Grado en Ingeniería Informática

Trabajo de Final de Grado

Persistencia de datos en dispositivos Internet Of Things (IOT)

Autor:
Enrique Cueto Rubio

Supervisor:
Juan Camilo Gómez Esguerra
Tutor académico:
Pedro García Sevilla

Fecha de lectura: 14 de julio de 2022
Curso académico 2021/2022

Resumen

El presente documento contiene la memoria técnica sobre la implementación de una nueva funcionalidad para el proyecto SUCRE (Sense yoUr Context and REact), más concretamente para su aplicación web, la cual permite crear proyectos aplicando la programación con bloques. Esta funcionalidad permite al usuario almacenar los datos obtenidos mediante un microcontrolador Argon [1], creado por la empresa Particle, y visualizarlos de manera gráfica y de forma detallada.

A lo largo del documento se explicarán los problemas que han surgido a lo largo del desarrollo de dicha funcionalidad, a la vez que las diferentes decisiones realizadas con la ayuda de los miembros de la empresa.

En los primeros apartados se incluirá una introducción en la cual se mencionará el contexto del proyecto y sus objetivos. A continuación se realizará una explicación acerca de la planificación llevada a cabo. Seguidamente se aportará el análisis y diseño del sistema y finalmente se mostrarán la implementación y las pruebas realizadas.

Esta funcionalidad ha sido desarrollada durante la estancia en prácticas en la empresa UBIK Geospatial Solutions.

Palabras clave

IOT, Angular, Blockly, MQTT, InfluxDB

Keywords

IOT, Angular, Blockly, MQTT, InfluxDB

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este apartado de la memoria, se describirá el contexto y la motivación del proyecto y se aportará una descripción del mismo. Además se mencionarán los objetivos a cumplir y la estructura de este mismo documento.

1.1. Contexto

La empresa en la que se ha realizado la estancia en prácticas es UBIK Geospatial Solutions, cuyo logotipo se muestra en la Figura 1.1. El objetivo principal de esta organización es la transferencia de conocimientos de desarrollo e investigación a la implementación de soluciones geoespaciales para así ayudar a los usuarios finales y/o organizaciones a tomar decisiones más inteligentes.

Figura 1.1: Logo de UBIK Geospatial Solutions

Esta empresa fue constituida por Joaquín Huerta, Ana Sanchís y Michael Gould, investigadores del grupo de investigación GEOTEC (Geospatial Technologies Research Group), y actualmente se ubica en la Universitat Jaume I. Ofrece servicios de programación para implementar aplicaciones web y móviles para la integración de información geoespacial. Entre estos servicios se incluyen:

- Desarrollo de aplicaciones móviles basadas en la localización.
- Infraestructuras de Datos Espaciales (IDE).

- Desarrollo de aplicaciones de cartografía Web.
- Integración de información de las redes sociales con otros sistemas de la información.

1.2. Descripción del proyecto

1.2.1. Estado inicial del proyecto

En este caso, no se ha creado un proyecto desde cero, si no que se ha trabajado sobre un proyecto que la empresa UBIK Geospatial Solutions ya creó con anterioridad, y al cual se le quiere añadir una nueva funcionalidad. Actualmente dentro de la empresa se están desarrollando distintos proyectos, pero en este caso en concreto el sistema con el que se trabajará será el conocido como SUCRE (Sense yoUr Context and REact).

El objetivo del proyecto SUCRE es el fomento de las vocaciones científicas, la promoción del pensamiento computacional y la programación, y se ha dividido en dos versiones: Sucre4Kids y Sucre4STEM. La primera de ellas, se centra en la etapa de la educación primaria, mientras que la segunda cubre la etapa de educación secundaria (ESO, Bachillerato y FP grado medio), y es la versión con la que se trabajará durante la estancia en prácticas.

Los recursos didácticos que ofrece Sucre4STEM se agrupan en un maletín conocido como SucreKit, el cual está compuesto por componentes electrónicos (sensores y actuadores) que se conectan a un microcontrolador llamado Argon, creado por Particle Industries, Inc. que puede ser programado mediante un sistema de programación por bloques y al cual nos referiremos como 'SucreCore'. Este dispositivo al cual se conectan los sensores y actuadores, será dotado de conexión a Internet, lo que hace que se considere un dispositivo del Internet de las Cosas (IOT). En la Figura 1.2 se puede observar un ejemplo de un proyecto realizado mediante la programación por bloques.

Figura 1.2: Ejemplo del sistema de programación por bloques

Para conocer mejor la situación inicial del proyecto, cabe mencionar que la aplicación creada para el proyecto SUCRE utiliza el framework Angular, el cual se explica con más detalle en el apartado 1.4.1. Tecnologías. Las funcionalidades que existían anteriormente a la incorporación del alumno al desarrollo del proyecto, son las siguientes:

- **Inicio de sesión:** el usuario puede registrarse o iniciar sesión en la aplicación a través de su dirección de correo electrónico y su contraseña. Cada usuario tiene un identificador único.

- **Modificar información personal:** el usuario puede cambiar datos personales como por ejemplo su nombre.

- **Gestionar proyectos:** en la aplicación SUCRE, se conoce como proyectos a los programas diseñados por los usuarios a través del uso de la programación por bloques. El usuario es capaz de crear nuevos proyectos, modificar proyectos existentes o eliminar aquellos que desee.

- **Enviar un proyecto al SucreCore:** a través de la aplicación se envían los programas creados al SucreCore, el cual los ejecuta.

- **Seleccionar SucreCore activo:** el usuario puede ser propietario de más de un SucreCore. Es por esto que se dispone de una herramienta que permite seleccionar el SucreCore al cual se le enviarán los programas creados. Resulta imprescindible para casos en los que se está trabajando con dos o más microcontroladores de forma simultánea.

- **Cambiar la red WiFi de un SucreCore:** para poder enviar proyectos al SucreCore, es necesario que este esté conectado a una red WiFi. Es por eso que se proporciona una herramienta que permite conectar el SucreCore a una red WiFi.

Finalmente, cabe comentar que en la aplicación Angular, para que el desarrollo de la misma resulte más cómodo, se han ordenado los ficheros en paquetes. Estos paquetes ordenan los ficheros según su funcionalidad. En concreto, el patrón que se ha seguido para este proyecto es el Modelo-Vista-Controlador (MVC), que se explica mejor en el apartado 4.2. Patrones y estrategias. La organización de los paquetes de la aplicación se observa en la Figura 1.3.

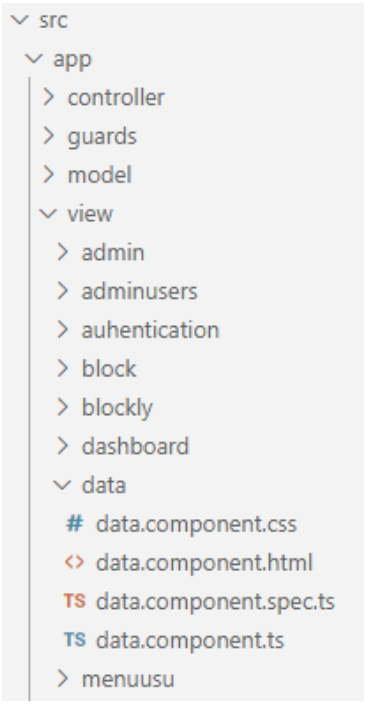

Figura 1.3: Organización de los paquetes del proyecto

1.3. Motivación del proyecto

En el estado actual en el que se encuentra el proyecto SUCRE, el sistema permite a los usuarios crear proyectos de forma sencilla a través de la programación por bloques. Debido a que los usuarios que van a usar este producto son, como ya se ha mencionado anteriormente, estudiantes de educación secundaria, el uso del proyecto SUCRE podría ser una opción realmente interesante, ya que de esta forma se podrían realizar experimentos y estudios a la misma vez que se aprende a programar de manera sencilla con la programación por bloques.

No obstante, no hay ninguna forma de almacenar los valores generados mediante programas creados por los usuarios de ninguna forma. Es por esto que la principal motivación del proyecto es solucionar el problema de la pérdida de los datos obtenidos que tiene ahora mismo el sistema, implementando una nueva funcionalidad a través de la cual se puedan almacenar dichos datos.

El hecho de ofrecer al usuario la posibilidad de almacenar datos y luego poder analizarlos, aportaría un beneficio clave para el proyecto SUCRE, puesto que a la hora de realizar experimentos y/o pruebas, es interesante realizar un análisis de los resultados obtenidos. Además, tener esta funcionalidad implementada en el sistema daría la oportunidad de proponer actividades diferentes a las tradicionales a los profesores de otras asignaturas a parte de la informática, como por ejemplo la física o la biología.

1.4. Objetivos del proyecto

Durante la estancia en prácticas, se quiere implementar una nueva funcionalidad al proyecto SUCRE que proporciona una mayor motivación a los usuarios para utilizar este producto. Esto es, como bien indica el título del documento, la persistencia de los datos. Es por esto que el principal objetivo de este proyecto es proporcionar a los usuarios que usen el sistema un método para almacenar los valores captados mediante los sensores conectados al microcontrolador y la posibilidad de consultar los mismos.

El objetivo principal se puede desglosar en los siguientes objetivos:

- Construir un bloque que permita al usuario incluir en sus programas la opción de guardar el valor deseado.
- Realizar la conexión entre el microcontrolador y la base de datos para realizar la operación de guardado.
- Implementar un menú simple a través del cual el usuario pueda elegir qué conjunto de valores desea ver.
- Mostrar al usuario los datos solicitados de forma sencilla y fácil de entender.

Por otro lado, la definición del alcance del proyecto se presenta a continuación dividida en tres partes, funcional, organizativo e informático.

Alcance funcional: son aquellas funcionalidades que se implementarán y serán usadas por los usuarios finales.

- Guardado de los valores recogidos por los sensores de forma automática.
- Consulta de dichos datos por parte de los usuarios.

Alcance organizativo: organizaciones cuyo funcionamiento se verá afectado por la aplicación.

- En los centros de educación secundaria, la persistencia de los datos permitiría a los alumnos revisar los valores obtenidos en los experimentos realizados y realizar estudios estadísticos sobre ellos, y otorgaría a los profesores un método de evaluación del trabajo realizado en los experimentos, permitiéndoles comprobar si los valores recogidos por los alumnos son coherentes o no.

Alcance informático: aquellas tecnologías con las que interactúa el sistema. En este apartado se mencionarán las tecnologías usadas y se aportará una breve descripción. En el apartado 1.4.1. Tecnologías se ofrece información de forma detallada acerca de las tecnologías usadas.

- Se utiliza Firebase, como herramienta para almacenar los datos acerca de las cuentas de usuario, los dispositivos vinculados a cada uno de estos, etc. y para facilitar la creación de la aplicación.

- Particle Cloud proporcionado por Particle, el cual es un servicio que usan los dispositivos para poder conectarse a Internet entre otros.
- El framework utilizado para desarrollar la aplicación es Angular, de Google.
- La librería Blockly para crear los bloques que permiten a los usuarios realizar programas de forma sencilla para sus SucreCores.
- Se utiliza una base de datos InfluxDB a parte para almacenar los datos recogidos por los usuarios. Además es de esta de la cual se recogerán los datos para mostrarlos a estos.
- La librería Ngx-Charts, la cual permite al desarrollador construir gráficas de forma sencilla.
- Node-RED, una aplicación que permite crear APIs entre otros, de forma visual y simple.

1.5. Tecnologías usadas en el proyecto

En este apartado se comentarán las tecnologías utilizadas en el proyecto SUCRE, tanto para el frontend como para el backend, aunque se describirán con más detalles en el apartado 3. Análisis y diseño del sistema. Además, se dividirán las tecnologías según si se usaban antes de la estancia en prácticas del alumno o si se han usado durante la misma.

1.5.1. Tecnologías anteriores a la estancia

En este apartado se mencionan las tecnologías que la empresa utilizaba antes de la incorporación del alumno. Estas son las siguientes:

- Para el frontend se utiliza Angular [2], un framework de código abierto mantenido por Google que permite crear y mantener aplicaciones web. Los lenguajes de programación usados en el mismo son sobre todo JavaScript, HTML y TypeScript, aunque el que más predomina es JavaScript.
- Para el backend se utiliza por un lado Firebase [3], un servicio creado por Google que proporciona una gran variedad de servicios para los desarrolladores de software. En el proyecto SUCRE se utilizan concretamente Firebase Auth [4], que permite implementar servicios de autenticación para el servicio de inicio de sesión, Firebase Cloud Firestore [5], para almacenar los datos generados por los usuarios (proyectos creados, SucreCores en propiedad, etc.) y finalmente Firebase Hosting [6] como servicio de hosting para la aplicación.
- Para la creación de los bloques que permiten a los usuarios programar de forma sencilla, se utiliza Blockly [7], una librería creada por Google que permite a los desarrolladores implementar estos bloques en sus aplicaciones.
- Finalmente cabe comentar que los SucreCores utilizan un servicio desarrollado por la empresa Particle Industries, Inc, el cual permite a los dispositivos conectarse a Internet y a través del cual los usuarios podrán enviar sus programas a los dispositivos. A este servicio se le conoce como Particle Cloud [8].

1.5.2. Tecnologías añadidas durante la estancia

En este apartado se mencionan las tecnologías que se han añadido al proyecto durante la estancia en prácticas del alumno. Estas son las siguientes:

- A causa de la funcionalidad implementada en la estancia en prácticas, se ha creado una base de datos InfluxDB [9], en la cual se guardarán los valores que los usuarios recojan con sus SucreCores. Este servicio se ha implementado a través de Amazon Web Services, y permite realizar operaciones de guardado de datos en la nube. Esta decisión la tomó el gestor de proyectos, y resultó ser muy cómoda por la compatibilidad que tiene con Node-RED, una tecnología que se comenta posteriormente.
- Aunque no se especificó en planificación inicial del proyecto, se han incluido gráficas en la aplicación, en las cuales se muestran los datos de forma visual. Para esto se ha utilizado la librería Ngx-charts [10], la cual permite construir estas gráficas en aplicaciones Angular de forma sencilla. El uso de esta librería fue una sugerencia del supervisor, y resultó ser una muy buena opción, ya que el proceso de implementación de las gráficas fue bastante factible y los resultados son excelentes.
- Finalmente se utilizó Node-RED [11] para diseñar una API que sirviera para que se pudieran recibir datos procedientes de la base de datos InfluxDB desde la aplicación Angular. Como se ha mencionado, el uso de este recurso resultó ser todo un éxito porque aportó mucha sencillez al proceso de lectura de la base de datos InfluxDB. Es por esto que esta decisión del gestor de proyectos también me pareció acertada.

Capítulo 2

Planificación del proyecto

En este capítulo se presenta la planificación llevada a cabo para el desarrollo del proyecto. Primero se describirá la metodología que se ha usado, seguidamente se explicará la planificación de las tareas del proyecto, se realizarán las estimaciones de los recursos y se calcularán los costes del proyecto. Finalmente se describirá el control del proyecto que se ha tenido a lo largo de su desarrollo.

2.1. Metodología

En cuanto a la metodología seguida para llevar a cabo el proyecto, la empresa propuso trabajar con metodologías ágiles. Como se ha observado en múltiples casos, el hecho de trabajar con metodologías tradicionales puede acarrear consecuencias como, por ejemplo, entregas no satisfactorias para el cliente, debido a la imposición de fechas límite entre otros. Es por esto que hoy en día cada vez más, las empresas, sobre todo las relacionadas con el mundo de la informática, se están empezando a decantar por el uso de las metodologías ágiles.

La principal diferencia de las metodologías ágiles frente a las tradicionales es que la calidad del producto desarrollado siempre se considera la mayor de las prioridades del proyecto. Para conseguir esto se pide una mayor implicación del cliente y del resto del equipo en el proyecto que en las metodologías tradicionales, de modo que se pueda obtener feedback de forma continuada y en caso que haya alguna discordancia, solucionarla lo antes posible.

La metodología ágil con la que se ha trabajado concretamente es SCRUM, la cual se explica con más detalle en el siguiente apartado.

2.1.1. SCRUM

La planificación del proyecto se realizará, como se ha mencionado en el apartado anterior, mediante una metodología ágil, más concretamente Scrum [12]. Este proceso se desarrolla de la siguiente forma:

Primero se identifican las funcionalidades a implementar y se dividen en tareas, a las cuales se les adjudica una cierta prioridad. Este conjunto de tareas se conoce como pila del producto.

A continuación, se dividen dichas tareas en sprints. Un sprint es un periodo de tiempo (normalmente de un mes o de 15 días) durante el cual se realizan las tareas adjudicadas. Durante la ejecución de los sprints se realizan reuniones diarias para comprobar el progreso que han realizado los programadores en las sesiones de trabajo. En el proyecto SUCRE, y por lo tanto durante la estancia en prácticas, los sprints tendrán una duración de exactamente dos semanas, y todos los martes y jueves se realizarán reuniones en las que tanto el supervisor como el gestor de proyectos se reunirán con el alumno y se revisará el progreso de las tareas asignadas. Si dichas tareas se han completado se asignan otras, y en caso de que algunas no se hayan terminado, se seguirá trabajando en ellas durante las siguientes semanas. Una tarea se considera completada cuando tanto el supervisor como el gestor de proyectos han dado su visto bueno.

Al finalizar el sprint, se realiza una reunión en la cual se comprueba el progreso del mismo, se definen las tareas a realizar a continuación, entre las cuales se pueden encontrar algunas que hayan quedado pendientes, y se da comienzo al siguiente sprint.

A diferencia de las metodologías tradicionales, con la metodología Scrum las diferentes fases de desarrollo pueden llegar a solaparse en caso de ser necesario, ya que la prioridad es asegurar la calidad del trabajo realizado. En la Figura 2.1 se observa de forma visual el marco de trabajo característico de la metodología SCRUM.

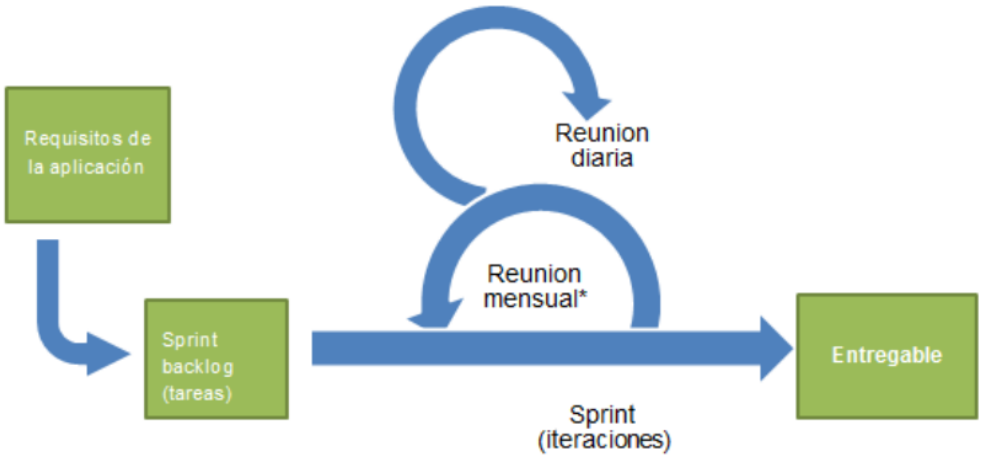

Figura 2.1: Marco de trabajo SCRUM [13]

2.2. Planificación inicial

En este apartado se describirá el proceso de planificación que se ha realizado para el proyecto descrito en este documento. Una de las grandes diferencias entre SCRUM y las metodologías predictivas es que en la metodología SCRUM, a diferencia de las predictivas, no se sabe con exactitud cuando termina una tarea y cuando empieza la siguiente, es por esto que en la planificación inicial, la cual se describe en este apartado, únicamente se puede aportar una primera versión de la pila del producto (*backlog*), la cual contiene información acerca de todas las tareas que se deben realizar para completar el proyecto, y el primero de todos los sprints, ya que hasta que este no termine, no sabremos qué tareas se realizarán en el siguiente.

Cabe mencionar que el hecho de que el *backlog* cambie a medida que avanza la fase de implementación es muy común, ya que se pueden dar incidencias, problemas de implementación y/o cambios de opinión. Este caso se ha dado durante la estancia en prácticas, y es por eso que en este apartado se ha proporcionado la primera versión de la pila del producto, en la cual hay funcionalidades que estaba planeado que fueran implementadas pero finalmente no ha sido así, a la vez que hay funcionalidades que no aparecen en el *backlog* y que finalmente sí se han implementado. En el apartado '4.1 Detalles de implementación' se escribe con detalle todo el proceso de toma de decisiones, las cuales provocan estos cambios en la pila del producto, y los resultados de los sprints. En la Figura 2.2 se muestra una primera versión de la pila del producto.

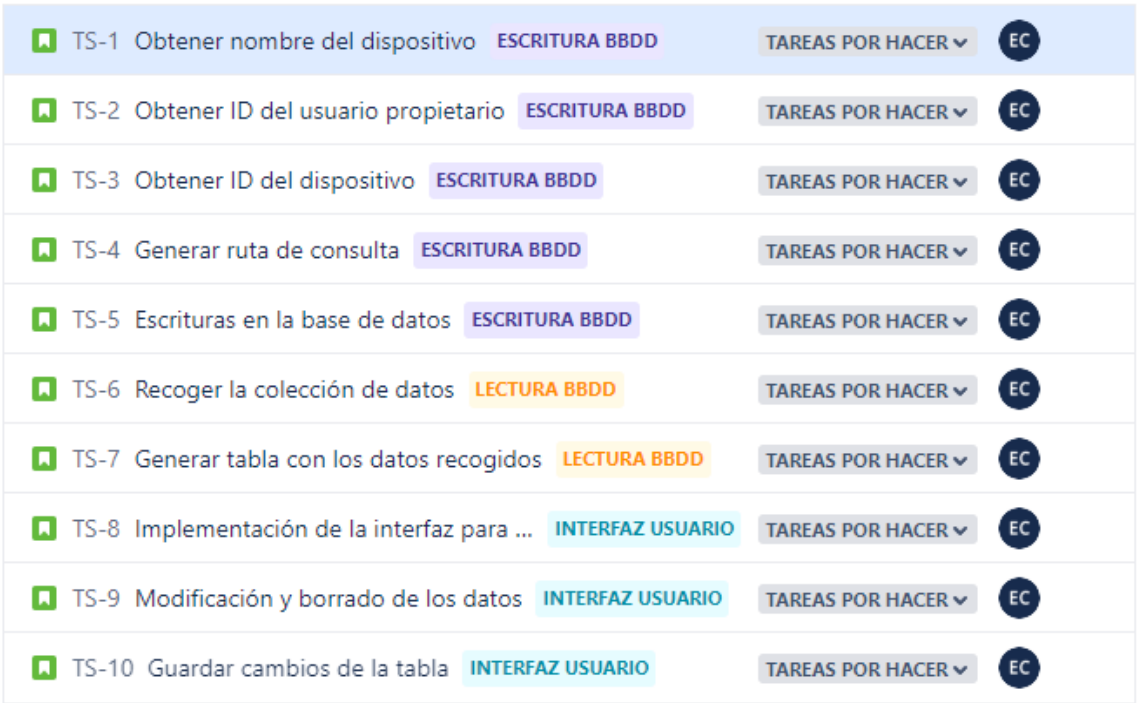

Figura 2.2: Backlog inicial

Como podemos observar en la Figura 2.2, se han agrupado las tareas en épicas. Las épicas son los grupos en los que se dividen las tareas para agruparlas según su propósito. En este caso solamente existen tres épicas:

- **Escritura BBDD**: tareas relacionadas con el proceso de escritura en la base de datos, desde las obtenciones de los datos pertinentes hasta la propia operación de escritura en sí.
- **Lectura BBDD**: tareas a través de las cuales se obtendrán los datos a mostrar y se crearán los elementos que los mostrarán.
- **Interfaz usuario**: aquellas tareas en las que se trabaja con los elementos con los que interactuará el usuario.

En la Tabla 2.1 se ofrece una versión más detallada de la pila del producto, en la cual se proporciona una prioridad y una descripción para cada tarea. Las prioridades se han asignado en base a la dificultad que se ha percibido que tenían las tareas en un primer momento (es decir, una estimación) para ser implementadas, la relevancia que tienen para el usuario final y el tiempo que pueden conllevar.

ID	Nombre	Prioridad	Descripción
T1	Obtener nombre del dispositivo	4	Obtener el nombre del microcontrolador para saber en qué lugar de la base de datos se encuentra el identificador del usuario propietario
T2	Obtener ID del usuario propietario	8	Comunicarse con la base de datos para obtener el identificador del usuario propietario del dispositivo y saber dónde escribir los valores que recogen los sensores
T3	Obtener ID del dispositivo	5	A parte del nombre, se debe de conocer el identificador único del dispositivo para escribir en la base de datos
T4	Generar ruta de consulta	3	Crear la ruta donde se deben escribir los datos recogidos a partir de los identificadores obtenidos anteriormente
T5	Escrituras en la base de datos	9	Realizar las escrituras de los valores recogidos por los sensores en la base de datos
T6	Recoger el conjunto de datos	8	Recoger los valores obtenidos y escritos en la base de datos desde la aplicación Angular
T7	Generar tabla con los datos recogidos	6	Crear una tabla con los valores recogidos de la base de datos con el formato adecuado para la lectura y modificación del usuario
T8	Implementación de la interfaz para consultar los datos	8	Implementar una interfaz con la que el usuario pueda interactuar con los datos recogidos
T9	Modificación y borrado de los datos	8	Proveer al usuario la posibilidad de modificar y borrar los datos de dicha tabla de valores recogidos por los sensores
T10	Guardar cambios de la tabla	9	Guardar los cambios realizados en la tabla en la base de datos, es decir si se eliminan campos en la tabla hacerlo también en la base de datos

Tabla 2.1: Backlog detallado

Como se ha mencionado anteriormente, los sprints tienen una duración de dos semanas, y en este caso se ha decidido tomar un primer contacto con la base de datos, intentando progra-

mar un SucreCore para que establezca una conexión con la base de datos Firebase y escriba manualmente un valor, sin tener en cuenta sensores ni programas hechos por un usuario real. Para ello, las tareas necesarias a implementar son las que aparecen en la Figura 2.3:

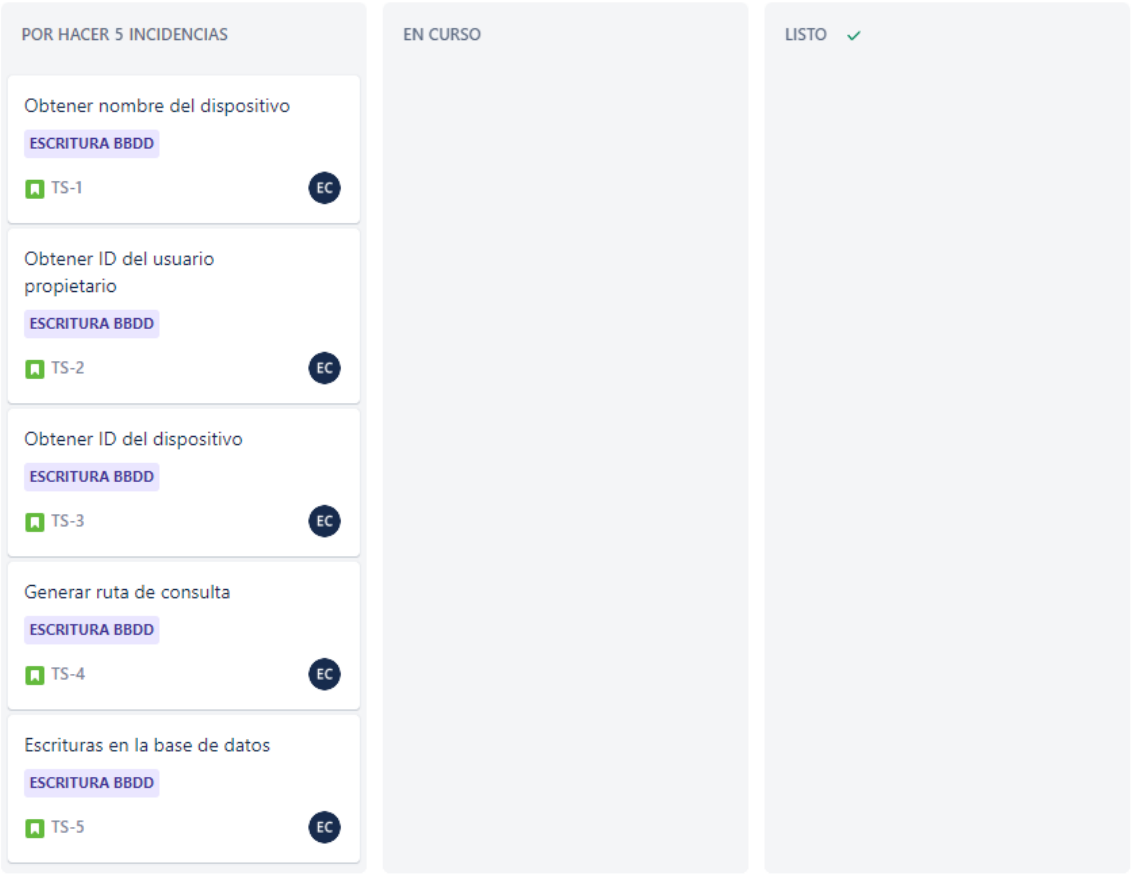

Figura 2.3: Tareas iniciales del primer sprint

En la Figura 2.3 aparecen tres apartados, 'Por hacer', 'En curso' y 'Listo'. Al principio del sprint, todas las tareas se muestran en el primero de estos apartados, pero a medida que se realizan las implementaciones, el programador debe mover las tareas de un lado a otro según el estado en el que se encuentren.

2.3. Estimación de recursos y costes del proyecto

En este apartado se mostrará una estimación inicial de los recursos que se van a utilizar, los cuales se pueden dividir en tres: humanos, software y hardware, y los costes que estos suponen.

- **Recursos humanos**
  La empresa UBIK Geospatial Solutions ha preferido mantener la remuneración de los

trabajadores en privado por lo tanto no podemos calcular con exactitud los costes de personal. De todas formas, se ha realizado una estimación de los salarios en base a la "Guía del mercado laboral de HAYS 2020" [14], usando los salarios pertenecientes a la localidad más cercana a la Universidad Jaume I, Valencia. Como las prácticas tienen una duración de tres meses, se especificará únicamente el salario total de los tres meses.

- Enrique Cueto Rubio, Alumno en prácticas y programador Front-end (se ha considerado que el alumno es un programador júnior), cuyo salario se estima que es de 29.000 euros. No obstante el alumno solamente trabajaba 25 horas a la semana, en lugar de las 40 horas semanales que se estiman en la "Guía del mercado laboral de HAYS 2020". Es por eso, que el salario sufre cambios, más concretamente en lugar de ser de 2.416,67 euros mensuales, y 7.250 euros por el desarrollo del proyecto, pasa a ser de 1.510,41 euros mensuales y 4.531,25 euros por el desarrollo del proyecto.
- Juan Camilo Gómez Esguerra, Supervisor y Consultor GIS en UBIK Geospatial Solutions, que trabaja 10 horas a la semana en este proyecto. Es por esto que si el salario anual estimado en su posición son 50.000 euros suponiendo que se trabajan 40 horas semanales, al mes son 4.166,67 euros mensuales, lo que suponen 12.500 euros para el desarrollo del proyecto. No obstante, como solamente se trabajan 10 horas semanales, el salario total pasa a ser de 3.125 euros.
- Gestor de proyectos de la empresa UBIK. En este caso pasa lo mismo que en el del supervisor, el gestor de proyectos le ha dedicado 10 horas semanales en lugar de las 40 horas semanales que se estiman en la 'Guía del mercado laboral de HAYS 2020' a la hora de calcular el salario anual. Si el gestor de proyectos trabajara 40 horas semanales, el salario sería de 61.000 euros anuales, lo que se traduce en 5.083,33 euros mensuales, y por lo tanto el salario total para este proyecto sería de 15.250 euros. No obstante, si tenemos en cuenta que solamente se trabajan 10 horas a la semana, el salario total pasa a ser de 3.812,50 euros para el desarrollo del proyecto.

- **Recursos hardware**

  - Equipo necesarios para la programación, en este caso se utilizará un ordenador portátil que el alumno tiene en propiedad. Concretamente, el Acer Aspire A515-51G, cuyo precio se sitúa sobre los 500 euros. La empresa me ofreció la posibilidad de trabajar con uno de los ordenadores que tenían en propiedad, pero finalmente decidí usar el mío propio por motivos de comodidad. Debido a que se usó durante tres meses, y el ordenador tiene una esperanza de vida de aproximadamente 5 años, se considera que este ordenador supone un coste de 25 euros.
  - Un SucreKit compuesto por un microcontrolador, sensores y actuadores propiedad de la empresa. Cada SucreKit contiene los siguientes elementos:
    - Seeed Studio Grove Chainable RGB Led V2.0 con un coste de 5,07 euros.
    - Grove - Ultrasonic Distance Sensor con un coste de 3,30 euros.
    - Grove - 4-Digit Display con un coste de 5,00 euros.
    - Grove - Temperature Humidity Sensor (DHT11) con un coste de 5,00 euros.
    - Grove - Rotary Angle Sensor con un coste de 2,46 euros.
    - Grove - Button con un coste de 1,61 euros.
    - Grove - Light Sensor v1.2 con un coste de 2,46 euros.
    - Grove - 4-Digit Display con un coste de 5,00 euros.
    - Grove - 4-Digit Display con un coste de 5,00 euros.

    - Grove - White LED con un coste de 1,61 euros.
    - Grove - Sound Sensor con coste de 4,15 euros.
    - Grove - Capacitive Moisture Sensor (Corrosion Resistant) con coste de 5,04 euros.
    - NEODYMIUM DISC MAGNET 3X1MM (PK50) con coste de 4,50 euros.
    - Particle Argon con coste de 21,99 euros.
    - Caja organizadora con un coste de 3,69 euros.

- **Recursos software**

    - Herramienta Jira para la gestión de las tareas del proyecto.
    - Visual Studio Code para tareas de programación tanto por la parte del microcontrolador como por la parte de la aplicación Angular.
    - Particle Build, un entorno de desarrollo web creado por Particle, el cual permite crear programas para microcontroladores creados por esta empresa, entre ellos el Argon que se utiliza en el SucreKit.
    - Node-RED, una herramienta de desarrollo basada en flujo para programación visual desarrollada originalmente por IBM para conectar dispositivos de hardware, APIs y servicios en línea como parte de Internet of Things.
    - Firebase como herramienta de hosting, autenticación y almacenamiento de datos. En este proyecto se ha utilizado la versión más básica de estos servicios, los cuales permiten a los desarrolladores utilizarlos sin tener que pagar un coste adicional.
    - Amazon Web Services como plataforma para establecer la base de datos InfluxDB en la que se almacenarán los datos que los usuarios generen con sus SucreCores. Amazon permite a los desarrolladores utilizar sus servicios de forma gratuita el primer año, es por eso que se considera que para el proyecto este elemento no ha supuesto ningún coste adicional. No obstante, superado este año, se cobra una cuota que no es fija, sino que depende del uso que el desarrollador haga de los servicios proporcionados por Amazon. Las tarifas aplicadas se pueden consultar en la página oficial de AWS Marketplace [15].
    - GitHub como herramienta para realizar guardados en la nube de los cambios realizados en la aplicación Angular.

Por un lado hay que calcular los costes directos, entre los cuales se incluyen los recursos humanos, hardware y software. Cabe mencionar que los porcentajes para el cálculo tanto del coste de la Seguridad Social como de los costes indirectos se nos proporcionaron en una asignatura del grado llamada 'Iniciativa empresarial'.

El coste total de los recursos humanos asciende a 11.468,75 euros, aunque a esto hace falta sumarle el coste de la Seguridad Social, el cual se estima que es de un 25 %, suponiendo un extra de 2.867,19 euros. Por lo tanto el total de los recursos humanos asciende a 14.335,94 euros. En cuanto a los recursos hardware, la suma de todos los productos asociados al SucreKit asciende a un total de 70,88 euros. Este kit ha sido adquirido específicamente para el desarrollo del proyecto, es por esto que se tiene en cuenta la totalidad del coste de este. A esto hay que sumarle el precio del ordenador portátil, que son 25 euros, haciendo un total de 95,88 euros. Los recursos software que se han utilizado permiten al usuario usarlos de forma gratuita, por lo que en este aspecto, el coste es de 0 euros. El precio total de los costes directos, por lo tanto, es de 14.431,82 euros.

Por otro lado hay que añadir los costes indirectos, que suponen aproximadamente un 20 % del total de los costes directos y entre los cuales se incluyen los gastos relacionados con la luz, el agua, el alquiler, etc. Dicho esto, el total de los costes indirectos supone un gasto de 2.886,36 euros.

Esto quiere decir que el coste total del proyecto es de 17.318,18 euros.

2.4. Seguimiento del proyecto

En este apartado se explicarán las medidas tomadas para poder realizar un seguimiento del proyecto de forma precisa y completa. En el apartado 4.1. Detalles de implementación se describe cómo se han utilizado estas medidas y se muestra los resultados de su uso.

Como se ha explicado en el apartado 2.1.1. SCRUM, durante el desarrollo del proyecto se realizarán dos tipos de reuniones. Por un lado se realizarán reuniones semanales, en las cuales se comprobará el progreso realizado en las tareas del sprint activo en el momento. Durante estas reuniones se hablará con uno de los miembros del equipo de trabajo, el cual está al cargo del proyecto y se responderán las siguientes cuestiones:

- Cual ha sido el trabajo realizado desde la última reunión.
- En qué se está trabajando actualmente.
- Problemas y/o opiniones que han surgido a lo largo de la semana.
- Qué está planeado hacer para la próxima semana.

Por otro lado, se realizarán además otras reuniones cuando acaben los sprints, los cuales duran dos semanas. En estas reuniones también se siguen unas determinadas pautas para conseguir la máxima eficiencia, entendimiento y calidad posible para el proyecto. Durante estas reuniones, el gestor de proyectos y el alumno tratan los siguientes asuntos:

- **Revisión del trabajo realizado:** se comprueba si se han completado las tareas asignadas para el sprint.
- **Demostración:** para las tareas completadas, se realiza una demostración delante del gestor de proyectos para que este aporte su feedback y en el caso de que se dé el visto bueno, se marca la tarea como completada.
- **Reporte de incidencias:** en el caso de que una tarea no haya sido aprobada por el gestor de proyectos, se le comenta al alumno el por qué de esta decisión y se propone una posible solución. Otro posible motivo por el cual una tarea no ha sido terminada es porque el alumno ha tenido problemas. En este caso el mismo debe explicar al gestor de proyectos el por qué y entre ambos deben intentar encontrar una solución para esta situación.
- **Planificación del siguiente sprint:** en esta última fase se decide cuales son las tareas con las que se va a trabajar en las siguientes dos semanas. En el siguiente sprint pueden aparecer nuevas tareas, tareas retrasadas o incidencias dadas en sprints anteriores.

A parte de estas reuniones, en la empresa siempre se aporta ayuda al alumno cuando es necesaria, incluso en periodos en los que no hay reuniones planeadas. Esto hace que el entorno de trabajo sea más cómodo y que la comunicación sea más fluida entre el alumno y el gestor de proyectos, aportando así un feedback de mayor calidad y de forma regular.

2.5. Riesgos del proyecto

En este apartado se mencionarán los riesgos que pueden aparecer durante la estancia en prácticas. Primero se realizará una identificación de los mismos y tras esto se aportará un análisis de los mismos.

2.5.1. Identificación de los riesgos

En la Tabla 2.2 aparecen los riesgos principales que se han tenido en cuenta para este proyecto. Para cada riesgo se ha aportado un identificador, una breve descripción y el tipo del riesgo.

ID	Descripción	Tipo
R01	Baja de un trabajador	Riesgo del proyecto
R02	Falta de experiencia del alumno	Riesgo del proyecto/producto
R03	Incorrecta comunicación entre los diferentes integrantes del equipo	Riesgo del proyecto
R04	Diseños incorrectos	Riesgo del producto
R05	Cálculo incorrecto de los costes del proyecto	Riesgo del proyecto

Tabla 2.2: Identificación de los riesgos

2.5.2. Análisis de los riesgos

En la tablas que se encuentran entre la Tabla 2.3 y la Tabla 2.7 inclusive, se incluyen los análisis de los riesgos mencionados en el apartado 2.5.1. Identificación de los riesgos. Concretamente se analizan los riesgos de la baja de un trabajador, la falta de experiencia del alumno en prácticas, la incorrecta comunicación entre los diferentes integrantes del equipo, los diseños incorrectos y el cálculo incorrecto de los costes del proyecto.

R01. Baja de un trabajador
Magnitud Impacto alto, debido a que el equipo está compuesto por tres personas. **Descripción** Algún miembro del proyecto no puede realizar tareas relacionadas con este debido a un motivo externo como puede ser una lesión o una enfermedad. **Impacto** Mayor carga de trabajo para el resto de trabajadores. **Indicadores** No hay. Al ser circunstancias externas nunca se puede predecir cuándo un trabajador puede darse de baja. **Plan de actuación** Las tareas del miembro que ha sufrido la baja se reparten entre los otros dos trabajadores.

Tabla 2.3: R01. Baja de un trabajador

R02. Falta de experiencia del alumno
Magnitud Medio/Alto. **Descripción** El alumno encargado de implementar la nueva funcionalidad para la aplicación no tiene un nivel lo suficientemente amplio como para realizar las tareas de codificación en el tiempo previsto. Requiere de tiempo extra para aprender cómo funcionan ciertos elementos. **Impacto** Se puede producir una pérdida de calidad del producto debido a que no se realizan las labores de programación de forma eficiente ni adecuada. **Indicadores** El alumno tarda mucho en realizar las tareas que se le encomiendan. **Plan de actuación** Se posponen las tareas en proceso para otorgar al alumno un mayor tiempo de formación.

Tabla 2.4: R02. Falta de experiencia del alumno

R03. Incorrecta comunicación entre los diferentes integrantes del equipo
Magnitud Alto debido a que en la metodología SCRUM la comunicación es vital. **Descripción** Los miembros del equipo no se comunican lo suficiente o comunican sus ideas, propuestas y/o opiniones de forma poco clara, irrespetuosa o confusa a la hora de realizar reuniones clave o cambios en alguna de las tareas. **Impacto** Se pueden provocar situaciones que bajan la moral al equipo lo que hace decaer tanto la eficiencia como la creatividad de los miembros. Además puede que este factor influya en la cantidad de tiempo necesario para realizar las tareas. **Indicadores** Diseños que no se corresponden con requisitos o con el código implementado, resultados mediocres o no aprobados por el resto del grupo. **Plan de actuación** Realizar un mayor número de reuniones a la semana y aumentar la duración de éstas en caso de ser necesario.

Tabla 2.5: R03. Incorrecta comunicación entre los diferentes integrantes del equipo

R04. Diseños incorrectos
Magnitud Aumenta conforme pasa el tiempo, cuantas más funcionalidades hayan sido implementadas más costoso resulta realizar cambios en la interfaz posteriormente. **Descripción** El diseño es inadecuado a la hora de realizar la programación. Puede que no sea lo suficientemente claro y que carezca de detalles, o simplemente que no sea apropiado para el público al cual va enfocado el producto. **Impacto** Se causarán inconvenientes sobre todo si se ha detectado el error después de haber programado las interfaces. Puede causar que se retrasen las tareas. **Indicadores** La interfaz resulta difícil de usar. El diseño no es atractivo o no cumple con los estándares especificados por el supervisor. **Plan de actuación** Rediseñar las interfaces y reunir a los miembros del equipo implicados en este proceso para obtener el visto bueno de todas las partes.

Tabla 2.6: R04. Diseños incorrectos

R05. Cálculo incorrecto de los costes del proyecto
Magnitud Medio. **Descripción** Se ha calculado de forma errónea el coste de los diferentes elementos necesarios para el proyecto. **Impacto** Puede producirse una situación en la que no se pueda pagar a los trabajadores o no se pueda adquirir todo el material necesario. **Indicadores** Se trabajan más horas de las previstas, o se utilizan ordenadores que no se tuvieron en cuenta en un primer lugar para realizar ciertas tareas. **Plan de actuación** Recortar gastos de los planificados inicialmente o tratar de obtener un presupuesto mayor.

Tabla 2.7: R05. Cálculo incorrecto de los costes del proyecto

Capítulo 3

Análisis y diseño del sistema

En este capítulo de la memoria, se explicará con detalle las características del sistema. Primero se realizará un análisis en el cual se especificarán los requisitos y se realizará un modelado del sistema, tras esto se describirán los componentes de dicho sistema y su interrelación, y finalmente se mostrará el diseño de la interfaz.

3.1. Análisis del sistema

En esta sección, se mostrará un análisis del sistema dividido en dos partes. La primera, constará de un diagrama de casos de uso, junto con casos de prueba para cada caso de uso utilizando la notación Gherkin [16], mientras que en la segunda se mostrarán los requisitos del sistema.

3.1.1. Diagrama de casos de uso

El diagrama de casos de uso [17] se utiliza en los proyectos de software para mostrar de una forma muy sencilla la forma en la que los diferentes usuarios interactúan con el sistema. En la Figura 3.1 se muestra el diagrama de casos de uso del sistema.

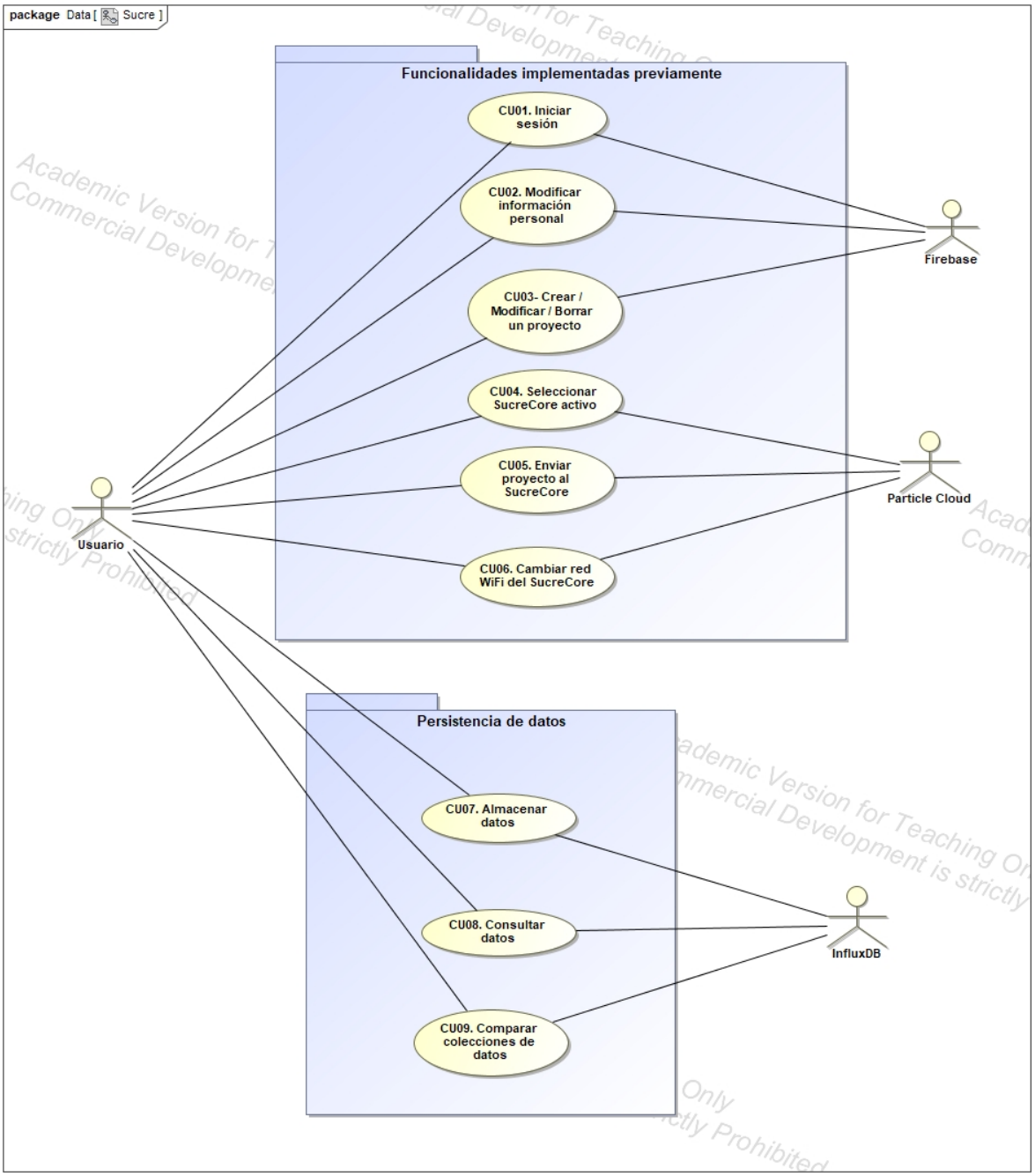

Figura 3.1: DCU del sistema

En el diagrama de casos de uso mostrado en la Figura 3.1, se han representado, por un lado, todas las funcionalidades de las que constaba la aplicación web del proyecto SUCRE antes del comienzo de la estancia en prácticas, y por otro lado, las funcionalidades que se van a implementar a lo largo de la misma.

A continuación, se muestran descripciones para los casos de uso que forman parte de la

persistencia de los datos, junto con aportaciones de escenarios posibles mediante el uso de la notación Gherkin. En las tablas situadas entre la Tabla 3.1 y la Tabla 3.3 inclusive, se describen el caso de uso relacionado con la acción de almacenar datos, el caso de uso relacionado con la acción de consultar datos y el caso de uso relacionado con la acción de comparar conjuntos de datos.

CU07. Almacenar datos
Como usuario necesito poder guardar un dato que tengo almacenado en una variable. **Recurso** Guardar un dato en la base de datos **Como** Usuario **Propósito** Poder ver posteriormente todos los valores recogidos
Escenario. Almacenar un dato recogido por un sensor Given Soy un estudiante con la sesión iniciada When Utilizo el bloque de guardado de datos con mi variable Then Mi variable se almacena en el sistema

Tabla 3.1: CU07. Almacenar datos

CU08. Consultar datos
Como usuario necesito poder consultar los datos pertenecientes a los conjunto de datos asociados a mis SucreCores. **Recurso** consultar datos de mis SucreCores **Como** usuario **Propósito** Poder observar los cambios en los valores almacenados a través de un SucreCore
Escenario Visualizar los datos recogidos por un sensor acoplado a mi SucreCore Given Soy un estudiante con la sesión iniciada When Accedo al apartado de lecturas e introduzco el SucreCore y el conjunto de datos del cual deseo ver los datos recogidos. Then Se muestra una tabla con los valores recogidos y la hora y fecha en las cuales fueron recogidos, además de un gráfico de líneas con estos datos.

Tabla 3.2: CU08. Consultar datos

CU09. Comparar conjuntos de datos
Como usuario necesito poder comparar datos pertenecientes a conjuntos de datos diferentes entre los que están asociados a mis SucreCores. **Recurso** Comparar los diferentes datos de mis SucreCores **Como** Usuario **Propósito** Poder observar las diferencias entre los valores recogidos por SucreCores diferentes o por el mismo SucreCore pero en diferentes instantes y/o condiciones.
Escenario Comparar las lecturas de dos SucreCores que tomaban valores mediante sensores en un mismo momento Given Soy un estudiante con la sesión iniciada When Accedo al apartado de lecturas e introduzco el SucreCore y el conjunto de datos del cual deseo ver los datos recogidos. Después selecciono el nuevo conjunto de datos que quiero añadir al gráfico. Then Se muestra en el gráfico una línea para cada conjunto, mostrando las diferencias de forma visual.

Tabla 3.3: CU09. Comparar conjuntos de datos

3.1.2. Requisitos del sistema

En este apartado se describirán los requisitos del sistema, por un lado los que hacen referencia a las funcionalidades que deben ser implementadas, por otro lado los relacionados con los datos que el sistema debe almacenar y finalmente los requisitos de calidad que el sistema debe cumplir.

3.1.3. Requisitos funcionales

En este apartado se describen los diferentes requisitos funcionales con los que debe cumplir el sistema. En las tablas situadas entre la Tabla 3.4 y la Tabla 3.8 inclusive, se describen los requisitos funcionales del sistema. Estos requisitos son, añadir el bloque de guardado de datos, consultar los datos almacenados, comparar conjuntos de datos, reiniciar la gráfica y editar la tabla.

RF01. Añadir bloque de guardado de datos
Descripción El sistema debe permitir al usuario añadir en su programa el bloque que guarda datos en la base de datos del sistema. **Prioridad** Alta **Comentarios** Se debe limitar el tiempo entre cada operación de guardado para no saturar la base de datos
Secuencia de acciones 1. Se accede a un proyecto o se crea uno nuevo 2. Se selecciona la categoría 'Online' de la lista de bloques 3. Se arrastra el bloque al marco donde se construye el programa 4. Se selecciona el tiempo que pasa entre cada operación de guardado, se escribe el nombre del conjunto de datos y se añade el dato que se desea guardar al bloque.

Tabla 3.4: RF01 Añadir bloque de guardado de datos

RF02. Consultar datos almacenados
Descripción El sistema debe permitir al usuario consultar datos asociados a alguno de sus SucreCores. **Prioridad** Alta **Comentarios** Se debe realizar a través de un formulario sencillo para seleccionar el SucreCore y el conjunto de datos que se quiere consultar. Se debe mostrar una tabla con información del conjunto de datos.
Secuencia de acciones 1. En la página de inicio, pulsar en la barra de herramientas en el icono de los tres puntos y seleccionar la opción 'Lecturas'. (Esta página se puede observar en la Figura 3.3, situada en el apartado 3.3. Diseño de la interfaz) 2. Se selecciona el SucreCore que se quiere consultar 3. Se selecciona el conjunto de datos que se quiere consultar

Tabla 3.5: RF02 Consultar datos almacenados

RF03. Comparar conjuntos de datos
Descripción El sistema debe permitir al usuario comparar conjuntos de datos de sus SucreCores. La comparación de los datos se realiza a través de la gráfica, ya que es una gráfica de líneas y por lo tanto cada línea representará un conjunto de datos **Prioridad** Media **Comentarios** Se debe implementar un formulario sencillo para poder seleccionar qué conjunto de datos añadir.
Secuencia de acciones 1. En la página de 'Lecturas' tras haber seleccionado un conjunto de datos a consultar, seleccionar otro en el formulario junto a la gráfica. 2. Pulsar el botón 'Añadir'.

Tabla 3.6: RF03 Comparar conjuntos de datos

RF04. Reiniciar gráfica
Descripción El sistema debe permitir al usuario borrar todas los conjuntos de datos añadidos a la gráfica excepto el primero seleccionado. **Prioridad** Baja **Comentarios** Se debe implementar un botón para poder ejecutar esta acción.
Secuencia de acciones 1. En la página de 'Lecturas' tras haber seleccionado un conjunto de datos a consultar y añadir otros conjuntos a la gráfica, pulsar el botón 'Reiniciar'.

Tabla 3.7: RF04 Reiniciar gráfica

RF05. Editar tabla
Descripción El sistema debe permitir al usuario editar la paginación de la tabla así como dirigirse a la página que desee en todo momento **Prioridad** Baja **Comentarios** Se debe incluir una serie de opciones a través de las cuales el usuario pueda seleccionar cuántos datos hay por cada página de la tabla.
Secuencia de acciones 1. En la página de 'Lecturas' tras haber seleccionado un conjunto de datos a consultar, dirigirse a la tabla. 2. Hacer clic en el selector donde se indica el número de valores mostrados por página. 3. Seleccionar la opción deseada.

Tabla 3.8: RF05 Editar tabla

3.1.4. Requisitos de datos

Durante el desarrollo del proyecto, no se implementó la función que permite a los usuarios crear cuentas e iniciar sesión, pero aun así, es importante tenerla en cuenta ya que es un requisito imprescindible para realizar cualquier acción en la aplicación.

En la Tabla 3.9 se describe el requisito de datos relacionado con la información del usuario, y en la Tabla 3.10 se describe el requisito de datos relacionado con los valores que el usuario tiene almacenados.

RD01. Información del usuario
Descripción El sistema debe almacenar la información de los usuarios, de forma obligatoria el correo electrónico y la contraseña, y de forma opcional el nombre y el centro al que pertenece. **Prioridad** Alta **Comentarios** La información del usuario permite al sistema saber en todo momento qué SucreCores tiene en propiedad, cuáles de ellos están activos, los proyectos que el usuario ha creado, etc.

Tabla 3.9: RD01 Información del usuario

RD02. Valores almacenados
Descripción El sistema debe almacenar los datos que el usuario guarde en sus programas, concretamente el valor del mismo, el SucreCore en el que se ha generado, el grupo de valores al que pertenece, cuyo nombre decide el usuario (p.ej. Temperatura, luz, humedad) y una marca de tiempo. **Prioridad** Alta **Comentarios** Para poder mostrar datos el usuario debe, como mínimo, haber guardado un conjunto de datos de datos en la base de datos.

Tabla 3.10: RD02 Valores almacenados

3.1.5. Requisitos de calidad

En este apartado se describen los diferentes requisitos de calidad con los que debe cumplir el sistema. En la Tabla 3.11, la Tabla 3.12 y la Tabla 3.13 se describen los requisitos de calidad acerca de la visualización de datos de forma rápida y sencilla, el aviso acerca de un SucreCore sin conjuntos de datos y el uso del patrón MVC para separar la lógica de la visualización de los datos.

RC01. Visualización de datos rápida y sencilla
Autor Enrique Cueto Rubio **Versión** 1.0 **Descripción** El sistema debe minimizar las interacciones que el usuario debe realizar para poder ver los conjuntos de datos de sus SucreCores. Las únicas interacciones que debe tener son para seleccionar el SucreCore y el conjunto de datos. **Actor principal** Interfaz

Tabla 3.11: RC01 Visualización de datos rápida y sencilla

RC02. Aviso acerca de un SucreCore sin conjuntos de datos
Autor Enrique Cueto Rubio **Versión** 1.0 **Descripción** El sistema debe mostrar un aviso cuando el usuario seleccione un SucreCore que no tiene conjuntos de datos almacenados en la interfaz de lecturas. El aviso no debe ser estresante ni exagerado, únicamente informativo, de modo que el usuario sepa que no ha almacenado datos con su SucreCore pero que tampoco ha cometido ningún error. **Actor principal** Interfaz

Tabla 3.12: RC02 Aviso acerca de un SucreCore sin conjuntos de datos

RC03. Uso del patrón MVC para separar la lógica de la visualización de los datos
Autor Enrique Cueto Rubio **Versión** 1.0 **Descripción** Se emplea el patrón de diseño MVC (modelo-vista-controlador) para así separar la lógica del programa de la visualización de los datos. Esto permite una mejor organización del código, cosa que facilita la modificación del mismo en caso de errores o la implementación de nuevas funcionalidades. **Actor principal** N/A

Tabla 3.13: RC03. Uso del patrón MVC para separar la lógica de la visualización de los datos

3.2. Diseño de la arquitectura del sistema

En esta sección del análisis y diseño del sistema se describirán los componentes del sistema y la relación que tienen entre ellos. Con el fin de realizar una explicación más clara, se ha dividido este apartado en varios capítulos según el objetivo con el cual se han utilizado los diferentes componentes el sistema.

3.2.1. Componentes iniciales

En este apartado se explicarán los componentes que se implementaron antes de que el alumno comenzara la estancia en prácticas.

Por un lado, el sistema se comunica con Firebase, una plataforma en la nube para el desarrollo de aplicaciones web y móviles. Este servicio proporcionado por Google actúa como *backend*, lo cual hace que la arquitectura del sistema se clasifique como *serverless*. "*Las arquitecturas sin servidor, o serverless, son diseños de aplicaciones que incorporan servicios 'Backend as a Service' (BaaS) de terceros y / o que incluyen código personalizado ejecutado en contenedores administrados y efímeros en una plataforma FaaS (Funciones como servicio)*" [18]. Firebase proporciona varias utilidades, de las cuales en el proyecto se utilizan:

- **Firebase Auth** Permite implementar en el sistema un método de autenticación, lo que permite crear un sistema de inicio de sesión y de reconocimiento de usuarios. Cuando un usuario se da de alta en la aplicación se genera un ID y se almacena junto con su contraseña en la base de datos, la cual se consulta cada vez que un usuario quiere iniciar sesión en su dispositivo.
- **Firebase Cloud Store** Es una base de datos NoSQL que permite a los desarrolladores almacenar datos en la nube. Esta base de datos se organiza en forma de documentos agrupados en colecciones. Estos documentos pueden almacenar campos de diferentes tipos (numéricos, texto, booleanos, etc.) u otras subcolecciones. En el proyecto SUCRE se utiliza este servicio para almacenar información acerca de los usuarios, SucreCores, proyectos creados por los usuarios, etc.
- **Firebase Hosting** La aplicación del proyecto SUCRE utiliza este servicio de *hosting* el cual cuenta con una red de entrega de contenido global, lo que implica que se puede acceder a la aplicación desde cualquier parte del mundo.

A parte de Firebase, el sistema también cuenta con el servicio Particle Cloud. Este servicio permite a los SucreCores conectarse a internet, haciendo posible que desde el sistema se puedan reconocer los dispositivos que estan activos en un momento determinado, que se puedan actualizar los SucreCores desde cualquier lugar, etc. Esto da una mayor flexibilidad y usabilidad al producto, ya que nos permite usar la aplicación y los SucreCores desde cualquier parte.

Cabe comentar que para el *frontend* se ha utilizado Angular, un frameweork utilizado para crear aplicaciones web escalables. Este framework de código abierto escrito en TypeScript tiene como objetivo principal el desarrollo de aplicaciones de una sola página. Una gran ventaja

que proporciona Angular es que Google se encarga del mantenimiento y las actualizaciones del mismo.

3.2.2. Base de datos InfluxDB

A parte de Firebase, el sistema necesitaba una base de datos independiente para almacenar los datos que el usuario genera y guarda para poder estudiar posteriormente. Es por esto que el gestor de proyectos implementó una base de datos InfluxDB mediante el uso de Amazon Web Services (AWS), en la cual se almacenarían estos datos. El motivo de la creación de esta base de datos se debe a que Firebase tiene un límite de operaciones de lectura y escritura que se pueden realizar diariamente. Se espera que el proyecto SUCRE escale y se utilice en institutos de todo el país, por lo tanto resultaba prácticamente imposible mantener las operaciones por debajo del límite y es por ello por lo que se decidió crear esta base de datos nueva.

3.2.3. Escrituras en InfluxDB

Las operaciones de escritura en la base de datos se realizan desde el microcontrolador, es decir, desde el SucreCore. Cuando un usuario incluya el bloque de almacenar datos, se generará el código necesario en el programa del microcontrolador, el cual escribe en un canal MQTT el mensaje que más tarde desencadenará en la operación de escritura deseada.

MQTT [19] son las siglas de Message Queuing Telemetry Transport y es un protocolo de mensajería simple que se basa en el modelo publicador-suscriptor. En este proceso, se establece un canal a través del cual se realizan las comunicaciones entre ambas partes, y el publicador se encarga de mandar mensajes los cuales llegan a aquellas partes que estén suscritas a ese canal. En nuestro caso, el SucreCore es el publicador, ya que es el que manda el mensaje al canal con la información necesaria: nombre del SucreCore, nombre del conjunto de datos en el que se almacenará el dato, y el valor del dato en sí. Por otro lado el servidor creado mediante AWS cumple con el rol de suscriptor, ya que se dedica a leer los mensajes que se escriben en el canal y escribir en la base de datos InfluxDB la información pertinente. Además de los datos pasados a través del SucreCore, la base de datos genera una marca de tiempo para cada valor.

3.2.4. Lecturas en InfluxDB

Se ha comentado el proceso de escritura en la base de datos pero, ¿qué hay de las lecturas? En cuanto a las lecturas, el sistema realiza peticiones siguiendo el protocolo HTTP a través de una API REST creada mediante Node-RED. Esta API la desarrolló el alumno con el objetivo de establecer comunicaciones con la base de datos InfluxDB donde se almacenan los datos de las lecturas de los SucreCores.

La implementación de la API como se ha mencionado, se realizó a través de Node-RED. Node-RED es una herramienta de programación utilizada para conectar dispositivos hardware, APIs y servicios en línea. Esta herramienta proporciona un editor al cual se accede mediante el

navegador, y 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. [20]

3.2.5. Esquema del sistema

En la Figura 3.2 se aporta un esquema para una comprensión más rápida y visual de los elementos del sistema y la relación que tienen entre ellos:

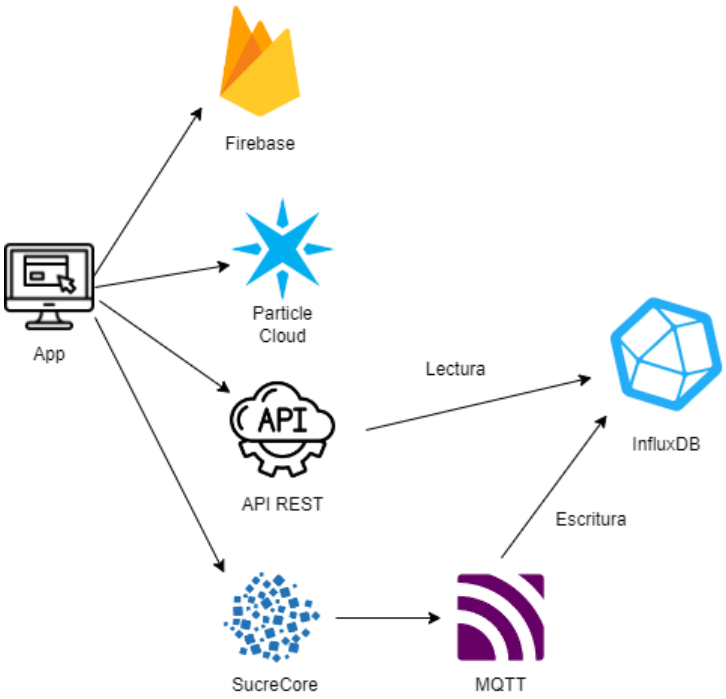

Figura 3.2: Esquema del sistema

3.3. Diseño de la interfaz

En el proyecto SUCRE ya se contaba con una aplicación construida con anterioridad, y por lo tanto lo que se debía hacer era generar una nueva página para que los usuarios pudieran observar los datos. Se quería mantener un formato sencillo para la interfaz dado que se está trabajando sobre una aplicación WEB que será utilizada tanto por ordenadores como para dispositivos móviles. Es por eso que implementar una gran variedad de páginas comportaría un mayor número de páginas que cargar y esto resultaría más tedioso para el usuario, y se decidió juntar todo en una misma interfaz.

Se pudieron aprovechar elementos que ya estaban creados con anterioridad como una barra de tareas en la parte superior de la pantalla, o una imagen de fondo con una temática y colores a juego con el logotipo del proyecto SUCRE. En la Figura 3.3 se observa la interfaz correspondiente

a la pantalla de inicio. En base a esta se creó un prototipo para la interfaz de lectura de datos.

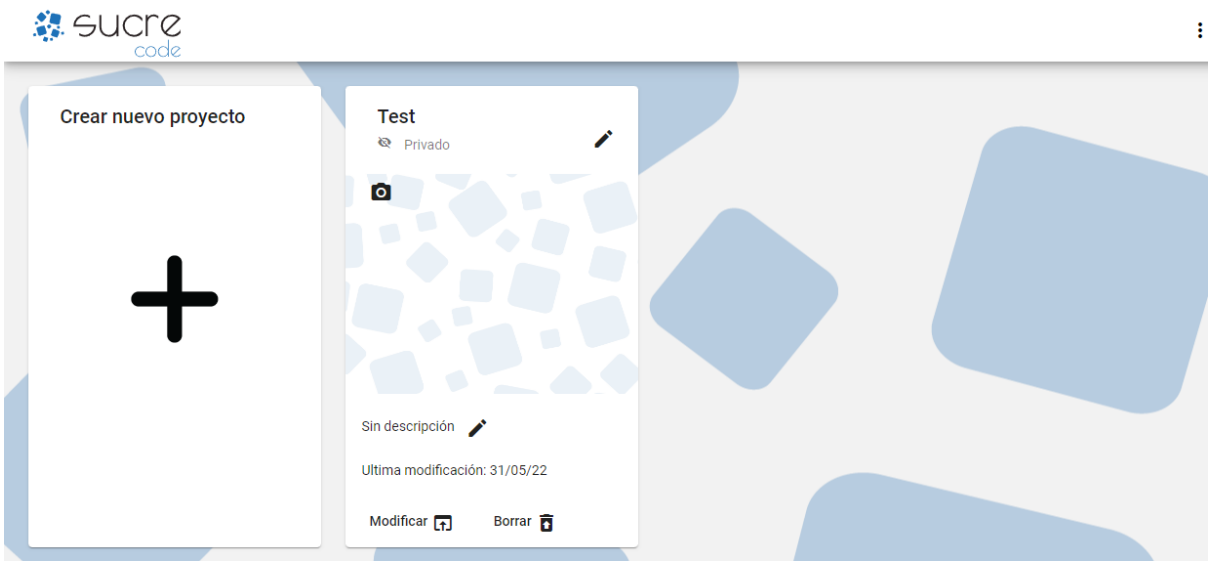

Figura 3.3: Prototipo inicial de la interfaz

Capítulo 4

Implementación y pruebas

En este capítulo se explicarán los detalles de la implementación y las pruebas que se han realizado para probar el sistema desarrollado.

4.1. Detalles de implementación

En esta primera sección del capítulo de implementación y pruebas se explicarán, como ya se ha comentado antes, el trabajo de programación realizado. Para un mejor entendimiento del por qué se han seleccionado ciertas tecnologías o se han realizado cambios respecto a las tareas planeadas inicialmente, se describirá el proceso de implementación en el cual se comentarán los resultados de los sprints, las incidencias sufridas y las decisiones tomadas. Por otro lado, se explicarán los patrones y estrategias utilizados.

4.1.1. Primer sprint. Comunicación SucreCore-Firebase

Tras haber realizado una primera planificación y haber asignado las tareas al sprint, el objetivo era encontrar un método para realizar una conexión entre el SucreCore y la base de datos Firebase para poder escribir en esta los valores.

Para poder establecer dicha conexión, primero se consultaron opciones proporcionadas por el equipo de Particle Industries, Inc., ya que dicha empresa es la creadora del microcontrolador al que en el proyecto SUCRE se le conoce como SucreCore. Desde Particle diseñaron un sistema que permite realizar conexiones con la base de datos Firebase [3], y realizar tanto lecturas como escrituras en esta. Se probó este sistema, y se comprobó que funcionaba, aunque presentaba un inconveniente clave. Desde la empresa Particle implementaron una limitación que impedía a los desarrolladores utilizar este recurso más de un cierto número de veces al mes, es decir, hay un límite de operaciones que un usuario puede realizar, y en el caso de sobrepasarlo se tendría que pagar el coste adicional. La empresa decidió buscar otra alternativa, ya que este gasto podría dispararse al no ser controlable, ya que puede que los usuarios, ya sea por error o de forma inconsciente, realicen una cantidad excesiva de lecturas o escrituras excesivas.

Tras descartar esta opción, se decidió buscar alguna librería que permitiera establecer la conexión SucreCore-Firebase. Particle desarrolló una librería para realizar peticiones HTTP, con la cual se podría hacer uso de la API REST desarrollada por Google para poder realizar peticiones a Firebase. No obstante Firebase utiliza el protocolo HTTPS, y por lo tanto, esta opción tampoco obtuvo éxito.

Otra vía mediante la cual el microcontrolador se puede conectar con Firebase y que se planteó es utilizar IFTTT. Este servicio permite crear y programar diferentes acciones entre aplicaciones de forma sencilla. La idea de usar esta aplicación era que se podrían generar gráficos directamente en una hoja de cálculo y almacenarla en la cuenta de Google del usuario, lo cual resultaría muy práctico. Aun así se decidió descartar esta opción ya que resultaba incómodo para el usuario final por el hecho de que este estaba obligado a crearse una cuenta en IFTTT y enlazarla con su cuenta de Google.

Finalmente se decidió intentar usar librerías desarrolladas por usuarios y que sí que permitieran realizar peticiones HTTPS. Desde un principio, se descubrieron pocas, pero la más conocida y recomendad era la desarrollada por el usuario *jersey99*, llamada httpsclient-particle [21]. Esta librería fue probada por los desarrolladores de Particle, los cuales ejecutaron los dos programas de ejemplo que proporcionaba el creador de la librería. Como uno de los programas si que funcionaba decidimos probar suerte, sin embargo el microcontrolador no podía conectarse a la base de datos tampoco con esta opción.

Tras realizar estas pruebas y quedarnos sin opciones, en la revisión del sprint el gestor de proyectos decidió que la mejor decisión por la que podíamos optar era tomar un primer contacto con Firebase a través de la aplicación Angular y comenzar el proceso de implementación de la interfaz mientras en la empresa se planteaban una posible solución para este problema. En la aplicación Sucre ya habían realizado operaciones relacionadas con Firebase, por lo tanto era seguro que no sucedería un caso parecido al anterior.

Por lo tanto, las tareas del primer sprint el cual se comenta en el apartado 2.2. Planificación Inicial y las cuales se observan en la Figura 4.1, quedan todas como pendientes, y se incluirán más adelante.

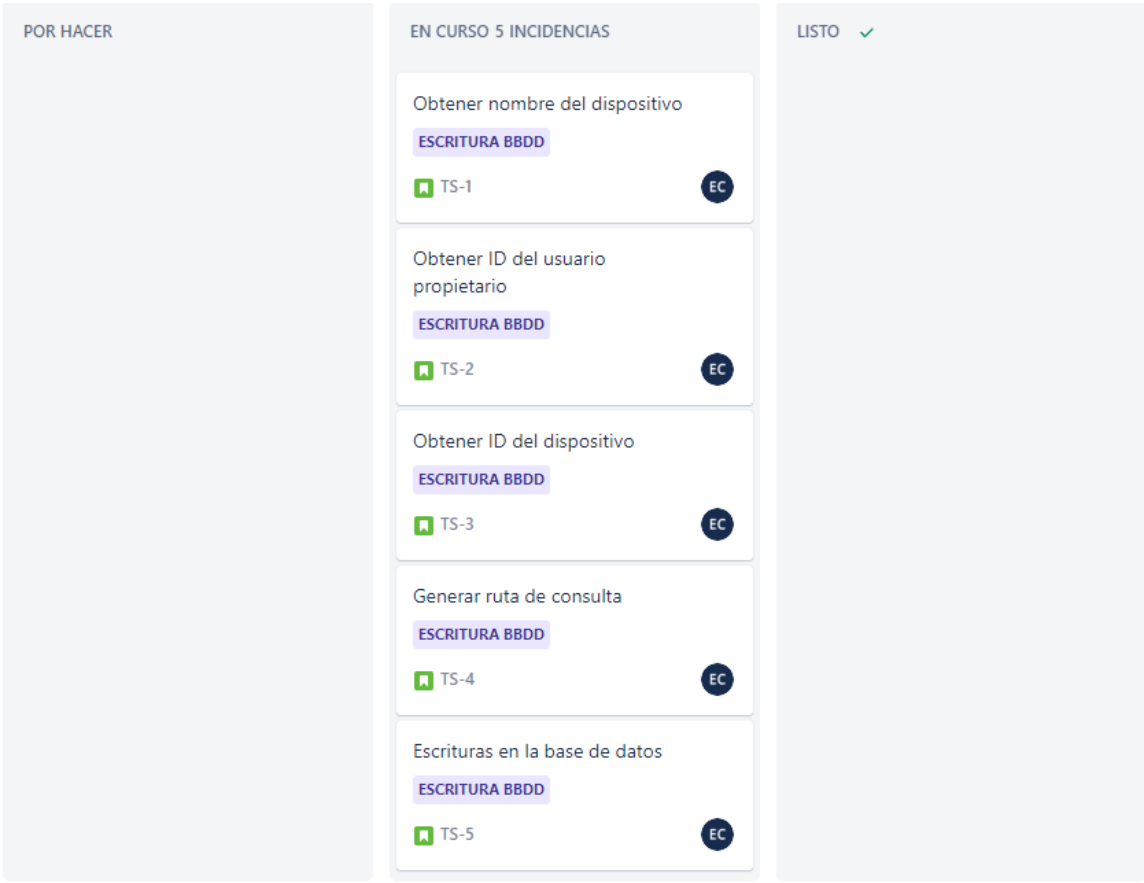

Figura 4.1: Resultados del primer sprint

El segundo sprint, como se ha comentado, se basará en realizar lecturas de la base de datos desde la parte de la aplicación Angular e implementar una primera versión de la interfaz. El objetivo de este segundo sprint es comprobar que se pueden obtener datos desde Firebase para que el usuario pueda visualizarlos. El sprint, por lo tanto, queda como se ve en la figura 4.2.

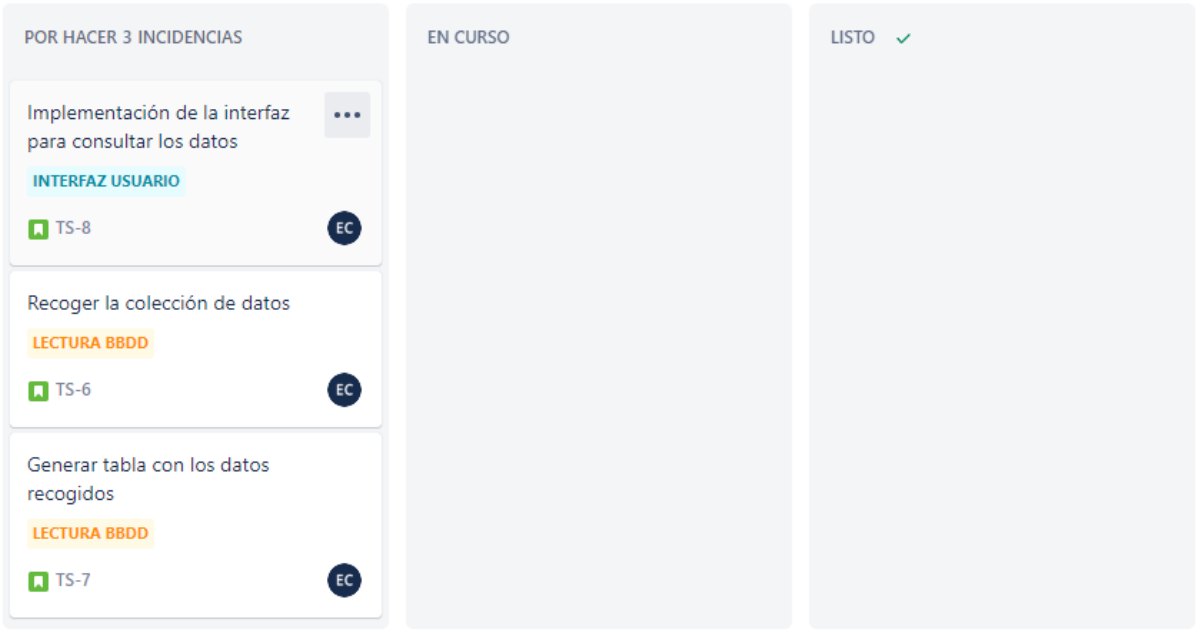

Figura 4.2: Tareas iniciales del segundo sprint

4.1.2. Segundo sprint. Lectura de datos e interfaz

En este sprint, como se ha mencionado en el apartado anterior, el objetivo es establecer una primera conexión con la base de datos Firebase mediante la aplicación Angular para recibir información de esta y construir una primera versión de la interfaz para poder visualizar dichos datos.

Primero, cabe comentar que Firebase Cloud Firestore es una base de datos NoSQL, cuya estructura se organiza en colecciones y documentos, en los cuales se pueden almacenar campos u otras subcolecciones. En la Figura 4.3 se muestra el modo en el que se almacenan los datos para las lecturas en la base de datos Firebase para el proyecto SUCRE.

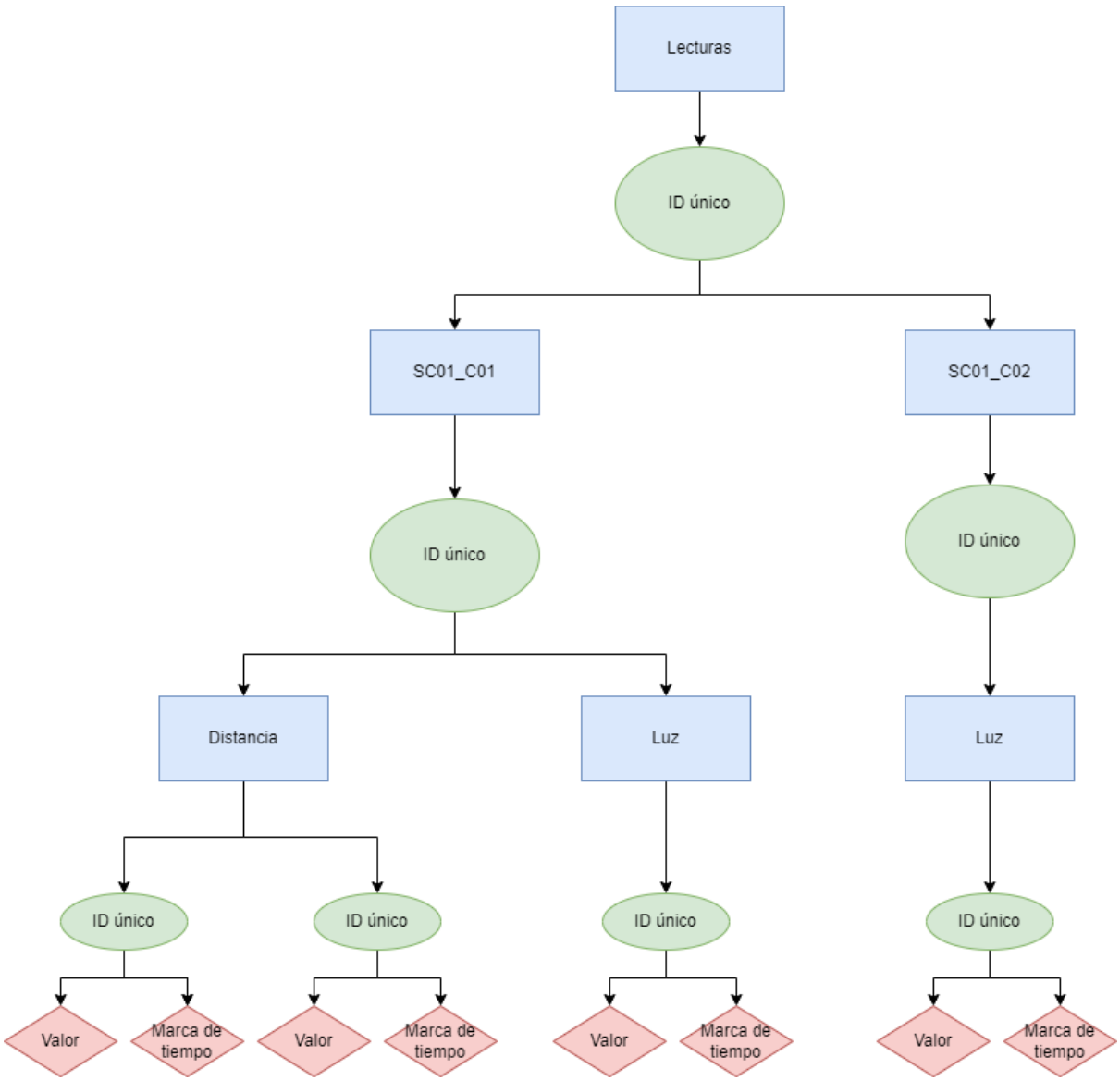

Figura 4.3: Estructura base de datos Firebase

Los rectángulos azules representan las colecciones. Dentro de estas únicamente puede haber documentos, es por eso que después de cada colección aparecen uno o más documentos, los cuales se identifican como círculos verdes en el diagrama. Como podemos observar, primero tenemos una colección llamada 'Lecturas', la cual ejerce de raíz. Tras esta, se realizan divisiones, y se crea una colección para cada SucreCore, con el fin de aportar una mejor organización a la base de datos. A continuación, vemos que los documentos que contienen las colecciones con nombres de SucreCores tienen asociadas una colección para cada conjunto de datos. Finalmente podemos comprobar que para cada conjunto de datos, se almacena un documento para cada valor recogido, el cual tiene dos campos representados con un rombo de color rojo, el valor del dato y una marca de tiempo que muestra cuándo se obtuvo el mismo.

Personalmente, el uso de este tipo de base de datos me parece poco óptimo para almacenar

datos de esta forma, pero entiendo que el equipo de UBIK Geospatial Solutions quisieran utilizar Firebase Cloud Firestore para este propósito, puesto que ya lo utilizaban para almacenar otro tipo de información como por ejemplo los datos de los usuarios, y no se disponía de un servidor a parte para crear una base de datos en la que almacenar estos datos. Es importante mencionar que, como en el sprint anterior no se pudo escribir datos en la base de datos utilizando SucreCores, se tuvieron que añadir datos de forma manual mediante la aplicación WEB de Firebase.

Desde la aplicación Angular se pudieron realizar las lecturas de la base de datos gracias a la implementación del método mostrado en la Figura 4.4.

```
getLecturasValues(scName: string, varName: string): Array<any>{
  var lectures = [];
  firebase.firestore().collection(colLect).limit(1).get().then(querySnapshot =>{
      querySnapshot.docs[0].ref.collection(scName).limit(1).get().then(querySnapshot =>{
        querySnapshot.docs[0].ref.collection(varName).orderBy("timestamp").onSnapshot((querySnapshot) => {
          querySnapshot.forEach((doc) => {
            var dateLecture = new Date(doc.get("timestamp").seconds*1000);
            var fechaString = dateLecture.toLocaleString();
            var horaString = dateLecture.toLocaleTimeString();
            lectures.unshift({"valor": doc.get("valor"), "fecha": fechaString, "hora": horaString});
          });
        });
      });
  });
  return lectures;
}
```

Figura 4.4: Método para recoger valores de una colección de un SucreCore

El procedimiento que se sigue en este método es el siguiente. Primero se accede a la colección 'Lecturas' y se obtiene el primer documento. Tras esto, usando el nombre del SucreCore, el cual se le ha pasado como argumento, se obtiene el documento que contiene las variables. A través del nombre del conjunto de datos que también le han pasado como argumento, se obtiene todo el conjunto de datos asociado a la colección. Finalmente se devuelve un vector, cada objeto del cual tiene tres atributos: valor, fecha y hora.

Este método se incluye dentro de una clase ya creada anteriormente, en la cual se incluyen todas las operaciones realacionadas con Firebase y que forma parte del paquete 'controller'

Durante este sprint, también se empezó a implementar una primera versión de la interfaz. Para esto, se tuvo que crear un nuevo paquete al que se le llamó 'data', debido a que el propósito del mismo era la construcción de una interfaz a través de la cual el usuario podría consultar los datos almacenados en sus SucreCores. En la Figura 4.5 se muestra el nuevo paquete creado junto con los ficheros que contiene.

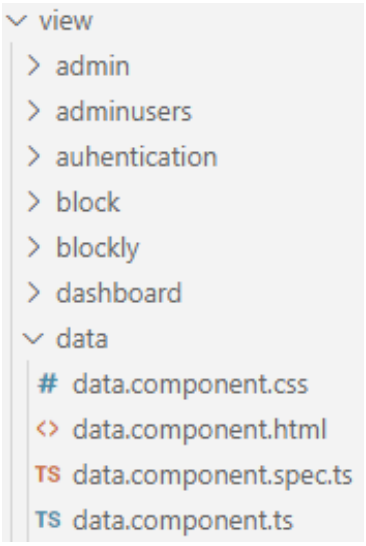

Figura 4.5: Paquete Data

Como se observa en la figura 4.5, al crear este paquete se generaron cuatro ficheros. Uno de estos es de tipo CSS, y se incluye para poder configurar el aspecto de la página, otro de estos es el archivo 'spec', el cual se genera de forma automática, y con el cual no trabajaremos en este caso. A continuación tenemos el fichero HTML con el que codificaremos la interfaz. En este archivo es donde se implementan los elementos que muestran información e interactúan con el usuario. Finalmente, tenemos el archivo 'component.ts', el cual se usa para darle vida a la interfaz, es decir, en este fichero es donde se incluyen las funciones que se ejecutan cuando se abre la página o cuando el usuario realiza una interacción con la interfaz a través de los diferentes elementos de esta (botones, listas, menús, etc.). De entre estos cuatro, los dos ficheros con los que más se ha trabajado son el HTML y el 'component.ts'.

Para la interfaz, se pensó incluir dos selectores que mostraran en el primero los SucreCores que el usuario tiene en propiedad y en el segundo los conjuntos de datos contenidos en el SucreCore seleccionado. Para la selección del SucreCore se utilizó código ya creado por los trabajadores de la empresa anteriormente y que servía para obtener los SucreCores que el usuario activo tiene en propiedad. Esta acción se realiza de forma automática cuando se entra en esta interfaz, lo que conlleva que el componente debía incluir un método llamado ngOnInit, el cual como indica su nombre, se ejecuta al inicio cuando se accede a esta interfaz.

Después de esto, se ejecuta otro método en el caso de que el usuario tenga al menos un SucreCore en propiedad, el cual obtiene los conjuntos de datos almacenados en el primero de los SucreCore que posee el usuario para así darle la opción a este de poder seleccionar el que quiera observar. Por defecto, se selecciona el primer conjunto de datos del primer SucreCore obtenido y se muestran los datos de este al usuario, dando un aspecto más dinámico y ofreciendo información de forma rápida. La forma de mostrar los datos, por el momento, es en una tabla que muestra los datos obtenidos Un ejemplo de como queda la interfaz tras estas operaciones automáticas es el que se puede apreciar en la Figura 4.6.

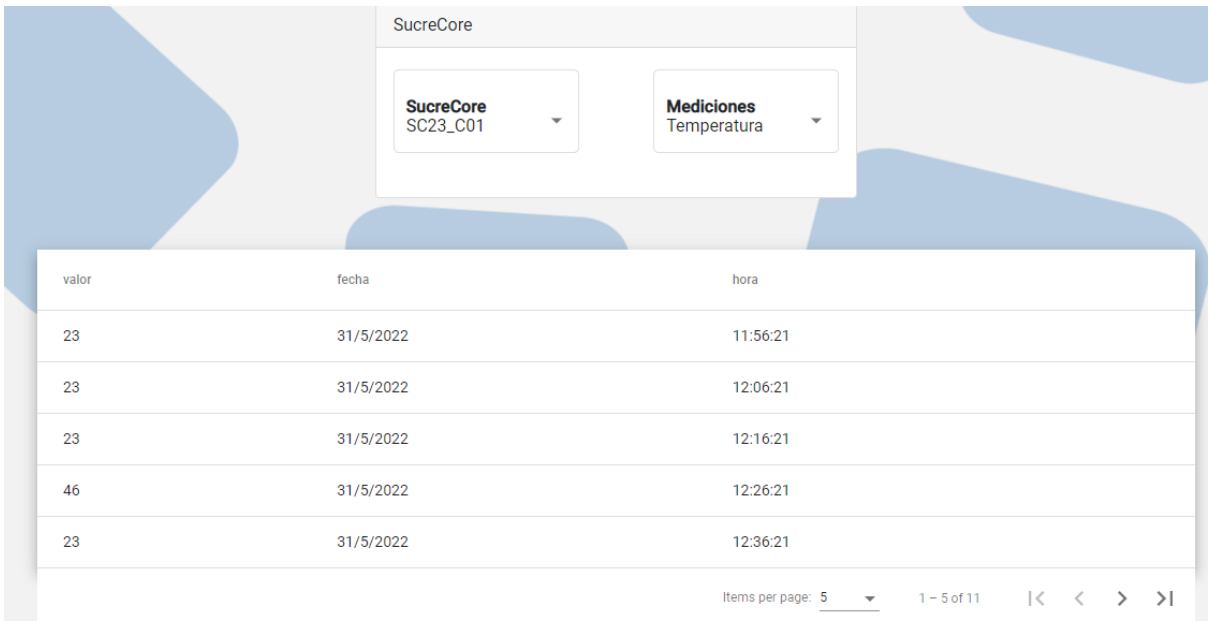

Figura 4.6: Interfaz de observación de los datos recogidos

No obstante, todavía quedaba pendiente implementar un método que recogiera todas las variables de un mismo SucreCore y las incluyera en el listado 'Mediciones' que se observa en la interfaz. Por lo tanto, el resultado del sprint es el que se muestra en la Figura 4.7.

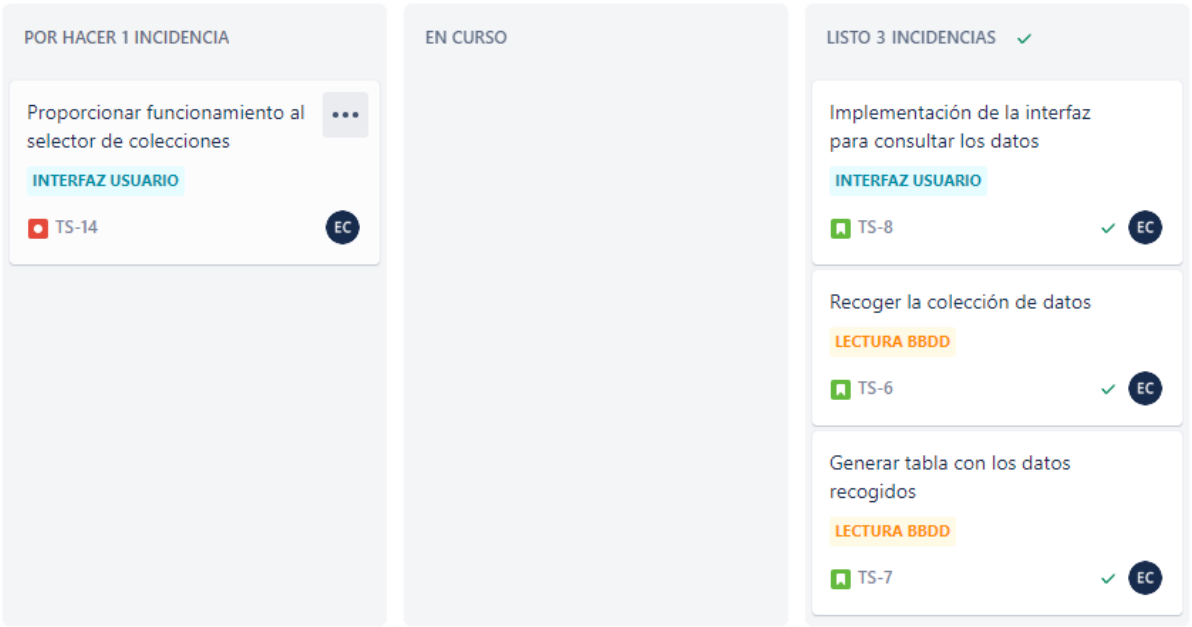

Figura 4.7: Resultados del segundo sprint

Se establecieron todas las tareas como terminadas, debido a que realmente el selector de

conjuntos de datos era un pequeño detalle. Es por eso que en lugar de marcar la tarea de la implementación de la interfaz como no acabada, decidimos crear una pequeña incidencia que representara esta tarea pendiente.

4.1.3. Tercer sprint. Inconvenientes de Firebase

En el apartado anterior, se ha mostrado el resultado del sprint 2, en el cual se incluye una incidencia que se arrastra al tercer sprint. Se decidió que era prioritario solucionar este error antes de decidir qué tareas eran las siguientes.

En la reunión de revisión del sprint, se mostraron los resultados al gestor de proyectos, y el alumno propuso incluir un atributo en los documentos de los SucreCores que contuviera los nombres de dichos conjuntos de datos (luz, distancia, etc.). Este atributo se establecería para poder conocer los nombres de las variables de cada SucreCore sin tener que realizar todas las lecturas de las siguientes colecciones. Los resultados cambiaron ligeramente el esquema de la base de datos, dándole el aspecto mostrado en la Figura 4.8.

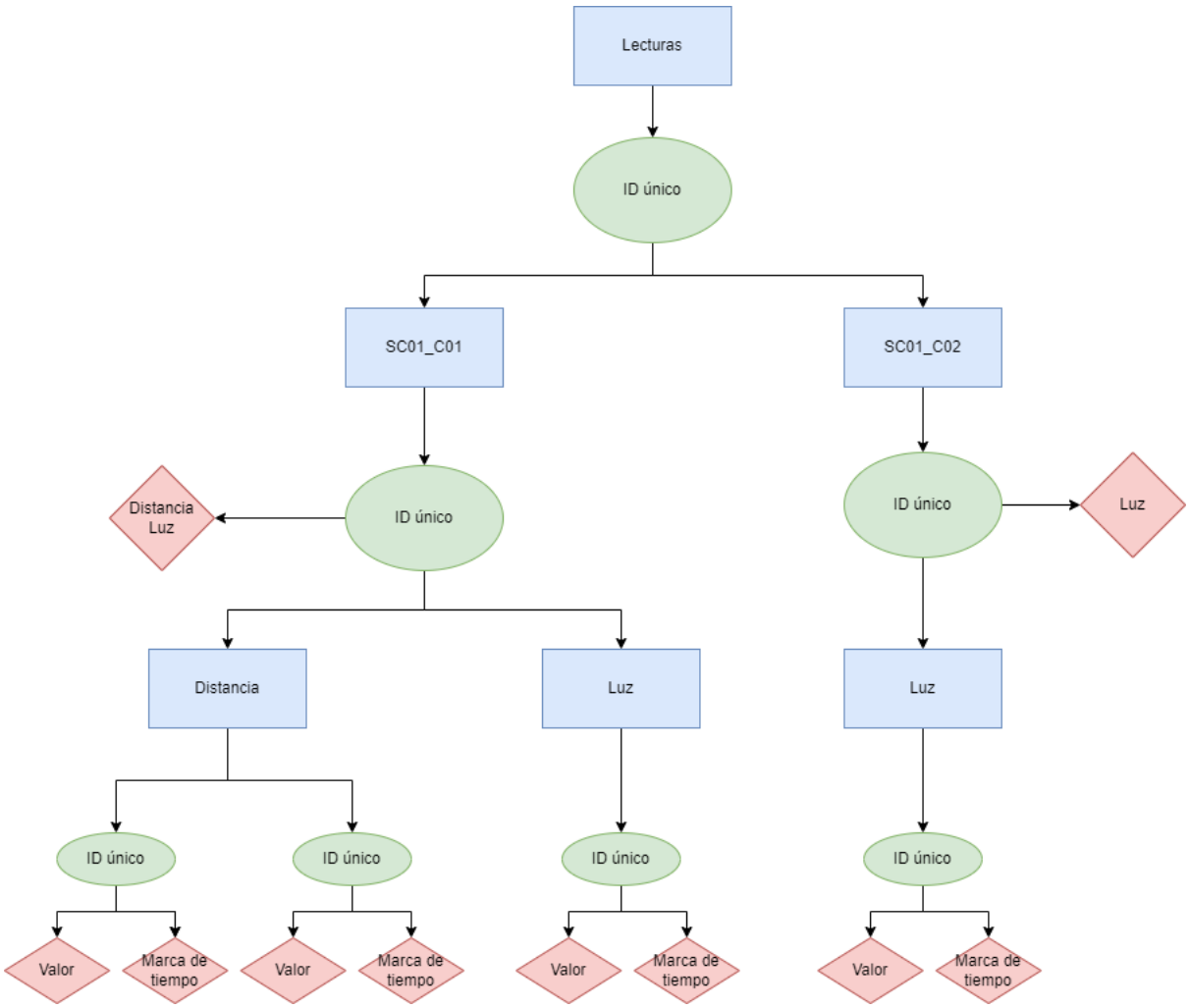

Figura 4.8: Nueva estructura de la base de datos Firebase

Tras realizar esta modificación, se implementó otro método, parecido al mostrado en la Figura 11, pero que esta vez accedía a este nuevo atributo, y devolvía un vector que contenía los nombres de todas los conjuntos de datos contenidos en el SucreCore pasado como argumento. Véase el código implementado en la Figura 4.9.

```
getVariables(scName: string): Array<any>{
  var variables = [];
  firebase.firestore().collection(colLect).limit(1).get().then(querySnapshot =>{
      querySnapshot.docs[0].ref.collection(scName).limit(1).get().then(querySnapshot =>{
          querySnapshot.forEach((doc) => {
            Array.prototype.push.apply(variables,(doc.get("subcolid")));
          });
      });
  });
  return variables;
}
```

Figura 4.9: Método para obtener conjuntos de datos contenidos en un SucreCore

Fue tras la implementación de este método, que se produjo un incidente clave y que cambiaría el desarrollo del proyecto por completo. Firebase es un servicio gratuito que permite a los desarrolladores almacenar información en la nube, no obstante, este servicio no ofrece almacenamiento ilimitado. Google limita la cantidad de datos que se pueden almacenar en una misma base de datos, y en caso de sobrepasar este límite, se le cobrará al desarrollador un coste extra en proporción al exceso de almacenamiento usado.

Otro problema del que nos dimos cuenta fue que Firebase también limita el número de operaciones de lectura y escritura que se realizan en la base de datos en un mismo día, y se cobrarían tarifas adicionales en caso de sobrepasar estos límites. Esto es un problema, ya que como se espera que el producto llegue a centros de todo el país, este límite se podría sobrepasar de forma muy fácil. En concreto, el número de operaciones diarias que se pueden ejecutar de forma gratuita son 50000 lecturas y 20000 escrituras. Además, hay que tener en cuenta también que por lectura nos referimos a la obtención de un valor, es decir si un usuario tiene un conjunto de datos con 100 valores, esto implicaría realizar 100 lecturas.

Con intención de reducir el número de operaciones de lectura, el alumno implementó un método cuya función era almacenar la información recogida en la caché del navegador, de forma que aunque se consultaran diferentes conjuntos de datos una y otra vez, solamente se consultara la base de datos la primera vez que se accediera a cada uno de estos conjuntos de datos. Por ejemplo, siguiendo el esquema proporcionado en la Figura 16, si el usuario consultara primero la colección ‘Distancia‘, luego la colección ‘Luz‘ y después otra vez la colección ‘Distancia‘, en lugar de realizar tres veces el acceso a la base de datos, solamente se haría dos, ya que la variable ‘Distancia‘ estaría almacenada en la memoria caché navegador y la segunda consulta de esta variable sería solucionada gracias a esta.

Aun así, este sistema tenía también un fallo, ya que con que el usuario añadiera un solo valor nuevo a una colección, se debería realizar otra consulta a la base de datos, cosa que supondría más operaciones de lecturas utilizadas.

Vista la situación, el alumno se puso en contacto con el gestor de proyectos y se organizó una reunión en la que se decidió que se buscaría una alternativa para solucionar este problema. Mientras tanto, el alumno debería implementar un sistema que representara los datos en formato de gráfica, para así aportar una forma más visual de mostrar al usuario los valores de su

SucreCore. Además, se decidió que no se implementaría la tarea que permitía al usuario borrar o modificar los valores mostrados en la tabla, ya que se llegó a la conclusión de que no era realmente necesario.

A causa de esta incidencia, la única tarea que pudo ser finalizada fue terminar de implementar el funcionamiento para la selección de conjuntos de datos. Por lo tanto, el sprint termina como se puede observar en la Figura 4.10.

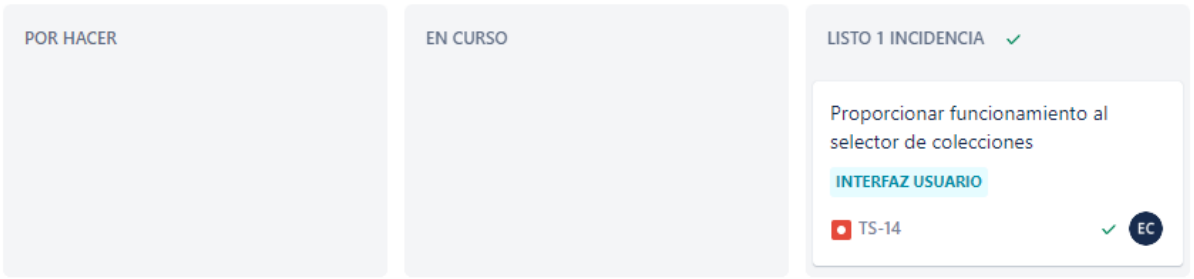

Figura 4.10: Resultados del tercer sprint

4.1.4. Cuarto sprint. Gráficos

Para este cuarto sprint, se puso como propósito incluir gráficos que representaran los datos recogidos por los SucreCores. Como este aspecto no estaba previsto en la planificación inicial, se crearon nuevas tareas y se añadieron a este cuarto sprint, el cual se puede observar en la Figura 4.11.

Figura 4.11: Tareas iniciales del cuarto sprint

Para llevar a cabo esta tarea, se utilizó ngx-charts [10], un recurso que permite implementar gráficos de forma sencilla y con varias características interesantes como el posible uso de leyendas, animaciones, líneas de tiempo, etc.

Su implementación fue realmente sencilla, se especifican ciertos parámetros para añadir o eliminar ciertos elementos de la gráfica, y se le pasan los valores en forma de vector. En este caso, se decidió implementar un gráfico de líneas, ya que por un lado, este es el que muestra los datos de una forma más clara y descriptiva para el usuario, y por otro lado, este tipo de gráfico es la mejor opción para representar datos que varían a medida que avanza el tiempo. En la Figura 4.12 se muestra el código HTML necesario para crear este gráfico.

```
<ngx-charts-line-chart
(window:resize)="onResize($event)"
[results]="vectorPruebaCharts"
[xAxisLabel]="'Tiempo'"
[legendTitle]="'Variables'"
[yAxisLabel]="'Valor'"
[legend]="true"
[showXAxisLabel]="true"
[showYAxisLabel]="true"
[xAxis]="true"
[yAxis]="true"
[autoScale]="true"
[timeline] = "true"
[view]="[700,400]">
</ngx-charts-line-chart>
```

Figura 4.12: Código HTML del gráfico

A continuación, se describen los elementos mostrados en la Figura 4.12 para una mejor comprensión.

- **window:resize:** esta característica se utiliza para alertar al sistema de que la ventana ha cambiado de tamaño. En este caso se usa para ejecutar el método 'onResize' el cual cambia el tamaño del gráfico y lo adapta al de la página.
- **results:** valores a mostrar en el gráfico.
- **xAxisLabel:** etiqueta para el eje de las X.
- **legendTitle:** título para la leyenda.
- **yAxisLabel:** etiqueta para el eje de las Y.
- **legend:** este elemento se configura como 'true' si se desea que se muestre una leyenda junto con el gráfico, y a 'false' en caso contrario.
- **showXAxisLabel:** este elemento se configura como 'true' si se desea que se muestre la etiqueta en el eje de las X, y a 'false' en caso contrario.
- **showYAxisLabel:** este elemento se configura como 'true' si se desea que se muestre la etiqueta en el eje de las Y, y a 'false' en caso contrario.
- **xAxis:** este elemento se configura como 'true' si se desea que se muestre el eje de las X, y a 'false' en caso contrario.

- **yAxis:** este elemento se configura como ‘true‘ si se desea que se muestre el eje de las Y, y a ‘false‘ en caso contrario.
- **autoScale:** este atributo cambia la escala del gráfico a la más adecuada en caso que se añadan datos nuevos.
- **timeline:** el timeline es una barra que aparece debajo del gráfico y sirve para acercar y alejar el gráfico a voluntad del usuario, para poder ver los datos más detalladamente.
- **view:** esta característica permite configurar el tamaño que tiene el gráfico por defecto.

Tras la implementación de este gráfico, se comprobó su correcto funcionamiento. En la Figura 4.13 se observa el resultado.

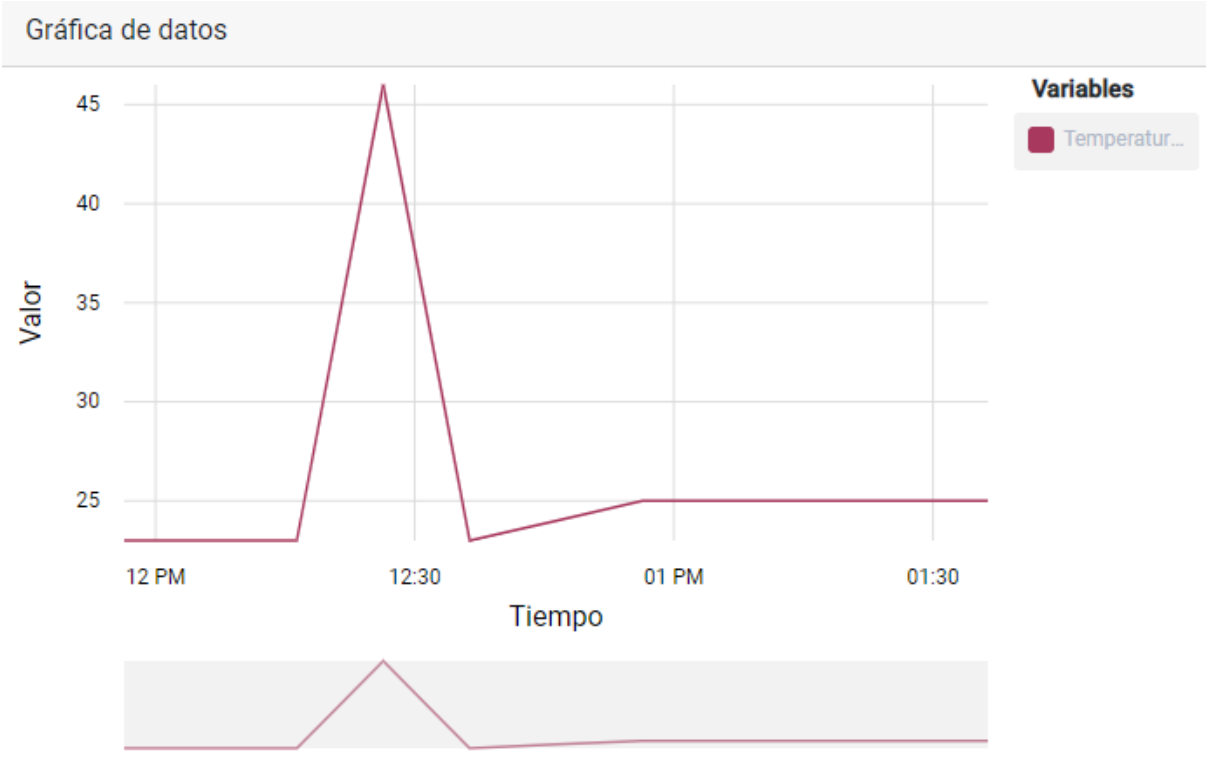

Figura 4.13: Gráfico de prueba

Tras esto, la siguiente tarea a implementar era un módulo que permitiese incluir en este gráfico otros conjuntos de datos, ya fueran del mismo SucreCore o de otro que el usuario tuviera en propiedad. De esta manera, se podrían comparar conjuntos de datos de forma gráfica y así ver las diferencias entre ellos de forma más rápida.

Así fue como se implementaron junto a la gráfica dos selectores iguales a los implementados anteriormente, junto con dos botones, uno para añadir la variable y otro para reiniciar el gráfico y devolverlo a su estado original, es decir mostrando únicamente el primer conjunto de datos seleccionado. El resultado de esta implementación se observa en la Figura 4.14.

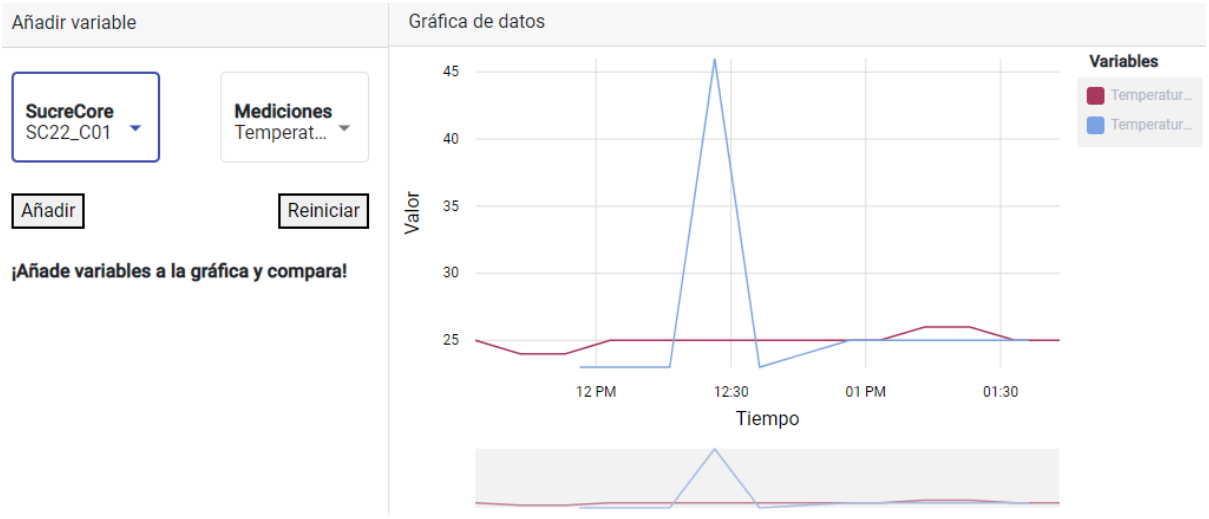

Figura 4.14: Gráfico con múltiples conjuntos de datos

Se implementaron varias medidas que no comprometieran el funcionamiento del sistema, como por ejemplo no dejar al usuario añadir dos veces el mismo conjunto de datos de datos al gráfico, o no mostrar en este apartado SucreCores que no tuvieran conjuntos de datos guardados para no causar problemas.

Tras haber implementado estas dos funcionalidades, se finalizó el sprint con todas las tareas listas y sin ninguna incidencia. El resultado del sprint se aprecia en la Figura 4.15.

Figura 4.15: Resultados del cuarto sprint

4.1.5. Quinto sprint. Blockly

En este sprint, se comunicó al alumno que ya se había establecido una nueva base de datos, por lo tanto se podrían ejecutar lecturas y escrituras de esta. El gestor de proyectos estableció un canal MQTT [19] para poder realizar escrituras en la base de datos desde los SucreCores, por lo tanto las tareas que se asignaron a este sprint fueron las que se ven en a continuación en la Figura 4.16.

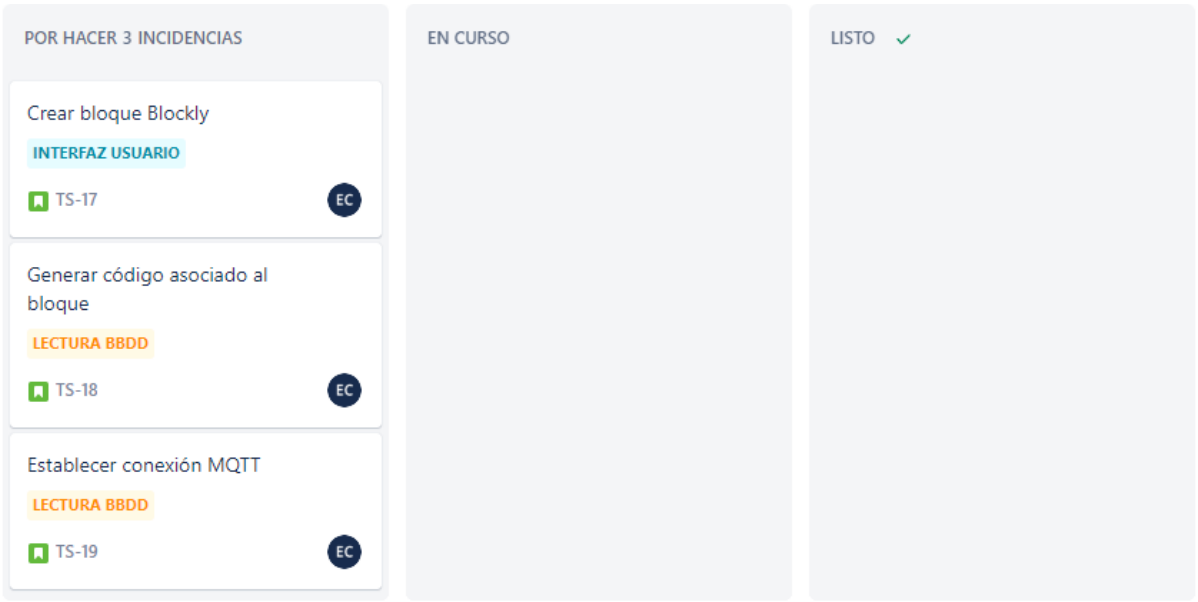

Figura 4.16: Tareas iniciales del quinto sprint

Como se ha comentado en el apartado de introducción, el proyecto SUCRE proporciona al usuario la posibilidad de programar a través de bloques. Esto significa, que para poder dar al usuario la opción de almacenar un dato en la base de datos, es necesario crear un bloque nuevo.

Blockly [7] es una librería creada por Google que permite mostrar fragmentos de código en forma de bloques visuales. Estos bloques se pueden unir arrastrándolos y soltándolos junto a otros para formar programas enteros. Esta biblioteca permite a los usuarios editar código de forma sencilla y sin necesidad de aprender lenguajes de programación.

Con el propósito de crear el nuevo bloque, se usó la herramienta desarrollada por Google 'Block factory', la cual ayuda a los desarrolladores a crear bloques de forma sencilla y se les proporciona el código en diferentes lenguajes de programación (JavaScript, Python, PHP, Lua, Dart) para poder construir el bloque desde su aplicación. En la Figura 4.17 se muestra la herramienta 'Block Factory', más concretamente la parte con la que el usuario interactúa para crear bloques.

Figura 4.17: Block Factory

A continuación, en la Figura 4.18 se puede observar el código JavaScript que proporciona Block Factory para que los desarrolladores puedan incluir el bloque creado en sus proyectos.

Block Definition: JavaScript

```
Blockly.Blocks['bloque_prueba'] = {
  init: function() {
    this.appendValueInput("coleccion")
        .setCheck(null)
        .appendField("Inserte variable aqui");
    this.setColour(230);
 this.setTooltip("");
 this.setHelpUrl("");
  }
};
```

Figura 4.18: Código generado en Block Factory

Gracias a esta herramienta, y cogiendo como referencia algunos de los bloques que ya fueron creados por la empresa, se implementó el bloque que aparece en la Figura 4.19. Como se puede ver en la Figura 4.19, hay diferentes secciones en las que se organizan los bloques, en este caso se ha colocado el bloque para el almacenamiento de datos en el apartado 'Online' ya que se establece comunicación con una base de datos externa.

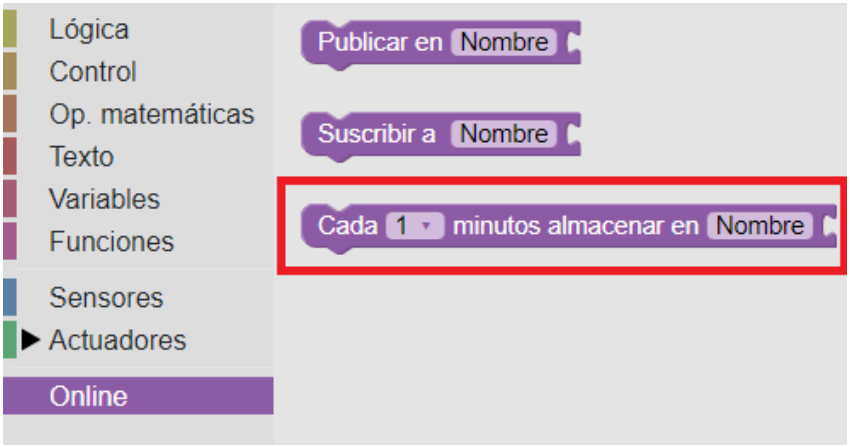

Figura 4.19: Bloque de almacenamiento de datos

En la Figura 4.19 se puede apreciar que en el bloque hay dos campos y luego una obertura en la parte derecha. En el primer campo, donde aparece un 1, se selecciona de entre los disponibles el periodo de tiempo que pasa entre cada guardado de datos. Este campo se ha establecido para no causar problemas de saturación en la base de datos. y los posibles valores a seleccionar son 1, 5, 10, 30 y 60 minutos. Por otro lado tenemos un campo de texto en el cual se escribirá el nombre del conjunto de datos en el cual se almacenarán los datos (temperatura, luz, distancia, etc). Por último, en la parte derecha tenemos un espacio para añadir otro bloque. En esta hendidura del bloque es donde se introduce la variable que se guardará en la base de datos. En la Figura 4.20 se puede observar el código mediante el cual se genera este bloque.

```
Blockly.Blocks['save_data'] = {
  /**
   * @this Blockly.Block
   */
  init: function() {
    this.appendValueInput("NAME")
        .setCheck(null)
        .appendField("Cada")
        .appendField(new Blockly.FieldDropdown([
          ["1", "60000"],
          ["5", "300000"],
          ["10", "600000"],
          ["30", "1800000"],
          ["60", "3600000"],
      ]), "tiempo")
        .appendField("minutos almacenar en")
        .appendField(new Blockly.FieldTextInput("Nombre"), "varName");
     this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(280);
 this.setTooltip("");
 this.setHelpUrl("");
  }
};
```

Figura 4.20: Código del bloque de persistencia de datos

Para realizar pruebas se creó un proyecto sencillo, el cual se puede apreciar en la Figura 4.21.

Figura 4.21: Proyecto de prueba

No obstante, esta parte del desarrollo únicamente genera el diseño del bloque. Hay que recordar que la verdadera acción que realiza el bloque es generar código para nuestro SucreCore. Es por eso que se debe mencionar que en este bloque, se generará el código necesario para realizar operaciones de escritura en la base de datos.

Como se ha mencionado al principio de este apartado, el gestor de proyectos estableció un canal MQTT para poder escribir los datos en la base de datos, no obstante antes de programar el comportamiento del bloque, se usó el entorno de desarrollo WEB Particle Build para realizar pruebas con pequeños programas a modo de test. Estos programas tenían como objetivo utilizar la librería MQTT de Particle para poder escribir valores en la base de datos InfluxDB. Cuando ya se consiguió una versión funcional, se procedió a trasladar este código a la parte en la aplicación

Angular en la que se implementa el código que genera el bloque. El concepto del funcionamiento del bloque se explica de forma visual en la Figura 4.22.

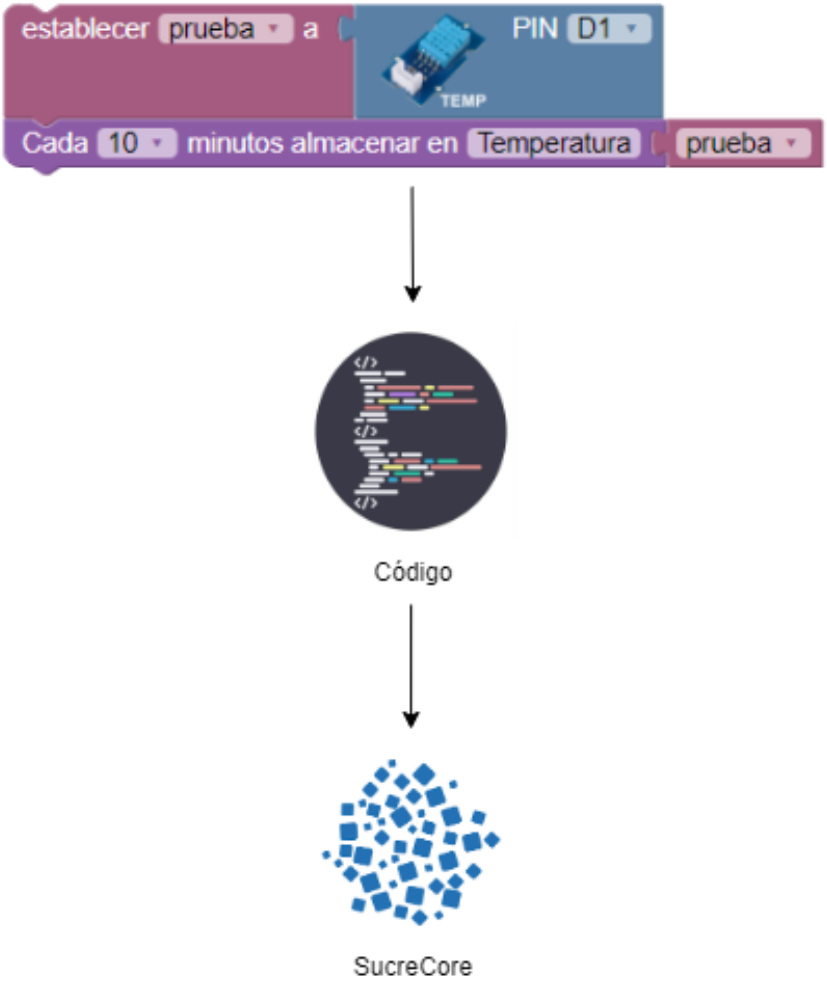

Figura 4.22: Funcionamiento programación por bloques

Al terminar el sprint, se realizó la reunión de revisión en la cual el supervisor dio el visto bueno, y por lo tanto se clasificaron las tareas del sprint como completadas. El resultado del sprint se muestra en la Figura 4.23.

Figura 4.23: Resultados del quinto sprint

4.1.6. Sexto sprint. Lecturas mediante una API REST

Tras haber implementado la parte que permite al usuario guardar los datos, el próximo objetivo era poder leer estos datos de la base de datos. Las tareas a cumplir durante este sprint se reflejan en la Figura 4.24.

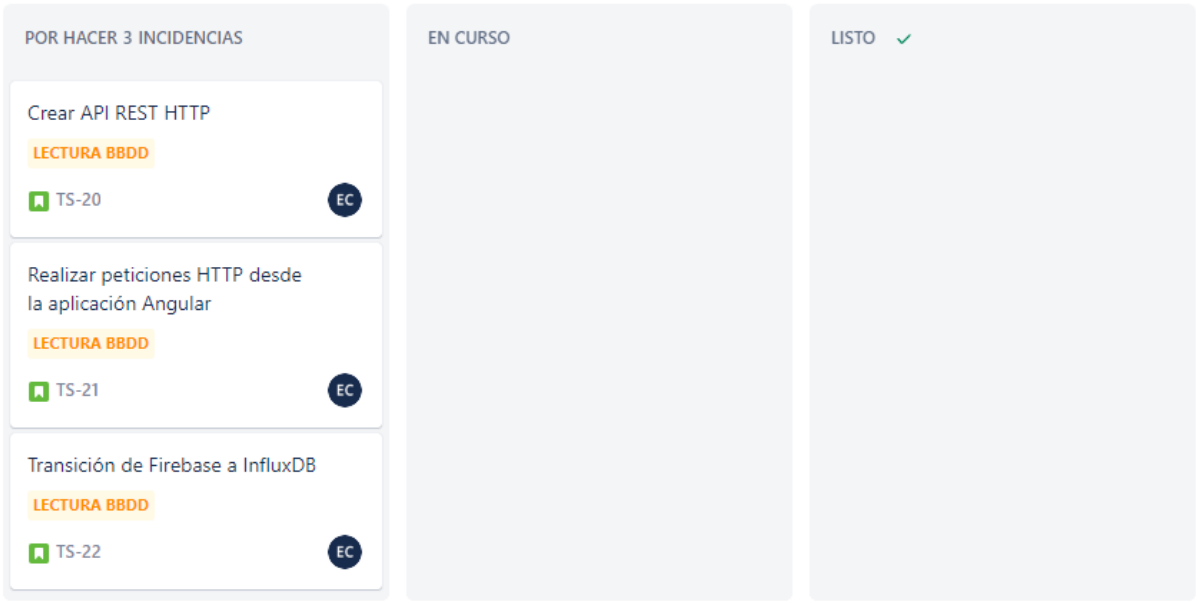

Figura 4.24: Tareas iniciales del sexto sprint

Primero se necesitó implementar una API REST que utilizara el protocolo HTTP, y que permitiese obtener información de InfluxDB. El gestor de proyectos sugirió utilizar Node-RED para crear esta API, ya que es la que él mismo utilizó para crear el canal MQTT que comunicaba la base de datos con los SucreCores.

Como ya se ha explicado en el apartado 3.2.4. Lecturas en InfluxDB, Node-RED [20] permite, entre otros, crear APIs de forma sencilla y a través de una aplicación web, utilizando nodos unidos por flujos. Node-RED proporciona a los desarrolladores bloques creados específicamente para realizar conexiones con determinadas aplicaciones, bases de datos,etc. Este es el caso de InfluxDB. En la Figura 4.25 se pueden ver los tres tipos de peticiones que se han implementado, todas del tipo GET, ya que la operación que se desea realizar sobre la base de datos es una lectura.

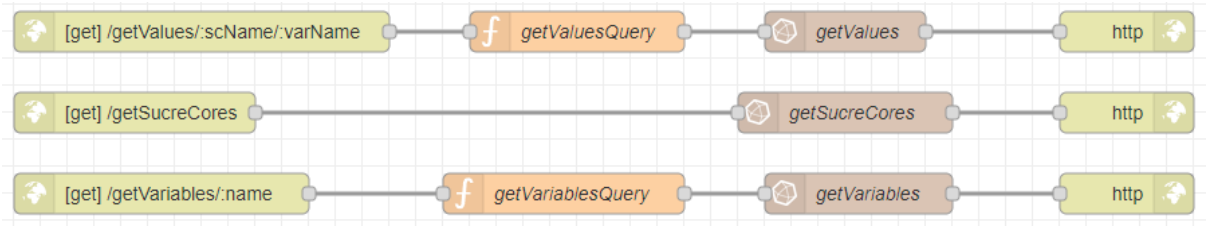

Figura 4.25: Node-RED API REST

Como podemos observar, se han implementado tres tipos de peticiones, una para obtener los valores de un conjunto de datos, otra para obtener los SucreCores almacenados en la base de datos y otra para obtener los conjuntos de datos enlazados a un SucreCore. Además, cabe mencionar que el modo en el que operan estas tres sentencias es casi idéntico para los tres casos.

Primero, se incluye un bloque, el cual simboliza una petición HTTP (HTTP request), y en el cual se incluyen los parámetros que se deben aportar en esta petición. A continuación aparecen para el primero y el tercer caso, dos nodos que simbolizan funciones, a través de las cuales se obtienen los parámetros pasados en las peticiones HTTP. En el segundo caso no se incluye este bloque debido a que no se pasan argumentos. Después se utiliza el bloque que realiza la consulta InfluxQL (un lenguaje casi idéntico a SQL) para obtener los datos solicitados, y finalmente se acaba el flujo con un nodo que representa la operación de respuesta, en la cual se proporcionan los datos recogidos.

Finalmente, la única tarea que quedaba pendiente tras la implementación de la API REST era realizar las peticiones desde la aplicación Angular cuando fueran necesarias y actualizar los elementos de la interfaz con los datos recogidos. Para esto, se creó un nuevo servicio en el paquete 'controller', a través del cual se realizarán estas peticiones HTTPS, y cuyos resultados se otorgarán a la interfaz para mostrarlos al usuario. El código que contiene este servicio es el que se muestra en la Figura 4.26.

```
export class InfluxService {
  totalAngularPackages;
  error;
  url: string = 'http://[redacted]';
  constructor(private http: HttpClient) { }

  getSucreCores(){
      return this.http.get<any>(this.url + 'getSucreCores');
  }

  getVariables(scName: String){
      return this.http.get<any>(this.url + 'getVariables/' + scName);
  }

  getValues(scName: String, varName: String){
      return this.http.get<any>(this.url + 'getValues/' + scName + '/' + varName);
  }
}
```

Figura 4.26: Peticiones HTTP desde Angular

Como se aprecia en la Figura 4.26, se ha censurado la IP por motivos de seguridad. Podemos observar que en esta clase se han implementado tres métodos, uno para cada operación de las comentadas anteriormente, y los parámetros se pasan como argumento, concretamente a través de la interfaz, la cual ejecuta uno de estos métodos cuando el usuario lo pide.

El código devuelve objetos de tipo *Observable*, los cuales contienen los datos recibidos en la respuesta de la petición. La forma de extraer estos datos es la que se aprecia en la Figura 4.27.

```
this.influx.getValues(scName,varName).subscribe((data => {
  data.forEach(lecture => {
    var date = new Date(lecture.time);
    vecDatos[0].series.unshift({value: lecture.value, name: date});
    ELEMENT_DATA.push({"valor": lecture.value, "fecha": date.toLocaleDateString(), "hora": date.toLocaleTimeString()});
  });
}));
```

Figura 4.27: Tratamiento de objetos *Observable*

Finalmente, se completaron las tareas indicadas, y tras esto se acabó la fase de implementación. El resultado del sexto y último sprint se muestra en la Figura 4.28.

Figura 4.28: Resultados del sexto sprint

4.2. Patrones y estrategias

En este sistema, para la parte del *frontend* se ha utilizado el patrón MVC [22], Modelo, Vista, Controlador. Este estilo de arquitectura de software separa la capa de datos (modelo), los componentes que forman la interfaz (vista) y la lógica de control (controlador) en tres componentes distintos.

El **modelo** es el componente que contiene la lógica de negocio, la representación de los datos que el sistema contiene y los mecanismos de persistencia. Entre las funciones del modelo encontramos el acceso a la capa de almacenamiento de datos y la definición de las reglas de negocio.

El **controlador** es el componente que actúa como vía de comunicación entre la vista y el modelo. Este gestiona la información entre las dos capas, realizando las transformaciones necesarias para que tanto la vista como el modelo puedan interpretar los datos correctamente y realizar las acciones pertinentes.

En la **vista** se implementan los mecanismos que muestran la información al usuario y los elementos que interactúan con este. La vista es responsable de mostrar los datos recibidos del modelo y mostrarlos al usuario y además también llamar al controlador cuando se necesite interactuar con el modelo debido a una interacción con el sistema por parte del usuario.

En la Figura 4.29 se muestra el funcionamiento del patrón MVC de forma más visual.

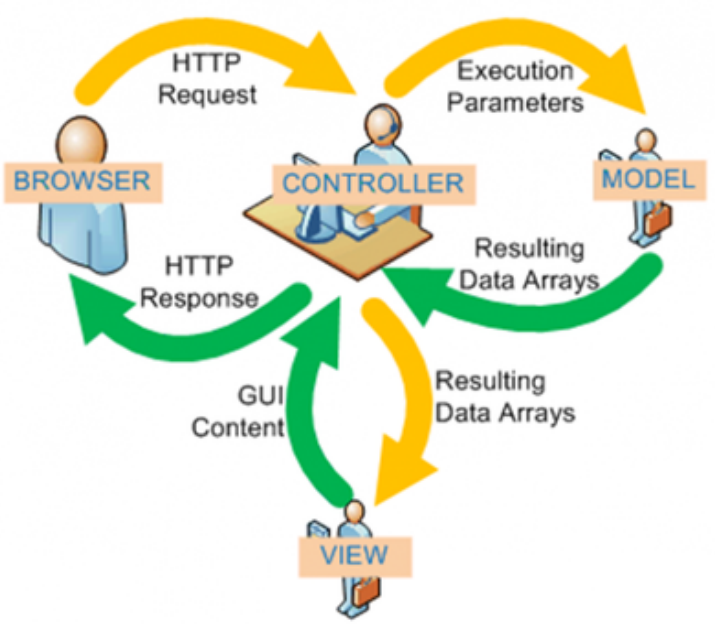

Figura 4.29: Patrón MVC

En el caso del proyecto SUCRE, como ya se ha mencionado anteriormente, para el *frontend* se ha implementado una aplicación Angular. En el caso de las aplicaciones construidas con este framework, es que su funcionamiento se basa en el uso de componentes. Esto quiere decir que, si tomamos como referencia el patrón MVC, en este caso la vista puede modificar el modelo, y viceversa. Aun así, el proyecto ha sido implementado siguiendo el patrón explicado para conseguir una mejor organización de los datos y el código.

4.3. Verificación y validación

En este apartado se comentarán los aspectos tenidos en cuenta para comprobar que el sistema funciona correctamente.

En primer lugar, cabe mencionar que la base de datos de Firebase con la que se hicieron las primeras pruebas era una diferente a la que se usa en la aplicación que está en funcionamiento actualmente. Es decir, para el proyecto SUCRE se crearon dos bases de datos, una para *testing* y otra para la versión operativa de la aplicación.

No se realizaron tests formales, ya que en la empresa no se trabajaba con estos. Aun así, en las reuniones realizadas tras cada sprint, tanto el gestor de proyectos como el supervisor, tras comprobar el progreso realizado en el sprint, si había alguna funcionalidad lista, la sometían a pruebas manuales de forma exhaustiva, y revisaban el código implementado para considerar si este era óptimo.

Además, para poner a prueba el sistema, se ejecutaban una serie de pasos estudiados por los

trabajadores de la empresa, para poner a prueba al sistema en diferentes escenarios, como por ejemplo comprobar que el sistema muestra un aviso cuando un SucreCore no tiene variables o que no se puede añadir dos veces la misma variable a la gráfica.

Por otro lado, para comprobar que el bloque Blockly de guardado de datos funcionaba correctamente, se accedía a la base de datos de forma manual y se comprobaba que la fecha y hora de las inserciones eran correctas, y que los datos coincidían con los del usuario que realizaba las pruebas (nombre del SucreCore, conjunto de datos y valor). Se probó también el impacto que el uso del bloque tenía sobre los SucreCores, es decir, se probaba que no se produjeran errores cuando el microcontrolador ejecutaba el programa, que no se detuviera durante la ejecución del mismo y/o que no aparecieran *bugs*.

Tras haber hecho un completo análisis de las funcionalidades implementadas y comprobar su éxito, el alumno realizaba un guardado de los cambios en la nube a través de Github.

Capítulo 5

Conclusiones

En este apartado se describirán las conclusiones desde diferentes puntos de vista.

En cuanto al ámbito formativo, considero que esta experiencia ha sido realmente fructífera, ya que por un lado, he aprendido cómo utilizar una gran variedad de tecnologías que no conocía anteriormente. He conocido nuevos aspectos como por ejemplo trabajar con Node-RED y crear una API REST lo cual me ha parecido realmente sencillo a la par que útil. Trabajar con nuevos protocolos de mensajería como MQTT me ha parecido realmente interesante, ya que este es un protocolo muy simple y fácil de entender.

En cuanto al ámbito profesional, puedo decir que durante la estancia en prácticas he trabajado codo con codo con profesionales de la informática, cosa que no había vivido anteriormente. Además también he notado que normalmente durante el grado siempre realizábamos trabajos desde cero hasta el final, es decir, realizábamos aplicaciones o trabajos desde el principio hasta que implementábamos todas las funcionalidades requeridas. No obstante, durante la estancia me he adaptado a un proyecto que ya estaba en funcionamiento y he aprendido a medida que realizaba el trabajo, cosa que da también un toque de realidad a la estancia en prácticas ya que en muchos casos cuando se consigue un trabajo uno se tiene que adaptar a un proyecto ya empezado.

Finalmente en cuanto a mi experiencia personal, únicamente puedo decir que me he sentido, por una parte agradecido con los trabajadores de la empresa, ya que en muchas ocasiones he necesitado ayuda y siempre me la han proporcionado de inmediato y con mucho entusiasmo. Además he de decir que a pesar de ser un estudiante en prácticas, se me ha tratado con mucho respeto y se escuchaban y valoraban mis opiniones, lo cual me ha hecho sentir como uno más.

Bibliografía

Al finalizar la realización de la memoria, se comprobaron todos los enlaces de nuevo para comprobar que todavía estaban en funcionamiento. Es por eso que para todos los casos, Se considera que la fecha de consulta es el día 22 de junio de 2022.

[1] Mouser Electronics, Inc. Particle Argon IoT Development Board. https://www.mouser.es/new/particle/particle-argon-dev-board/.

[2] Google. Angular. https://angular.io.

[3] Google. Firebase. https://firebase.google.com/docs/guides.

[4] Google. Firebase Authentication. https://firebase.google.com/docs/auth.

[5] Google. Firebase Cloud Firestore. https://firebase.google.com/docs/firestore.

[6] Google. Firebase Hosting. https://firebase.google.com/docs/hosting.

[7] Google. Blockly. https://developers.google.com/blockly.

[8] Particle Industries, Inc. Particle Cloud. https://www.particle.io/platform/particle-cloud/.

[9] InfluxData Inc. InfluxDB: Open Source Time Series Database. https://firebase.google.com/docs/hosting.

[10] Swimlane. Ngx-charts. https://swimlane.gitbook.io/ngx-charts/.

[11] Wikipedia. Node-RED. https://en.wikipedia.org/wiki/Node-RED.

[12] Atlassian. Scrum: que es, cómo funciona y por qué es excelente. https://www.atlassian.com/es/agile/scrum.

[13] Wikipedia. Scrum (desarrollo de software). https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)#/media/Archivo:Scrumm.PNG

[14] Hays plc. Hays plc. https://www.hays.es/documents/63345/4314146/GUIA+DEL+MERCADO+LABORAL+DE+HAYS+2020+-+Online.pdf.

[15] Amazon Web Services, Inc. AWS Marketplace: InfluxDB Cloud. https://aws.amazon.com/marketplace/pp/prodview-4e7raoxoxsl4y

[16] Miguel Alejandro Esteban Ordoñez. Qué es Gherkin y por qué es necesario. https://openwebinars.net/blog/que-es-gherkin/

[17] Wikipedia. Diagrama de casos de uso. https://es.wikipedia.org/wiki/Diagrama_de_casos_de_uso.

[18] Marta Benedet. Arquitectura serverless: qué es y qué no es. https://blog.mdcloud.es/arquitectura-serverless/.

[19] MQTT.org. MQTT - The Standard for IoT Messaging. https://mqtt.org/.

[20] OpenJS Foundation. Node-RED. https://nodered.org/.

[21] jersey99. httpsclient-particle. https://github.com/glowfishapi/httpsclient-particle.

[22] Universidad de Alicante. Modelo vista controlador (MVC). https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html#:~:text=Modelo%20Vista%20Controlador%20(MVC)%20es,control%20en%20tres%20componentes%20distintos..