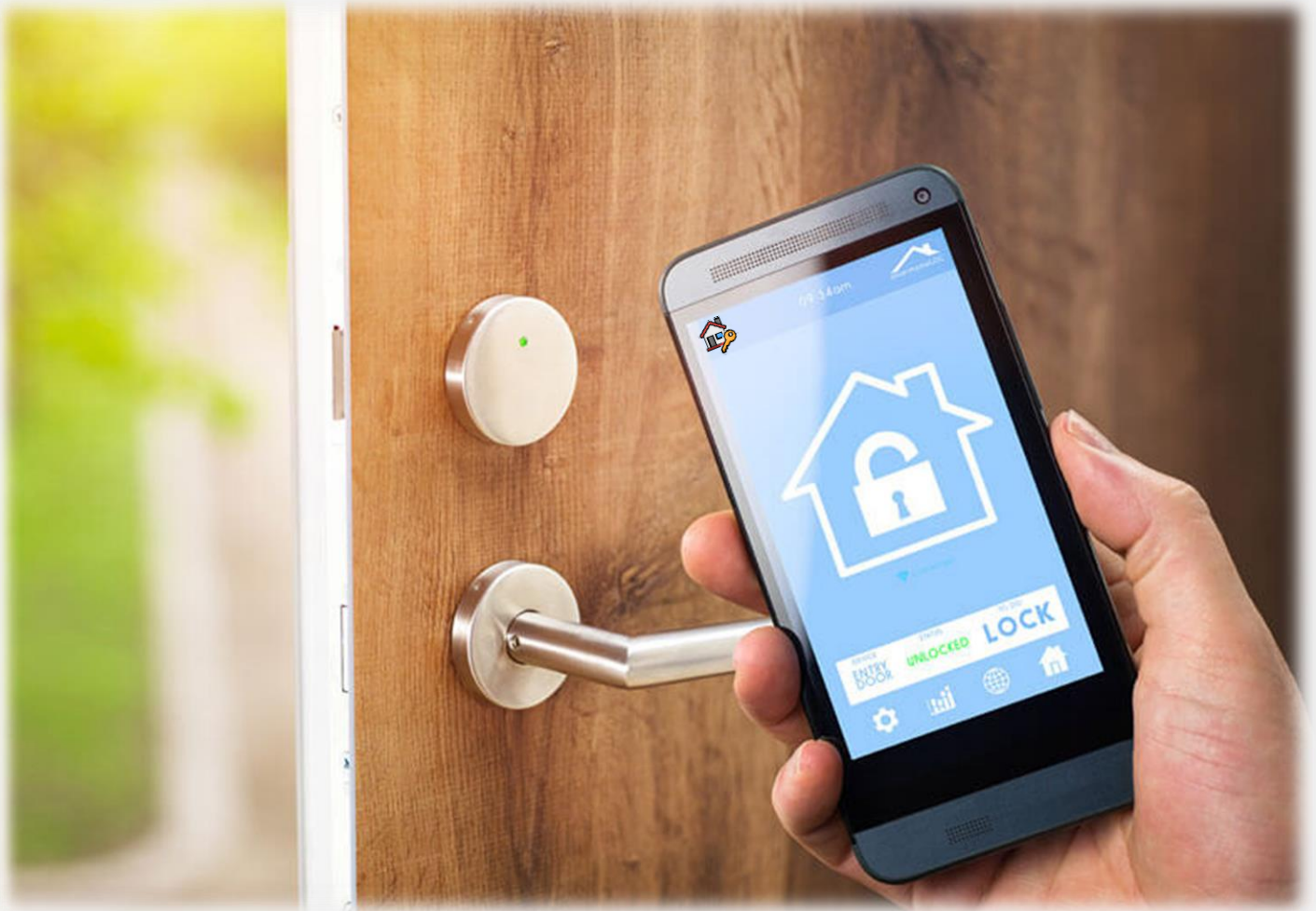


# ESP-Lock Opener



Aythami Déniz Morales  
Jesús Medina Naranjo  
3° Desarrollo de Aplicaciones Multiplataforma

## Índice:

<b>Descripción del proyecto .....</b>	<b>1</b>
<b>Tecnologías usadas en el desarrollo de la aplicación.....</b>	<b>2</b>
<b>Esquema/Modelo de la Base de Datos .....</b>	<b>6</b>
<b>Roles en la Aplicación.....</b>	<b>7</b>
<b>Desarrollo de la aplicación.....</b>	<b>8</b>
<b>Integración y Costes Empresariales .....</b>	<b>21</b>
<b>Conclusiones .....</b>	<b>22</b>
<b>Bibliografía .....</b>	<b>23</b>
<b>Anexos.....</b>	<b>24</b>
Anexo 1: Visual Studio Code.....	24
Anexo 2: Productos Google Firebase .....	24
Anexo 3: Estructura de la Base de Datos.....	26
Anexo 4: Diagrama Casos de Uso .....	27
Anexo 5: Desarrollo de la Aplicación.....	28
5.1 Componente Menú .....	28
5.2 Servicio alert.service.ts .....	29
5.3 Servicio img.service.ts .....	29
5.4 Servicio firestore.service.ts .....	30
5.5 Componente header-tabs.....	30
5.6 Componente maps .....	31
5.7 Botones panel configuración.....	31
5.8 Ion-item-sliding .....	32
5.9 Registro de usuarios.....	33
5.10 Código controladora ESP32 .....	33

## Descripción del proyecto

En el siguiente documento se explicará cómo hemos diseñado y desarrollado la aplicación **ESP-Lock Opener** y cuáles son sus funcionalidades, así como las diferentes tecnologías que hemos usado en el proyecto.

Nuestra aplicación ha sido diseñada en base a la idea de poder gestionar, por un lado, las reservas de viviendas vacacionales y apartamentos, y por otro lado, poder gestionar las cerraduras electrónicas de dichos alojamientos, todo ello a través de una app.

Por otro lado, también implementaremos nuestro propio sistema de cerraduras electrónicas, las cuales serán instaladas en los alojamientos que quieran implementar nuestro sistema. Éste sistema constará de la cerradura electrónica propiamente dicha y de la controladora que se encargará de abrirla cuando detecte el código de reserva asociado a la misma, el cual será enviado desde el móvil del usuario a través de la app.

La app constará de una página principal en la que podremos ver un listado de tarjetas por cada alojamiento disponible, así como imágenes de dicho alojamiento. Cada tarjeta tendrá un botón en forma de calendario en el que al pulsar en él accederemos a una página donde podremos seleccionar en un calendario la fecha de llegada y salida del alojamiento, pudiendo así generar la reserva y el código que quedará asociado a la cerradura. También contará con un botón con forma de mapa en el que, al clicar en él, nos abrirá una página con un mapa mostrándonos la geolocalización del alojamiento. Podremos interactuar con él para hacer zoom movernos por el mapa etc.

Una vez generada la reserva, la controladora de la cerradura recibirá el código de reserva que le pertenece y lo almacenará durante la estancia del usuario. Una vez finaliza la el periodo de estancia el código se elimina, quedando el alojamiento disponible para otros usuarios.

## Tecnologías usadas en el desarrollo de la aplicación

A continuación se detallarán las tecnologías y herramientas que hemos utilizado en el desarrollo de la aplicación (lenguajes de programación, **frameworks**<sup>1</sup>, IDE, etc.)

➤ La aplicación móvil la hemos desarrollado utilizando, por un lado, **Angular**, un framework desarrollado en TypeScript y de código abierto que es mantenido por Google. Por otro lado hemos usado el **SDK (Kit de Desarrollo de Software)**<sup>2</sup> de **Ionic**, que no es más que otro framework de código abierto que usaremos sobre Angular y nos permitirá desarrollar la aplicación tanto para Android como IOS. Además también nos ofrece un amplio catálogo de **plugins** que nos permiten acceder a las diferentes características de que dispone un smartphone (cámara, almacenamiento, GPS, bluetooth...) Hemos elegido dichas tecnologías ya que han sido con las que más hemos trabajado y por las funcionalidades que nos ofrecen.



➤ Para la parte del backend hemos usado **Google Firebase**. Se trata de una plataforma de desarrollo y despliegue de apps móviles y web, en donde tenemos diversas utilidades y herramientas como: Servicios de autenticación, hosting, bases de datos NoSQL, estadísticas de uso... Más adelante lo explicaremos un poco más en detalle. En nuestro caso hacemos uso de **Cloud Firestore** como base de datos, la cual no se trata de una BBDD relacional sino documental. Éste tipo de BBDD se caracterizan por no tener tablas como las conocemos normalmente. Para entenderlo relacionaremos cada parte de nuestra BBDD documental con su respectiva equivalencia en una BBDD relacional. Cloud Firestore cuenta, en lugar de tablas, con **colecciones** donde almacenamos **documentos**, que serían los equivalentes a las filas en las BBDD relaciones, con su respectivo ID único. A su vez, cada documento cuenta con una serie de **campos**, que serían los equivalentes a las columnas en una BBDD relacional. Este tipo de base de datos no usan lenguaje SQL, emplean otros lenguajes para el almacenaje como JSON o XML.



---

<sup>1</sup> Entorno de trabajo predispuesto que posee herramientas para agilizar el desarrollo de proyectos de programación

<sup>2</sup> Conjunto de herramientas proporcionado por el fabricante de una plataforma hardware, sistema operativo o lenguaje de programación.

➤ Para el control de versiones del proyecto utilizamos Git, que es un sistema de control de versiones desarrollado por Linus Torvalds (creador del kernel del SO linux). Éste se encuentra optimizado para guardar cambios de forma incremental, cuenta con un historial que permite volver a versiones anteriores de un proyecto y cuenta con un registro de cambios que otras personas hacen sobre el proyecto. Además contamos con un [repositorio](#) en GitHub, que no es más que un portal web creado para alojar el código de las aplicaciones de cualquier desarrollador. En él almacenaremos nuestro proyecto y todos tendremos acceso al proyecto de forma remota pudiendo descargar y visualizar el código fuente del mismo, así como participar sobre el proyecto siempre y cuando el propietario no lo tenga restringido



➤ Para poder probar la aplicación y realizar pruebas de funcionamiento y rendimiento utilizaremos Android Studio, un **IDE (Entorno de Desarrollo Integrado)**<sup>3</sup> creado para la plataforma Android, para cargar la aplicación en un dispositivo real. También podremos realizar pruebas en un dispositivo emulado. De esta forma podremos ir asegurándonos que la app funcionará correctamente en dispositivos móviles, pudiendo así detectar errores que corregir.

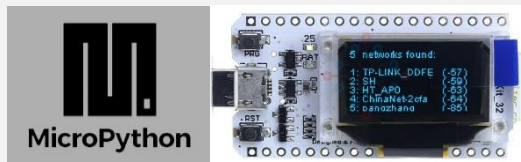


➤ La controladora que usaremos para la cerradura electrónica es un módulo ESP32. Se trata de una placa microcontroladora que incluye Wi-Fi y Bluetooth (BLE) todo en uno, integrada y certificada que proporciona no solo la radio inalámbrica, sino también un procesador de dos núcleos que se pueden ajustar entre 80 y 240 MHz, integrado con interfaces para conectarse con varios periféricos (interfaz serie, Ethernet, tarjeta SD, interfaces táctiles y capacitivas...) Para simular las cerraduras en las pruebas, contamos con una placa de varios

---

<sup>3</sup> Aplicación que proporciona servicios integrales para facilitarle al programador el desarrollo de software

relés conectada al ESP32, que actuarán cuando demos la orden de abrir. Para codificar el microcontrolador el lenguaje de programación usado ha sido MicroPython por su sencillez y posibilidades en el ámbito de la electrónica. Más adelante explicaremos el código que se encarga de manejar la cerradura electrónica.



➤ El editor de código que utilizamos es **Visual Studio Code**<sup>4</sup>, un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. Además cuenta con un amplio catálogo de extensiones de todo tipo que ayudan al desarrollo de aplicaciones en casi cualquier lenguaje de programación. Nosotros le hemos instalado los siguientes extensiones que nos ayudaran en el desarrollo de nuestra aplicación y la codificación de la misma:

1. **AB HTML Formatter:** esta extensión nos ayudará a dar formato a la parte de código HTML que tiene nuestra aplicación, de esta forma tendremos un código más limpio y ordenado.
2. **Activitus Bar:** se trata de una extensión que oculta la barra de actividades de Visual Studio Code, por lo que ganamos más espacio en pantalla para trabajar. Es muy útil para quienes trabajan en portátiles.
3. **Angular Snippets:** Lo primero es saber que es un snippet; pues no es más que un pequeño fragmento de texto que escribimos el cual nos muestra parte del contenido que está asociado al texto. Con ésta extensión podremos codificar más rápido ya que nos permitirá encontrar de forma más eficiente funciones, clases, importaciones etc. Además cuenta con atajos que nos permitirá generar pequeñas estructuras de código ya predefinidas
4. **Auto Close Tag:** Esta extensión nos ayuda a la hora de codificar HTML o XML, cerrando automáticamente por nosotros las etiquetas de cada parte del código
5. **Auto Rename Tag:** Relacionada con la anterior, si quisiéramos cambiar un `<div></div>` por un `<content></content>` por ejemplo, tendríamos que buscar y editar primero la etiqueta de apertura y luego la etiqueta de cierre. Con ésta extensión basta con editar una de ellas ya que la otra se modificará simultáneamente.

---

<sup>4</sup> [Ver anexo 1](#)



**6. HTML CSS Support:** Ésta es una extensión que nos resumirá bastante la escritura, a pesar de que Visual Studio Code ya cuente con un “completador de sintaxis”. Html CSS Support mejora la experiencia a la hora de “picar código” de HTML o CSS aún más, además de que soporta una lista de lenguajes bastante extensa.

**7. HTML Snippets:** Al igual que el Angular Snippets, nos ayudará a codificar más rápido permitiéndonos obtener un listado de opciones de código HTML mientras escribimos, pudiendo seleccionar el que deseamos.

**8. Ionic Snippets:** Otro snippet que nos ayudará a codificar en ionic y hacer uso de sus diversos componentes. Al igual que Angular Snippets, nos ofrece atajos para generar estructuras de código ya predefinidas.

**9. TypeScript Importer:** Como ya sabemos Angular está desarrollado en Typescript y éste cuenta con diversas librerías de utilidades y funciones. Pues ésta extensión busca automáticamente definiciones de TypeScript en los archivos de nuestro espacio de trabajo y proporciona todos los símbolos conocidos como elementos de finalización para permitir la finalización del código y las importaciones de clases.

**10. Firebase Explorer:** Con ésta extensión podremos gestionar nuestros proyectos de Firebase. Ya que nuestro proyecto usa dicha tecnología nos será muy útil para poder trabajar con nuestra base de datos sin necesidad de verlo en el navegador.



➤ Para realizar el modelo relacional y los diagramas de casos de uso de nuestra aplicación hemos usado una web llamada [diagramas.net](https://diagramas.net), que podemos encontrar en Google por draw.io. En éste sitio web podemos crear prácticamente cualquier tipo de diagrama que necesitemos, así como exportarlo a nuestro equipo, a One Drive, Google Drive, Dropbox, GitHub y GitLab.

## Esquema/Modelo de la Base de Datos

Como ya hemos comentado, para la base de datos hemos decidido utilizar Firestore, una BBDD documental que nos ofrece [Google Firebase](#). Éste servicio en la nube cuenta con más **productos y funciones**<sup>5</sup> que ayudan a los desarrolladores en la creación y despliegue de aplicaciones móviles y web. Por lo que la principal característica de Firebase es facilitar a los desarrolladores un backend robusto y actualizado que les permita centrarse en el frontend.

Pasemos a explicar cómo hemos estructurado nuestra **base de datos**<sup>6</sup>; recordemos que se trata de una BBDD documental, es decir, formada por colecciones de documentos. Nuestra base de datos cuenta con cuatro colecciones de documentos: alquileres, apartamentos, cerraduras y usuarios. Explicaremos cada una de forma detallada.

**Colección de alquileres:** contendrá los documentos que almacenarán los datos de la reserva, estos datos o campos que contiene el documento son: fecha de inicio (se almacenará las fechas separadas por comas), el **ID**<sup>7</sup> de la propiedad y el ID del usuario. Recordemos que cada documento que se encuentre en una colección tendrá un ID único y el cual podremos usar para generar referencias a dicho documento desde cualquier colección.

**Colección de apartamentos:** contendrá los documentos que almacenarán los datos de cada alojamiento. Los campos que contendrá dicho documento son: descripción del alojamiento, la dirección, el estado (ocupado o libre), el ID de la cerradura, una imagen almacenada como string, y las coordenadas para ubicarlo en el mapa (latitud y longitud).

**Colección de cerraduras:** contendrá los documentos que almacenarán los datos de las cerraduras registradas. Los campos que contendrá los documentos de esta colección serán: activa, que nos indicará si la cerradura está asignada o no a un alojamiento; y código, que es la clave que se usará para comparar con la aplicación para verificar que puede abrirse la cerradura.

**Colección de usuarios:** contendrá los documentos que almacenarán los datos de los usuarios que se registren en la aplicación. Éstos documentos contendrán tres tipos de campos: el email de registro, el rol del usuario (administrador o usuario) y el ID del usuario, que se genera al registrarse el usuario. Más adelante veremos cómo lo hemos hecho con Google Authentication.

---

<sup>5</sup> [Ver anexo 2](#)

<sup>6</sup> [Ver anexo 3](#)

<sup>7</sup> Identificador único



## Roles en la Aplicación

En nuestra aplicación contamos con dos roles principales: El administrador y el usuario. Vamos a explicar las funciones de cada uno y además mostraremos unos **diagramas de casos de uso para dichos roles**<sup>8</sup>.

Por un lado disponemos del **rol administrador**, el cual podrá gestionar todo lo relacionado con los alojamientos y sus cerraduras, es decir, podrá dar de alta nuevos alojamientos en la app y asignarles una cerradura, también podrá tener un listado donde consultar los alojamientos que están registrados y ver si están ocupados o no además podrá editar los datos de dichos alojamientos. Finalmente podrá eliminar alojamientos y desactivar la cerradura que tenga asignada, ya que dicha cerradura puede ser instalada nuevamente en otro alojamiento.

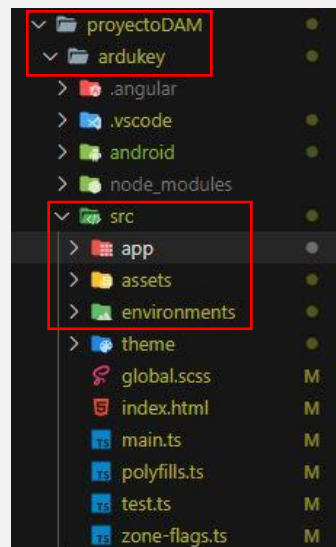
Por otro lado tenemos el **rol usuario** quien podrá registrarse en la aplicación mediante un formulario añadiendo correo y contraseña, una vez registrado podrá navegar en busca de alojamientos y reservar el que prefiera. Finalmente podrá abrir la cerradura del alojamiento reservado mediante la app, puesto que tendrá asignado un código de desbloqueo de cerradura.

---

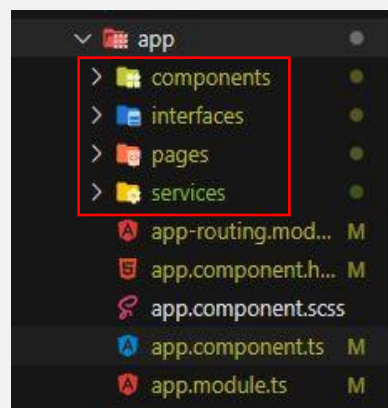
<sup>8</sup> [Ver anexo 4](#)

## Desarrollo de la aplicación

Nuestra aplicación ha sido desarrollada siguiendo un orden de directorios de manera que nos sea fácil trabajar y mantener el programa. En un proyecto de Angular el directorio donde principalmente trabajaremos es: ardukey/src, y dentro de éste podremos trabajar en /app, /assets y /environments. En la carpeta app ubicaremos los archivos y directorios principales de la aplicación, es decir, toda la lógica de ésta y el código HTML que conforma el diseño estará localizado dentro de la carpeta /app. La carpeta /assets contendrá los elementos estáticos de nuestra aplicación, es decir, imágenes, PDFs, documentos de Word, archivos mp3, archivos JSON, etc. Por último en la carpeta /environments encontraremos dos archivos en los que se especifican todas las variables, constantes, funciones, clases, módulos etc. que nos permite definir si estamos desplegando en entorno de producción o de desarrollo.



Una vez estamos dentro de la carpeta /app podemos observar la estructura de directorios y ficheros que hemos creado para nuestro proyecto. En primer lugar vemos el directorio /components que, en Angular, no es más que un módulo de componentes. Los módulos de Angular nos permiten agrupar un conjunto de componentes que no son más que estructuras de código utilizado en un ámbito concreto de la aplicación, o una funcionalidad específica. Por otro lado tenemos el directorio /interfaces; dentro de éste directorio encontraremos un fichero llamado interfaces.ts en el que almacenaremos, como su propio nombre indica, las interfaces<sup>9</sup> que usaremos en el programa. El siguiente directorio que tenemos es /pages, donde podremos encontrar las páginas con las que cuenta nuestra aplicación (página inicial, página de usuarios...) ordenados en una serie de sub directorios en función de su uso. Finalmente nos encontramos con el directorio /services donde encontraremos los servicios necesarios para nuestra aplicación. Dichos servicios serán los encargados de proveer los datos provenientes de nuestro backend, así como enviarle datos.



<sup>9</sup> Conjunto de métodos abstractos y de constantes cuya funcionalidad es la de determinar el funcionamiento de una clase, es decir, funciona como un molde o como una plantilla. Al ser sus métodos abstractos estos no tiene funcionalidad alguna, sólo se definen su tipo, argumento y tipo de retorno.

Pasemos a explicar con detalle cada parte de nuestro programa para así comprender su funcionamiento, empecemos por las interfaces que hemos creado. Partiendo de las colecciones que tenemos en nuestra BBDD, hemos creado cuatro interfaces para poder trabajar el envío y recepción de los datos, de manera que en nuestro programa los trataremos como objetos del tipo requerido en cada situación. Como podemos ver, cada interfaz debe ser exportada para poder ser utilizada en cualquier parte de nuestro programa, y cada una de ellas se llama igual y cuenta exactamente con las mismas propiedades que la respectiva colección a la que está asociada de la BBDD.

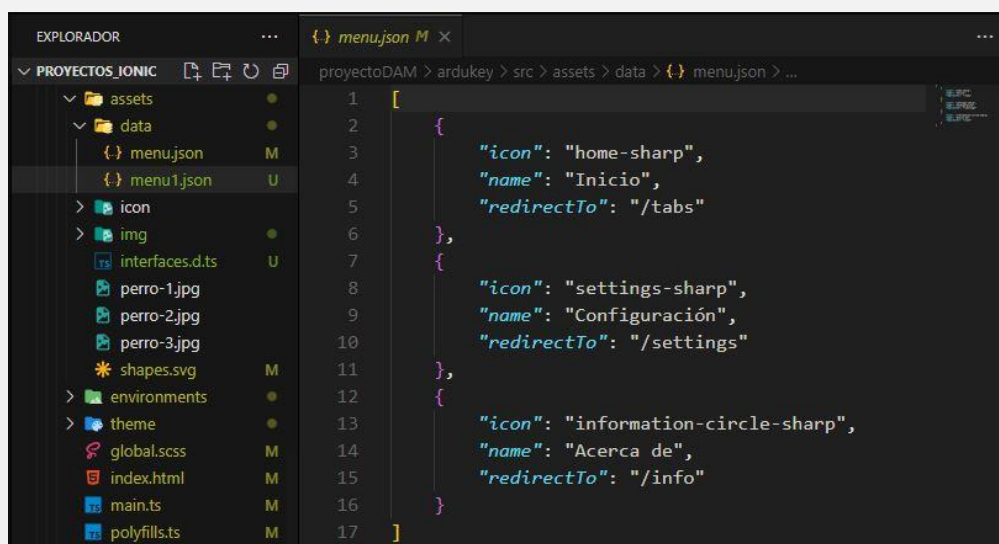


```

1 export interface menuOpts {
2   icon: string;
3   name: string;
4   redirectTo: string;
5 }
6
7 export interface Apartamento {
8   DESCRIPCION: string;
9   DIRECCION: string;
10  LAT: number;
11  LON: number;
12  IMG: string;
13  ESTADO: string;
14  IDKEY: string;
15 }
16
17 export interface Cerradura {
18  ACTIVA: boolean;
19  CODIGO: string;
20 }
21
22 export interface Alquiler {
23  IDPROP: string;
24  F_INICIO: Date;
25  UID: string;
26 }
27
28 export interface Usuario {
29  EMAIL: string;
30  ROL: string;
31  UID: string;
32 }

```

Por otro lado podemos ver una quita interfaz llamada **menuOpts**, con dicha interfaz manejaremos las opciones del menú que tendrá nuestro programa. Cada elemento del menú tiene unas propiedades como podemos observar: un icono, un nombre y una redirección. Estas opciones, al ser elementos estáticos, los hemos ubicado en un archivo JSON dentro de /assets/data. Como podemos observar en la imagen el archivo JSON cuenta con las mismas propiedades de los elementos del menú de opciones.

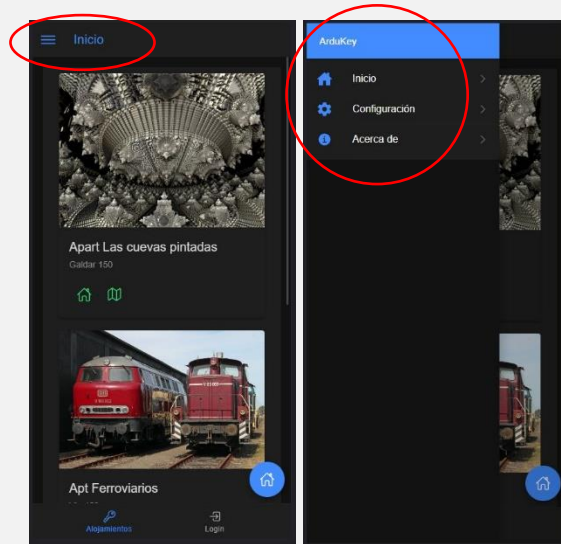


```

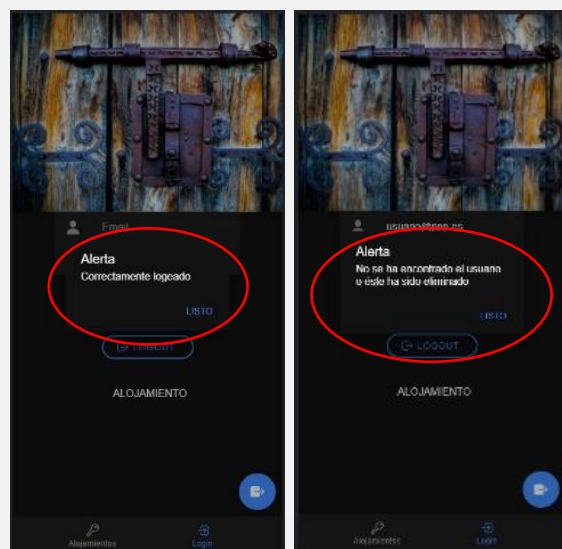
1 [
2   {
3     "icon": "home-sharp",
4     "name": "Inicio",
5     "redirectTo": "/tabs"
6   },
7   {
8     "icon": "settings-sharp",
9     "name": "Configuración",
10    "redirectTo": "/settings"
11  },
12   {
13     "icon": "information-circle-sharp",
14     "name": "Acerca de",
15     "redirectTo": "/info"
16   }
17 ]

```

Por otro lado hemos creado un servicio con una función que se encargara de cargar el archivo JSON con las propiedades que hemos predefinido del menú y enviarlo al **componente menú**<sup>10</sup> que será cargado al iniciar la aplicación. Se ha preparado cierta parte de código para poder implementar que se oculten partes del menú si el rol del usuario que inicie sesión no es de administrador. Dicho menú nos aparecerá en la aplicación en la parte superior izquierda en forma de toogle.



Continuaremos explicando los servicios que tenemos en la app para así poder comprender el funcionamiento de los componentes que hacen uso de ellos. Comencemos por el **alert.service.ts**<sup>11</sup> En éste servicio hemos implementado una función la cual recibe dos parámetros de tipo string, por un lado el estado y por otro el mensaje. Con éste manejaremos todos los mensajes de alerta que aparecerán en la aplicación como, por ejemplo, un registro de usuario correcto o un login fallido.



<sup>10</sup> [Ver anexo 5.1](#)

<sup>11</sup> [Ver anexo 5.2](#)

Otro servicio que hay que destacar es el **img.service.ts**<sup>12</sup>. Con este servicio nos encargamos de verificar que la imagen que vayamos a subir a la BBDD sea verificada. Para ello contamos con el DomSinitizer que no es más que un servicio que nos provee Angular para ayudar a prevenir ataques a las aplicaciones. Lo que hace, básicamente, es evitar que se inyecte código malicioso en el lado del cliente. ¿Y cómo lo evita? Pues no dejando que se renderice todo lo que le parezca sospechoso.

Por último tenemos el servicio **firestore.service.ts**<sup>13</sup>, el cual se encargará de gestionar las conexiones con nuestra base de datos, es decir, nos permitirá ejecutar las consultas típicas: Crear, eliminar, editar y consultar. Tenemos que destacar que hemos creado cuatro funciones de consulta en el servicio, en primer lugar tenemos una consulta global que nos proporcionará la colección completa de documentos de la que queremos obtener datos. Luego tenemos otra función que realiza una consulta, sobre la colección indicada, por el ID que le pasemos. Las dos siguientes funciones de consulta realizan una búsqueda por campo en la colección de documentos indicada, con la diferencia de que una de las funciones obtiene los datos una sola vez y la otra obtiene los datos y mantiene abierto un flujo de datos que sincroniza los cambios que se hagan en la BBDD.

Ahora ya conocemos las interfaces y el funcionamiento de los servicios que proveemos en nuestra aplicación, pasaremos a explicar las diferentes páginas de que disponemos para navegar en la app, así como los diferentes componentes que las componen. No vamos a explicar todo el código línea por línea ya que la memoria quedaría demasiado extensa y en nuestro proyecto contamos con muchos archivos de código HTML e Ionic, y código de TypeScript para toda la lógica de la aplicación. Por lo tanto destacaremos partes de código que consideramos más importantes en el programa o funciones que hayamos creado y que merezcan ser mencionadas.

Comencemos por la página de inicio de la aplicación, que es la que hemos llamado Alojamientos, donde podemos destacar tres partes: en la parte superior tenemos un header (encabezado), en la parte inferior tenemos una barra de tabs (botones agrupados) donde tenemos dos tab, uno llamado alojamientos, con icono en forma de llave y nuestra página principal, y otro llamado login, con icono en forma de flecha entrante y que explicaremos posteriormente. Justo entre ellos tenemos el content (contenido o cuerpo) de la página principal, que a su vez incluye en la esquina inferior derecha un FAB (Floating Action Button) con el icono de una

---

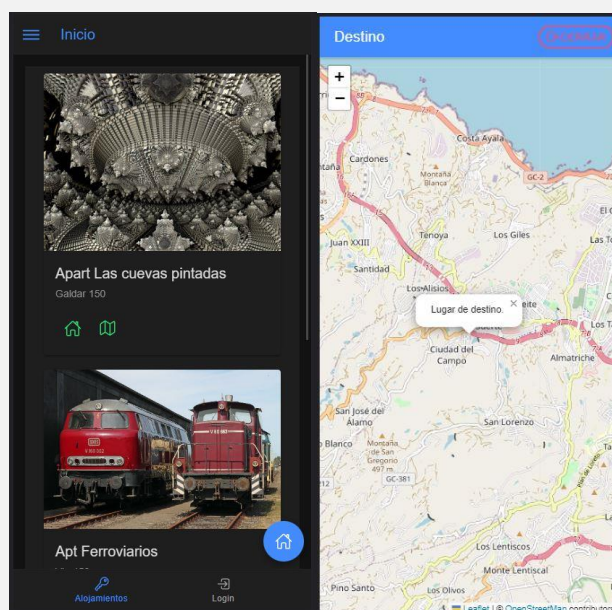
<sup>12</sup> [Ver anexo 5.3](#)

<sup>13</sup> [Ver anexo 5.4](#)

casa. Éste botón nos llevará a una página donde podremos observar nuestras reservas activas. Éste botón solo estará activo si estamos registrados y logeados en la aplicación

El **header-tabs**<sup>14</sup> los hemos creado como un componente a parte el cual se invoca desde la página de inicio, de manera que si la página cambia, el header permanecerá igual. En dicho header tenemos el botón de menú que al clicar nos despliega las tres opciones comentadas anteriormente: Inicio, Configuración y Acerca de. Con la opción Inicio volvemos a la página principal de la aplicación. Con la opción Acerca de nos abrirá una nueva página donde nos encontraremos información de la app como puede ser la versión, el nombre, la compañía, un correo electrónico... Y por último en la opción de Configuración nos abrirá otra página, la cual solo podrá ser gestionada por los administradores, la veremos más adelante.

En el cuerpo de la página principal podemos ver un listado con los diferentes alojamientos que hayan sido registrados en la app. Los alojamientos son presentados en tarjetas a las que en Ionic son llamadas **ion-card**<sup>15</sup>. Cada tarjeta contendrá una serie de elementos descriptivos del alojamiento, estos son: una imagen identificativa, debajo un nombre y la dirección del sitio, y finalmente dos botones tipo icono donde el que tiene forma de casa nos permitirá acceder a la página para iniciar el proceso de reserva (solo activo si se está registrado y logeado); y el otro con forma de plano plegado que al clicar en él nos abrirá una página con un mapa donde saldrá marcada la localización del alojamiento. El mapa también lo hemos generado como un componente aparte, podemos ver el código de configuración en los **anexos**<sup>16</sup>.



<sup>14</sup> [Ver anexo 5.5](#)

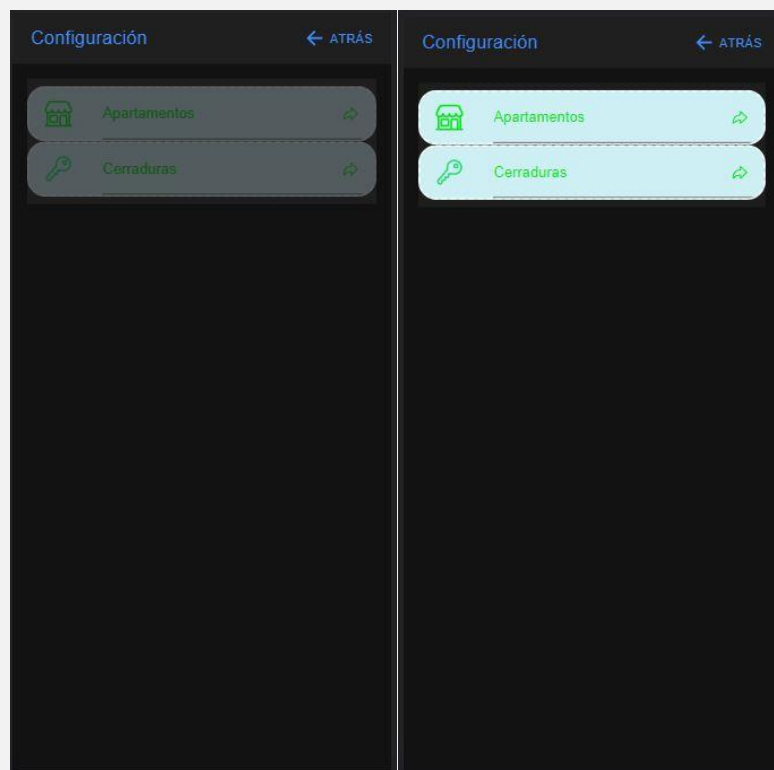
<sup>15</sup> Las tarjetas son contenedores que muestran contenido como texto, imágenes, botones y listas.

<sup>16</sup> [Ver anexo 5.6](#)



Volvamos ahora a nuestro menú en la página principal y centrémonos en el apartado de configuración. Cuando cliquemos en esta opción del menú se abrirá una página donde tendremos un header que cuenta con un botón para retroceder, y luego en el cuerpo tendremos dos botones grandes, los cuales hemos formado con el componente **ion-item**<sup>17</sup>, donde uno de ellos está nombrado como Apartamentos y cuenta con dos iconos en su interior, el de la izquierda tiene forma de tienda. De igual forma el otro botón, que hemos llamado Cerraduras, cuenta con sus dos iconos, siendo el de la izquierda con forma de llave. Los iconos de la derecha son iguales en ambos botones y son flechas direccionales a la derecha. En esta página debemos destacar que dichos botones solo son gestionados por administradores, si el usuario que se logea no tiene dicho rol no podrá interactuar con los botones. Para ello hemos extraído de los datos del usuario local, una vez logeado, el rol con el que está registrado; posteriormente mediante una variable de control de tipo booleana y una sentencia condicional IF activaremos o no dichos botones a través de la propiedad [disabled] que tenemos en nuestro ion-item en el HTML (**ver anexo**<sup>18</sup>).

Veamos en unas capturas de pantalla la diferencia que podemos apreciar si nos logeamos como usuario o como administrador. Como usuario los botones estarán inactivos.

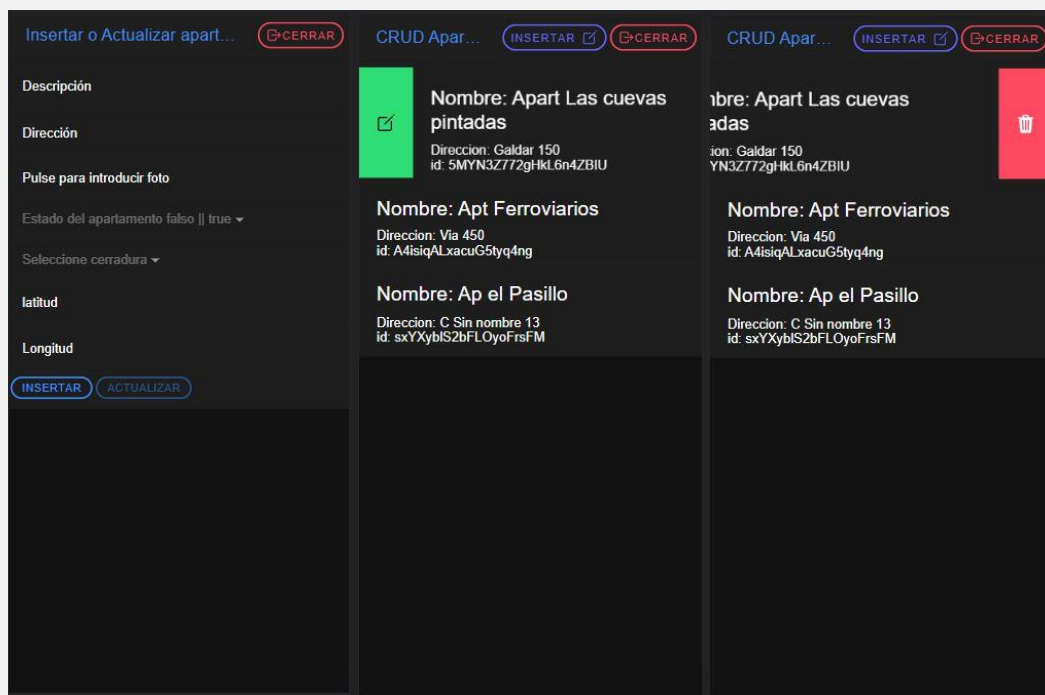


<sup>17</sup> Los items pueden contener texto, íconos, avatares, imágenes, entradas y cualquier otro elemento nativo o personalizado.

<sup>18</sup> [Ver anexo 5.7](#)



Aprovechemos que estamos en el menú de configuración y expliquemos las páginas de configuración de apartamentos y cerraduras que están asociadas a cada botón. Empezando por la de apartamentos, una vez entramos podemos ver un header con el nombre de la página y a la derecha de éste dos botones: uno para insertar apartamentos y otro que cierra la página. Justamente debajo veremos un listado con todos los alojamientos registrados y sus datos con el añadido de que tendremos el ID de la base de datos. Si clicamos en el botón de insertar se nos abrirá una página donde podremos ver un formulario para rellenar con los datos del alojamiento, podremos sacar una foto con el Smartphone o cogerla de la galería, con la geolocalización permitirá localizar el apartamento automáticamente con la latitud y longitud, y finalmente le asignaremos una cerradura e indicamos si está ocupado o no. Para acabar clicamos en insertar y el alojamiento quedará guardado en la BBDD y nuestro listado se actualizará automáticamente. También podremos editar o eliminar dichos alojamientos, para ello deslizaremos hacia la izquierda o la derecha, mediante un **ion-item-sliding**<sup>19</sup>, con el que queramos interactuar. El slide verde nos permite editar el apartamento y el slide rojo lo elimina. La página para editar es exactamente la misma que la de insertar con la diferencia de que tendrá un botón actualizar habilitado. Podremos ver el código del ion-item-sliding en los **anexos**<sup>20</sup>.

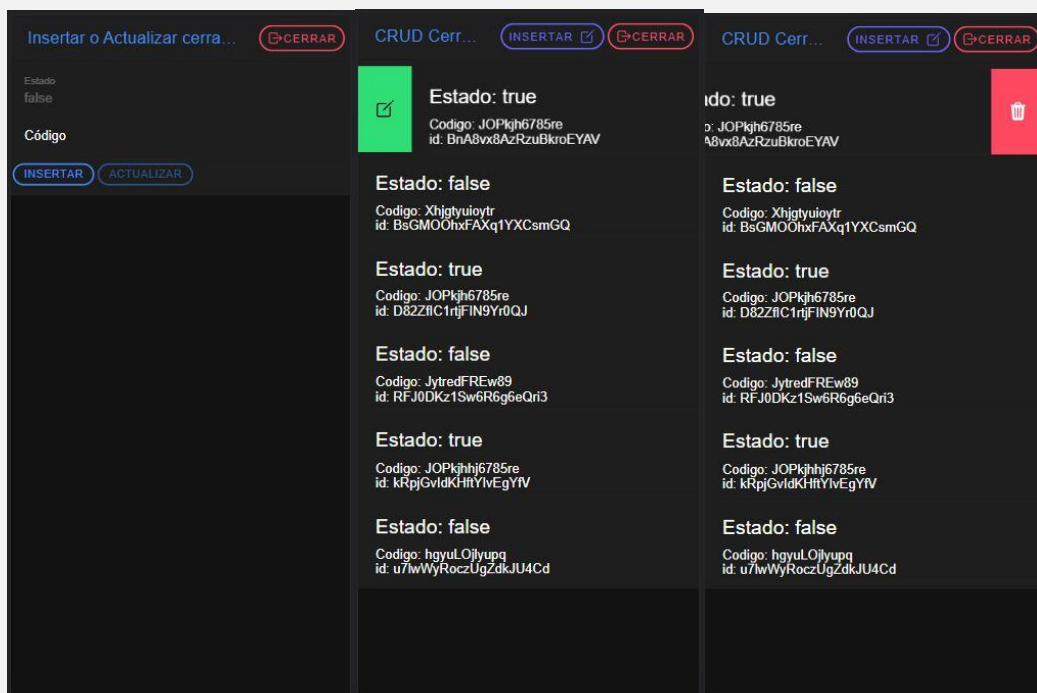


Pasemos a explicar ahora la página de cerraduras. El diseño de ésta es el mismo que la de apartamentos con la única diferencia de que el listado que aparece es el de las cerraduras que

<sup>19</sup> Elemento que se puede arrastrar para revelar los botones de opción.

<sup>20</sup> [Ver anexo](#)

tenemos registradas. Cada cerradura será registrada en la BBDD con dos campos: el estado (true o false dependiendo de si está activada o no) y el código de la cerradura con el que podremos asignarla a los respectivos alojamientos. La forma de editar y eliminar cerraduras es igual que el de apartamentos, mediante un ion-item-sliding.



Ahora ya conocemos el funcionamiento de la parte que gestiona un administrador, pasemos a la parte del login, logout y registro de usuarios. En ésta página podemos ver que en el contenido tenemos en primer lugar un pequeño formulario donde podemos introducir el email y contraseña para logearnos. Justo debajo tenemos tres botones: El botón de login permitirá al usuario acceder a la app siempre que el correo y contraseña del formulario sean correctos. El botón de Create Account nos lanzará a otra página con un formulario para registrarnos, la cual veremos posteriormente. Finalmente el botón de logout que como indica su nombre cerrará la sesión si estamos logeados. También contamos con un FAB (Floating Action Button) en la esquina inferior derecha que nos permitirá cerrar la aplicación en nuestro Smartphone.

El formulario lo hemos creado mediante un **ion-input**<sup>21</sup> con el cual podremos recoger los datos introducidos y validarlos antes de realizar el login ya que pueden surgir errores en el inicio de sesión. Estos errores nos los devuelve Firebase y para gestionarlos hemos creado una función que recibe como parámetro el código de error que nos lanza Firebase y en función del tipo de error nos retornará un mensaje que mostraremos en una alerta en pantalla, que a su vez

<sup>21</sup> Contenedor para el elemento de entrada HTML con estilo personalizado y funcionalidad adicional.

es gestionada por nuestro servicio de alertas. Veamos en una captura de pantalla cuales son los errores que nos podemos encontrar al realizar un login.

```

error(err: string){
  if (err==='FirebaseError: Firebase: The email address is badly formatted. (auth/invalid-email).')
  {
    err='El campo mail no está correctamente formado';
  }
  // eslint-disable-next-line eqeqeq
  if (err==='FirebaseError: Firebase: An internal AuthError has occurred. (auth/internal-error).')
  {
    err='No se ha introducido ninguna contraseña';
  }

  // eslint-disable-next-line max-len
  if (err==='FirebaseError: Firebase: There is no user record corresponding to this identifier. The user may have been deleted. (auth/user-not-found).')
  {
    err='No se ha encontrado el usuario o éste ha sido eliminado';
  }
  // eslint-disable-next-line max-len
  if (err==='FirebaseError: Firebase: A network AuthError (such as timeout, interrupted connection or unreachable host) has occurred. (auth/network-request-failed).')
  {
    err='Error de red hay un fallo en la conexión';
  }
  return err;
}

```

Lo siguiente que explicaremos será nuestra página de Create Account, en donde podrán registrarse nuevos usuarios. De igual manera contamos en ésta página con un formulario que nos solicitará un correo electrónico, la contraseña para nuestra cuenta y la confirmación de la misma.

El sistema de registro y autenticación lo hemos delegado en otra de las funciones de Google Firebase, Google Authentication. Con éste servicio podemos habilitar diferentes proveedores de acceso a nuestra aplicación como, por ejemplo, Google, Facebook, Apple... O tan solo un correo y una contraseña, que es el método que hemos usado nosotros. Además contamos con muchas más funciones como restablecimiento de contraseñas, verificación en dos pasos, cambio de correo electrónico...

## Authentication

Users **Sign-in method** Templates Usage Settings

Proveedores de acceso

[Agregar proveedor nuevo](#)

Proveedor	Estado
<p>Selecciona un proveedor de acceso (paso 1 de 2)</p>	
<p><b>Proveedores nativos</b></p> <div style="margin-bottom: 10px;"> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;">  Correo electrónico/contraseña ✓         </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;">  Teléfono         </div> <div style="border: 1px solid #ccc; padding: 5px;">  Anónimo         </div> </div>	<p><b>Proveedores adicionales</b></p> <div style="display: grid; grid-template-columns: repeat(3, 1fr); gap: 5px;"> <div style="border: 1px solid #ccc; padding: 5px; text-align: center;">  Google         </div> <div style="border: 1px solid #ccc; padding: 5px; text-align: center;">  Facebook         </div> <div style="border: 1px solid #ccc; padding: 5px; text-align: center;">  Play Juegos         </div> <div style="border: 1px solid #ccc; padding: 5px; text-align: center;">  Game Center         </div> <div style="border: 1px solid #ccc; padding: 5px; text-align: center;">  Apple         </div> <div style="border: 1px solid #ccc; padding: 5px; text-align: center;">  GitHub         </div> <div style="border: 1px solid #ccc; padding: 5px; text-align: center;">  Microsoft         </div> <div style="border: 1px solid #ccc; padding: 5px; text-align: center;">  Twitter         </div> <div style="border: 1px solid #ccc; padding: 5px; text-align: center;">  Yahoo         </div> </div>
<p><b>Proveedores personalizados</b></p> <div style="margin-bottom: 10px;"> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;">  OpenID Connect         </div> <div style="border: 1px solid #ccc; padding: 5px;">  SAML         </div> </div>	
<div style="display: flex; justify-content: space-around; align-items: center;"> <div>  Correo electrónico/contraseña         </div> <div> <span style="color: green;">Habilitado</span> </div> </div>	

De ésta manera damos más seguridad a los datos de los usuarios ya que quedarán almacenados en el servicio de Google, para ello hemos importado un módulo de interfaces llamado AngularFireAuth, que se encargará de conectar nuestra app con Google Authentication para posteriormente realizar la inserción del usuario en la BBDD. Podemos ver el código en los **anexos**<sup>22</sup>. Además podremos ver y gestionar el listado de usuarios registrados desde la consola de administración de Google Firebase. Podremos ver los parámetros de encriptado de contraseña, inhabilitar o borrar cuentas o resetear las contraseñas

## Authentication

Users **Sign-in method** Templates Usage Settings

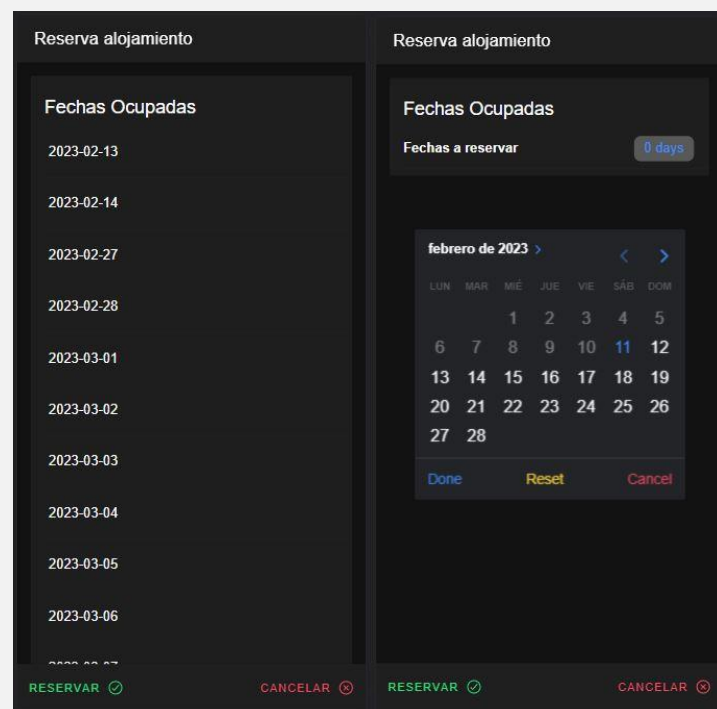
[Agregar usuario](#)

Identificador	Proveedores	Fecha de creación	↓	Fecha de acceso	UID de usuario
adm2@ppp.es		11 feb 2023		11 feb 2023	TJWdWwPb9cPwlj6KJopYzeZaEbB2
adm@ppp.es		5 feb 2023		11 feb 2023	1VwyuAWTVmPvqo9Wo1V6wMoo...
jmn2@ppp.es		1 feb 2023		6 feb 2023	yRlswm8LYdZqu50fiQW2NBaksh1
jmn1@ppp.es		30 ene 2023		6 feb 2023	6WJJWv7I43XDti5z5m3CqVuF4xV2
jmnuser@ppp.es		19 ene 2023		1 feb 2023	4VtVeZdgqDOuWi50phijxa6U4BD2
jmnadmin@ppp.es		19 ene 2023		5 feb 2023	VH6eSoWKIoN287VLNds28o25Jls2

Filas por página: 50    1 - 6 of 6    < >

<sup>22</sup> [Ver anexo 5.9](#)

Volvamos a la página de inicio para explicar los dos últimos puntos de nuestra app: el proceso de reservar alojamientos y la página para ver las reservas activas. Como ya hemos visto, en la página principal nos saldrá un listado con los alojamientos que hay registrados. Éstos están presentados en tarjetas individuales con sus respectivos datos y en la parte inferior tenemos el botón de mostrar en el mapa y el botón de reservar. Una vez cliquemos en el botón reservar nos llevará a una página donde nos mostrará un listado de fechas en las cuales el alojamiento está ya ocupado, impidiendo así que reserven varios usuarios el mismo alojamiento en las mismas fechas. Ahora para reservar nos dirigimos a la parte inferior de la página, donde podremos ver un botón que pone Fechas a reservar, una vez cliquemos nos abrirá un calendario en el que podremos seleccionar los días que queramos reservar, quedando inactivos los días que no podamos seleccionar por estar ocupado. Una vez seleccionemos los días que queramos clicamos en done y se guardará la reserva de forma temporal en el dispositivo, para confirmarla debemos clicar en el botón reservar que encontramos en el pie de la página. Una vez tengamos la reserva nos aparecerá en el apartado de Mis Reservas que veremos ahora.

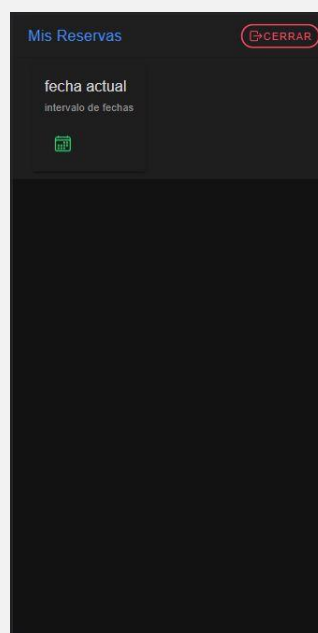


Explicuemos el código del calendario ya que cuenta con una gran cantidad de funciones que podemos usar para personalizarlo según el uso que queramos darle. Toda la configuración del calendario va en el HTML ya que es un componente de Ionic. El calendario forma parte del componente **ion-datetime** y para que éste funcione debe ser englobado dentro de un modal, que es una ventana que nos impide interactuar con el resto de la aplicación hasta finalizar las interacciones con dicho modal; y a su vez en un **ng-template**, que es una plantilla que se

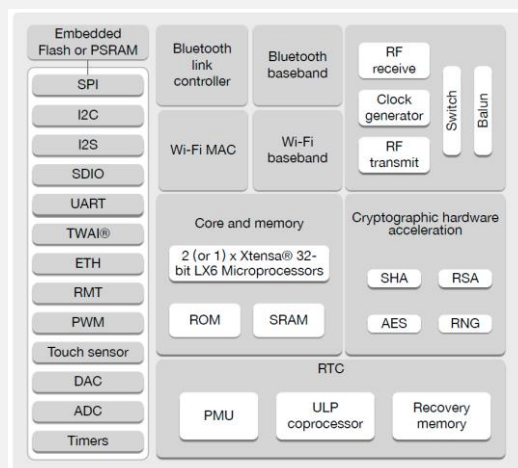
presenta en la app cuando nosotros queramos que se renderice. Ahora expliquemos los diferentes parámetros que tiene nuestro calendario: En primer lugar tenemos que asignarle un identificador para poder invocarlo y hacer que se genere al abrir la página; luego hemos hecho que se presente como si fuera en un dispositivo IOS y que solo sea un calendario; también hemos activado la opción múltiple para seleccionar varios días a la vez; hemos ajustado como primer día de la semana el lunes y como fecha mínima disponible el día de hoy, de manera que no podremos seleccionar días ya pasados. Finalmente tiene un evento asociado llamado llegada, el cual se encarga de coger las fechas seleccionadas y cargarlas en un arreglo para poder trabajar con ellas.

```
<!-- Montamos el calendario de fecha de llegada -->
<ion-modal [keepContentsMounted]="true">
  <ng-template>
    <ion-datetime #date0
      id="llegada"
      mode="ios"
      presentation="date"
      [multiple]="true"
      [firstDayOfWeek]="1"
      [min]="fechaActual.toISOString()"
      (ionChange)="llegada($event)">
    <ion-buttons slot="buttons">
      <ion-button color="primary" (click)="date0.confirm()">Done</ion-button>
      <ion-button color="warning" (click)="date0.reset()">Reset</ion-button>
      <ion-button color="danger" (click)="date0.cancel()">Cancel</ion-button>
    </ion-buttons>
  </ion-datetime>
</ng-template>
</ion-modal>
```

Finalmente expliquemos la página de Mis Reservas, aquí el usuario logeado podrá visualizar sus reservas así como gestionirlas. Además desde ésta página es desde donde el usuario tendrá que abrir la cerradura clicando en su respectivo código.



Ahora ya conocemos como funciona nuestra aplicación móvil, pero ahora debemos explicar cómo funciona el apartado de apertura de la cerradura con nuestro ESP32.



En principio el ESP32 es un microcontrolador que tiene todo lo necesario para conectarse al exterior si necesidad de conexión física por cable, salvo cuando se programa. Es de origen Chino y compite abiertamente con Arduino, Ardafruit Microchip... La ventaja que tiene es su precio competitivo y la cantidad de funciones que posee, aunque para nuestro desarrollo se eligió por el precio, la conexión inalámbrica sin necesidad de shields y lo fácil que es conseguirlo. Para su programación tiene un IDE Oficial, pero pocos lo usan, normalmente se usa Arduino si lo programas en C, también para este lenguaje puedes usar PlatformIO.

En nuestro caso el lenguaje que se usó es MicroPython, por lo que se optó por Thonny VCode. En nuestro caso Python es un lenguaje más accesible, a cambio tiene la desventaja que no es el lenguaje nativo del Sistema lo que conlleva que hay que añadirle el intérprete al ESP32.

La Mecánica del programa es sencilla utilizando las funcione BLE (bluetooth de baja energía) podemos controlar una de las GPIO (Puertos de entrada o salida) para que nos genere un pulso cuando nos interese usando un código predeterminado.

Podremos ver en los **anexos**<sup>23</sup> la estructura de código que maneja la controladora ESP32 para gestionar la conexión con bluetooth y el protocolo de red MQTT para la conexión con el servidor externo.

<sup>23</sup> [Ver anexo 5.10](#)



## Integración y Costes Empresariales

Como ya hemos visto, nuestra aplicación está orientada a dispositivos móviles, y pensada para aquellas personas que cuenten con viviendas vacacionales que quieran alquilar y obtener una rentabilidad y para facilitar a los usuarios que quieran alquilar una vivienda o apartamento el proceso de reservar y tener la llave de la cerradura electrónica en su Smartphone.

La codificación de la aplicación ha superado con creces las 90 horas de trabajo, pudiendo así generar, en ese tiempo, una aplicación funcional y lo más completa posible, un prototipo que es totalmente escalable y con margen de mejora. En España un programador promedio cobra 14,38€ la hora aproximadamente o 28.040€ al año.

Por otro lado tenemos la parte de la cerradura que consta de la propia cerradura y la controladora con el ESP32. Cada placa ESP32 WiFi Kit, que es el que hemos usado, podemos encontrarla a partir de 5 o 10 € en [Aliexpress](#). Y podemos encontrar cerraduras controladas electrónicamente en [Amazon](#) a partir de 15€. Estos costes son orientativos ya que con el tiempo y crecimiento de la aplicación las prestaciones también crecerán, pudiendo ofrecer mejores dispositivos y servicios.

Finalmente el coste de despliegue de la aplicación, hosting, base de datos etc. Lo hemos centralizado todo en Google Firebase, ya que cuenta con muchas posibilidades y servicios muy escalables. En un principio la aplicación puede ser desplegada de forma gratuita hasta ciertos mínimos que nos expone el servicio de Firebase. Una vez empiece a crecer el número de usuarios podremos ampliar nuestros planes de productos sin ningún problema. Podemos ver la tabla de precios de los servicios de Google Firebase en su [web](#).

## Conclusiones

Inicialmente la aplicación fue llamada ArduKey, ya que se pensó en usar un Arduino, pero posteriormente se decidió usar un ESP32 por ofrecernos más posibilidades en el ámbito de la domótica, por lo que ahora la hemos llamado ESP-Lock Opener.

Nuestra aplicación ha sido desarrollada como prototipo funcional, pero aún se puede mejorar mucho más la estética e incluso añadirle o mejorarle algún servicio de los que ofrecemos. Hemos alcanzado los objetivos propuestos dentro del límite de tiempo del que disponemos y estamos en parte contentos con el resultado final de la aplicación pero, por otro lado, nos hubiera gustado poder ofrecer algo con un diseño muchísimo más atractivo y trabajado y con funciones más completas.

## Bibliografía

<https://angular.io/>

<https://ayudaleyprotecciondatos.es/bases-de-datos/documentales/>

<https://biblus.us.es/bibing/proyectos/abreproy/40048/fichero/VOLUMEN+1.+MEMORIA%252F4.+Tecnolog%C3%AD+Bluetooth.pdf>

<https://docs.micropython.org/en/latest/library/bluetooth.html#class-ble>

<https://firebase.google.com/?hl=es>

<https://github.com/repository/activity/angular/angularfire>

<https://ionicframework.com/>

<https://ionicframework.com/docs/v3/native/ble/>

<https://javadesde0.com/analizando-la-estructura-del-directorio-src-de-un-proyecto-de-angular/>

<https://leafletjs.com/>

<https://learn.adafruit.com/adafruit-feather-m0-bluefruit-le/bleuart>

<https://medium.com/canariasjs/estructurando-datos-en-cloud-firestore-3dce2cb1ceae>

<https://www.openstreetmap.org/#map=6/40.007/-2.488>

<https://openwebinars.net/blog/que-es-firebase-de-google>

<https://stackoverflow.com/questions/74745954/error-angular-fire-build-incorrectly-extends-interface>

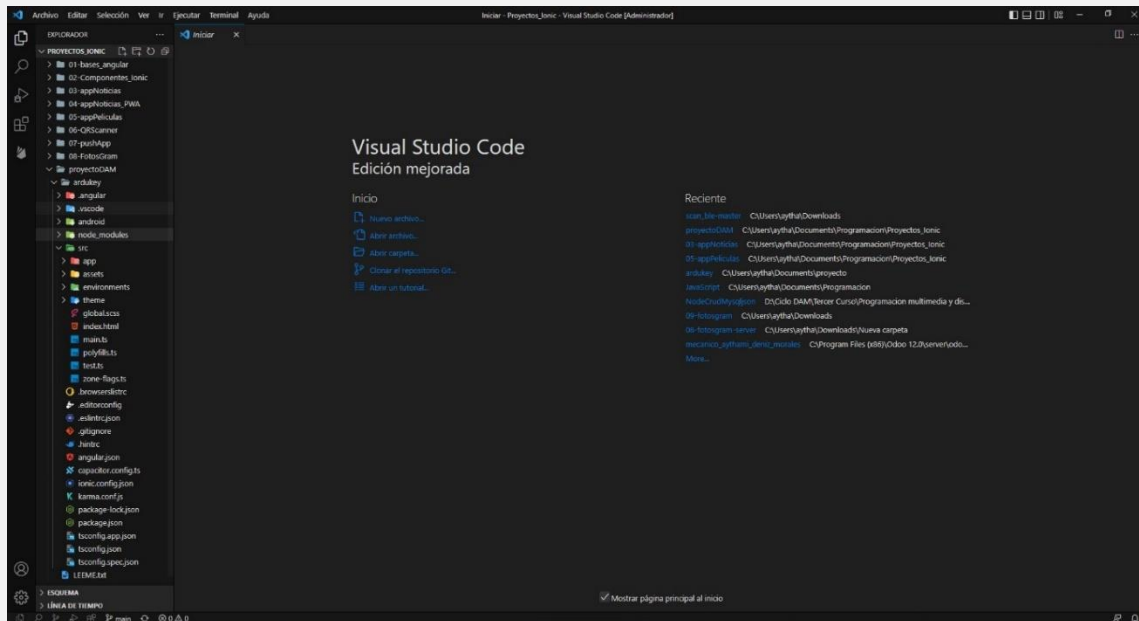
<https://www.electrosoftcloud.com/esp32-empezando-a-usar-el-bluetooth-spp/>

<https://www.juanjobeunza.com/esp32-ble/>

<https://www.mokoblue.com/es/mokoblue-esp32-gateway/>

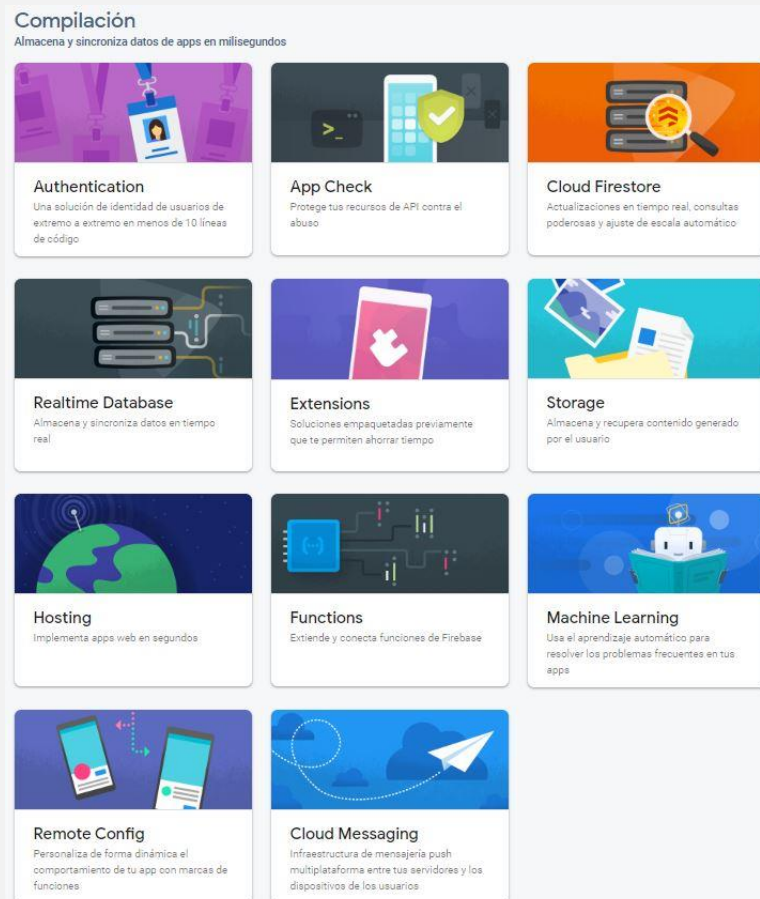
## Anexos

### Anexo 1: Visual Studio Code



[Volver al apartado](#)

### Anexo 2: Productos Google Firebase



## Lanzamiento y supervisión

Controla la calidad de la app



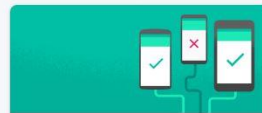
### Crashlytics

Haz un seguimiento en tiempo real de los problemas de estabilidad que afectan la calidad de una app, para priorizarlos y corregirlos



### Performance

Obtén estadísticas sobre el rendimiento de la app



### Test Lab

Realiza pruebas en una amplia gama de dispositivos



### App Distribution

Distribuye versiones de app previas al lanzamiento



### Remote Config

Lanza nuevas funciones de forma lenta y segura en tu app sin implementar una versión nueva

## Participación

Aumenta tu público y haz que participe



### A/B Testing

Mejora los flujos y las notificaciones clave



### Cloud Messaging

Haz que los usuarios vuelvan a participar con notificaciones orientadas, automatizadas y personalizadas



### In-App Messaging

Envía mensajes para atraer a los usuarios correctos en el momento indicado



### Remote Config

Personaliza y optimiza la experiencia para tus usuarios



### Dynamic Links

Usa vínculos directos para enviar a los usuarios al lugar adecuado en tu app



### Crashlytics

Descubre cómo los problemas de estabilidad afectan a las métricas empresariales, como los ingresos y la participación



### Authentication

Genera conversiones de integración con herramientas como Autenticación anónima

## Estadísticas

Análisis de apps sin cargo y sin límites



### Google Analytics

Mide y analiza la participación de los usuarios

[Volver al apartado](#)

## Anexo 3: Estructura de la Base de Datos

<a href="#">🏠</a> > alquileres > u17CE1CWuPS... <span>Más funciones en Google Cloud</span>		
ardukey-c06e8	alquileres	u17CE1CWuPSRX93n6E2i
<a href="#">+ Iniciar colección</a>	<a href="#">+ Agregar documento</a>	<a href="#">+ Iniciar colección</a>
alquileres >	u17CE1CWuPSRX93n6E2i >	<a href="#">+ Agregar campo</a>
apartamentos cerraduras usuarios		F_FIN: 29 de enero de 2023, 12:00:00 UTC F_INICIO: 29 de diciembre de 2022, 12:00:00 UTC IDPROP: /apartamentos/BjSolkjLWnCL8guc63qi IDUSER: "4VtVeZdgqDOuWi50phlJxa6U4BD2"

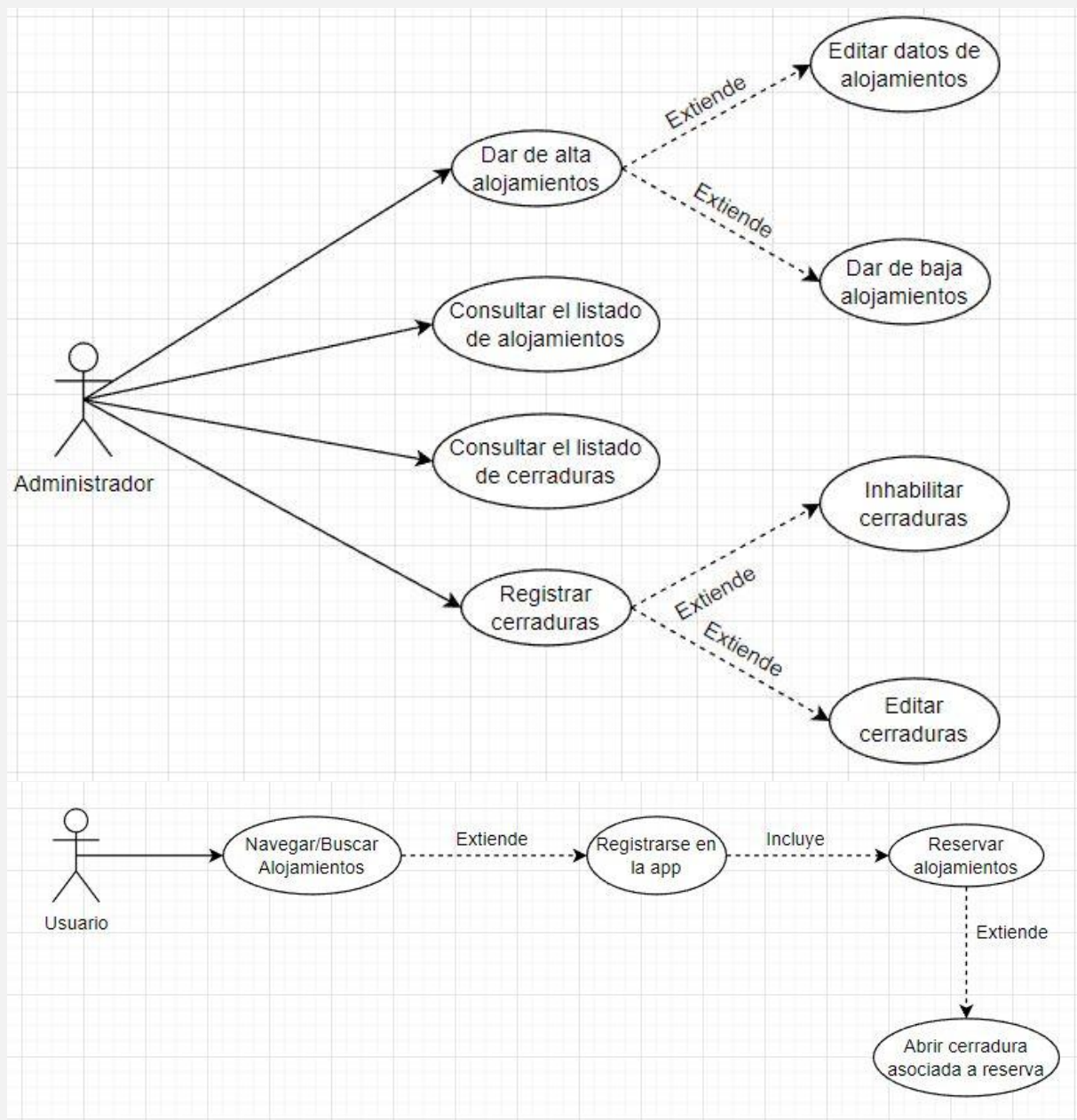
<a href="#">🏠</a> > apartamentos > A4isiqALxacuG... <span>Más funciones en Google Cloud</span>		
ardukey-c06e8	apartamentos	A4isiqALxacuG5tyq4ng
<a href="#">+ Iniciar colección</a>	<a href="#">+ Agregar documento</a>	<a href="#">+ Iniciar colección</a>
alquileres <b>apartamentos</b> > cerraduras usuarios	A4isiqALxacuG5tyq4ng > Ho3i4BVegGo8tw3c2i8b sxYXyb1S2bFL0yoFrsFM	<a href="#">+ Agregar campo</a>
		DESCRIPCION: "Apt Ferroviarios" DIRECCION: "Via 450" ESTADO: "no ocupado" IDKEY: "D82ZfIC1rtjFIN9Yr0QJ" IMG: "data:image/png;base64,/9j/4AAQSkZJRgABAQGAIAAAAD/2wBDAAUDBAQE... LAT: 28.128248789341917 LON: -15.446629308226036

<a href="#">🏠</a> > cerraduras > D82ZfIC1rtjFIN9... <span>Más funciones en Google Cloud</span>		
ardukey-c06e8	cerraduras	D82ZfIC1rtjFIN9Yr0QJ
<a href="#">+ Iniciar colección</a>	<a href="#">+ Agregar documento</a>	<a href="#">+ Iniciar colección</a>
alquileres apartamentos <b>cerraduras</b> > usuarios	BnA8vx8AzRzuBkroEYAV BsGM00hxFAQ1YXCsmGQ <b>D82ZfIC1rtjFIN9Yr0QJ</b> > kRpjGvIdKHftYIvEgYfV u7lwWYRoczUgZdkJU4Cd	<a href="#">+ Agregar campo</a>
		ACTIVA: true CODIGO: "JOPkjh6785re"

<a href="#">🏠</a> > usuarios > AXkYcIO6Z9EA... <span>Más funciones en Google Cloud</span>		
ardukey-c06e8	usuarios	AXkYcIO6Z9EABNrWbUzI
<a href="#">+ Iniciar colección</a>	<a href="#">+ Agregar documento</a>	<a href="#">+ Iniciar colección</a>
alquileres apartamentos cerraduras <b>usuarios</b> >	9gy9Kq4X1Kgk01mcCd8G <b>AXkYcIO6Z9EABNrWbUzI</b> > B0FeRpLUWLqhHTGGk8fz Bc6x9Vju4GmJsOQKHk4q KB0BW5K9tXjWfEdsK4Sn PECFTMNw45t1m8Xmx3ba	<a href="#">+ Agregar campo</a>
		EMAIL: "jmnuser@ppp.es" ROL: "user" UID: "4VtVeZdgqDOuWi50phlJxa6U4BD2"

[Volver al apartado](#)

## Anexo 4: Diagrama Casos de Uso

[Volver al apartado](#)



## Anexo 5: Desarrollo de la Aplicación

### 5.1 Componente Menú

```

1  <ion-menu side="start" type="overlay" menuId="menu" contentId="principal">
2
3      <ion-header [translucent]="true">
4          <ion-toolbar color="primary">
5              <ion-title size="small">ArduKey</ion-title>
6          </ion-toolbar>
7      </ion-header>
8
9      <ion-content [fullscreen]="true">
10
11          <ion-list>
12              <ion-menu-toggle *ngFor="let opcion of opcionesMenu | async" autoHide="false">
13
14                  <ion-item [routerLink]="opcion.redirectTo" detail>
15                      <ion-icon [name]="opcion.icon" slot="start" color="primary"></ion-icon>
16                      <ion-label>{{opcion.name}}</ion-label>
17                  </ion-item>
18              </ion-menu-toggle>
19          </ion-list>
20      </ion-content>
21  </ion-menu>

```

```

12  export class MenuComponent implements OnInit {
13      rol = '';
14      uid = '';
15      coleccionUsuarios: any = [{
16          id: '',
17          data: {} as Usuario
18      }];
19      opcionesMenu: Observable<menuOpts[]> | undefined;
20
21      constructor(
22          private menuOptsService: MenuOptsService,
23          private firestoreService: FirestoreService
24      ) {}
25
26      ngOnInit() {
27          this.obtenerListaUsuarios()
28          // this.comprobarRol()
29          this.opcionesMenu = this.menuOptsService.getMenuOptions(localStorage.getItem('rol')!);
30      }
31
32      obtenerListaUsuarios(){
33          console.log('dentro de obtener Lista ' + localStorage.getItem('uid'));
34          this.firestoreService.consultar("usuarios").subscribe((consulta: any[]) => {
35              this.coleccionUsuarios = [];
36              console.log('dentro de resultadoConsulta');
37              consulta.forEach((datosTarea: any) => {
38                  this.coleccionUsuarios.push({
39                      id: datosTarea.payload.doc.id,
40                      data: datosTarea.payload.doc.data()
41                  });
42                  //console.log(this.coleccionUsuarios);
43              })
44              //console.log('obtener lista usuario ' + this.coleccionUsuarios[0].data.ROL);
45              this.coleccionUsuarios.forEach((element: any) => {
46                  if (element.data.UID === localStorage.getItem('uid')) {localStorage.setItem('rol', element.data.ROL)}
47              });
48              console.log('Rol Usuario es ' + localStorage.getItem('rol'))
49          });
50          console.log('Rol Usuario es ' + localStorage.getItem('rol'))
51      }
52  }

```

```

menu-opts.service.ts X
proyectoDAM > ardukey > src > app > services > menu-opts.service.ts > MenuOptsService
1 import { Injectable } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3 import { menuOpts } from '../interfaces/interfaces';
4
5 @Injectable({
6   providedIn: 'root'
7 })
8
9 export class MenuOptsService {
10
11   constructor(private http: HttpClient) { }
12
13   getMenuOptions(rol:string) {
14     return this.http.get<menuOpts[]>('/assets/data/menu.json');
15   }
16 }
17
18
19
20

```

[Volver al apartado](#)

## 5.2 Servicio alert.service.ts

```

alert.service.ts X
proyectoDAM > ardukey > src > app > services > alert.service.ts > AlertService > registerAlert
1 import { Injectable } from '@angular/core';
2 import { AlertController } from '@ionic/angular';
3
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class AlertService {
8
9   constructor(private alertCtrl: AlertController) { }
10
11   async registerAlert(status: string, sms: string) {
12     const alert = await this.alertCtrl.create({
13       header: status,
14       subHeader: sms,
15       buttons: ['Listo']
16     });
17     await alert.present();
18   }
19 }
20
21

```

[Volver al apartado](#)

## 5.3 Servicio img.service.ts

```

img.service.ts X
proyectoDAM > ardukey > src > app > services > img.service.ts > ImgService
1 import { Injectable } from '@angular/core';
2 import { DomSanitizer } from '@angular/platform-browser';
3
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class ImgService {
8
9   constructor(private sanitizer: DomSanitizer) { }
10   getImage(foto: any){
11     return this.sanitizer.bypassSecurityTrustResourceUrl(foto);
12   }
13 }
14

```

[Volver al apartado](#)

## 5.4 Servicio firestore.service.ts

```

5  ~ @Injectable({
6    providedIn: 'root'
7  })
8  ~ export class FirestoreService {
9
10   constructor(private angularFirestore: AngularFirestore) { }
11   //insertar
12   ~ public insertar(coleccion: any, datos: any) {
13     return this.angularFirestore.collection(coleccion).add(datos);
14   }
15
16   ~ public consultar(coleccion: any) {
17     return this.angularFirestore.collection(coleccion).snapshotChanges();
18   }
19
20   ~ public borrar(coleccion: string, documentId: string) {
21     return this.angularFirestore.collection(coleccion).doc(documentId).delete();
22   }
23
24   ~ public actualizar(coleccion: string, documentId: string, datos: any) {
25     return this.angularFirestore.collection(coleccion).doc(documentId).set(datos);
26   }
27
28   ~ public consultarPorId(coleccion: string, documentId: string) {
29     return this.angularFirestore.collection(coleccion).doc(documentId).get();
30   }
31 }
32
33 ~ /**
34   this.comments$ = afs.collection('Comments', ref => ref.where('user', '==', userId))
35     .snapshotChanges();
36   */
37 ~ public consultarPorCampo (coleccion: string, campo: string, predicado: any) {
38   console.log(coleccion ,campo, predicado);
39   return this.angularFirestore.collection(coleccion, ref => ref.where(campo, '==', predicado)).snapshotChanges();
40 }
41 ~ public consultarPorCampoGet (coleccion: string, campo: string, predicado: any) {
42   console.log(coleccion ,campo, predicado);
43   return this.angularFirestore.collection(coleccion, ref => ref.where(campo, '==', predicado)).get();
44 }

```

[Volver al apartado](#)

## 5.5 Componente header-tabs

```

1  <ion-header class="ion-no-border" [translucent]="true">
2    <ion-toolbar>
3      <ion-buttons slot="start">
4        <ion-menu-button color="primary" menu="menu" mode="ios" autoHide="false"></ion-menu-button>
5      </ion-buttons>
6      <ion-title color="primary">{{titulo}}</ion-title>
7    </ion-toolbar>
8  </ion-header>

```

[Volver al apartado](#)

## 5.6 Componente maps

```

maps.component.html x
proyectoDAM > ardukey > src > app > components > maps > maps.component.html > ion-header
1 <ion-header [translucent]="true">
2   <ion-toolbar color="primary">
3     <ion-title>
4       Destino
5     </ion-title>
6     <ion-buttons slot="end">
7       <ion-button slot="end" shape="round" color="danger" fill="outline" size="small" (click)="cerrar()">
8         <ion-icon shape="round" name="log-out-outline"></ion-icon> Cerrar
9       </ion-button>
10    </ion-buttons>
11  </ion-toolbar>
12 </ion-header>
13
14 <ion-content >
15   <div id="mapId" style="width: 100%; height: 100%">
16   </div>
17 </ion-content>
18
19
20 1 import { Component, OnInit } from '@angular/core';
21 2 import { ModalController } from '@ionic/angular';
22 3 import * as L from 'leaflet'; // Importamos L de leaflet para renderizar el mapa.
23 4
24 5
25 6 @Component({
26 7   selector: 'app-maps',
27 8   templateUrl: './maps.component.html',
28 9   styleUrls: ['./maps.component.scss'],
29 10 })
30 11 export class MapsComponent implements OnInit {
31 12   lon:any;
32 13   lat:any;
33 14   map: L.Map | undefined;
34 15   center:L.PointTuple | undefined;
35 16   miMarker:L.Icon = new L.Icon({ iconUrl: 'assets/img/coordenada.png', iconsize: [48, 48], iconAnchor:[24, 43] })
36 17
37 18   constructor(private modalCtrl: ModalController) { }
38 19
39 20   ngOnInit() {
40 21
41 22   }
42 23   ionViewDidEnter(){
43 24     this.center=[this.lat,this.lon];
44 25     this.map = L.map('mapId').setView(this.center, 13);
45 26     L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
46 27       attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
47 28     }).addTo(this.map);
48 29     L.marker(this.center, {icon: this.miMarker}).addTo(this.map)
49 30     .bindPopup('Lugar de destino.')
50 31     .openPopup();
51 32
52 33   }
53 34
54 35   cerrar(){this.modalCtrl.dismiss();}
55 36
56 37
57 38

```

[Volver al apartado](#)

## 5.7 Botones panel configuración

```

proyectoDAM > ardukey > src > app > pages > settings > settings.page.html > ion-content.ion-padding > ion-list > ion-item
1 <app-header titulo="Configuración"></app-header>
2
3 <ion-content class="ion-padding" [fullscreen]="true">
4
5   <ion-list fixed>
6     <ion-item detail="true" (click)="apartamentos()" [disabled]="control" detailIcon="arrow-redo-outline">
7       <ion-label>Apartamentos</ion-label>
8       <ion-icon slot="start" size="large" color="success" name="storefront-outline"></ion-icon>
9     </ion-item>
10    <ion-item (click)="cerraduras()" detail="true" [disabled]="control" detailIcon="arrow-redo-outline">
11      <ion-icon slot="start" size="large" color="success" name="key-outline"></ion-icon>
12      <ion-label>Cerraduras</ion-label>
13    </ion-item>
14  </ion-list>
15
16
17
18
19
20
21 </ion-content>

```

```

@Component({
  selector: 'app-settings',
  templateUrl: './settings.page.html',
  styleUrls: ['./settings.page.scss'],
})
export class SettingsPage implements OnInit {

  control = true;

  constructor(private modalCtrl: ModalController) { }

  ngOnInit() {
    if(localStorage.getItem('rol') === 'admin'){
      this.control=false;
    }
  }

  async apartamentos(){
    const modal = await this.modalCtrl.create({
      component: ApartamentosComponent
    });
    await modal.present();
    console.log('presenta apartamento')
  }

  async cerraduras(){
    const modal = await this.modalCtrl.create({
      component: CerradurasComponent
    });
    await modal.present();
    console.log('presenta cerraduras')
  }
}

```

[Volver al apartado](#)

## 5.8 Ion-item-sliding

```

<ion-list>
  <ion-item-sliding *ngFor="let documentApartamento of coleccionApartamentos">
    <ion-item-options side="start">
      <ion-item-option color="success" (click)="actualizar(documentApartamento.id)">
        <ion-icon slot="icon-only" name="create-outline"></ion-icon>
      </ion-item-option>
    </ion-item-options>
    <ion-item>
      <ion-grid>
        <ion-row>
          <h2>Nombre: {{documentApartamento.data.DESCRIPCION}}</h2>
        </ion-row>
        <ion-row>
          Direccion: {{documentApartamento.data.DIRECCION}}
        </ion-row>
        <ion-row>
          id: {{documentApartamento.id}}
        </ion-row>
      </ion-grid>
    </ion-item>
    <ion-item-options side="end" >
      <ion-item-option color="danger"
        (click)="borrar( documentApartamento.id,documentApartamento.data.ESTADO,documentApartamento.data.IDKEY)">
        <ion-icon slot="icon-only" name="trash"></ion-icon>
      </ion-item-option>
    </ion-item-options>
  </ion-item-sliding>
</ion-list>
</ion-content>

```

[Volver al apartado](#)



## 5.9 Registro de usuarios

```

register() {
  const user = {
    email: this.registerForm.controls['mail'].value,
    password: this.registerForm.controls['password'].value
  };
  if (this.registerForm.status === 'VALID') {
    if (user.password === this.registerForm.controls['confirm'].value) {
      this.registro={ EMAIL: '',ROL: 'user',UID: ''}

      this.auth.createUserWithEmailAndPassword(user.email, user.password)
        .then(userData => {

          this.registro={ EMAIL: userData.user?.email!,ROL: 'user',UID: userData.user?.uid!}

          this.alert.registerAlert('Success', 'El usuario a sido creado correctamente');
          //this.db.database.ref('user/' + userData.user.uid).set(this.registerForm.value);

          this.firestoreService.insertar("usuarios", this.registro).then(() => {
            console.log('Tarea creada correctamente!');
            //this.cerraduraEditando = {} as Cerradura;
          }, (error) => {
            console.error(error);
          });
          this.cerrar()
          console.log(userData);
        }).catch(e => {
          console.log(e);
          // eslint-disable-next-line max-len
          this.alert.registerAlert('¡Error!', this.error(e));
        });
    } else {
      this.alert.registerAlert('¡Error!', 'Las contraseñas no coinciden');
    }
  } else {
    this.alert.registerAlert('¡Error!', 'Hay algún campo vacío');
  }
}

```

[Volver al apartado](#)

## 5.10 Código controladora ESP32

Clase main:

```

C:\Users\aylla > Downloads 2 ando_python > main.py
1 import machine, network, time, urequests, ubinascii, micropython, esp, onewire, gc, _thread, bluetooth
2 from umqtt.simple import MQTTClient
3 from machine import Pin, I2C, Signal
4 from bme680 import *
5 from ssd1306 import SSD1306_I2C
6 from BLE import BLEUART
7 #####
8 esp.osdebug(None)
9 gc.collect()
10 pin = Pin(16, Pin.OUT)
11 pin.value(1) # reset de la pantalla a 1 Para activarla
12 #####
13 ssid = 'nombre de la red'
14 password = 'Password'
15 #####
16 ## Mosquito
17 mqtt_server = '192.168.100.5'
18 user_mosquitto = 'User'
19 password_mosquitto = 'pass'
20 #####
21 client_id = ubinascii.hexlify(machine.unique_id()) # id unica del ECP para crear un cliente MQTT
22 ## tópicos a publicar
23 topic_pub_temp = b'esp/bme680/temperature'
24 topic_pub_pres = b'esp/bme680/presion'
25 topic_pub_hume = b'esp/bme680/humedad'
26
27 rx_buffer = ''
28 toggle = False
29 #lock = _thread.allocate_lock
30 station = network.WLAN(network.STA_IF)
31 #Signal(led2_pin, invert=True) ejemplo de signal
32 led25 = Pin(25, Pin.OUT)
33 #####Definición para placa relés arduino
34 led26_pin = Pin(26, Pin.OPEN_DRAIN)
35 led26 = Signal(led26_pin, invert=True)#invierte la lógica 0 enciende 1 apaga
36
37 led27_pin = Pin(27, Pin.OPEN_DRAIN)
38 led27 = Signal(led27_pin, invert=True)
39
40 led14_pin = Pin(14, Pin.OPEN_DRAIN)
41 led14 = Signal(led14_pin, invert=True)
42

```

```

43 led33_pin = Pin(33, Pin.OPEN_DRAIN)
44 led33 = Signal(led33_pin, invert=True)
45 #####
46 led25.off()
47 led26.off()
48 led27.off()
49 led14.off()
50 led33.off()
51 #####
52
53 def pulso1():
54     led26.on()
55     #led25.on()
56     time.sleep(1)
57     led26.off()
58     # led25.off()
59
60 def pulso2():
61     led27.on()
62     time.sleep(1)
63     led27.off()
64
65 def pulso3():
66     led14.on()
67     time.sleep(1)
68     led14.off()
69
70 def pulso4():
71     led33.on()
72     time.sleep(0.6)
73     led33.off()
74 #Init bluetooth
75
76 name = 'ESP32'
77 ble = bluetooth.BLE()
78 uart = BLEUART(ble,name)
79
80 #Bluetooth RX event
81 def on_rx():
82     global rx_buffer
83     rx_buffer=uart.read().decode().strip()
84     uart.write('ESP32 says: ' + str(rx_buffer)+'\n')

```

C:\Users\aytha > Downloads > ardu\_python > main.py

```

85     if (rx_buffer == 'apagar'):
86         led25.off()
87         led33.off()
88     if (rx_buffer == 'encender'):
89         led25.on()
90         led33.on()
91     if (rx_buffer == 'JOPkjh6785re'):
92         _thread.start_new_thread(pulso1, ())
93     if (rx_buffer == 'Liu67AQWERjkl'):
94         _thread.start_new_thread(pulso2, ())
95     if (rx_buffer == 'khdfRTUILOgq12'):
96         _thread.start_new_thread(pulso3, ())
97
98 #Register Bluetooth event
99 uart.irq(handler=on_rx)
100
101 # Conexion a la red wifi
102 def connection_wifi():
103     global station
104     station = network.WLAN(network.STA_IF)
105     station.active(True)
106     if not station.isconnected():
107         station.connect(ssid, password)
108         while station.isconnected() == False:
109             pass
110         print('Connection successful')
111 #####
112 ##### PANTALLA
113 oled = SSD1306_I2C(128, 64, I2C(1, scl=Pin(15), sda=Pin(4))) # I2C pantalla e instancia
114 oled.fill(0)
115 oled.text("Estacion Metereológica", 0, 0)
116 oled.show()
117 i2c = I2C(0, scl=Pin(22), sda=Pin(21)) #I2C sensor
118 bme = BME680_I2C(i2c-i2c) #instancia del sensor
119
120 ultima_peticion =0
121 intervalo_peticiones =60
122 #####
123 def connect_mqtt():
124     global client_id, mqtt_server
125     #client = MQTTClient(client_id, mqtt_server)
126     client = MQTTClient(client_id, mqtt_server, user=user_mosquitto, password=password_mosquitto)

```



```

127     client.connect()
128     print('Connected to %s MQTT broker' % (mqtt_server))
129     return client
130
131     def restart_and_reconnect():
132         global toggle
133         print('Failed to connect to MQTT broker. Reconnecting...')
134         time.sleep(10)
135         toggle = True
136         _thread.exit()
137
138     # coneccion red wifi
139     #_thread.start_new_thread(connection_wifi, ())
140
141     #enviar datos al broker mosquito
142     def datos_mosquitto():
143         #connection_wifi()
144         #conexion al broker mosquito
145         try:
146             client = connect_mqtt()
147         except OSError as e:
148             restart_and_reconnect()
149         last_message = 0 # ultimo mensaje
150         message_interv = 10 # intervalo de tiempo
151         while True:
152             if (time.time() - last_message) > message_interv:
153                 #lock.acquire
154                 temp = str(round(bme.temperature, 2))
155                 hum = str(round(bme.humidity, 2))
156                 pres = str(round(bme.pressure, 2))
157                 #lock.release
158                 client.publish(topic_pub_temp,temp)
159                 client.publish(topic_pub_pres,pres)
160                 client.publish(topic_pub_hume,hum)
161                 last_message = time.time()
162
163     #crear un hilo para enviar datos al servidor mosquitos
164     if station.isconnected():
165         _thread.start_new_thread(datos_mosquitto, ())
166
167     #secuencia
168     while True:
169
170
171         if (time.time() - ultima_peticion) > intervalo_peticiones:
172
173             if (toggle == True):
174                 toggle = False
175                 _thread.start_new_thread(datos_mosquitto, ())
176             #lock.acquire
177             temp = str(round(bme.temperature, 2))
178             hum = str(round(bme.humidity, 2))
179             pres = str(round(bme.pressure, 2))
180             gas = str(round(bme.gas/1000, 2))
181             #lock.release
182             ultima_peticion = time.time()
183             oled.fill(0)
184             oled.text("Estacion Metereológica", 0, 0)
185             oled.text("Temp "+str(temp)+" C", 0, 10)
186             oled.text("Hum "+str(hum)+" %", 0, 20)
187             oled.text("Pres "+str(pres)+" Hpas", 0, 30)
188             oled.text("Gas "+str(gas)+" KOhms", 0, 40)
189             oled.text("BT dice "+str(rx_buffer),0, 50)
190             oled.show()
191
192             #time.sleep(0.1)

```

Clase umqttsimple:

```

1  try:
2      import usocket as socket
3  except:
4      import socket
5  import ustruct as struct
6  from ubinascii import hexlify
7
8  class MQTTException(Exception):
9      pass
10
11  class MQTTClient:
12
13      def __init__(self, client_id, server, port=0, user=None, password=None, keepalive=0,
14                  ssl=False, ssl_params={}):
15          if port == 0:
16              port = 8883 if ssl else 1883
17          self.client_id = client_id
18          self.sock = None
19          self.server = server
20          self.port = port
21          self.ssl = ssl
22          self.ssl_params = ssl_params
23          self.pid = 0
24          self.cb = None
25          self.user = user
26          self.pswd = password
27          self.keepalive = keepalive
28          self.lw_topic = None
29          self.lw_msg = None
30          self.lw_qos = 0
31          self.lw_retain = False
32
33      def _send_str(self, s):
34          self.sock.write(struct.pack("!H", len(s)))
35          self.sock.write(s)
36
37      def _recv_len(self):
38          n = 0
39          sh = 0
40          while 1:
41              b = self.sock.read(1)[0]
42              n |= (b & 0x7f) << sh
43              if not b & 0x80:
44                  return n
45              sh += 7
46
47      def set_callback(self, f):
48          self.cb = f
49
50      def set_last_will(self, topic, msg, retain=False, qos=0):
51          assert 0 <= qos <= 2
52          assert topic
53          self.lw_topic = topic
54          self.lw_msg = msg
55          self.lw_qos = qos
56          self.lw_retain = retain
57
58      def connect(self, clean_session=True):
59          self.sock = socket.socket()
60          addr = socket.getaddrinfo(self.server, self.port)[0][-1]
61          self.sock.connect(addr)
62          if self.ssl:
63              import ssl
64              self.sock = ssl.wrap_socket(self.sock, **self.ssl_params)
65          premsg = bytearray(b"\x10\0\0\0\0\0")
66          msg = bytearray(b"\x04MQTT\x04\x02\0\0")
67
68          sz = 10 + 2 + len(self.client_id)
69          msg[6] = clean_session << 1
70          if self.user is not None:
71              sz += 2 + len(self.user) + 2 + len(self.pswd)
72              msg[6] |= 0xC0
73          if self.keepalive:
74              assert self.keepalive < 65536
75              msg[7] |= self.keepalive >> 8
76              msg[8] |= self.keepalive & 0x00FF

```

```

77         if self.lw_topic:
78             sz += 2 + len(self.lw_topic) + 2 + len(self.lw_msg)
79             msg[6] |= 0x4 | (self.lw_qos & 0x1) << 3 | (self.lw_qos & 0x2) << 3
80             msg[6] |= self.lw_retain << 5
81
82         i = 1
83         while sz > 0x7f:
84             premsg[i] = (sz & 0x7f) | 0x80
85             sz >>= 7
86             i += 1
87         premsg[i] = sz
88
89         self.sock.write(premsg, i + 2)
90         self.sock.write(msg)
91         #print(hex(len(msg)), hexlify(msg, ":"))
92         self._send_str(self.client_id)
93         if self.lw_topic:
94             self._send_str(self.lw_topic)
95             self._send_str(self.lw_msg)
96         if self.user is not None:
97             self._send_str(self.user)
98             self._send_str(self.pswd)
99         resp = self.sock.read(4)
100         assert resp[0] == 0x20 and resp[1] == 0x02
101         if resp[3] != 0:
102             raise MQTTException(resp[3])
103         return resp[2] & 1
104
105     def disconnect(self):
106         self.sock.write(b"\xe0\0")
107         self.sock.close()
108
109     def ping(self):
110         self.sock.write(b"\xc0\0")
111
112     def publish(self, topic, msg, retain=False, qos=0):
113         pkt = bytearray(b"\x30\0\0\0")
114         pkt[0] |= qos << 1 | retain
115         sz = 2 + len(topic) + len(msg)
116         if qos > 0:
117             sz += 2
118         assert sz < 2097152
119
120         i = 1
121         while sz > 0x7f:
122             pkt[i] = (sz & 0x7f) | 0x80
123             sz >>= 7
124             i += 1
125         pkt[i] = sz
126         #print(hex(len(pkt)), hexlify(pkt, ":"))
127         self.sock.write(pkt, i + 1)
128         self._send_str(topic)
129         if qos > 0:
130             self.pid += 1
131             pid = self.pid
132             struct.pack_into("!H", pkt, 0, pid)
133             self.sock.write(pkt, 2)
134         self.sock.write(msg)
135         if qos == 1:
136             while 1:
137                 op = self.wait_msg()
138                 if op == 0x40:
139                     sz = self.sock.read(1)
140                     assert sz == b"\x02"
141                     rcv_pid = self.sock.read(2)
142                     rcv_pid = rcv_pid[0] << 8 | rcv_pid[1]
143                     if pid == rcv_pid:
144                         return
145             elif qos == 2:
146                 assert 0
147
148     def subscribe(self, topic, qos=0):
149         assert self.cb is not None, "Subscribe callback is not set"
150         pkt = bytearray(b"\x82\0\0\0")
151         self.pid += 1
152         struct.pack_into("!BH", pkt, 1, 2 + 2 + len(topic) + 1, self.pid)
153         #print(hex(len(pkt)), hexlify(pkt, ":"))
154         self.sock.write(pkt)
155         self._send_str(topic)
156         self.sock.write(qos.to_bytes(1, "little"))
157         while 1:
158             op = self.wait_msg()
159             if op == 0x90:
160                 resp = self.sock.read(4)
161                 #print(resp)

```

```

C:\> Users > aytha > Downloads > ardu_python > umqtsimple.py
161         assert resp[1] == pkt[2] and resp[2] == pkt[3]
162         if resp[3] == 0x80:
163             raise MQTTException(resp[3])
164         return
165
166     # Wait for a single incoming MQTT message and process it.
167     # Subscribed messages are delivered to a callback previously
168     # set by .set_callback() method. Other (internal) MQTT
169     # messages processed internally.
170     def wait_msg(self):
171         res = self.sock.read(1)
172         self.sock.setblocking(True)
173         if res is None:
174             return None
175         if res == b"":
176             raise OSError(-1)
177         if res == b"\xd0": # PINGRESP
178             sz = self.sock.read(1)[0]
179             assert sz == 0
180             return None
181         op = res[0]
182         if op & 0xf0 != 0x30:
183             return op
184         sz = self._recv_len()
185         topic_len = self.sock.read(2)
186         topic_len = (topic_len[0] << 8) | topic_len[1]
187         topic = self.sock.read(topic_len)
188         sz -= topic_len + 2
189         if op & 6:
190             pid = self.sock.read(2)
191             pid = pid[0] << 8 | pid[1]
192             sz -= 2
193         msg = self.sock.read(sz)
194         self.cb(topic, msg)
195         if op & 6 == 2:
196             pkt = bytearray(b"\x40\x02\0\0")
197             struct.pack_into("IH", pkt, 2, pid)
198             self.sock.write(pkt)
199         elif op & 6 == 4:
200             assert 0
201

```

Clase BLE:

```

1  # Proof-of-concept of a REPL over BLE UART.
2  #
3  # Tested with the Adafruit Bluefruit app on Android.
4  # Set the EoL characters to \r\n.
5
6  import bluetooth
7  import io
8  import os
9  import micropython
10 import machine
11 from micropython import const
12 import struct
13 import time
14 from machine import Pin
15
16 pbt = Pin(2, Pin.OUT)
17 pbt.off()
18
19 _IRQ_CENTRAL_CONNECT = const(1)
20 _IRQ_CENTRAL_DISCONNECT = const(2)
21 _IRQ_GATTS_WRITE = const(3)
22
23 _FLAG_WRITE = const(0x0008)
24 _FLAG_NOTIFY = const(0x0010)
25
26 _UART_UUID = bluetooth.UUID("6E400001-B5A3-F393-E0A9-E50E24DCCA9E")
27 _UART_TX = (
28     bluetooth.UUID("6E400003-B5A3-F393-E0A9-E50E24DCCA9E"),
29     _FLAG_NOTIFY,
30 )
31 _UART_RX = (
32     bluetooth.UUID("6E400002-B5A3-F393-E0A9-E50E24DCCA9E"),
33     _FLAG_WRITE,
34 )
35 _UART_SERVICE = (
36     _UART_UUID,
37     (_UART_TX, _UART_RX),
38 )
39
40 _ADV_APPEARANCE_GENERIC_COMPUTER = const(128)
41

```



```

42
43 class BLEUART:
44     def __init__(self, ble, name, rxbuf=1000):
45         self.ble = ble
46         self.ble.active(True)
47         self.ble.irq(self._irq)
48         ((self.tx_handle, self.rx_handle),
49          ) = self.ble.gatts_register_services((_UART_SERVICE,))
50         # Increase the size of the rx buffer and enable append mode.
51         self.ble.gatts_set_buffer(self.rx_handle, rxbuf, True)
52         self._connections = set()
53         self._rx_buffer = bytearray()
54         self._handler = None
55         # Optionally add services=[_UART_UUID], but this is likely to make the payload too large.
56         self._payload = advertising_payload(
57             name=name, appearance=_ADV_APPEARANCE_GENERIC_COMPUTER)
58         self._advertise()
59
60     def irq(self, handler):
61         self._handler = handler
62
63     def _irq(self, event, data):
64         # Track connections so we can send notifications.
65         if event == _IRQ_CENTRAL_CONNECT:
66             conn_handle, _, _ = data
67             print("_IRQ_CENTRAL_CONNECT")
68             pbt.on()
69             self._connections.add(conn_handle)
70         elif event == _IRQ_CENTRAL_DISCONNECT:
71             conn_handle, _, _ = data
72             pbt.off()
73             print('_IRQ_CENTRAL_DISCONNECT')
74             if conn_handle in self._connections:
75                 self._connections.remove(conn_handle)
76             # Start advertising again to allow a new connection.
77             self._advertise()
78         elif event == _IRQ_GATTS_WRITE:
79             conn_handle, value_handle = data
80             if conn_handle in self._connections and value_handle == self._rx_handle:
81                 self._rx_buffer += self.ble.gatts_read(self._rx_handle)
82                 if self._handler:
83                     self._handler()
84
85     def any(self):
86         return len(self._rx_buffer)
87
88     def read(self, sz=None):
89         if not sz:
90             sz = len(self._rx_buffer)
91         result = self._rx_buffer[0:sz]
92         self._rx_buffer = self._rx_buffer[sz:]
93         return result
94
95     def write(self, data):
96         for conn_handle in self._connections:
97             self.ble.gatts_notify(conn_handle, self.tx_handle, data)
98
99     def close(self):
100         for conn_handle in self._connections:
101             self.ble.gap_disconnect(conn_handle)
102         self._connections.clear()
103
104     def _advertise(self, interval_us=500000):
105         self.ble.gap_advertise(interval_us, adv_data=self._payload)
106
107
108 # Advertising payloads are repeated packets of the following form:
109 #   1 byte data length (N + 1)
110 #   1 byte type (see constants below)
111 #   N bytes type-specific data
112
113 _ADV_TYPE_FLAGS = const(0x01)
114 _ADV_TYPE_NAME = const(0x09)
115 _ADV_TYPE_UUID16_COMPLETE = const(0x3)
116 _ADV_TYPE_UUID32_COMPLETE = const(0x5)
117 _ADV_TYPE_UUID128_COMPLETE = const(0x7)
118 _ADV_TYPE_UUID16_MORE = const(0x2)
119 _ADV_TYPE_UUID32_MORE = const(0x4)
120 _ADV_TYPE_UUID128_MORE = const(0x6)
121 _ADV_TYPE_APPEARANCE = const(0x19)
122
123

```

```

124 # Generate a payload to be passed to gap_advertise(adv_data=...).
125 def advertising_payload(limited_disc=False, br_edr=False, name=None, services=None, appearance=0):
126     payload = bytearray()
127
128     def _append(adv_type, value):
129         nonlocal payload
130         payload += struct.pack("BB", len(value) + 1, adv_type) + value
131
132     _append(
133         _ADV_TYPE_FLAGS,
134         struct.pack("B", (0x01 if limited_disc else 0x02) +
135                     (0x18 if br_edr else 0x04)),
136     )
137
138     if name:
139         _append(_ADV_TYPE_NAME, name)
140
141     if services:
142         for uuid in services:
143             b = bytes(uuid)
144             if len(b) == 2:
145                 _append(_ADV_TYPE_UUID16_COMPLETE, b)
146             elif len(b) == 4:
147                 _append(_ADV_TYPE_UUID32_COMPLETE, b)
148             elif len(b) == 16:
149                 _append(_ADV_TYPE_UUID128_COMPLETE, b)
150
151     # See org.bluetooth.characteristic.gap.appearance.xml
152     if appearance:
153         _append(_ADV_TYPE_APPEARANCE, struct.pack("<h", appearance))
154
155     return payload
156
157
158 def demo():
159     print("demo")
160
161
162 if __name__ == "__main__":
163     demo()
164

```

[Volver al apartado](#)