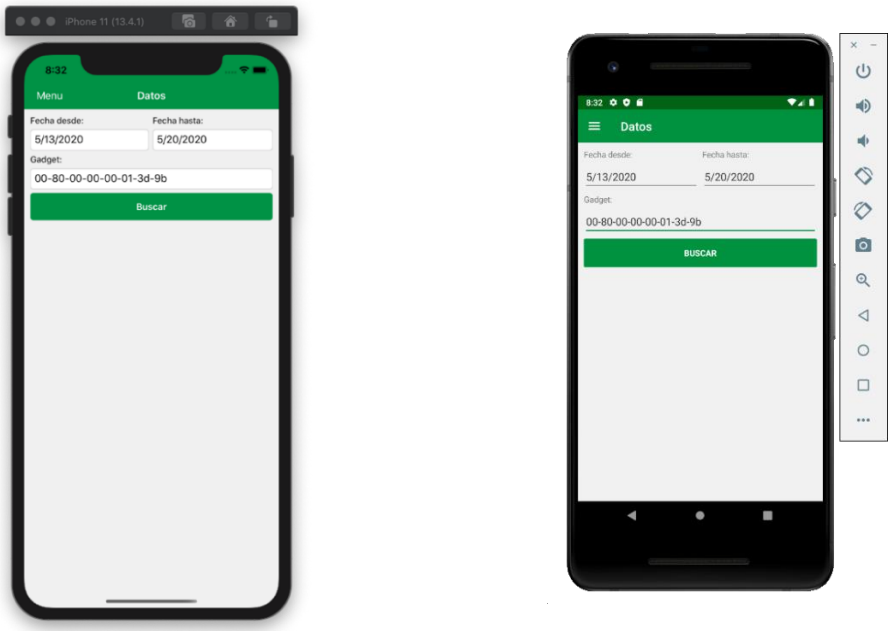

Ilustración 119 - Pantalla de parámetros de datos.

Una vez introducidos los datos deseados se realiza la búsqueda pulsando sobre el botón búsqueda, aparece una pantalla como la mostrada en la ilustración 120, donde se disponen de tres opciones: Resumen, Gráfico y Tabla.

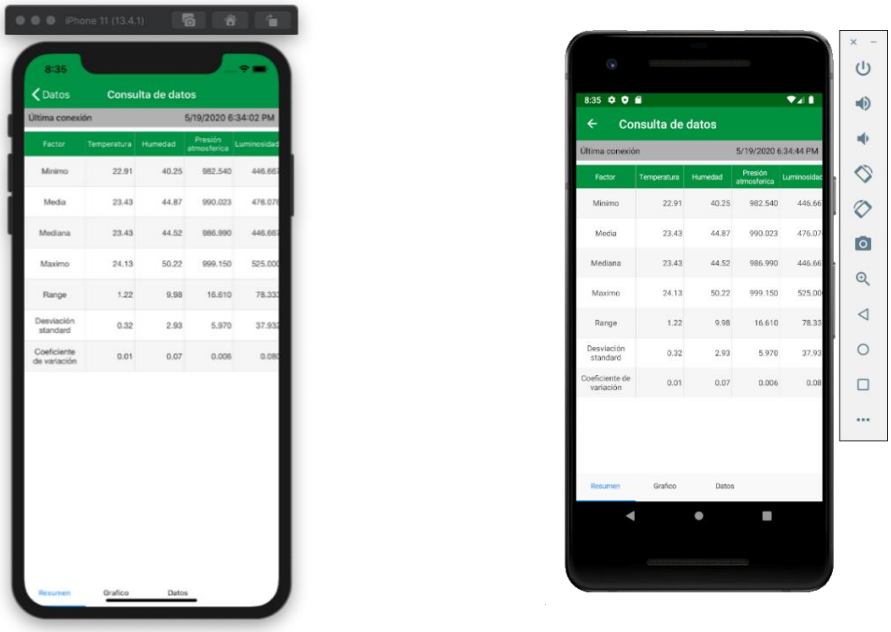

Ilustración 120 - Consulta de datos

En la ilustración 120, se puede ver la consulta de un Gadget para el intervalo de fecha que comprende entre el 13 de mayo del 2020 al 20 de mayo del 2020. En la pantalla de resumen aparece un resumen estadístico de los datos

para el intervalo de tiempo seleccionado, como se puede ver en la ilustración 120. Mientras para la opción Gráfico, se permite visualizar un gráfico, para cada uno de los factores que este monitorizando el Gadget permitiendo visualizar la evolución de dicho parámetro. Ver ilustración 121. Esta pantalla tiene un desplegable que permite seleccionar el factor a visualizar.

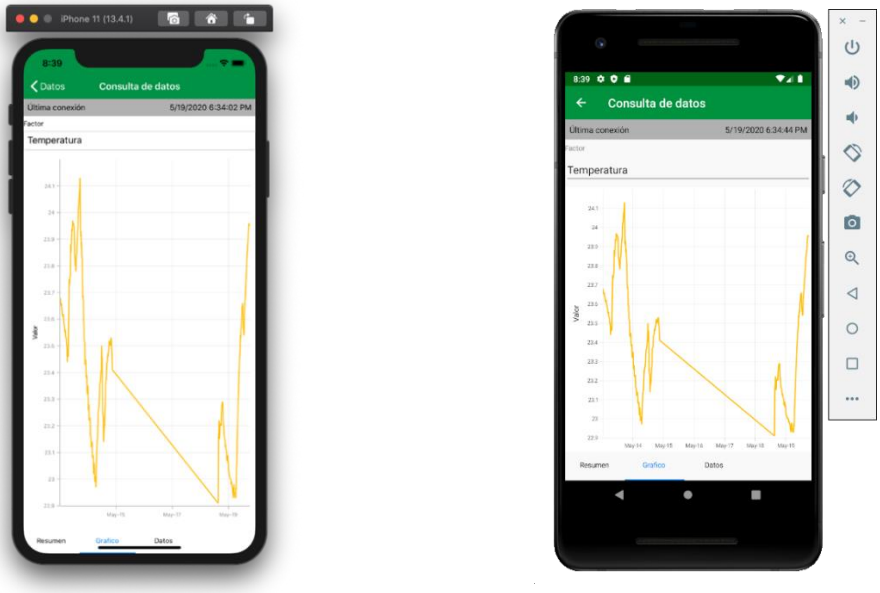

Ilustración 121 - Grafico de temperatura.

Por último, también está disponible la pestaña Tabla donde se puede visualizar los datos en bruto monitorizados por el Gadget como se puede ver en la ilustración 122.

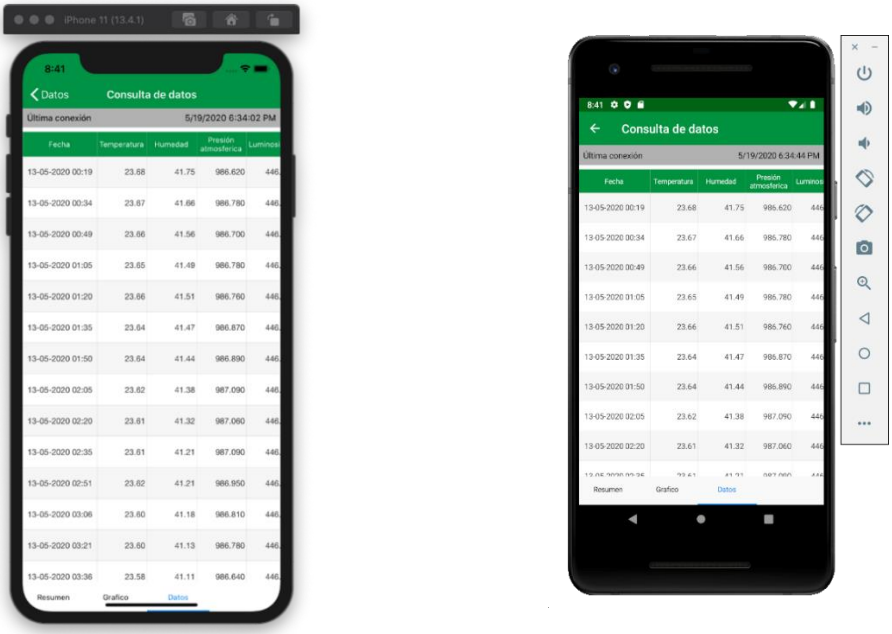

Ilustración 122 - Tabla de datos de un Gadget.

12.2.2.7. Resumen

En la opción **Resumen**, permite para un intervalo de fecha, un Gadget y un factor, un resumen. Inicialmente aparecerá una pantalla donde se solicita dichos parámetros como se puede ver en la ilustración 123.

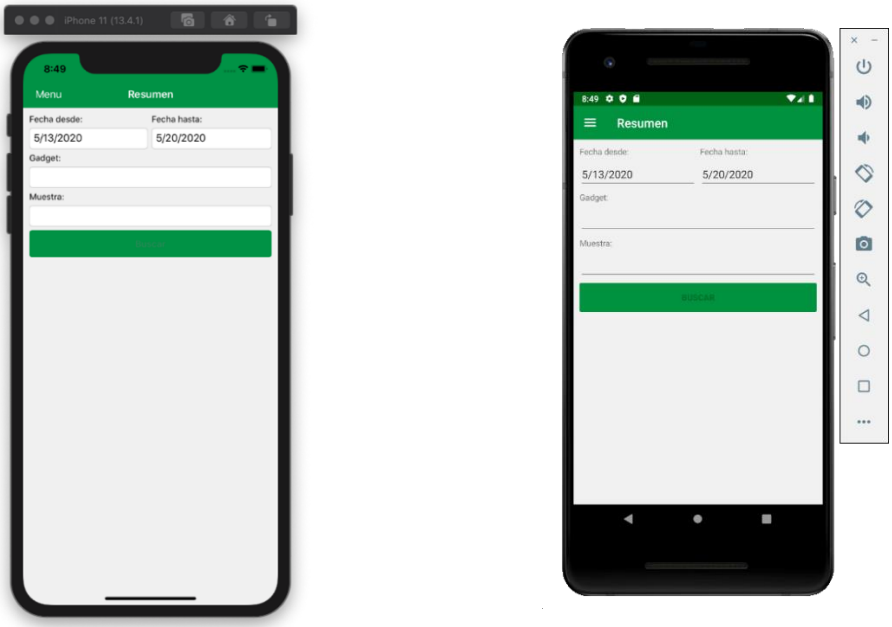

Ilustración 123 - Pantalla de parámetros de resumen.

Una vez introducidos los parámetros adecuados, se pulsa sobre buscar y apareceré una nueva pantalla como la que se puede ver en la ilustración 124. Se puede ver para el intervalo de fecha del 13 de mayo del 2020 al 20 de mayo del 2020, para un Gadget en concreto y el factor Presión atmosférica la gráfica con la evolución que ha desarrollado. En donde se puede ver una línea de color negra que representa la mediana y la de color rojo que representa la media, además de ver el rango por día coloreado en amarillo, es decir el valor máximo y mínimo por día.

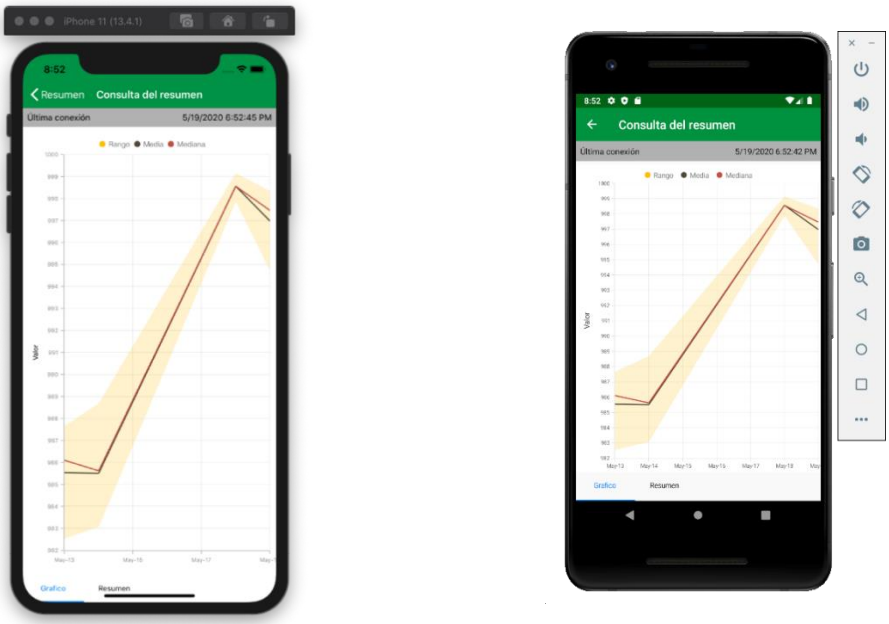

Ilustración 124 - Grafico de presión atmosférica resumen.

También hay disponible la pestaña Resumen que permite visualizar los datos estadísticos por día como se puede ver en la ilustración 125.

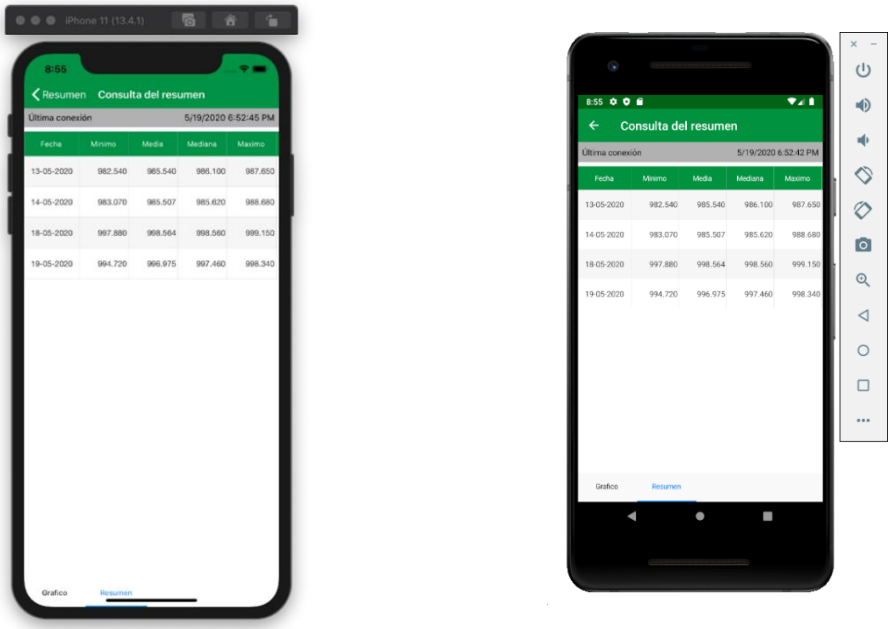

Ilustración 125 - Tabla resumen.

12.2.2.8. Mapa

En la opción **Mapa** se puede consultar la localización de los Gadget y Gateways como se puede ver en la ilustración 126.

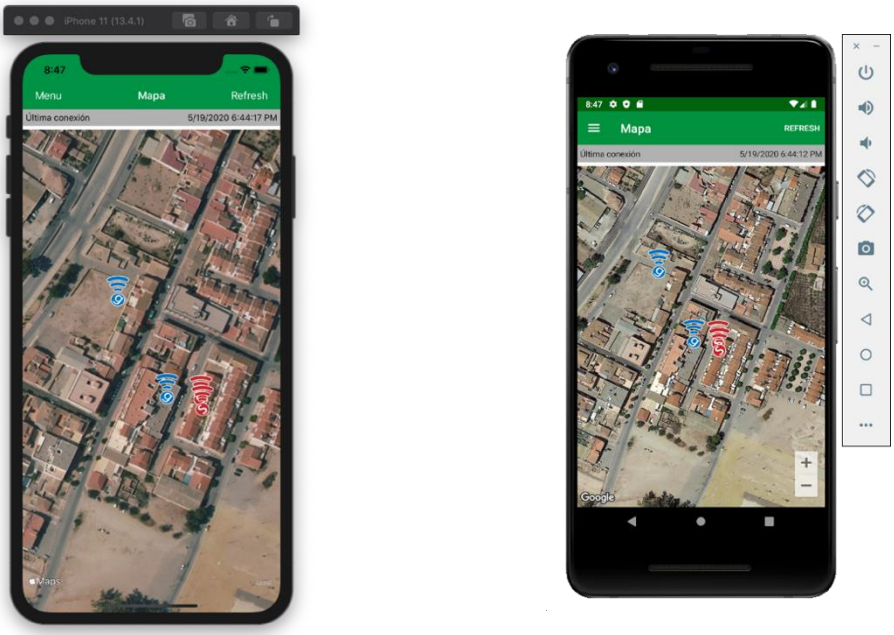

Ilustración 126 - Localización de Gadgets y Gateways.

Se puede apreciar que existen dos tipos de dispositivos, los iconos de color rojo representan los Gadget y los de color azul son los Gateways. Sobre

cada uno de estos iconos se puede pulsar para obtener una viñeta de información de estado como se puede ver en la ilustración 127.

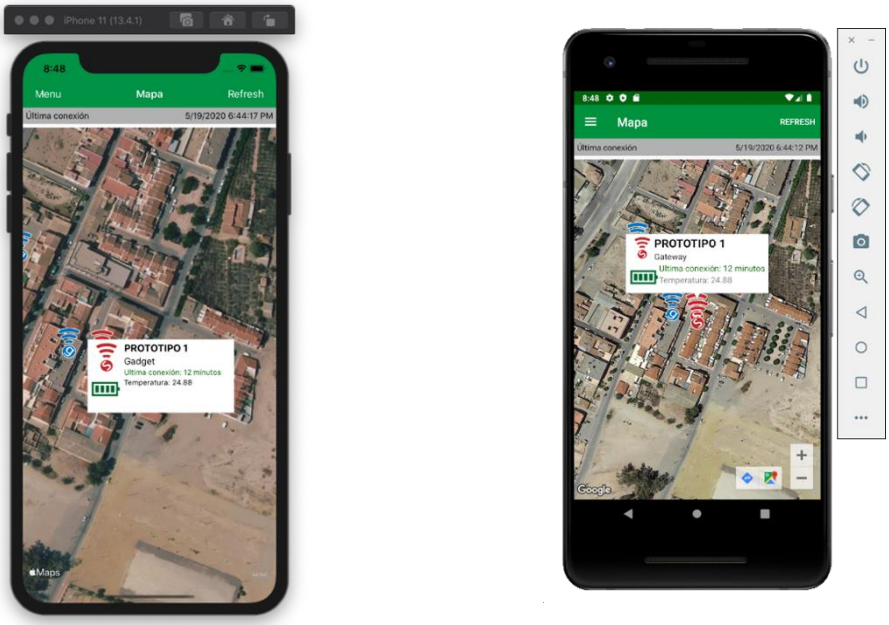

Ilustración 127 - Viñeta de estado de un sistema.

Además, si se pulsa sobre esta viñeta de información aparece una pantalla detallando la información del estado de Gadget o del Gateway. Ver ilustración 128.

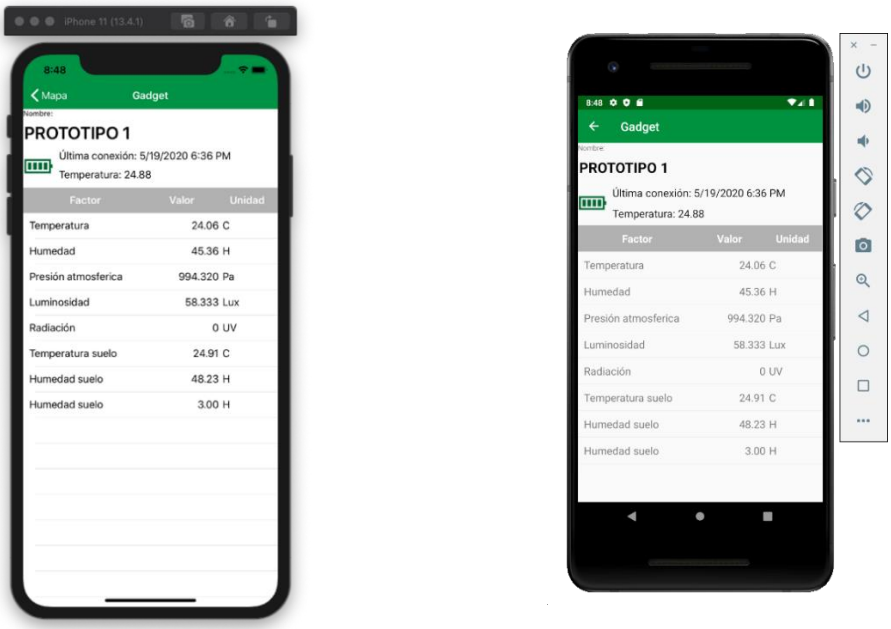

Ilustración 128 - Estado de un Gadget.

12.2.2.9. Eventos

En la opción de **Eventos**, se puede visualizar los eventos del sistema que se están registrando como se puede ver en la ilustración 129.

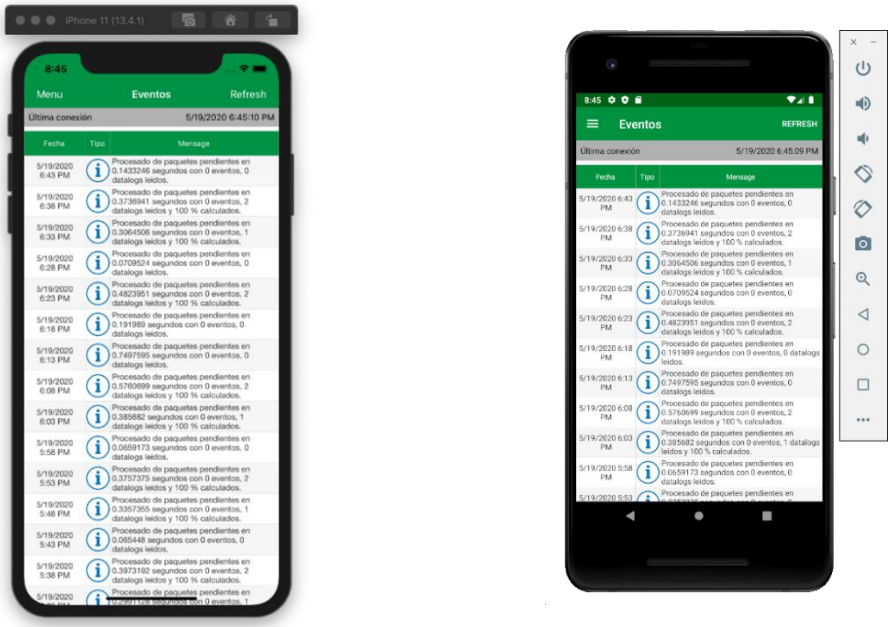

Ilustración 129 - Pantalla de eventos.

En esta pantalla se pueden ver los eventos producidos en el sistema, como el tipo de evento que se ha producido que puede ser de tres tipos: Notificación, Advertencia, Error o Critico. Así como la información sobre el evento producido.

12.3. Otros documentos, manuales

12.3.1. Extreme Programming (XP)

La metodología de Extreme Programming o Programación Extrema está basada en el manifiesto ágil y destaca por que enfatiza la satisfacción del cliente, entregando a este la parte que necesita a medida que lo necesita, la entrega del producto será de forma incremental. Permitiendo además adaptarse a los requisitos cambiantes de los clientes. Para la metodología XP las pruebas son la base de la construcción, propone que los desarrolladores sean los que escriban las pruebas a medida que van construyendo el código, realizándose una integración continua, consiguiendo que el software sea estable. Las pruebas automáticas se realizan de forma constante, lo antes posible, frecuentes y automatizadas, para poder detectar los fallos lo antes posible y resolverlos sin que las consecuencias e impactos se incrementen.

Antes de cada iteración se planifica el trabajo que se va a realizar y seguidamente se realizan de forma simultánea el análisis, el desarrollo, el diseño y las pruebas de código.

La metodología XP enfatiza el trabajo en grupo, participando gerentes, clientes y desarrolladores para obtener los resultados deseados por el proyecto. El trabajo en equipo debe continuar incluso si se cambian las reglas para ajustarse a las necesidades específicas de la empresa.

La metodología XP se basa en un conjunto de ideales a los que llama valores a la hora de desarrollar un proyecto, que son los siguientes:

- Comunicación.
- Simplicidad.
- Retroalimentación (Feeback).
- Respeto.
- Valentía.

La metodología XP propone algunos principios útiles para un mejor desarrollo, que son los siguientes:

- Humanidad.
- La economía.
- La búsqueda del beneficio mutuo.
- La auto semejanza.
- Mejora continua.
- Diversidad.
- Reflexión.
- Simultaneidad de fases o flujo.
- Oportunidad.
- Redundancia buscando soluciones.
- Aprender de los fallos.
- Búsqueda constante de la calidad.
- Avanzar con pequeños pasos.
- Aceptar la responsabilidad de todos los implicados en el desarrollo del producto.

La aplicación de estos valores y principios en un proyecto se hará aplicando las practicas primarias y practicas corolario que la metodología XP propone.

A continuación, enumero las prácticas primarias:

- Trabajar con historias de usuario.
- Realizar ciclos semanales de desarrollo.
- Organizar revisiones trimestrales.
- Trabajar con holgura.
- El equipo debe sentarse junto.
- El equipo debe ser completo, es decir, estar compuesto por todas las personas necesarias para llevar a cabo el producto con éxito.
- Tener información sobre el proyecto en el puesto de trabajo.
- Mantener la energía en el trabajo a un ritmo sostenible.

- Realizar con frecuencia la programación en parejas.
- Diseño incremental.
- Realizar las pruebas antes de programar.
- Construir en diez minutos.
- Integración continua.

Y las practicas corolario son:

- Participación real de los clientes.
- Despliegue incremental.
- Negocie el alcance del contrato.
- Pague por funcionalidad.
- Continuidad de los equipos.
- Reducir los equipos.
- Análisis de las causad.
- Código y pruebas.
- Código compartido.
- Código base único.
- Despliegue diario.

En la metodología XP se proponen una serie de roles, que pueden ser combinados por una misma persona, aunque hay algunos roles que no deberían ser lo. Estos roles serían los siguientes:

- **Traker**. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real consumido para mejorar los resultados de futuras estimaciones.
- **Customer**. Es el cliente que escribe las historias de usuario asignado las prioridades, además de realizar las pruebas funcionales de la implementación presentada.
- **Programmer**. Es el encargado de escribir las pruebas unitarias y el código del sistema.

- **Coach**. Guía a los miembros del equipo para seguir el proceso correctamente.
- **Tester**. Se encarga de ejecutar las pruebas regularmente difundiendo los resultados en el equipo además de ser responsable de las herramientas de soporte para las pruebas.
- **Big Boss**. Coordina la comunicación entre clientes y programadores.
- **Consultor**. Miembro externo del equipo con conocimiento en un tema en concreto que ayuda al equipo a resolver un problema específico.
- **Manager**. Encargado de organizar las reuniones, registrar los resultados de las reuniones y trae la información importante a estas, además de intentar mantener al equipo motivado.
- **Doomnsayer**. Se asegura que todo el mundo conozca los riesgos, que las malas noticias no se oculten ni se pasen por alto y que no se magnifiquen.

También existen anti-roles en la metodología XP que tienden a meter ruido en la realización del proyecto, que son:

- **Standards and Metodology Guy**. Puede indicar que se está haciendo mal.
- **GoodHead**. Memoriza detalles del sistema para entrar en discusión acerca del diseño además de tomar responsabilidad en todo lo que ver.

La estructura de la metodología XP seria la siguiente:

- **Planificación**. Consiste en definir los requisitos de las historias escritas por el cliente, poniendo todos los requisitos necesarios y dándole la prioridad que tiene para el negocio. Se crean historias de usuario que describen el software que se va a desarrollar. Los clientes y desarrolladores trabajaran juntos para agrupar las historias de la siguiente entrega. Además, después de la entrega el equipo XP debe calcular la velocidad de trabajo y ver la cantidad de historia que han sido capaces de implementar, para ayudar a estimar la fecha de la entrega final. A medida que el proyecto avanza, el usuario puede introducir nuevas historias,

modificar las realizadas, descomponerlas o eliminarlas y el equipo XP tiene que estar preparado para cualquier cambio.

- **Diseño**. Implementación de una historia tal y como se describe. Si en el diseño la historia es muy compleja, la metodología XP recomienda la creación de un prototipo operativo con el objetivo de disminuir el riego cuando comience la implementación.
- **Codificación**. Antes de empezar la codificación se realizan una serie de pruebas unitarias a cada una de las historias que se va a incluir en la entrega en curso. A continuación, el desarrollador tiene la capacidad para centrarse en lo que tiene que implementar para superar las pruebas unitarias. Además, la metodología XP recomienda la programación en parejas, que dos personas trabajen juntas en un ordenador con el objetivo de crear código para una historia, donde cada persona adopta un papel diferente. A medida que las parejas de programadores terminan su trabajo, el código desarrollado se integra con los trabajos de los demás.
- **Pruebas**. Las pruebas unitarias que se crean deben de poder implementarse para poder realizarlas automáticamente permitiendo ejecutarse varias veces y fácilmente, estimulando una estrategia de pruebas de regresión. Permitiendo lanzar pruebas de integración y validación a diario, dando al equipo XP una indicación continua y detectar señales de alerta si algo falla. Las pruebas de aceptación XP, también llamadas pruebas de cliente son especificadas por el cliente y se centran en las características y funcionalidades generales del sistema que son visibles y revisables por el cliente, que se han basado en la implementación de las historias de usuarios que se ha desarrollado como parte de la liberación del proyecto.

Ilustración 130 - Metodología de la programación extrema. Imagen obtenida de http://www.diegocalvo.es

Para la elaboración de este apartado la información ha sido obtenida de las siguientes referencias (Extreme Programing, 2019), (Carmen Lasa Gómez, 2018) y (López Gil, 2018).

12.3.2. SCRUM

La metodología SCRUM está también basada en el manifiesto ágil y el objetivo de esta metodología es planificar y controlar proyectos con un contexto complejo.

La metodología SCRUM establece un método para trabajar en equipo a partir de un Product Backlog del proyecto que se desea desarrollar, que está conformado por una lista de requisitos que el Product Owner establece en orden de prioridad.

El equipo SCRUM extrae un trozo de alguna de estas funcionalidades, definiendo la cantidad a realizar en cada una de las iteraciones o Sprints. Estas iteraciones ocupan un espacio cerrado de tiempo entre una a cuatro semanas, con revisiones diarias, el equipo Scrum se concentrará en desarrollar el conjunto de funcionalidades que se comprometieron a realizar.

Al terminar cada sprint o iteración se obtiene como resultado un incremental del producto potencialmente utilizable, que será resisando para su validación y en función de los resultados obtenidos, se priorizan y planificaran las actividades del siguiente sprint.

La metodología Scrum se centra en ajustar sus resultados y responder a las exigencias reales del cliente, que va revisando en cada entrega para comprobar que cumple los requisitos establecidos.

La metodología Scrum se basa en los siguientes principios:

- **Autoorganización y colaboración**. El equipo se gestiona y organiza, lo que implica responsabilidad y un gran nivel de compromiso por parte de todos. Los líderes y clientes colaborarán igualmente con el equipo de desarrollo en todo momento, facilitando el trabajo, resolviendo dudas y eliminado posibles impedimentos.
- **Priorización**. Los requisitos tienen que estar priorizados.

- **Inspección y adaptación**. Se trabajan con iteraciones llamadas Sprints, que termina cuando la iteración se convierte en un producto entregable, mostrándolo al cliente para que lo valore. Con cada entrega adquirimos más experiencia, permitiéndonos mejorar iteración tras iteración.
- **Mantener un latido**. El latido marca la pauta del trabajo ayudando a los equipos a optimizar su trabajo, permitiendo al equipo ser predecible para conocer la cantidad de trabajo que puede comprometerse.

La metodología SCRUM define unos roles con las siguientes características, que representa una responsabilidad en el proceso y no la posición dentro de la organización:

- **Product owner**. Es el encargado que el proyecto se desarrolle conforme la estrategia establecida. Escribe las historias de usuario, las coloca en el Producto Backlog y las ordena por prioridad. Debe saber la velocidad que el equipo está trabajando, para realizar estimaciones para conocer cuando estará implementada la solución.
- **Scrum Master**. Participa en las reuniones para asegurarse que se cumpla el tiempo y el objetivo establecido. Ayuda al equipo que cumpla con su objetivo pues es el líder del equipo, pero no gestiona el equipo, pues el grupo es autogestionado. También es responsable que asegurar que siguen los valores, prácticas y normas de SCRUM.
- **Equipo**. Se recomienda que sea un equipo multidisciplinar, compuesto entre 5 a 10 miembros, con las habilidades necesarias para crear el incremento. Este debe ser autoorganizado, nadie dice al equipo como convertir el Product Backlog en incremento de funcionalidad entregable.

La metodología SCRUM define los siguientes artefactos:

- **Product Backlog** (Pila del Producto). Lista de requisitos del Producto Owner, que se define inicialmente, pero a medida que se va desarrollando el proyecto puede ir cambiando para añadir otros requisitos.
- **Sprint Backlog** (Pila de Sprint). Lista de trabajos que debe realizar el equipo durante el sprint para generar el incremento del producto.

- **Burndown Chart**. Una gráfica que representa el trabajo pendiente del equipo. Existen dos tipos de graficas principales: la relacionada con el Sprint y la relacionada con la totalidad del proyecto.

Para conseguir los objetivos con la metodología SCRUM se realizan una serie de reuniones que detallamos a continuación:

- **Planificación de Sprint** (Sprint Planning). En esta reunión se determina para cada sprint que se va a entregar y como se va a realizar el trabajo propuesto.
- **Reunión Diaria** (Daily Meeting). Reunión con duración muy breve, de unos 15 minutos que se realiza a diario con el fin que el equipo de desarrollo consiga entender que se hizo, que se va a hacer y obtener listado de bloqueos o problemas a resolver.
- **Revisión de Sprint** (Sprint Review). Reunión que se lleva a cabo al finalizar un Sprint, con el objetivo es validar el resultado de la iteración y feedback del producto o cuestiones a mejorar o resolver del incremento.
- **Retrospectiva del Sprint** (Sprint Retrospectiva). Revisar como fue el Sprint en lo que respecta a las persona, relaciones, procesos y herramientas. Identificar y ordenar los temas que salieron bien y cuales poder mejorar. Además de crear un plan para mejorar la forma de trabajar el equipo SCRUM.
- **Refinamiento del Backlog**. No es una reunión obligatoria, pero participan todos los miembros y se realiza una revisión del Backlog del producto, revisión de historias de usuarios que probablemente formen parte de iteraciones próximas y detectar riegos vinculados con esas historias.

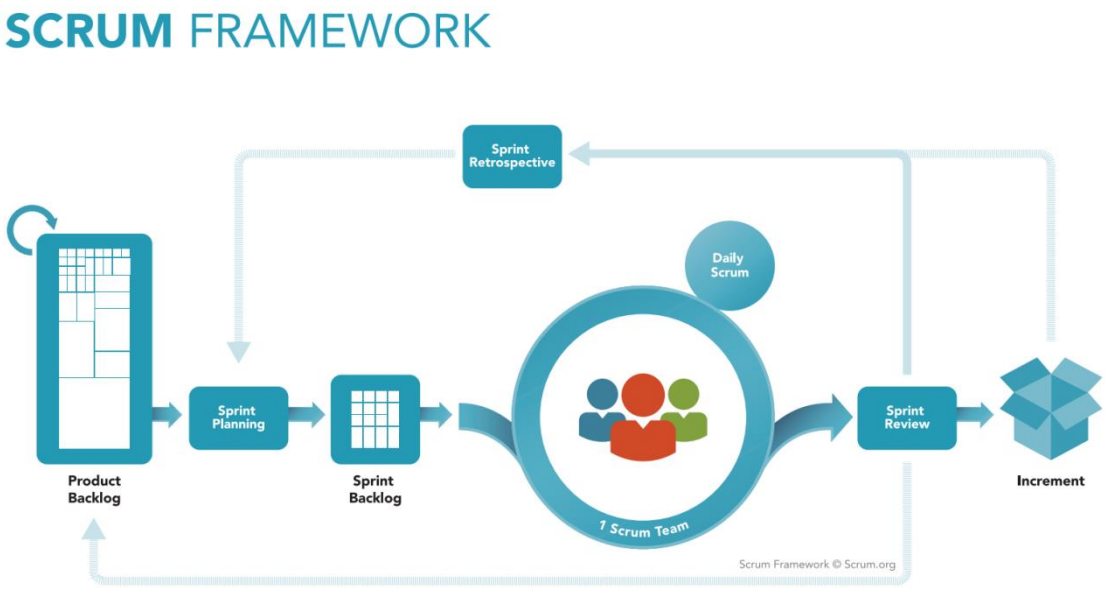

Ilustración 131 - Marco técnico de Scrum. Imagen recuperada de https://www.scrum.org/resources/scrum-framework-poster

Para la elaboración de este apartado la documentación ha sido obtenida de las siguientes referencias (scrum.org, 2019), (Carmen Lasa Gómez, 2018) y (López Gil, 2018).

12.3.3. Kanban

La metodología Kanban está basada en las metodologías agiles y en los procesos de manufactura Just-In-Time (JIT), en las que se pretende realizar proyectos que tiene una naturaleza cambiante de los requisitos, permitiendo adaptarse a estas. En lugar de diseñar un gran plan para llevar a cabo y esperar que no se produzca algo inesperado como pasa en las metodologías tradicionales. En la metodología Kanban los elementos de trabajo intangibles se visualizan para presentar a todos los miembros del proyecto una vista del progreso de los elementos individuales y el progreso desde la definición de la tarea hasta la entrega al cliente.

Los miembros del equipo van consumiendo los trabajos una vez estos tienen la capacidad de poder desarrollarlo, en lugar de que el trabajo sea empujado al proceso cuando se solicita.

La metodología Kanban proporciona un sistema visual de gestión de procesos que ayuda a la toma de decisiones sobre qué, cuanto y cuando producir, permite ver de una manera visual cuantas cosas faltan por hacer.

Está basado en los principios de Lean que se resumen en los siete principios de detallamos a continuación:

- **Eliminar desperdicios**. Eliminar lo que no aporte un beneficio al cliente y no sea valioso para él.
- **Aprender constantemente.** Ir aprendiendo y entendiendo lo que se va necesitando a medida que se desarrolla el producto.
- **Reaccionar rápido.** Implementar rápidamente y con calidad las soluciones que el cliente necesite a medida que vaya detectando nuevas necesidades.
- **Mejora continua**. Mejora en las personas y en los procesos que se utilizan en la construcción del producto.

- **Calidad integrada**. Realizar todo tipo de pruebas para verificar que los errores se corrijan lo antes posible, realización de pruebas de integración, pruebas del producto completo, etc..
- **Optimizar el todo**. Pensar desde un punto de vista global y orientado a largo plazo.
- **Cuidar el equipo de trabajo**. Proporcionar cierto grado de autonomía a la hora de tomar decisiones con sentido, ofreciendo a cada persona la posibilidad de aprender y mejorar de manera permanente, haciendo que su trabajo sea valioso en cada momento.

Hay seis prácticas centrales que deben estar presentes para implementar Kanban que describimos a continuación:

- **Visualizar el fujo de trabajo**. Lo que se pretender es dividir el trabajo en partes, escribiendo cada elemento en una tarjeta que se colocara en un tablero con columnas. Las columnas son identificadas con nombre para conocer en cada momento donde se encuentra localizado cada trabajo, es decir, si se tiene las columnas “Por hacer”, “En progreso” y “Terminado”, como se puede ver en la siguiente ilustración, se puede seguir el progreso y detectar los cuellos de botella.

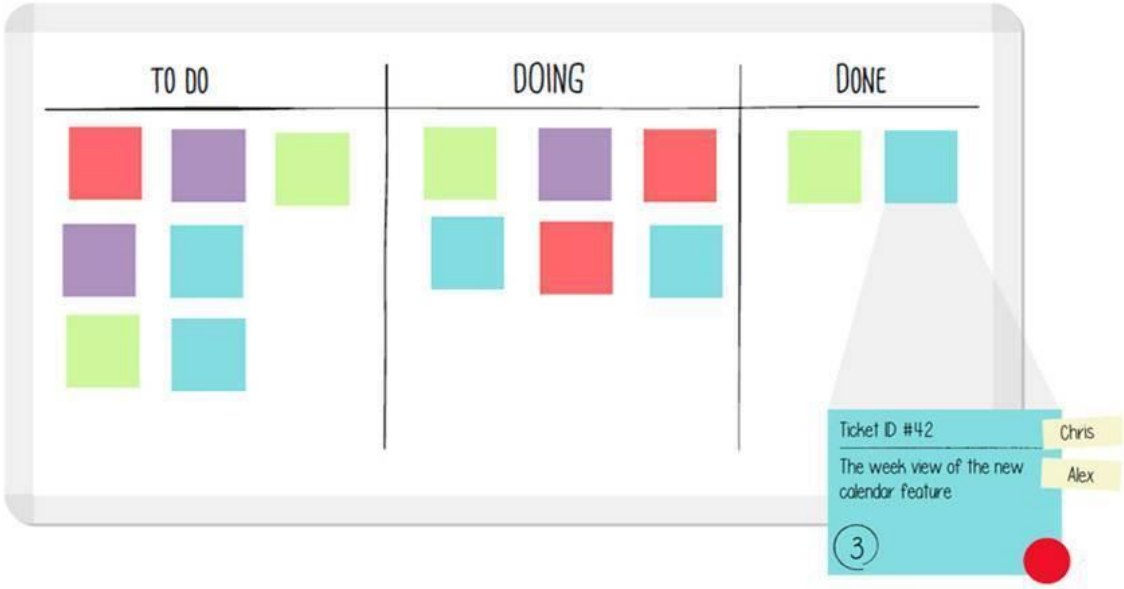

Ilustración 132 - Tablero Kanban. Imagen obtenida de https://abantian.es

- **Eliminar las interrupciones**. Kanban se centra en establecer límites del trabajo en proceso (los limites Work In Progress o WIP), porque si no hay límites de trabajo en proceso, no se está aplicando está metodología correctamente. Establecer un número máximo de elementos por etapa asegura que una tarjeta se arrastra al siguiente paso sólo cuando haya capacidad disponible. Tales restricciones permiten visualizar las áreas problemáticas en el flujo para que puedan ser identificadas y resueltas.
- **Gestionar el flujo**. Se pretende un flujo rápido e ininterrumpido, que significa que está creando valor, minimizando el riesgo y evitando costes inesperados. Por flujo se refiere al movimiento de elementos de trabajo a través del proceso de producción.
- **Hacer las políticas explicitas**. Todos los miembros tienen que estar en conocimientos del objetivo común, para trabajar y tomar decisiones con respecto a cambios que les mueva en una dirección positiva.
- **Circuitos de retroalimentación**. Se ha de realizar reuniones regulares que permita realizar una trasferencia de conocimiento. Tenemos reuniones de pie para sincronizar el equipo, que deben durar entre 10 y 15 minutos delante del tablero, cada miembro comparte con los demás lo que hizo el día anterior y lo que va a hacer en el día. Luego están las reuniones para la revisión de entrega de servicios, la revisión de operaciones y la revisión de riesgos, que pueden durar sobre una hora, en función del tamaño del equipo. Estas han de ser regulares, a una hora estipulada, directas a los problemas y no innecesariamente largas.
- **Mejorar colaborando**. Los equipos ha de analizar los problemas que ha surgido en el desarrollo de los proyectos para tener una compresión compartida de estos para sugerir acciones para mejorar en futuros proyectos por consenso.

Para la elaboración de este apartado la información ha sido obtenida de las siguientes referencias (Carmen Lasa Gómez, 2018) y (López Gil, 2018).

12.3.4. Relación y costo de varios componentes de electrónica

Unidades	Componentes de electrónica	Precio
2	SparkFun GPS	80,00 €
1	Kit tubos térmico-retráctiles multicolor - 10cm	11,75 €
10	Regulador de tensión 3,3V	17,00 €
10	Regulador fijo LM1117T-3.3V 1A TO-220	28,70 €
10	Transistores BC547 NPN 50V 0,1A 0,5W 300Mhz TO-92	8,80 €
1	Kit de Conectores 2,54mm KF2510	8,90 €
1	Clipadora para Dupont	33,00 €
1	Maleta para guardar componentes	48,66 €
5	Barra de contactos de 40 pines	9,46 €
8	SparkFun GPS	294,44 €
2	Adafruit Ref. 1298	87,24 €
6	Cable Macho-Macho	34,41 €
3	Protoboard	4,81 €
10	GY-291 Acelerómetro	7,74 €
10	VEML6070 UV	20,98 €
10	BMS PCM Protector batería 18650	5,78 €
10	XBEE Shield	13,35 €
3	Extractor Chip	5,54 €
10	GY-BME280-3.3V Humedad, temperatura y presión	22,53 €
1	HS-D1 Crimper cortador de Cable Pelacables	9,25 €
10	Pin Macho-Hembra Header	3,53 €
2	JSP SM 100cm 5 Pares conector Macho-Hembra	12,86 €
2	Pin Header 2,54mm	14,62 €
2	Pin Header 2,54mm	5,75 €
1	Varios conectores	18,08 €
1	Kit para soporte de placas PCB	14,42 €
5	GY-30 BH1750FVI Luminosidad	5,04 €
5	XBEE Shield	6,97 €
5	XBEE Shield	4,40 €
4	VEML6070 UV	9,99 €
1	1000 piezas Resistencias 4,7 Ohm	5,59 €
5	Sensor de humedad suelo	37,01 €
5	ADS1015 ADC	8,39 €
3	CJMCU-750 UART I2C	8,62 €
2	PCF8575 16IO I2C	6,88 €
2	Conectores 40 pines	5,07 €
6	Caja baterías 18650	5,47 €
3	Sensor humedad	2,91 €
6	Kit PCB Prototipo	18,43 €
5	BMS PCM Protector batería 18650 15A	2,64 €
6	BMS PCM Protector batería 18650 5A	4,02 €
1	Sensor temperatura/humedad suelo SHT10	16,26 €

2	Sensor temperatura/humedad suelo SHT20 SHT10	35,78 €
3	Sensor humedad capacitivo	4,26 €
1	TMP36 y 100 Piezas 100omh	9,16 €
1	10000 piezas resistencias 10komh	9,61 €
1	VELM6070 UV	3,50 €
1	Sensor Temperatura/humedad suelo SHT10	19,34 €
5	BME280 Temperatura/Humedad/Presión 1.8-5V	12,40 €
1	Sensor Humedad Resistencia	8,81 €
1	BH1750 Luminosidad	4,86 €
1	Sensor temperatura/humedad suelo SHT20	15,64 €
	Total…	1.092,65 €

12.3.5. MultiTech mDot Pinout Diagram

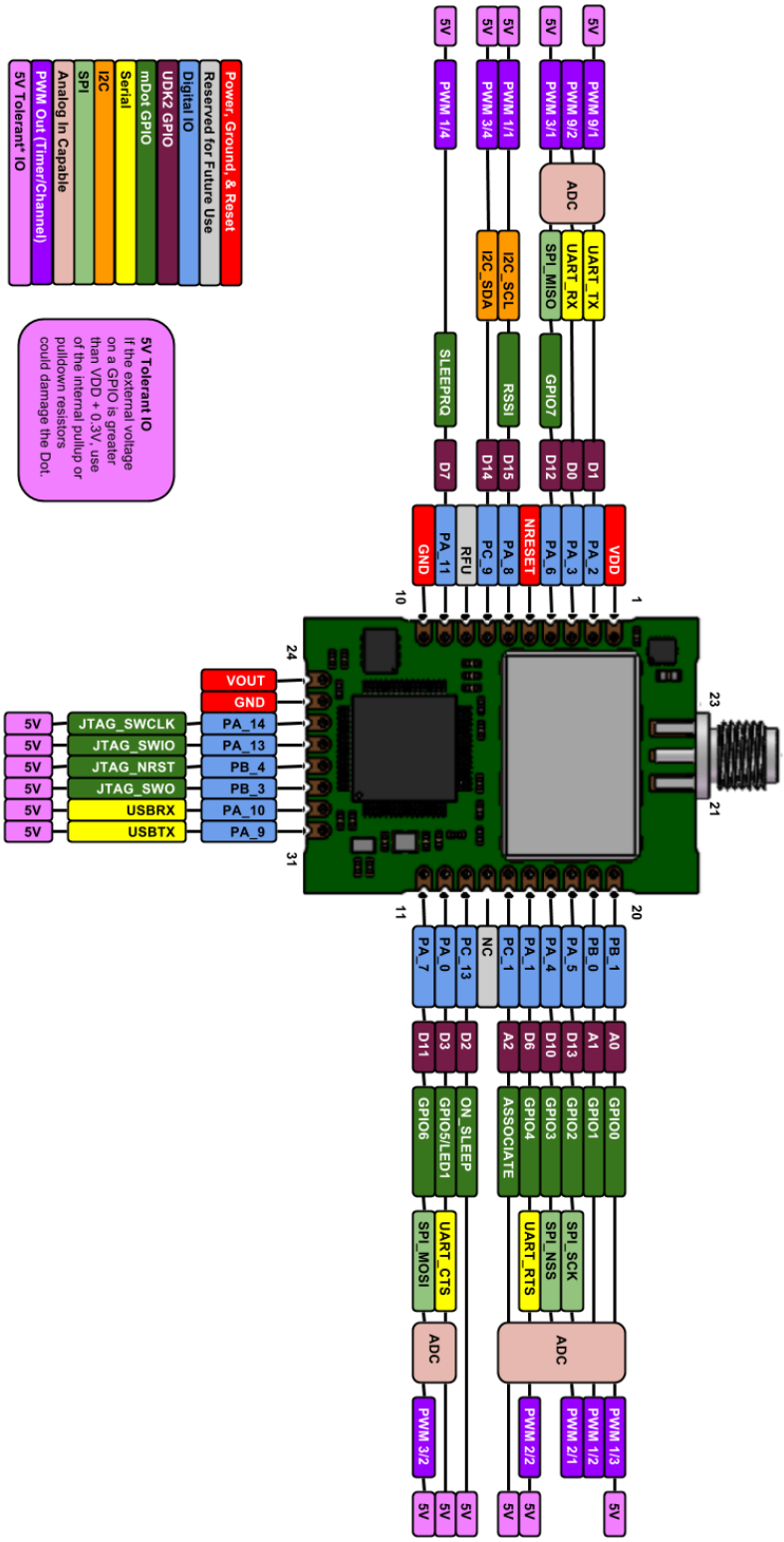

Ilustración 133 - mDot Diagrama de Pin, imagen optenida de https://os.mbed.com/platforms/MTS-mDot-F411/