Grado en Ingeniería Informática

Trabajo de Final de Grado

Creación de un conjunto de bloques de programación visual para la formación de niños en pensamiento computacional

Autor:
María del Carmen López Girona

Supervisor:
Juan Camilo Gómez Esguerra
Tutor académico:
María del Mar Marcos López

Fecha de lectura: 13 de Septiembre de 2023
Curso académico 2022/2023

Resumen

En este TFG se ha realizado una ampliación del proyecto Sucre (Sense yoUr Content and REact). Concretamente, se crea una nueva versión de Sucre4STEM adaptada a alumnos de edad de educación primaria.

La nueva versión consiste en la creación de un conjunto de bloques de programación adaptado a las necesidades del usuario. También se han realizado mejoras sobre la aplicación tanto a nivel estético como a nivel funcional, entre estas mejoras se encuentra la creación de una colección de bloques musicales.

El proyecto ha sido desarrollado en la empresa UBIK Geospatial Solutions durante la estancia en prácticas.

Palabras clave

Blockly, Particle, Angular, firebase, programación visual.

Keywords

Blockly, Particle, Angular, firebase, visual programming.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este apartado se explicará el contexto del proyecto, así como su motivación y una descripción del mismo. También se mencionarán los objetivos y alcance y se realizará la planificación que se llevará a cabo.

1.1. Contexto del proyecto

La empresa donde se han realizado las prácticas y desarrollado el proyecto es Ubik Geospatial Solution, cuyo logotipo se muestra en la Figura 1.1. Ubik fue creada por Joaquín Huerta, Ana Sanchís y Michael Gould, investigadores del grupo de investigación GEOTEC (Geospatial Technologies Research Group), cuenta con más de 15 años de experiencia en proyectos del ámbito de las Tecnologías de la Información y las Comunicaciones (TIC). La empresa está ubicada en el edificio Espaitec II, en la Universitat Jaume I y se dedica a diseñar y construir soluciones geográficas innovadoras, funcionales y económicas. Proporciona al mercado servicios sobre programación TIC donde implementa aplicaciones tanto móviles como web para la integración de información geoespacial.

Figura 1.1: Logo Ubik Geospatial Solutions SL.

Entre los servicios que ofrece destacan los siguientes:

- Desarrollo de aplicaciones móviles basadas en la localización.
- Desarrollo de aplicaciones de cartografía web.
- Infraestructuras de Datos Espaciales (IDE).
- Desarrollo de aplicaciones Smart City.
- Integración de información de las redes sociales con otros sistemas de información.

1.2. Estado inicial del proyecto

Durante la estancia se ha trabajado sobre el proyecto Sucre (Sense yoUr Context and REact), un proyecto que la empresa inició en 2016 ([2]) y cuyo objetivo es el fomento de vocaciones científicas, la promoción del pensamiento computacional y la programación.

Debido a la necesidad de adecuar los contenidos didácticos y tecnológicos por grupos de edad, Sucre se encuentra dividido en dos versiones: Sucre4KIDS ([3], [8], [10], [11], [12], [9]) y Sucre4STEM ([4], [13], [14]), la primera de ellas pensada para alumnos de la etapa de Educación Primaria y la segunda para alumnos de educación secundaria (ESO, Bachiller y FP de grado medio).

Sucre4KIDS, consiste en un microcontrolador con una pantalla y un lector NFC. Al microcontrolador se le montan los sensores y los actuadores y al lector NFC se le pasan una serie de tarjetas para generar condiciones, también hay una tarjeta que indica que el programa debe iniciar.

Sucre4STEM consiste en un microcontrolador que se conecta a internet y al que se conectan sensores y actuadores. El microcontrolador se programa por medio de una aplicación de programación por bloques, la cual devuelve al dispositivo las señales necesarias para que este active los actuadores pertinentes en función de los valores recogidos por los sensores. Durante la estancia se trabajará sobre la versión de Sucre4STEM.

Ambas versiones funcionan de modo similar, se conectan sensores y actuadores al microcontrolador para medir el entorno y responder con ciertas acciones respectivamente. La diferencia fundamental es que en Sucre4KIDS, el microcontrolador se programa por NFC mediante tarjetas, mientras que en Sucre4STEM se utiliza una aplicación que utiliza programación por bloques.

La versión de Sucre4STEM, al ser programada por el usuario, permite un abanico más amplio de opciones para sensores y actuadores, mientras que la versión de Sucre4KIDS los sensores tienen umbrales predefinidos, de modo que lo que se mide es si se cumplen o no dichos umbrales. En resumen, los sensores dan respuestas binarias. Respecto a los actuadores, la versión de Sucre4STEM, permite seleccionar si están activos o no, y el resto de parámetros, si los hubiera En la versión de Sucre4KIDS cada tarjeta representa un actuador en un estado predefinido, por ejemplo un led verde, en lugar de dar libertad al usuario para escoger un color cualquiera de entre la escala disponible.

Durante la estancia en prácticas, se va a crear una nueva versión dentro de Sucre4STEM. La idea es que en esta nueva versión, por medio de programación por bloques, se simulen las tarjetas de la versión de Sucre4KIDS. Para ello se creará un nuevo conjunto de bloques, que se incluirán en la nueva versión, básica. Los bloques actuales se englobarán la versión avanzada. El llamarlas versión básica y avanzada, es por el hecho de que la nueva colección de bloques está pensada para alumnos de edades más tempranas.

Los recursos didácticos de Sucre4STEM se encuentran en un maletín como el que se muestra en la Figura 1.2, conocido como SucreKit, el cual contiene un microcontrolador Argon, denominado SucreCore y un conjunto de 8 sensores y 6 actuadores.

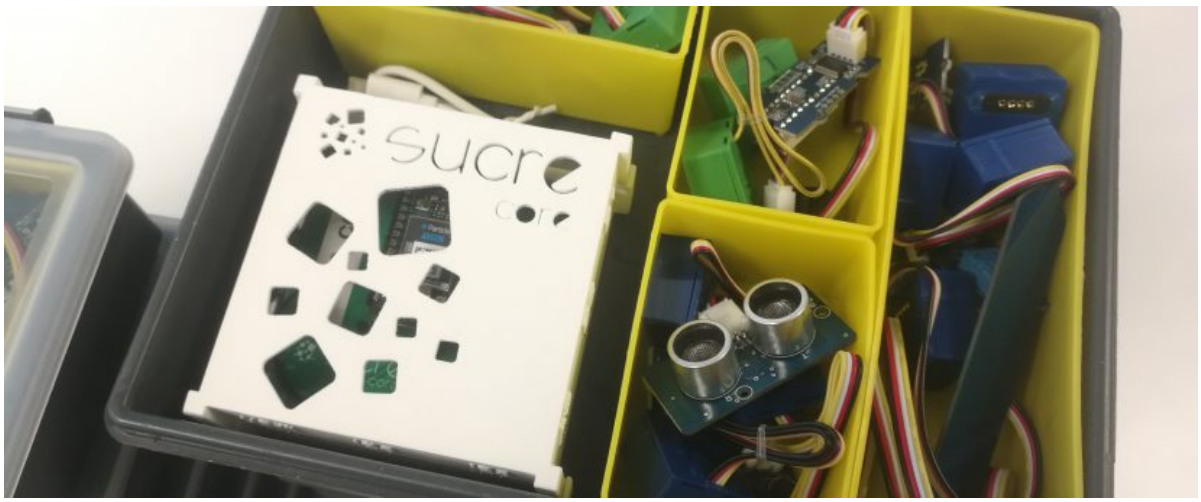


Figura 1.2: Maletín Sucre4STEM

Las funcionalidades con las que contaba el proyecto en su estado original son las siguientes:

- Inicio de sesión.
- Crear un nuevo proyecto.
- Vincular un dispositivo.
- Cambiar la red Wi-Fi del SucreCore.
- Realizar cambios en los proyectos existentes.
- En el caso de los superusuarios:
  - Modificar nombre de usuario.
  - Asignación de permisos para utilizar SucreCores a usuarios.
  - Eliminar proyectos.
  - Crear o eliminar centros educativos.
  - Asignar o eliminar usuarios de un centro educativo.

1.3. Motivación del proyecto

El proyecto Sucre, en su estado actual, permite a los usuarios crear proyectos de manera sencilla utilizando la programación por bloques ([6]).

Como se ha mencionado anteriormente, Sucre nació con el objetivo de crear vocaciones computacionales. En la actualidad se utiliza en centros de educación secundaria como una opción

innovadora para realizar experimentos y estudios, mientras los alumnos aprenden a programar de manera sencilla haciendo uso de bloques de programación.

No obstante, esta versión no está preparada para poder realizar experimentos en centros educativos de educación primaria, debido a la complejidad que podría suponer el uso de la aplicación para niños menores a 12 años.

Tras el éxito de la primera versión, y teniendo en cuenta que los niños muestran un creciente interés por la tecnología desde edades cada vez más tempranas, surge la idea de llegar a los alumnos de edad de primaria con Sucre4STEM. De esta idea nació este proyecto, que consiste en un versionado, adaptado a los conocimientos y necesidades del nuevo público.

1.4. Objetivos del proyecto

Durante la estancia en prácticas se ha desarrollado una nueva versión de Sucre4STEM, de modo que se adapte a las necesidades de los alumnos de la etapa de educación primaria. Esta nueva versión se denominará *versión básica* de Sucre4STEM. También se realizarán algunas modificaciones en la versión actual de Sucre4STEM. Por tanto, este proyecto tiene los siguientes objetivos y subobjetivos:

- O1: Mejoras de Sucre4STEM:
  - O1.S1.: Corrección de errores de Sucre4STEM. La actual versión deja ensamblar bloques cuyo tipo de datos no se corresponde en la entrada/salida, por lo que el código resultante es incorrecto.
  - O1.S2: Modificación del código de los sensores cuyos valores son complicados de ver en tiempo real por su constante cambio. El objetivo es crear un nuevo bloque, el cual devuelva la medición por intervalos de tiempo, de modo que el valor devuelto pueda ser visible.
- O2: Adaptación de los bloques actuales a las necesidades de niños de edades más tempranas:
  - O2.S1: Creación de un nuevo conjunto de bloques, para obtener bloques de sensores con un comportamiento similar al obtenido con las tarjetas que representan las diferentes condiciones para los sensores en la versión de Sucre4KIDS.
  - O2.S2: Creación de un nuevo conjunto de bloques que realicen tareas sencillas para los actuadores, con estados fijos, actuando de igual modo que las tarjetas de actuadores en la versión Sucre4KIDS.
- O3: Mejoras en la aplicación, comunes a ambas versiones:
  - O3.S1: Creación de una opción que almacene el estado actual del montaje de bloques y su código asociado, para su posterior reutilización.
  - O3.S2: Creación de un menú que permita seleccionar la versión de Sucre que se va a

utilizar.

- O3.S3: Creación de un conjunto de bloques con las notas y figuras musicales básicas para ser interpretados por el zumbador (se crearán dos conjuntos diferentes de bloques, uno adaptado a cada una de las versiones).

1.5. Tecnologías usadas en el proyecto

En este apartado se comentan las tecnologías utilizadas tanto para el frontend como para el backend del proyecto Sucre.

En el frontend se utiliza Angular ([5]), que es un framework para el desarrollo de aplicaciones web tanto en versión móvil como de escritorio, es de código abierto y está mantenido por Google. Utiliza diversos lenguajes de programación, que mayoritariamente son JavaScript, HTML y TypeScript.

En el backend se utiliza Firebase ([7]), una plataforma digital gestionada por Google para facilitar el desarrollo de aplicaciones web y móviles. En el proyecto se usa Firebase Auth para autenticar a los usuarios en el inicio de sesión, Firebase Hosting como servicio de hosting para la aplicación y Firebase Cloud Firestores, para almacenar tanto los datos de los usuarios como sus SucreCores, proyectos, etc. También se utiliza una base de datos InfluxDB, donde se almacenan los valores medidos por los dispositivos SucreCore. Está implementado a través de AmazonWebServices ([1]).

El lenguaje C también está presente, pues la aplicación consiste en realidad en un entorno de programación visual, el cual genera código C a partir de los bloques ensamblados en el proyecto. Así pues, todas las funciones relativas al tratamiento de los datos recogidos por los sensores o devueltos por los actuadores, así como las necesarias para interactuar con el SucreCore, deben estar escritas en C.

Para cargar los programas a los dispositivos SucreCore y mostrar los resultados que provocan su ejecución en los actuadores se utiliza Particle Cloud. Es un servicio creado por la empresa Particle, que permite a los dispositivos conectarse a Internet y por tanto, enviar los programas a los dispositivos.

Para la creación de los bloques de programación se ha utilizado una librería de Google llamada Blockly. La librería Blockly proporciona un editor de código visual, por lo que deja la sintaxis de los programas en segundo plano, permitiendo así generar programas complejos de manera sencilla.

Para la creación de gráficas se utiliza la librería Ngg-charts, que permite crear gráficas de modo sencillo dentro de aplicaciones Angular.

1.6. Estructura de la memoria

En primer lugar, en el capítulo 2 hace referencia a la planificación del proyecto, en él se muestra una descripción más detallada del proyecto, que comprende la metodología seguida, la planificación, el seguimiento, la estimación de recursos que implica y los riesgos que pueden surgir en

las distintas fases.

Seguidamente, en el capítulo 3 se habla del análisis del proyecto, aquí se engloban la definición de requisitos y su análisis.

A continuación, en el capítulo 4 se trata el diseño de la aplicación, así como el diseño de Blockly y la definición de los datos.

Después, el capítulo 5 se centra en la implementación de dicha aplicación, aquí se incluirán también las pruebas de validación.

En el capítulo 6 se muestran los resultados obtenidos en los distintos bloques de programación desarrollados y las interfaces gráficas.

Por último, el documento concluye con un capítulo en que se redactan las conclusiones obtenidas una vez finalizado el proyecto.

Capítulo 2

Planificación del proyecto

En este capítulo se presenta la planificación seguida en el desarrollo del proyecto. Primero se va a describir la metodología que se ha usado, a continuación se expone la planificación de las tareas de las que consta el proyecto, se realiza una estimación de los recursos necesarios y se calculan los costes del proyecto. En el último apartado del capítulo de describirán los riesgos del proyecto.

2.1. Metodología

En cuanto a la metodología seguida para llevar a cabo el proyecto se centra en el ciclo de vida clásico, también llamado modelo en cascada. Esta es una de las primeras metodologías que se propusieron y consiste en un enfoque secuencial y lineal para el desarrollo de software.

Su punto crítico es la necesidad de tener una correcta definición de requisitos al inicio del proyecto, lo cuál no siempre es posible. Además en los proyectos no es habitual que los pasos a seguir sigan un ciclo estrictamente secuencial, sino que se producen iteraciones y surgen problemas. Tampoco ayuda que los resultados no sean visibles hasta las últimas etapas.

Para crear una aplicación desde cero sería una práctica muy desaconsejada el uso de esta metodología. Como en este caso se van a hacer modificaciones sobre un producto de software ya existente, los requisitos son fácilmente definibles, por lo que puede resultar sencillo llevar esta metodología a cabo.

El ciclo de cascada comprende las siguientes etapas:

- Definición y análisis de requisitos.
- Diseño.
- Programación y prueba individual.
- Prueba del sistema.

2.2. Planificación inicial

En este apartado se describe el proceso de planificación necesaria para realizar el proyecto, incluyendo una estimación del tiempo invertido en realizar el proyecto. Cabe destacar que los tiempos podrían sufrir modificaciones, ya que esto es una primera planificación temporal y se puede retrasar o pueden surgir problemas como los riesgos que se citarán más adelante.

La planificación inicial cuenta con las siguientes fases:

- **Definición y análisis de requisitos** (45h). Se dedicará a leer documentación sobre programación en bloques, estudiar el uso de la librería blockly, documentos sobre Angular, Lua, microcontroladores, etc. Se estudiará el proyecto Sucre, se realizará la preparación del entorno de trabajo, se mantendrán reuniones con el supervisor para aclarar las dudas respecto al proyecto. Se estudiarán y planificarán los cambios necesarios. En esta parte se incluirá también la elaboración de la definición de requisitos y objetivos finales.

- **Diseño** (25h). Se realizará el prototipado de los diferentes conjuntos de bloques que se van a crear a lo largo del proyecto, así como las nuevas interfaces para seguir una guía de diseño. Estos prototipos se revisarán por el supervisor para que todo esté listo para el desarrollo. Dichos prototipos pueden sufrir modificaciones posteriores durante el desarrollo. El prototipado en este proyecto será un periodo breve teniendo en cuenta todos los conjuntos que hay que prototipar. El motivo de este breve periodo es que los nuevos bloques e interfaces van a seguir muy de cerca la línea de los ya existentes.

- **Programación y prueba individual** (205h).

  - **Modificaciones de versión avanzada** (15h).

    - Corrección de errores de Sucre4STEM y prueba (5h).

    - Modificación del código de sensores que no pueden observarse en tiempo real y pruebas. (10h)

  - **Creación de versión básica** (90h).

    - Creación de colección de bloques nuevos para los sensores y prueba (45h).

    - Creación de colección de bloques nuevos para los actuadores y prueba (45h).

  - **Mejoras aplicación** (100h).

    - Creación de colecciones para reproducir música por medio del zumbador (45h).

    - Creación de variables y menú donde el usuario escoja qué versión de Sucre utilizará en sus proyectos y prueba (25h).

    - Creación de opción para almacenar y recargar el espacio de trabajo (30h).

- **Prueba del sistema** (25h).

En la Figura 2.1 se muestra la planificación en el diagrama de Gantt

Id	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Pred
1		**Definición y análisis de requisitos**	**45 horas**	**lun 20/03/23**	**lun 03/04/23**	
2		Estudio de las tecnologías utilizadas	9 horas	lun 20/03/23	mar 21/03/23	
3		Estudio de programación por bloques	8 horas	mar 21/03/23	vie 24/03/23	2
4		Preparación del entorno de trabajo	5 horas	vie 24/03/23	vie 24/03/23	3
5		Estudio del proyecto Sucre	13 horas	lun 27/03/23	jue 30/03/23	4
6		Definición y análisis de requisitos	10 horas	jue 30/03/23	lun 03/04/23	5
7		**Diseño**	**25 horas**	**lun 03/04/23**	**vie 14/04/23**	1
8		Diseño entrada/salida bloques actuales	6 horas	lun 03/04/23	mar 04/04/23	1
9		Diseño apariencia y comportamiento bloques nuevos	19 horas	mar 04/04/23	vie 14/04/23	8
10		**Programación y pruebas individuales**	**205 horas**	**vie 14/04/23**	**mar 06/06/23**	
11		**Modificaciones Sucre4STEM**	**15 horas**	**vie 14/04/23**	**jue 20/04/23**	
12		Corrección de errores	5 horas	vie 14/04/23	lun 17/04/23	9
13		Modificación sensores	10 horas	lun 17/04/23	jue 20/04/23	12
14		**Adaptación Sucre4Kids**	**90 horas**	**jue 20/04/23**	**vie 12/05/23**	
15		Creación bloques sensores y pruebas	45 horas	jue 20/04/23	mar 02/05/23	13
16		Creación bloques actuadores y pruebas	45 horas	mar 02/05/23	vie 12/05/23	15
17		**Mejoras conjuntas**	**100 horas**	**vie 12/05/23**	**mar 06/06/2**	
18		Colección de bloques musicales	45 horas	vie 12/05/23	mié 24/05/23	16
19		Interfaz panel modalidad Sucre	25 horas	mié 24/05/23	mar 30/05/23	18
20		Funcionalidad guardar conjunto de bloques	30 horas	mar 30/05/23	mar 06/06/23	19
21		**Prueba del sistema**	**25 horas**	**mar 06/06/2**	**mar 13/06/2**	
22		Pruebas	25 horas	mar 06/06/2	mar 13/06/2	20

Figura 2.1: Planificación detallada

En la Figura 2.2 se muestra el gráfico del diagrama de Gantt

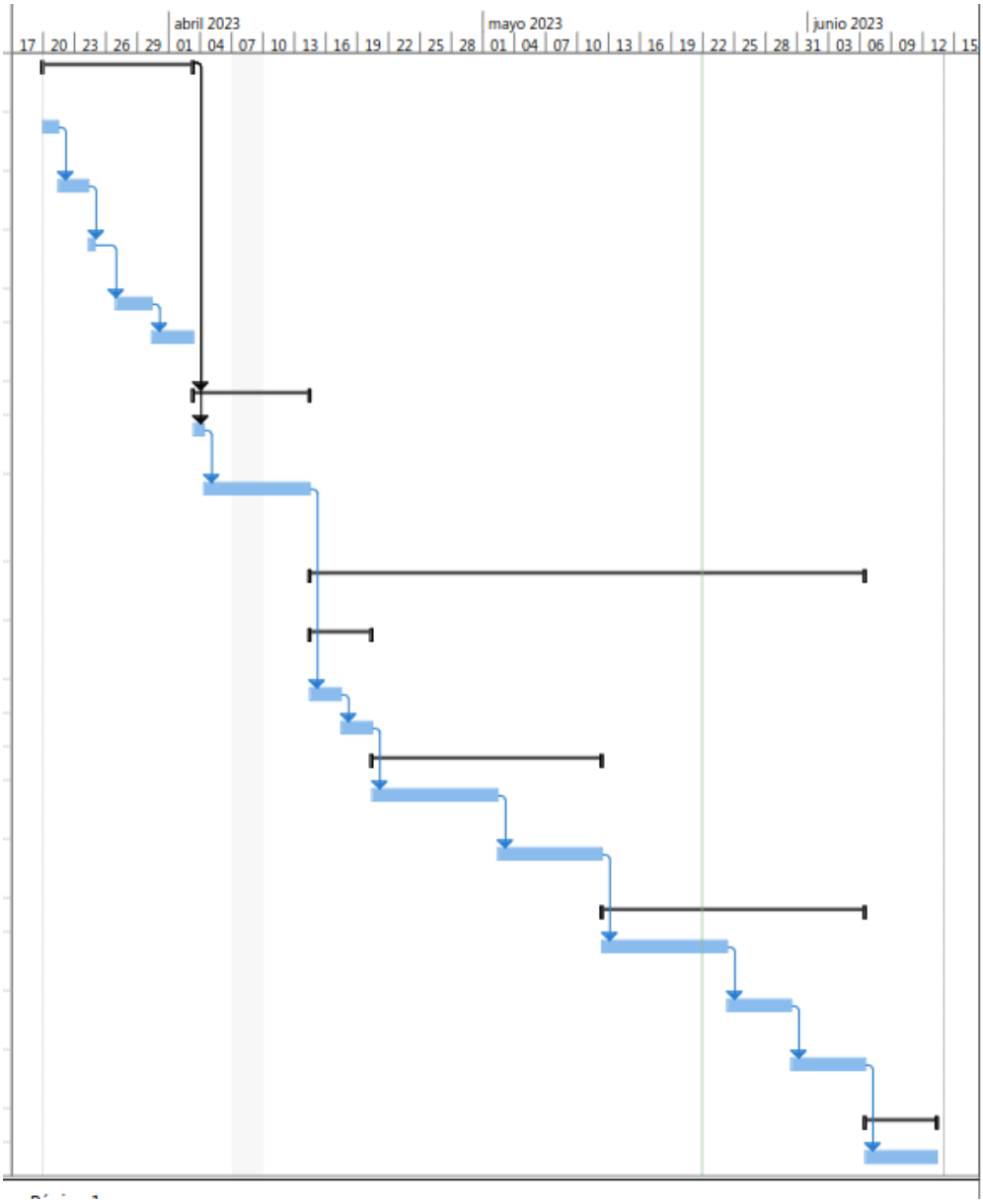


Figura 2.2: Diagrama de Gantt

2.3. Seguimiento del proyecto

Para realizar el seguimiento del proyecto se han llevado a cabo informes quincenales con la tutora y se han mantenido reuniones semanales con el supervisor de la empresa de prácticas. Además en la empresa he tenido un segundo supervisor(Sergi Trilles), con el que he mantenido reuniones diarias sobre el trabajo realizado el día anterior y una previsión del trabajo a realizar ese mismo día.

Siguiendo con la planificación, en la primera quincena se dedicó el tiempo al estudio del proyecto y la documentación sobre la programación por bloques, la cual desconocía hasta el momento. También se dedicó parte del tiempo al estudio de las librerías Blockly y Particle. Se realizó la instalación de todo el software necesario para poder ejecutar SUCRE4STEM. Una vez estudiado el proyecto en profundidad se llevó a cabo la definición y análisis de requisitos. Los últimos días de la quincena se dedicaron también a empezar el prototipado del nuevo conjunto de bloques.

En la segunda quincena, tal y como estaba planificado se prosiguió con las tareas de prototipado tanto del conjunto de bloques, como de los bloques de la versión original que requerían modificaciones. Hacia el final de la quincena se empezó el desarrollo de los sensores de la versión avanzada que requerían modificaciones o mejoras. Estos son: el sensor de ruido y los tipos de datos de salida de los sensores.

En la tercera quincena se continuó con la tarea de desarrollo, en este caso se realizó el desarrollo del nuevo conjunto de bloques de sensores para la versión básica. Una vez creado el conjunto de bloques se realizaron las pruebas pertinentes y las correcciones de errores hasta conseguir el resultado deseado.

Durante la cuarta quincena, se desarrolló el conjunto de bloques perteneciente a los actuadores de la versión básica. Se realizaron las pruebas y correcciones necesarias y se inició el proceso de desarrollo de la colección de bloques musicales.

Durante la quinta quincena, se continuó con el desarrollo de los bloques de música. Posteriormente se realizaron las tareas de prueba y corrección de errores. Los últimos días de la quincena se dedicaron a iniciar las mejoras visuales de la aplicación, separando ambas versiones que, hasta la fecha, eran conjuntos de bloques en un menú común.

Finalmente, en la sexta quincena, se terminó la creación y diferenciación tanto visual como funcional de las dos versiones de Sucre4STEM. Se crearon las funcionalidades de guardar y cargar el estado de un proyecto, descargar pdf con gráficas y documentos csv con tablas de los datos almacenados en la base de datos, etc. Por último, se realizaron las pruebas del sistema necesarias.

2.4. Estimación de recursos

En este apartado se muestra una estimación de los recursos necesarios. Podemos dividir los recursos en tres tipos: humanos, software y hardware.

- **Recursos humanos** La empresa Ubik, ha preferido en todo momento mantener sus salarios en privado, por tanto se va a realizar una estimación en base al salario medio.

- Programador sénior, en este caso el supervisor del proyecto, Juan Camilo Gómez Esguerra. Teniendo en cuenta una implicación en el proyecto de 40h semanales, tendría un salario anual de 40000€, por tanto para una implicación semanal de unas 15h aproximadamente y que la duración del proyecto son 3 meses, estamos hablando de un coste de 3750€.

- Programadora junior, en este caso la alumna Carmen López Girona. Con una implicación a tiempo completo en el proyecto, es decir, 40h semanales, el salario medio anual sería de unos 25000€. Teniendo en cuenta una implicación de unas 25h semanales durante 3 meses, el coste asciende a 3906.25€.

El coste de los recursos humanos supone 7656.25€, pero a este precio deberíamos sumarle la Seguridad Social, que suele costar al rededor de un 25 %, lo cuál significaría un 1914.06€. Por tanto, el coste total invertido en recursos humanos para este proyecto pasa a ser de 9570.31€.

- **Recursos hardware**. Para el cálculo de los componentes hardware debemos tener en cuenta el equipo necesario para la programación y los componentes hardware de un SucreKit.

  Respecto al equipo para la programación se ha contado con un ratón Ergonómico Inalámbrico Perixx 719 para zurdos y un ordenador Slimbook Eclipse, ambos propiedad de la alumna, cuyo precio en el mercado se sitúa en 19.99€ para el ratón y 1289.95€ para el ordenador.

  La empresa me ofreció la posibilidad de trabajar con uno de sus equipos, pero hubieron algunos problemas con la conexión por cable y finalmente decidí hacer uso de mi propio ordenador por motivos de comodidad.

  Debido a que el plazo de amortización de un portátil se sitúa en torno a unos 5 años, se calcula que el uso durante los 3 meses supone un coste de 64.50€.

  A continuación se van a desglosar los costes de los componentes hardware de un SucreKit, los cuáles han sido enumerados anteriormente, en el apartado *1.2 Estado inicial del proyecto*.

  - Seeed Studio Grove Chainable RGB Led V2.0, 5.07€.

  - Grove Ultrasonic Distance Sensor, 3.30€.

  - Grove 4-Digit Display, 5€.

  - Grove Temperature Humidity Sensor (DHT11), 5€.

  - Grove Rotary Angle Sensor, 2.46€.

  - Grove Button, 1.61€.

  - Grove Light Sensor v1.2, 2.46€.

- Grove 4-Digit Display, 5€.
- Grove White LED, 1.61€.
- Grove Sound Sensor, 4.15€.
- Grove Capacitive Moisture Sensor (Corrosion Resistant), 5.04€.
- Grove NEODYMIUM DISC MAGNET 3X1MM (PK50), 4.5€.
- Particle Argon, 21.99€.
- Caja organizadora, 3.69.

El coste de un SucreKit completo supone 70.88€. Puesto que tanto el ratón como el SucreKit se ha comprado para este proyecto, por lo que se contabiliza el total de su precio, los recursos hardware ascienden a un total de 155.37€.

- **Recursos software**
  - Visual Studio Code para la programación de la aplicación Angular y del microcontrolador.
  - Firebase, como herramienta de autenticación y almacenamiento de datos.
  - GitHub como herramienta de almacenamiento en la nube y control de versiones.

  Los recursos software que se utilizan durante este proyecto, permiten al usuario acceder de manera gratuita, por lo que el coste en recursos software es 0€.

El precio total de los costes directos para este proyecto suma un total de 9725.68€. Sin embargo, se deben también añadir los costes indirectos, los cuales suponen aproximadamente un 20 % del total de los costes directos. Entre los costes indirectos podemos incluir luz, agua, alquiler, etc. Por tanto los costes indirectos supondrían un aumento de 1945.14€.

El coste total del proyecto asciende a 11670.82€.

2.5. Riesgos

En este apartado se explican los riesgos que pueden surgir durante el desarrollo del proyecto. En primer lugar se mencionan los posibles riesgos para, posteriormente realizar un análisis de los mismos.

2.5.1. Identificación de los riesgos

En la Tabla 2.1 se citan los principales riesgos que se han tenido en cuenta para este proyecto. Se ha aportado un identificador único para cada riesgo, una descripción breve y el tipo de riesgo del que se trata.

ID	Descripción	Tipo
R01	Baja en el equipo de desarrollo	Riesgo del proyecto
R02	Falta de experiencia en alguna tecnología	Riesgo del proyecto/producto.
R03	Modificación de los requisitos	Riesgo del producto
R04	Poca experiencia en planificación	Riesgo del proyecto.
R05	Daños o funcionamiento anómalo en el hardware.	

Tabla 2.1: Tabla de riesgos

2.5.2. Análisis de los riesgos

A continuación, en las Tablas 2.2 a 2.6 se exponen los riesgos mencionados en el apartado anterior, señalando su importancia, y acompañándolos de una breve descripción que los contextualice y su origen.

R01 Baja en el equipo de desarrollo	
Magnitud	Baja si el trabajador afectado es el supervisor/Alta si el trabajador afectado es la alumna.
Descripción	Algún miembro no puede seguir con sus tareas, debido a una enfermedad que lo imposibilite, una lesión, o algún motivo externo.
Impacto	Si el trabajador que causa la baja modificación de la planificación del proyecto.
Indicadores	Salvo enfermedades largas y muy evidentes, no se puede prever cuándo un trabajador va a necesitar coger una baja.
Plan de actuación	Si el trabajador que causa baja es el supervisor, otro trabajador de la empresa asumirá su trabajo. Si el trabajador que causa baja es la alumna nadie sufrirá su trabajo. Se debe rehacer la planificación, haciendo una interrupción desde el momento de la baja hasta la fecha de alta.

Tabla 2.2: R01. Baja en el equipo de desarrollo

R02 Falta de experiencia en alguna tecnología	
Magnitud	Media/Alta, ya que esto puede causar muchos problemas y atrasos.
Descripción	La alumna encargada de realizar el proyecto no tiene nivel suficiente para realizar los tareas encomendadas en el plazo establecido.
Impacto	Puede haber retrasos o incluso se puede perder calidad en el producto final.
Indicadores	La alumna necesita más tiempo del necesario para implementar las tareas que se le encomiendan.
Plan de actuación	Posponer un poco las tareas y proporcionar documentación adecuada a la alumna para que se forme en aquellos ámbitos en que muestre carencias.

Tabla 2.3: R02. Falta de experiencia en alguna tecnología

R03 Modificación de los requisitos	
Magnitud	Alta.
Descripción	Los centros con los que se trabaja solicitan una nueva funcionalidad o cambiar alguna que ya se había acordado y estaba ya implementada.
Impacto	Reestructura de la planificación si fuese necesaria para cambios notables, para cambios menores simplemente breve retraso de la planificación original.
Indicadores	Los responsables de los centros cambia de opinión con facilidad o se muestran inseguros con las funcionalidades acordadas.
Plan de actuación	Rehacer la planificación si se requiere. Hacer los cambios lo antes posible e intentar seguir con el proyecto sin que suponga un gran retraso.

Tabla 2.4: R03 Modificación de los requisitos

R04 Poca experiencia en planificación	
Magnitud	Media.
Descripción	Desconocimiento de técnicas y buenas prácticas para una planificación realista..
Impacto	Mala planificación con hitos no realistas, que llegado un punto se tiene que rehacer y atrasar todo el proyecto.
Indicadores	Mala comprensión o desconocimiento de técnicas de planificación.
Plan de actuación	Pedir ayuda a expertos. Disponer de documentación de consulta acerca de planificación de proyectos.

Tabla 2.5: R04 Poca experiencia en planificación

R05 Daños o funcionamiento anómalo del hardware	
Magnitud	Alta.
Descripción	Parte o la totalidad de componentes hardware no funcionan, o lo hacen de manera incorrecta.
Impacto	No se pueden comprobar los códigos generados por la aplicación.
Indicadores	Empieza a fallar la conexión, los sensores no varían los valores medidos, los actuadores no realizan el comportamiento esperado.
Plan de actuación	Comprar más componentes y montar más SucreKits en caso de disponer de stock.

Tabla 2.6: R05 Daños o funcionamiento anómalo del hardware

Capítulo 3

Análisis del sistema

En este capítulo se explican las características que el sistema debe tener. En primer lugar se explican los componentes hardware necesarios para la utilización del sistema, se describen las funcionalidades con las que contaba la aplicación original y posteriormente se realiza un análisis, donde se especifican los requisitos.

3.1. Hardware

Como se ha comentado en capítulos anteriores, la aplicación requiere un maletín llamado SucreKit para su utilización. A continuación se explican los componentes que contiene cada SucreKit:

- Un microcontrolador Argon, denominado SucreCore, el cual requiere programación por bloques. Este dispositivo está dotado de conexión a Internet, lo que hace que se considere un dispositivo de Internet de las Cosas (IOT).

- Un conjunto compuesto por 8 sensores que se conectan al SucreCore para registrar mediciones.

    - Sensor de temperatura: Es un sensor digital capaz de registrar temperaturas de entre 0 y 50ºC.

    - Sensor de humedad ambiental: Se trata de un sensor digital que permite medir la humedad del aire, en el rango entre el 20 y el 90 %.

    - Sensor de luz: Sensor analógico para medir la luminosidad de un espacio, mide valores dentro del rango 0-625V.

    - Sensor de humedad en el suelo: Sensor analógico para conocer la humedad del suelo, rango de valores: 380-620V.

    - Sensor de ruido: sensor analógico que capta el ruido del ambiente, rango de valores: 0-1024V.

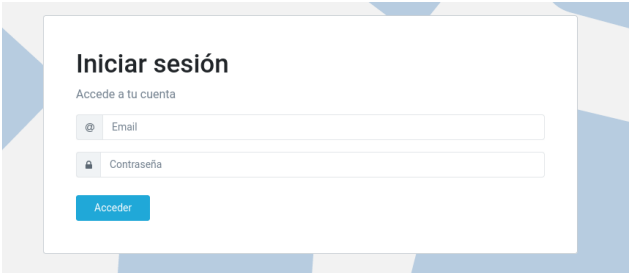
- Sensor de distancia: sensor digital que mide la distancia a un objeto, rango de valores: 0-450cm.
- Sensor de pulsación: consiste en un sensor digital con un botón y detecta si está o no pulsado, rango de valores HIGH, pulsado y LOW, no pulsado.
- Sensor rotacional: este sensor tiene una palanca que se puede rotar, el sensor capta la rotación realizada. Rango de valores: 0-1024V.

- Un conjunto compuesto por 6 actuadores que se conectan al SucreCore para realizar acciones como respuesta.
  - Actuador led simple: este actuador enciende/apaga un led.
  - Actuador led de colores: el led puede estar encendido o apagado, además se puede configurar el color deseado.
  - Actuador relé: este actuador permite abrir o cerrar circuitos eléctricos.
  - Actuador zumbador: puede emitir sonido.
  - Actuador barra led: permite encender de 0 a 10 leds.
  - Actuador pantalla dígitos: permite mostrar valores entre 0000 y 9999.

3.2. Funcionalidades aplicación

A continuación se van a explicar las funcionalidades con las que contaba el proyecto original:

- **Inicio de sesión:** El usuario puede iniciar sesión en la aplicación con su e-mail y contraseña, como muestra la Figura 3.1a.
- **Crear proyectos:** El usuario puede crear un nuevo proyecto del cual será propietario Figura 3.1b. En Sucre se consideran proyectos los programas diseñados por los usuarios por medio del ensamblaje de bloques de programación, como muestra la Figura 3.2.



(a) Inicio de sesión

(b) Crear nuevo proyecto

Figura 3.1: Inicio de sesión y creación de nuevo proyecto

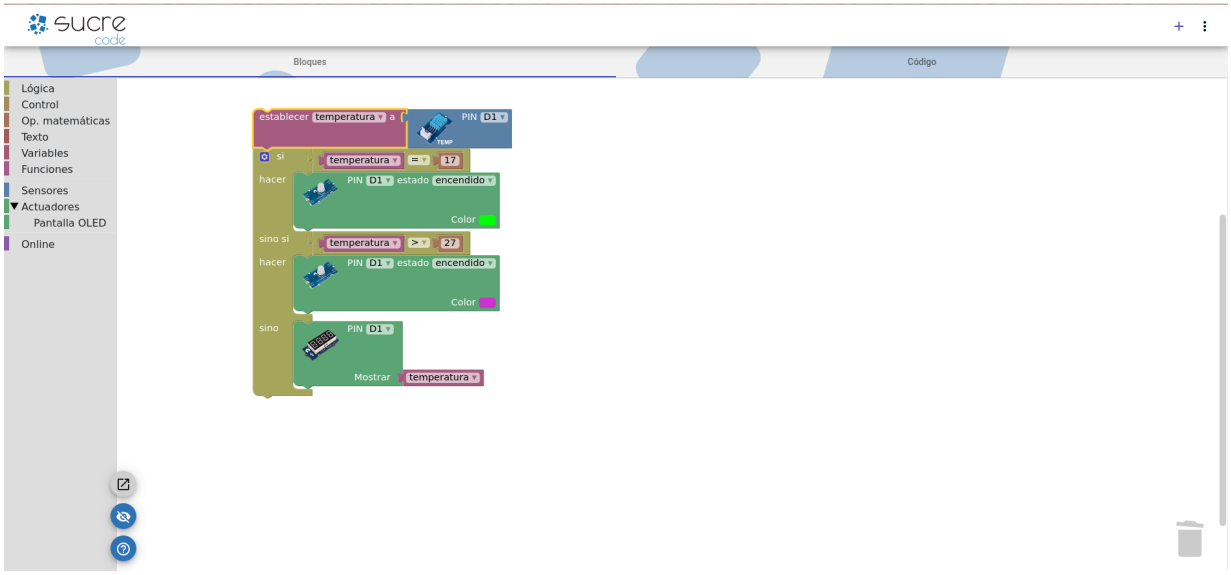
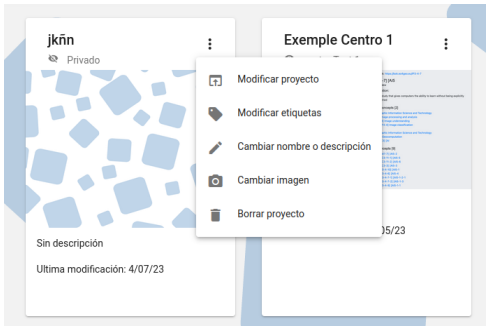
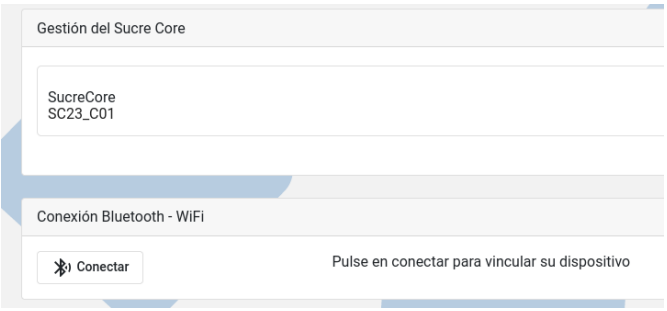


Figura 3.2: Ejemplo de proyecto

- **Vincular dispositivos:** El usuario puede vincular un dispositivo (siempre y cuando sea uno de los que tiene asignado). Esta funcionalidad permite enviar los programas creados al SucreCore para ver los efectos en los actuadores Figura 3.3b.



(a) Gestión de proyectos



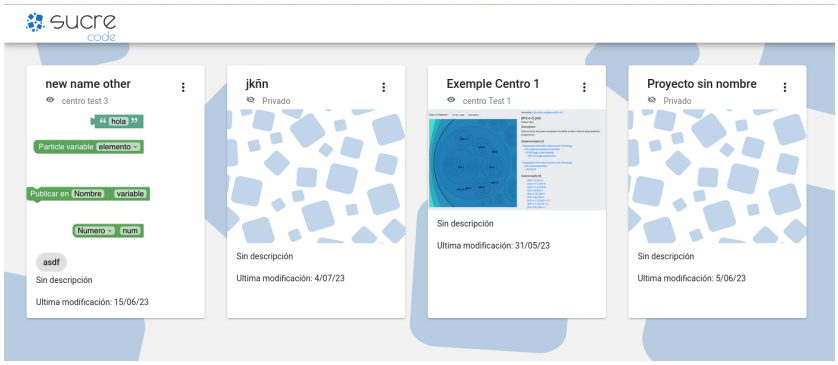
(b) Vincular dispositivo

Figura 3.3: Gestión de proyectos y vinculación de dispositivos

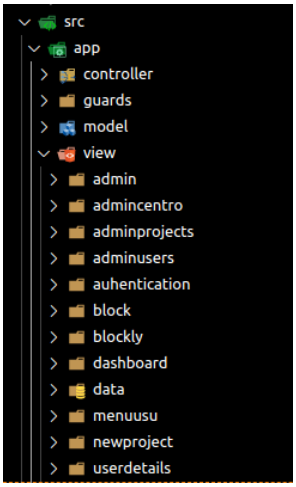
- **Cambiar red Wi-Fi:** El usuario puede conectar el SucreCore a otra red Wi-Fi diferente a la que tenía almacenada.

- **Gestionar proyectos:** El usuario puede modificar el nombre, la descripción o las etiquetas de sus proyectos, cambiar la imagen de la tarjeta del proyecto en el panel principal (dashboard, Figura 3.3a), entrar al proyecto para modificarlo generando nuevos montajes de bloques y, por último, el usuario también puede eliminar sus propios proyectos.

- **Gestiones propias de superusuarios:** Los superusuarios pueden modificar su nombre de usuario en la aplicación, asignar SucreCores a usuarios, eliminar proyectos, crear nuevos perfiles para centros educativos, eliminarlos o añadir o eliminar usuarios de los centros.



(a) Dashboard



(b) Estructura de los paquetes.

Figura 3.4: Estructura de los paquetes del proyecto y dashboard.

El proyecto se ha desarrollado en una aplicación Angular, en la cual para facilitar el desarrollo se han ordenado los ficheros en paquetes según su funcionalidad. En particular se ha seguido el patrón Modelo-Vista-Controlador (MVC), se puede observar la organización de los paquetes en la Figura 3.4b.

3.3. Definición y análisis de requisitos

En esta sección se describen los requisitos del sistema, tanto requisitos funcionales, que se refieren aquellas funcionalidades que deben ser implementadas, como requisitos de datos, esto son los datos que el sistema debe almacenar.

3.3.1. Requisitos funcionales

En las Tablas 3.1 a 3.7 se describen los requisitos funcionales con los que que el sistema contará.

Los requisitos serán: añadir bloque de melodía, cambiar la versión de Sucre, modificar el nombre de usuario sin ser administrador, descargar en PDF la gráfica mostrada, descargar en un fichero CSV las tablas con los valores almacenados, añadir cualquiera de los bloques de sensores o actuadores de la versión básica.

Como se ha comentado en los capítulos anteriores, durante la estancia en prácticas se ha añadido una nueva versión de Sucre4STEM. Para que los proyectos anteriores sigan siendo accesibles y se puedan crear proyectos en ambas versiones, el usuario debe poder cambiar la versión de Sucre en la que se encuentra. Para ello es imprescindible el requisito funcional mostrado en la Tabla 3.1

RF01 Cambiar la versión de Sucre	
Descripción	El sistema debe permitir al usuario cambiar la versión de Sucre con la que creará sus próximos proyectos.
Prioridad	Alta, los usuarios tienen por defecto la versión avanzada activada. Sin este requisito la versión básica quedaría anulada.
Secuencia de acciones	1. Se accede a los tres puntos del menú superior derecho. 2. Se selecciona el nombre de usuario de entre las opciones del menú desplegable. Se selecciona en el menú desplegable el nombre de la versión deseada, básica o avanzada.

Tabla 3.1: RF01 Cambiar la versión de Sucre

En el proyecto original un usuario con el rol de administrador podía cambiar su nombre de usuario. Actualmente los usuarios de cualquier tipo de rol pueden cambiar su nombre de usuario, este requisito funcional se muestra en la Tabla 3.2

RF02 Modificar el nombre de usuario	
Descripción	El sistema debe permitir al usuario cambiar su nombre de usuario aunque no sea administrador.
Prioridad	Baja, el nombre de usuario no es relevante en la aplicación.
Secuencia de acciones	1. Se accede a los tres puntos del menú superior derecho. 2. Se selecciona el nombre de usuario de entre las opciones del menú desplegable. 3. Se escribe el nombre de usuario que se desee tener.

Tabla 3.2: RF02 Modificar el nombre de usuario

Se ha creado un bloque llamado melodía, el cual ha sido implementado en ambas versiones. En la Tabla 3.3 se muestra el requisito funcional correspondiente.

RF03 Añadir bloque de melodía	
Descripción	El sistema debe permitir al usuario, desde cualquiera de las versiones, añadir en su proyecto un bloque de melodía.
Prioridad	Media/Alta.
Secuencia de acciones	1. Se accede a un proyecto o se crea uno nuevo. 2. Se selecciona la categoría Actuadores del menú de bloques. 3. Se arrastra el bloque al workspace del proyecto. 4. Se selecciona el pin en el que se va a conectar este actuador en el microcontrolador.

Tabla 3.3: RF03 Añadir bloque de melodía

A lo largo de este proyecto, se ha creado un conjunto de bloques de programación, estos conjuntos incluyen los bloques correspondientes a sensores y actuadores adaptados a la versión básica de Sucre4STEM. Ambos requisitos funcionales se muestran en las Tablas 3.4 a 3.5.

RF04 Añadir uno de los bloques de sensores de la versión básica	
Descripción	El sistema debe permitir al usuario añadir en su proyecto bloques correspondientes a cualquiera de los sensores de la versión básica.
Prioridad	Alta.
Secuencia de acciones	1. Se accede a un proyecto que aparezca con la etiqueta versión básica o, si no hay ninguno para seleccionar, se verifica que se tiene configurada la versión básica en el usuario y se crea un proyecto nuevo. 2. Se selecciona la categoría *Sensores* del menú de bloques. 3. Se selecciona el tipo se sensor que se desea entre Temperatura, Humedad ambiental, Distancia, Luz, Humedad suelo, Ruido, Botón o Ángulo. 4. Se arrastra el bloque al workspace del proyecto. 5. Se selecciona el pin en el que se va a conectar este sensor en el microcontrolador.

Tabla 3.4: RF04 Añadir uno de los bloques de sensores de la versión básica

RF05 Añadir uno de los bloques de actuadores de la versión básica	
Descripción	El sistema debe permitir al usuario añadir en su proyecto bloques correspondientes a cualquiera de los actuadores de la versión básica .
Prioridad	Alta.
Secuencia de acciones	1. Se accede a un proyecto que aparezca con la etiqueta versión básica o, si no hay ninguno para seleccionar, se verifica que se tiene configurada la versión básica en el usuario y se crea un proyecto nuevo. 2. Se selecciona la categoría *Actuadores* del menú de bloques. 3. Se selecciona el tipo se actuador que se desea entre Led simple, Led multicolor, Barra led, Pantalla de dígitos, Zumbador o Música(Incluye figuras musicales, notas y melodías). 4. Se arrastra el bloque al workspace del proyecto. 5. Se selecciona el pin en el que se va a conectar este actuador en el microcontrolador.

Tabla 3.5: RF05 Añadir uno de los bloques de actuadores de la versión básica

En el proyecto original era posible visualizar gráficas que mostraban los valores almacenados por los diferentes SucreCores. Durante este proyecto se ha implementado una nueva funcionalidad que permite descargar esas gráficas en un documento PDF. La Tabla 3.6 muestra dicho requisito funcional.

RF06 Descargar gráficas en pdf	
Descripción	El sistema debe permitir al usuario descargar las gráficas de los valores almacenados en los dispositivos SucreCore.
Prioridad	Alta, tiene relevancia para realizar informes en los centros educativos en que se utiliza Sucre4STEM.
Secuencia de acciones	1. Se accede a los tres puntos del menú superior derecho. 2. Se selecciona la opción *Almacenamiento* de entre las opciones del menú desplegable. 3. Selecciona la opción Gráfica en la página que aparece. 4. Se seleccionan el SucreCore y el nombre de variable deseados. 5. Se selecciona añadir, para que añada los valores registrados en la tabla y gráfica que aparecen debajo. 6. Se selecciona la opción descargar.

Tabla 3.6: RF06 Descargar gráficas en pdf

El proyecto original también permitía visualizar tablas con los valores almacenados en una variable por cierto SucreCore. En este proyecto se ha implementado una nueva funcionalidad que permite descargar esa tabla como un fichero csv. Esta funcionalidad permitirá realizar estadísticas en los centros educativos. En la tabla 3.7 se muestra dicho requisito funcional.

RF07 Descargar tablas en ficheros csv	
Descripción	El sistema debe permitir al usuario descargar las tablas con los valores almacenados en un fichero CSV.
Prioridad	Media/Alta.
Secuencia de acciones	1. Se accede a los tres puntos del menú superior derecho. 2. Se selecciona la opción *Almacenamiento* de entre las opciones del menú desplegable. 3. Selecciona la opción Tabla en la página que aparece. 4. Se seleccionan el SucreCore y el nombre de variable deseadas. 5. Se selecciona añadir, para que añada los valores registrados en la tabla que aparece debajo. 6. Se selecciona la opción descargar.

Tabla 3.7: RF07 Descargar tablas en fichero csv

El proyecto en su estado original, almacenaba el estado del proyecto en el momento de cerrarlo. Durante la estancia en prácticas se ha creado una funcionalidad para poder guardar el proyecto en el momento que se desee, sin necesidad de salir del mismo. También se ha creado una funcionalidad que permite restaurar el último estado almacenado.Las Tablas 3.8 a 3.9 muestran los requisitos funcionales.

RF08 Guardar estado actual del proyecto	
Descripción	El sistema debe permitir al usuario guardar las modificaciones realizadas sobre un proyecto en cualquier momento.
Prioridad	Media.
Secuencia de acciones	1. Se accede a cualquiera de los proyectos del usuario. 2. Se realizan las modificaciones necesarias. 3. Se selecciona el botón de guardar proyecto (representado por un disquete).

Tabla 3.8: RF08 Guardar estado actual del proyecto

RF09 Cargar último estado almacenado	
Descripción	El sistema debe permitir al usuario cargar el proyecto que anteriormente había guardado.
Prioridad	Media.
Secuencia de acciones	1. Se accede a uno de los proyectos del usuario. *Almacenamiento* de entre las opciones del menú desplegable. 3. Se realizan las modificaciones necesarias, y en el momento que se desea se selecciona el botón de cargar proyecto (representado por un fichero con una flecha).

Tabla 3.9: RF09 Cargar último estado almacenado

3.3.2. Requisitos de datos

Durante la estancia se ha añadido un nuevo atributo en la base de datos Firebase para el usuario, el cual permite identificar qué versión de Sucre está utilizando actualmente, es decir, en qué versión se crearán sus nuevos proyectos. En la Tabla 3.10 se describe el requisito de datos relacionado con la información del usuario.

RD01 Información del usuario	
Descripción	El sistema debe almacenar la información relativa a los usuarios, correo electrónico, contraseña y versión de Sucre (por defecto Avanzada) de manera obligatoria y, de forma opcional, nombre y centro al que pertenece.
Prioridad	Alta.
Comentarios	La información del usuario es imprescindible, pues permite a la aplicación conocer qué dispositivos SucreCore tiene en propiedad, cuáles están activos, qué proyectos ha creado el usuario, en qué versión se tienen que crear los nuevos proyectos, etc.

Tabla 3.10: RD01 Información del usuario

En este proyecto también se ha añadido un atributo a cada proyecto, el cual permite cargar el menú de bloques correcto según la versión de Sucre a la que pertenece, para ello necesitamos otro requisito de datos, el cual se muestra en la Tabla 3.11.

RD02 Información del proyecto	
Descripción	El sistema debe almacenar la información relativa al proyecto, id del proyecto, id del propietario, estado actual de los bloques, nombre y versión de Sucre (la versión que tiene el propietario en el momento de su creación) de manera obligatoria y, de forma opcional, descripción e imagen del proyecto.
Prioridad	Alta.
Comentarios	La información del proyecto es imprescindible, pues permite a la aplicación reconocer quién puede abrir ese proyecto, qué imagen, etiqueta de versión y nombre se deben mostrar en el dashboard, qué menú de bloques se debe cargar, qué montaje de bloques se debe cargar al abrir el proyecto, etc.

Tabla 3.11: RD02 Información del proyecto

Durante este proyecto se han creado dos funcionalidades que son descargar gráficas de los datos almacenados en pdf y descargar ficheros csv con las tablas de los datos almacenados en la base de datos Influx. Por tanto el último requisito de datos son los valores almacenados, que se muestran en la Tabla 3.12

RD03 Valores almacenados	
Descripción	El sistema debe almacenar las variables que el usuario guarde en sus programas, en particular debe almacenar el nombre del SucreCore en que se ha generado, el grupo de valores al que pertenece (variable escogida por el usuario) y la marca de tiempo en que se registró ese valor.
Prioridad	Alta.
Comentarios	Para poder mostrar las variables almacenadas, en ese SucreCore se debe haber almacenado al menos un conjunto de datos.

Tabla 3.12: RD03 Información del proyecto

3.4. Casos de uso

El diagrama de casos de uso se utiliza en los proyectos de software para mostrar la comunicación y el comportamiento de un sistema mediante la interacción del mismo con los usuarios u otros sistemas.

A partir de los requisitos funcionales, se definen los siguientes casos de uso que se van a implementar a lo largo de la estancia en prácticas, los cuales ampliarán los que ya tenía la aplicación existente:

- CU01: Cambiar versión de Sucre.
- CU02: Modificar nombre usuario.

- CU03: Descargar gráfica en pdf.
- CU04: Descargar tabla de datos en csv.
- CU05: Seleccionar bloques musicales.
- CU06: Guardar estado actual del proyecto.
- CU07: Restaurar último estado guardado del proyecto.

En la Figura 3.5 se muestra el diagrama de casos de uso, en el cual se muestran los casos de uso citados anteriormente, que como se ha mencionado, son los correspondientes a los requisitos funcionales implementados durante la estancia en prácticas. Después, en las Tablas 3.13 a 3.19 se van a describir los mismos.

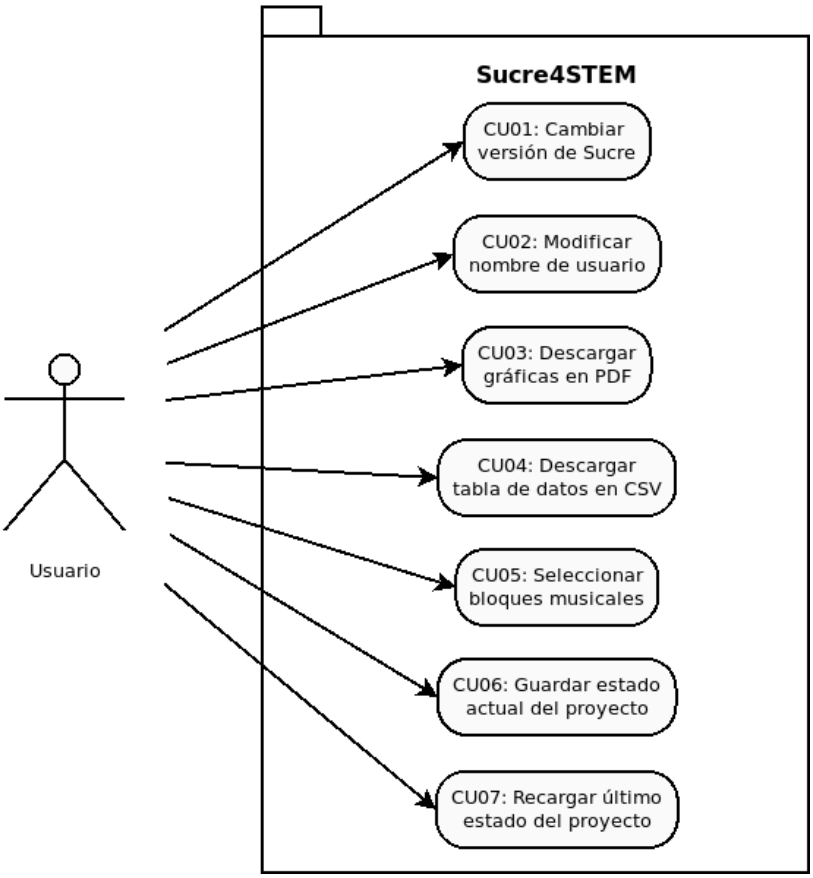


Figura 3.5: Diagrama de Casos de Uso

Como se ha mencionado anteriormente, la aplicación va a poder usar la versión avanzada, que es la original y la versión básica. Para ello es importante que el usuario pueda seleccionar qué versión de Sucre desea utilizar en cada momento, pues si no se le permite cambiar de versión, sólo podrá crear proyectos de la versión actual. En la Tabla 3.13 se muestra el caso de uso 01, correspondiente a cambiar la versión de Sucre.

CU01 Cambiar versión de Sucre
Como usuario necesito poder cambiar mi versión de Sucre.
Recurso Modificar un dato en la base de datos.
Como Usuario.
Propósito Poder crear nuevos proyectos de la versión deseada en cada momento.

Tabla 3.13: CU01 Cambiar versión de Sucre

En la versión original de Sucre4STEM, cambiar el nombre de usuario era una funcionalidad que sólo se permitía a los administradores. En la nueva versión Sucre permite a cualquier tipo de usuario poder cambiar su nombre de usuario. En la Tabla 3.14 se describe el caso de uso 02, correspondiente a cambiar el nombre de usuario.

CU02 Cambiar nombre de usuario
Como usuario quiero poder cambiar mi nombre de usuario.
Recurso Modificar un dato en la base de datos.
Como Usuario.
Propósito Visualizar de forma diferente mi nombre de usuario en el menú.

Tabla 3.14: CU02 Cambiar nombre de usuario

Como se muestra en la Tabla 3.15 el caso de uso 03, corresponde a descargar en formato PDF las gráficas mostradas por la aplicación. Un usuario puede seleccionar las variables almacenadas de cualquier SucreCore y mostrar una gráfica con las mismas. Dicha gráfica se podrá descargar en un documento PDF.

CU03 Descargar gráficas en PDF
Como usuario necesito poder descargar las gráficas que se muestran.
Recurso Obtener datos de la base de datos Influx.
Como Usuario.
Propósito Poder crear estadísticas y registrar los experimentos realizados.

Tabla 3.15: CU03 Descargar PDF con las gráficas

El caso de uso 04, corresponde a descargar un fichero csv con los datos de la tabla mostrada sobre una variable. La aplicación permite mostrar en forma de tabla las mediciones almacenadas en una variable por un SucreCore. Esta tabla se podrá descargar en un fichero csv. El caso de uso se muestra en la Tabla 3.16.

CU04 Descargar tablas en fichero CSV
Como usuario necesito poder descargar las tablas mostradas.
Recurso Obtener datos de la base de datos Influx.
Como Usuario.
Propósito Poder crear estadísticas y registrar los experimentos realizados.

Tabla 3.16: CU04 Descargar fichero CSV con las variables almacenadas

Durante el proyecto se ha creado un conjunto de bloques tanto para la versión avanzada, en caso de los musicales, que están incluidos en ambas versiones, como para la versión básica, en caso de los nuevos bloques de sensores y actuadores. Un caso de uso de la aplicación, por tanto, será el CU05, seleccionar bloques musicales. En la Tabla 3.17 se muestra el caso de uso.

CU05 Seleccionar bloques musicales
Como usuario necesito poder seleccionar bloques musicales, y cualquiera del menú.
Como Usuario.
Propósito Poder crear proyectos adaptados a la asignatura de música, dar respuestas no visuales para alumnos con discapacidad visual, etc.

Tabla 3.17: CU05 Seleccionar bloques musicales

Un usuario necesita guardar el estado un proyecto en cualquier momento, sin necesidad de salir del mismo. También necesita cargar el último estado almacenado en cualquier momento. En las Tablas 3.18 y 3.19 se muestran los caso de uso correspondientes.

CU06 Guardar estado actual del proyecto
Como usuario necesito poder almacenar los montajes de bloques que he realizado.
Recurso Guardar un dato en la base de datos.
Propósito Almacenar un estado para mostrar una y otra vez cierto experimento en las aulas sin necesidad de ocupar tiempo en realizar el ensamblaje.

Tabla 3.18: CU06 Guardar estado actual del proyecto

CU07 Cargar último estado almacenado del proyecto
Como usuario necesito poder recargar el último guardado de mi proyecto.
Recurso Obtener un dato de la base de datos.
Como Usuario.
Propósito Poder recrear escenarios de prueba.

Tabla 3.19: CU07 Cargar último estado almacenado del proyecto

Capítulo 4

Diseño del sistema

4.1. Diseño de la base de datos

La base de datos NoSQL está implementada en Firebase conforme al diseño del diagrama de clases de la aplicación, tal y como muestra la figura 4.1.

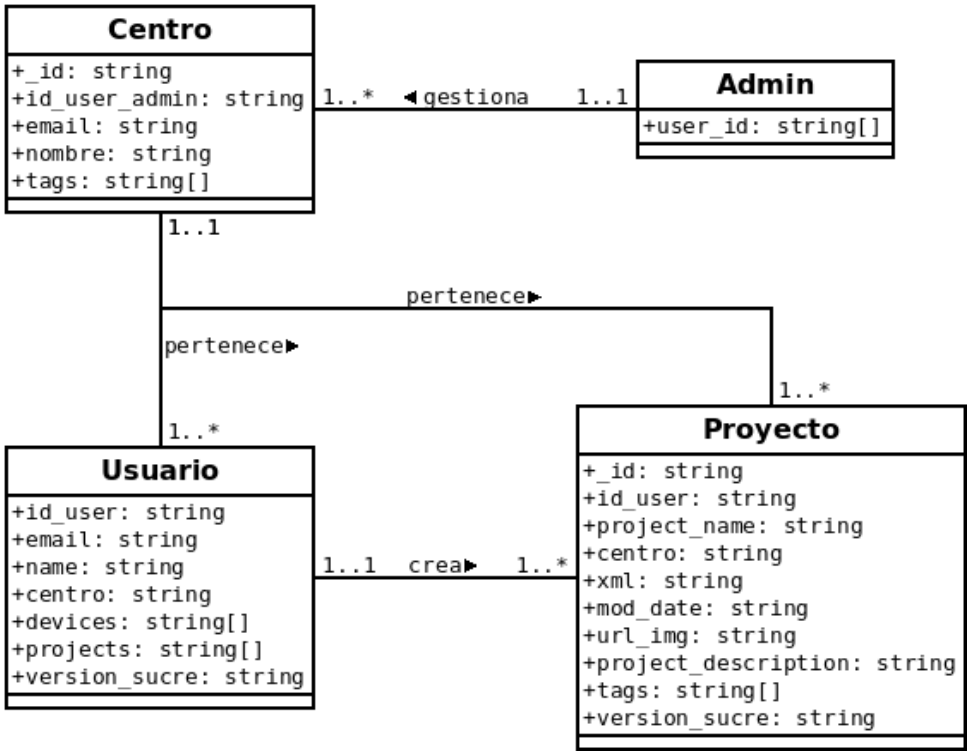


Figura 4.1: Diagrama de clases de la aplicación

Para cumplir con las funcionalidades mencionadas anteriormente, respecto a la versión original de Sucre, en este diagrama únicamente se han añadido los atributos *versión_sucre*, de tipo String, tanto en la tabla Proyecto, como en la tabla Usuario.

Para guardar el estado actual del proyecto no ha hecho falta ningún atributo adicional, pues la base de datos ya contaba con el campo xml, de tipo String. Este campo almacena el estado del proyecto en el momento en que se sale del mismo.

4.2. Diseño de software

En esta sección se detalla como está diseñada la aplicación Sucre4STEM sobre la que se han añadido las nuevas funcionalidades. El código del proyecto está estructurado como se muestra en la Figura 4.2, y se explicará su contenido en la siguiente sección.

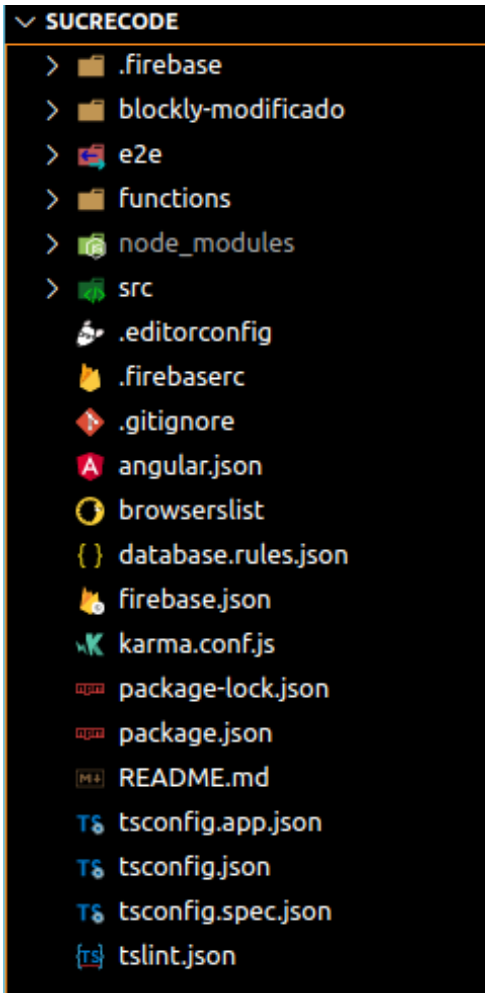


Figura 4.2: Estructura del proyecto

4.2.1. Aplicación Sucre4STEM

En el fichero src/app es donde se encuentra todo lo relativo a la aplicación. Este fichero contiene los paquetes controller, model y view, que son los que necesitaremos modificar para llevar a cabo el proyecto.

En el paquete controller se encuentran los siguientes ficheros:

- database.service.ts. Este fichero es el encargado de interactuar con la base de datos Fireba-

se, por medio de él obtenemos los proyectos de un usuario, los datos del usuario, podemos realizar inserciones, actualizaciones y borrados, etc.

- influx.service.ts. Este fichero se encarga de interactuar con la base de datos InfluxDB, aquí es donde se almacenan las variables medidas en los proyectos por cada SucreCore. También es el intermediario para eliminar variables almacenadas.

- bluetooth.service.ts. Este fichero es el responsable de la conexión del SucreCore por bluetooth, también es quien gestiona las redes WiFi almacenadas y proporciona la información sobre la conexión establecida.

- particle.service.ts. Este fichero es el encargado de las traducciones a C del código de los bloques, cargar las librerías correspondientes a Particle y necesarias para la utilización de los sensores, las cuales se mencionarán en el capítulo 5.

- Este fichero se ocupa de actualizar las propiedades del usuario cuando el estado de autenticación del usuario cambia, utilizando la biblioteca AngularFireAuth.

En el paquete model se encuentra firestore-model.ts, que es el fichero responsable de la creación de la base de datos Firebase. También se encuentra el fichero accesPoint.ts, cuya misión definir y manejar la lógica relacionada con el acceso a un punto de acceso inalámbrico, es decir, el fichero responsable de la conexión por wifi del dispositivo SucreCore.

Del paquete view comentaremos únicamente los directorios y ficheros en los que hemos trabajado durante el proyecto:

- blockly: directorio que contiene el fichero blockly.component.html, el cual controla el aspecto visual de los proyectos.
  También contiene el fichero blockly.component.ts, que es un fichero escrito en typescript que contiene las funciones que indican qué debe hacer la aplicación al iniciar o cerrar un proyecto, qué menú tiene que cargar, contiene las funciones necesarias para actualizar un proyecto, cargar el último montaje guardado de un proyecto, etc.

- El fichero dashboard.component.html, que indica lo que se va a mostrar en el dashboard de la aplicación, en función de si los proyectos son propios o de otro usuario, según la versión de Sucre que utilizan, etc.

- data: contiene el fichero data.component.html, que contiene la vista que se muestra de las variables almacenadas para cada SucreCore. La vista se muestra agrupada por SucreCore y variable y dentro de la selección se puede cambiar a visualizar tablas o gráficas, también muestra los botones para descargar las gráficas en pdf o las tablas en csv.
  En el mismo directorio también se incluye el fichero data.component.ts, encargado de obtener los datos almacenados para cada SucreCore y generar las gráficas, las tablas, hacer las conversiones a pdf o csv, etc.

- El fichero userdetails.component.ts, encargado de interactuar con el fichero database.service.ts para actualizar los datos del usuario en la base de datos en función de lo seleccionado en userdetails.component.html, que es la vista que permite a un usuario cambiar su nombre o versión de Sucre.

- app.component.html: fichero que contiene el menú donde los usuarios pueden seleccionar las opciones de vincular un dispositivo, ver las variables almacenadas, cambiar su nombre o versión de Sucre, etc.

4.2.2. Bloques de programación

La aplicación de Sucre4STEM, como se ha comentado antes, permite ensamblar bloques que, automáticamente, se traducen a código de programación escrito en lenguaje C.

En el directorio blockly-modificado, mostrado en la Figura 4.2 es donde se deben realizar todas las acciones referentes a la creación o modificación de los bloques de programación. En particular, en los subdirectorios blocks y generators/dart es donde se escribe el código referente a la apariencia y funcionalidad de los bloques, respectivamente.

Cada uno de estos subdirectorios está compuesto por ficheros JavaScript correspondientes a cada tipo de bloque. Es decir, hay un fichero math.js para los bloques de operaciones matemáticas, un fichero logic.js para los bloques correspondientes a la lógica, un fichero loop.js para operaciones de control...

El fichero grove.js es el correspondiente a los bloques de tipo sensor o actuador. Por tanto para generar un nuevo bloque de programación para un sensor o actuador necesitamos seguir los siguientes pasos:

- Crear el bloque en blockly-modificado/blocks/grove.js. En este fichero se crea aquello referente al aspecto del bloque, tanto en color, forma, entradas que tendrá, como si contendrá algún desplegable. También se insertan las imágenes, el texto que se mostrará al pasar por encima el ratón, etc. También se puede definir qué tipo de datos recibe o devuelve un determinado bloquee, por ejemplo, si un bloque tiene una entrada que definimos como *Number*, no se podrá ensamblar con un bloque cuya salida sea un *Boolean*.

- Crear el bloque con el mismo nombre en blockly-modificado/generators/dart/grove.js. Este fichero es el que se encarga de definir el comportamiento del bloque, aquí es donde debemos definir si se pasa alguna función o variable al código generado. Este fichero es también el encargado de recibir los valores del bloque (pin, selects, inputs de otros bloques, etc.), hacer las operaciones necesarias y devolver el resultado obtenido.

4.2.3. Diseño de los bloques

Como se detalla en el objetivo O2, del apartado Objetivos del proyecto, la idea es generar bloques de programación, para conseguir en nuestra aplicación un comportamiento similar al de las tarjetas de la versión Sucre4KIDS. Consiste en crear los bloques relativos los sensores y actuadores de la versión básica de Sucre4STEM.

Todos los bloques correspondientes a los sensores en la versión básica devuelven valores booleanos. Por cada sensor se han creado dos bloques, y dependiendo si cumplen o no cierto umbral predefinido en el fichero grove.js de blockly-modificado/generator/dart, devuelven true o false. Los bloques se han creado de la siguiente manera:

- Temperatura: Se ha creado un bloque correspondiente a calor, el cual engloba las tem-

peraturas a partir de 25ºC y otro correspondiente a frío, para temperaturas menores a 25ºC.

- Humedad ambiental: Se han creado dos bloques, uno para cuando hay poca humedad, que acepta humedades de hasta un 55 % y otro bloque para definir mucha humedad, que se corresponde con humedad por encima del 55 %.

- Distancia: se han creado dos bloques, correspondientes a cerca, con distancia hasta 10cm y otro para indicar que un objeto está lejos, lo que implica una distancia mayor a 10cm.

- Luz: se ha creado un bloque que corresponde a que haya suficiente luz, cuya luz medida es hasta 11V, y otro para indicar que hay poca luz, que se corresponde con valores medidos por debajo de 11V.

- Humedad suelo: se ha creado un bloque que equivale a suelo agua, el cual se activa para valores mayores a 475V y otro bloque, suelo seco para valores hasta 475V.

- Ruido: Para el sensor de ruido, se usa la función definida anteriormente, en el Código 5.1 para extraer el valor medio de ruido, si este es un valor menor a 120V se considera que hay silencio y si es un valor a partir de 120V se considera que hay ruido ambiental.

- Pulsación: Se han creado dos bloques correspondientes a botón pulsado y botón no pulsado, en función de si el valor medido es HIGH o LOW respectivamente, para medirlo se traduce a entero y se compara con 0 (no pulsado) o 1 (pulsado).

- Ángulo: Se han creado dos bloques, ángulo pequeño, ángulo grande, en función de si es hasta 512V o mayor, respectivamente.

4.3. Diseño de la arquitectura del sistema

Los dispositivos del sistema se interconectan a través de un router. El ordenador del usuario y el SucreCore deben pertenecer a la misma red local para operar y, por medio del router, se accede a la base de datos alojada en Amazon Web Service. La figura 4.3 ilustra como sería esta arquitectura del sistema

4.4. Diseño de las interfaces

En esta sección se van a comparar los diseños de la aplicación original con los que posteriormente se añadirán al sistema.

4.4.1. Dashboard

Como se muestra en la Figura 4.4 en el dashboard aparecen los proyectos organizados por tarjetas, que llamaremos card de proyecto, estas contienen etiquetas con el centro al que pertenecen, también aparece si son o no privadas, la descripción en caso de tener y una imagen. La imagen de la card de proyecto es una imagen genérica igual al fondo del dashboard para los proyectos que no tienen una imagen guardada. En caso de ser un proyecto de otro usuario, la imagen sale con mayor transparencia.

Para las cards de nuestros propios proyectos de la versión avanzada(que no tengan una imagen personalizada asignada), vamos a utilizar un diseño similar, pero más colorido, con el fin de darle un toque más infantil. El nuevo diseño se muestra en la Figura 4.5a, en caso de ser cards de proyectos de la versión básica pero de otros usuarios utilizaremos la imagen mostrada en la Figura 4.5b

4.4.2. Bloques

En cuanto a los bloques, tanto para sensores como actuadores, la versión anterior mostraba imágenes en cada bloque del sensor o actuador del que se trataba. Puesto que en la versión básica va a haber varios bloques por sensor o actuador, se van a sustituir las imágenes por otras más descriptivas y que resulten intuitivas para el usuario.

En la tabla 4.1 se muestran las imágenes que se van a utilizar para los actuadores, mientras que en la tabla 4.2 muestra las imágenes que se van a utilizar para los sensores.

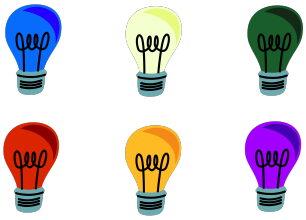
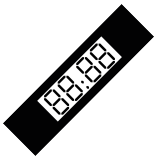
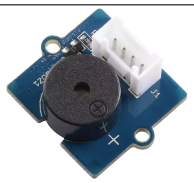
Actuador	Versión Avanzada	Versión Básica
Led simple/encendido y apagado		
Led variable/azul, amarillo, verde, rojo, naranja y morado		
Barra led		
Pantalla dígitos		
Zumbador/on y off		

Tabla 4.1: Imágenes empleadas en los bloques de actuadores para las versiones avanzada y básica.

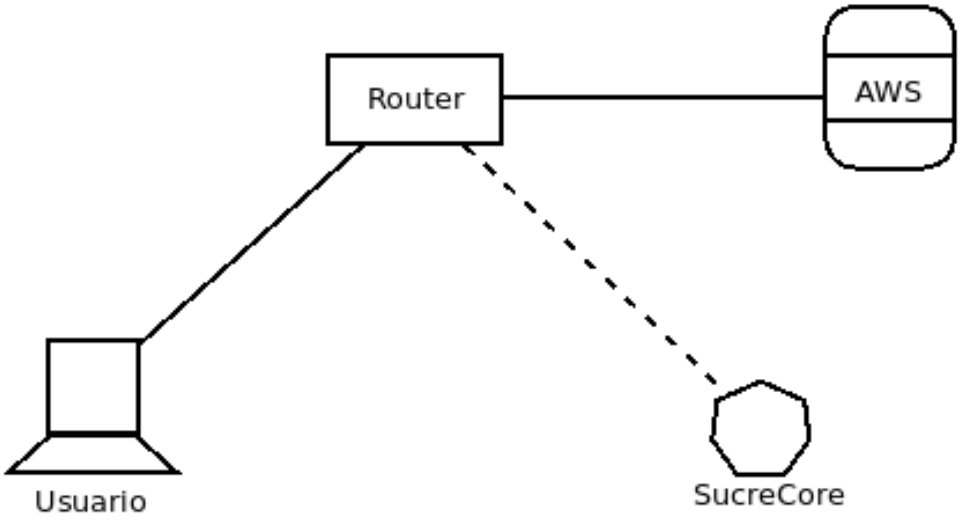


Figura 4.3: Arquitectura del sistema

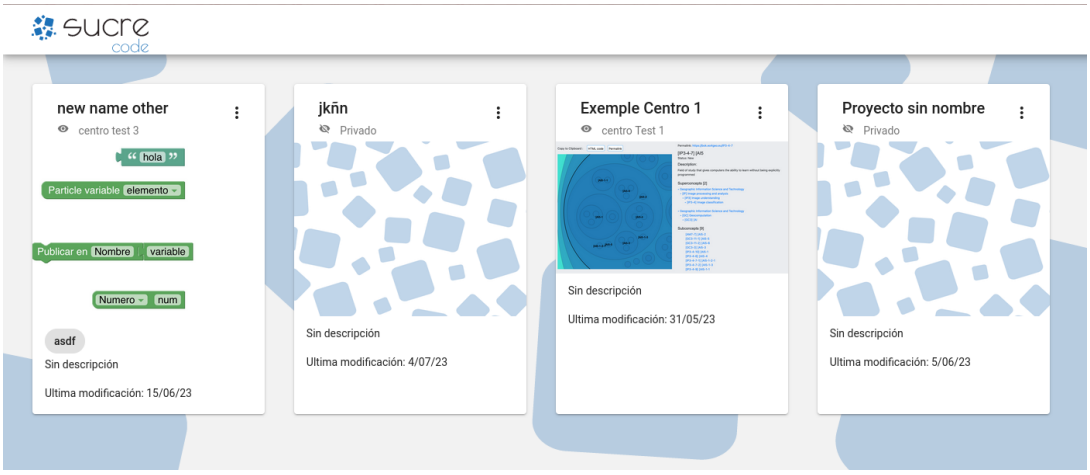


Figura 4.4: Dashboard

(a) Proyecto propio (b) Proyecto ajeno

Figura 4.5: Imágenes card de proyecto en versión básica

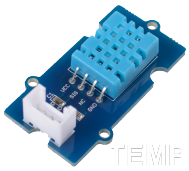
Sensor	Versión Avanzada	Versión Básica
Temperatura/frío y calor		 
Humedad ambiental/poca y mucha humedad		 
Distancia/cerca y lejos		 
Luz/oscuridad y luz		 
Humedad suelo/Poca y mucha humedad		 
Ruido/Silencio y ruido		 
Botón/No pulsado y pulsado		 
Ángulo/Ángulo pequeño y grande		 

Tabla 4.2: Imágenes empleadas en los bloques de sensores para las versiones avanzada y básica.

En el proyecto se han añadido bloques musicales. Cabe destacar que estos bloques serán diferentes para las versiones básica y avanzada, siendo el bloque de la versión avanzada único y permitiendo seleccionar la frecuencia y duración de la nota por medio de dos desplegables. Para la versión básica habrá un bloque por nota y un bloque por duración. Así, para generar música en la versión básica se deberá ensamblar un bloque de nota con un bloque de tiempo. Las imágenes que se utilizan para identificar los bloques musicales se muestran en la Tabla 4.3

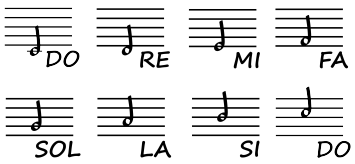
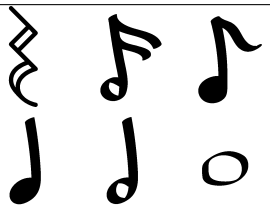
Música	Versión Avanzada	Versión Básica
Actuador/Notas		
Actuador/Tiempos		
Melodía		

Tabla 4.3: Imágenes empleadas en los bloques de música para las versiones avanzada y básica.

Capítulo 5

Implementación y pruebas

En este capítulo se detallará el proceso de implementación y las pruebas realizadas para verificar que el sistema desarrollado funciona correctamente.

La implementación se ha estructurado por tareas, cada una correspondiente a uno de los objetivos O1, O2 y O3 del listado proporcionado en el apartado de objetivos del proyecto.

Cabe destacar que Blockly devuelve el valor como una cadena de texto que añade a la pestaña *Código* de la aplicación, concretamente dentro de loop(), de modo que se ejecuta en bucle una y otra vez. Todo el código contenido en la pestaña de *Código* será lo que se compilará y ejecutará para ser subido al dispositivo SucreCore. Por tanto, todos los bloques tendrán como return una cadena de texto con el código a ejecutar.

Todos los códigos implementados y a los que haremos referencia en este capítulo se encuentran detallados en el anexo A.

5.1. O1: Mejoras de la versión actual de Sucre4STEM

En primer lugar, la aplicación existente de Sucre dejaba ensamblar bloques cuyos tipos de datos de entrada/salida no coincidía, se puede ver un ejemplo en la Figura 5.1 en que un bloque relativo a operación lógica que espera un dato booleano (True o False) se puede ensamblar con un bloque correspondiente a un sensor, el cual devuelve un valor de tipo Integer.

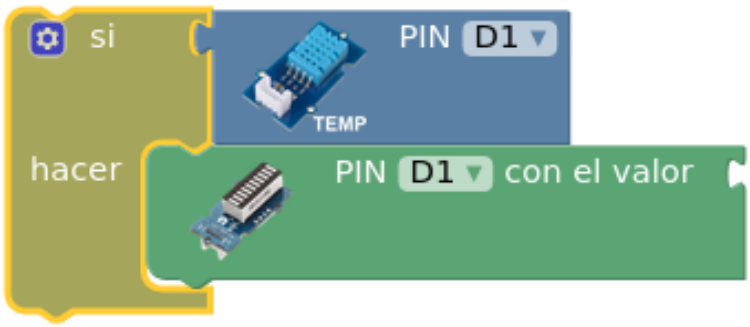


Figura 5.1: Montaje con incoherencias de tipo de dato

Para subsanar este error en el fichero grove.js del directorio blockly-modificado/blocks, se debe modificar el campo de salida de los bloques de todos los sensores para que el valor devuelto por el campo de salida sea de tipo *Number*. El Anexo A nos muestra ejemplos detallados del código implementado (Código A.1).

Por otro lado, el uso del sensor de ruido se complica bastante, puesto que el valor recogido por el sensor cambia constantemente con cualquier mínimo ruido ambiental. Es por ello que se decidió crear una función que devuelva la cantidad media de ruido medido en un intervalo de tiempo (Código 5.1).

Esta función está disponible para su utilización cada vez que se use el sensor de ruido, como se muestra en la línea 8 del Código 5.2

```
1  int mediaRuido(int dropdown_pin, int tiempo)
2  {
3    int acum = 0;
4    int n = 0;
5    NtpTime t_time;
6    unsigned long tInicio = t_time.now();
7    while((int) (t_time.now() - tInicio) < tiempo)
8    {
9      int code = analogRead(dropdown_pin) / 4;
10     acum += code;
11     n+=1;
12   }
13   int res = acum/n;
14   return res;
15 }
```

Código 5.1: Función para los sensores de ruido

```
1  Blockly.Dart['ruido'] = function(block)
2  {
3      var dropdown_pin = this.getFieldValue('PIN');
4      var umbral = 120;
5      var tiempo = 2;
6
7      Blockly.Dart.definitions_['define_ruido'] = '#include "NtpTime.h"';
8      Blockly.Dart.definitions_['sensor_ruido'] = '\n\nint mediaRuido(int
       dropdown_pin, int tiempo)\n{\n\tint acum = 0;\n\tint n = 0;\n\tNtpTime
       t_time;\n\tunsigned long tInicio = t_time.now();\n\twhile((int) (t_time.now
```

```
    () - tInicio) < tiempo)\n\t{\n\t\tint code = analogRead(dropdown_pin) / 4;\n
    \t\tacum += code;\n\t\tn+=1;\n\t}\n\tint res = acum/n;\n\treturn res;\n}\n\n
    ';

    var code = "\tmediaRuido("+dropdown_pin+","+tiempo+") >= "+umbral;
    return [code, Blockly.Dart.ORDER_ATOMIC]
};
```

Código 5.2: Bloque sensor ruido con función mediaRuido

5.2. O2: Creación de bloques de programación para la versión básica

5.2.1. Bloques relativos a los sensores

Los diferentes bloques relativos a los sensores en la versión básica funcionan de la siguiente manera:

- Se recibe el valor seleccionado para el pin en el bloque.
- Se añaden las librerías o funciones necesarias para cada tipo de sensor.
- Finalmente se crea una variable code de tipo string, cuyo contenido es la instrucción necesaria para realizar la medición correspondiente y se compara con el valor del umbral para ese bloque. Por ejemplo, en el caso del bloque de calor: `code='(int)dht.getHumidity () >= 25'`. De modo que, el valor devuelto al ejecutar el código en la aplicación sea un booleano.

En el Código 5.3 se muestra un ejemplo de creación de un bloque de programación, en particular el bloque que representa el sensor de calor.

```
Blockly.Dart['calor'] = function(block)
{
    var dropdown_pin = block.getFieldValue('PIN');

    Blockly.Dart.definitions_['define_Adafruit_DHT'] = '#include "Adafruit_DHT.
    h"';
    Blockly.Dart.definitions_['define_Adafruit_pin'] = '#define DHTPIN ' +
    dropdown_pin;

    Blockly.Dart.definitions_['define_Adafruit_DHT11'] = '#define DHTTYPE DHT11
    ';

    Blockly.Dart.definitions_['define_DHT'] = 'DHT dht(DHTPIN, DHTTYPE);';

    Blockly.Dart.setups_['setup_dhtbegin_'] = 'dht.begin();';
    var code = '(((int)dht.getTempCelcius()) >= 25)';


    return [code, Blockly.Dart.ORDER_ATOMIC];
};
```

Código 5.3: Código bloque calor en generators/dart/grove.js

Respecto a la apariencia, cada bloque se fija en el mismo color que todos los sensores de la versión avanzada (210), se añade un campo de salida de tipo *Boolean*, se añade un mensaje de texto descriptivo, que se mostrará al pasar el ratón por encima. Además se añaden los siguientes campos de tipo input:

- Una imagen que identifique de qué bloque se trata.
- La url de ayuda.
- Un campo que permita seleccionar el PIN.
- Un campo que define si el PIN será analógico o digital.

En el Código 5.4 se muestra un ejemplo del código que genera el aspecto de un bloque de programación, con él se genera el aspecto del sensor de calor.

```
Blockly.Blocks['calor'] =
{
    init: function()
    {
        this.appendDummyInput()
            .appendField(new Blockly.FieldImage("./assets/media/sensores/
    temperatura/calor.png", 64, 50, "*"))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.digital), "
    PIN");
        this.setOutput(true, 'Boolean');
        this.setColour(210);
        this.setTooltip("CALOR. Este sensor sirve para medir la temperatura, en
     particular para medir si hace calor. Tipo de sensor: digital. Rango de
    valores: true/false, [0-24)°C = false,  [25-50]°C = true");
        this.setHelpUrl("http://www.sucre.uji.es/docs/sensores/#temperatura-
    humedad");
    }
```

```
14 };
```

Código 5.4: Código bloque calor en blocks/grove.js

A continuación se citan las librerías o funciones necesarias para cada tipo de bloque, en función del sensor necesario:

- Sensor de temperatura y humedad ambiental, los bloques relacionados son frío, calor, humedad_aire_poca y humedad_aire_demasiada:
  - `#include "Adafruit_DTH.h".`
  - `#define DHTPIN' + dropdown_pin.`
  - `#define DHTTYPE DHT11.`
  - `Blockly.Dart.Definitions_['define_DHT'] = 'DHT dht(DHTPIN DHTTYPE);'.`
  - `Blockly.Dart.setups_['setup_dht_begin_'] = 'dht_begin();'.`
  - Para medir la temperatura, utiliza la instrucción: `(int) dht.getTempCelsius()`.
  - Para medir la humedad, requiere la siguiente instrucción: `(int) dht.getHumidity()`.
- Sensor de distancia, los bloques relacionados son cerca y lejos:
  - `#include "Grove-Ultrasonic-Ranger.h".`
  - `Blockly.Dart.definitions_['var_ultrasonic'] = 'Ultrasonic ultrasonic(' + dropdown_pin + ');'`
  - Para medir la distancia, utiliza la instrucción: `(int)ultrasonic.MeasureInCentimeters()`.
- Sensor de luz, los bloques relacionados son luz y noLuz:
  - `Blockly.Dart.setups_['setup_button_' + dropdown_pin] = 'pinMode(' + dropdown_pin + ', INPUT);'.`
  - Para medir la cantidad de luz, requiere la siguiente instrucción: `(analogRead(' + dropdown_pin + ') / 4)`.
- Sensor de humedad en el suelo, los bloques relacionados son suelo_agua y suelo_seco:
  - Para medir la cantidad de humedad, requiere la siguiente instrucción: `(analogRead('+ dropdown_pin+') / 4)`.
- Sensor de pulsación, los bloques relacionados son boton_pulsado y boton_NO_pulsado:
  - `Blockly.Dart.setups_['setup_button_' + dropdown_pin] = 'pinMode(' + dropdown_pin + ',`

`INPUT);'.`

  - Para registrar si el pulsador está apretado o no, requiere la siguiente instrucción: `(int) digitalRead(' + dropdown_pin + ').`

- Sensor rotacional, los bloques relacionados son angulo_pequeño y angulo_grande:

  - `(analogRead(' + dropdown_pin + ') / 4)`

5.2.2. Bloques relativos a los actuadores

Los diferentes bloques relativos a los actuadores en la versión básica funcionan de la siguiente manera, en primer lugar se recibe el valor seleccionado para el pin en el bloque, se añaden las librerías o funciones necesarias para cada tipo de actuador, se crea una variable code de tipo string, cuyo contenido es la instrucción necesaria para poner el estado deseado en ese sensor. Por ejemplo, el código necesario para activar el led multicolor en color morado será el siguiente: `leds.setColorRGB(0,204,51,204).`

Respecto a la apariencia, cada bloque se fija en el mismo color que todos los actuadores de la versión avanzada (140), se añaden dos campos que permiten apilar los diferentes actuadores, tanto por arriba, como por abajo. También se añade un mensaje de texto descriptivo, que se mostrará al pasar el ratón por encima. Además se añaden los siguientes campos de tipo input:

- Una imagen que identifique de qué bloque se trata.
- La url de ayuda.
- Un campo que permita seleccionar el PIN.
- Un campo que define si el PIN será analógico o digital.

A continuación se citan las librerías o funciones necesarias para cada tipo de bloque, en función del actuador necesario:

- Led simple, los bloques relacionados son led_simple_encendido y led_simple_apagado:

  - `Blockly.Dart.setups_['setup_green_led_' + dropdown_pin] = 'pinMode(' + dropdown_pin + ', OUTPUT);'.`

  - Para encender el led simple, requiere la siguiente instrucción: `digitalWrite(' + dropdown_pin + ',HIGH);.`

  - Para apagar el led simple, requiere la siguiente instrucción: `digitalWrite(' + dropdown_pin + ',LOW);.`

- Led multicolor, los bloques relacionados son led_color_amarillo, led_color_naranja, led_color_rojo, led_color_morado, led_color_azul y led_color_verde:

  - `Blockly.Dart.definitions_['define_led_variable'] = '#include "ChainableLED.h"'.`

- `Blockly.Dart.definitions_['define_grove_led_variable'] = 'ChainableLED leds(' + dropdown_pin + ',' + next_pin + ',1);'`
- `Blockly.Dart.setups_['setup_led_variable_begin'] = 'leds.init();'.`
- Para poner el led multicolor en amarillo se utiliza la siguiente instrucción: `leds.setColorRGB(0,255, 255, 0).`
- Para poner el led multicolor en naranja se utiliza la siguiente instrucción: `leds.setColorRGB(0,255, 102, 0).`
- Para poner el led multicolor en rojo se utiliza la siguiente instrucción: `leds.setColorRGB(0,204, 0, 0).`
- Para poner el led multicolor en morado se utiliza la siguiente instrucción: `leds.setColorRGB(0,204, 51, 204).`
- Para poner el led multicolor en azul se utiliza la siguiente instrucción: `leds.setColorRGB(0,51, 51, 255).`
- Para poner el led multicolor en verde se utiliza la siguiente instrucción: `leds.setColorRGB(0,0, 255, 0).`

- Barra led, barra_led_select:
  - En el bloque hay un desplegable cuyo valor seleccionado se almacena en la variable *valor*. Para encender los leds recogidos en la variable valor, se utiliza la siguiente instrucción: `bar.setLevel('+valor+').`
- Pantalla dígitos, pantalla_digitos_input:
  - `Blockly.Dart.definitions_['define_TM1637Display'] = '#include "TM1637Display.h"'.`
  - `Blockly.Dart.definitions_['define_4digit'] = 'TM1637Display display(' + dropdown_pin + ',' + clk_pin + ').`
  - `Blockly.Dart.setups_['setup_4_digit'] = 'display.setBrightness(0x0a).`
  - En el bloque hay un input cuyo valor seleccionado se almacena en la variable *numero*. Para mostrar por pantalla el valor recogido en la variable numero, se utiliza la siguiente instrucción: `display.showNumberDec(' + numero + ').`
- Zumbador, los bloques relacionados son zumbador_encendido y zumbador_apagado:
  - `Blockly.Dart.setups_['setup_piezo_buzzer_' + dropdown_pin] = '\tpinMode(' + dropdown_pin + ', OUTPUT);'.`
  - Para activar el zumbador se utiliza la siguiente instrucción: `digitalWrite(' + dropdown_pin + ', HIGH).`

- Para desactivar el zumbador se utiliza la siguiente instrucción `digitalWrite(' + dropdown_pin + ', LOW)`.

5.3. O3: Mejoras en la aplicación

Esta sección trata sobre mejoras del sistema, las cual son independientes de la versión de Sucre.

5.3.1. Versión del proyecto

La primera mejora consiste en añadir una etiqueta en los proyectos para determinar si un proyecto está realizado en la versión básica o avanzada. Para ello sólo hay que añadir las sentencias que aparecen en el Código 5.5 en el fichero dashboard.component.html. El código hace lo siguiente: para los proyectos que tengan versión de Sucre, les añade una etiqueta con la versión correspondiente, y para los proyectos que no tienen versión, les añade una etiqueta con la versión Avanzada. Esto último es por seguridad, ya que al principio todos los códigos eran de versión Avanzada, pues la otra versión no existía.

```
<mat-chip-list aria-label="version sucre">
    <mat-chip class = "iconoVersion">
        <div *ngIf="project.version_sucre">
            Versi n {{project.version_sucre}}
        </div>
        <div *ngIf="!project.version_sucre">
            Versi n Avanzada
        </div>
    </mat-chip>
</mat-chip-list>
```

Código 5.5: Código etiqueta versión en dashboard

5.3.2. Gestión de usuario

La siguiente mejora es permitir a cualquier usuario poder cambiar su nombre, ya que anteriormente sólo podían cambiar de nombre los usuarios que eran administradores de centros. Permitirle también cambiar la versión de Sucre.

Respecto a permitir que cualquier usuario cambie su nombre, en el fichero userdetails.components.ts aparece el Código5.6:

```
<mat-form-field appearance="outline" *ngIf="usuario" class="form-group col-md
    -12">
    <label for="nombreUsuario">Nombre</label>
    <input matInput required type="text" class="form-control" id="nombreUsuario
    "
 name="nombreUsuario" [(ngModel)]='usuario.name' (keyup)="guardarCambiosUser()"
    >
</mat-form-field>
```

Código 5.6: Cambiar nombre de usuario

Sin embargo, sólo los usuarios que son administradores podían cambiar su nombre, esto es debido a que en el fichero app-routing-module.ts el path estaba condicionado a ello. La condición se muestra en el Código 5.7. Para que cualquier usuario pueda cambiar su nombre, es necesario eliminar la referencia a la variable CentroadminGuard.

```
  { path: 'details', component: UserdetailsComponent, canActivate: [
    AngularFireAuthGuard, CentroadminGuard], data: { authGuardPipe:
    redirectUnauthorizedToLogin } },
```

Código 5.7: Cambiar nombre usuario (sólo admins)

Por otro lado, para permitir que los usuarios puedan cambiar la versión de Sucre con la que crear sus nuevos proyectos se deberán añadir a la vista userdetails.compontent.html el Código 5.8, el cual añade en la vista un selector que actualiza la versión escogida por el usuario.

```
<mat-form-field appearance="outline" *ngIf="usuario" class="form-group col-md
    -12">
    <label for="version_sucre">Version de Sucre </label>
    <mat-select required [(ngModel)]="usuario.version_sucre" (selectionChange)=
    "guardarCambiosUser()" name="tipoSucre">
        <mat-option value="Basica">Basica</mat-option>
        <mat-option value="Avanzada">Avanzada</mat-option>
      </mat-select>
</mat-form-field>
```

Código 5.8: Selector versión Sucre

Además en el fichero userdetails.component.ts se deberá agregar el Código 5.9, que llama al método updateUser pasándole todos los datos necesarios.

```
guardarCambiosUser()
{
    this.db.updateUser(this.usuario.id_user, this.usuario.name, this.usuario.
    devices, this.usuario.centro, this.usuario.version_sucre);
}
```

Código 5.9: guardar cambios usuario

Y en database.service.ts, por medio de updateUser se hace la actualización necesaria en la base de datos. Se muestra en el Código 5.10

```
updateUser(uid, nam, devs, centroId, tipoSucre)
{
    if (!nam)
    {
        nam = '';
    }
    if (!centroId)
    {
        centroId = '';
    }
    if(!tipoSucre)
    {
        tipoSucre = 'Avanzada';
    }

    this.db.collection(colUsu).doc(uid).update({ name: nam, centro: centroId,
    devices: devs, version_sucre: tipoSucre});
}
```

Código 5.10: Actualizar nombre o versión Sucre

Una vez más, en caso de que la versión de Sucre no exista, añadiremos la versión avanzada.

5.3.3. Ordenar proyectos

Otra mejora consiste en ordenar los proyectos en el dashboard por fecha de modificación, de modo que los proyectos que hace más tiempo que nadie modifica se queden los últimos. Además mostrar la última modificación en fecha y hora.

Para ordenar los proyectos es necesario crear en database.service.ts el método mostrado en el Código 5.11. Este método en primer lugar crea una copia del array de proyectos, después utiliza el método sort para ordenar los proyectos mediante una función de comparación personalizada. Este método crea un objeto Date para cada proyecto, de modo que obtiene cadenas de fecha comparables. Finalmente devuelve un número negativo si la primera fecha es menor a la segunda, cero si ambas son iguales o un número positivo si la primera fecha es mayor. Este número le sirve para hacer la ordenación. Una vez ordenada la copia del array se devuelve.

```
getProyectosOrdenados(proyectos: Proyecto[]): Proyecto[]
{
    const copiaProyectos = JSON.parse(JSON.stringify(proyectos));

    copiaProyectos.sort((a, b) =>
    {
        const fechaA = new Date(a.mod_date);
        const fechaB = new Date(b.mod_date);
        return fechaB.getTime() - fechaA.getTime();
    });

    return copiaProyectos;
}
```

Código 5.11: Ordenar proyectos

El fichero dashboard.component.ts llama al método anterior como de muestra en el Código5.12

```
this.projectos\_ordenados = this.db.getProyectosOrdenados(this.projectos);
```

Código 5.12: Ordenar proyectos

A través de la vista dashboard.component.html es mostrado el listado de proyectos ordenados, como se muestra en el Código 5.13

```
<ng-container *ngFor="let project of projectos_ordenados; let i = index"
```

Código 5.13: Vista ordenada de proyectos

5.3.4. Colección musical

La colección musical será diferente en la versión avanzada y en la básica, contendrá únicamente un bloque común, llamado *cumpleanyosFeliz*, que se corresponderá con una melodía de ejemplo de los bloques musicales, en particular la melodía de la canción cumpleaños feliz, este bloque será idéntico para ambas versiones.

El bloque *cumpleanyosFeliz*, como todos los otros bloques que representan algún sensor o actuador, en primer lugar contiene un desplegable con el pin al que se ha conectado el actuador zumbador. Por otro lado el zumbador requiere ser configurado como pin de salida para poder emitir sonidos.

Posteriormente se escriben una a una cada nota que contiene la melodía. Cada nota tiene la estructura tone, delay, noTone, esto se debe a que debe estar en silencio un tiempo igual al que ha estado en sonido para que se aprecie el cambio de una nota a otra, si no se tuviera este silencio, el sonido no sería el adecuado, entremezclándose las notas. Finalmente devuelve la variable code, con todas las notas. En el Código5.14 se muestra el código necesario para la creación del bloque de programación.

```
1    Blockly.Dart['cumpleanyosFeliz'] = function(block)
2    {
3        var dropdown_pin = this.getFieldValue('PIN');
4
5        Blockly.Dart.setups_['setup_piezo_buzzer_' + dropdown_pin] = '\tpinMode
    (' + dropdown_pin + ', OUTPUT);';
6
7        var code = 'tone(' + dropdown_pin +', '+sol+', '+tcorchea+');\ndelay('+
    tcorchea+');\nnoTone('+dropdown_pin+');\n';
8        code += 'tone(' + dropdown_pin +', '+sol+', '+tcorchea+');\ndelay('+
    tcorchea+');\nnoTone('+dropdown_pin+');\n';
9        code += 'tone(' + dropdown_pin +', '+la+', '+tnegra+');\ndelay('+tnegra
    +');\nnoTone('+dropdown_pin+');\n';
10       code += 'tone(' + dropdown_pin +', '+sol+', '+tnegra+');\ndelay('+
    tnegra+');\nnoTone('+dropdown_pin+');\n';
11       code += 'tone(' + dropdown_pin +', '+doAgudo+', '+tnegra+');\ndelay('+
    tnegra+');\nnoTone('+dropdown_pin+');\n';
12       code += 'tone(' + dropdown_pin +', '+si+', '+tblanca+');\ndelay('+
    tblanca+');\nnoTone('+dropdown_pin+');\n';
13       code += 'tone(' + dropdown_pin +', '+sol+', '+tcorchea+');\ndelay('+
    tcorchea+');\nnoTone('+dropdown_pin+');\n';
14       code += 'tone(' + dropdown_pin +', '+sol+', '+tcorchea+');\ndelay('+
    tcorchea+');\nnoTone('+dropdown_pin+');\n';
15       code += 'tone(' + dropdown_pin +', '+la+', '+tnegra+');\ndelay('+tnegra
    +');\nnoTone('+dropdown_pin+');\n';
16       code += 'tone(' + dropdown_pin +', '+sol+', '+tnegra+');\ndelay('+
    tnegra+');\nnoTone('+dropdown_pin+');\n';
17       code += 'tone(' + dropdown_pin +', '+reAgudo+', '+tnegra+');\ndelay('+
    tnegra+');\nnoTone('+dropdown_pin+');\n';
18       code += 'tone(' + dropdown_pin +', '+doAgudo+', '+tblanca+');\ndelay('+
    tblanca+');\nnoTone('+dropdown_pin+');\n';
19       return code;
20   };
21
```

Código 5.14: Comportamiento bloque cumpleaños feliz

En el fichero blockly-modificado/generators/dart/grove.js se han añadido las declaraciones de variables correspondientes a cada nota y figura musical, concretamente para cada nota su frecuencia y para cada figura musical su duración.

Respecto al aspecto, se ha añadido la url de ayuda, una imagen para reconocer el bloque, el selector del PIN, dos campos que indican que se acepta que se enganchen bloques tanto por encima como por debajo, sin ningún tipo de entrada ni salida y por último la descripción al pasar el ratón por encima del bloque. En el Código5.15 se muestra el aspecto del bloque *cumpleanyosFeliz*.

```
1    Blockly.Blocks['cumpleanyosFeliz'] =
2    {
3        helpUrl: 'http://www.sucre.uji.es/docs/actuadores/#zumbador',
```

```
        init: function()
        {
            this.setColour(140);
            this.appendDummyInput()
                .appendField("Cumplea os feliz")
                .appendField(new Blockly.FieldImage("./assets/media/actuadores/
    musica/notas.png", 150, 100, "*"))
                .appendField("PIN")
                .appendField(new Blockly.FieldDropdown(profile.default.digital)
    , "PIN")
            this.setPreviousStatement(true, null);
            this.setNextStatement(true, null);
            this.setTooltip('Descripci n: Canci n "Cumplea os feliz".');
        }
    };

```

Código 5.15: Aspecto bloque cumpleaños feliz

Notas versión Avanzada

Como se ha comentado anteriormente, el bloque de nota de la versión avanzada consta de un desplegable en que se selecciona, la nota y la duración. Además del pin al que está conectado el zumbador, como ocurre con todos los sensores y actuadores.

Respecto al aspecto se ha añadido una imagen de una clave de sol, se le ha añadido la url de ayuda y el mensaje al pasar por encima. Se ha configurado el bloque del mismo color que todos los bloques de actuadores. En el Código 5.16 se muestra cómo se ha definido el aspecto del bloque nota.

```
    Blockly.Blocks['nota'] =
    {
        helpUrl: 'http://www.sucre.uji.es/docs/actuadores/#zumbador',
        init: function()
        {
            this.setColour(140);
            this.appendDummyInput()
                .appendField(new Blockly.FieldImage("./assets/media//actuadores
    /musica/claveSol.png", 64, 64))
                .appendField("PIN")
                .appendField(new Blockly.FieldDropdown(profile.default.digital)
    , "PIN")
                .appendField("nota")
                .appendField(new Blockly.FieldDropdown
                ([
                    ["do", "261"],
                    ["re", "293"],
                    ["mi", "329"],
                    ["fa", "349"],
                    ["sol", "392"],
                    ["la", "440"],
                    ["si", "493"]
                ],null, {'class':'my-dropdown'}), "frecuencia")
                .appendField("tiempo")
                .appendField(new Blockly.FieldDropdown
                    ([
                        ["♩", "1000"],
                        ["𝅗𝅥", "2000"],
```

```
                        ["𝅝", "4000"],
                        ["♪", "500"],
                        ["𝅘𝅥𝅯", "250"]
                    ]), "tiempo");
            this.setPreviousStatement(true, null);
            this.setNextStatement(true, null);
            this.setTooltip('Descripci n: Este bloque representa cualquier
    nota b sica.');
        }
    };

```

Código 5.16: Aspecto bloque notas versión avanzada

Respecto a la funcionalidad, en primer lugar se obtienen el pin, la frecuencia de la nota y la duración seleccionados. Seguidamente se configura el zumbador como salida, para permitirle emitir sonidos.

Después se crea una variable llamada code, de tipo texto, en la que se inserta el texto relativo a emitir el sonido de esa nota durante el tiempo que se ha seleccionado. Finalmente se devuelve la cadena de texto code. En el Código 5.17 se muestra cómo se ha creado el bloque nota.

```
    Blockly.Dart['nota'] = function(block)
    {
        var dropdown_pin = this.getFieldValue('PIN');
        var frecuencia = this.getFieldValue('frecuencia');
        frecuencia = parseInt(frecuencia);
        var tiempo = this.getFieldValue('tiempo');
        tiempo = parseInt(tiempo);

        Blockly.Dart.setups_['setup_piezo_buzzer_' + dropdown_pin] = '\tpinMode
    (' + dropdown_pin + ', OUTPUT);';

        var code = 'tone('+dropdown_pin+', '+frecuencia+', '+tiempo+');\ndelay(
    '+tiempo+');\nnoTone('+dropdown_pin+');\n';
        return code;
    }

```

Código 5.17: Bloque notas versión avanzada

Notas y figuras musicales versión Básica

En cuanto a las notas, como se comentó anteriormente, para emitir el sonido de una nota en la versión básica serán necesarios dos bloques ensamblados. Por un lado tendremos el bloque correspondiente a las nota musical (do, re, mi,etc.), mientras que por otro lado tendremos que ensamblar la figura musical, correspondiente al tiempo que queramos que esa nota esté sonando (blanca, negra, redonda, corchea o semicorchea). Entre las notas es donde incluiremos el silencio, pues su característica es que la frecuencia es 0.

Respecto al aspecto de las notas, se añade a cada nota el icono correspondiente a la representación de dicha nota en un pentagrama y a cada duración la imagen asociada a cada figura musical. El color de bloque se fija en el mismo color verde que el resto de actuadores. Se le ha añadido la url de ayuda y el mensaje de ayuda al pasar el ratón por encima.

En el caso de las notas se ha añadido un desplegable para seleccionar el pin al que se ha conectado el zumbador y un campo de entrada, por el que se ensamblará y se recibirá el valor, de tipo entero, devuelto por los bloques de figuras musicales, el cual será la duración.

Para los bloques de figuras musicales no habrá selector de pin, en su defecto se ha configurado un campo de salida, por el que se devuelve la duración de la figura musical a la que representa el bloque.

Las frecuencias representadas por cada nota son las siguientes:

- do, 249Hz.
- re, 293Hz.
- fa, 329Hz.
- mi, 349Hz.
- sol, 392Hz.
- la, 440Hz.
- si, 493Hz.
- do', 522Hz.
- silencio, 0Hz.

En el caso de las figuras musicales, el tiempo de duración de cada figura es el siguiente:

- negra, 1000ms.
- blanca 2000ms.
- redonda, 4000ms.
- corchea, 500ms.
- semicorchea, 250ms.

En el Anexo A se muestran los Códigos A.56 y A.57 que se utilizan para crear el comportamiento y el aspecto del aspecto de la nota do. También se incluye el Código A.58 y A.59 que se utilizan para representar el comportamiento y aspecto del bloque equivalente a la nota negra.

5.3.5. Descargar gráficas en pdf y tablas en csv

Esta funcionalidad permite descargar las gráficas en pdf y las tablas de variables almacenadas en un fichero CSV. El primer paso para ambos es común, ha sido reestructurar el fichero data.component.html para poner en primer lugar la selección del SucreCore y de la variable

almacenada.

Una vez hemos escogido SucreCore y variable, se escoge en una tabla una de las siguientes pestañas: Tabla, Gráfica o Eliminar.

En el momento en que se escoge la pestaña *Tabla*, si ese SucreCore tiene valores almacenados, aparece el botón para descargar la tabla en un fichero csv. El Código 5.18

```
    <div class="botonCsv">
        <button [disabled]="scSinVariables" mat-flat-button color="primary"
    style="align-self: center;" (click)="descargarTablaCSV()">
            Descargar CSV
        </button>
    </div>

```

Código 5.18: descargar csv

Este botón llama al método descargarTablaCSV() del fichero data.component.ts, que a su vez llama a descargarDocCSV, ambos mostrados en el Código A.60.

Cuando se escoge la pestaña Gráfica, si el SucreCore seleccionado tiene valores almacenados, se deberá seleccionar el botón añadir. En este momento se añadirá a la tabla mostrada por pantalla, se puede ir añadiendo las variables que se deseen, una vez se tengan todas las variables deseadas en la tabla, se selecciona el botón descargar PDF y se descarga la imagen que contiene tanto la tabla con los elementos de la gráfica, como la gráfica en sí.

El botón Descargar PDF, que se muestra en el Código A.61 se encuentra en el fichero data.component.html, y llama al método descargarGraficaPdf() del fichero data.compoent.ts, cuyo código se muestra en el anexo A, Código A.62.

5.3.6. Almacenar montaje

En esta sección se explica cómo se ha implementado la funcionalidad que permite a un usuario almacenar el montaje de bloques actual. En primer lugar, en la vista del proyecto se añade el botón que permite guardar el proyecto. Se muestra en el Código 5.19.

```
    <div class="fab fab-top-3">
        <button mat-mini-fab color="primary" aria-label="Guardar montaje de
    bloques" (click)="updateProject();">
            <mat-icon>save</mat-icon>
        </button>
    </div>

```

Código 5.19: Botón guardar proyecto

Posteriormente en blockly.component.ts, está el método al que se llama con el botón, el cual realiza una llamada al método updateProject(), en el fichero database.service.ts. Se muestra en el Código 5.20

```
    updateProject()
    {
        console.log("se va a guardar el proyecto " + this.project_name)
```

```
        this.db.updateProject(this.project_name, this.ref, this.workspace.toXml
    (), this.project_centro, this.project_description);
    }

```

Código 5.20: Guardar proyecto

Desde el fichero database.service.ts, es desde donde se interacciona con la base de datos para actualizar el proyecto almacenando su estado actual. Se muestra en el Código 5.21

```
    updateProject(name, ref, xml, c, desc)
    {
        let date = new Date().toString();
        desc = desc ? desc : '';
        this.db.collection(colProy).doc(ref).update({ project_name: name,
    centro: c, xml: xml, mod_date: date, project_description: desc});
    }

```

Código 5.21: Guardar proyecto en la base de datos

5.3.7. Restaurar montaje

Esta funcionalidad permite a un usuario restaurar el último montaje de bloques guardado. En primer lugar en la vista del proyecto creamos el botón que permite cargar el proyecto. Se muestra en el Código 5.22

```
    <div class="fab fab-top-4">
        <button mat-mini-fab color="primary" aria-label="Cargar montaje de
    bloques" (click)="cargarUltimoProyecto()">
            <mat-icon>unarchive</mat-icon>
        </button>
    </div>

```

Código 5.22: Botón cargar proyecto

Posteriormente en blockly.component.ts, está el método al que se llama con el botón, este método, llama al método getProuectoById, para extraer del mismo el valor almacenado en el atributo xml y, posteriormente cargarlo en el workspace del proyecto. Se muestra en el Código 5.23

```
    cargarUltimoProyecto()
    {
        this.ref = this.route.snapshot.paramMap.get('ref');

        this.db.getProyectoById(this.ref).subscribe(proy =>
        {
            this.workspace.fromXml(proy.xml);
        })
    }

```

Código 5.23: Cargar proyecto

El método getProyectoById() devuelve un observable de tipo proyecto, que permite obtener y observar los datos del proyecto con el identificador dado. Se muestra en el Código 5.24

```
    getProyectoById(proyId: string): Observable<Proyecto>
    {
        return this.db
            .collection(colProy)
            .doc<Proyecto>(proyId)
            .valueChanges();
    }

```

Código 5.24: Obtener proyecto de la base de datos

Para que tanto guardar como cargar proyecto sean efectivos, al salir de un proyecto la aplicación preguntará si se quiere guardar el estado actual, ya que en la versión original (que no daba opción de guardar o cargar cuando quisieras) al salir del proyecto guardaba automáticamente. El supervisor de las prácticas y yo pensamos que al generar un botón para guardar en el momento que quisieras, no tenía sentido que al salir, en caso de haber realizado algún cambio, se destruyera ese proyecto que se había guardado deliberadamente.

Capítulo 6

Resultados

En esta sección se van a mostrar los resultados de implementar el código anterior.

6.1. O1: Mejoras de la versión actual de Sucre4STEM

6.1.1. Incoherencias de tipo de dato

En la versión original de Sucre, había ciertas incoherencias de datos. Un bloque condicional, permitía ser montado con un bloque de Sensor. En el Código A.1 se mostraba cómo subsanarlo. El resultado de ejecutar ese código es el siguiente:

Figura 6.1: Incoherencias de dato subsanadas

En la Figura 6.1 se muestra que, tras haber ejecutado el código para subsanar esos errores de datos, ahora la aplicación no permite ensamblar un bloque condicional con uno de sensor, pues al intentar unirlos es como si el bloque del sensor rebotase y la aplicación desplaza el sensor para que no lleguen a unirse.

6.2. O2: Creación de bloques de programación para la versión básica

6.2.1. Funciones para el sensor de ruido

También se comentaba en la sección anterior que se hacía difícil saber cuánto ruido ambiental había, ya que es un valor muy cambiante y se dificultaba mucho su observación, para ello en el Código 5.1 se muestra la función que se ha utilizado para medir el ruido, a continuación en

las Figuras 6.2 a 6.4 se muestra el resultado de utilizar la función en los diferentes bloques del sensor de ruido.

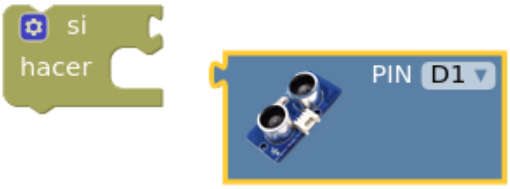


Figura 6.2: Función ruido en sensor versión avanzada

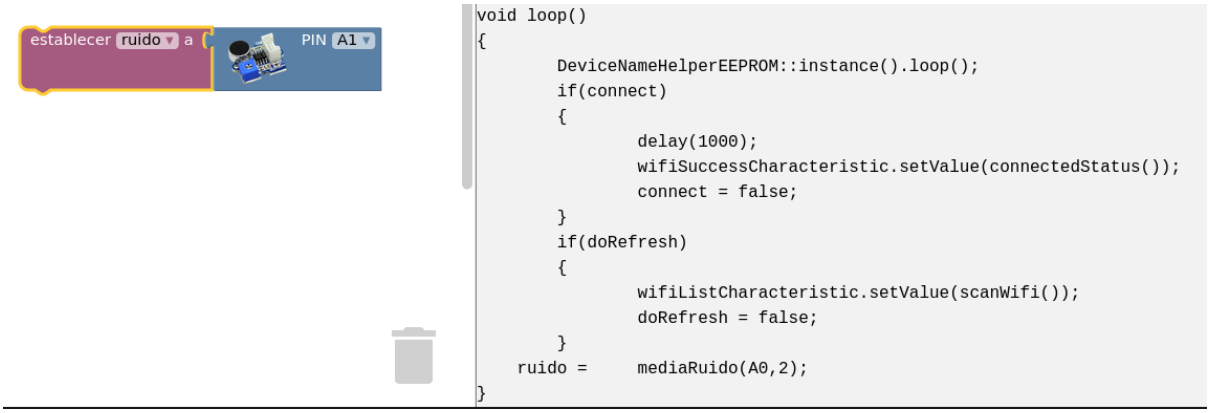
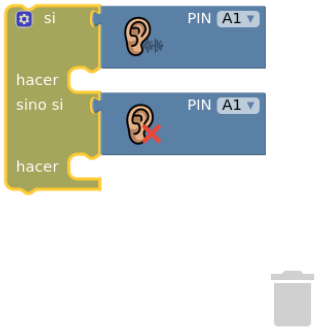


Figura 6.3: Función ruido en sensor ruido

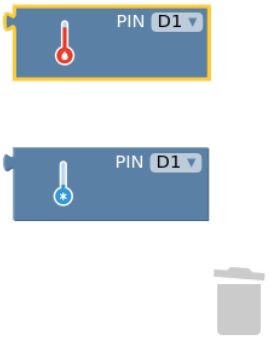



```
void loop()
{
        DeviceNameHelperEEPROM::instance().loop();
        if(connect)
        {
                delay(1000);
                wifiSuccessCharacteristic.setValue(connectedStatus());
                connect = false;
        }
        if(doRefresh)
        {
                wifiListCharacteristic.setValue(scanWifi());
                doRefresh = false;
        }
    if (        mediaRuido(A0,2) >= 120) {
    }else if (  mediaRuido(A0,2) < 120) {
    }
}
```

Figura 6.4: Función ruido en sensor silencio

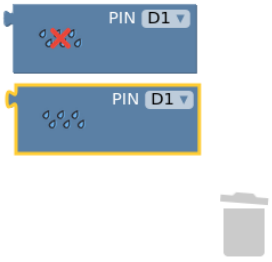
6.2.2. Sensores versión básica

A continuación, en las Figuras 6.5 a 6.12 se va a mostrar el resultado visual de los sensores de la versión básica y los códigos que los mismos generan.



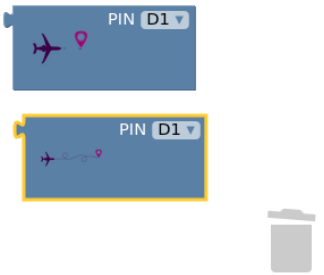

```
void loop()
{
        DeviceNameHelperEEPROM::instance().loop();
        if(connect)
        {
                delay(1000);
                wifiSuccessCharacteristic.setValue(connectedStatus());
                connect = false;
        }
        if(doRefresh)
        {
                wifiListCharacteristic.setValue(scanWifi());
                doRefresh = false;
        }
   (((int)dht.getTempCelcius()) >= 25);

   (((int)dht.getTempCelcius()) < 25);
```

Figura 6.5: Sensores básicos temperatura



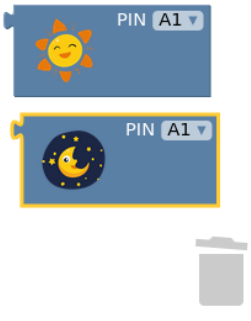

```
void loop()
{
        DeviceNameHelperEEPROM::instance().loop();
        if(connect)
        {
                delay(1000);
                wifiSuccessCharacteristic.setValue(connectedStatus());
                connect = false;
        }
        if(doRefresh)
        {
                wifiListCharacteristic.setValue(scanWifi());
                doRefresh = false;
        }
   (((int)dht.getHumidity()) <= 55);

   (((int)dht.getHumidity()) > 55);
}
```

Figura 6.6: Sensores básicos humedad ambiental




```
void loop()
{
        DeviceNameHelperEEPROM::instance().loop();
        if(connect)
        {
                delay(1000);
                wifiSuccessCharacteristic.setValue(connectedStatus());
                connect = false;
        }
        if(doRefresh)
        {
                wifiListCharacteristic.setValue(scanWifi());
                doRefresh = false;
        }
    (((int) ultrasonic.MeasureInCentimeters()) <= 10);

    (((int) ultrasonic.MeasureInCentimeters()) > 10);
}
```

Figura 6.7: Sensores básicos distancia

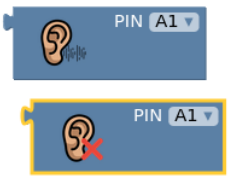



```
void loop()
{
        DeviceNameHelperEEPROM::instance().loop();
        if(connect)
        {
                delay(1000);
                wifiSuccessCharacteristic.setValue(connectedStatus());
                connect = false;
        }
        if(doRefresh)
        {
                wifiListCharacteristic.setValue(scanWifi());
                doRefresh = false;
        }
    ((analogRead(A0) / 4) >= 11);

    ((analogRead(A0) / 4) < 11);
```

Figura 6.8: Sensores básicos luz

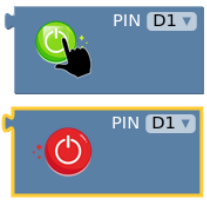

```
void loop()
{
        DeviceNameHelperEEPROM::instance().loop();
        if(connect)
        {
                delay(1000);
                wifiSuccessCharacteristic.setValue(connectedStatus());
                connect = false;
        }
        if(doRefresh)
        {
                wifiListCharacteristic.setValue(scanWifi());
                doRefresh = false;
        }
    ((analogRead(A0) / 4) <= 475);

    ((analogRead(A0) / 4) > 475);
```

Figura 6.9: Sensores básicos humedad suelo



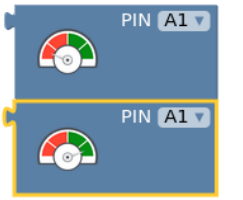

```
void loop()
{
        DeviceNameHelperEEPROM::instance().loop();
        if(connect)
        {
                delay(1000);
                wifiSuccessCharacteristic.setValue(connectedStatus());
                connect = false;
        }
        if(doRefresh)
        {
                wifiListCharacteristic.setValue(scanWifi());
                doRefresh = false;
        }
        mediaRuido(A0,2) >= 120;

        mediaRuido(A0,2) < 120;
}
```

Figura 6.10: Sensores básicos ruido




```
void loop()
{
        DeviceNameHelperEEPROM::instance().loop();
        if(connect)
        {
                delay(1000);
                wifiSuccessCharacteristic.setValue(connectedStatus());
                connect = false;
        }
        if(doRefresh)
        {
                wifiListCharacteristic.setValue(scanWifi());
                doRefresh = false;
        }
    (((int) digitalRead(D2)) == 1);

    (((int) digitalRead(D2)) == 0);
}
```

Figura 6.11: Sensores básicos pulsador

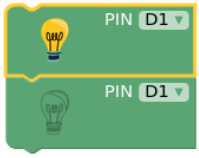



```
void loop()
{
        DeviceNameHelperEEPROM::instance().loop();
        if(connect)
        {
                delay(1000);
                wifiSuccessCharacteristic.setValue(connectedStatus());
                connect = false;
        }
        if(doRefresh)
        {
                wifiListCharacteristic.setValue(scanWifi());
                doRefresh = false;
        }
    ((analogRead(A0) / 4) <= 512);

    ((analogRead(A0) / 4) > 512);
}
```

Figura 6.12: Sensores básicos ángulo

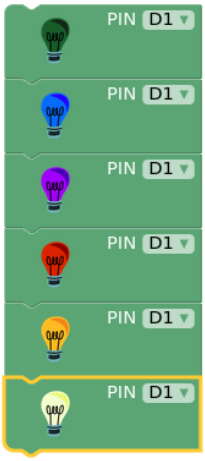
6.2.3. Actuadores versión básica

A continuación, en las Figuras 6.13 a 6.18 se va a mostrar el resultado visual de los sensores de la versión básica y los códigos que los mismos generan



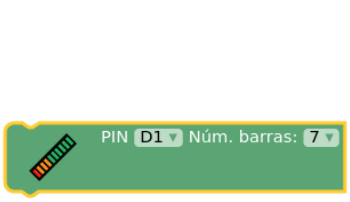

```
void loop()
{
        DeviceNameHelperEEPROM::instance().loop();
        if(connect)
        {
                delay(1000);
                wifiSuccessCharacteristic.setValue(connectedStatus());
                connect = false;
        }
        if(doRefresh)
        {
                wifiListCharacteristic.setValue(scanWifi());
                doRefresh = false;
        }
    digitalWrite(D2,HIGH);
    digitalWrite(D2,LOW);
}
```

Figura 6.13: Actuadores básicos led simple



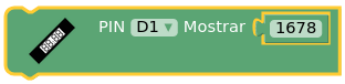

```
void loop()
{
        DeviceNameHelperEEPROM::instance().loop();
        if(connect)
        {
                delay(1000);
                wifiSuccessCharacteristic.setValue(connectedStatus());
                connect = false;
        }
        if(doRefresh)
        {
                wifiListCharacteristic.setValue(scanWifi());
                doRefresh = false;
        }
    leds.setColorRGB(0,0, 255, 0);
    leds.setColorRGB(0,51, 51, 255);
    leds.setColorRGB(0,204, 51, 204);
    leds.setColorRGB(0,204, 0, 0);
    leds.setColorRGB(0,255, 102, 0);
    leds.setColorRGB(0,255, 255, 0);
}
```

Figura 6.14: Actuadores básicos led variable



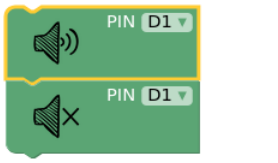

```
void loop()
{
        DeviceNameHelperEEPROM::instance().loop();
        if(connect)
        {
                delay(1000);
                wifiSuccessCharacteristic.setValue(connectedStatus());
                connect = false;
        }
        if(doRefresh)
        {
                wifiListCharacteristic.setValue(scanWifi());
                doRefresh = false;
        }
 }
```

Figura 6.15: Actuador básico barra led




```
void loop()
{
        DeviceNameHelperEEPROM::instance().loop();
        if(connect)
        {
                delay(1000);
                wifiSuccessCharacteristic.setValue(connectedStatus());
                connect = false;
        }
        if(doRefresh)
        {
                wifiListCharacteristic.setValue(scanWifi());
                doRefresh = false;
        }
        display.showNumberDec(1678);
}
```

Figura 6.16: Actuador básico pantalla dígitos

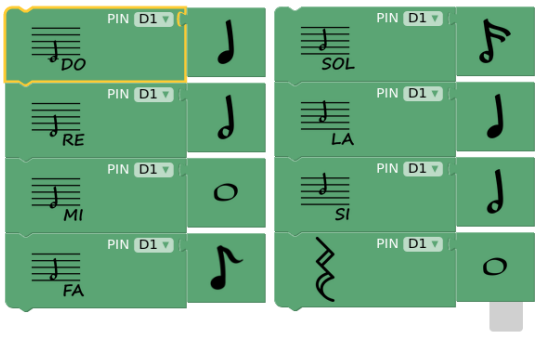



```
void loop()
{
        DeviceNameHelperEEPROM::instance().loop();
        if(connect)
        {
                delay(1000);
                wifiSuccessCharacteristic.setValue(connectedStatus());
                connect = false;
        }
        if(doRefresh)
        {
                wifiListCharacteristic.setValue(scanWifi());
                doRefresh = false;
        }
    digitalWrite(D2, HIGH);
    digitalWrite(D2, LOW);
}
```

Figura 6.17: Actuadores básicos zumbador

6.3. O3: Mejoras en la aplicación

6.3.1. Colección musical




```
void loop()
{
        DeviceNameHelperEEPROM::instance().loop();
        if(connect)
        {
                delay(1000);
                wifiSuccessCharacteristic.setValue(connectedStatus());
                connect = false;
        }
        if(doRefresh)
        {
                wifiListCharacteristic.setValue(scanWifi());
                doRefresh = false;
        }
    tone(D2, 261, );
    delay();
    noTone(D2);
    tone(D2, 293, );
    delay();
    noTone(D2);
    tone(D2, 329, );
    delay();
    noTone(D2);
    tone(D2, 349, );
    delay();
    noTone(D2);
    tone(D2, 392, );
    delay();
    noTone(D2);
}
```

Figura 6.18: Bloque notas y figuras versión básica

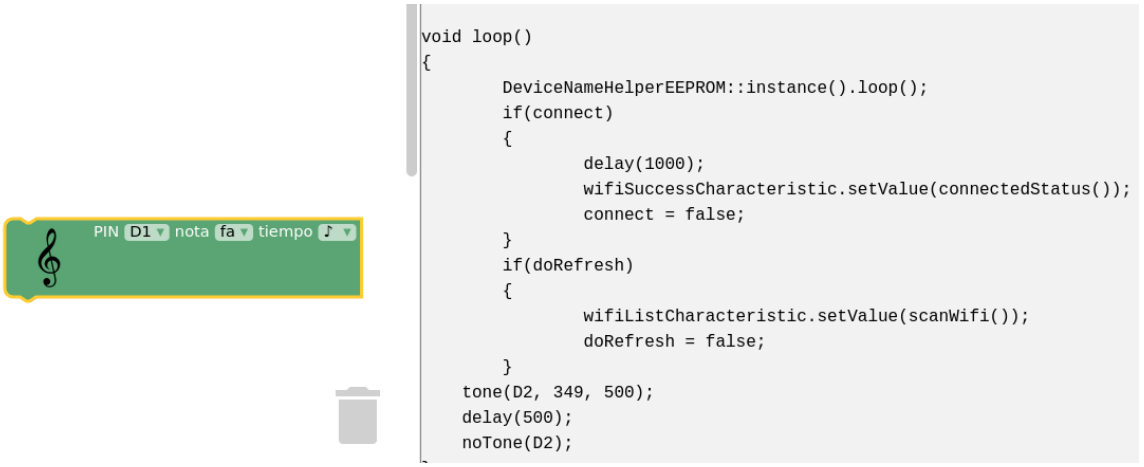


Figura 6.19: Bloque notas versión avanzada

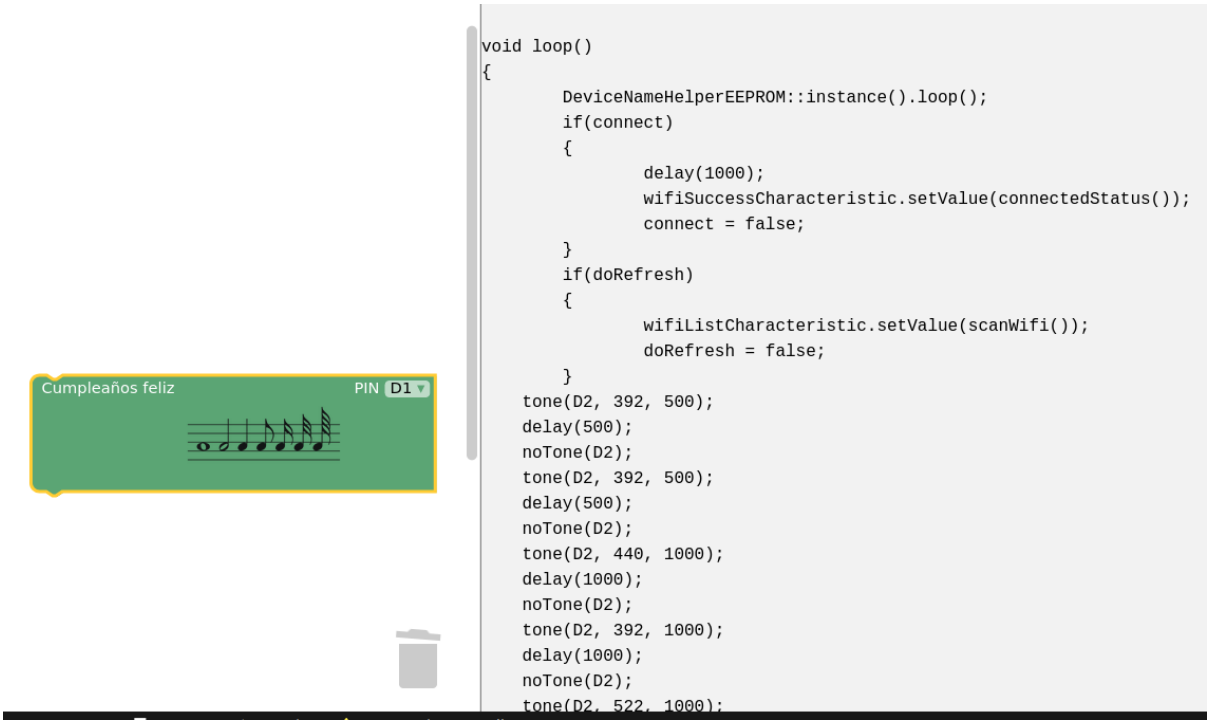


Figura 6.20: Bloque melodía

6.3.2. Dashboard

Para el dashboard se habían propuesto varias mejoras:

- Añadir etiqueta en los proyectos para determinar la versión de Sucre.
- Cambiar la imagen de la card de proyecto para los proyectos de la versión básica, añadiendo una imagen más colorida que proporcionase una imagen más infantil.
- Ordenar los proyectos por fecha y hora de modificación, poniendo delante los modificados más recientemente.

En la Figura 6.21 se pueden observar los resultados de todas estas mejoras de la aplicación.

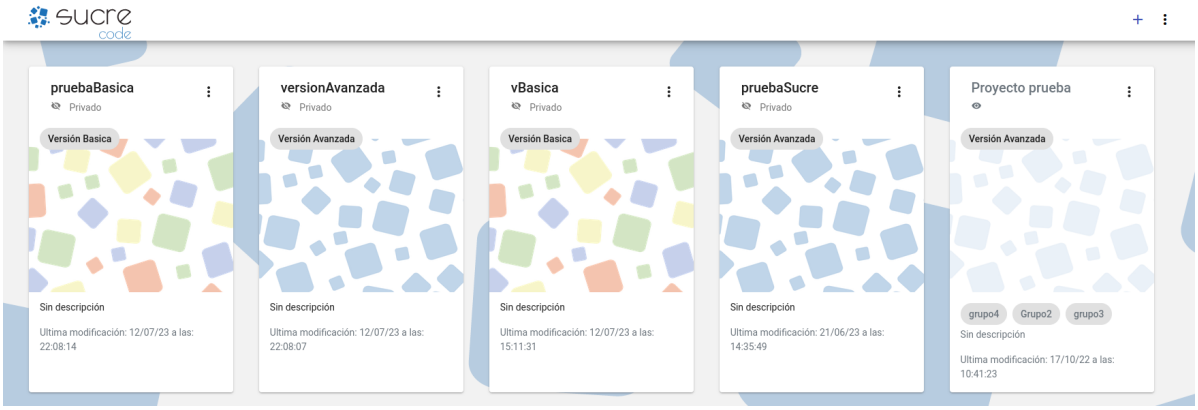
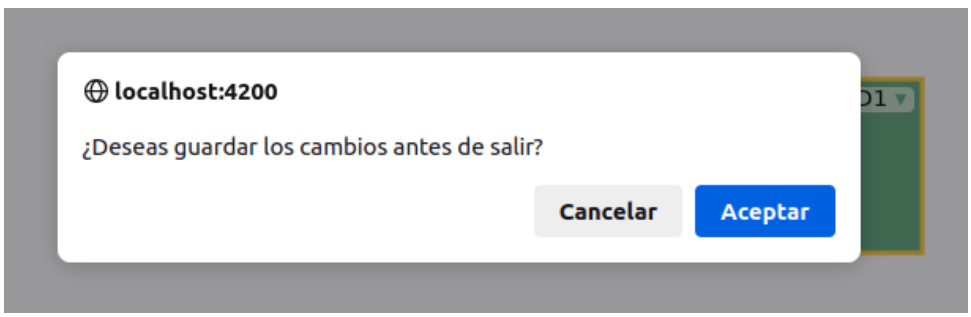


Figura 6.21: Mejoras dashboard

6.3.3. Workspace proyectos

En los proyectos se pretendía crear opciones que almacenaran el montaje de bloques actual. También se debía de poder restaurar el último montaje almacenado. Al salir del proyecto debía pedir confirmación para guardar los cambios. En las Figuras 6.22a y 6.22b se muestran los resultados de estas mejoras.

(a) Guardar/-Cargar



(b) Confirmación de guardado al salir

Figura 6.22: Cambios en el workspace

6.3.4. Descargar elementos tabla en fichero csv

Actualmente no se puede mostrar la tabla, pues ha caducado la versión gratuita de la base de datos Influx, donde estaban almacenadas las variables medidas por cada SucreCore. En la Figura 6.23 se muestra lo que la aplicación mostraría, y a continuación, en las Figuras 6.24 y 6.25 se muestran dos ejemplos de los últimos ficheros csv descargados en el proceso de pruebas, antes de que caducase la versión gratuita.

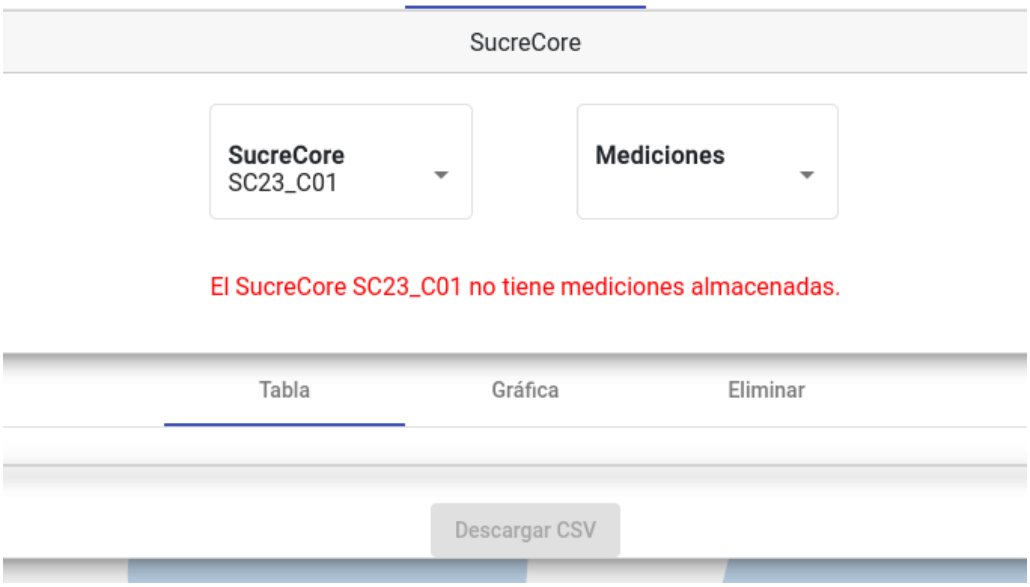
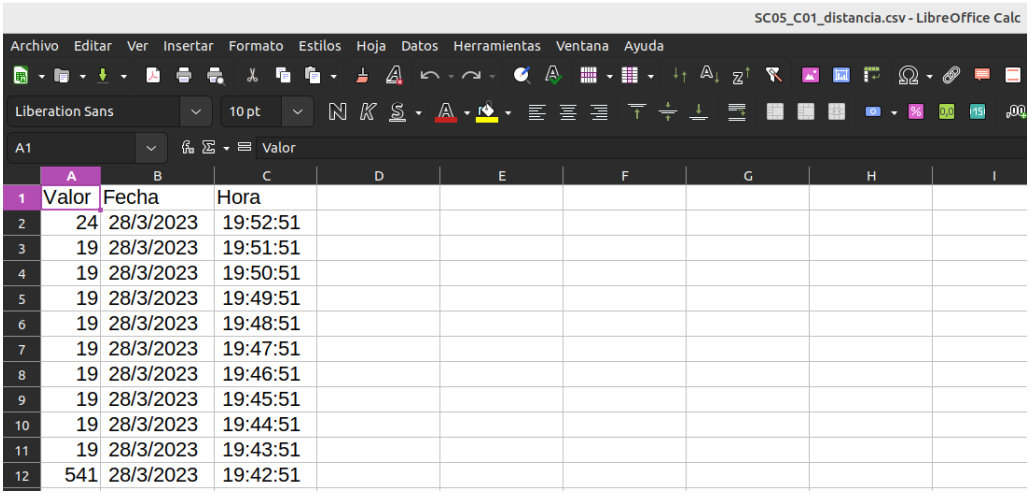


Figura 6.23: Descargar elementos almacenados en fichero csv



	A	B	C	D	E	F	G	H	I
1	Valor	Fecha	Hora						
2	24	28/3/2023	19:52:51						
3	19	28/3/2023	19:51:51						
4	19	28/3/2023	19:50:51						
5	19	28/3/2023	19:49:51						
6	19	28/3/2023	19:48:51						
7	19	28/3/2023	19:47:51						
8	19	28/3/2023	19:46:51						
9	19	28/3/2023	19:45:51						
10	19	28/3/2023	19:44:51						
11	19	28/3/2023	19:43:51						
12	541	28/3/2023	19:42:51						

Figura 6.24: Ejemplo de fichero csv de la variable distancia

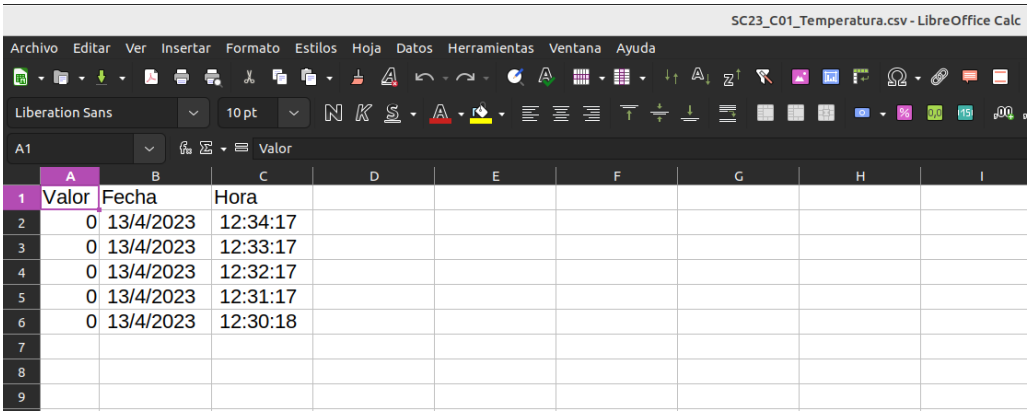


Figura 6.25: Ejemplo de fichero csv de la variable temperatura

6.3.5. Descargar gráfica en pdf

Sucre4STEM permite la posibilidad de mostrar en una gráfica una o más variables almacenadas de un mismo SucreCore o de diferentes.

Al igual que ocurre en el apartado anterior, actualmente no es posible mostrar el resultado de la aplicación. Sin embargo, en la Figura 6.26 se muestra la interfaz que mostraría la gráfica en la aplicación en caso de tener variables almacenadas.

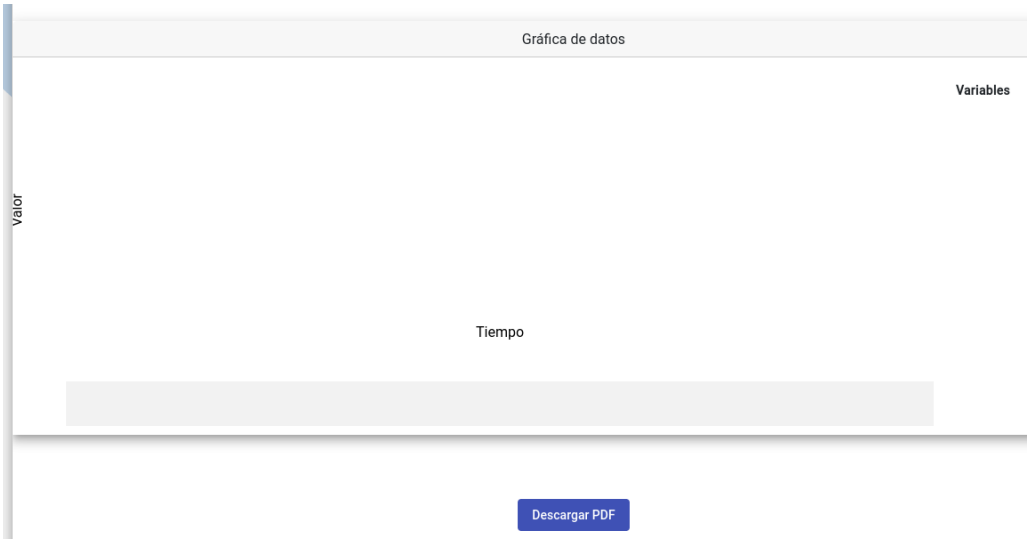


Figura 6.26: Botón descargar gráfica en pdf

En las Figuras 6.27 y 6.28 se muestran ejemplos de los últimos documentos pdf descargados en el proceso de pruebas, antes de que la versión gratuita de la base de datos Influx caducase.

Figura 6.27: Ejemplo documento pdf de una gráfica descargado

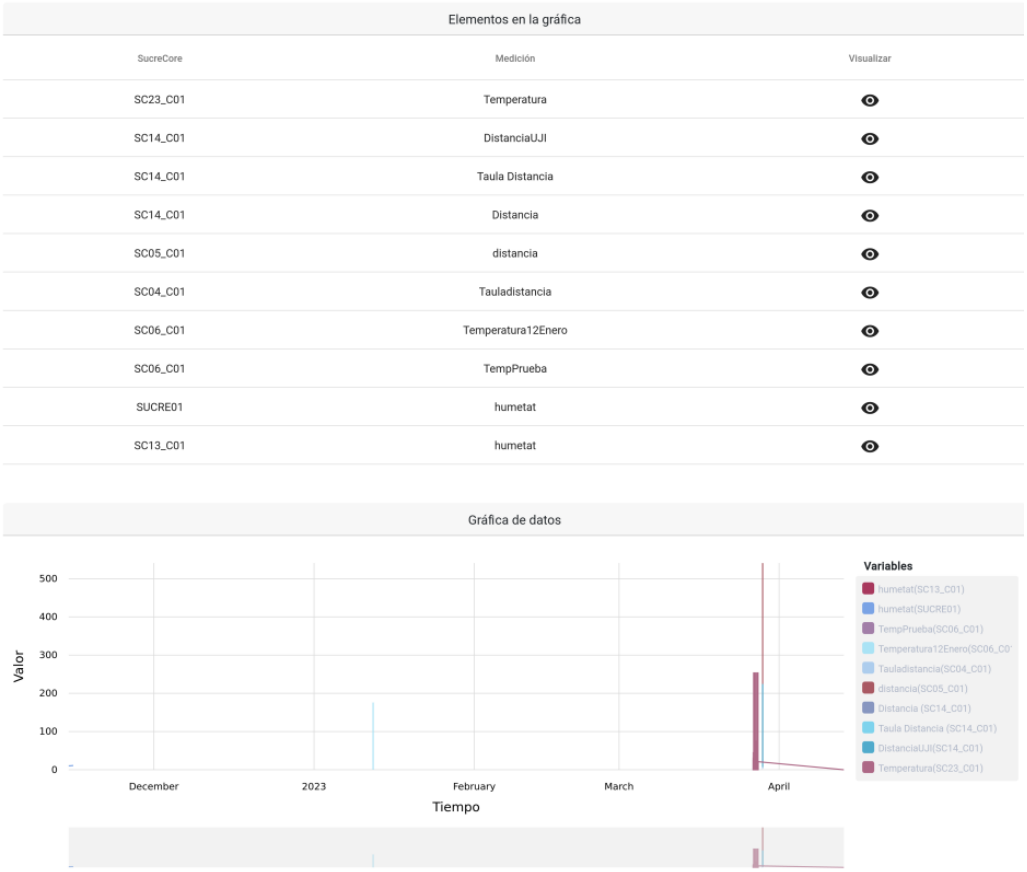


Figura 6.28: Ejemplo documento pdf descargado que muestra los valores de diferentes variables de los diversos SucreCores

6.3.6. Cambiar nombre de usuario y versión se Sucre

En la Figura 6.29 se muestra la interfaz que permite a un usuario cambiar su nombre y versión de Sucre.

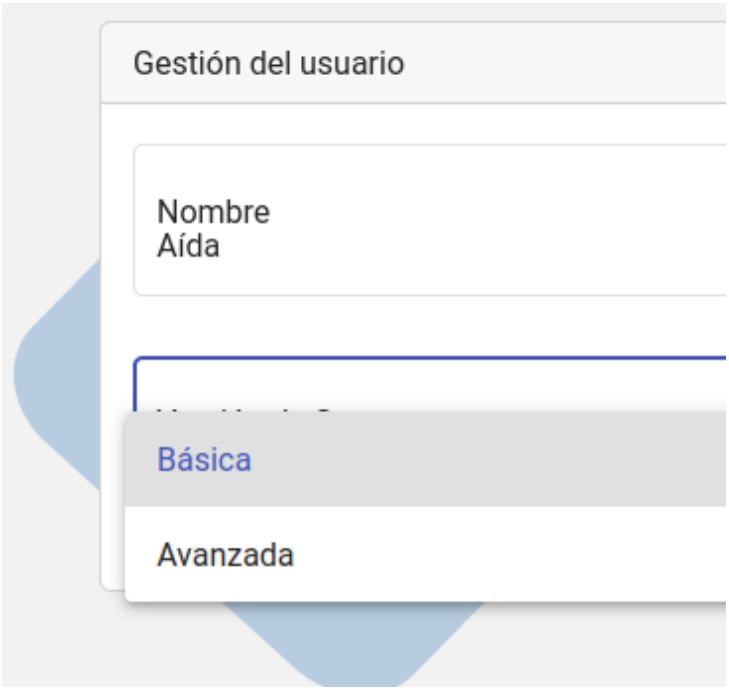


Figura 6.29: Cambiar nombre usuario y versión de Sucre

Capítulo 7

Conclusiones y trabajo futuro

En esta sección se van a describir las conclusiones desde diferentes puntos de vista. Por un lado se describirán los problemas encontrados durante el proyecto, luego se redactarán las conclusiones, tanto del proyecto como la valoración personal de la estancia, y por último se propondrá posible trabajo futuro para la aplicación.

7.1. Problemas encontrados

A lo largo de la estancia, me he topado con varios problemas. La primera barrera fue el software, ya que la aplicación requiere diversos lenguajes y aplicaciones para su funcionamiento, las cuales desconocía como Angular o Firebase. Por ejemplo, este último menaje bases de datos no relacionales, cuando en la carrera había visto bases relacionales. Al tratarse de una aplicación ya en uso, no servía simplemente con aprender a crear aplicaciones con Angular o a construir bases de datos con Firebase, pues la aplicación tenía ya muchas funcionalidades implementadas a las que me tuve que adaptar.

Por otra parte, una de las funcionalidades que se quería implementar era la creación de una opción que guardase los bloques ensamblados en ese momento, para posteriormente poder seleccionar cuál de ellos queríamos restaurar. Esta funcionalidad hubo que readaptarla, ya que, el hecho de guardar versiones daba más problemas que ventajas, pues una vez almacenada una versión era difícil mostrarle al usuario cuál era esa versión, pues un simple nombre podía no describirla adecuadamente. Finalmente se revisó y la funcionalidad desarrollada pasó a ser guardar el montaje actual y restaurar el último montaje almacenado.

Otro problema estuvo relacionado con las funcionalidades de descargar archivo csv y pdf, concretamente surgió una vez implementadas ambas funcionalidades. La base de datos influx, que utilizaban para almacenar las variables, tenía un periodo de prueba gratuito. Pasado el periodo gratuito la empresa no pagó por mantenerla.

Por tanto, una vez desarrollada la funcionalidad para descargar pdf y csv con las gráficas y tablas no se pudo mejorar, ni se han podido mostrar los resultados de dichas funcionalidades en la memoria, pues actualmente no hay ninguna base de datos con las variables almacenadas.

El tiempo de espera para realizar cada prueba fue otro problema muy grave. Cada prueba que se deseaba realizar requería varios minutos para actualizar el SucreCore y mostrar los resultados. Además los SucreCores se desconectan con frecuencia y necesitaban bastante tiempo para volver a estar conectados a la red. En ocasiones requerían un reset, lo cual también incrementaba el tiempo de espera.

7.2. Conclusiones

Respecto al proyecto, la planificación se llevó al día, salvo la pequeña readaptación de guardar versiones mencionada en la sección anterior. Todos los objetivos se han alcanzado y los resultados obtenidos han sido satisfactorios.

A partir de la creación de bloques de la versión básica, se ha logrado una aplicación más usable, que permite a los más pequeños usar una versión adaptada a sus edades y conocimientos. Con las mejoras de la versión avanzada de Sucre, los alumnos podrán ensamblar únicamente montajes válidos, con lo que se evitarán errores, además podrán usar el sensor de ruido de una manera más cómoda, ya que observarán valores más estables. Finalmente, las mejoras comunes a ambas versiones han logrado una aplicación estéticamente más agradable, a la vez que se ha dotado al usuario de mayor flexibilidad.

Gracias a los objetivos alcanzados y resultados obtenidos, se ha podido contribuir al desarrollo de un artículo de investigación que está en proceso de revisión.

7.2.1. Valoración personal

En cuanto al ámbito profesional considero que esta experiencia ha sido enriquecedora, pues normalmente en las asignaturas se nos enseña a crear aplicaciones desde cero y diseñarlas y construirlas hasta el final, implementando todas las funcionalidades requeridas. En este caso, durante la estancia me he adaptado a un proyecto ya existente, realizando modificaciones y añadiendo funcionalidades al mismo. Por tanto esto ha aportado una dosis de realidad a la estancia en prácticas, pues muchas veces cuando llegas a una empresa debes adaptarte a proyectos que ya están en marcha.

En cuanto a mi experiencia, realmente me he sentido agradecida con los trabajadores de la empresa, pues siempre que he necesitado ayuda me la han proporcionado de inmediato, bien proporcionándome los materiales de consulta necesarios, o bien estando a mi lado enseñándome y ayudándome personalmente. Además, a pesar de mi falta de experiencia en el sector, puesto que soy una estudiante de prácticas, en todo momento me han tratado con mucho respeto y han escuchado con interés mis ideas para mejorar la aplicación, lo cual me ha hecho sentir como un miembro más de la empresa.

Personalmente me ha encantado formar parte de este proyecto, mi aspiración es realizar el máster en educación y ser docente de informática, con lo que trabajar en un proyecto cuyo objetivo es promover entre el estudiantado vocaciones científicas creo que es la mejor estancia que me habría podido tocar.

7.3. Trabajo futuro

En un futuro, se podría ampliar la aplicación incluyendo colecciones de bloques para otras materias, esta aplicación fomenta el pensamiento computacional, realizando experimentos que podrían englobarse en el área de biología, física y química o tecnología. Dado que actualmente en educación se están imponiendo las competencias, creo que sería un buen momento para ampliar Sucre y generar un conjunto de bloques por asignatura, de modo que en cualquier asignatura se fomentasen las TIC y el pensamiento computacional. Se podría por ejemplo generar bloques para la asignatura de geografía que permitiesen montar mapas o para la asignatura de matemáticas para resolución de ecuaciones.

Otra versión podría adaptarse a alumnos con necesidades educativas especiales.

Apéndice A

Fragmentos de código implementados

En este apéndice se mostrarán los fragmentos de código implementados.

A.1. O1: Mejoras de la versión actual de Sucre4STEM

```
Blockly.Blocks['sensor_temperatura'] =
{
    init: function()
    {
        this.appendDummyInput()
            .appendField(new Blockly.FieldImage("./assets/media/
    sensor_temperatura.png",
            64, 50, "*"))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.digital), "
    PIN");
        this.setOutput(true, 'Number');
        this.setColour(210);
        this.setTooltip("Este sensor sirve para medir la temperatura. Tipo de
    sensor: digital. Rango de valores: 0-50 grados(°)");
        this.setHelpUrl("http://www.sucre.uji.es/docs/sensores/#temperatura-
    humedad");
    }
};
```

Código A.1: Corrección incoherencia datos

A.2. O2: Creación de bloques de programación para la versión básica

A.2.1. Sensor de temperatura

Bloque frío:

```
Blockly.Dart['frio'] = function(block)
{
```

```
    var dropdown_pin = block.getFieldValue('PIN');

    Blockly.Dart.definitions_['define_Adafruit_DHT'] = '#include "Adafruit_DHT.
    h"';
    Blockly.Dart.definitions_['define_Adafruit_pin'] = '#define DHTPIN ' +
    dropdown_pin;

    Blockly.Dart.definitions_['define_Adafruit_DHT11'] = '#define DHTTYPE DHT11
    ';

    Blockly.Dart.definitions_['define_DHT'] = 'DHT dht(DHTPIN, DHTTYPE);';

    Blockly.Dart.setups_['setup_dhtbegin_'] = 'dht.begin();';
    var code = '(((int)dht.getTempCelcius()) < 25)';


    return [code, Blockly.Dart.ORDER_ATOMIC];
};
```

Código A.2: Código bloque frío en generators/dart/grove.js

Aspecto bloque frío:

```
Blockly.Blocks['frio'] =
{
    init: function()
    {
        this.appendDummyInput()
            .appendField(new Blockly.FieldImage("./assets/media/sensores/
    temperatura/frio.png", 64, 50, "*"))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.digital), "
    PIN");
        this.setOutput(true, 'Boolean');
        this.setColour(210);
        this.setTooltip("FRIO. Este sensor sirve para medir la temperatura, en
    particular para medir si hace frio. Tipo de sensor: digital. Rango de
    valores: true/false, [0-25)°C = true,  [25-50]°C = false");
        this.setHelpUrl("http://www.sucre.uji.es/docs/sensores/#temperatura-
    humedad");
    }
};
```

Código A.3: Código bloque frío en blocks/grove.js

Bloque poca humedad:

```
Blockly.Dart['humedad_aire_poca'] = function(block)
{
    var dropdown_pin = block.getFieldValue('PIN');

    Blockly.Dart.definitions_['define_Adafruit_DHT'] = '#include "Adafruit_DHT.
    h"';
    Blockly.Dart.definitions_['define_Adafruit_pin'] = '#define DHTPIN ' +
    dropdown_pin;

    Blockly.Dart.definitions_['define_Adafruit_DHT11'] = '#define DHTTYPE DHT11
    ';

    Blockly.Dart.definitions_['define_DHT'] = 'DHT dht(DHTPIN, DHTTYPE);';
```

```
11
12      Blockly.Dart.setups_['setup_dhtbegin_'] = 'dht.begin();';
13
14      var code = '(((int)dht.getHumidity()) <= 55)';
15
16      return [code, Blockly.Dart.ORDER_ATOMIC]
17  };
```

Código A.4: Código bloque poca humedad aire en generators/dart/grove.js

Aspecto bloque poca humedad aire:

```
1  Blockly.Blocks['humedad_aire_poca'] =
2  {
3      init: function()
4      {
5          this.appendDummyInput()
6              .appendField(new Blockly.FieldImage("./assets/media/sensores/
       humedadAire/humedadAirePoca.png", 64, 50, "*"))
7              .appendField("PIN")
8              .appendField(new Blockly.FieldDropdown(profile.default.digital), "
       PIN");
9          this.setOutput(true, 'Boolean');
10         this.setColour(210);
11         this.setTooltip("CLIMA SECO. Este sensor sirve para medir el nivel de
       humedad del aire. Tipo de sensor: digital. Rango de valores: 20 - 90%");
12         this.setHelpUrl("http://www.sucre.uji.es/docs/sensores/#temperatura-
       humedad");
13     }
14 };
```

Código A.5: Código bloque poca humedad en blocks/grove.js

Bloque mucha humedad aire:

```
1  Blockly.Dart['humedad_aire_demasiada'] = function(block)
2  {
3      var dropdown_pin = block.getFieldValue('PIN');
4
5      Blockly.Dart.definitions_['define_Adafruit_DHT'] = '#include "Adafruit_DHT.
       h"';
6      Blockly.Dart.definitions_['define_Adafruit_pin'] = '#define DHTPIN ' +
       dropdown_pin;
7
8      Blockly.Dart.definitions_['define_Adafruit_DHT11'] = '#define DHTTYPE DHT11
       ';
9
10     Blockly.Dart.definitions_['define_DHT'] = 'DHT dht(DHTPIN, DHTTYPE);';
11
12     Blockly.Dart.setups_['setup_dhtbegin_'] = 'dht.begin();';
13
14     var code = '(((int)dht.getHumidity()) < 55)';
15
16     return [code, Blockly.Dart.ORDER_ATOMIC]
17 };
```

Código A.6: Código bloque mucha humedad en generators/dart/grove.js

Aspecto bloque mucha humedad aire:

```
Blockly.Blocks['humedad_aire_demasiada'] =
{
    init: function()
    {
        this.appendDummyInput()
            .appendField(new Blockly.FieldImage("./assets/media/sensores/humedadAire/humedadAireMucha.png", 64, 50, "*"))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.digital), "PIN");
        this.setOutput(true, 'Boolean');
        this.setColour(210);
        this.setTooltip("DEMASIADA HUMEDAD AIRE. Este sensor sirve para medir el nivel de humedad del aire. Tipo de sensor: digital. Rango de valores: 20 - 90%");
        this.setHelpUrl("http://www.sucre.uji.es/docs/sensores/#temperatura-humedad");
    }
};
```

Código A.7: Código bloque mucha humedad en blocks/grove.js

A.2.2. Sensor de distancia

Bloque cerca:

```
Blockly.Dart['cerca'] = function(block)
{
    var dropdown_pin = this.getFieldValue('PIN');
    Blockly.Dart.definitions_['define_ultrasonic'] = '#include "Grove-Ultrasonic-Ranger.h"';
    Blockly.Dart.definitions_['var_ultrasonic'] = 'Ultrasonic ultrasonic(' + dropdown_pin + ');';
    var code = '(((int) ultrasonic.MeasureInCentimeters()) <= 10)';

    return [code, Blockly.Dart.ORDER_ATOMIC];
};
```

Código A.8: Código bloque cerca en generators/dart/grove.js

Aspecto bloque cerca:

```
Blockly.Blocks['cerca'] =
{
    helpUrl: 'http://www.sucre.uji.es/docs/sensores/#distancia',
    init: function()
    {
        this.setColour(210);
        this.appendDummyInput()
            .appendField(new Blockly.FieldImage("./assets/media/sensores/distancia/cerca.png", 64, 64))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.digital), "PIN")
        this.setOutput(true, 'Boolean');
        this.setTooltip('CERCA. Este sensor sirve para medir la distancia entre el sensor y un objeto. Se puede utilizar para conocer la proximidad de un objeto y as  aplicar cualquier tipo de acci n. Ejemplo de esto puede ser un sensor de aparcamiento. Tipo de sensor: digital. Rango de valores: 0-450 cm');
```

```
13      }
```

Código A.9: Código sensor cerca en blocks/grove.js

Bloque lejos:

```
1  Blockly.Dart['lejos'] = function(block)
2  {
3      var dropdown_pin = this.getFieldValue('PIN');
4      Blockly.Dart.definitions_['define_ultrasonic'] = '#include "Grove-
       Ultrasonic-Ranger.h"';
5      Blockly.Dart.definitions_['var_ultrasonic'] = 'Ultrasonic ultrasonic(' +
       dropdown_pin + ');';
6      var code = '(((int) ultrasonic.MeasureInCentimeters()) > 10)';
7
8      return [code, Blockly.Dart.ORDER_ATOMIC];
9  };
```

Código A.10: Código bloque lejos en generators/dart/grove.js

Aspecto bloque lejos:

```
1  Blockly.Blocks['lejos'] =
2  {
3      helpUrl: 'http://www.sucre.uji.es/docs/sensores/#distancia',
4      init: function()
5      {
6          this.setColour(210);
7          this.appendDummyInput()
8              .appendField(new Blockly.FieldImage("./assets/media/sensores/
       distancia/lejos.png", 64, 64))
9              .appendField("PIN")
10             .appendField(new Blockly.FieldDropdown(profile.default.digital), "
       PIN")
11         this.setOutput(true, 'Boolean');
12         this.setTooltip('LEJOS. Este sensor sirve para medir la distancia entre
        el sensor y un objeto. Se puede utilizar para conocer la proximidad de un
       objeto y as  aplicar cualquier tipo de acci n. Ejemplo de esto puede ser
       un sensor de aparcamiento. Tipo de sensor: digital. Rango de valores: 0-450
       cm');
13     }
14 };
```

Código A.11: Código bloque lejos en blocks/grove.js

A.2.3. Sensor de luz

Bloque luz:

```
1  Blockly.Dart['luz'] = function(block)
2  {
3      var dropdown_pin = this.getFieldValue('PIN');
4      Blockly.Dart.setups_['setup_button_' + dropdown_pin] = 'pinMode(' +
       dropdown_pin + ', INPUT);';
5
6      var code = '((analogRead(' + dropdown_pin + ') / 4) >= 11)';
7      return [code, Blockly.Dart.ORDER_ATOMIC];
8  };
```

Código A.12: Código bloque luz en generators/dart/grove.js

Aspecto bloque luz:

```
Blockly.Blocks['luz'] =
{
    helpUrl: 'http://www.sucre.uji.es/docs/sensores/#luz',
    init: function()
    {
        this.setColour(210);
        this.appendDummyInput()
            .appendField(new Blockly.FieldImage("./assets/media/sensores/luz/luz.png", 64, 64))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.analog), "PIN");
        this.setOutput(true, 'Boolean');
        this.setTooltip('LUZ: Este sensor sirve para determinar que hay suficiente luz en un espacio. Se puede utilizar para conocer si hay falta de luminosidad y as  encender la luz o no. Tipo de sensor: anal gico. Rango de valores: true/false. No luz: 0-10 = false. Luz: 11-625 = true');
    }
};
```

Código A.13: Código bloque luz en blocks/grove.js

Bloque no luz:

```
Blockly.Dart['noLuz'] = function(block)
{
    var dropdown_pin = this.getFieldValue('PIN');
    Blockly.Dart.setups_['setup_button_' + dropdown_pin] = 'pinMode(' + dropdown_pin + ', INPUT);';

    var code = '((analogRead(' + dropdown_pin + ') / 4) < 11)';
    return [code, Blockly.Dart.ORDER_ATOMIC];
};
```

Código A.14: Código bloque no luz en generators/dart/grove.js

Aspecto bloque no luz:

```
Blockly.Blocks['noLuz'] =
{
    helpUrl: 'http://www.sucre.uji.es/docs/sensores/#luz',
    init: function()
    {
        this.setColour(210);
        this.appendDummyInput()
            .appendField(new Blockly.FieldImage("./assets/media/sensores/luz/noLuz.png", 64, 64))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.analog), "PIN");
        this.setOutput(true, 'Boolean');
        this.setTooltip('NO LUZ: Este sensor sirve para determinar que hay suficiente luz en un espacio. Se puede utilizar para conocer si hay falta de luminosidad y as  encender la luz o no. Tipo de sensor: anal gico. Rango de valores: true/false. No luz: 0-10 = false. Luz: 11-625 = true');
    }
};
```

Código A.15: Código bloque no luz en blocks/grove.js

A.2.4. Sensor de humedad en el suelo

Bloque suelo seco:

```
Blockly.Dart['suelo_seco'] = function(block)
{
    var dropdown_pin = this.getFieldValue('PIN');
    var code = '((analogRead(' +dropdown_pin+ ') / 4) <= 475)';
    return [code, Blockly.Dart.ORDER_ATOMIC]
};
```

Código A.16: Código bloque suelo seco en generators/dart/grove.js

Aspecto bloque suelo seco:

```
Blockly.Blocks['suelo_seco'] =
{
    init: function()
    {
        this.appendDummyInput()
            .appendField(new Blockly.FieldImage("./assets/media/sensores/
    humedadSuelo/humedadSueloPoca.png", 64, 50, "*"))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.analog), "
    PIN")
        this.setOutput(true, 'Boolean');
        this.setColour(210);
        this.setTooltip("SUELO SECO. Este sensor sirve para medir el nivel de
    humedad del suelo. Se puede utilizar para conocer si una planta necesita ser
     regada. Tipo de sensor: analogico. Rango de valores: 380-620. Seco:
    620-530. Humedo: 531-430. Agua: 431-380");
        this.setHelpUrl("http://www.sucre.uji.es/docs/sensores/#humedad");
    }
};
```

Código A.17: Código bloque suelo seco en blocks/grove.js

Bloque suelo agua:

```
Blockly.Dart['suelo_agua'] = function(block)
{
    var dropdown_pin = this.getFieldValue('PIN');
    var code = '((analogRead('+dropdown_pin+') / 4) > 475)';
    return [code, Blockly.Dart.ORDER_ATOMIC]
};
```

Código A.18: Código bloque suelo agua en generators/dart/grove.js

```
Blockly.Blocks['suelo_agua'] =
{
    init: function()
    {
        this.appendDummyInput()
            .appendField(new Blockly.FieldImage("./assets/media/sensores/
    humedadSuelo/humedadSueloMucha.png", 64, 50, "*"))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.analog), "
    PIN")
        this.setOutput(true, 'Boolean');
        this.setColour(210);
```

```
        this.setTooltip("SUELO LIQUIDO. Este sensor sirve para medir el nivel
    de humedad del suelo. Se puede utilizar para conocer si una planta necesita
    ser regada. Tipo de sensor: analogico. Rango de valores: 380-620. Seco:
    620-530. Humedo: 531-430. Agua: 431-380");
        this.setHelpUrl("http://www.sucre.uji.es/docs/sensores/#humedad");
    }
};
```

Código A.19: Código bloque suelo agua en blocks/grove.js

A.2.5. Sensor de ruido

Bloque ruido:

```
Blockly.Dart['ruido'] = function(block)
{
    var dropdown_pin = this.getFieldValue('PIN');
    var umbral = 120;
    var tiempo = 2;

    Blockly.Dart.definitions_['define_ruido'] = '#include "NtpTime.h"';
    Blockly.Dart.definitions_['sensor_ruido'] = '\n\nint mediaRuido(int
    dropdown_pin, int tiempo)\n{\n\tint acum = 0;\n\tint n = 0;\n\tNtpTime
    t_time;\n\tunsigned long tInicio = t_time.now();\n\twhile((int) (t_time.now
    () - tInicio) < tiempo)\n\t{\n\t\tint code = analogRead(dropdown_pin) / 4;\n
    \t\tacum += code;\n\t\tn+=1;\n\t}\n\tint res = acum/n;\n\treturn res;\n}\n\n
    ';

    var code = "\tmediaRuido("+dropdown_pin+","+tiempo+") >= "+umbral;
    return [code, Blockly.Dart.ORDER_ATOMIC]
};
```

Código A.20: Código bloque ruido en generators/dart/grove.js

Aspecto bloque ruido:

```
Blockly.Blocks['ruido'] =
{
    init: function()
    {
        this.appendDummyInput()
            .appendField(new Blockly.FieldImage("./assets/media/sensores/ruido/
    sonido.png", 64, 50, "*"))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.analog), "
    PIN")
        this.setOutput(true, 'Boolean');
        this.setColour(210);
        this.setTooltip("RUIDO AMBIENTAL. Este sensor sirve para medir la
    cantidad de ruido en el  ambiente. Un ejemplo de uso puede ser captar el
    ruido que producen las personas  o animales en un entorno y as  saber si
    hay  alguien. Tipo de sensor: anal gico. Rango de valores: 0-1024");
        this.setHelpUrl("http://www.sucre.uji.es/docs/sensores/#ruido");
    }
};
```

Código A.21: Código bloque ruido en blocks/grove.js

Bloque silencio:

```
Blockly.Dart['silencio'] = function(block)
{
    var umbral = 120;
    var tiempo = 2;
    var dropdown_pin = this.getFieldValue('PIN');

    Blockly.Dart.definitions_['define_ruido'] = '#include "NtpTime.h"'
    Blockly.Dart.definitions_['sensor_ruido'] = '\n\nint mediaRuido(int
    dropdown_pin, int tiempo)\n{\n\tint acum = 0;\n\tint n = 0;\n\tNtpTime
    t_time;\n\tunsigned long tInicio = t_time.now();\n\twhile((int) (t_time.now
    () - tInicio) < tiempo)\n\t{\n\t\tint code = analogRead(dropdown_pin) / 4;\n
    \t\tacum += code;\n\t\tn+=1;\n\t}\n\tint res = acum/n;\n\treturn res;\n}\n\n
    ';

    var code = "\tmediaRuido("+dropdown_pin+","+tiempo+") < "+umbral;
    return [code, Blockly.Dart.ORDER_ATOMIC]
};
```

Código A.22: Código bloque silencio en generators/dart/grove.js

Aspecto bloque silencio:

```
Blockly.Blocks['silencio'] =
{
    init: function()
    {
        this.appendDummyInput()
            .appendField(new Blockly.FieldImage("./assets/media/sensores/ruido/
    silencio.png", 64, 50, "*"))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.analog), "
    PIN")
        this.setOutput(true, 'Boolean');
        this.setColour(210);
        this.setTooltip("SILENCIO. Este sensor sirve para medir la cantidad de
    ruido en el  ambiente. Un ejemplo de uso puede ser captar el ruido que
    producen las personas  o animales en un entorno y as  saber si hay  alguien
    . Tipo de sensor: anal gico. Rango de valores: 0-1024");
        this.setHelpUrl("http://www.sucre.uji.es/docs/sensores/#ruido");
    }
};
```

Código A.23: Código bloque silencio en gblocks/grove.js

A.2.6. Sensor de pulsación

Bloque botón pulsado:

```
Blockly.Dart['boton_pulsado'] = function(block)
{
    var dropdown_pin = this.getFieldValue('PIN');
    Blockly.Dart.setups_['setup_button_' + dropdown_pin] = 'pinMode(' +
    dropdown_pin + ', INPUT);';
    var code = '(((int) digitalRead(' + dropdown_pin + ')) == 1)';
    return [code, Blockly.Dart.ORDER_ATOMIC];
};
```

Código A.24: Código bloque botón pulsado en generators/dart/grove.js

Aspecto bloque botón pulsado:

```
Blockly.Blocks['boton_pulsado'] =
{
    helpUrl: 'http://www.sucre.uji.es/docs/sensores/#boton',
    init: function()
    {
        this.setColour(210);
        this.appendDummyInput()
            .appendField(new Blockly.FieldImage("./assets/media/sensores/boton/pulsado.png", 64, 64))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.digital), "PIN");
        this.setOutput(true, 'Boolean');
        this.setTooltip('BOTON PULSADOEste sensor facilita la interacci n mediante pulsaciones. Cuando este es presionado obtiene un valor HIGH, y cuando se libera, ser  LOW. Por ejemplo, puede ser utilizado para encender un LED. Tipo de sensor: digital. Rango de valores: Alto (HIGH) y Bajo (LOW)');
    }
};
```

Código A.25: Código bloque botón pulsado en blocks/grove.js

Bloque botón no pulsado:

```
Blockly.Dart['boton_NO_pulsado'] = function(block)
{
    var dropdown_pin = this.getFieldValue('PIN');
    Blockly.Dart.setups_['setup_button_' + dropdown_pin] = 'pinMode(' + dropdown_pin + ', INPUT);';
    var code = '(((int) digitalRead(' + dropdown_pin + ')) == 0)';
    return [code, Blockly.Dart.ORDER_ATOMIC];
};
```

Código A.26: Código bloque botón no pulsado en generators/dart/grove.js

Aspecto bloque botón no pulsado:

```
Blockly.Blocks['boton_NO_pulsado'] =
{
    helpUrl: 'http://www.sucre.uji.es/docs/sensores/#boton',
    init: function()
    {
        this.setColour(210);
        this.appendDummyInput()
            .appendField(new Blockly.FieldImage("./assets/media/sensores/boton/noPulsado.png", 64, 64))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.digital), "PIN");
        this.setOutput(true, 'Boolean');
        this.setTooltip('BOTON NO PULSADO. Este sensor facilita la interacci n mediante pulsaciones. Cuando este es presionado obtiene un valor HIGH, y cuando se libera, ser  LOW. Por ejemplo, puede ser utilizado para encender un LED. Tipo de sensor: digital. Rango de valores: Alto (HIGH) y Bajo (LOW)');
    }
};
```

Código A.27: Código bloque botón no pulsado en blocks/grove.js

A.2.7. Sensor de rotación

Bloque ángulo grande:

```
Blockly.Dart['angulo_grande'] = function(block)
{
    var dropdown_pin = this.getFieldValue('PIN');
    var code = '((analogRead(' + dropdown_pin + ') / 4) > 512)';
    return [code, Blockly.Dart.ORDER_ATOMIC];
};
```

Código A.28: Código bloque ángulo grande en generators/dart/grove.js

Aspecto bloque ángulo grande:

```
Blockly.Blocks['angulo_grande'] =
{
    helpUrl: 'http://www.sucre.uji.es/docs/sensores/#angulo-rotativo',
    init: function()
    {
        this.setColour(210);
        this.appendDummyInput()
        .appendField(new Blockly.FieldImage("./assets/media/sensores/angulo/
    mucho.png", 64, 64))
        .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.analog), "
    PIN");
        this.setOutput(true, 'Boolean');
        this.setTooltip('ANGULO AGUDO Este sensor va tomando diferentes valores
     (0 a 1024) dependiendo de la posici n del  ngulo  de rotaci n. Puede ser
    utilizado para encender o apagar m s barras en la barra de LEDs. Tipo de
    sensor: anal gico. Rango de valores: 0-1024');
    }
};
```

Código A.29: Código bloque ángulo grande en blocks/grove.js

Bloque ángulo pequeño:

```
Blockly.Dart['angulo_peque o'] = function(block)
{
    var dropdown_pin = this.getFieldValue('PIN');
    var code = '((analogRead(' + dropdown_pin + ') / 4) <= 512)';
    return [code, Blockly.Dart.ORDER_ATOMIC];
};
```

Código A.30: Código bloque ángulo pequeño en generators/dart/grove.js

Aspecto bloque ángulo pequeño:

```
Blockly.Blocks['angulo_peque o'] =
{
    helpUrl: 'http://www.sucre.uji.es/docs/sensores/#angulo-rotativo',
    init: function()
    {
        this.setColour(210);
        this.appendDummyInput()
            .appendField(new Blockly.FieldImage("./assets/media/sensores/angulo
    /poco.png", 64, 64))
            .appendField("PIN")
```

```
10              .appendField(new Blockly.FieldDropdown(profile.default.analog), "
    PIN");
11          this.setOutput(true, 'Boolean');
12          this.setTooltip('ANGULO AGUDO Este sensor va tomando diferentes valores
     (0 a 1024) dependiendo de la posici n del  ngulo  de rotaci n. Puede ser
    utilizado para encender o apagar m s barras en la barra de LEDs. Tipo de
    sensor: anal gico. Rango de valores: 0-1024');
13      }
14  };
```

Código A.31: Código bloque ángulo pequeño en blocks/grove.js

A.2.8. Actuador led simple

Bloque led encendido:

```
1  Blockly.Dart['led_simple_encendido'] = function(block)
2  {
3      var dropdown_pin = this.getFieldValue('PIN');
4      Blockly.Dart.setups_['setup_green_led_' + dropdown_pin] = 'pinMode(' +
    dropdown_pin + ', OUTPUT);';
5      var code = 'digitalWrite(' + dropdown_pin + ',HIGH);\n'
6      return code;
7  };
```

Código A.32: Código bloque led encendido en generators/dart/grove.js

Aspecto led encendido:

```
1  Blockly.Blocks['led_simple_encendido'] =
2  {
3      helpUrl: 'http://www.sucre.uji.es/docs/actuadores/#led',
4      init: function() {
5          this.setColour(140);
6          this.appendDummyInput()
7          .appendField(new Blockly.FieldImage("./assets/media/actuadores/
    ledSimple/encendido.png", 64, 50, "*"))
8          .appendField("PIN")
9              .appendField(new Blockly.FieldDropdown(profile.default.digital), "
    PIN");
10         this.setPreviousStatement(true, null);
11         this.setNextStatement(true, null);
12         this.setTooltip('Este actuador activa el led. Tipo de sensor: digital.
    Valor: Alto (HIGH)');
13     }
14 };
```

Código A.33: Código bloque led encendido en blocks/grove.js

Bloque led apagado:

```
1  Blockly.Dart['led_simple_apagado'] = function(block)
2  {
3      var dropdown_pin = this.getFieldValue('PIN');
4      Blockly.Dart.setups_['setup_green_led_' + dropdown_pin] = 'pinMode(' +
    dropdown_pin + ', OUTPUT);';
5      var code = 'digitalWrite(' + dropdown_pin + ',LOW);\n'
6      return code;
```

```
};
```

Código A.34: Código bloque led apagado en generators/dart/grove.js

Aspecto bloque led apagado:

```
Blockly.Blocks['led_simple_apagado'] =
{
    helpUrl: 'http://www.sucre.uji.es/docs/actuadores/#led',
    init: function() {
        this.setColour(140);
        this.appendDummyInput()
        .appendField(new Blockly.FieldImage("./assets/media/actuadores/
    ledSimple/apagado.png", 64, 50, "*"))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.digital), "
    PIN");
        this.setPreviousStatement(true, null);
        this.setNextStatement(true, null);
        this.setTooltip('Este actuador apaga el led. Tipo de sensor: digital.
    Valor: Alto (HIGH)');
    }
};
```

Código A.35: Código bloque led apagado en blocks/grove.js

Actuador led variable

Bloque led verde:

```
Blockly.Dart['led_color_verde'] = function(block)
{
    var dropdown_pin = this.getFieldValue('PIN');
    var next_pin = 'D' + (parseInt(dropdown_pin.slice(1, dropdown_pin.length))
    + 1);
    Blockly.Dart.definitions_['define_led_variable'] = '#include "ChainableLED.
    h"';
    Blockly.Dart.definitions_['define_grove_led_variable'] = 'ChainableLED leds
    (' + dropdown_pin + ',' + next_pin + ',1);';
    Blockly.Dart.setups_['setup_led_variable_begin'] = 'leds.init();';
    return 'leds.setColorRGB(0,0, 255, 0);\n';
};
```

Código A.36: Código bloque led verde en generators/dart/grove.js

Aspecto led verde:

```
Blockly.Blocks['led_color_verde'] =
{
    helpUrl: 'http://www.sucre.uji.es/docs/actuadores/#led-variable',
    init: function()
    {
        this.setColour(140);
        this.appendDummyInput()
        .appendField(new Blockly.FieldImage("./assets/media/actuadores/ledColor
    /verde.png", 64, 50, "*"))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.digital), "
    PIN");
```

```
        this.setPreviousStatement(true, null);
        this.setNextStatement(true, null);
        this.setTooltip('Este actuador pone a verde el led. Tipo de sensor:
    digital.');
    }
};
```

Código A.37: Código bloque led verde en blocks/grove.js

Bloque led azul:

```
Blockly.Dart['led_color_azul'] = function(block)
{
    var dropdown_pin = this.getFieldValue('PIN');
    var next_pin = 'D' + (parseInt(dropdown_pin.slice(1, dropdown_pin.length))
    + 1);
    Blockly.Dart.definitions_['define_led_variable'] = '#include "ChainableLED.
    h"';
    Blockly.Dart.definitions_['define_grove_led_variable'] = 'ChainableLED leds
    (' + dropdown_pin + ',' + next_pin + ',1);';
    Blockly.Dart.setups_['setup_led_variable_begin'] = 'leds.init();';
    return 'leds.setColorRGB(0,51, 51, 255);\n';
};
```

Código A.38: Código bloque led azul en generators/dart/grove.js

Aspecto bloque led azul:

```
Blockly.Blocks['led_color_azul'] =
{
    helpUrl: 'http://www.sucre.uji.es/docs/actuadores/#led-variable',
    init: function()
    {
        this.setColour(140);
        this.appendDummyInput()
        .appendField(new Blockly.FieldImage("./assets/media/actuadores/ledColor
    /azul.png", 64, 50, "*"))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.digital), "
    PIN");
        this.setPreviousStatement(true, null);
        this.setNextStatement(true, null);
        this.setTooltip('Este actuador pone a azul el led. Tipo de sensor:
    digital.');
    }
};
```

Código A.39: Código bloque led azul en blocks/grove.js

Bloque led naranja:

```
Blockly.Dart['led_color_naranja'] = function(block)
{
    var dropdown_pin = this.getFieldValue('PIN');
    var next_pin = 'D' + (parseInt(dropdown_pin.slice(1, dropdown_pin.length))
    + 1);
    Blockly.Dart.definitions_['define_led_variable'] = '#include "ChainableLED.
    h"';
    Blockly.Dart.definitions_['define_grove_led_variable'] = 'ChainableLED leds
    (' + dropdown_pin + ',' + next_pin + ',1);';
```

```
    Blockly.Dart.setups_['setup_led_variable_begin'] = 'leds.init();';
    return 'leds.setColorRGB(0,255, 102, 0);\n';
};
```

Código A.40: Código bloque led naranja en generators/dart/grove.js

Aspecto led naranja:

```
Blockly.Blocks['led_color_naranja'] =
{
    helpUrl: 'http://www.sucre.uji.es/docs/actuadores/#led-variable',
    init: function()
    {
        this.setColour(140);
        this.appendDummyInput()
        .appendField(new Blockly.FieldImage("./assets/media/actuadores/ledColor/naranja.png", 64, 50, "*"))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.digital), "PIN");
        this.setPreviousStatement(true, null);
        this.setNextStatement(true, null);
        this.setTooltip('Este actuador pone a naranja el led. Tipo de sensor: digital.');
    }
};
```

Código A.41: Código bloque led naranja en blocks/grove.js

Bloque led amarillo:

```
Blockly.Dart['led_color_amarillo'] = function(block)
{
    var dropdown_pin = this.getFieldValue('PIN');
    var next_pin = 'D' + (parseInt(dropdown_pin.slice(1, dropdown_pin.length)) + 1);
    Blockly.Dart.definitions_['define_led_variable'] = '#include "ChainableLED.h"';
    Blockly.Dart.definitions_['define_grove_led_variable'] = 'ChainableLED leds(' + dropdown_pin + ',' + next_pin + ',1);';
    Blockly.Dart.setups_['setup_led_variable_begin'] = 'leds.init();';
    return 'leds.setColorRGB(0,255, 255, 0);\n';
};
```

Código A.42: Código bloque led amarillo en generators/dart/grove.js

Aspecto bloque led amarillo:

```
Blockly.Blocks['led_color_amarillo'] =
{
    helpUrl: 'http://www.sucre.uji.es/docs/actuadores/#led-variable',
    init: function()
    {
        this.setColour(140);
        this.appendDummyInput()
        .appendField(new Blockly.FieldImage("./assets/media/actuadores/ledColor/amarillo.png", 64, 50, "*"))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.digital), "PIN");
```

```
        this.setPreviousStatement(true, null);
        this.setNextStatement(true, null);
        this.setTooltip('Este actuador pone a amarillo el led. Tipo de sensor:
    digital.');
    }
};
```

Código A.43: Código bloque led amarillo en blocks/grove.js

Bloque led morado:

```
Blockly.Dart['led_color_morado'] = function(block)
{
    var dropdown_pin = this.getFieldValue('PIN');
    var next_pin = 'D' + (parseInt(dropdown_pin.slice(1, dropdown_pin.length))
    + 1);
    Blockly.Dart.definitions_['define_led_variable'] = '#include "ChainableLED.
    h"';
    Blockly.Dart.definitions_['define_grove_led_variable'] = 'ChainableLED leds
    (' + dropdown_pin + ',' + next_pin + ',1);';
    Blockly.Dart.setups_['setup_led_variable_begin'] = 'leds.init();';
    return 'leds.setColorRGB(0,204, 51, 204);\n';
};
```

Código A.44: Código bloque led morado en generators/dart/grove.js

Aspecto led morado:

```
Blockly.Blocks['led_color_morado'] =
{
    helpUrl: 'http://www.sucre.uji.es/docs/actuadores/#led-variable',
    init: function()
    {
        this.setColour(140);
        this.appendDummyInput()
        .appendField(new Blockly.FieldImage("./assets/media/actuadores/ledColor
    /morado.png", 64, 50, "*"))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.digital), "
    PIN");
        this.setPreviousStatement(true, null);
        this.setNextStatement(true, null);
        this.setTooltip('Este actuador pone a rojo el led. Tipo de sensor:
    digital.');
    }
};
```

Código A.45: Código bloque led morado en blocks/grove.js

Bloque led rojo:

```
Blockly.Dart['led_color_rojo'] = function(block)
{
    var dropdown_pin = this.getFieldValue('PIN');
    var next_pin = 'D' + (parseInt(dropdown_pin.slice(1, dropdown_pin.length))
    + 1);
    Blockly.Dart.definitions_['define_led_variable'] = '#include "ChainableLED.
    h"';
    Blockly.Dart.definitions_['define_grove_led_variable'] = 'ChainableLED leds
    (' + dropdown_pin + ',' + next_pin + ',1);';
```

```
    Blockly.Dart.setups_['setup_led_variable_begin'] = 'leds.init();';
    return 'leds.setColorRGB(0,204, 0, 0);\n';
};
```

Código A.46: Código bloque led rojo en generators/dart/grove.js

Aspecto bloque led rojo:

```
Blockly.Blocks['led_color_rojo'] =
{
    helpUrl: 'http://www.sucre.uji.es/docs/actuadores/#led-variable',
    init: function()
    {
        this.setColour(140);
        this.appendDummyInput()
        .appendField(new Blockly.FieldImage("./assets/media/actuadores/ledColor/rojo.png", 64, 50, "*"))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.digital), "PIN");
        this.setPreviousStatement(true, null);
        this.setNextStatement(true, null);
        this.setTooltip('Este actuador pone a rojo el led. Tipo de sensor: digital.');
    }
};
```

Código A.47: Código bloque led rojo en blocks/grove.js

A.2.9. Actuador barra led

Bloque barra led:

```
Blockly.Dart['barra_led_select'] = function(block)
{
    var dropdown_pin = this.getFieldValue('PIN');
    var next_pin = 'D' + (parseInt(dropdown_pin.slice(1, dropdown_pin.length)) + 1);

    var numLeds = this.getFieldValue('num1');
    numLeds = parseInt(numLeds);

    if(numLeds > 10)
    {
        numLeds = 10;
    }
    else if(numLeds < 0)
    {
        numLeds = 0;
    }

    Blockly.Dart.definitions_['define_led_bar'] = '#include "Grove_LED_Bar.h"';
    Blockly.Dart.definitions_['define_DHT'] = 'Grove_LED_Bar bar(' + next_pin + ',' + dropdown_pin + ',0);';
    Blockly.Dart.setups_['setup_dhtbegin_'] = 'bar.begin();';

    return 'bar.setLevel('+valor+');\n';
};
```

Código A.48: Código bloque barra led en generators/dart/grove.js

Aspecto barra led:

```
Blockly.Blocks['barra_led_select'] =
{
    init: function()
    {
        this.setColour(140);
        this.appendDummyInput()
            .appendField(new Blockly.FieldImage("./assets/media/actuadores/barraLed.png", 64, 50, "*"))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.digital), "PIN")
            .appendField("N  m. barras:")
            .appendField(new Blockly.FieldDropdown
            ([
                ["0", "0"],
                ["1", "1"],
                ["2", "2"],
                ["3", "3"],
                ["4", "4"],
                ["5", "5"],
                ["6", "6"],
                ["7", "7"],
                ["8", "8"],
                ["9", "9"],
                ["10", "10"]
            ]), "num1")
        this.setPreviousStatement(true, null);
        this.setNextStatement(true, null);
        this.setTooltip("Este actuador permide decidir qu  n mero de barras deja encendidas. Tipo de sensor: digital. Color rojo: 0.");
        this.setHelpUrl("http://www.sucre.uji.es/docs/actuadores/#barra-de-leds");
    }
};
```

Código A.49: Código bloque barra led en blocks/grove.js

A.2.10. Actuador pantalla dígitos

Bloque pantalla dígitos:

```
Blockly.Dart['pantalla_digitos_input'] = function(block)
{
    var dropdown_pin = this.getFieldValue('PIN');
    var clk_pin = 'D' + (parseInt(dropdown_pin.slice(1, dropdown_pin.length)) + 1);

    var numero = Blockly.Dart.valueToCode(block, 'pantalla', Blockly.Dart.ORDER_ATOMIC);

    if(numero > 9999)
    {
        numero = 9999;
    }
    else if(numero < 0)
    {
        numero = 0;
    }
```

```
16
17      Blockly.Dart.definitions_['define_TM1637Display'] = '#include "
        TM1637Display.h"';
18      Blockly.Dart.definitions_['define_4digit'] = 'TM1637Display display(' +
        dropdown_pin + ',' + clk_pin + ');\n';
19      Blockly.Dart.setups_['setup_4_digit'] = 'display.setBrightness(0x0a);\n';
20
21      var code = '\tdisplay.showNumberDec(' + numero + ');\n';
22      return code;
23  };
```

Código A.50: Código bloque pantalla dígitos en generators/dart/grove.js

Aspecto pantalla dígitos:

```
1   Blockly.Blocks['pantalla_digitos_input'] =
2   {
3       helpUrl: 'http://www.sucre.uji.es/docs/actuadores/#pantalla-segmentos',
4       init: function()
5       {
6           this.setColour(140);
7           this.appendDummyInput()
8               .appendField(new Blockly.FieldImage("./assets/media/actuadores/
        pantallaDigitos.png", 64, 50, "*"))
9               .appendField("PIN")
10              .appendField(new Blockly.FieldDropdown(profile.default.digital), "
        PIN")
11          this.appendValueInput("pantalla", 'Number')
12              .appendField("Mostrar")
13              .setCheck('Number');
14          this.setInputsInline(true);
15          this.setPreviousStatement(true, null);
16          this.setNextStatement(true, null);
17          this.setTooltip('Este actuador muestra 0000. Tipo de sensor: digital.')
        ;
18        }
19  };
```

Código A.51: Código bloque pantalla de dígitos en blocks/grove.js

A.2.11. Actuador zumbador

Bloque zumbador encendido:

```
1   Blockly.Dart['zumbador_encendido'] = function(block)
2   {
3       var dropdown_pin = this.getFieldValue('PIN');
4       Blockly.Dart.setups_['setup_piezo_buzzer_' + dropdown_pin] = '\tpinMode(' +
         dropdown_pin + ', OUTPUT);';
5       var code = 'digitalWrite(' + dropdown_pin + ', HIGH);\n'
6       return code;
7   };
```

Código A.52: Código bloque zumbador encendido en generators/dart/grove.js

Aspecto zumbador encendido:

```
1   Blockly.Blocks['zumbador_encendido'] =
2   {
```

```
    helpUrl: 'http://www.sucre.uji.es/docs/actuadores/#zumbador',
    init: function()
    {
        this.setColour(140);
        this.appendDummyInput()
        .appendField(new Blockly.FieldImage("./assets/media/actuadores/zumbador
    /sonido.png", 64, 50, "*"))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.digital), "
    PIN");
        this.setPreviousStatement(true, null);
        this.setNextStatement(true, null);
        this.setTooltip('Descripci n: este actuador es capaz de realizar
    pitidos cuando esta a valor HIGH (encendido). Por ejemplo, puede ser
    utilizado para indicar la proximidad con un objeto, similar a un sensor de
    aparcamiento de los coches.');
    }
};
```

Código A.53: Código bloque zumbador encendido en blocks/grove.js

Bloque zumbador apagado:

```
Blockly.Dart['zumbador_apagado'] = function(block)
{
    var dropdown_pin = this.getFieldValue('PIN');
    Blockly.Dart.setups_['setup_piezo_buzzer_' + dropdown_pin] = '\tpinMode(' +
     dropdown_pin + ', OUTPUT);';
    var code = 'digitalWrite(' + dropdown_pin + ', LOW);\n';
    return code;
};
```

Código A.54: Código bloque zumbador apagado en generators/dart/grove.js

Aspecto bloque zumbador apagado:

```
Blockly.Blocks['zumbador_apagado'] =
{
    helpUrl: 'http://www.sucre.uji.es/docs/actuadores/#zumbador',
    init: function()
    {
        this.setColour(140);
        this.appendDummyInput()
        .appendField(new Blockly.FieldImage("./assets/media/actuadores/zumbador
    /silencio.png", 64, 50, "*"))
            .appendField("PIN")
            .appendField(new Blockly.FieldDropdown(profile.default.digital), "
    PIN");
        this.setPreviousStatement(true, null);
        this.setNextStatement(true, null);
        this.setTooltip('Descripci n: este actuador es capaz de realizar
    pitidos cuando esta a valor HIGH (encendido). Por ejemplo, puede ser
    utilizado para indicar la proximidad con un objeto, similar a un sensor de
    aparcamiento de los coches.');
    }
};
```

Código A.55: Código bloque zumbador apagado en blocks/grove.js

A.3. O3: Mejoras en la aplicación

A.3.1. Colección musical

Notas versión Básica

```
    Blockly.Dart['do'] = function(block)
    {
        var dropdown_pin = this.getFieldValue('PIN');
        var frecuencia = 261;
        Blockly.Dart.setups_['setup_piezo_buzzer_' + dropdown_pin] = '\tpinMode
    (' + dropdown_pin + ', OUTPUT);';
        var tiempo = Blockly.Dart.valueToCode(this, 'tiempo', Blockly.Dart.
    ORDER_UNARY_POSTFIX);
        var code = 'tone(' + dropdown_pin +', '+frecuencia+', '+tiempo+');\
    ndelay('+tiempo+');\nnoTone('+dropdown_pin+');\n';
        return code;
    };

```

Código A.56: Bloque nota do

```
    Blockly.Blocks['do'] =
    {
        helpUrl: 'http://www.sucre.uji.es/docs/actuadores/#zumbador',
        init: function()
        {
            this.setColour(140);
            this.appendValueInput("tiempo")
                .setCheck("Number")
                .appendField(new Blockly.FieldImage("./assets/media/actuadores/
    musica/notas/do.png", 90, 70, "*"))
                .appendField("PIN")
                .appendField(new Blockly.FieldDropdown(profile.default.digital)
    , "PIN");
            this.setPreviousStatement(true, null);
            this.setNextStatement(true, null);
            this.setTooltip('Descripci n: nota Do (frecuencia 261).');
        }
    };

```

Código A.57: Aspecto bloque do

Figuras musicales:

```
    Blockly.Dart['negra'] = function(block)
    {
        var code = '1000';
        return [code, Blockly.Dart.ORDER_NONE]
    };

```

Código A.58: Bloque nota negra

```
    Blockly.Blocks['negra'] =
    {
        init: function()
        {
            this.setColour(140);
```

```
            this.appendDummyInput()
            .appendField(new Blockly.FieldImage("./assets/media/actuadores/musica/tiempo/notaNegra.png", 64, 64))
            this.setOutput(true, "Number");
            this.setTooltip('Descripci n: Negra (1 tiempo).');
        }
    };

```

Código A.59: Aspecto bloque nota negra

A.3.2. Descargar gráficas

```
    descargarTablaCSV()
    {
        // Obtener el elemento HTML con el id de la tabla de la que queremos generar el csv
        const tablaMediciones = document.querySelector('#tablaMediciones');
        //Contenido primera fila csv.
        let stringCsv = "Valor;Fecha;Hora\n";
        var contador = 0;
        // Obtener todas las filas de la tabla
        const rows = tablaMediciones.querySelectorAll('td');
        //Recorremos los elementos de la tabla por filas
        for (var i = 0; i < rows.length; i++)
        {
            //Rellenamos el doc, separando cada celda por ; e incrementamos el contador
            stringCsv += rows[i].textContent+";";
            contador ++;
            //La tabla solo tiene 3 columnas, si el contador llega a 3. Fin de fila. (Salto de l nea en csv).
            if(contador == 3)
            {
                stringCsv += "\n";
                contador = 0;
            }
        }
        var filename = this.deviceName+"_"+this.variableName+".csv";
        this.descargarDocCSV(stringCsv, filename);
    }

    descargarDocCSV(csv, filename)
    {
        //Crear archivo csv
        var csvFile = new Blob([csv], {type: "text/csv"});
        // Descargar link
        var downloadLink = document.createElement("a");
        // Nombre del fichero
        downloadLink.download = filename;
        // Crear un link para el fichero
        downloadLink.href = window.URL.createObjectURL(csvFile);
        // Ocultar link
        downloadLink.style.display = "none";
        // A adir link al DOM
        document.body.appendChild(downloadLink);
        // Click download link
        downloadLink.click();
    }
```

```

```

Código A.60: Descargar CSV

```
<button mat-flat-button color="primary" style="align-self: center;" (click)
="descargarGraficaPdf()">
    Descargar PDF
</button>

```

Código A.61: Botón descargar gráfica

```
descargarGraficaPdf()
{
    const elementoGrafica = document.getElementById('datosGraficaPDF');
    const docGrafica = new jsPDF('p', 'pt', 'a4');
    const opcionesGrafica =
    {
      background: 'white',
      scale: 3
    };

    html2canvas(elementoGrafica, opcionesGrafica).then((canvas) =>
    {
      const imgagenCanvas = canvas.toDataURL('image/PNG');
      const imgProps = (docGrafica as any).getImageProperties(imgagenCanvas
);
      const pdfWidth = docGrafica.internal.pageSize.getWidth() - 30;
      const pdfHeight = (imgProps.height * pdfWidth) / imgProps.width;
      docGrafica.addImage(imgagenCanvas, 'PNG', 15, 15, pdfWidth, pdfHeight
, undefined, 'FAST');
      return docGrafica;
    }).then(() =>
    {
      docGrafica.save(this.deviceName+"_graficas.pdf");
    });
}

```

Código A.62: Método descargarGraficaPdf()

Bibliografía

[1] Amazon. *Amazon Web Services*. [Consulta: 07 de Julio de 2023].

[2] GEOTEC. *Programa Sucre*. [Consulta: 07 de Julio de 2023].

[3] GEOTEC. *Sucre4Kids*. [Consulta: 07 de Julio de 2023].

[4] GEOTEC. *Sucre4STEM*. [Consulta: 07 de Julio de 2023].

[5] Google. *Angular*. [Consulta: 07 de Julio de 2023].

[6] Google. *Blockly*. [Consulta: 07 de Julio de 2023].

[7] Google. *Firebase*. [Consulta: 07 de Julio de 2023].

[8] Carlos Granell-Canut y Sergio Trilles-Oliver. "El proyecto SUCRE4Kids: una iniciativa de hardware y software libre para la introducción a la programación". En: 240. 13 de nov. de 2018. ISBN: 2444-6629. published.

[9] Sergio Trilles-Oliver y Carlos Granell-Canut. "Advancing preuniversity students' computational thinking skills through an educational project based on tangible elements and virtual block-based programming". En: vol. 28. 6. 5 de mar. de 2020, págs. 1490-1502. ISBN: 1099-0542. published.

[10] Sergio Trilles-Oliver y Carlos Granell-Canut. "SUCRE4Kids: El fomento del pensamiento computacional a través de la interacción social y tangible". En: *XXIV Jornadas sobre la Enseñanza Universitaria de la Informática (JENUI 2018)*. jenui, 1 de ene. de 2018. published.

[11] Sergio Trilles-Oliver, Carlos Granell-Canut y Estefanía Aguilar-Moreno. "SUCRE4Kids: tres años de experiencia en la incentivación del pensamiento computacional en edades preuniversitarias". En: ed. por Andrés Vaz Fidalgo y O. (Eds.) TICAI 2018: TICs para el Aprendizaje de la Ingeniería Óscar Martínez Bonastre. Vigo: Universidad de Vigo, 8 de jul. de 2019, págs. 49-56. published.

[12] Sergio Trilles-Oliver, David Tortosa y Carlos Granell-Canut. "La evolución del proyecto Sucre4Kids mediante el paradigma del Internet de las Cosas". En: *Actas de las Jornadas sobre Enseñanza Universitaria de la Informática (JENUI) Valencia, Spain, Jul 2020*. Vol. 5. 9 de jul. de 2020, págs. 53-60. published.

[13] Sergio Trilles-Oliver et al. "Sucre4Stem: Collaborative Projects Based on IoT Devices for Students in Secondary and Pre-University Education". En: vol. 17. 2. 1 de mayo de 2022, págs. 150-159. published.

[14] Sergio Trilles-Oliver et al. "Sucre4Stem: Internet of things in classrooms". En: *2022 Congreso de Tecnología, Aprendizaje y Enseñanza de la Electrónica (XV Technologies Applied to Electronics Teaching Conference)*. IEEE, 9 de ago. de 2022, págs. 1-4. ISBN: 978-1-6654-2161-4. published.