Grado en Ingeniería Informática

Trabajo de Final de Grado

Desarrollo de una aplicación web para programar dispositivos *IoT* mediante programación visual

Autor:
David Tortosa escudero

Supervisor:
Joaquín Torres Sospedra
Tutor académico:
Dolores María llidó escrivá

Fecha de lectura: 14 de julio de 2020
Curso académico 2019/2020

Resumen

El proyecto cuya propuesta se presenta en este documento se va a desarrollar en Ubik geospatial solutions. El proyecto consiste en el desarrollo de una aplicación web implementada en *Angular* capaz de programar dispositivos *IoT* mediante programación visual por bloques que permite enviar el código ejecutable al dispositivo *IoT*.

El proyecto consta de cuatro partes. En primer lugar, la creación del portal llamado *SucreCode* que es una mejora de un portal previo ya existente. El nuevo portal ha de gestionar los proyectos de programación de los dispositivos *IoT* de los usuarios. En segundo lugar, mejorar la integración del sistema de programación visual mediante bloques *Blockly* en el portal web adaptándolo para que soporte *C* . En tercer lugar, crear un sistema que almacene los proyectos que generen los usuarios y que permita la gestión de los mismos. En cuarto lugar, crear la comunicación entre el portal web y los dispositivos *IoT* mediante *Particle* para probar el funcionamiento de nuestros programas en los dispositivos.

Palabras clave

Angular, IoT, Firebase, Blockly, programación visual, Particle

Keywords

Angular, IoT, Firebase, Blockly, visual programming, Particle

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

La empresa en la que se han desarrollado las prácticas y el proyecto ha sido Ubik geospatial solutions. Esta empresa ofrece servicios *TIC* (*Tecnologías de la información y la comunicación*) a usuarios y empresas. Se proporcionan aplicaciones web y móviles personalizadas para ayudar a resolver problemas en campos como la integración de *Smart Cities, Environment o Social Media.*

Ubik geospatial solutions es una empresa con 15 años de experiencia en el sector. Ubik se separó del grupo de investigación Geotec de la Universitat Jaume I (UJI) y comenzó a operar a finales de 2016 debido a la necesidad de proporcionar desarrollo de software de grado comercial, gestión de proyectos, entrega y soporte a organizaciones de todo el mundo. El equipo de Ubik ha estado trabajando con tecnologías geoespaciales desde 1998 en la UJI, y ha participado en numerosos proyectos de I + D en el campo "geo". Estas iniciativas incluyen proyectos financiados por la Unión Europea que involucran la creación e integración de servicios web, consultoría técnica a empresas y agencias de la ONU, e investigación financiada a nivel nacional (Ministerio de Educación o gobierno regional de Valencia).

Este proyecto surgió de la necesidad de facilitar la gestión de los dispositivos *IoT*[1]. Esto no solo se refiere a como gestionarlos, sino también a poder identificarlos fácilmente y a poder subirles código ejecutable desde la interfaz amigable *Blockly* [5] a estos dispositivos a través de la librería de *Particle*. Además, también se mejora la eficiencia en la gestión de los propios dispositivos ya que se puede subir código a cualquier dispositivo dado de alta en la base de datos, que este conectado a internet.

Blockly permite generar código en distintos lenguajes pero no en *C*, por ello como parte del proyecto, hay que adaptar *Blockly* para que genere código ejecutable en *C* adaptado a *Arduino* y añadirlo a la web para gestionar el código que usarán los usuarios. Además, permite generar código mediante bloques, lo que permite a los usuarios sin conocimientos de programación crear pequeños programas para poder manipular sus dispositivos *IoT* de una forma sencilla.

[1]Internet de las cosas - *Internet of things*

1.2. Objetivos del proyecto

El objetivo principal del proyecto se centra en la creación de una nueva aplicación web llamada *SucreCode*[2] basándome en una versión anterior llamada *Sucre4Kids*[3]. Como en *Sucre4Kids* se habían detectado ciertas deficiencias surge la necesidad de crear una nueva aplicación web. *SucreCode* tiene que ser capaz de programar dispositivos *IoT* mediante programación visual por bloques, permitir el envío de código ejecutable al dispositivo *IoT* a través de las funcionalidades proporcionadas por *Particle*. Desde la propia aplicación los usuarios deben ser capaces de gestionar sus proyectos; para ello, la misma aplicación usará una base de datos *Firebase* donde se leerán y guardarán los proyectos, los usuarios y los dispositivos *IoT*.

1.3. Objetivos detallados

En esta sección se muestran más en detalle los objetivos del proyecto para cada una de las partes:

- Portal web
  - Sistema de autorización/autentificación de usuarios.
  - Sistema de gestión de proyectos de los usuarios.
  - Mejorar la interfaz de *Blockly* para que el usuario programe sus dispositivos *IoT* para:
    - Enviar el código a los dispositivos.
    - Permita cargar y guardar los proyectos.
    - Mejorar la programación visual para permitir la programación en *C*.
  - Crear un transpilador para generar código en *C - Arduino*.
  - Crear los bloques necesarios para poder realizar la programación visual.

Por lo tanto se requiere crear una base de datos para:

- Crear las colecciones necesarias para almacenar y gestionar los proyectos de programación de los usuarios.
- Mantener la información de la autenticación de usuarios Oauth.

La empresa ha especificado las tecnologías a utilizar en el desarrollo del proyecto. Además, los dispositivos *IoT* son placas *Particle* que requieren que su código se implemente en *C*, por lo que se requiere que *Blockly* genere el código en *C*.

[2] http://elcano.init.uji.es/SucreCode2
[3] http://elcano.init.uji.es/sucre4kids/

1.4. Gestión de Riesgos

La gestión de riesgos es una de las fases críticas del desarrollo de un proyecto, es necesaria una correcta identificación, planificación y gestión para poder garantizar que el proyecto tenga éxito pese a las posibles adversidades que vaya surgiendo durante su desarrollo.

Los riesgos son gestionados actualmente por los programadores, ya que ellos son los que más conocen el proyecto y tienen una visión más amplia de lo que puede afectar al proyecto a corto, medio y largo plazo.

1.4.1. Identificación de riesgos

Se van a exponer los distintos riesgos que se pueden dar en la realización del proyecto. Para cada riesgo que se identifique, se añadirá una descripción para entender el contexto, el tipo de riesgo y la gravedad del mismo según afecte al proyecto. En la tabla 1.1 se muestran los riesgos identificados.

Id	Descripción del riesgo	Tipo del riesgo
R01	Modificación de los requisitos	Riesgo del producto
R02	Desconocimiento de las tecnologías	Riesgo del producto
R03	Código difícil de adaptar a *C*	Riesgo del producto
R04	Cambios legislativos	Riesgo del proyecto
R05	Poca experiencia en planificación	Riesgo del proyecto

Tabla 1.1: Identificación de riesgos

1.4.2. Desglose de riesgos

A continuación, se van a exponer los riesgos descritos en el apartado anterior, señalando la importancia que tiene cada riesgo, una breve descripción para contextualizarlos y finalmente identificar su origen. Tal como se muestra en la tabla 1.2

Id	Riesgo
R01	**Importancia:** Alta **Descripción:** Son las funcionalidades que se han de cumplir. Las pautas que se han de seguir para todo el desarrollo de la aplicación. **Origen:** Añadir, quitar o modificar funcionalidades.
R02	**Importancia:** Media-Alta **Descripción:** Habilidades previas adquiridas para el desarrollo del producto con las tecnologías propuestas. **Origen:** Desconocimiento tecnológico.
R03	**Importancia:** Alta **Descripción:** Este punto se centra en la capacidad de generar códigos correctos en *C* mediante la interacción del usuario con *Blockly* **Origen:** Desconocimiento tecnológico
R04	**Importancia:** Alta **Descripción:** Los cambios legislativos impuestos por el estado que modifiquen, retrasen, paralicen o detengan el desarrollo del proyecto. **Origen:** Creación de leyes o modificaciones sobre las leyes previas desfavorables.
R05	**Importancia:** Media **Descripción:** Desconocimiento de buenas prácticas para una planificación realista. **Origen:** Mala comprensión o desconocimiento de las técnicas de planificación.

Tabla 1.2: Desglose de riesgos

1.5. Estructura de la memoria

Esta memoria está estructurada en 6 capítulos. Este capítulo es una introducción donde hemos visto el contexto, los objetivos y la gestión de riesgos del proyecto. En el capítulo 2 veremos una descripción del proyecto donde se expone la situación inicial del proyecto, las tecnologías que se usaran y el hardware usado para su desarrollo. Luego en el capítulo 3 hablaremos acerca de la planificación del

proyecto, este muestra la información relacionada con la metodología usada, la estimación de recursos y cómo ha sido el seguimiento del proyecto. A continuación, en el capítulo 4 se detalla el análisis y el diseño de la aplicación, diferenciando la parte dirigida a la aplicación web como la dirigida al diseño de *Blockly* y el formato de los datos. Después, el capítulo 5 muestra la implementación de dicha aplicación, tanto de *SucreCode* así como el uso que se le da a *Firebase* y *Particle* y su integración en el proyecto además, de las pruebas de validación. En el capítulo 6 se encuentran las conclusiones obtenidas tras el proceso de desarrollo.

Además, al final se ha detallado algunos apartados como apéndices. En el apéndice 1, se muestran fragmentos de código de la aplicación para complementar la explicación de su implementación en el capítulo 5. A continuación en el apéndice 2, se detalla con más detalle los pasos seguidos para la configuración de *Firebase*. Finalmente en el apéndice 3 se muestran los pasos seguidos para la configuración de los dispositivos *IoT* en *Particle*.

Capítulo 2

Descripción del proyecto

2.1. Situación inicial

Ubik geospatial solutions contaba con una página web con la que trabajaban los usuarios sin ningún tipo de control sobre el acceso a la aplicación, sobre los proyectos ni sobre los dispositivos *IoT*. Esto conllevaba una serie de inconvenientes debido a que esta página web estaba programada únicamente con *JavaScript*, *HTML* y una versión antigua de *Blockly*. La forma en que estaba planteada y programada, esta web no permitía tener una gestión centralizada de los usuarios y los dispositivos. Otro de los problemas principales, es que la aplicación web original no contaba con una base de datos, lo que planteaba una serie carencias que se explican a continuación:

- No hay autentificación ni control de usuarios de ningún tipo por lo que cualquiera puede acceder al portal y usar las funcionalidades del mismo.
- Al no existir un sistema centralizado de gestión de proyectos solo se puede trabajar con el proyecto actual, al cambiar de ordenador el programa que ha generado el usuario se pierde a no ser que lo guarde en una memoria portátil para cargarlo posteriormente en el otro equipo que vaya a usar.
- El código generado por un usuario se envía al dispositivo mediante una conexión física desde el ordenador donde esta conectado.

Dadas las limitaciones de la aplicación web expuestas anteriormente, se plantea crear un nuevo portal web desarrollado en *Angular* utilizando las librerías de *Particle*, *Blockly* y *Firebase* para ampliar su funcionalidad. Para este nuevo portal web, se van a realizar una serie de mejoras:

- Crear un sistema centralizado para almacenar los datos del usuarios, de los dispositivos *IoT* y de los proyectos de los usuarios.
- Crear un sistema de gestión de proyectos para que los usuarios tengan acceso a sus proyectos desde cualquier equipo, y sean capaces modificarlos, borrarlos y de crear nuevos.

- Filtrar los dispositivos *IoT* por usuario para que solo tenga acceso a los dispositivos asignados.
- Modificar la librería de *Blockly* para que sea capaz de generar código en *C*.
- Sistema de envío para poder mandar el código que genere el usuario en *SucreCode* a sus dispositivos *IoT* seleccionado. Esto es posible gracias a que *Particle* ofrece un servicio *REST*[15] accesible desde su *API*.

2.2. Tecnologías

El proyecto se ha desarrollado mediante el uso de *Angular* como entorno de trabajo, usando una base de datos *Firebase*, las librerías de *Blockly* y *Particle* y finalmente los lenguajes de programación *TypeScript* y *JavaScript*.

2.2.1. Angular

Angular [1] es un entorno de trabajo para aplicaciones web, desarrollado en *TypeScript*. Con este entorno de trabajo se crean aplicaciones ejecutadas en el cliente con un patrón *Modelo-Vista-Controlador*[14].

Este entorno de trabajo ofrece una serie de ventajas tales como que es estable y escalable además de ser compatible con un gran número de librerías. El uso de componentes que son pequeñas partes lógicas de la aplicación, que representan un trozo de la pantalla. Un componente de *Angular* es un bloque, que contiene un modelo, estilos y una parte lógica. Esto permite la reutilización de código de forma sencilla.

2.2.2. Firebase

Firebase [8] es una base datos no relacional alojada en la nube, los datos se almacenan en formato *JSON*[1] y se sincronizan en tiempo real. Las funciones de búsqueda de la base de datos se compilan en sus propios *SDK*[2] por lo que todos los usuarios que se conecten comparte una instancia y todos acceden a las mismas funciones y datos.

Algunas ventajas de usar este tipo de base de datos es que siga funcionando sin conexión y se sincronice automáticamente al volver a conectarse. Las funciones se programan en *TypeScript* [18] o *JavaScript* [13] independientemente del dispositivo donde se ejecuten y finalmente la información que contiene la base de datos se binariza para que ocupe menos espacio. Como desventaja, es que no tenemos una base de datos relacional con la cual estoy más familiarizado a trabajar, sino que los datos se almacenan en colecciones y ficheros lo cual requiere trabajar con un lenguaje de consulta distinto para acceder a los datos por *URLs*[3].

[1] Notación de objeto JavaScrip - *JavaScript Object Notationt*
[2] Kit de Desarrollo de Software - *Software Development Kit*
[3] Localizador Uniforme de Recursos - *Uniform Resource Locator*

2.2.3. Blockly

Blockly [5] es una librería de Google de programación visual que permite arrastrar los distintos componentes, estos pueden ser de control, lógica, operaciones matemáticas, texto y listados. Usando los componentes mencionados anteriormente, podemos crear programas los cuales se generaran en distintos lenguajes como *JavaScript, Dart, Python o XML* que son los que incluye por defecto.

2.2.4. Particle

Particle [16] es una plataforma de dispositivos *IoT* escalable, confiable y segura que permite construir, conectar y gestionar las soluciones de *IoT*. Particle proporciona un entorno integrado para desarrollar y gestionar productos *IoT*.

La comunicación de *SucreCode* con *Particle* se realiza mediante la librería que ofrece Particle. A través de las funcionalidades de esta librería se realiza las consultas a Particle para obtener los dispositivos y toda la información de los mismos.

2.3. Hardware usado

Para el proyecto se necesita de un *hardware* para su desarrollo, también es necesario cierto equipamiento para poder hospedar el portal web y poder trabajar con el. El equipamiento usado es el siguiente:

- Ordenador servidor web, este equipo es el encargado de tener un servidor web activo para poder acceder al portal web.
- Ordenador de desarrollo, usado para el desarrollo de la aplicación.
- Ordenador base de datos, equipo con una base de datos *Firebase*, el cual es el encargado de guardar los datos que usa la aplicación web.
- Dispositivos *IoT* con los que se trabajara en el proyecto son placas del tipo *Boron* y *Xenon*.

Se ha utilizado esta combinación de dispositivos *IoT* porque con el funcionamiento de estos dos tipos de placas podemos mantener las *Boron* siempre conectadas a través de una tarjeta *SIM* [4] y que las *Xenon* obtengan conexión a la red conectándose por *bluetooth*[5] a la *Boron*.

2.4. Software usado

Además de las tecnologías mencionadas anteriormente, también se ha usado *software* específico durante el desarrollo del proyecto:

[4] Módulo de Identificación de Abonado - *Subscriber Identity Module*
[5] Red inalámbrica para intercambiar datos de corto alcance

- *Visual Studio Code*, es el entorno de desarrollo de *Microsoft* usado para programar el proyecto. Este ofrece soporte para *TypeScript* además de una gran cantidad de extensiones que facilitan el desarrollo.
- *Trello*, esta aplicación web se ha elegido para facilitar la distribución de tareas y mejorar la organización del proyecto.
- *Github*, nos permite almacenar y compartir el código del proyecto en un entorno colaborativo. También nos da la posibilidad de poder trabajar en equipo y desde cualquier dispositivo.
- *Npm*, permite la gestión y administración de módulos del proyecto de una forma más sencilla. Además, podemos compartir las librerías que usa el proyecto gracias al registro que lleva y la posibilidad de instalar todas las dependencias con un solo comando.
- *Overleaf*, aplicación web para la generación de documentos de texto usando como procesador de texto LaTeX.

Capítulo 3

Planificación del proyecto

3.1. Metodología

La metodología que se ha usado en el desarrollo del proyecto es una metodología tradicional. Con esta metodología se ha de definir una lista con todas las tareas iniciales necesarias, el orden en que se ejecutarán, interdependencias de las tareas y la duración estimada de cada una para poder estructurar, desarrollar y finalizar el proyecto con éxito en los plazos previstos.

Usar este tipo de metodología conlleva una serie de inconvenientes asociados al propio método de trabajo tales como:

- **Dificultad al definir requisitos al inicio**: Es difícil establecer unos objetivos factibles y acertados desde el principio sin saber como se desarrollará el proyecto y que problemas pueden aparecer.
- **Alteración del orden de actividades establecido**: Si el proyecto sufre modificaciones o problemas en alguna de las fases o simplemente se desarrolla más lento de lo debido, es habitual comenzar con la siguiente tarea establecida y adelantarla en la medida de lo posible para intentar minimizar el retraso generado en el calendario del proyecto.

3.2. Planificación

Al principio del proyecto, se realizó una planificación inicial utilizando una lista de tareas, esta se puede observar en la Tabla 3.1. En el diagrama se puede apreciar el tiempo que se ha de invertir para la realización de la tarea, las dependencias que tiene la tarea además de quién ha de realizarla.

Nº	Proyecto	TIempo (h)	Dependencias	Rol
1	**SucreCode**	**300h**		
1.1	**Documentar y planificar el proyecto**	**28 h**		
1.1.1	Revisar contexto y buscar información	20		Analista
1.1.2	Identificar alcance y objetivos	8		Analista
1.2	**Planificar el proyecto**	**31 h**		Analista
1.2.1	Definir tareas y definir fechas	8	1.1	Analista y Programador
1.2.2	Crear diagrama de Gantt	8	1.2.1	Analista
1.2.3	Documentar la propuesta del proyecto	15	1.2.1	Analista
1.2.4	Entregar propuesta técnica	0	1.2.3	
2	**Desarrollo técnico del proyecto**			
2.1	**Definir requisitos del proyecto**	**30h**		
2.1.1	Crear diagrama de casos de uso	15	1.2	Analista
2.1.2	Estado del arte del contexto del proyecto	5	1.2	Analista
2.1.3	Definir requisitos tecnologicos y de plataforma	10	1.2	Analista y Programador
2.2	**Análisis**	**35 h**		
2.2.1	Crear diagrama de clases	15	2.1	Analista
2.2.2	Documentar clases	5	2.2.1	Analista
2.2.3	Diagrama de actividades	15	2.1	Analista
2.3	**Diseño**	**30 h**		
2.3.1	Identificar y clasificar usuarios	10	2.2	Analista
2.3.2	Diseñar interfaces gráficas	20	2.2	Analista y Programador
2.4	**Desarrollo del producto**	**140 h**		
2.4.1	Programación	120	2.3	Programador
2.4.2	Pruebas	20	2.4.1	Programador
2.5	**Puesta en marcha**	**6 h**		
2.5.1	Implantación	6	2.4	Programador
2.5.2	Entrega final	0	2.5.1	

Tabla 3.1: Organización del proyecto

3.3. Estimación de recursos y costes del proyecto

En este apartado se va a realizar un calculo de los costes del software y hardware necesario para la realización de las distintas actividades y fases del proyecto. Para poder obtener el coste total del proyecto se ha de tener en cuenta el tiempo total empleado en el desarrollo del producto.

El hardware se amortiza en 4 años y el total se calcula sobre los 4 meses de la estancia, en la tabla 3.2 se pueden ver los productos usados y el coste de cada uno.

Coste hardware			
Producto	**Coste**	**Coste amortizado/mes**	**Coste total amortizado**
Ordenador MacBook Pro	2.000€	41,6€	166,4€
Monitor Samsung U28E590D	240€	5€	20€
Ratón	20€	0.4€	1,6
Placa Particle Boron	45€	0,9€	3,6€
Placa Particle Xenon	10€ x 6	1,25€	5€
Total			**196,2€**

Tabla 3.2: Resumen costes hardware

La mayor parte de las tecnologías que se usan son gratuitas, por lo que no supone un gran problema económico. Los programas que si hay que pagar una licencia son *Visual Studio Code* para el desarrollo del proyecto, *GitHub* para mantener versiones, copias de seguridad y poder trabajar de manera colaborativa y *Trello* para coordinar el proyecto, en la tabla 3.3 lo podemos ver mas en detalle.

Coste software		
Producto	**Suscripción/Mes**	**Coste total**
Visual Studio Code	45€	180€
GitHub	4€	16€
Trello	12,5€	50€
Total		**246€**

Tabla 3.3: Resumen costes software

Para obtener el coste de desarrollo tomaremos como referencia la figura 3.1, el trabajo se divide en dos roles, programador[11] y analista[10] por lo que se calcularán los costes en función del rol ,la cantidad de horas por rol, horas dedicadas al proyecto y el coste económico total en la tabla 3.4.

Coste desarrollo				
Rol	**Sueldo bruto**	**Coste por hora**	**Horas dedicadas**	**Coste Total**
Analista	2.443€	15€/h	135h	2.025€
Programador	1.914€	12€/h	165h	1.980€
Total				**4.005€**

Tabla 3.4: Resumen costes desarrollo

El coste total lo obtenemos sumando los totales de las tablas 3.2 , 3.3 y 3.4, el resumen de estos totales se muestra en la tabla 3.5

Coste total	
Recursos	**Total**
Total hardware	196,2€
Total software	246€
Total desarrollo	4.733€
Total	**4.447,2€**

Tabla 3.5: Resumen costes totales

En la Tabla 3.6 podemos ver las desviaciones que ha sufrido el proyecto. Estas desviaciones se han producido hacia la segunda mitad de las prácticas, esto es causado por la suspensión de las prácticas presenciales, lo que ha obligado a acelerar el desarrollo del análisis y el diseño e incrementar el tiempo dedicado al desarrollo del producto.

Nº	Proyecto	TIempo (h)	Dependencias	Rol
1	**SucreCode**	**300h**		
1.1	**Documentar y planificar el proyecto**	**28 h**		
1.1.1	Revisar contexto y buscar información	20		Analista
1.1.2	Identificar alcance y objetivos	8		Analista
1.2	**Planificar el proyecto**	**31 h**		Analista
1.2.1	Definir tareas y definir fechas	8	1.1	Analista y Programador
1.2.2	Crear diagrama de Gantt	8	1.2.1	Analista
1.2.3	Documentar la propuesta del proyecto	15	1.2.1	Analista
1.2.4	Entregar propuesta técnica	0	1.2.3	
2	**Desarrollo técnico del proyecto**			
2.1	**Definir requisitos del proyecto**	**30h**		
2.1.1	Crear diagrama de casos de uso	15	1.2	Analista
2.1.2	Estado del arte del contexto del proyecto	5	1.2	Analista
2.1.3	Definir requisitos tecnologicos y de plataforma	10	1.2	Analista y Programador
2.2	**Análisis**	**35 h**		
2.2.1	Crear diagrama de clases	15	2.1	Analista
2.2.2	Documentar clases	5	2.2.1	Analista
2.2.3	Diagrama de actividades	15	2.1	Analista
2.3	**Diseño**	**20 h**		
2.3.1	Identificar y clasificar usuarios	10	2.2	Analista
2.3.2	Diseñar interfaces gráficas	10	2.2	Analista y Programador
2.4	**Desarrollo del producto**	**150 h**		
2.4.1	Programación	135	2.3	Programador
2.4.2	Pruebas	15	2.4.1	Programador
2.5	**Puesta en marcha**	**6 h**		
2.5.1	Implantación	6	2.4	Programador
2.5.2	Entrega final	0	2.5.1	

Tabla 3.6: Organización del proyecto con desviaciones

Capítulo 4

Análisis y diseño del sistema

4.1. Análisis del sistema

En este capítulo se muestra un análisis del sistema, donde se detallan tanto los requisitos funcionales como los requisitos de datos que ha de cumplir la aplicación web.

Se pide desarrollar un portal web en *Angular* capaz de generar código en *C* mediante la interacción del usuario para programar dispositivos *IoT* con programación visual desarrollada con *Blockly* y guardar los proyectos en *Firebase*.

4.1.1. Diagramas de casos de uso

El proyecto va a requerir un diagrama de caso de uso para el desarrollo de la web 4.1, y otro caso de uso para la compatibilidad del la programación de nuestros dispositivos *IoT* 4.2.

Actores de la aplicación:

- **Usuario:** Representa a cualquier persona con acceso a la aplicación.
- **Administrador**: Representa al administrador de *SucreCode*.
- ***Blockly*:** Representa al sistema que genera el código ejecutable del usuario y que además lo genera en un formato capaz de cargarlo y guardarlo en la base de datos.

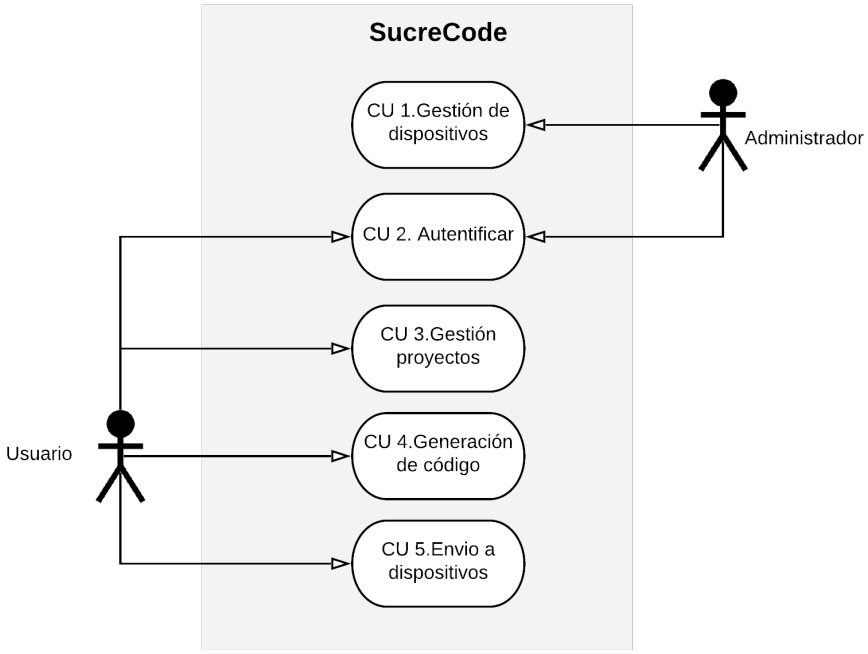

Figura 4.1: Diagrama de casos de uso de SucreCode

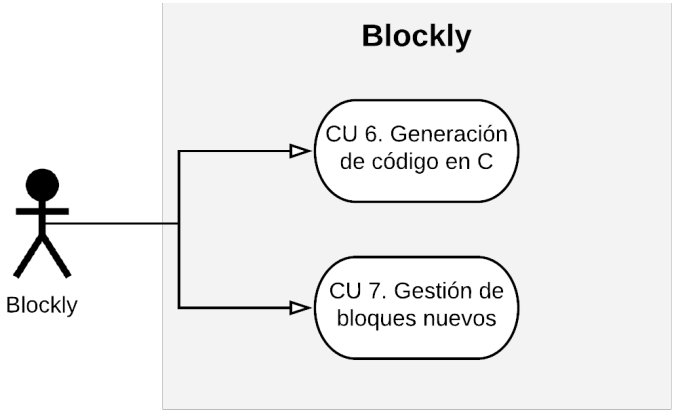

Figura 4.2: Diagrama de casos de uso de Blockly

Actor	Casos de uso
Usuario	CU 2. Autentificar CU 3. Gestión proyectos CU 4. Generación de código CU 5. Envío a dispositivos
Administrador	CU 1. Gestión de dispositivos CU 2. Autentificar
Blockly	CU 6. Generación de código en C CU 7. Gestión de bloques nuevos

Tabla 4.1: Resumen casos de uso

Las tablas 4.2 y 4.3, constituyen la especificación de los casos de uso del administrador. Las tablas 4.3, 4.4, 4.5, 4.6, 4.7 y 4.8 forman la especificación de los casos de uso del usuario. Finalmente las tablas 4.9 y 4.10 forman la especificación de los casos de uso de *Blockly*.

Nombre caso de uso	**CU 1. Gestión de dispositivos**
Complejidad	Alta
Actores	Administrador
Objetivo	Asignar y eliminar los dispositivos que puede usar cada usuario
Pre-requisitos	Para poder gestionar los dispositivos, tiene que acceder a la aplicación, tener los permisos necesarios y ser un usuario administrador
Relaciones	
Asociación	Actor Administrador
Escenarios	
Flujo básico de eventos	1. El administrador accede a la aplicación. 2. La aplicación comprueba si es un usuario administrador. 3. El administrador accede a la vista de gestión. 4. El administrador selecciona el usuario a modificar. 5. El administrador configura las placas que puede usar el usuario seleccionado. 6. La aplicación guarda la nueva configuración para el usuario modificado. 6.1 En caso de error al guardar los datos, se avisa de que no se ha podido guardar.

Tabla 4.2: CU 1. Backend - Gestión dispositivos

Nombre caso de uso	**CU 2. Autentificar**
Complejidad	Media
Actores	Usuario y Administrador
Objetivo	Poder autentificar a los usuarios para acceder a la web
Pre-requisitos	Disponer de una cuenta
Relaciones	
Asociación	Actores Usuario y administrador
Escenarios	
Flujo básico de eventos	1. El usuario accede a la aplicación. 2. La aplicación muestra la pantalla de autentificación. 3. El usuario introduce su usuario y contraseña. 4. La aplicación comprueba los datos contra Firebase y lo redirige a su área personal. 4.1. Si la autenticación falla, la aplicación informa al usuario y no lo redirige. 5.Usuario administrador 5.1. La aplicación le dará acceso al apartado de gestión y a sus propios proyectos. 6.Usuario no administrador 6.1. La aplicación solo le dará acceso a sus proyectos.

Tabla 4.3: CU 2.Frontend - Autenticar

Nombre caso de uso	**CU 3. Gestión de proyectos - Crear proyecto**
Complejidad	Media
Actores	Usuario
Objetivo	Crear un nuevo proyecto con el usuario actual
Pre-requisitos	Para poder crear proyectos, el usuario tiene que autentificarse
Relaciones	
Asociación	Actor Usuario
Escenarios	
Flujo básico de eventos	1. El usuario selecciona la opción de creación de un nuevo proyecto. 2. La aplicación le informa de que su proyecto se está creando. 3. La aplicación crea un nuevo proyecto en la base de datos con un nombre de proyecto por defecto y el id del usuario. 4. La aplicación espera a recibir la confirmación de que se ha creado el proyecto en la base de datos. 5. La aplicación redirige el usuario al nuevo proyecto que se acaba de crear.

Tabla 4.4: CU 3. Frontend - Gestion de proyecgos - Crear proyecto

Nombre caso de uso	**CU 3. Gestión de proyectos - Guardar proyecto**
Complejidad	Media
Actores	Usuario
Objetivo	Guardar el estado del proyecto
Pre-requisitos	Para guardar proyectos, el usuario tiene que autentificarse y acceder a un proyecto
Relaciones	
Asociación	Actor Usuario
Escenarios	
Flujo básico de eventos	1. La aplicación muestra los proyectos. 2. El usuario accede a alguno de sus proyectos. 3. La aplicación cada cierto tiempo solicita a Blockly que le de el código del proyecto. 4. La aplicación obtiene el código y lo actualiza en la base de datos. 5. El usuario sale del proyecto. 6. La aplicación muestra los proyectos del usuario, solicita el código del proyecto que se ha cerrado a Blockly y guarda el estado del proyecto del que acaba de salir en la base de datos.

Tabla 4.5: CU 3. Frontend - Gestión de proyectos/Guardar proyecto

Nombre caso de uso	**CU 3. Gestión de proyectos - Cargar proyecto**
Complejidad	Alta
Actores	Usuario , Blockly y Firebase
Objetivo	Acceder a un proyecto y recuperar el estado con el que se guardó
Pre-requisitos	Para cargar proyectos, el usuario tiene que autentificarse y acceder a un proyecto
Relaciones	
Asociación	Actores usuario, Blockly y Firebase
Escenarios	
Flujo básico de eventos	1. La aplicación muestra los proyectos del usuario. 2. El usuario accede a alguno de sus proyectos. 3. La aplicación obtiene el código del usuario en ese proyecto de la base de datos. 4. La aplicación manda el código que ha obtenido a Blockly. 4. Blockly recibe ese código y lo muestra en el proyecto.

Tabla 4.6: CU 3. Frontend - Gestión de proyectos/Cargar proyecto

Nombre caso de uso	**CU 4. Generación código**
Complejidad	Media
Actores	Usuario y Blockly
Objetivo	Crear código ejecutable mediante la interacción del usuario y Blockly
Pre-requisitos	Para crear código ejecutable, el usuario tiene que autentificarse y acceder a un proyecto
Relaciones	
Asociación	Actores Usuario y Blockly
Escenarios	
Flujo básico de eventos	1. La aplicación muestra los proyectos del usuario. 3. El usuario accede a alguno de sus proyectos. 4. Blockly carga el aspecto visual de los bloques que tiene programados. 5. Blockly carga el código asociado a los bloques. 6. Blockly inserta los bloques al menú de su interfaz para que el usuario pueda interactuar con ellos. 7. El usuario interactúa con la interfaz de Blockly. 8. Blockly genera y actualiza el código a medida que el usuario hace cambios en el proyecto.

Tabla 4.7: CU 4. Frontend - Generación de código

Nombre caso de uso	**CU 5. Envío a dispositivos**
Complejidad	Alta
Actores	Particle, Usuario y Firebase
Objetivo	Obtener el token de autentificación para enviar código a los dispositivos IoT que el usuario tenga acceso
Pre-requisitos	Acceder a la aplicación y estar autenticado
Relaciones	
Asociación	Actor Particle
Escenarios	
Flujo básico de eventos	1. La aplicación inicia sesión con Particle para obtener un token válido. 2. Particle genera el token y lo manda a la aplicación. 3. La aplicación muestra los proyectos del usuario. 4. El usuario accede a un proyecto o genera uno nuevo. 5. La aplicación carga el proyecto. 6. La aplicación obtiene los dispositivos disponibles de Particle. 7. La aplicación obtiene un listado de los dispositivos asignados al usuario de Firebase. 8. La aplicación muestra los dispositivos disponibles para el usuario. 9. El usuario selecciona el dispositivo al que quiere enviar su código. 10. El usuario pulsa el botón enviar. 11. La aplicación realiza una petición post con el token que ha obtenido previamente al servicio rest de Particle mandando el código del usuario 12. La aplicación muestra por pantalla si se ha enviado correctamente el código o si han habido errores.

Tabla 4.8: CU 5. Frontend - Envío a dispositivos

Nombre caso de uso	**CU 6. Generación de código en C**
Complejidad	Alta
Actores	Blockly
Objetivo	Ser capaz de generar código en C
Pre-requisitos	Tener Blockly configurado para generar el código en C
Relaciones	
Asociación	Actor Blockly
Escenarios	
Flujo básico de eventos	1. Blockly genera la estructura básica. 2. Blockly detecta si hay bloques en el área de trabajo. 3. Blockly interpretar esos bloques y añade el código a la estructura que ha formado al principio.

Tabla 4.9: CU 6. Blockly - Interacción con los bloques

Nombre caso de uso	**CU 7. Gestión de bloques nuevos**
Complejidad	Media
Actores	Blockly
Objetivo	Ser capaz de incorporar bloques con nuevas funcionalidades
Pre-requisitos	Tener Blockly configurado para generar el código en C
Relaciones	
Asociación	Actor Blockly
Escenarios	
Flujo básico de eventos	Secuencia normal de pasos 1. Blockly genera la estructura básica. 2. Blockly carga el aspecto visual de los bloques que tiene programados. 3. Blockly carga el código asociado a los mismos. 4. Blockly inserta los bloques al menú de su interfaz para que el usuario pueda interactuar con ellos. 4.1 En caso de que se llamen bloques que no existen, Blockly mostrara un rectángulo gris indicando que ese bloque no existe. 4.2 En el caso de que el código asociado a un bloque este mal configurado, se muestra un mensaje de error por la consola y no actualiza el código actual.

Tabla 4.10: CU 7. Blockly - Gestión de bloques nuevos

4.1.2. Requisitos de datos de la aplicación

RD01 Usuario

Nombre	Tipo	Descripción
Id	String	Identificador del usuario, se genera automáticamente
Email	String	Correo electrónico del cliente para iniciar sesión
Password	String	Contraseña encriptada del cliente con la que inicia sesión
Rol	String	Define el rol que tendrá el usuario en la aplicación web

Tabla 4.11: RD 01. Usuario

RD02 Proyecto

Nombre	Tipo	Descripción
Name	String	Nombre del proyecto
Date	String	Fecha de la última modificación
Owner	String	Id del usuario propietario del proyecto
Xml	String	Contenido del proyecto

Tabla 4.12: RD 02. Proyecto

RD03 Placas usuario

Nombre	Tipo	Descripción
Devices-id	Array[String]	Listado de dispositivos a los que el usuario tiene acceso
User-ID	String	Id del usuario con acceso al listado de dispositivos

Tabla 4.13: RD 03. Placas usuario

RD04 Placas Particle

Nombre	Tipo	Descripción
Id	String	Identificador único de la placa
Name	String	Nombre de la placa
Last Heard	Date	Última vez que se mandó información a la placa
Last Handshake	Date	Última vez que se estableció conexión con la placa
Serial Number	String	Identificador único para reclamar la placa manualmente
Type	Integer	Indica el modelo de la placa
Network	String	Red de Particle a la que pertenece

Tabla 4.14: RD 04. Placas usuario

4.2. Diseño de la arquitectura del sistema

En esta sección se presenta la arquitectura del sistema desarrollado, así como una breve explicación de las herramientas y tecnologías usadas, la cual se explica de forma más detallada en el Capítulo 5. El sistema está desarrollado siguiendo la filosofía de aplicación sin servidor. En la Figura 4.3 se observa el esquema de esta estructura.

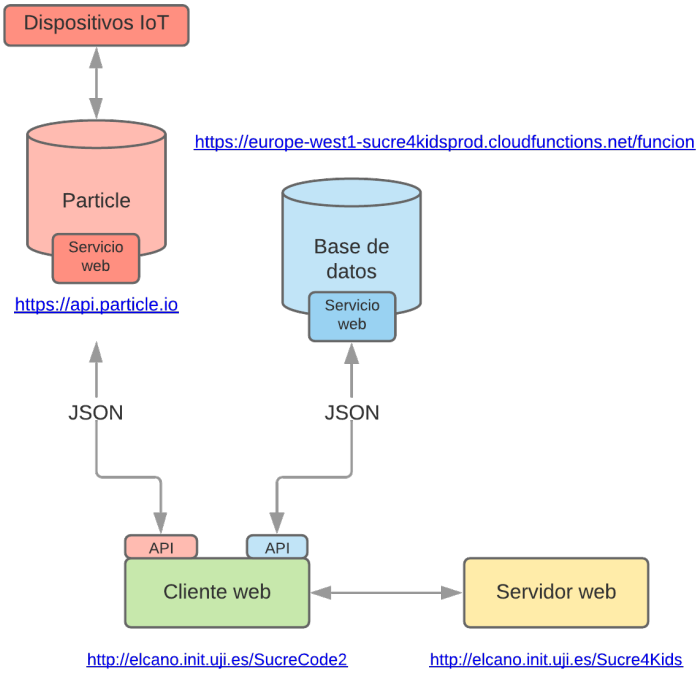

Figura 4.3: Esquema de la arquitectura

La comunicación entre el cliente con los servicios de *Firebase* y *Particle* se realizan mediante el intercambio de datos en formato *JSON*[20] a través de conexiones seguras *HTTPS* [1][7]. La comunicación se realiza a través de las *APIs* [2][21] que ofrecen los mismos, estas *APIs* usan el estandar *OAuth* [4] [19] para establecer un sistema de seguridad basado en *tokens*. Los *tokens* permiten una comunicación segura entre la aplicación y los proveedores de servicios (*Firebase y Particle*), ya que se proporciona acceso a los datos y a la vez se protegen las credenciales de la cuenta.

4.3. Diseño de la interfaz

En esta sección se presentan las interfaces de la aplicación, el diseño de las interfaces se discutió con el encargado del proyecto hasta obtener el diseño deseado e iniciar su implementación. Conforme al paso del tiempo, surgieron cambios en el diseño de algunos componentes como en la visualización del código en la vista de proyecto y se propuso la creación vistas adaptables para dispositivos móviles.

En la Figura 4.4 se aprecian las páginas que contiene la aplicación web y por las que los usuarios pueden navegar el cliente de la aplicación y en la Figura 4.5 se muestran las páginas de la aplicación por las que los usuarios administradores pueden navegar. Como podemos ver, la aplicación diferencia los roles de los usuarios y muestra un contenido u otro.

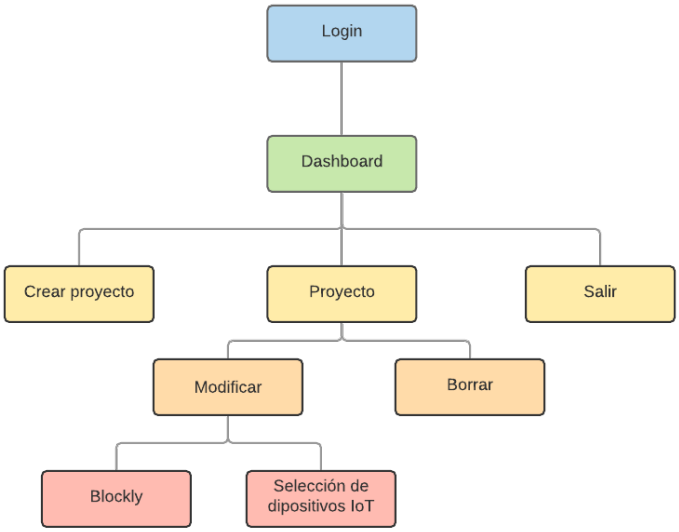

Figura 4.4: Mapa de navegación de la aplicación desde el usuario

[1]Protocolo seguro de transferencia de hipertexto - *Hypertext Transfer Protocol Secure*
[2]Interfaz de programación de aplicaciones - *Application Programming Interface*

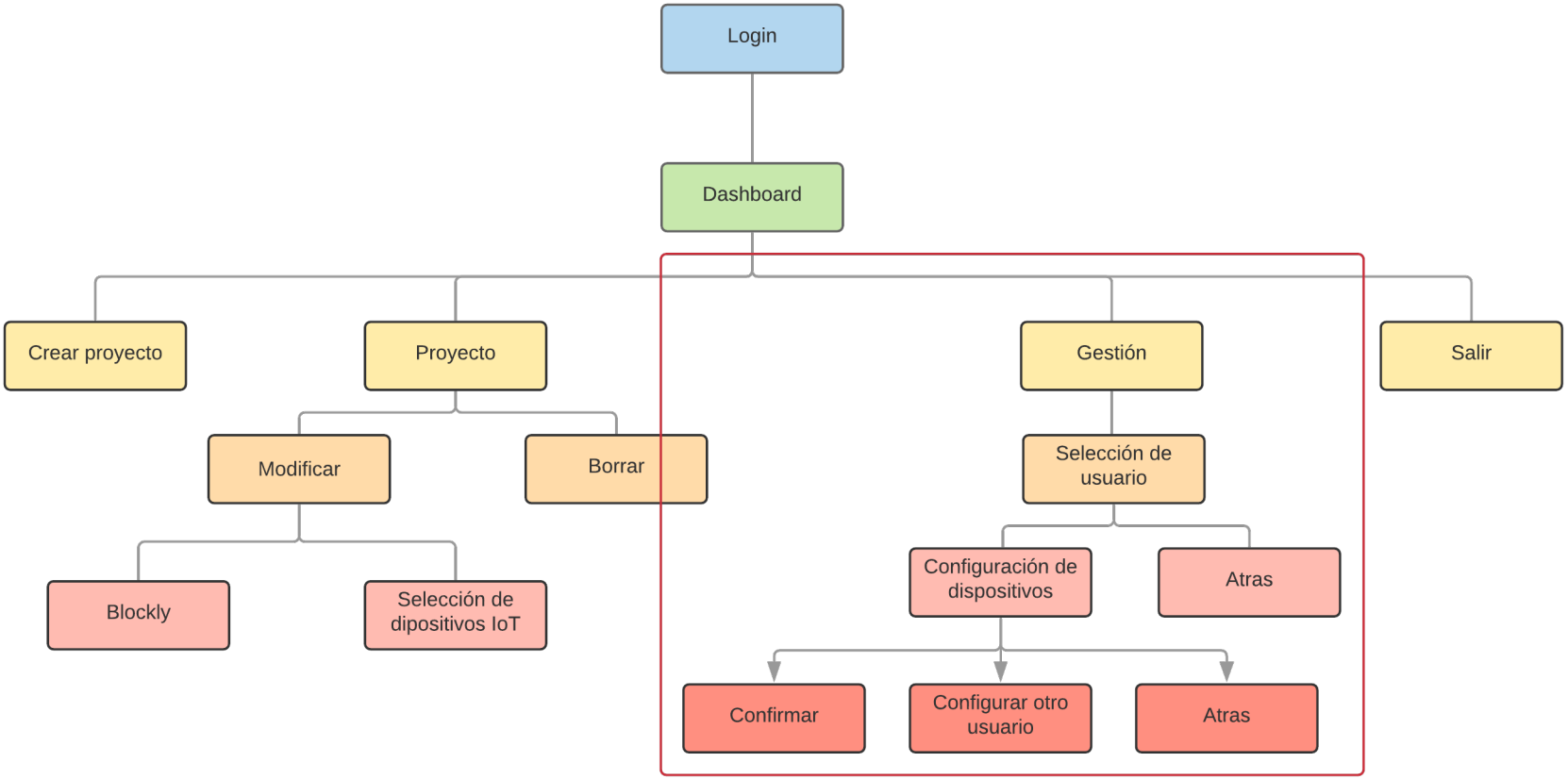

Figura 4.5: Mapa de navegación de la aplicación desde el administrador

Se ha decidido implementar los 2 roles en el mismo servidor. Por ello al acceder al portal web *SucreCode* nos mostrará el login, una vez autenticados nos enviará a un dashboard distinto según el rol Figuras 4.4 y 4.5.

Una vez se ha iniciado sesión, la aplicación redirige al usuario a la vista de dashboard, aquí el menú superior ya muestra el nombre de la cuenta con la que hemos accedido, la opción de acceder a la vista de administración en el caso de tener los permisos necesarios y la opción de cerrar la sesión. Además de mostrar los proyectos los usuarios y darle la opción de crear nuevos proyectos, modificarlos y eliminarlos, en este caso creamos un proyecto nuevo.

Al crear un nuevo proyecto, la aplicación redirige al usuario de forma automática al nuevo proyecto que se acaba de crear que corresponde a modificar proyecto, desde esta pantalla, el usuario puede generar el código de su proyecto mediante *Blockly* y seleccionar la placa a la que se le ha de mandar el código y modificar el nombre del proyecto.

Finalmente en la vista de gestión Figura 4.5, el usuario administrador puede ver un listado de los usuarios registrados en la aplicación y seleccionar el que quiera gestionar. Una vez seleccionado un usuario, el administrador ya puede pasar a la siguiente fase de la administración del usuario, donde se mostrará una caja doble donde aparecen en una caja las placas asignadas y en otra las placas sin asignar al usuario. El administrador puede modificar esta configuración, una vez ha acabado de realizar la nueva configuración de placas disponibles para el usuario, este pasa a la última fase de la administración del usuario que sería guardar la nueva configuración, salir o modificar otro usuario.

Una vez planteada la estructura de navegación y funcionalidad de las páginas tanto desde la vista de un cliente como desde la vista de un administrador, se plantean una serie de diseños previos a la implementación de las vistas.

El primero es el de la vista de inicio de sesión tal como se muestra en la Figura 4.6 y en la Figura 4.7 visto desde el móvil.

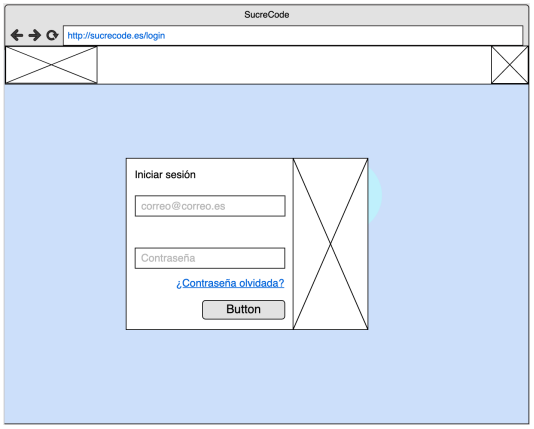

Figura 4.6: Diseño inicio sesión

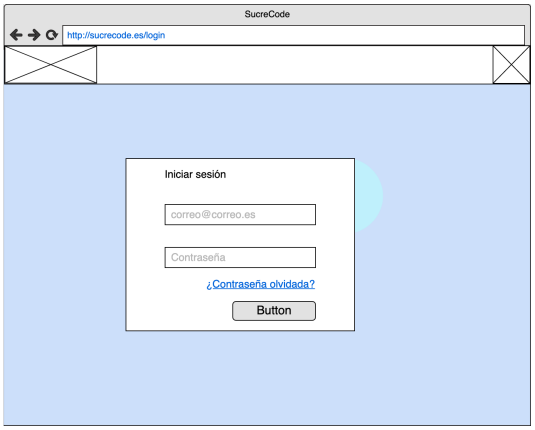

Figura 4.7: Diseño inicio sesión desde móvil

A continuación se muestra el diseño del área personal del usuario en la Figura 4.8, en este caso la vista en móvil es igual, únicamente se ajustan la cantidad de elementos en pantalla.

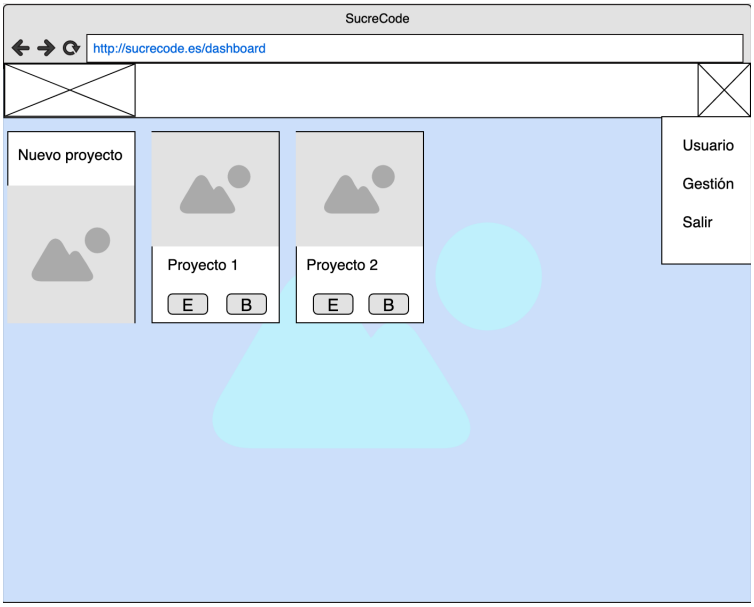

Figura 4.8: Diseño área personal

Paralelamente se muestra el diseño del proyecto Figura 4.9, donde el usuario puede generar el código de su proyecto y su versión móvil Figura 4.10.

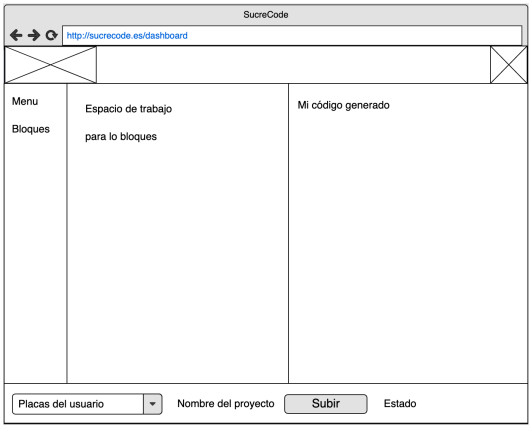

Figura 4.9: Diseño proyecto

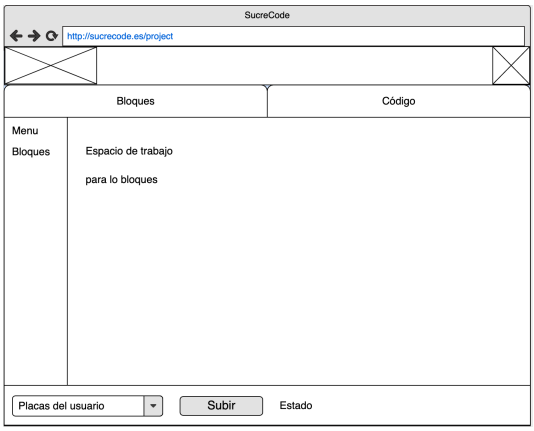

Figura 4.10: Diseño proyecto desde móvil

Finalmente se muestra el diseño de la vista de gestión 4.11, al igual que con el área personal, esta vista se adaptará al tamaño de la pantalla manteniendo el mismo diseño.

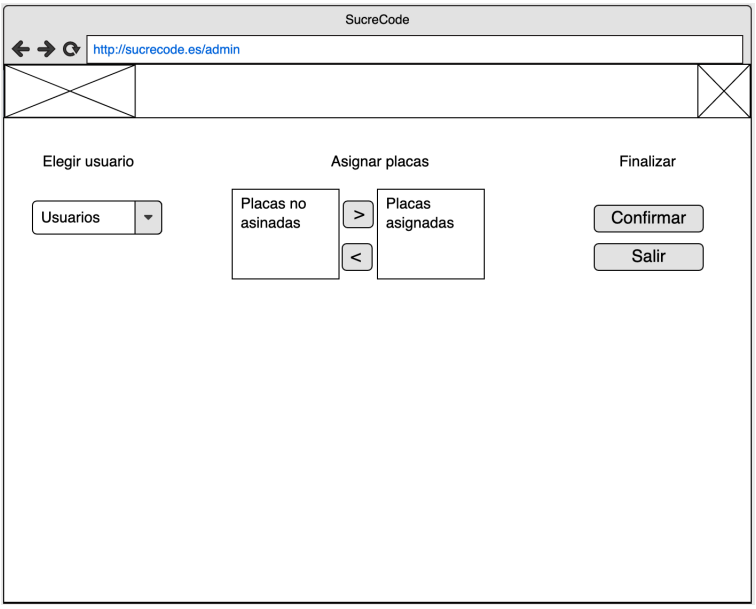

Figura 4.11: Diseño gestión

4.4. Diagrama de clases

Los diagramas de clases sirven para representar de una forma visual la información y la estructura del proyecto. Para poder cumplir con los requisitos descritos de la aplicación, se crea un diagrama de clases Figura 4.12.

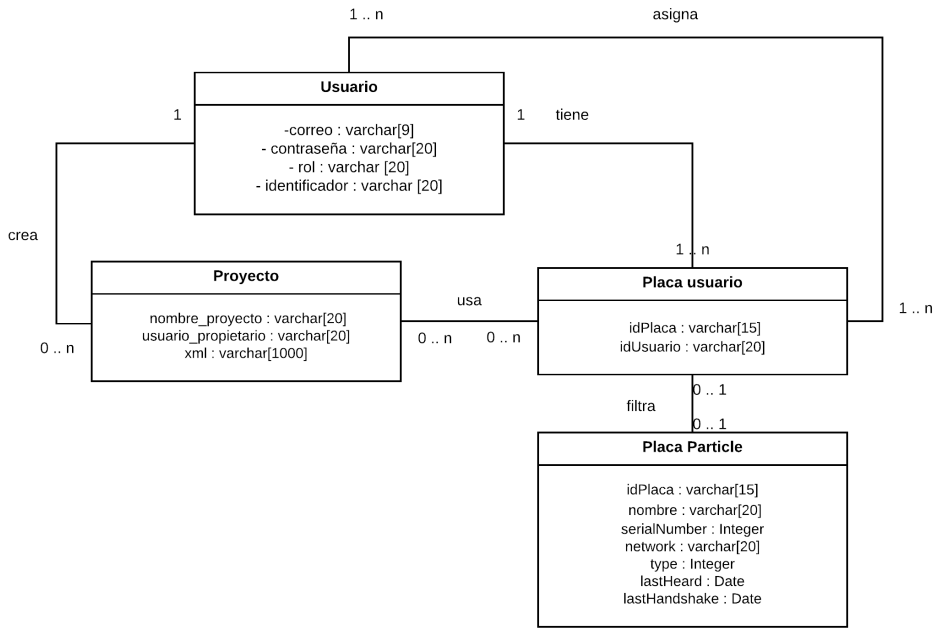

Figura 4.12: Diagrama de clases del sistema

Los usuarios administradores también tiene sus propios proyectos y dispositivos *IoT* asignados, se diferencian de un usuario normal por la asignación del campo rol.

Capítulo 5

Implementación y pruebas

5.1. Detalles de implementación

Angular, como se ha mencionado anteriormente, es un entorno de trabajo de *TypeScript* que ha ido ganando terreno con el paso del tiempo hasta convertirse es uno de los entornos de trabajo más famosos y utilizados. Es un entorno de trabajo progresivo, ya que está dividido en módulos y estos se van llamando y usando a medida que se van requiriendo.

Este entorno de trabajo proporciona una estructura de ficheros muy organizada, al crear un componente desde *Angular-CLI*, se genera toda la estructura de fichero de proyecto ya configurada de forma automática. A medida que vamos añadiendo funcionalidades al proyecto, también lo hacemos desde *Angular-CLI* para que genere todos los archivos necesarios y añada de forma automática las dependencias de los ficheros que se acaban de crear a los archivos de configuración del proyecto. Al añadir las dependencias automáticamente, lo hace de forma óptima siguiendo un patrón *LazyLoad*[3], lo cual permite que la aplicación cargue las librerías y los módulos a medida que los va necesitando, esto mejora el tiempo de carga y el rendimiento.

Otra de las características de este entorno de trabajo es la capacidad de trabajar tanto con móviles como con ordenadores y de programarlo de forma simultánea, esto se logra gracias al uso de *Media Queries*[12] y de que muchas de las librerías de *Angular* ya están pensadas y adaptadas para funcionar en móviles.

Además *Angular* tiene una estructura de ficheros muy concreta por lo que el patrón de diseño que más se adapta al entorno de desarrollo de *Angular* es el patrón *MVVM*. Este patrón permite que la aplicación sea mantenible y escalable, permite el desacople del modelo y que no tengamos que preocuparnos de su renderizado.

También se han utilizado una serie de librerías que han permitido mejorar el funcionamiento de la aplicación:

- *Angular Material*[9]: Es un modulo que nos permite implementar diseños ya predefinidos y que siguen unos estándares a nuestra página web. Esto mejora el aspecto visual y facilita que el diseño

sea *responsive*.

- *RxJS*[17]: Librería para ampliar la gestión de las eventos asíncronos añadiendo nuevas estructuras y métodos más actuales de trabajo.
- *Angular-Router*[2]: Se encarga del enrutado de la aplicación, asocia las rutas con los componentes y las restricciones de las mismas de una forma sencilla lo cual hace que resulte fácil de entender. Permite parametrizar las rutas de forma cómoda y además se integra bien en el código de la aplicación haciendo que su uso sea intuitivo.
- *Ngx-Blockly*: Librería de envoltura de *Blockly*, facilita su administración y su integración con la aplicación.
- *Firebase-CLI*[6]: Permite hacer llamadas a la base de datos para obtener los datos y añadir datos nuevos.
- *Particle API*: Poniendo los parámetros de configuración podemos gestionar los dispositivos *IoT* desde la aplicación web.

5.2. Configuración del servicio web Firebase

Firebase es una base de datos no relacional que funciona en la nube de *Google*. La base de datos de *Firebase* almacena y sincroniza los datos con nuestra base de datos, todo esto se aloja en la nube. Estos datos que están en la nube son almacenadas en *JSON* y se pueden agregar reglas para permitir *request* con token o solo desde una *URL*.

Para este proyecto usamos una base de datos *Firebase*, la cual está en un servidor de *Google*. Para acceder y configurar la base de datos, se ha de usar la aplicación web que ofrece *Google*. Las bases de datos *Firebase* van asociadas a una cuenta de correo, por lo que es necesario tener una cuenta de correo válida para poder crear la base de datos.

Cuando tengamos el *proyecto Firebase* que será la base de datos, debemos preparar la estructura de datos que almacenará los datos de los proyectos, roles y placas de los usuarios de *SucreCode*. Como sabemos, *Firebase* es una base de datos no relacional, por lo que las tablas no existen, en su lugar usamos *colecciones y documentos*. En la figura B.3, se pueden ver las colecciones que usa la aplicación web con sus documentos y campos por documento. En las colecciones podemos añadir, borrar o modificar documentos y campos de los documentos desde *SucreCode* gracias a la *API* que nos ofrece *Firebase*.

Firebase ofrece una versión gratuita para poder hacer pruebas y depurar la aplicación y dos planes de pago. Para este proyecto se ha optado por un plan de pago por uso, donde el coste varia según la cantidad de llamadas, tiempo de procesamiento de las funciones y usuarios que se creen.

5.3. Configuración del servicio web Particle

Para realizar la gestión de los dispositivos *IoT*, se ha creado una cuenta de *Particle* y se han asociado los dispositivos que se van a usar en el proyecto. De esta manera podemos usar la *API* que ofrece *Particle*

iniciando sesión desde *SucreCode* con la cuenta que hemos creado en *Particle* y obtener un *token* para poder realizar una comunicación segura.

Usando las funcionalidades de la *API* es como obtendremos la información necesaria para poder trabajar con los dispositivos *IoT* y también como mandaremos el código al dispositivo seleccionado por el usuario. Por motivos de seguridad, se han configurado los *tokens* que se obtiene de la *API* para que caduquen cada 3 días.

5.4. Estructura de ficheros de nuestro proyecto Angular

En la Figura 5.1 tenemos la jerarquía de directorios y ficheros de *Angular* donde se pueden diferenciar los distintos apartados del patrón utilizado, además de los componentes, router y ficheros de configuración.

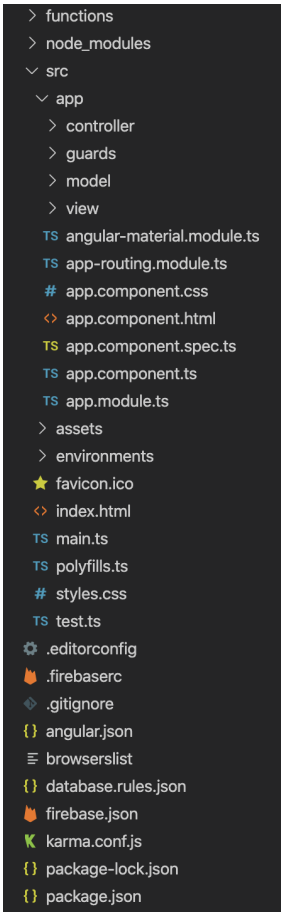

Figura 5.1: Estructura de ficheros de nuestro proyecto angular

Esta estructura de ficheros, se compila usando *Angular-cli* para obtener una versión de la aplicación para producción. El resultado de compilación de los ficheros, es una carpeta que contiene un *index, assets* y una serie de ficheros *JavaScript*. No es necesario modificar estos archivos si la configuración de la versión de desarrollo es igual al entorno de producción, si las *URLs* cambian, se ha de adaptar los

archivos para que el servidor web pueda localizar los recursos.

Para que la versión de producción funcione, se ha de poner la carpeta que se genera en el servidor web *Apache* como cualquier otra web, ya que la configuración de la aplicación ya la hemos hecho previamente.

5.4.1. Funciones de Firebase

Una de las características de usar *Firebase* en este proyecto es cómo se integra con *Angular*. En la parte superior de la Figura 5.1 se ve un directorio especial llamado *functions*, en este directorio podemos subir, actualizar y borrar las consultas de la base de datos para que todas las aplicaciones que usen la base de datos, tengan las mismas consultas. En *functions* escribiremos las consultas y desde la línea de comandos gracias a las funcionalidades de *Firebase-CLI* subimos los cambios a la base de datos. Estas funciones se pueden ver en la Figura B.6. Usar las consultas de esta manera permite mejorar la mantenibilidad de la aplicación. Podemos ver un ejemplo de consulta en el Código A.1, los datos que obtendremos de la base de datos son en formato *JSON* como los que se muestran en la Figura 5.10.

5.4.2. Estructura y configuración de Blockly

El segundo directorio que vemos en la imagen 5.1 es *node_modules*. Esta carpeta almacena todas las librerías del proyecto de programación, una de las librerías más importantes es *Ngx-Blockly* la cual envuelve *Blockly* para que se integre con *Angular*. Una de las necesidades del proyecto es la necesidad de adaptar *Blockly* para que generara código en *C adaptado a Arduino*, para ello se ha modificado el código de librería, creando un nuevo generador y unos bloques específicos para el proyecto.

Blockly tiene una estructura de directorios para clasificar la funcionalidad de cada fichero, como vemos en la Figura 5.2 los directorios más relevantes son:

- *Generators*: Este fichero contiene el núcleo de la aplicación, donde se forman las diferentes estructuras que luego irán insertando el código. Veamos un fragmento del código en Código A.2. Además también están los ficheros que asignan el código a los bloques como se muestra en Código A.3.
- *Blocks*: En este directorio están los ficheros encargados de gestionar el aspecto visual de los bloques en el Código A.4 podemos ver la definición de un bloque.
- *Build.py*: Archivo ejecutable en *Python* que se encarga de compilar los ficheros mencionados previamente y generar los ficheros comprimidos que se usan en la aplicación.

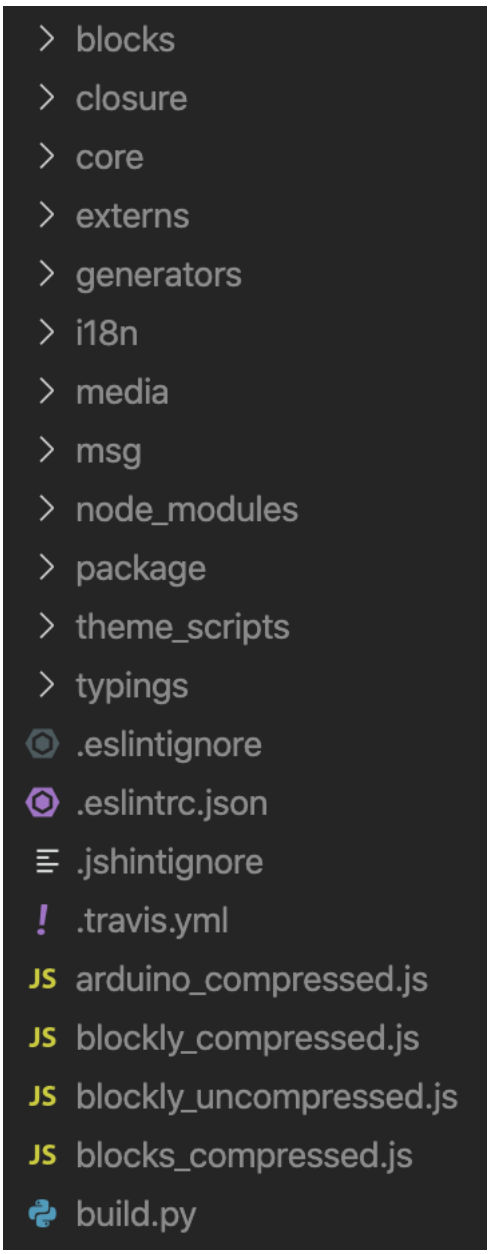

Figura 5.2: Estructura de Blockly

5.4.3. Implementación de la aplicación

En tercer lugar tenemos el directorio *src*, en este directorio es donde tenemos toda la lógica de la aplicación, las carpetas y ficheros más importantes son:

- *Controller*: Esta carpeta contiene todos los servicios encargados de comunicarse con todos los componentes externos de la aplicación, estos pasan los resultados obtenidos al servicio asociado de *model* y este a controlador de la vista de *view*. Uno de los servicios más importantes es el de *Particle*, en el Código A.5 podemos ver como la aplicación manda código a una placa.

- *Guards*: En esta carpeta están unos ficheros especiales llamados *Guards*, estos son los encargados de controlar quien puede acceder a la vista o vistas que está protegiendo. Tenemos un ejemplo en Código A.6.
- *View*: Contiene los componentes de la aplicación, con su vista y sus estilos. Un ejemplo de componente se muestra en el Código A.7.

  Además el componente también contiene una parte *HTML* la cual podemos apreciar en el Código A.8, esta tiene una serie de características interesantes como que se comunica con su controlador, se le puede poner cierta lógica dentro del *HTML* y finalmente si nos fijamos en la última línea, podemos ver una notación especial, esta es una instrucción propia de *Angular*, indica que a partir de esa línea se vaya añadiendo el resto del código que conforma la vista.
- *Routing-module*: El fichero de *routing* es donde se configuran todas las rutas de navegación de la aplicación y se indican las vista que irán protegidas con *guards*. El Código A.8 es un ejemplo de *router*.

5.4.4. Configuración del entorno de la aplicación

Finalmente los archivos de configuración, los más importantes son *enviroment* y *angular.json*.

Enviroment nos permite poner variables de entorno para poder acceder y leerlas desde cualquier lugar de la aplicación. En este caso, contiene los *tokens* de autentificación de la base de datos como vemos en el Código A.10.

El archivo *angular.json* contiene la configuración del proyecto, de toda la información que contiene nos centraremos en los estilos y el apartado *scripts*, en este apartado, indicamos los ficheros que debe cargar para el correcto funcionamiento de *Blockly*.

5.5. Resultados de la interfaz de usuario

El resultado del diseño utilizando las librerías de *Angular Material* y *Bootstrap4* es el que se muestra en las Figuras 5.3 y 5.4 siguiendo los diseños propuestos para el inicio de sesión. En la Figura 5.4 se puede apreciar el diseño más compacto para dispositivos móviles.

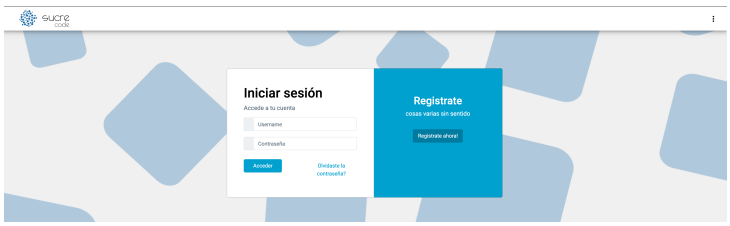

Figura 5.3: Página de inicio

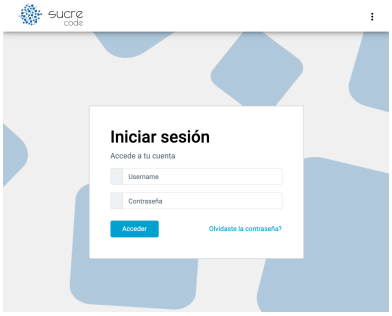

Figura 5.4: Página de inicio en móvil

El área personal como se puede ver en la Figura 5.5, ha sufrido cambios respecto al diseño propuesto. Se ha añadido la fecha de modificación del proyecto y se han cambiado los botones por iconos para que la interfaz sea más intuitiva. Respecto a *Sucre4Kids* esto supone un gran cambio, ya que no dispone de esta pantalla debido a que esta pantalla es posible gracias a la gestión de usuarios y proyectos. En la Figura 5.6 se aprecia, al igual que en todas las pantallas anteriores, que los elementos de la pantalla se adaptan al espacio disponible.

Figura 5.5: Página de área personal

Figura 5.6: Página de área personal en móvil

Al acceder a un proyecto, se muestra la interfaz de *Blockly* en las Figuras 5.7 y 5.8 aquí también podemos ver cambios de diseño respecto al diseño planteado originalmente. Se ha añadido una flecha en la barra lateral de bloques que permite esconderla para ganar espacio en la pantalla y mejorar la interacción del usuario con los bloques. Respecto a la aplicación original, esta interfaz supone un gran cambio, ya que tenemos la pantalla dividida en dos secciones distintas, hay un menú en la parte inferior de la pantalla para cambiar el nombre del proyecto y la selección de dispositivos, finalmente esta pantalla es completamente responsive lo que permite que se ajuste al tamaño de pantalla al reescalar la ventana y que alterne el diseño como se ve en la Figura 5.8.

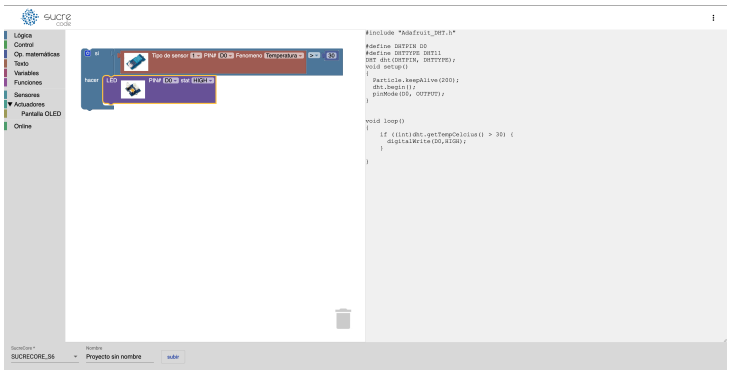

Figura 5.7: Página de proyecto

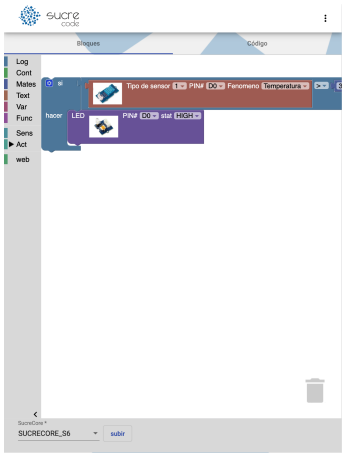

Figura 5.8: Página de proyecto en móvil

Finalmente, la vista de gestión como se muestra en la Figura 5.9, esta es la que ha sufrido la modificación más grande respecto al diseño original. Para facilitar la tarea de gestión del administrador, esta pantalla se ha implementado siguiendo una secuencia de pasos la cual se aprecia en la parte superior de la imagen. Esta secuencia de pasos requiere se va validando para evitar errores y permite avanzar y retroceder en caso de que se quiera cambiar algún parámetro.

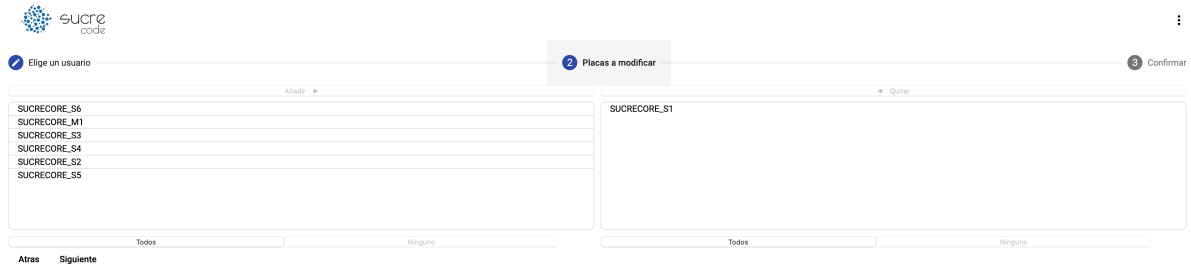

Figura 5.9: Página de gestión

5.6. Verificación y validación

Para comprobar que la aplicación funciona correctamente se ha usado la consola del navegador y el *localStorage*. Con *localStorage* podemos almacenar la información en formato clave, valor y con la consola podemos ver los datos a medida que van llegando como se muestra en la Figura 5.10. De esta manera podemos almacenar datos e ir comprobando y contrastando la información.

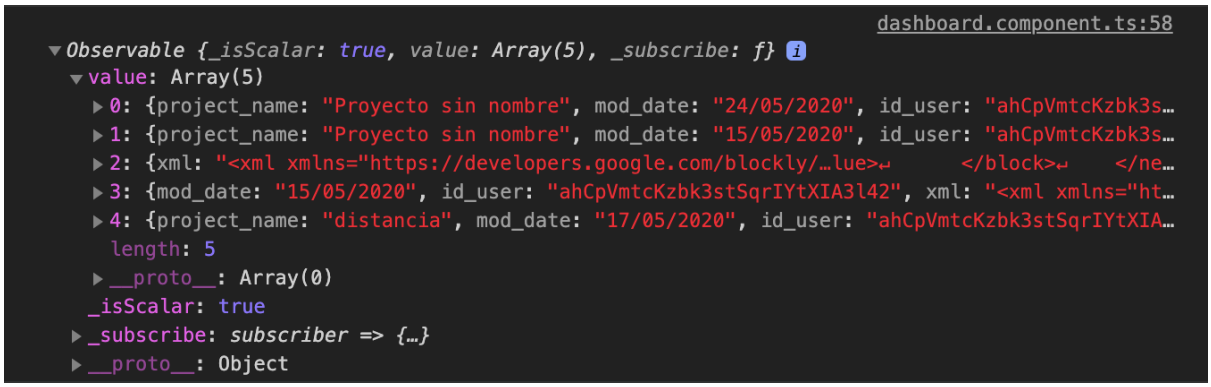

Figura 5.10: Datos mostrados por consola

Se han realizado las siguientes pruebas sobre la aplicación:

- Comprobar el acceso de los usuarios autenticados, para comprobar el acceso de usuarios, se ha intentado acceder con un usuario con los datos mal introducidos, con un usuario que no existe y con un usuario correcto. El funcionamiento de la aplicación ha sido el esperado, en el primer caso, ha mostrado el mensaje de error, en el segundo caso también y en el último caso ha accedido a sus proyectos.

- Comprobar el acceso de los usuarios sin autenticar.
- Comprobar el acceso con datos erróneos.
- Comprobar los roles de usuario al acceder, al comprobar los roles, simplemente se ha accedido a la aplicación con un usuario cliente, la aplicación no le muestra el enlace de acceso a la página de gestión. Además, escribiendo la *URL* la aplicación tampoco le da acceso, redirige al usuario a sus proyectos. Con un usuario administrador la aplicación si le muestra el enlace de acceso a la página de gestión, al acceder por *URL* el administrador accede sin problemas.
- La sesión se queda almacenada aunque se cierre la web, para comprobar que la sesión se queda guardada aunque se cierre la web, se ha iniciado sesión, se ha cerrado el navegador y se ha vuelto a acceder.
- La sesión se cierra correctamente, el usuario tiene que poder cerrar sesión de forma segura, se ha comprobado que la sesión se cierra correctamente.
- La asignación de placas del usuario es correcta.
- Correcto funcionamiento de la modificación de placas del panel del administrador.
- Se modifican las placas del usuario seleccionado, la comprobación de las placas se ha realizado contrastando los datos obtenidos por consola con los asignados en la base de datos.
- Protección de vistas al acceder por *URL*.
- Los usuarios no administradores no tienen acceso a la vista de gestión desde el menú de la web.
- Los usuarios no pueden acceder a la página de gestión mediante *URL*.
- Los usuarios no pueden acceder a los proyectos de otros usuarios.
- Los proyectos se guardan correctamente, la comprobación del guardado de proyectos se ha realizado contrastando los datos obtenidos por consola con los asignados en la base de datos.
- Los proyectos se cargan correctamente.
- El código generado por *Blockly es correcto*.
- El usuario puede subir el código de su proyecto a su dispositivo *IoT* seleccionado.
- El código se sube al dispositivo *IoT* correcto, para comprobar que el código del usuario se sube al dispositivo correcto, se ha subido un programa a varias placas y sé ha comprobado que este se ejecutaba en el dispositivo correcto.

Uno de los métodos de comprobación utilizados, es ver los eventos ejecutados de cada componente, estos se muestran por consola los datos, de esta manera podemos comprobar si los datos obtenidos son correctos.

Otro de los métodos usados es comprobar los datos que se muestran con lo que están almacenados en la base de datos, los datos tienen que coincidir, ya que en caso contrario estamos perdiendo información en algún paso del tratamiento de los datos.

Capítulo 6

Conclusiones

Realizar este proyecto no solo me ha servido para aprender sobre nuevas tecnologías sino que también me ha servido para aprender como se crea un proyecto, los pasos a seguir y la impotencia de la organización para poder sacar el proyecto adelante.

Por un lado, aprender a manejar *Angular* ha sido uno de los puntos fuertes del proyecto dado que es una tecnología nueva y que tiene mucho futuro por delante. Además aprender a integrarlo con otras tecnologías puede ser bastante útil para otros proyectos o para mejorar los proyectos existentes.

Por otro lado, usar una base de datos como *Firebase* también ha sido bastante útil, ya que aunque se manejen de una forma distinta a las bases de datos relacionales, cada vez se usan más y están cobrando más importancia dado que cada vez la demanda de datos es mayor.

Finalmente, conocer cómo funciona una empresa y cómo se organiza, planifica y lleva a cabo los proyectos es una de las cosas que más curiosidad me ha suscitado.

En conclusión, ha sido una experiencia que me ha permitido desarrollar mis habilidades en el ámbito de la informática y conocer más de cerca el mundo laboral.

Apéndice A

Anexo I: Código de ejemplo de la implementación

En este anexo, se explica más en detalle como está implementada la aplicación mostrando el código fuente.

El Código A.1 muestra cómo se llama a la base de datos *Firebase* para establecer los dispositivos a los que tiene acceso un usuario.

```
exports.setDevices= functions
.region('europe-west1')
.https.onCall(async (data, context)=>{
    return admin.firestore().collection("usuarios").doc(data.ref).set({
        devices : data.devices,
        id_user : data.user
    })
    .then(done =>{
        console.log("actualizado")
        return true
    }
    ).catch((error => {
        console.log("error al actualizar");
        return false
        }));;
});
```

Código A.1: Ejemplo consulta Firebase

Los Códigos A.2, A.3 y A.4 son parte del código creado para adaptar *Blockly* a *C*. El Código A.2 es el encargado de generar las estructuras necesarias y la configuración básica para poder ir generando el código de forma correcta.

```
var analog_pins = ["A0", "A1", "A2", "A3", "A4", "A5", "A6", "A7", "DAC", "
    WKP"].map(pinmap);
var digital_pins = ["D0", "D1", "D2", "D3", "D4", "D5", "D6", "D7"].map(
    pinmap);
```

```
var pwm_pins = ["D0", "D1", "D2", "D3", "A4", "A5", "WKP", "RX", "TX"].map(
    pinmap);

var profile = {
  arduino: {
    description: "Arduino standard-compatible board",
    digital: digital_pins,
    analog: analog_pins,
    pwm: pwm_pins,
    serial: '115200',
  },
};

profile["default"] = profile["arduino"];

Blockly.Arduino.init = function(workspace) {
  Blockly.Arduino.definitions_ = Object.create(null);
  Blockly.Arduino.setups_ = Object.create(null);
  Blockly.Arduino.functionNames_ = Object.create(null);
```

Código A.2: Ejemplo generador

Tal y como se aprecia en el Código A.3, tiene el código en *C* que se añadirá cuando se ponga el bloque en la interfaz de trabajo de *Blockly* en el proyecto.

```
Blockly.Arduino['grove_led_bar'] = function(block) {
    var dropdown_pin = this.getFieldValue('PIN');
    var next_pin='D'+(parseInt(dropdown_pin.slice(1,dropdown_pin.length))
        +1);
    var nivel = Blockly.Arduino.valueToCode(this, 'nivel', Blockly.Arduino.
        ORDER_UNARY_POSTFIX);

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

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

Código A.3: Ejemplo asignación de código a bloque

El Código A.4 muestra como se define la parte gráfica de un bloque.

```
Blockly.Blocks['grove_led_bar'] = {
  init: function() {
    this.appendValueInput("nivel")
        .appendField("Barra de leds")
        .appendField(new Blockly.FieldImage("https://raw.githubusercontent.
            com/SeeedDocument/Grove-LED_Bar/master/img/Grove-LED_Bar-1.jpg",
             65, 50, "*"))
        .appendField("PIN#")
```

```
        .appendField(new Blockly.FieldDropdown(profile.default.digital), "
            PIN")
        .appendField("con el valor ");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
 this.setColour(260);
 this.setTooltip("");
 this.setHelpUrl("http://wiki.seeedstudio.com/Grove-LED_Bar/");
  }
};
```

Código A.4: Ejemplo definición aspeto visual de un bloque

Para poder mandar el código generado por *Blockly* es necesario cargar las librerías que se van a usar y almacenar el código tal como se muestra en Código A.5.

```

async flashCode(code : string, device: string){
    let buf = Buffer.from(code);
    let firmwareBlob2 = new Blob([buf], { type: 'text/plain' });
    let files = {};

    //cargando librerias
    for (var f in this.appFilesData) {
      files[f] = new Blob([this.appFilesData[f]], { type: "text/plain" });
    };

    files["main.ino"] = firmwareBlob2;
    console.log(files["main.ino"])
     return await new Promise(async (resolve, reject) => {
      let opts = {
        deviceId: device,
        files: files,
        auth: this.token
      };

      return await this.particle.flashDevice(opts)
        .then((result) => {
          if (!result.body.ok){
            resolve("Placa apagada")
          }
          else{
              if (result.body.message=="timed out waiting for device to
                  start"){
                resolve("Error al actualizar")
              }
              else{
                (async()=>{ await this.delay(10000).then(data =>{
                  resolve("Placa actualizada")
                }
                )});
              }
          }
        }).catch((err) => {
```

```
38            resolve("Error al actualizar")
39          });
40        })
41    }
```

Código A.5: Ejemplo actualización de placa

En la aplicación se han protegido las vistas para que únicamente puedan entrar los usuarios autenticados, en el Código A.6 se puede ver cómo se comprueba si un usuario está autenticado.

```
1  import { Injectable } from '@angular/core';
2  import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree,
       Router } from '@angular/router';
3  import { Observable } from 'rxjs';
4  import {AuthenticationService} from '../controller/authentication.service'
5  import { map } from 'rxjs/operators';
6
7  @Injectable({
8    providedIn: 'root'
9  })
10 export class AuthGuard implements CanActivate {
11   constructor(private auth: AuthenticationService, private router: Router)
       {}
12   canActivate(
13     next: ActivatedRouteSnapshot,
14     state: RouterStateSnapshot): Observable<boolean> {
15       return this.auth.userData.pipe(this.checkLogin());
16   }
17
18   private checkLogin() {
19     return map(loggedUser => {
20       if (loggedUser) { return true; }
21       this.router.navigate(['/login']);
22       return false;
23     });
24   }
25
26 }
```

Código A.6: Ejemplo guard

En el Código A.7 tenemos la parte lógica de un componente de *Angular* donde también podemos ver como se comunica con otros servicios para saber si el usuario es un usuario administrador. El código A.8 es la parte gráfica del componente anterior Código A.7.

```
1  import { Component,OnInit } from '@angular/core';
2  import {AuthenticationModelService} from './model/authentication-model.
     service';
3  import {AuthenticationService} from './controller/authentication.service';
4  import {DatabaseModelService} from './model/database-model.service';
5
6  @Component({
7    selector: 'app-root',
```

```
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  constructor(public authenticationModelService: AuthenticationModelService
      , public authenticationService: AuthenticationService, public db :
      DatabaseModelService) {}
  admin


  ngOnInit() {
    this.adminMenu();
  }


  adminMenu(){
    this.db.isAdmin().then(data =>{
      this.admin=data
    } )
  }

}
```

Código A.7: Ejemplo componente controlador

```

<mat-toolbar color="primary" class="app-header">
    <div class="logo"><img src='assets/sucreCodeBasic.svg' height="150"
       routerLink="/dashboard" ></div>
    <span class="nav-tool-items" (click)="adminMenu()">
      <button mat-icon-button [matMenuTriggerFor]="menu" aria-label="
         Example icon-button with a menu">
        <mat-icon>more_vert</mat-icon>
      </button>
      <mat-menu #menu="matMenu">
        <button mat-menu-item *ngIf="authenticationService.userData | async
            ">
          <span  style="font-weight: bolder;" >{{ (authenticationService.
             userData | async)?.email }}</span>
        </button>
        <button mat-menu-item *ngIf="!(authenticationService.userData |
           async)">
          <a  class="black" mat-button routerLink="login" routerLinkActive=
             "active">Iniciar sesi n</a>
        </button>
        <button mat-menu-item *ngIf="admin" routerLink="admin"
           routerLinkActive="active">
          <span  class="black" mat-button routerLink="admin"
             routerLinkActive="active">Gesti n</span>
        </button>
        <button mat-menu-item *ngIf="authenticationService.userData | async
            ">
          <span  class="black" mat-button routerLink="login"
             routerLinkActive="active" (click)="authenticationModelService.
```

```
              salir()">Cerrar sesi n</span>
        </button>
      </mat-menu>
    </span>

  </mat-toolbar>

  <router-outlet></router-outlet>
```

Código A.8: Ejemplo componente HTML

Con el Código A.9 se asocian las rutas de la aplicación a los componentes que se han de usar y las comprobaciones que se harán antes de mostrar la vista.

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import {AuhenticationComponent} from './view/auhentication/auhentication.
    component';
import {DashboardComponent} from './view/dashboard/dashboard.component';
import {BlockComponent} from './view/block/block.component'
import { Routes, RouterModule } from '@angular/router';
import {AuthGuard} from './guards/auth.guard';
import {BlocklyComponent} from './view/blockly/blockly.component'
import {AdminComponent} from './view/admin/admin.component'
import {AdminGuard} from './guards/admin.guard';


export const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: '/dashboard' },
  { path: 'login', component: AuhenticationComponent },
  { path: 'dashboard', component: DashboardComponent, canActivate: [
      AuthGuard]},
  { path: 'blockly', component: BlockComponent, canActivate: [AuthGuard]},
  { path: 'project/:ref', component: BlocklyComponent, canActivate: [
      AuthGuard]},
  { path: 'admin', component: AdminComponent, canActivate: [AuthGuard,
      AdminGuard ]}
];

@NgModule({
  declarations: [],
  imports: [
    CommonModule, RouterModule.forRoot(routes)
  ],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Código A.9: Ejemplo router

Angular tiene un archivo creado por defecto para poner variables de entorno, en este caso, lo hemos usado para que almacene los *tokens* de *Firebase* y podamos leerlos cuando sean necesarios. Se puede ver en el Código A.10.

```
export const environment = {
  production: true,
  firebaseConfig : {
    apiKey: "AIzaSXXXXXXXXXXXhxoiBY",
    authDomain: "XXXXXXXXXXX",
    databaseURL: "https://XXXXXXXX.firebaseio.com",
    projectId: "XXXXXXXX",
    storageBucket: "XXXXXXXX.appspot.com",
    messagingSenderId: "5XXXXXXX9",
    appId: "1:5088XXXXX129:XXXXX:5de073XXXXXXXX38e780",
    measurementId: "G-FSXXXXXXXXX"
  }
};
```

Código A.10: Tokens Base de datos

En el archivo de configuración *angular.json*, en el apartado de *script*, tenemos que definir la ruta de los ficheros que se vayan a usar tal y como se ve en el Código A.11.

```
"assets": [
              "src/favicon.ico",
              "src/assets"
            ],
            "styles": [
              "./node_modules/@angular/material/prebuilt-themes/indigo-pink
                 .css",
              "src/styles.css"
            ],
            "scripts": [
              "node_modules/ngx-blockly/scripts/blockly/blockly_compressed.
                 js",
              "node_modules/ngx-blockly/scripts/blockly/blocks_compressed.
                 js",
              "node_modules/ngx-blockly/scripts/blockly/arduino_compressed.
                 js",
              "node_modules/ngx-blockly/scripts/blockly/msg/js/es.js",
              "node_modules/particle-api-js/dist/particle.min.js"
            ]
          },
```

Código A.11: Configuración angular.json

Apéndice B

Anexo II: Configuración de Firebase

En este anexo se va a explicar más en detalle cómo se crea y se configura una base de datos *Firebase* y cómo se ha configurado para este proyecto.

El primer paso es crear la base de datos, para ello resulta indispensable registrarse en *Firebase* , una vez registrado hay que añadir un proyecto en el panel de tu usuario *Firebase*. En la Figura B.1 podemos ver las distintas bases de datos asociadas a una cuenta y la opción de crear proyecto.

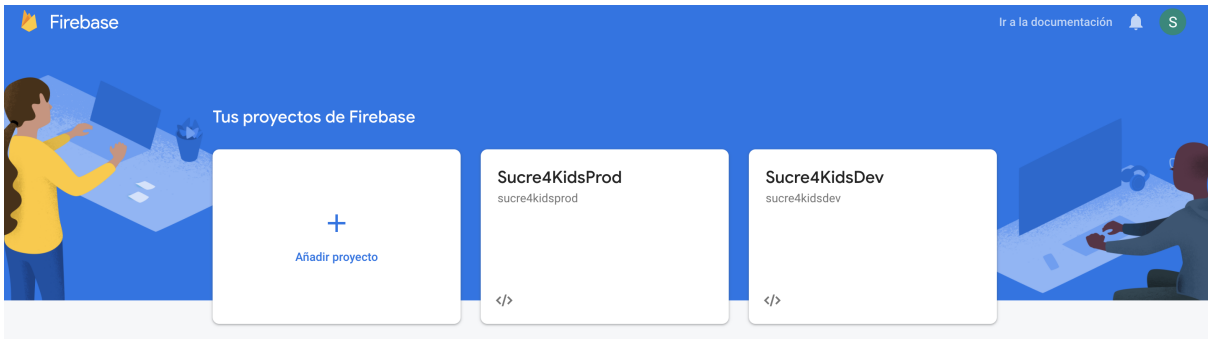

Figura B.1: Proyectos de Firebase. Fuente: Firebase

Firebase ofrece un servicio de autenticación de usuarios, este servicio nos ofrece diferentes métodos de autenticación de los usuarios, como se muestra en la Figura B.2 aunque permite autentificarse con servidores externos, se utiliza el correo electrónico y una contraseña que la almacena *Firebase*.

Para los roles de los usuarios, se ha creado una colección *admins* como se ven la Figura B.3, la cual tiene un vector con los identificadores de los usuarios administradores, de esta manera podemos consultar el rol del usuario rápidamente con una consulta simple y rápida.

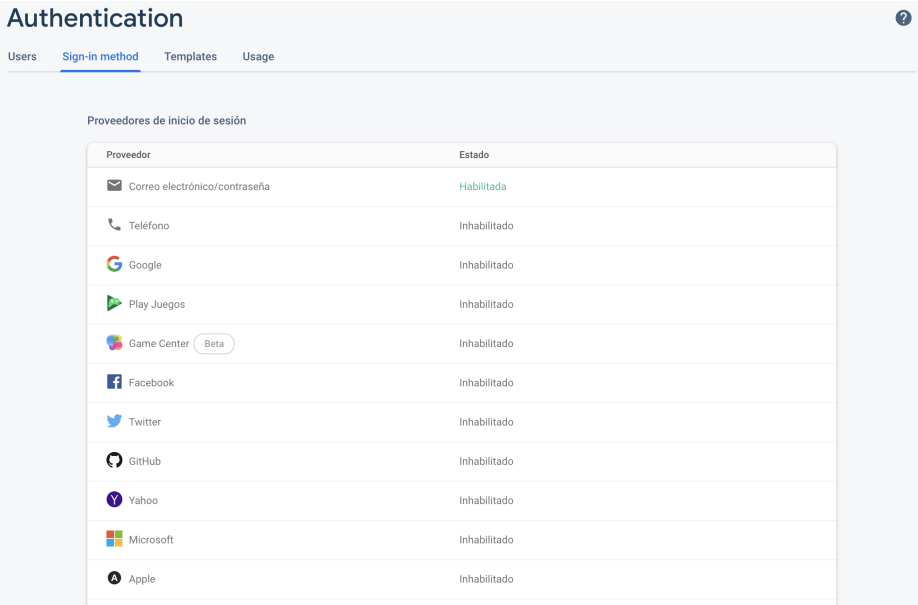

Figura B.2: Métodos de autenticación de usuario de Firebase. Fuente: Firebase

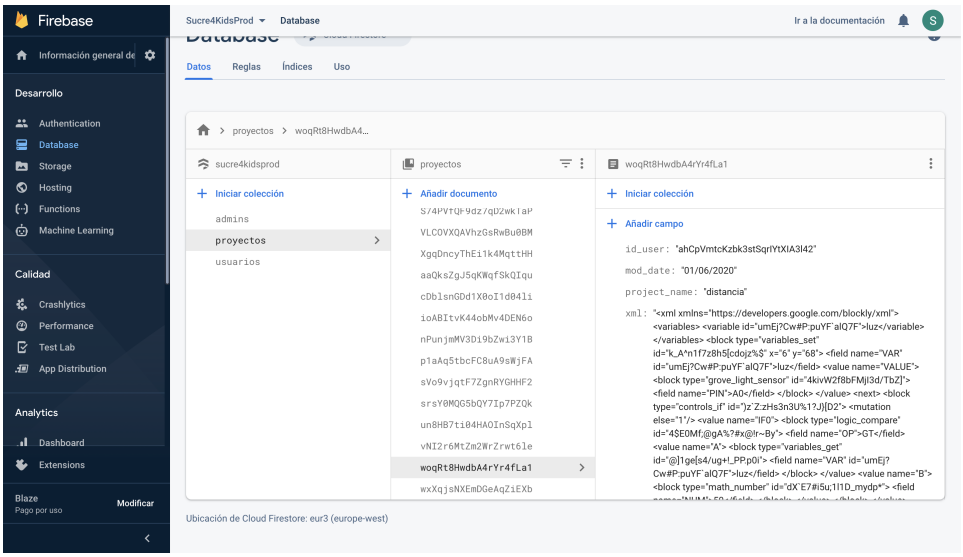

Figura B.3: Estructura de datos de la base de datos. Fuente: Firebase

También podemos dar de alta a los distintos usuarios desde la propia página web de *Firebase* lo que permite añadir usuarios de una forma muy cómoda y sencilla evitando tener que implementar un registro de usuario en *SucreCode*, en la Figura B.4 podemos ver como dar de alta a un usuario y una parte del conjunto de usuarios ya creados que tienen acceso a la aplicación.

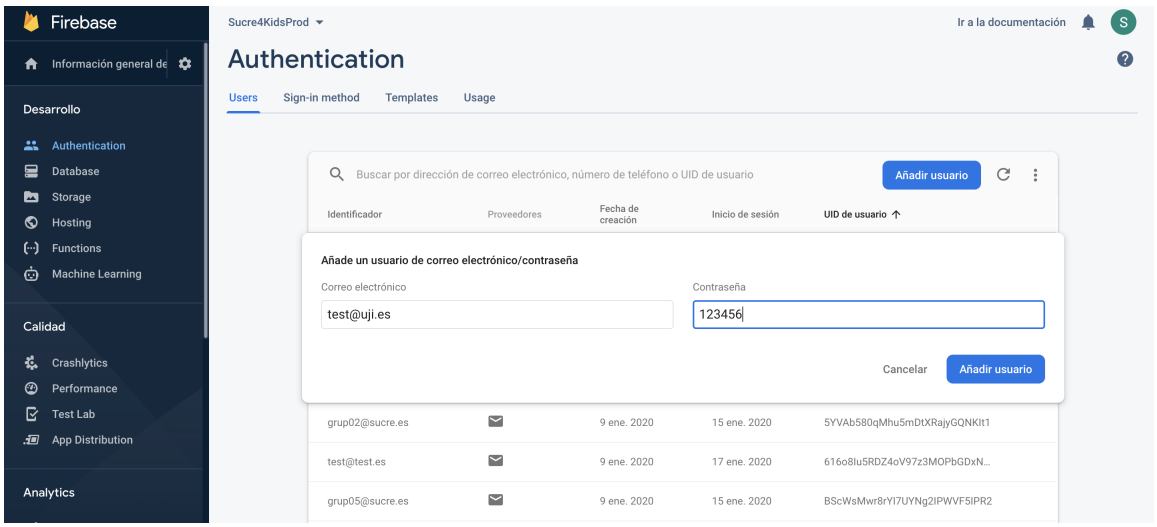

Figura B.4: Registro de usuarios desde Firebase. Fuente: Firebase

Necesitamos obtener los *tokens* necesarios para poder hacer que *SucreCode* pueda acceder a la base de datos que hemos creado. *Firebase* ofrece un asistente muy sencillo para generar estas claves de forma automática. Como se muestra en la Figura B.5 en la configuración del proyecto, añadimos una nueva aplicación web y ya podremos poner los *tokens* en *SucreCode*

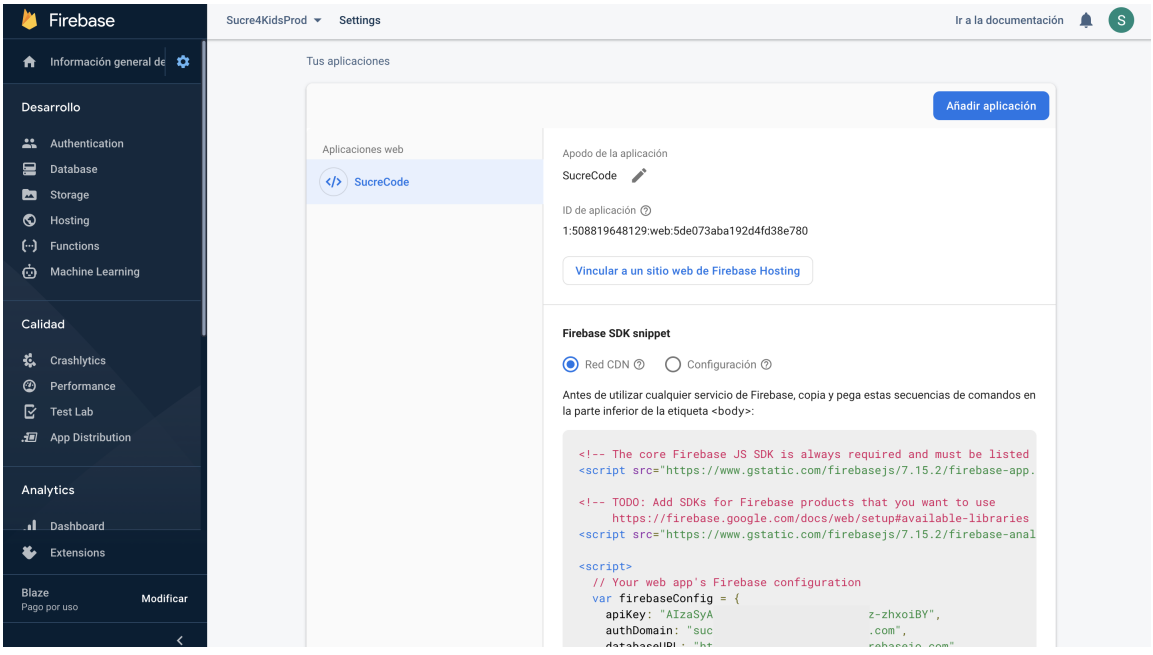

Figura B.5: Obtención de tokens de la base de datos. Fuente: Firebase

Para obtener los datos necesarios de la base de datos, *Firebase* ofrece un servicio llamado *Functions* para que las consultas a la base de datos se almacenan en el *proyecto Firebase* como se aprecia en la Figura B.6. De esta manera conseguimos que todos los usuarios ejecuten la misma consulta independiente del dispositivo además, de que el código de la aplicación web quede más limpio y depurado.

En la Figura B.6 podemos ver el nombre de la consulta, la *URL* a la que está asociada la consulta y otros datos respecto a la ubicación del servidor *Firebase* y los recursos asociados.

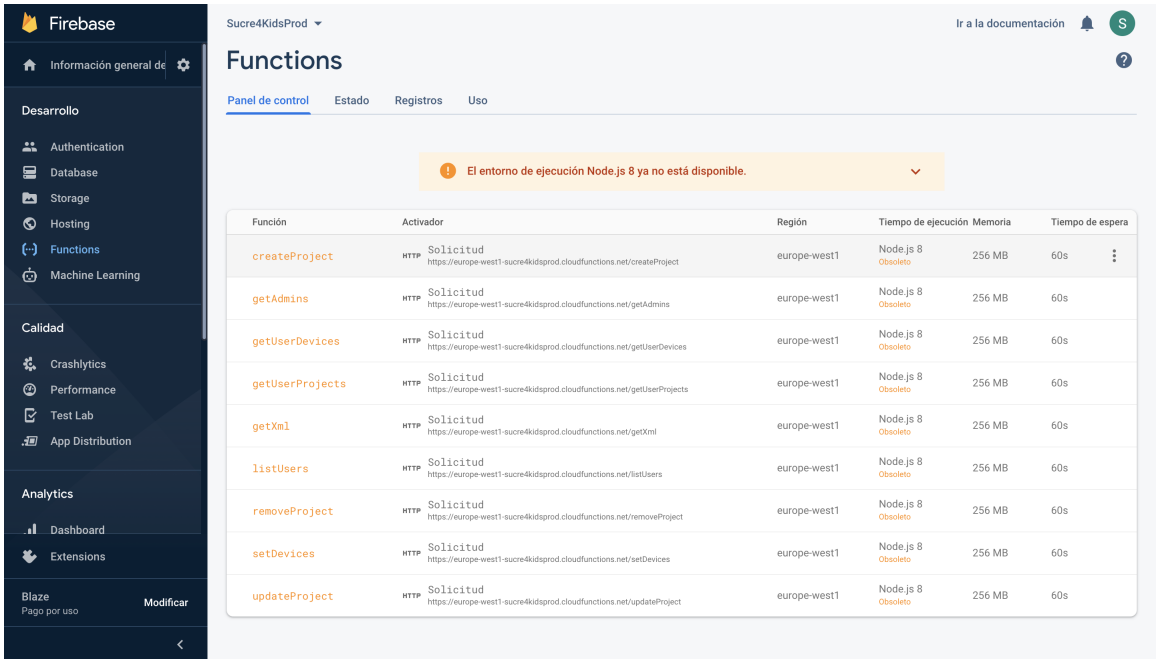

Figura B.6: Consultas creadas y guardadas en el proyecto de Firebase. Fuente: Firebase

Finalmente se han definido unas reglas de acceso a la base de datos para que no puedan acceder usuarios sin identificarse a la base de datos. Solo los usuarios autenticados pueden leer y escribir en la base de datos, con esta configuración evitamos que puedan acceder a los datos personas ajenas al proyecto.

Apéndice C

Anexo III: Configuración de Particle

En este anexo, se va a explivar como se asocian dispositivos *IoT* a una cuenta *Particle*.

Para el proyecto hay que configurar los dispositivos *IoT* para que pertenezcan a una cuenta de *Particle* y usando la *API* autenticarnos en *SucreCode* para poder gestionar y mandar el código generado por el usuario a los dispositivos.

Para añadir los dispositivos, podemos hacerlo de dos formas. La primera es desde la aplicación móvil escaneando el código de la placa. La segunda es desde la aplicación web como se muestra en la Figura C.1 donde tenemos que poner él id del dispositivo seguir los pasos del asistente.

Desde la cuenta podemos ver todos los dispositivos asociados a la cuenta y la información de cada uno como se muestra en la Figura C.2

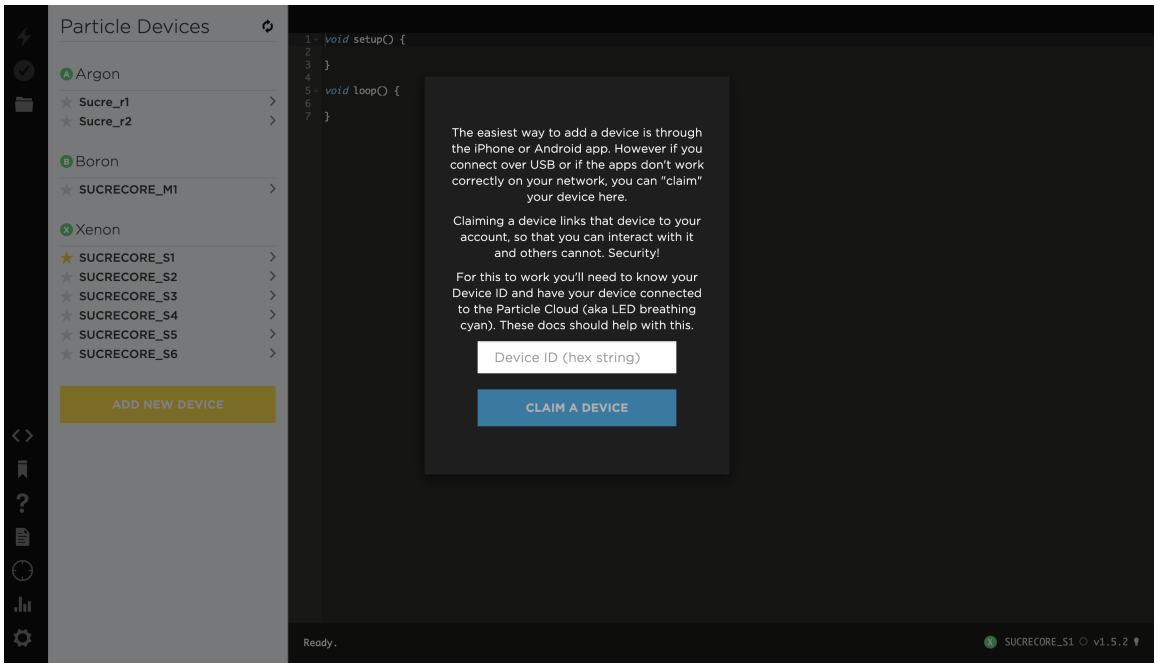

Figura C.1: Asociar dispositivo IoT a la cuenta Particle. Fuente: Particle

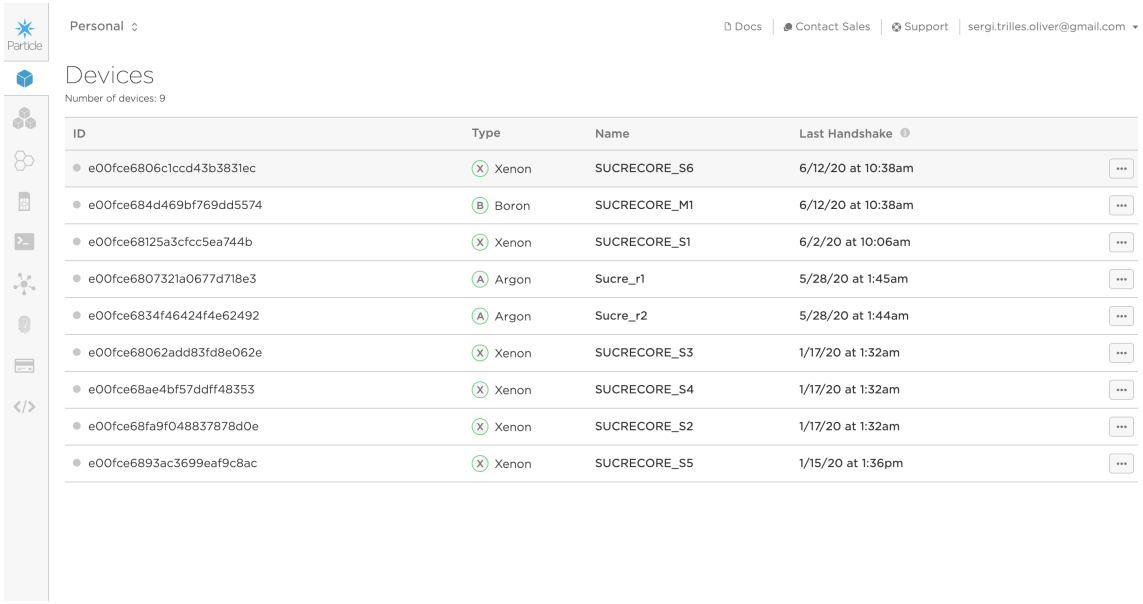

Figura C.2: Dispositivos asociados a la cuenta Particle. Fuente: Particle

Bibliografía

[1] Angular. Angular como framework. https://angular.io/. [Consulta: 24 de Marzo de 2020].

[2] Angular. Documentacion angular router. https://angular.io/guide/router. [Consulta: 24 de Mayo de 2020].

[3] Angular. Lazy-loading feature modules. https://angular.io/guide/lazy-loading-ngmodules. [Consulta: 24 de Mayo de 2020].

[4] Auth0. Tokens como funcionan. https://auth0.com/learn/token-based-authentication-made-easy/. [Consulta: 24 de Mayo de 2020].

[5] Google. Blockly para desarrolladores. https://developers.google.com/blockly. [Consulta: 25 de Marzo de 2020].

[6] Google. Documentación firebase cli. https://firebase.google.com/docs/cli. [Consulta: 24 de Mayo de 2020].

[7] Google. Https. https://support.google.com/webmasters/answer/6073543?hl=es. [Consulta: 24 de Mayo de 2020].

[8] Google. Qué es firebase y como funciona. https://firebase.google.com/?hl=es. [Consulta: 25 de Marzo de 2020].

[9] Google. Uso y configuración de angular material. https://material.angular.io/guide/getting-started. [Consulta: 24 de Mayo de 2020].

[10] indeed. Sueldo de un analista. https://www.indeed.es/salaries/analista-programador-Salaries. [Consulta: 2 de Abril de 2020].

[11] indeed. Sueldo de un programador. https://www.indeed.es/salaries/programador-web-Salaries. [Consulta: 2 de Abril de 2020].

[12] MDN. Media query que es y como usarlo. https://developer.mozilla.org/es/docs/CSS/Media_queriesp. [Consulta: 24 de Mayo de 2020].

[13] Mozilla. Tutorial de javascript. https://developer.mozilla.org/es/docs/Web/JavaScript. [Consulta: 25 de Marzo de 2020].

[14] Nacho Blanco. ¿qué patrón usa angular? mvc o mvvm. https://openwebinars.net/blog/que-patron-usa-angular-mvc-o-mvvm/. [Consulta: 2 de Abril de 2020].

[15] openwebinars. ¿qué es un servicio rest? https://openwebinars.net/blog/que-es-rest-conoce-su-potencia/. [Consulta: 24 de Mayo de 2020].

[16] Particle. Particle dispositivos iot. https://www.particle.io/. [Consulta: 24 de Marzo de 2020].

[17] RxJS. Rxjs como funciona. https://rxjs-dev.firebaseapp.com/. [Consulta: 24 de Mayo de 2020].

[18] TypeScript. ¿qué es typescript? https://www.typescriptlang.org/. [Consulta: 24 de Marzo de 2020].

[19] Wikipedia. Estandar oauth y la autenticación. https://es.wikipedia.org/wiki/OAuth. [Consulta: 24 de Mayo de 2020].

[20] Wikipedia. Json estructura y ejemplos. https://es.wikipedia.org/wiki/JSON. [Consulta: 24 de Mayo de 2020].

[21] Wikipedia. Que es una api. https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones. [Consulta: 24 de Mayo de 2020].