

MAESTRÍA EN SISTEMAS EMBEBIDOS

MEMORIA DEL TRABAJO FINAL

Control de acceso de fácil instalación con administración remota

Autor:
Esp. Lic. Leopoldo A. Zimperz

Director:
Mg. Ing. Gonzalo E. Sanchez (FIUBA)

Jurados:
Mg. Lic. Santiago Germino (FIUBA)
Mg. Ing. Franco Bucafusco (FIUBA)
Mg. Ing. Pablo Slavkin (FIUBA)

*Este trabajo fue realizado en la Ciudad Autónoma de Buenos Aires,
entre junio de 2020 y mayo de 2021.*

Resumen

La presente memoria describe el desarrollo realizado para la empresa Iris Tecnología S.R.L, en el que se implementa un prototipo funcional de un sistema de control de acceso modular. Se trata de un dispositivo capaz de ser administrado en forma remota, destinado a casas, edificios, hoteles y oficinas. En el proyecto se pretende integrar distintas formas de conectividad y una interfaz de usuario que conformen un producto altamente competitivo.

Para llevar a cabo el trabajo fueron de suma importancia los conocimientos adquiridos a lo largo de la carrera, tanto en aspectos organizativos como técnicos. Durante la planificación se han utilizado diversas herramientas, como el modelo Canvas para conocer e identificar aspectos claves del negocio asociado al proyecto. En la etapa de implementación se aplicaron conocimientos relacionados a problemas de sincronización y plataformas de Internet de las cosas. Se logró también evaluar y mejorar la fiabilidad del sistema empleando técnicas de análisis de riesgos y conceptos básicos de sistemas críticos profundizados en la materia.

Agradecimientos

A Guillermina, mi pareja, por haberme ayudado con sus minuciosas revisiones poniendo “toda su paciencia”.

A mi hermano Alfredo por haber insistido acertadamente para que me inscribiera en la carrera y por haber ayudado a que le dedique el tiempo necesario.

A Gonzalo Sanchez, el director del trabajo, por su predisposición al haber aceptado esta responsabilidad.

A Ariel Lutemberg, docente de la asignatura Taller de trabajo final, por estar atento a todas las necesidades y observaciones de los alumnos del posgrado.

A todos los profesores y a quienes organizan la Maestría en Sistemas Embebidos.

Índice general

Resumen	I
1. Introducción General	1
1.1. Sistemas de control de acceso a edificios	1
1.1.1. Conceptos básicos y componentes	2
1.1.2. Factores de autenticación	3
1.1.3. Clasificación	4
1.1.4. Estudio de mercado	5
1.2. Sistema desarrollado	6
1.3. Objetivos	8
1.4. Motivación	8
1.5. Alcance	8
1.6. Normativa aplicable	9
1.7. Requerimientos	10
1.7.1. Generales	10
1.7.2. Sobre arquitectura	10
1.7.3. Específicos	11
2. Introducción específica	15
2.1. Tarjetas de proximidad	15
2.2. Kit de desarrollo ESP32-DevKitC	16
2.2.1. Características principales del módulo ESP32	17
2.2.2. Plataforma de desarrollo ESP-IDF	18
2.3. Módulo GSM/GPRS SIM800C	19
2.4. Módulo Bluetooth Low Energy RN4870	20
2.5. Módulo lector/grabador MRFC-522	22
2.6. Microcontrolador PIC16F1718	23
2.7. Detector de proximidad	23
2.8. Protocolos destacados en el proyecto	24
2.8.1. Protocolo MQTT	24
2.8.2. Protocolo PPPoS	25
2.9. Plataformas en la nube	26
2.9.1. Google Cloud Platform	27
2.9.2. Microsoft Azure Cloud	27
2.10. Aplicación celular e interfaz de usuario	28
2.10.1. Framework Flutter	28
2.10.2. Lenguaje Dart	29
3. Diseño e implementación	31
3.1. Descripción del funcionamiento general del sistema	31
3.1.1. Lector inalámbrico de tarjetas RFID	33
3.1.2. Dispositivo central del sistema de control de acceso	34
3.1.3. Infraestructura en la nube (<i>backend</i>)	38

3.1.4. Aplicación celular (<i>frontend</i>)	38
3.2. Hardware	39
3.2.1. Integración de componentes del dispositivo central	39
3.2.2. Integración de componentes del dispositivo lector	42
3.3. Firmware	45
3.3.1. Firmware del controlador central	47
3.3.2. Firmware del dispositivo lector	56
3.4. Software	59
3.4.1. Interfaz de programación de aplicaciones (API <i>backend</i>) . . .	59
3.4.2. Implementación de la base de datos	62
3.4.3. Aplicación para teléfono celular (<i>frontend</i>)	62
4. Ensayos y resultados	67
4.1. Testeos unitarios	67
4.2. Testeos funcionales y pruebas de sistema	69
4.2.1. Ensayos sobre firmware y hardware	72
4.2.2. Ensayos sobre APIs (<i>backend</i>)	82
4.2.3. Ensayos sobre APP celular (<i>frontend</i>)	84
4.2.4. Pruebas de sistema	86
4.2.5. Comparación de resultados con otras soluciones	88
5. Conclusiones	89
5.1. Resultados obtenidos	89
5.2. Próximos pasos	90
Bibliografía	93

Índice de figuras

1.1.	Diagrama en bloques simplificado del sistema	7
2.1.	Kit de desarrollo ESP32-DevKitC V1.	17
2.2.	Módulo ESP32-WROOM-32.	17
2.3.	Módulo GSM/GPRS SIM800	19
2.4.	Distribución de pines del módulo SIM800	20
2.5.	Módulo BLE RN4870 de Microchip	21
2.6.	Distribución de pines del módulo RFID-RC522	22
2.7.	Módulo detector de movimiento PIR HC-SR501	24
3.1.	Diagrama en bloques del sistema y flujo de información	32
3.2.	Diagrama en bloques del dispositivo central	39
3.3.	Disposición de pines del kit ESP32-DevKitC	40
3.4.	Prototipo del dispositivo central en etapa 1	41
3.5.	Prototipo final del dispositivo central	42
3.6.	Diagrama en bloques del dispositivo lector	42
3.7.	Disposición de pines del microcontrolador PIC16F1718	43
3.8.	Prototipo del dispositivo lector en etapa 1	44
3.9.	Prototipo lector RFID montado en placa experimental.	45
3.10.	Prototipo lector ensamblado	45
3.11.	Ciclo de vida en espiral	46
3.12.	Arquitectura en capas usando ESP-IDF y FreeRTOS	48
3.13.	Administrador de dispositivos IoT de Google	51
3.14.	Proceso de aprovisionamiento de credenciales en GCP	53
3.15.	Diagrama de flujo del dispositivo lector	56
3.16.	Matriz para configuración de pines en MPLAB-X	57
3.17.	Asignación de pines del PIC16F1718	57
3.18.	Diagrama entidad-relación	62
3.19.	Diagrama de árbol de layout de UI	63
3.20.	Capturas de pantalla de la aplicación celular.	66
4.1.	Ejecución automática de test en módulo shared-lock.c	68
4.2.	Ejecución interactiva de test en módulo shared-lock.c	69
4.3.	Esquemas de bancos de prueba	71
4.4.	Banco de pruebas para testeos 1.x	72
4.5.	Test lectura tarjeta MIFARE vacía	72
4.6.	Test lectura tarjeta MIFARE bloqueada	73
4.7.	Test lectura tarjeta MIFARE válida	73
4.8.	Captura de pantalla de terminal VS-Code en testeos 2.x	74
4.9.	Capturas de la aplicación celular BLE Scanner.	75
4.10.	Módulo RN4870 conectado a la PC por interfaz serial	75
4.11.	Terminal serial conectada al módulo RN4870	77
4.12.	Captura de pantalla de terminal VS-Code en testeos 3.x	77

4.13. Banco de pruebas para subsistema GSM	78
4.14. Conexión a Internet con el módulo SIM800C por comandos AT	78
4.15. Datos del prestador celular y el módem GSM	79
4.16. Direcciones IP relacionadas a la conexión GSM	79
4.17. Secuencia de conexión a Google Cloud IoT en el módulo ESP32	80
4.18. Vista de conexión en administrador de dispositivos IoT Core	81
4.19. Recepción de comando vía MQTT en el módulo ESP32	81
4.20. Envío de comando desde administrador IoT Core	82
4.21. Colección de testeos en la aplicación Postman	83
4.22. Resultados de los testeos realizados a las APIs	84
4.23. Simulador Android intentando acceder a la aplicación	85
4.24. <i>Debug</i> de la aplicación celular en Android Studio	85
4.25. Sistema de control de accesos configurado para pruebas	86

Índice de tablas

1.1. Características según autonomía del sistema	6
1.2. Características principales del sistema	7
2.1. Diferencias entre chips EM4200 y MIFARE	16
2.2. Módulos BLE analizados	20
2.3. Características del microcontrolador PIC16F1718	23
3.1. Funciones principales del lector de tarjetas	34
3.2. Comandos aceptados por el dispositivo central	36
3.3. Indicadores LED y buzzer	37
3.4. Eventos enviados por el dispositivo central	37
3.5. Uso de pines del kit ESP32-DevKitC	41
3.6. Uso de pines del microcontrolador PIC16F1718	44
3.7. Método de programación por dispositivo	47
3.8. Tareas del controlador central	48
3.9. Módulos de software del dispositivo central	49
3.10. Comandos AT para conexión por PPPoS	55
3.11. Módulos de software del dispositivo central	58
3.12. Modelos definidos dentro de la API del <i>backend</i>	60
3.13. <i>Endpoints</i> incluidos en las APIs del <i>backend</i>	61
3.14. Módulos que componen la aplicación del <i>frontend</i>	65
4.1. Organización de los ensayos realizados	70
4.2. Comandos AT para testeos en módulo RN4870	76
4.3. Pruebas de sistema	87
4.4. Comparativa del prototipo y productos comerciales	88

Dedicado a mi madre, Liliana

Capítulo 1

Introducción General

En el presente capítulo se realiza una introducción a los sistemas de control de acceso, se describen los elementos que los componen y sus prestaciones. Se exponen también los motivos por los cuales se realizó el proyecto, detallando sus objetivos, alcance y requerimientos.

1.1. Sistemas de control de acceso a edificios

Un control de acceso es un sistema automatizado que permite, de forma eficaz, aprobar o negar el paso de personas a zonas restringidas en función de ciertos parámetros de seguridad establecidos por una empresa, comercio, edificio o incluso propietarios de una vivienda. Para bloquear o no el paso se emplea algún dispositivo mecánico como electrocerraduras, barreras o puertas controladas que se accionan según el resultado de una evaluación de identificación y permisos.

La identificación de los usuarios se puede realizar por distintos medios como tarjetas de proximidad, contraseñas o reconocimiento de parámetros biométricos, que se abordarán en la sección 1.1.3. Estos métodos pueden ser combinados para aumentar el nivel de seguridad, dando lugar a los sistemas de autenticación multifactor.

Los sistemas de control de accesos, además de ofrecer seguridad, pueden aportar funciones adicionales como control de tiempos y de aforos, registro de eventos o localización de personas. Pueden clasificarse teniendo en cuenta distintos aspectos, pero en este trabajo se dividirán según los métodos de identificación y en virtud de su autonomía o su necesidad de formar parte de un gran sistema de acceso en red.

En función de las necesidades, los costos y el presupuesto disponible, se suele escoger entre sistemas con algunas de las características mencionadas anteriormente u otros que combinan a varias de ellas.

1.1.1. Conceptos básicos y componentes

Se definen a continuación los conceptos y elementos que componen un sistema de control de accesos [1]:

- **Usuarios:** cada uno de los individuos que se va a habilitar en el sistema. Puede ser desde uno a varios miles.
- **Identificadores:** cada uno de los elementos o medios que identifican a los usuarios en el sistema. Deben de ser únicos para cada usuario, aunque un usuario puede tener distintos identificadores para distintos tipos de acceso, por ejemplo: tarjetas de proximidad, huella, código de acceso, etc.
- **Lectores:** son los elementos electrónicos ubicados en cada uno de los accesos y se encargan de leer la información de los identificadores.
- **Accesos o puertas:** son los puntos controlados por los que se permite pasar a las zonas protegidas. Cada acceso tiene un lector en el que el usuario se identifica con su identificador. El lector interroga al sistema si el usuario tiene permiso para acceder a esa zona en ese horario y, en caso afirmativo, activa el mecanismo de acceso correspondiente. Puede haber accesos peatonales o vehiculares.
- **Perfiles de usuarios:** en sistemas centralizados con un número elevado de usuarios, los perfiles permiten agrupar a los usuarios para facilitar la gestión de sus permisos. Se pueden crear, por ejemplo, los perfiles de producción, administración, limpieza, mantenimiento, gerencia, etc. A cada uno de estos perfiles se le asignan los permisos correspondientes y al gestionar usuarios sólo tenemos que asignarle un perfil para que adquiera los permisos del mismo.
- **Zonas:** son las áreas protegidas. El acceso a ellas se puede realizar a través de una o varias puertas. Cada puerta sólo da acceso a una zona, pero puede que para llegar a algunas zonas haya que pasar por varias puertas.
- **Horarios:** son los períodos de tiempo en los que se va a permitir el acceso de determinados usuarios o perfiles de usuario a determinadas zonas. Se pueden definir varios períodos por día, días festivos, vacaciones, jornadas, etc.
- **Permisos:** se definen para cada perfil de usuarios y son la unión entre las zonas y los horarios. Dicho de otro modo, se permite a determinado perfil el acceso a determinadas zonas en determinados horarios.

1.1.2. Factores de autenticación

En un proceso de identificación, sea físico o digital, intervienen al menos dos entidades o personas cuya identidad debe ser verificada. Esta comprobación puede llevarse a cabo utilizando uno o varios factores de autenticación. Si la autenticación utiliza más de un método, se dice que es multifactor y si las dos entidades intervenientes garantizan una a la otra su identidad, se dice que es recíproca.

La Unión Internacional de Telecomunicaciones (UIT) es el organismo especializado de las Naciones Unidas en las tecnologías de la información y la comunicación. Regula las telecomunicaciones a nivel internacional entre las distintas administraciones y empresas operadoras. Si bien las publicaciones y recomendaciones de la UIT no están destinadas al ámbito de los controles de acceso, algunas aplican perfectamente y no solo al área que involucra tecnologías de la información o comunicaciones. En el caso de la recomendación ITU-T X.1254 [2], se abordan temas como los factores de autenticación, que son empleados a lo largo de todo el ciclo de vida del proceso de identificación.

Los factores de autenticación se dividen en cuatro categorías que se detallan a continuación:

- Algo que la entidad tiene (por ejemplo, firma de dispositivo, pasaporte, dispositivo físico que contiene una credencial o clave privada).
- Algo que la entidad sabe (por ejemplo, contraseña, PIN).
- Algo intrínseco a la identidad (por ejemplo, características biométricas).
- Algo que la entidad suele hacer (por ejemplo, patrón de conducta).

Cabe mencionar que una de las mayores tendencias en los últimos años es el control de acceso móvil, el cual consiste en usar un dispositivo móvil (un teléfono celular, una tableta o un reloj inteligente) para acceder a puertas, compuertas, redes, servicios y otros espacios de acceso restringido.

Además del uso de dispositivos móviles, que por su naturaleza piden al menos un factor de autenticación, esta modalidad representa el compromiso por aportar comodidad y conveniencia de los usuarios, pilar fundamental en los sistemas de control de acceso actuales. [3].

Es importante aclarar que la aplicación que soporte una credencial móvil, debe contar con seguridad criptográfica u otro mecanismo que evite que una identificación pueda ser transferida a otro dispositivo.

1.1.3. Clasificación

A continuación se describen posibles clasificaciones para los sistemas de control de acceso en base a distintos criterios.

Por su grado de autonomía:

- Sistema totalmente autónomo: se trata de un control de acceso básico de un solo lector que controla un único acceso o puerta. Toda la gestión de usuarios se hace en el propio lector mediante un identificador de administrador. En algunos casos también es posible conectar el lector a un ordenador para facilitar la gestión y obtener un registro básico de incidencias. Existen lectores autónomos que utilizan distintos métodos de identificación, y en general, son muy sencillos de instalar.
- Sistemas centralizados y en red: en este tipo de equipos los lectores se conectan mediante un bus de datos cableado a una o varias unidades centrales que son las que almacenan toda la información de la instalación, tanto de configuración como de registro de incidencias. Son sistemas escalables en los que se pueden gestionar miles de usuarios y accesos, y trabajar con los conceptos avanzados de perfiles, zonas, horarios y permisos. La gestión se realiza desde uno o varios ordenadores trabajando en arquitectura cliente-servidor. En los sistemas centralizados, los accesos se pueden integrar en placas de portero electrónico o videoportero y completar la instalación con una o varias centrales de conserjería, permitiendo así la comunicación directa con los puestos de control para solucionar posibles incidencias.
- Sistema gestionado: en estos equipos se dispone de una aplicación de software desde la que se permite gestionar altas y bajas de usuarios, horarios y permisos. Además, resulta posible realizar una monitorización del sistema, con una auditoría completa de quien ha accedido, por donde y a qué hora. Estas funcionalidades parecen combinar las ventajas de las otras dos categorías a un costo intermedio, pero suelen presentar limitaciones en su escalabilidad y su instalación incluye mayores requerimientos específicos.

Por los métodos de identificación:

- Teclado: la identificación se realiza por medio de un código numérico de entre 4 y 6 dígitos. El código puede ser distinto para cada usuario, aunque varios usuarios podrían tener el mismo código. Es un sistema de identificación económico ya que no requiere identificadores. Existe la posibilidad de que cada usuario pueda cambiar su código cuando quiera.
- Proximidad: la identificación se realiza por medio de una tarjeta o llavero con una numeración única que transmite por radiofrecuencia al lector cuando se acerca al mismo. Las tarjetas no requieren mantenimiento (cambio de batería) y tienen un bajo costo, se gestionan en el propio sistema y pueden darse de baja en caso de robo o extravío.

- Radiofrecuencia (RF): el identificador es un mando a distancia de radiofrecuencia que transmite el código de identificación al pulsar un botón. Tienen un alcance de varias decenas de metros. Los mandos de RF tienen un costo superior al que presentan las tarjetas de proximidad y necesitan baterías para funcionar. Estos identificadores son especialmente indicados para accesos vehiculares.
- Bluetooth: permite el uso de teléfonos móviles como dispositivos de identificación. Pueden funcionar de forma automática de modo que al aproximarnos al acceso, si el móvil está dado de alta en el sistema, se abra la puerta automáticamente. También se puede configurar para que se requiera la pulsación de una tecla o introducción de un código numérico con el teclado del móvil.
- Biométrico: La identificación se produce mediante la lectura de datos físicos individuales que imposibilitan la suplantación al ser intransferibles, por lo que se consideran los sistemas más seguros. Su empleo implica el cumplimiento de normativas en materia de protección de datos y no están permitidos en todas las empresas. Dentro de este grupo se incluyen los métodos de reconocimiento facial y huella dactilar.

1.1.4. Estudio de mercado

Antes de implementar un sistema de control de acceso, es necesario reflexionar sobre las necesidades y el costo de la inversión. En temas de tecnología, cuando se desconocen los riesgos, es muy probable que se termine tomando una decisión muy influenciada por el factor precio. Teniendo en cuenta lo mencionado, se realizó un análisis de diferentes productos disponibles en el mercado, dentro de un amplio rango de presupuestos, para lograr incluir dentro del estudio soluciones que van desde las más simples basadas en pequeños sistemas autónomos, hasta equipos centralizados complejos, abordando en forma bastante completa a las distintas clases de controles de acceso tratadas en la sección 1.1.3.

En el análisis realizado se detectaron ventajas y desventajas para cada una de las distintas tecnologías observadas. Se hizo foco en las desventajas con el fin de mitigarlas y aprovecharlas para lograr un mejor producto. Las mayores ventajas se presentaron en equipos costosos, validando la teoría de que la calidad, seguridad, escalabilidad y prestaciones del sistema en la actualidad aún dependen en forma casi directa de la inversión requerida. En este sentido, se buscó comprender el motivo por el cual no se logran trasladar dichos beneficios a dispositivos económicos, mientras que se cree que existe la tecnología suficiente para hacerlo posible.

En la tabla 1.1 se exponen las características más relevantes que se desprendieron de la observación llevada a cabo y se relacionan con las distintas clases de equipamiento según su grado de autonomía.

TABLA 1.1. Análisis de características relevantes del sistema según su grado de autonomía.

Característica	Autónomo	Gestionado	En red
Conectividad Wi-Fi	No	Sí	Sí
Conectividad BLE	Sí	No	No
Conectividad GPRS	No	No	No
Autenticación multifactor	Sí	No	No
Apertura por teclado	Sí	No	No
Apertura por Wi-Fi	No	Sí	Sí
Apertura por BLE	Sí	No	No
Apertura por RFID	Sí	Sí	Sí
Apertura por huella	Sí	Sí	No
Apertura sin intervención	No	No	No
Aplicación celular	Sí	Sí	No
Aplicación web	No	No	Sí
Aplicación de escritorio	No	No	Sí
Posibilidad de escalabilidad	Nula	Baja	Alta
Grado de seguridad	Bajo	Alto	Alto
Posibilidad de integración	No	No	No
Necesidad de tarjetas o tags	No	Sí	Sí
Gestión remota integral	No	No	No
Registro de eventos completo	No	Sí	Sí
Configuración por horarios	No	Sí	Sí
Funciones de geolocalización	No	No	No
Auto-actualización de firmware	No	No	No

En el capítulo 4, sección 4.2.5, se muestra un análisis análogo en el que se comparan los resultados obtenidos en el prototipo resultante del presente trabajo, con tres soluciones específicas disponibles en el mercado.

1.2. Sistema desarrollado

El desarrollo propuesto por Iris Tecnología S.R.L. radica en un sistema de control de acceso para personas que reúna las mejores características de los equipos disponibles en el mercado y que a su vez sea de bajo costo. El dispositivo planteado, permite ser instalado por personas que no se encuentren especialmente capacitadas o que carezcan de conocimientos técnicos específicos. Además, posibilita ser administrado y gestionado por usuarios finales o administraciones de consorcio sin ningún tipo de entrenamiento.

Se considera que existe un vacío y una oportunidad de negocio dentro de las gamas iniciales de equipos, de costo bajo o medio. Se pretende obtener una solución dentro del rango mencionado, que integre características de los grandes sistemas y que permita escalabilidad. Iris Tecnología S.R.L. tiene la convicción de que se necesita un cambio de enfoque en cuanto a la infraestructura típica destinada a

controles de acceso, ya que no tiene que ser necesaria una gran cantidad de costoso hardware para que un sistema cuente con buenas prestaciones y posibilidades de escalabilidad.

Por otra parte, se busca lograr la transformación de un negocio en el que habitualmente solo se comercializa hardware, por otro en el que además se generen ingresos por servicios a través de abonos mensuales.

En la tabla 2.3 se detallan las características principales del presente desarrollo y en la figura 1.1 se puede ver un diagrama en bloques simplificado del sistema.

TABLA 1.2. Características principales del sistema desarrollado.

Característica	Valor / Detalle
Costo	Bajo / medio
Conectividad	Wi-Fi / BLE / GSM
Interfaz de usuario	Aplicación celular
Métodos de identificación	RFID / BLE
Tipo de cerraduras admitidas	Eléctrica / electromecánica
Conexión del lector RFID	Inalámbrica
Forma de gestión y administración	Remota

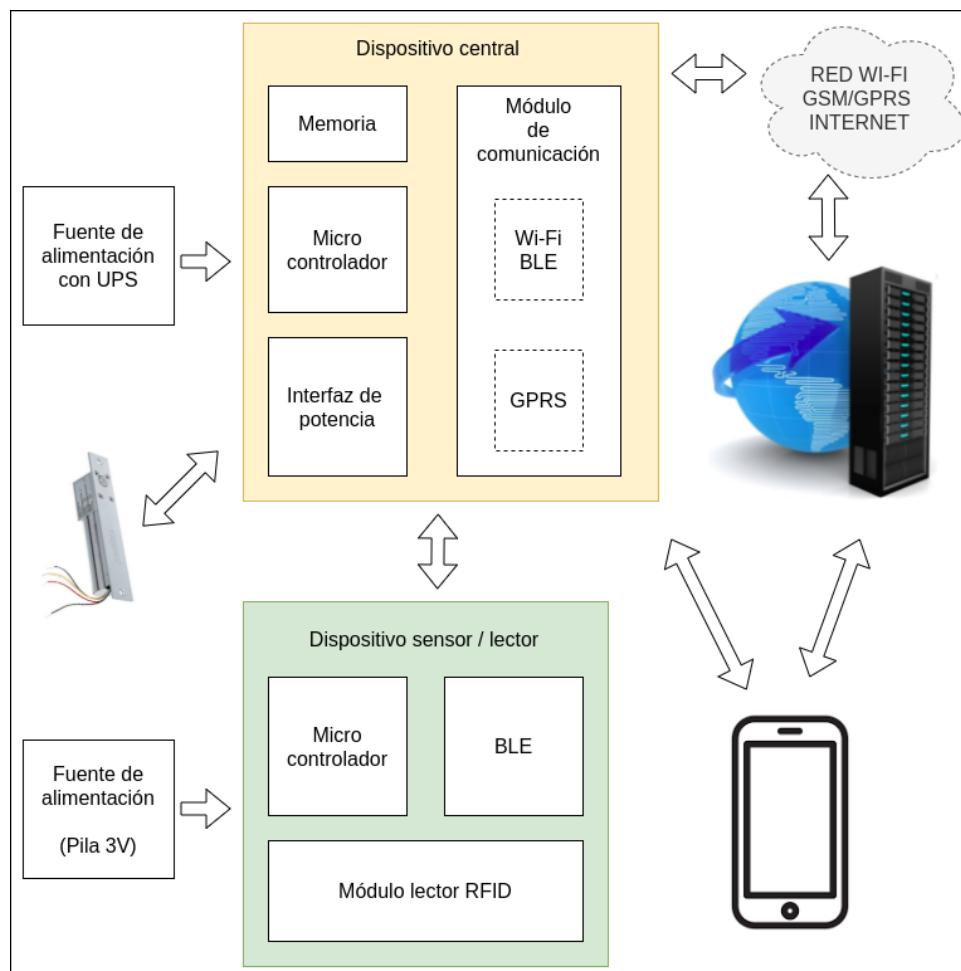


FIGURA 1.1. Diagrama en bloques simplificado del sistema.

1.3. Objetivos

El propósito de este proyecto es desarrollar el prototipo funcional de un sistema de control de acceso para que sea utilizado como punto de partida de una futura implementación con fines comerciales. Se pretende que reúna la mayor cantidad de ventajas destacadas de los diferentes productos analizados en el estudio de mercado.

1.4. Motivación

Las necesidades y soluciones relacionadas con la seguridad y controles de acceso a edificios continúan evolucionando. En la actualidad no existen empresas nacionales que aborden esta materia ofreciendo soluciones completas y a precios accesibles, lo que representa una oportunidad de negocio capaz de ser explotada por Iris Tecnología S.R.L.

Las distintas temáticas a tratar en la maestría de sistemas embebidos brindan las herramientas adecuadas para afrontar este proyecto.

El desafío es implementar un sistema de control de accesos y toda la plataforma e infraestructura necesaria para realizar una prueba de concepto integral, que luego permita su escalabilidad sin presentar limitaciones.

1.5. Alcance

En esta sección se detallan los alcances del desarrollo.

El presente desarrollo incluye:

1. Estudio y selección de componentes para dispositivo central y dispositivo sensor.
 - Biblioteca para controlar módulo de comunicación GPRS.
 - Módulo de firmware para permitir acceder a datos de un servidor web.
 - Creación de módulo de comunicaciones para enlazar al dispositivo sensor.
 - Módulo para control de cerraduras eléctricas o electromecánicas.
 - Módulo para acceso a memoria externa o tarjeta SD.
2. Desarrollo de firmware para el dispositivo de control central.
3. Desarrollo de firmware para el dispositivo lector.
 - Biblioteca para controlar el lector de tarjetas de acceso.
 - Módulo de firmware para controlar el transceptor Bluetooth.

- Creación de módulo de comunicaciones para enlazar al dispositivo sensor.
 - Módulo para medición y control de nivel de batería del sensor.
4. Desarrollo de aplicación celular para configurar y operar el prototipo.
 5. Montaje de prototipo funcional utilizando módulos o placas de desarrollo.

El presente desarrollo no incluye:

1. Desarrollo de hardware para equipo final.
2. Desarrollo de gabinetes o matrices de inyección.
3. Desarrollo de fuentes de alimentación.
4. Estudio y adecuación a normas aplicables.

1.6. Normativa aplicable

Para todo desarrollo de un dispositivo electrónico resulta fundamental el estudio de la normativa aplicable desde el inicio del proyecto, pero por razones de tiempo y recursos no se abordó el tema en este trabajo. Además, se preveía que el dispositivo aún no sería comercializado al finalizar esta implementación, en esta etapa se trata de un prototipo que representa una prueba de concepto.

Dependiendo de las aplicaciones y mercados de destino del producto podrían resultar aplicables distintas normas. Aunque su estudio se deja fuera del alcance como se mencionó anteriormente, se realizó una pequeña investigación preliminar al respecto. A continuación se detallan algunas normas que se observaron y que podrían aplicar:

- IEC 60730-1:2013+AMD1:2015+AMD2:2020 Controles eléctricos automáticos [4]: se aplica a los controles eléctricos automáticos para uso en, sobre o en asociación con equipos para uso doméstico y similares. El equipo puede utilizar electricidad, gas, petróleo, combustible sólido, energía solar térmica, etc., o una combinación de los mismos. Esta norma es aplicable a los controles para la automatización de edificios dentro del alcance de la norma ISO 16484.
- ISO/IEC 19790:2012 Tecnología de la información, técnicas de seguridad, requisitos de seguridad para módulos criptográficos [5]: esta norma no es mandatoria para el tipo de producto, pero podría aportarle valor agregado. Es una norma internacional que define cuatro niveles de seguridad para los módulos criptográficos con el objetivo de proporcionar un amplio espectro de sensibilidad de datos y una diversidad de entornos de aplicación como por ejemplo una instalación vigilada, una oficina o medios extraíbles.

1.7. Requerimientos

1.7.1. Generales

RGE01 Operación: el dispositivo central será el encargado de controlar el accionamiento de una cerradura eléctrica o electromecánica, otorgando o rechazando el acceso a un determinado lugar, contrastando las credenciales de acceso provistas por el usuario a través de un método de identificación, contra la base de datos cargada en la memoria del dispositivo.

El dispositivo lector, se ocupará de leer las credenciales de acceso del tipo RFID y enviarlas al controlador central. Este dispositivo funcionará a baterías, con un consumo reducido, y por lo tanto deberá monitorear en forma continua su estado y enviar alertas al controlador central en caso de resultar necesario.

RGE02 Conexión: el dispositivo central se conectará a un servidor web a fin de sincronizar los datos de los usuarios habilitados. Esta conexión se realizará por medio de una red Wi-Fi o a través de una red celular GPRS. La comunicación entre el dispositivo central y el dispositivo lector deberá ser inalámbrica y se sugiere utilizar módulos transceptores de RF en las bandas de 433 MHz o 915 MHz.

RGE03 Plataforma: el dispositivo central se basa en el módulo ESP32, que incluye un procesador doble núcleo Xtensa® 32-bit LX6 cuya frecuencia de reloj puede ascender hasta los 240 MHz. El módulo cuenta también con transceptores Wi-Fi y BLE.

1.7.2. Sobre arquitectura

El sistema de control de acceso se partitionará en los subsistemas detallados a continuación:

RAR01 Subsistema de comunicaciones Wi-Fi: debe permitir establecer la comunicación del dispositivo central con un servidor web, a través de una red Wi-Fi e Internet. Se accederá al servidor y se consultará una API, cuya respuesta será en formato JSON.

RAR02 Subsistema de comunicaciones GPRS: debe ser capaz de recibir y decodificar mensajes o paquetes provenientes desde una red de celular GPRS, a fin de gestionar las altas y bajas de las credenciales de acceso en la memoria del dispositivo central.

RAR03 Subsistema de sincronización y evaluación de accesos: deberá ser capaz de gestionar las altas y bajas de credenciales de acceso, valiéndose de la información intercambiada por los subsistemas WiFi y GPRS. Deberá contrastar cada credencial recibida en el lector contra la base de datos interna, a fin de otorgar o rechazar el acceso, enviando o no la orden de apertura al dispositivo electromecánico.

- RAR04 Subsistema de lectura de credenciales de acceso: se encargará de atender y procesar los datos provenientes de los distintos métodos de identificación, generando la información de acceso que se enviará al subsistema de sincronización y evaluación de acceso.
- RAR05 Subsistema de monitoreo de estado y batería: se ocupará de monitorear periódicamente la batería del dispositivo lector, y dará aviso al controlador central en caso de ser necesario.
- RAR06 Subsistema de enlace de módulos: deberá permitir establecer una conexión entre el dispositivo central y el dispositivo lector, con el fin de intercambiar información relacionada a las credenciales y al estado del lector.

1.7.3. Específicos

Requerimientos funcionales:

- RFU01 El sistema deberá permitir el desbloqueo de un dispositivo electromecánico con el fin de permitir el acceso a un área restringida a una persona autorizada.
- RFU02 El sistema deberá ser capaz de admitir credenciales recibidas por Bluetooth.
- RFU03 El sistema deberá permitir la apertura de los accesos a distancia.
- RFU04 El sistema deberá ser capaz de utilizar tarjetas RFID del tipo MIFARE de 13.56 MHz como credencial de acceso.
- RFU05 El sistema deberá permitir la gestión de credenciales y administración de permisos en forma totalmente remota.
- RFU06 El sistema deberá ser desarrollado de manera tal que su posibilidad de escalabilidad esté garantizada.
- RFU07 La interfaz del usuario deberá desplegarse en una aplicación celular.

Requerimientos de hardware:

- RHA01 El dispositivo central debe ser diseñado en base al módulo ESP32.
- RHA02 El módulo de comunicaciones vía red celular deberá ser el SIM800C.
- RHA03 La comunicación entre el procesador central y el módulo SIM800C se realizará a través del protocolo serial.
- RHA04 El dispositivo central deberá incluir un buzzer para emitir confirmaciones y avisos al usuario.

- RHA05 El dispositivo lector estará basado en un microcontrolador de 8 bits de bajo costo.
- RHA06 El dispositivo lector deberá ser alimentado con dos baterías estándar de tamaño AA y será prioritario minimizar su nivel de consumo para aumentar al máximo la duración de las baterías utilizadas.
- RHA07 El dispositivo lector deberá ser capaz de leer y escribir tarjetas del tipo MIFARE de 13.56 MHz.
- RHA08 El dispositivo lector deberá ser capaz de establecer una comunicación inalámbrica con el dispositivo central.
- RHA09 El dispositivo lector deberá monitorear su nivel de batería utilizando el conversor analógico digital interno del su microcontrolador.
- RHA10 El dispositivo lector deberá incluir un pulsador o detector de proximidad, que habilite el ciclo de lectura y transmisión en el momento adecuado, permitiendo luego desenergizar el sensor RFID e ingresar en modo sueño profundo.

Requerimientos de interfaz de usuario:

- RUI01 La interfaz de usuario deberá desplegarse en una aplicación celular.
- RUI02 La aplicación celular se deberá programar utilizando el framework Flutter y lenguaje Dart.
- RUI03 La interfaz deberá permitir crear un usuario y acceder al sistema de forma segura.
- RUI04 La interfaz deberá permitir desbloquear accesos presionando un botón para utilizar Wi-Fi y otro para utilizar BLE.
- RUI05 La interfaz deberá permitir gestionar credenciales RFID.
- RUI06 La interfaz deberá permitir agregar dispositivos a la cuenta del usuario escaneando un código QR.

Requerimientos de software y firmware:

- RSF01 El firmware desarrollado para el módulo ESP32 deberá utilizar el sistema operativo FreeRTOS.
- RSF02 La API que cubrirá las funciones requeridas por el sistema deberá ser implementada sobre el framework Dotnet Core en lenguaje C#.
- RSF03 El software del lado del servidor correrá en un App Service de Microsoft Azure.
- RSF04 La API rest a desarrollar deberá conectarse a una base de datos MySql versión 5.7.

Requerimientos sobre la documentación:

- RDO01 Al finalizar el proyecto se deberá presentar una memoria técnica.
- RDO02 Dentro de los dos primeros meses de iniciado el proyecto se deberá contar con una planificación detallada.
- RDO03 Al finalizar el proyecto se deberá contar con una presentación que resuma todo el proceso de desarrollo e implementación.

Capítulo 2

Introducción específica

En el presente capítulo se describen las principales tecnologías utilizadas para la realización del trabajo, entre las que se incluyen plataformas de software, infraestructura, protocolos y componentes de hardware.

2.1. Tarjetas de proximidad

RFID o identificación por radiofrecuencia (por las siglas en inglés de Radio Frequency IDentification) es un sistema de almacenamiento y recuperación de datos remotos que usa dispositivos denominados etiquetas, tarjetas o transpondedores. El propósito fundamental de la tecnología es transmitir la identidad de un objeto mediante ondas de radio [6].

La solución de identificación por radiofrecuencia más básica incluye tres componentes de hardware principales: la etiqueta RFID, el lector RFID y la antena. Esto es, por supuesto, una simplificación excesiva de lo que se necesita para aplicar la tecnología, pero estos son los bloques de construcción fundamentales. Comprender los fundamentos de RFID es la clave que permite tener éxito en la aplicación de la tecnología [7].

Los sistemas de control de acceso representan una de las aplicaciones en las que más se utiliza esta tecnología. En este ámbito se emplean mayormente etiquetas del tipo pasivo, que son aquellas que no utilizan fuente de energía propia, sino que la consiguen del campo electromagnético que genera el lector o interrogador. Existen tags RFID que trabajan en distintas bandas de frecuencia. En este resumen se hace foco en los dos tipos de tarjeta mayormente utilizados para esta aplicación, por su disponibilidad, costo y características, que son las EM4100/EM4200 y las MIFARE.

EM-Marin es una tecnología RFID que se ha convertido en estándar y es utilizada por muchos sistemas de control de acceso, operando en la frecuencia de 125 kHz. EM4200 [8] es el chip que se utiliza en la actualidad y es compatible con sus predecesores EM4102 y EM4100. Ofrece un rango de lectura de hasta 20 cm según el entorno, el lector y la antena. Su nivel de seguridad es bajo, pero útil en una variedad de entornos que requieren poca seguridad o en un entorno ya seguro. Este tipo de tarjetas es de solo lectura, envía al lector simplemente un número de serie que representa la identificación de la entidad y se supone único. Para el presente trabajo se consideró que este grado de seguridad no era aceptable y se descartó su uso.

MIFARE Classic® [9] es una de las tecnologías RFID de más rápido crecimiento en el mundo y es una marca registrada de NXP Semiconductors. Trabaja en una frecuencia de 13,56 MHz y a menudo es llamada tarjeta inteligente sin contacto, lo que significa que no solo envía un número de serie o identificación cuando entra en contacto con un interrogador, sino que también permite guardar información en su memoria interna protegida mediante el uso de llaves criptográficas. Se encuentran contempladas dentro de la norma ISO/IEC 14443-3:2018 [10].

La versión Classic 1K cuenta con una capacidad de 1.024 bytes, se encuentra dividida en 16 sectores y protegida por 2 claves de seguridad diferentes (A y B). Cada sector está estructurado en 4 bloques de 16 bytes. De la capacidad total se aprovechan 752 bytes para su uso en aplicaciones, debido a que ciertos bloques están destinados a alojar datos del fabricante. Utiliza un mecanismo de cifrado desarrollado por la empresa Philips que se llama CRIPTO1 [11].

Para el desarrollo del trabajo se optó por el modelo recién mencionado. Su manejo resulta más complejo que en el caso de las EM, pero el costo es similar y su capacidad de encriptación ofrece la seguridad que la implementación requiere. En la tabla 2.1 se observa una comparativa de ciertas características de las tarjetas, que resultaron relevantes en la toma de decisión.

TABLA 2.1. Diferencias entre las tarjetas EM4200 y MIFARE Classic 1K.

Característica	EM-Marin	MIFARE
Capacidad de escritura	No	Sí
Memoria disponible	-	752 bytes
Capacidad de encriptación	No	Sí
Distancia de lectura	10 a 15 cm	10 a 30 cm
Frecuencia de trabajo	125 kHz	13.56 MHz
Algoritmo anticolisión	No	Sí
Costo aproximado	U\$S 0,26	U\$S 0,38

2.2. Kit de desarrollo ESP32-DevKitC

Para este trabajo se utilizó el kit de desarrollo ESP32-DevKitC [12] comercializado por la empresa Espressif Systems que se observa en la figura 2.1. Este kit integra un módulo ESP32-WROOM-32 [13] de la misma empresa y por este motivo puede que no todos sus periféricos y entradas-salidas de propósito general (GPIO por sus siglas en inglés) estén disponibles, pero cuenta con los suficientes para esta implementación.



FIGURA 2.1. Kit de desarrollo ESP32-DevKitC V1¹.

A continuación se detallan las principales características del kit ESP32-DevKitC:

- Doble hilera de pines con casi todas las entradas-salidas de propósito general del ESP32-WROOM-32.
 - Puente USB-UART conectado al módulo ESP32-WROOM-32. Esta interfaz es muy útil para programación y depuración.
 - Regulador LDO para proveer alimentación a los elementos del kit.

2.2.1. Características principales del módulo ESP32

El módulo ESP32-WROOM-32 es un microcontrolador de doble núcleo con interfaces Wi-Fi y Bluetooth, se puede observar en la figura 2.2. . Además de las interfaces anteriores, el módulo cuenta con interfaces UART, SPI, numerosas entradas-salidas de propósito general, conversores analógico-digital y conversores digital-analógico.



FIGURA 2.2. Módulo ESP32-WROOM-32².

¹<https://www.espressif.com/en/products/devkits/esp32-devkitc/overview>

²<https://www.espressif.com/en/products/modules>

De las especificaciones del módulo ESP32-WROOM-32 se pueden destacar las siguientes:

- Procesador dual core Xtensa® LX6 de 32 bits.
- Velocidad de reloj de hasta 240 MHz.
- 520 KB de RAM.
- 4 MB SPI flash.
- Wi-Fi integrado con posibilidad de trabajar como AP (*Access Point*) y SM (*Station Mode*).
- Bluetooth V4.2.
- 36 GPIO.
- Hasta 18 conversores analógico-digital (ADC) de 12 bits de resolución.
- 2 conversores digital-analógico (DAC) de 8 bits de resolución.
- 3 UART.
- 2 canales I2C.
- 4 canales SPI.
- Interfaz JTAG.

2.2.2. Plataforma de desarrollo ESP-IDF

Espressif Systems provee un entorno de desarrollo de software denominado ESP-IDF [14]. El entorno ESP-IDF contiene todo lo necesario para desarrollar aplicaciones para los módulos ESP-32 sobre cualquier sistema operativo: Windows, Linux o MAC OS. Este entorno de desarrollo es *Open-Source* y puede ser clonado desde un repositorio de GitHub³. La empresa constantemente realiza actualizaciones y correcciones de fallas.

A continuación se listan algunas de las características más destacables del entorno ESP-IDF:

- Soporta FreeRTOS [15].
- Soporta diferentes controladores de periféricos (SPI, I2C, UART, GPIO, I2S, ADC, DAC, etc) [16].
- Soporta librerías para Wi-Fi [17] y Bluetooth [18].
- Soporta varios *stacks* de redes, por ejemplo TCP/IP.
- Soporta varias implementaciones de protocolos (DHCP cliente y servidor, HTTP cliente y servidor, MQTT, etc) [19].
- Soporta extensiones para Eclipse [20] y Visual Studio Code [21].
- Está basado en CMake [22].

³Repositorio GitHub para el ESP-IDF <https://github.com/espressif/esp-idf>

El entorno ESP-IDF no es parte del proyecto del usuario sino que debe ser enlazado por medio de la variable de entorno IDF_PATH. Esto último ayuda a separar el entorno de desarrollo del proyecto particular del usuario. Para utilizar ESP-IDF se requiere una estructura particular de archivos y carpetas para el proyecto [23].

2.3. Módulo GSM/GPRS SIM800C

Los nuevos escenarios que continúan surgiendo en la actualidad relacionados con las conexiones de máquina a máquina (M2M), presentan fuertes necesidades asociadas a la conectividad. Para satisfacer estas demandas, las compañías de telefonía celular se encuentran trabajando en acelerar la evolución y mejorar la cobertura de dos tecnologías pertenecientes a la familia de redes inalámbricas de bajo consumo y área amplia, LPWAN [24] por sus siglas en inglés. Se trata de las tecnologías LTE-M, conocida como CAT-M1, y Narrow Band IoT (NB-IoT), que trabajan sobre bandas licenciadas, sin riesgos de interferencias y bajo los estándares de la 3GPP (3rd Generation Partnership Project) que aseguran compatibilidad.

Si bien la tendencia son las tecnologías antes mencionadas, por las dificultades de acceso a los servicios necesarios y la baja disponibilidad de los correspondientes módulos de hardware en el mercado local, en este trabajo se decidió utilizar la red GSM [25]. De todas formas, el desarrollo queda planteado y preparado para poder operar con otro tipo de módulos de comunicación por red celular, y esto se ve favorecido gracias a los estándares utilizados y a la elección del protocolo descripto en la sección 2.8.2. Hubiese resultado más sencillo establecer una conexión a un servidor utilizando las funciones ya programadas dentro del firmware de módulos más modernos, pero al hacerlo, la arquitectura del firmware no resultaría la deseada.

Para la solución implementada se eligió el módulo SIM800C, que se introduce a continuación:

El módulo SIM800 [26] que se muestra en la figura 2.3 es capaz de operar en las siguientes cuatro bandas: GSM-850, EGSM-900, DCS-1800 y PCS-1900. Puede transmitir información de voz, mensajes SMS y datos con bajo consumo de energía. Su tamaño es de solo 24 mm x 24 mm x 3 mm y se controla a través de los comandos AT [27] estandarizados y establecidos por el Instituto Europeo de Normas de Telecomunicaciones (ETSI).



FIGURA 2.3. Módulo GSM/GPRS SIM800⁴.

⁴<https://www.simcom.com/product/SIM800.html>

Debido a que el módulo GSM descripto se presenta en un encapsulado de montaje superficial, y esto genera dificultades al montar un prototipo, se adquirió el módulo SIM800C, que incluye al anterior y agrega la antena, el zócalo para la tarjeta SIM y los componentes mínimos necesarios para su funcionamiento. En la figura 2.4 se ve una imagen del componente y se identifican las funciones de sus pines.



FIGURA 2.4. Distribución de pines del módulo SIM800⁵.

2.4. Módulo Bluetooth Low Energy RN4870

Con el objetivo de lograr establecer una comunicación confiable y con un bajo consumo, entre el dispositivo central y el dispositivo lector, se estudiaron distintas alternativas. Inicialmente el cliente había propuesto el uso de transceptores de radiofrecuencia, en las bandas de 433 MHz o 915 MHz, pero se creyó que esto derivaría en una solución insegura, dado que surgiría la necesidad de crear un protocolo propietario, poco desarrollado y que no habría sido suficientemente testeado. En este sentido, se sugirió utilizar también una conexión Bluetooth Low Energy [28] (BLE) que aprovecharía esa capacidad ya presente en el dispositivo central.

Una vez tomada la decisión respecto a la tecnología a utilizar, se buscó un hardware que satisfaga las necesidades del sistema. En la tabla 2.2 se listan las alternativas evaluadas. Las dos características de mayor interés tomadas en cuenta para el trabajo fueron el nivel de consumo y el tamaño de los módulos, todas las opciones evaluadas cuentan con las certificaciones necesarias para evitar los costos relacionados al momento de la producción en serie y comercialización.

TABLA 2.2. Módulos Bluetooth Low Energy analizados.

Fabricante	Número de parte	Consumo pico	Tamaño
Fanstel (nRF52822)	BT832A	5.4 mA	Reducido
Dialog Semiconductor	TINY DA14531	4.1 mA	Reducido
Cypress Semiconductor	EZ-BLE	5.9 mA	Reducido
Texas Instruments	CC2650MODA	9.4 mA	Reducido
Espressif	ESP32	120 mA	Grande
Microchip	RN4780	10 mA	Medio

⁵<https://descubrearduino.com/sim800l-gsm/>

Finalmente, si bien varios de los módulos analizados cumplieron con los requisitos, por disponibilidad, tamaño y consumo se optó por la solución ofrecida por la empresa Microchip, con su módulo BLE con número de parte RN4870 [29].

El módulo RN4870 Bluetooth Low Energy está diseñado para una fácil implementación en una amplia gama de aplicaciones. Ofrece una mejora del rendimiento de hasta 2,5 veces y conexiones más seguras en comparación con los productos basados en Bluetooth 4.1. Se puede interactuar fácilmente con el dispositivo a través de una interfaz UART estándar, disponible en la mayoría de los microcontroladores.

El RN4870 tiene un stack Bluetooth completamente integrado y ofrece una versión certificada blindada con antena incorporada. De este modo los desarrolladores se liberan de las complejidades del software específico de bajo nivel y del desarrollo de RF. Según el fabricante resulta perfecto para aplicaciones de IoT (Internet de las cosas).

A continuación se listan sus características principales y en la figura 2.5 se expone la imagen del dispositivo:

- Certificado Bluetooth 5.
- Módulo certificado para su uso en EE. UU., Canadá, Europa, Japón, Corea, Taiwán y China.
- Interfaz de comando ASCII para la comunicación con el microcontrolador host.
- Soporta conexiones seguras en niveles 1 a 3.
- Admite cargas útiles de hasta 151 bytes con extensiones de longitud de datos.
- Lista blanca para determinar conexiones.
- Funciones de secuencias de comandos para permitir el funcionamiento sin un microcontrolador anfitrión.
- Antena: antena de chip de cerámica o conexión de antena externa.



FIGURA 2.5. Módulo BLE RN4870 de Microchip ⁶.

⁶<https://www.microchip.com/en-us/product/RN4870>

2.5. Módulo lector/grabador MRFC-522

En relación a la tecnología RFID utilizada, se seleccionó uno de los circuitos integrados lectores de tarjetas sin contacto que fabrica la misma empresa proveedora de los chips MIFARE. Se trata del circuito integrado MFRC522 [30].

El MFRC522 es un lector/escritor para comunicación sin contacto altamente integrado, que trabaja en una frecuencia de 13,56 MHz y es compatible con la norma ISO/IEC 14443, MIFARE y NTAG.

El transmisor interno del MFRC522 puede manejar una antena lectora / escritora diseñada para comunicarse con tarjetas y transpondedores ISO / IEC 14443 A / MIFARE sin circuito activo. El módulo receptor proporciona una implementación robusta y eficiente para una demodulación y decodificación de señales de tarjetas y transpondedores compatibles con la norma mencionada. Incluye la funcionalidad de detección de errores, para lo que utiliza paridad y CRC. El MFRC522 es compatible con los productos MF1xxS20, MF1xxS70 y MF1xxS50. Además admite la comunicación sin contacto y MIFARE a las velocidades de transferencia más altas, de hasta 848 kBd en ambas direcciones.

Al igual que ya ocurrió con otros circuitos integrados cuyo encapsulado es de montaje superficial, para implementar el prototipo en este proyecto se buscó un módulo de desarrollo que incluya a este chip. A futuro, en el momento de diseñar un circuito impreso dedicado, también se tendrá en cuenta sustituir este lector por otro más moderno y eficiente que el fabricante recomienda: el IC CLRC663 [31], que presenta un menor consumo y una performance superior a nivel general.

En la figura 2.6 se ve el módulo genérico RFID-RC522 adquirido, que incluye el chip lector, la antena y los componentes asociados necesarios. En la imagen se puede observar también su disposición de pines.

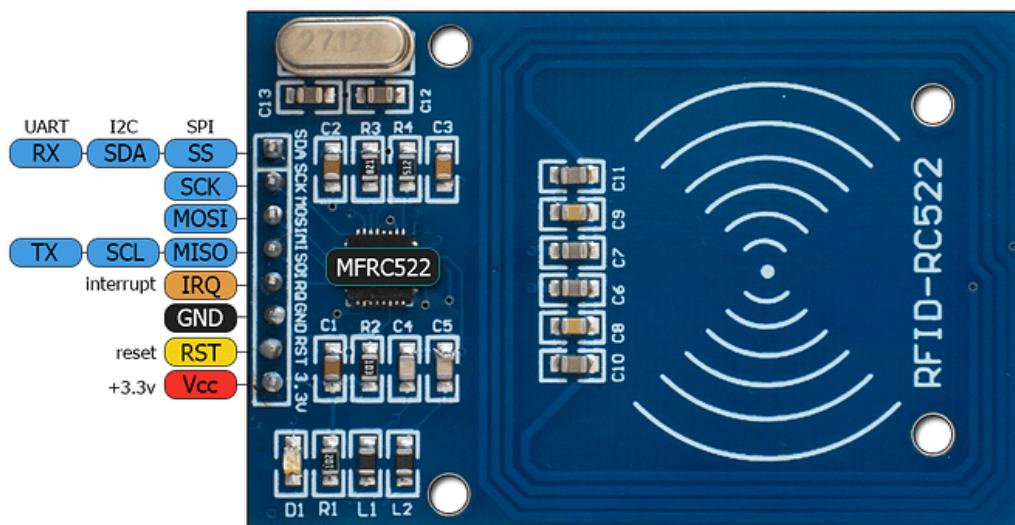


FIGURA 2.6. Distribución de pines del módulo genérico RFID-RC522⁷.

⁷https://http2.mlstatic.com/D_NQ_NP_2X_646702-MLA31047912812_062019-F.webp/

2.6. Microcontrolador PIC16F1718

En el caso del dispositivo lector, cuyo nivel de consumo debe ser lo más bajo posible por estar alimentado a baterías, se utilizó un microcontrolador de 8 bits de bajo costo. En este caso no se realizó un análisis profundo debido a que no se necesitaron características muy específicas o avanzadas. A su vez, el cliente propuso el uso del microcontrolador PIC16F1718 [32] a raíz de que actualmente lo importa y tiene disponible para el montaje de otros productos.

El PIC16F1718 es fabricado por la empresa Microchip y es un microcontrolador de 8 bits perteneciente la familia de gama media mejorada [33], según como la denomina la firma.

El chip combina una integración analógica inteligente de bajo costo y potencia extremadamente baja (XLP) para adaptarse a una variedad de aplicaciones de uso general. Este microcontrolador ofrece amplificadores operacionales en chip, periféricos independientes del núcleo como celdas lógicas programables, oscilador con control numérico o generador de ondas complementario. En lo relacionado a este trabajo, se utilizarán sus interfaces SPI y UART para controlar el dispositivo lector de tarjetas descripto en la sección 2.5 y el módulo Bluetooth Low Energy detallado en la sección 2.4. En la tabla 2.3 se puntualizan las características que resultaron de interés.

TABLA 2.3. Características del microcontrolador PIC16F1718 relevantes para el sistema.

Característica	Valor / Detalle
Costo	Bajo
Características de bajo consumo	eXtreme Low-Power (XLP)
Consumo en modo sleep	50 nA
Consumo por MHz	32 uA
Memoria para programa	16 Kwords
Memoria no volátil de alta duración	128 B
Oscilador de reloj interno	31 kHz a 32 MHz
Interfaces serial	SPI - UART
Conversor analógico digital	8 bits

2.7. Detector de proximidad

Con el objetivo de minimizar el consumo del dispositivo lector resulta mandatorio desactivar el transmisor del interrogador de tarjetas RFID cuando no se encuentre en uso. Para este propósito se utilizó un detector de movimientos basado en un sensor de infrarrojos pasivo, pero no se tiene la certeza de que se adapte a todos los escenarios. Este punto queda fuera del alcance de esta etapa del trabajo y se continuará investigando para hallar la mejor solución que se aplicará en la implementación comercial.

Se utilizó el módulo PIR modelo HC-SR501 [34] que se observa en la figura 2.7, es de bajo costo, pequeño y se basa en el chip LHI778 [35] y el controlador BISS0001 [36]. El sensor utiliza 2 potenciómetros y un jumper que permiten modificar sus parámetros y adaptar su sensibilidad de detección, tiempo de activación, y respuesta, acorde a las necesidades de la aplicación.



FIGURA 2.7. Módulo detector de movimiento PIR HC-SR501 ⁸.

2.8. Protocolos destacados en el proyecto

2.8.1. Protocolo MQTT

MQTT (*Message Queuing Telemetry Transport*) es un estándar de OASIS [37] utilizado para la conectividad de dispositivos IoT y comunicaciones M2M principalmente. Es un protocolo de mensajería de publicación / suscripción, extremadamente simple y liviano, diseñado para dispositivos restringidos y redes de bajo ancho de banda, alta latencia o poco confiables. Los principios de diseño son minimizar el ancho de banda de la red y los requisitos de recursos del dispositivo, al mismo tiempo que se intenta garantizar la confiabilidad y cierto grado de garantía de entrega. Estos principios también resultan en hacer que el protocolo sea ideal para el mundo de los dispositivos IoT y para aplicaciones móviles.

Actualmente se encuentran en vigencia dos estándares de MQTT publicados por Oasis, las versiones v3.1.1 [38] y v5.0 [39]. Únicamente el estandar v3.1.1 se encuentra ratificado por la Organización Internacional de Estandarización (ISO) en su norma ISO/IEC 20922:2016 [40].

El protocolo corre sobre TCP/IP u otros protocolos de red que proporcionen conexiones bidireccionales, que aseguren el orden de recepción de los paquetes y garanticen su entrega. Entre sus características principales se encuentran las siguientes:

- Uso del patrón de mensajes de publicación/suscripción que permite la distribución de un mensaje a varios suscriptores.
- Requiere mínimos recursos de red y hardware, es liviano y eficiente.
- Puede escalar a millones de dispositivos conectados.

⁸https://http2.mlstatic.com/D_NQ_NP_668294-MLA46511955947_062021-O.webp/

- Introduce solo una pequeña sobrecarga de datos de control, minimizada para reducir el tráfico de la red.
- Incluye un mecanismo para notificar a las partes interesadas cuando ocurren errores de conexión.
- Está preparado para trabajar con los protocolos de seguridad y autenticación más conocidos, admite por ejemplo TLS.
- Tiene tres niveles de calidad de servicio (QoS) para la entrega de los mensajes.

A continuación se detallan los distintos niveles de calidad de servicio que ofrece el protocolo:

- QoS0 (A lo sumo una vez): los mensajes se entregan de acuerdo con los mejores esfuerzos del entorno operativo. Puede ocurrir la pérdida de mensajes. Este nivel se puede utilizar por ejemplo con datos provenientes de sensores ambientales donde no importa si una lectura individual es perdida, ya que el próximo mensaje se publicará poco después.
- QoS1 (Al menos una vez): se asegura la llegada de mensajes pero se pueden entregar en forma duplicada.
- QoS2 (Exactamente una vez): se asegura que los mensajes lleguen exactamente una vez. Este nivel podría ser utilizado por sistemas de facturación, donde los mensajes duplicados o perdidos podrían dar lugar a que se apliquen cargos incorrectos.

Por último, se listan y describen los actores que intervienen en el patrón de mensajes de publicación/suscripción:

- Publicador (*Publisher*): cliente encargado de generar la información sobre un tema y publicarla en la infraestructura.
- Suscriptor (*Subscriber*): cliente interesado en recibir información sobre un tema. Este se suscribe a los temas de interés en la infraestructura y esta le notifica cada vez que recibe información sobre dichos temas.
- Servidor o infraestructura (*broker*): situado entre el *publisher* y el *subscriber*. Se encarga de recibir las peticiones de suscripción de los clientes suscriptores, las publicaciones de información de los clientes publicadores y a su vez de retransmitirlas a los clientes suscriptos a esos temas.

2.8.2. Protocolo PPPoS

El protocolo punto a punto (PPP) (en inglés *Point-to-Point Protocol*), es un protocolo del nivel de enlace de datos, utilizado para establecer una conexión directa entre dos nodos de una red. PPP es empleado en varios tipos de redes físicas, incluyendo una red o enlace serial. Se encuentra estandarizado en el documento RFC 1661 [41].

El protocolo punto a punto está diseñado para utilizar enlaces que transportan paquetes entre dos partes. Los enlaces usados deben proporcionar una comunicación bidireccional simultánea, y asegurar el orden y la entrega de los paquetes transmitidos.

En el caso de este trabajo se aprovecha la implementación de PPP contenida dentro del stack lwIP [42], que consiste en una pequeña implementación independiente de la suite de protocolos TCP/IP que fue inicialmente desarrollada por Adam Dunkels. Este stack reducido es especialmente utilizado en el ámbito de los sistemas embebidos.

Puntualmente en este desarrollo, el módulo de comunicación celular toma el rol de servidor PPP y el dispositivo central actúa como cliente PPP. Ambos dispositivos se encuentran conectados a través de un enlace serial, y utilizan PPPoS (en inglés Point-to-Point Protocol over Serial). Por el enlace mencionado, se transportan datos y comandos AT, que deben ser analizados y separados, con la ayuda de buffers circulares y el uso de punteros por ejemplo.

El uso de este protocolo permite que la aplicación se conecte a Internet exactamente del mismo modo en que lo hace a través de la conexión Wi-Fi, y brinda una gran independencia respecto al modelo de módulo de comunicaciones utilizado, permitiendo su futuro reemplazo por uno que utilice las redes LTE sin ningún inconveniente. Además, evita que se haga necesario manejar de dos formas distintas el enlace con el servidor o broker, dependiendo del tipo de conexión en uso.

2.9. Plataformas en la nube

Los avances tecnológicos han originado que hoy en día no sea necesario contar con infraestructura propia, servidores, redes, seguridad o costosas licencias, para poder ofrecer soluciones digitales, productos o servicios.

El cloud computing, o plataforma de cómputo en la nube, es una de las tendencias tecnológicas que se ha impuesto en los últimos años. Permite una importante reducción de costos y da lugar a que pequeñas empresas o desarrolladores puedan tener acceso a una variada y completa infraestructura, que de otra manera no podrían haber considerado. Además, garantiza cierta calidad de servicio, disponibilidad y escalabilidad.

Los mayores proveedores de servicios en la nube con centros de datos que permiten el escalamiento masivo se llaman hiperescaladores. Los cuatro grandes hiperescaladores actualmente son: Microsoft Azure [43], Amazon Web Services o AWS [44], Alibaba Cloud [45], y Google Cloud [46]. Otros proveedores de nubes importantes y conocidos son IBM [47] y Oracle [48].

Para el desarrollo del presente trabajo se decidió utilizar servicios en la nube, con el objetivo de acercarlo a los estándares de esta tecnología y garantizar una futura escalabilidad. A su vez, el costo de la implementación se vio reducido, ya que en este tipo de productos usualmente se generan cargos en relación a lo consumido. Se hizo uso de los servicios provistos por Google Cloud y Microsoft Azure que se describen brevemente a continuación.

2.9.1. Google Cloud Platform

Google IoT Core

IoT Core [49] es un servicio que permite gestionar e ingerir datos de millones de dispositivos repartidos por todo el mundo. Es compatible con protocolos MQTT y HTTP, y permite transferir información en forma bidireccional e incluso enviar actualizaciones de firmware a los dispositivos. Este servicio utiliza TLS 1.2 [50] para garantizar su seguridad e incluye un administrador de dispositivos muy intuitivo.

Google Cloud Functions

Cloud Functions [51], trabajando en conjunto con Cloud IoT Core, permite procesar en tiempo real y analizar datos telemétricos de dispositivos de la Internet de las cosas, permite aplicar una lógica personalizada a los eventos a medida que se producen y esta posibilidad resulta sumamente valiosa para esta implementación.

Google BigQuery

BigQuery [52] es una base de datos administrada, con una interfaz SQL [53], que permite almacenar los datos de la IoT. El elevado rendimiento y bajo costo de BigQuery brinda la posibilidad de mantener grandes volúmenes de valiosos datos, por más tiempo, en lugar de borrarlos periódicamente para ahorrar espacio en disco. Se trata de una plataforma altamente escalable, capaz de analizar incluso petabytes de datos, lo que la convierte en un soporte ideal para concentrar datos, eventos y registros de millones de dispositivos IoT.

2.9.2. Microsoft Azure Cloud

Microsoft Azure App Service

Azure App Service [54] permite crear y alojar aplicaciones web, backends móviles y APIs REST en distintos lenguajes de programación, sin administrar la infraestructura. Ofrece escalado automático y alta disponibilidad, es compatible con Windows y Linux y permite implementaciones automatizadas desde GitHub, Azure DevOps o cualquier repositorio de Git.

Microsoft se encarga de la seguridad, el balanceo de carga, autoescalado y administración de manera automatizada. Los recursos informáticos que Azure usa, quedan determinados o limitados por el plan contratado para ejecutar las aplicaciones.

Microsoft Azure Active Directory B2C

Azure Active Directory B2C [55] es un servicio de administración de identidades que permite personalizar y controlar la manera en que los clientes se registran,

inician sesión y administran sus perfiles al usar las aplicaciones iOS, Android, .NET, de una sola página (SPA) y otras. Incorpora y permite utilizar los protocolos OAuth 2.0 y OpenID Connect, para autorización y autenticación respectivamente.

La autenticación es el proceso para demostrar que una entidad es quien dice ser. La plataforma de identidad de Microsoft usa el protocolo OpenID Connect para administrar la autenticación.

La autorización es el acto de conceder a una parte autenticada permiso para hacer algo. Especifica a qué datos se puede acceder y qué se puede hacer con ellos. La plataforma de identidad de Microsoft usa el protocolo OAuth 2.0.

2.10. Aplicación celular e interfaz de usuario

La interfaz de usuario representa un componente fundamental para la solución desarrollada en este trabajo. La configuración, puesta en marcha y operación del sistema de control de acceso propuesto, debe ser simple e intuitiva.

Se decidió implementar una aplicación celular, que si bien inicialmente fue planteada para funcionar sobre el sistema operativo Android, luego deberá extenderse para su uso en iOS. En este sentido, se optó por una herramienta de desarrollo multiplataforma, que es Flutter. Además de cubrir los requisitos necesarios, su estudio y aplicación resultaron de especial interés al autor de esta memoria.

2.10.1. Framework Flutter

Flutter [56] es una plataforma de código abierto para desarrollo de aplicaciones móviles creada por Google. Suele usarse para desarrollar interfaces de usuario en aplicaciones Android, iOS y Web. Con este kit también se escribe código para aplicaciones e interfaces de usuario implementadas en Fuchsia, un reciente sistema operativo de tiempo real, destinado a dispositivos conectados, que continúa siendo desarrollado por la compañía.

En el último año ha sufrido un crecimiento muy grande en cuanto a su popularidad. Eso se debe a su velocidad de desarrollo y a que logra una experiencia de usuario y una velocidad de renderización tan eficiente como la alcanzada con código nativo. El 3 de marzo de 2021, en el evento virtual llamado "Flutter Engage", Google lanzó Flutter 2. Este fue el cambio oficial más grande que tuvo el SDK.

El conjunto de herramientas que ofrece Flutter para la creación de interfaces de usuario multiplataforma está diseñado para permitir la reutilización de código en sistemas operativos como iOS y Android, al mismo tiempo que permite que las aplicaciones interactúen directamente con distintos servicios de estas plataformas, como ser audio, geolocalización, Bluetooth, cámaras o sensores. Su objetivo es permitir que los desarrolladores entreguen aplicaciones de alto rendimiento y que se sientan cómodos trabajando en diferentes plataformas.

Durante el desarrollo, las aplicaciones de Flutter se ejecutan en una máquina virtual que ofrece una recarga en caliente, mostrando los cambios de estado sin necesidad de una recompilación completa. Para su lanzamiento, las aplicaciones de Flutter se compilan directamente en el código nativo de la plataforma de destino, ya sea Intel x64 o ARM, o JavaScript si tienen como objetivo la web. Este framework posee un gran ecosistema de paquetes de terceros que complementan su funcionalidad.

Flutter fue concebido con la idea de que permitiese crear interfaces de usuario en cualquier dispositivo con pantalla, hasta en una Raspberry Pi, por ejemplo. Las interfaces generadas se basan en la agrupación y anidamiento de objetos o componentes denominados widgets.

Los *widgets*

Los *widgets* son los elementos que afectan y controlan la vista y la interfaz de una aplicación. En Flutter, cada elemento de la pantalla es un widget, siendo estos los bloques de construcción sobre los que se levanta una aplicación. La idea central es que el desarrollador sea quien construya su propia UI con widgets. Flutter permite interactuar con ellos de manera sencilla y limpia. Todo lo que Flutter requiere de la plataforma de destino es un canvas en el que renderizar los widgets para que aparezcan en la pantalla del dispositivo, y el acceso a eventos y servicios como toques, temporizadores o localización, que permitan la interacción con el usuario.

2.10.2. Lenguaje Dart

Flutter usa el lenguaje de programación de código abierto Dart [57], también creado por Google. El objetivo de Dart no era reemplazar a JavaScript como el principal lenguaje de programación web en los navegadores, sino ofrecer una alternativa más moderna. Sin embargo, su simbiosis con Flutter ha catapultado su popularidad. Las siguientes son algunas de las características que junto con Flutter los hacen únicos:

- Dart es uno de los pocos lenguajes que está bien adaptado para ser compilado tanto en forma AOT (*ahead-of-time*, refiriéndose a compilación anticipada), como JIT (*just-in-time*). Usa la compilación JIT durante el desarrollo, utilizando un compilador que es especialmente rápido y que incluye la opción *Stateful Hot Reload* para agilizar el desarrollo, permitiendo al desarrollador ver los resultados de su diseño casi de manera inmediata. Luego, cuando una aplicación está lista para su lanzamiento, se compila AOT, para obtener un código con mayor nivel de optimización. Consecuentemente, con la ayuda de herramientas y compiladores avanzados, Dart puede ofrecer lo mejor de ambos mundos: ciclos de desarrollo extremadamente rápidos y excelentes tiempos de ejecución.
- Dart facilita la creación de animaciones y transiciones suaves que se ejecutan a 60 cuadros por segundo. Dart es capaz de asignar memoria para

objetos y recolectar basura (*garbage collection*) en forma simultánea y sin bloqueos, esto ayuda a evitar demoras no deseadas en la interfaz de usuario. Debido a que las aplicaciones Flutter están compiladas en código nativo, no requieren un puente lento entre dominios, el llamado “puente Javascript”, que a menudo genera cuellos de botella en el rendimiento (este puente es típico en otros frameworks como React Native).

- Dart evita que Flutter tenga la necesidad de utilizar un lenguaje de diseño declarativo como XML o la necesidad de crear la interfaz visual de manera separada. El diseño en Dart, simultáneamente declarativo y programático, es fácil de leer y visualizar. Todo el desarrollo se realiza en un solo lenguaje, que proporciona herramientas avanzadas para simplificar y minimizar el tiempo de desarrollo.

Capítulo 3

Diseño e implementación

En el presente capítulo se describen los aspectos principales relacionados al diseño e implementación. Se explican decisiones sobre selección de componentes y detalles de infraestructura, software, firmware y hardware para el sistema propuesto.

3.1. Descripción del funcionamiento general del sistema

Con el objetivo de cumplir los requerimientos planteados y cubrir las características detalladas en la tabla 2.3, el sistema de control de acceso propuesto fue dividido en cuatro partes principales para su desarrollo e implementación. Como resultado se da lugar a dos componentes físicos de hardware, que son el dispositivo lector de tarjetas RFID y el dispositivo central, y otros dos elementos que consisten en la infraestructura necesaria que conforma el *backend* y una aplicación celular que actúa como *frontend*, cumpliendo la función de interfaz de usuario.

Para cada uno de los cuatro elementos mencionados se eligieron cuidadosamente sus componentes, teniendo en cuenta una serie de aspectos que se consideraron fundamentales.

En el caso del hardware, se prefirió utilizar tecnologías probadas, que tengan buena disponibilidad en el mercado local, que sean de bajo costo, y posean una documentación adecuada y correctamente mantenida. Además, se usaron las plataformas de desarrollo oficiales de cada fabricante, a fin de tener disponible el mayor nivel de soporte en caso de necesitarlo.

En el caso de la infraestructura, se decidió que no se configurarían servidores ni software en forma local. Todos los servicios e infraestructura correspondientes al proyecto se encuentran ubicados en la nube y son provistos por las empresas de tecnología de información más conocidas y mejor calificadas. Se seleccionaron servicios en la nube ofrecidos por Microsoft Azure y por Google Cloud, procurando establecer la mejor configuración que cumpla los requisitos y minimice el costo de operación.

En la figura 3.1 se observa un diagrama en bloques del sistema junto a la infraestructura configurada para dar soporte al mismo y se muestra de manera muy resumida el flujo de los datos que se intercambian los distintos actores.

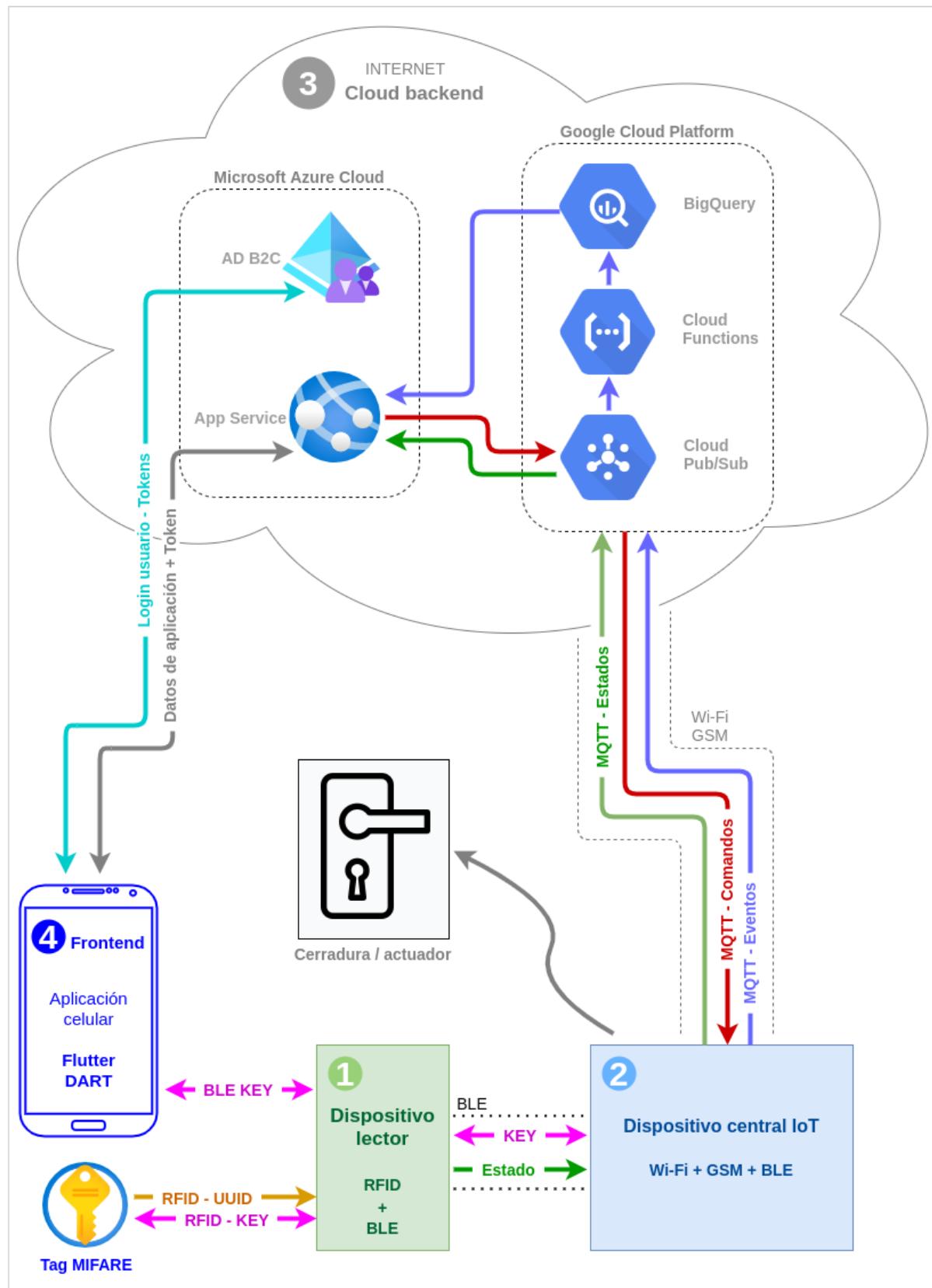


FIGURA 3.1. Diagrama en bloques del sistema y su flujo de información entre componentes.

A continuación se describen brevemente las funciones y características principales de cada una de las cuatro partes que componen al sistema. Se muestran también algunos casos de uso básicos. Luego se ampliará y complementará la información, dentro de este mismo capítulo, en las secciones específicas referidas al hardware, firmware y software:

3.1.1. Lector inalámbrico de tarjetas RFID

El lector es el dispositivo de hardware que hace de nexo entre el dispositivo central y las tarjetas o llaveros MIFARE que se utilizan para abrir las puertas controladas por el sistema de acceso. Dispone de un lector/grabador de tarjetas RFID o NFC (*Near Field Communication*) [58] y un módulo transceptor BLE que le permite intercambiar información con la central. Se encuentra identificado con el número 1 en el diagrama de la figura 3.1.

El lector de tarjetas se implementó con conexión inalámbrica y alimentado a baterías para que su instalación sea sumamente simple y permita ser aplicado en frentes vidriados o con especiales detalles de diseño, sin deteriorar su estética y sin aplicar ningún tipo de cableado.

A continuación se listan sus características y luego se describen sus funciones principales en la tabla 3.1:

- Lector/grabador de tarjetas NFC/MIFARE.
- Frecuencia de portadora del transmisor 13.56 MHz.
- Tensión de alimentación 3 V.
- Ultra bajo consumo.
- Activación por detección de proximidad.
- Alimentado por 2 baterías estándar tamaño AA.
- Inalámbrico - Comunicación BLE con dispositivo central.
- Microcontrolador de 8 bits.

TABLA 3.1. Funciones principales del lector inalámbrico de tarjetas RFID/NFC.

Función	Descripción o secuencia de acciones
Lectura y evaluación de tarjeta de acceso.	<ol style="list-style-type: none"> 1. El dispositivo lee la identificación única de la tarjeta y, de ser válida, continúa con el paso siguiente. 2. Se lee el token con el uso de las claves correspondientes y se comprueba su validez. De ser válido, continúa con el paso siguiente. 3. Se ejecuta la secuencia de actualización de estado. 4. Se transmite hacia el dispositivo central la credencial y el estado.
Inicialización de una tarjeta de acceso.	<ol style="list-style-type: none"> 1. El dispositivo lee la identificación única de la tarjeta. 2. Comprueba que la tarjeta se encuentre desbloqueada. 3. Consulta a la central si esa tarjeta debe inicializarse. 4. Graba el token correspondiente con una clave determinada. 5. Se ejecuta el proceso de lectura desde el inicio.
Actualización de estado.	Cada vez que se activa el dispositivo lector de tarjetas, se lee el nivel de la batería utilizando el conversor analógico digital y se almacena el resultado en la estructura de estado.

3.1.2. Dispositivo central del sistema de control de acceso

El dispositivo central es el corazón del sistema de control de accesos en cuanto a hardware. Se encarga de mantener una conexión persistente con la infraestructura, evalúa los accesos y se ocupa de enviar la señal de apertura al actuador que desbloquea la puerta. Esta unidad central incluye también indicadores lumínicos y sonoros para dar información de estado al usuario. Se encuentra identificado con el número 2 en el diagrama de la figura 3.1.

Para asegurar la conexión con la infraestructura se incluyen dos métodos distintos, el equipo es capaz de conectarse a Internet a través de una red Wi-Fi o por medio de la red GSM. En caso de hacer uso de la segunda alternativa, se debe contar con un servicio de datos con cobertura en la zona de instalación y un chip celular o SIM (por las siglas de *Subscriber Identity Module*) que otorgue acceso a la red. El tráfico de datos que genera el equipo es muy reducido, lo que permite el uso de planes de datos de muy poca capacidad y bajo costo.

Actualmente, varias compañías de telefonía celular ofrecen planes para su uso en comunicaciones M2M. En el caso de este proyecto, se utiliza un plan de datos M2M de la empresa Claro que incluye un paquete de datos de 50 megabytes mensuales. El costo del plan es de \$ 45 o un equivalente en Dólares de U\$S 0,45.

Luego de establecer una conexión a Internet, el equipo se conecta al broker MQTT y se subscribe a distintos tópicos, quedando listo para intercambiar información sobre su estado, transmitir eventos y recibir comandos. En el caso de la presente implementación, el broker MQTT en donde cada dispositivo se encuentra registrado es Google Cloud IoT Core. Este broker incluye a su vez un puente HTTP que se aprovecha para actuar de interfaz con las APIs que se implementan en el servidor de aplicaciones.

A continuación se listan las principales características de este componente:

- Conectividad Wi-Fi.
- Conectividad BLE.
- Protocolo de comunicación MQTT.
- Protocolo de seguridad TLS 1.2.
- Permite autenticación multifactor.
- Permite apertura a distancia por Internet.
- Permite gestión remota.
- Registra eventos de acceso.
- Registra eventos de fallos, de intentos de intrusión y de sistema.
- Configuración por aplicación celular.
- Modos de conexión “siempre Wi-Fi”, “siempre GSM”, “GSM backup”.
- Capacidad de memoria para 1000 usuarios.
- Tensión de alimentación de 12 V suministrada por fuente externa.
- 5 indicadores de estado lumínicos (LEDs).
- *Buzzer* para emitir señal auditiva.
- Entrada para sensor de cerradura.
- Salida para actuador.
- Tiempo de espera de cierre de puerta configurable.
- Tiempo de activación del actuador configurable.
- Salida para buzzer externo.
- Zócalo para SIM nano.
- Montaje: riel DIN.

Como resultado de la interacción del usuario con la aplicación celular, el *backend* compuesto por el conjunto de servicios detallados en el diagrama de la figura 3.1, podrá enviar uno o más comandos al dispositivo central. En la tabla 3.2 se detallan los comandos que el dispositivo acepta, los parámetros que espera y se describe la acción realizada.

TABLA 3.2. Descripción de los comandos y parámetros aceptados por el dispositivo central.

Comando	Parámetro	Descripción
Open	-	Desbloquea el acceso controlado.
Add RFID	UUID	Agrega la identificación de una tarjeta RFID a la lista de permitidas.
Del RFID	UUID	Elimina la identificación de una tarjeta RFID de la lista de permitidas.
Add BLE-KEY	token	Agrega un token de acceso por BLE a la lista de permitidos.
Del BLE-KEY	token	Elimina un token de acceso por BLE de la lista de permitidos.
Set OUT-Time	int (0 - 60)	Configura el tiempo de activación de la salida para el actuador.
Set Alrm-Time	int (0 - 60)	Configura el tiempo máximo de puerta abierta antes de generar un evento de alarma.
Set Int-Bzr	int (0 - 1)	Habilita o deshabilita el buzzer interno.
Set Con-Mode	int (0 - 2)	Configura el método de conexión a Internet a utilizar: Wi-Fi / GSM / Backup.

Según el estado del sistema y el resultado de las acciones ejecutadas, se brinda información al usuario por medio de los indicadores incluidos en el dispositivo. En la tabla 3.3 se detallan las funciones de los distintos indicadores lumínicos y el significado de los sonidos emitidos por el *buzzer*.

TABLA 3.3. Descripción de los indicadores LED y sonidos emitidos por el buzzer.

Indicador	Valor / Detalle
Led 1 - color amarillo	Conexión Wi-Fi establecida.
Led 2 - color azul	Conexión por red celular establecida.
Led 3 - color blanco	Comando recibido.
Led 4 - color verde	Acceso desbloqueado.
Led 5 - color rojo	Acceso bloqueado.
<i>Buzzer</i> - activado 1 segundos	Confirmación de acceso / ok
<i>Buzzer</i> - emite 3 beeps	Aviso de error

Tal como indica el detalle de características presentado anteriormente en esta sección, el sistema es capaz de registrar eventos relacionados con los accesos, y también relacionados a fallos, intentos de intrusión o estados del sistema. Estos eventos se publican vía MQTT y a su vez se actualizan las estructuras de estados en el dispositivo y en Google IoT Core. En la tabla 3.4 se describen los eventos contemplados al momento de esta implementación.

TABLA 3.4. Eventos enviados por el dispositivo central hacia Google Cloud IoT Core.

Evento	Descripción
Tipo de conexión en uso	Indica si el dispositivo central se encuentra enlazado a través de una red Wi-Fi o de la red de telefonía celular.
Reinicio por error desconocido	Indica en un contador la cantidad de reinicios por causa desconocida.
Reinicio por corte de energía	Indica en un contador la cantidad de reinicios por corte de energía.
Alarma de puerta abierta	Envía una alerta que se propaga por el sistema y guarda el último evento en la estructura correspondiente.
Alarma de intento de intrusión	Envía una alerta que se propaga por el sistema y guarda el último evento en la estructura correspondiente.
Evento de acceso correcto	Envía el evento de acceso, especifica el usuario y tipo de credencial para su registro en la bitácora de eventos.

3.1.3. Infraestructura en la nube (*backend*)

La infraestructura que da soporte al sistema se encuentra en la nube en su totalidad y es provista por Microsoft Azure y por Google Cloud. El conjunto de servicios que conforman al *backend* se encuentra identificado con el número 3 en el diagrama de la figura 3.1 y a continuación se describe brevemente el uso que se le da a cada uno de ellos dentro de la presente implementación:

El servicio Google Cloud IoT Core actúa como broker MQTT dentro del sistema y todos los dispositivos conectados deben estar registrados en él. Brinda la posibilidad de mantener una conexión persistente y segura con cada dispositivo. Además, actúa como intermediario entre los dispositivos y el resto del *backend*, ya que desde las centrales la comunicación se realiza por medio del protocolo MQTT y en el resto del sistema se hace a través de HTTP.

Cuando un dispositivo central del sistema genera un evento, lo transmite a Google Cloud IoT Core y este último activa un disparador o *trigger* que despierta o da inicio al servicio Google Cloud Functions. El evento es analizado por el algoritmo alojado dentro de Cloud Functions y genera una acción o solicitud que puede tener como destino a los servicios Google Cloud BigQuery y/o Microsoft Azure App Service.

El servicio Google Cloud BigQuery es una base de datos que resulta muy eficiente para grandes volúmenes de datos y dentro del sistema se ocupa de almacenar la información correspondiente a eventos. Si el sistema escala, el número de eventos generados diariamente por los dispositivos conectados ascendería rápidamente y el servicio BigQuery es adecuado para soportarlo.

El servicio Microsoft Azure Active Directory B2C se utiliza en el sistema para generar los tokens que permiten a un usuario acceder a las APIs hospedadas en la infraestructura. Cuando un usuario pretende registrarse o ingresar a la aplicación del sistema, es derivado a un sitio personalizado de Microsoft Azure que le solicita las credenciales necesarias. AD B2C puede ser configurado también para que requiera autenticación multifactor. Una vez que las credenciales del usuario son validadas, AD B2C devuelve un token que da acceso a los *endpoints* ofrecidos por las APIs del sistema.

El servicio **Microsoft Azure App Service** hospeda al software correspondiente a las APIs que son accedidas por la interfaz de usuario del sistema, le permite al *frontend* solicitar información o ejecutar acciones.

3.1.4. Aplicación celular (*frontend*)

El *frontend* del sistema consiste en una aplicación celular que en un primer momento está destinada a dispositivos Android. La aplicación se conecta al *backend* para obtener información y realizar determinadas acciones y, a su vez, es capaz de conectarse en forma directa con los dispositivos centrales a través de Bluetooth Low Energy con el objetivo de enviar credenciales para desbloqueo de accesos.

Las principales funciones de la aplicación se listan a continuación:

- Gestionar alta y *login* de usuarios.
- Agregar controles de acceso a los usuarios escaneando código QR.
- Gestionar credenciales RFID.
- Configurar parámetros de los controles de acceso.
- Enviar señales de apertura vía BLE a los dispositivos.
- Enviar señales de apertura a distancia a los dispositivos.
- Permite compartir el control de acceso con otros usuarios.
- Permite gestionar roles de los usuarios.

3.2. Hardware

En esta sección se describe la implementación de los dos módulos de hardware desarrollados en el trabajo. Se muestra un detalle del conexionado de componentes para el dispositivo lector de tarjetas MIFARE y para el dispositivo central.

3.2.1. Integración de componentes del dispositivo central

En la figura 3.2 se muestra un esquema del hardware del dispositivo central, cuyo componente central es el kit ESP32-DevKitC. En el diagrama se observa el uso de sus periféricos y su relación con el entorno.

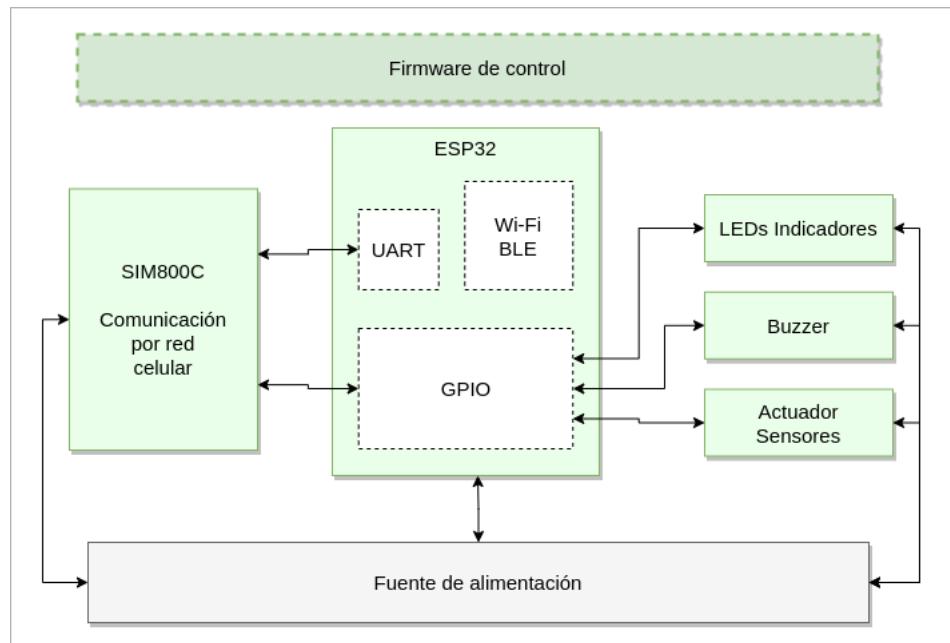


FIGURA 3.2. Diagrama en bloques correspondiente al hardware del dispositivo central.

Para el conexionado de los elementos del sistema se utilizó una placa universal de 100 mm x 100 mm. Esta permitió un rápido conexionado de los módulos y componentes con el kit ESP32-DevKitC y la posibilidad de trabajar con hardware y software sin la necesidad de contar con todo el hardware final diseñado y construido. En la placa universal se montaron los siguientes componentes:

- Kit ESP32-DevKitC.
- Módulo GSM SIM800C.
- Dos reguladores de tensión ajustable tipo step-down.
- El *buzzer*.
- Resistencias limitadoras de corriente para los LEDs indicadores.
- Un transistor de propósito general para activar el *buzzer*.
- Conectores tipo *header* para conectar los LEDs indicadores.

En la figura 3.3 se puede ver la disposición de pines del kit de desarrollo ESP32-DevKitC y en la tabla 3.5 se detallan aquellos utilizados en el dispositivo central desarrollado en el proyecto. En general, todos los componentes fueron conectados directamente al kit, con excepción del *buzzer* que requirió un transistor como intermediario porque el ESP32 no es capaz de entregar la corriente suficiente para su activación en forma directa.

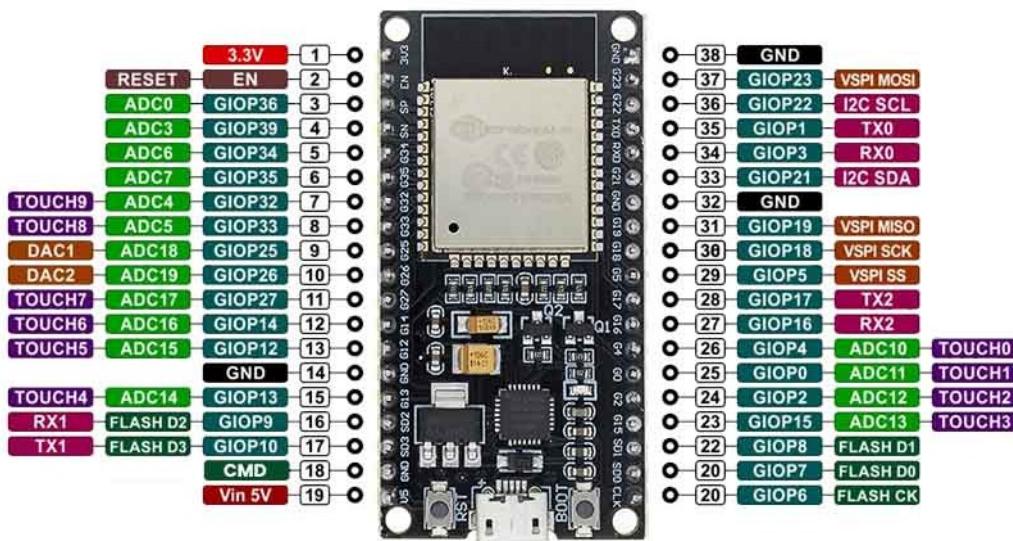


FIGURA 3.3. Disposición de pines del kit de desarrollo ESP32-DevKitC y detalle de sus funciones.

TABLA 3.5. Uso de los pines del kit ESP32-DevKitC.

Pin ESP32 kit	Componente / Módulo
Indicadores LED	
GPIO16	LED indicador de conexión BLE.
GPIO17	LED indicador de conexión Wi-Fi.
GPIO18	LED indicador de acceso desbloqueado.
GPIO19	LED indicador de acceso bloqueado.
GPIO21	LED indicador de comando recibido.
Módulo SIM800C	
GPIO25	UART RX
GPIO26	UART TX
GPIO33	RST
JTAG Debugger	
GPI12	TDI
GPI13	TLK
GPI14	TMS
GPI15	TDO
EN	TRST
Otros	
GPIO4	Buzzer
GPIO5	Sensor de cerradura.

El prototipo se realizó en dos etapas. En un principio, y con el objetivo de poder realizar las primeras pruebas de funcionamiento, se montaron los componentes sobre la placa experimental y en la figura 3.4 se puede ver el resultado. Como paso siguiente, ya en fechas cercanas a la finalización del proyecto, se montó otro circuito experimental que se colocó dentro de un gabinete apto para ser aplicado en un riel DIN estándar. Se observa este prototipo final en la figura 3.5.

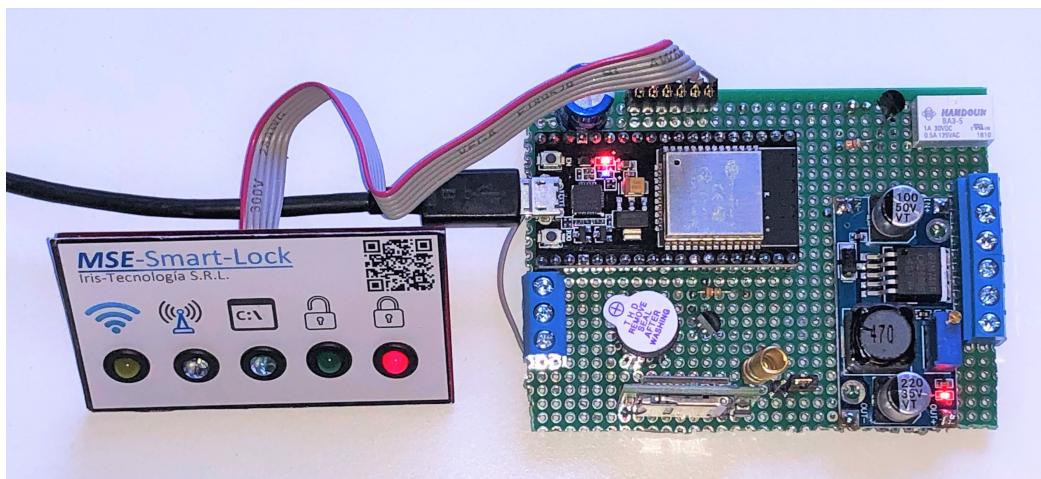


FIGURA 3.4. Prototipo de hardware del dispositivo central en su etapa inicial.



FIGURA 3.5. Prototipo final del dispositivo central.

3.2.2. Integración de componentes del dispositivo lector

En la figura 3.6 se muestra un esquema del hardware del dispositivo lector inalámbrico de tarjetas MIFARE, cuyo componente central es el microcontrolador PIC16F1718. En el diagrama se observa el uso de sus periféricos y su relación con el entorno.

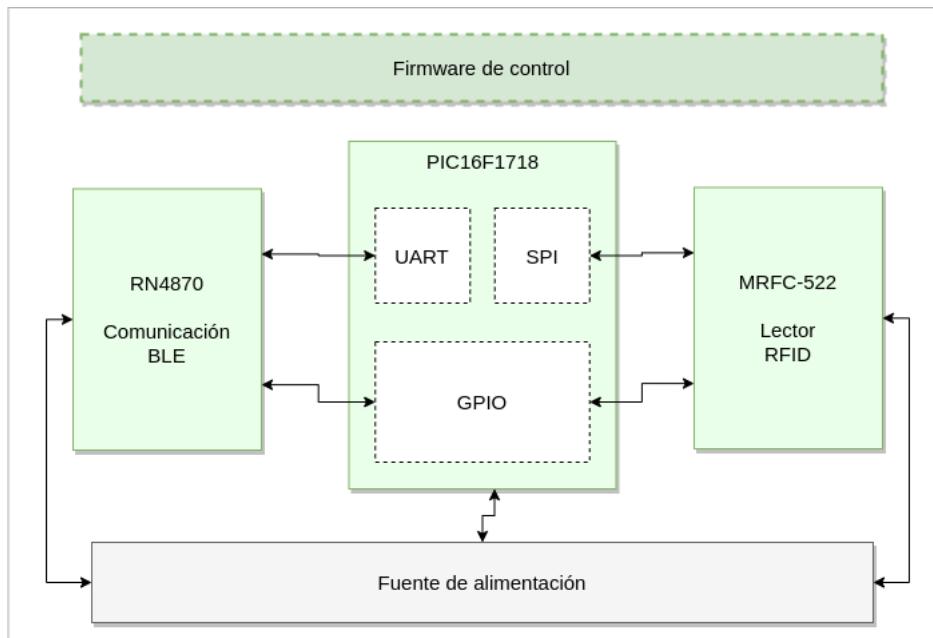


FIGURA 3.6. Diagrama en bloques correspondiente al hardware del dispositivo lector.

Para el conexionado de los elementos del dispositivo lector se utilizó una placa universal de 100 mm x 80 mm. Esta permitió un rápido conexionado de los componentes y la posibilidad de trabajar con hardware y software sin contar aún con todo el prototipo final diseñado. En la placa universal se montaron los siguientes componentes:

- PIC16F1718.
- Módulo BLE RN4870.
- Módulo lector MRFC-522.
- Portapilas para dos baterías tamaño AA.
- Conectores tipo *header* para programación y *debug*.
- Componentes varios requeridos.

En la figura 3.7 se puede ver la disposición de pines del microcontrolador PIC16F1718. En la imagen no se asociaron funciones de periféricos a los pines porque en este controlador se puede asignar libremente casi cualquier pin a cualquier periférico, es totalmente configurable. En la tabla 3.6 se detallan los pines utilizados en el dispositivo lector desarrollado en el proyecto. Todos los componentes fueron conectados directamente al microcontrolador, incluso el módulo RN4870 que por su escaso consumo es alimentado directamente por una salida de propósito general. Este recurso de alimentación directa permite aplicar un *hard reset* al módulo RN cuando resulta necesario y también resulta útil para desenergizarlo en el modo de ahorro de energía.

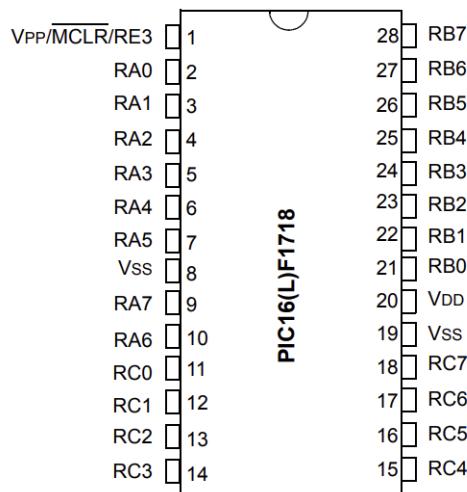


FIGURA 3.7. Disposición de pines del microcontrolador PIC16F1718.

TABLA 3.6. Uso de pines del microcontrolador PIC16F1718.

Pin PIC16F1718	Componente / Módulo
Módulo RN4870	
RA0	Alimentación
RC6	UART TX
RC7	UART RX
Módulo MRFC-522	
RA2	EN
RB1	SCK
RB3	MOSI
RB2	MISO
RA3	RST
<i>ICSP Programming</i>	
RE3	Vpp/MCLR
RB6	ICSPCLK
RB7	ICSPDAT

Este prototipo también se realizó en dos etapas. En la primera, se montaron los componentes sobre la placa experimental y en la figura 3.8 se puede ver el resultado. Como segundo paso, se montó otro circuito experimental más pequeño y se incluyó dentro de un gabinete plástico. Se observa este prototipo final en las figuras 3.9 y 3.10.

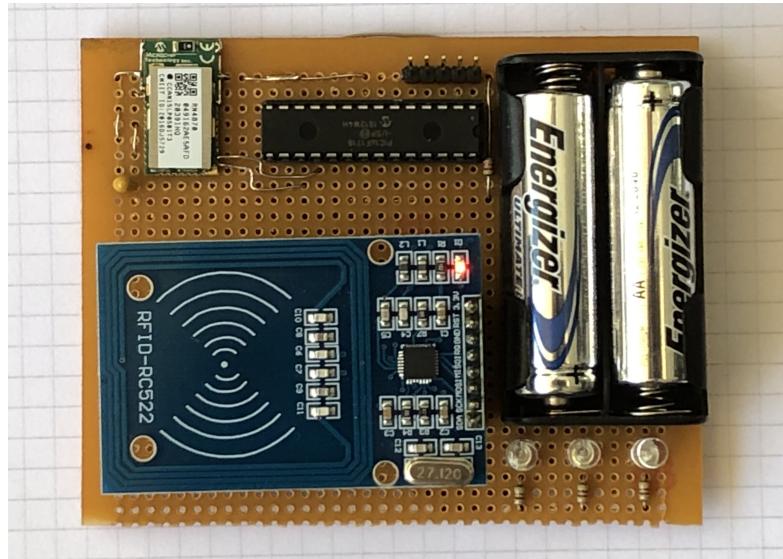
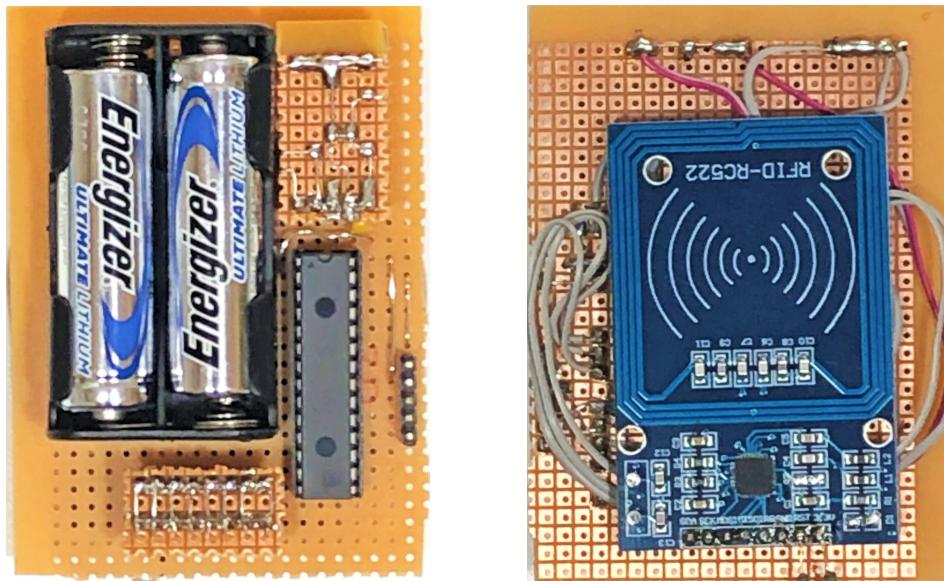


FIGURA 3.8. Prototipo de hardware del dispositivo lector en su etapa inicial.



(A) Reverso.

(B) Frente.

FIGURA 3.9. Prototipo del lector RFID montado en circuito impreso experimental.



FIGURA 3.10. Prototipo del lector ensamblado.

3.3. Firmware

En esta sección se explica como fue llevado a cabo el desarrollo del firmware que compone a todo el sistema completo, comenzando por algunas decisiones de nivel general y luego exponiendo algunas particularidades sobre cada dispositivo.

Ciclo de vida en espiral

Para este desarrollo se optó por el modelo de ciclo de vida en espiral, que es iterativo y expande la complejidad en cada ciclo. Permite la obtención de un prototipo funcional al finalizar cada iteración. Esta característica lo hace valioso para el presente trabajo porque permite realizar distintos ensayos en instancias intermedias, sin necesidad de tener listo el prototipo final. Puede verse en la figura 3.11 un esquema de este modelo.

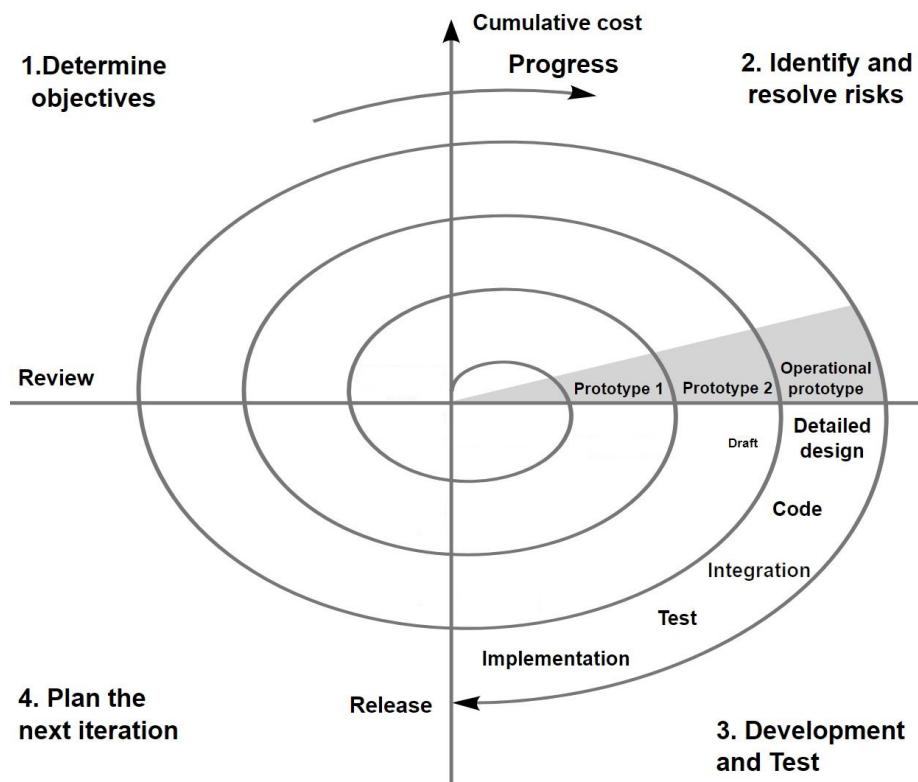


FIGURA 3.11. Modelo de ciclo de vida en espiral (Boehm, 1988)¹.

Buenas prácticas de programación

El firmware fue desarrollado por medio de la utilización de buenas prácticas de programación, como por ejemplo: el control de versiones, la modularización y testeo del código fuente. Para llevar adelante el control de versiones dentro del proyecto, fueron creados cuatro repositorios en la plataforma GitHub, de los cuales dos están destinados al firmware.

Baremetal vs. sistema operativo

Al afrontar las tareas de diseño del firmware, una de las principales decisiones a tomar fue la de si utilizar un sistema operativo de tiempo real o no. Para definirlo se analizó qué ventajas y desventajas relevantes implicaba introducirlo en cada uno de los dispositivos. Se analizó FreeRTOS, un sistema operativo en tiempo

¹https://es.wikipedia.org/wiki/Proceso_del_desarrollo_del_software

real libre, muy bien documentado y sumamente utilizado en sistemas embebidos. Para este proyecto en especial, se han tenido en cuenta los siguientes aspectos:

- "Preemption": esta característica del sistema operativo representa su capacidad de suspender una tarea y ejecutar otra, sea por ventanas de tiempo, prioridad u otras razones. Esta es una gran ventaja y facilita la subdivisión del sistema organizándolo en tareas claras y específicas, asignando niveles de prioridad.
- Acceso a recursos compartidos: el uso de este sistema operativo puede aportar elementos de sincronización, como ser semáforos o mutex, que ayudan a utilizar recursos de manera concurrente pero segura.
- Características de los microcontroladores: si bien la aplicación de un sistema operativo puede resultar de mucha ayuda, debe tenerse en cuenta que agrega una sobrecarga al sistema, que puede ser significativa y hasta contraproducente en microcontroladores de bajos recursos.
- Tareas asíncronas: la inclusión del sistema operativo facilita notablemente la implementación de tareas de ejecución asíncrona.
- Ampliación del sistema: el uso de un sistema operativo facilita la incorporación de nuevas funcionalidades a futuro. Permitirá en tal caso, agregar tareas con su respectiva prioridad, que puedan cooperar con las ya existentes.

En base a los factores mencionados se detalla en la tabla 3.7 que metodología se adoptó para cada uno de los dispositivos:

TABLA 3.7. Modo de programación aplicado por cada dispositivo.

Nombre del dispositivo	Metodología adoptada
Dispositivo central	Uso de FreeRTOS
Dispositivo lector	Programación baremetal

3.3.1. Firmware del controlador central

El firmware correspondiente al dispositivo central fue el que más trabajo y tiempo de estudio requirió para ser implementado. Tiene la responsabilidad de gestionar y evaluar los accesos, controlar el actuador de bloqueo de la zona, registrar los eventos y mantener correctamente atendidas a todas las vías de comunicación que maneja en simultáneo.

Como se detalló anteriormente en la tabla 3.7, el dispositivo central utiliza el sistema operativo FreeRTOS. Su uso facilita ampliamente la ejecución de tareas paralelas continuas, la ejecución de tareas asíncronas y ayuda a compartir recursos, lo que se hace imprescindible para este firmware en particular. En la figura 3.12 se puede ver un esquema de capas que resulta del uso de la plataforma ESP-IDF junto al sistema operativo FreeRTOS en el presente proyecto.

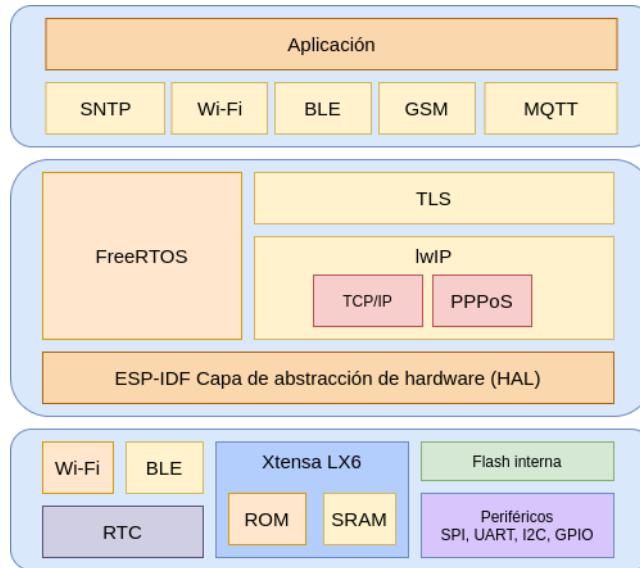


FIGURA 3.12. Arquitectura en capas resultante del uso de la plataforma ESP-IDF y FreeRTOS específicamente para este proyecto.

A continuación, en la tabla 3.8 se observa el detalle de las tareas que se ejecutan en el sistema operativo, su función, tipo de ejecución y nivel de prioridad. Este detalle corresponde únicamente a las tareas desarrolladas, ya que hay varias adicionales en ejecución que responden a funcionalidades de la plataforma ESP-IDF, como ser las tareas para las conexiones Wi-Fi y BLE.

TABLA 3.8. Principales tareas a nivel sistema operativo que se ejecutan en el controlador central, detallando continuidad y prioridad.

Nombre de la tarea	Función que lleva a cabo	Ejecución	Prioridad
Inicializar_sistema	Inicializa los componentes del sistema y lanza al resto de las tareas.	Única vez	1
Adm_conexiones	Supervisa y gestiona las conexiones en base a la configuración elegida y según la disponibilidad de las redes. Los modos posibles son “siempre Wi-Fi”, “siempre GSM” o “backup”.	Continua	2
Atender_GSM	Atiende los eventos y supervisa los estados y errores del módem celular.	Continua	3
Smart-Lock	Gestiona los accesos y credenciales, registra eventos y sincroniza el hardware del actuador.	Continua	1
MQTT_GCP	Da tratamiento a los eventos relacionados con la publicación y suscripción en Google IoT Core .	Continua	1

Para este desarrollo, el código fue modularizado, y con el objetivo de lograr un mayor encapsulamiento e independencia entre componentes de firmware, se utilizaron técnicas para aplicar el paradigma orientado a objetos, en un lenguaje como C, que no está diseñado para esto. En la tabla 3.9 se detallan los módulos principales que fueron desarrollados.

TABLA 3.9. Principales módulos de firmware que conforman al dispositivo central.

Nombre del módulo	Función que lleva a cabo
app_main.c	Inicio de la aplicación, realiza seteos iniciales y lanza las tareas de inicialización.
app_error.c	Biblioteca con funciones comunes para tratamiento de errores.
app_gpio.c	Biblioteca para configuración y uso de los puertos de entrada y salida.
ble_gatt.c	Biblioteca para definir y atender los servicios y características que BLE requiere.
jwt.c	Módulo para la generación de tokens JWT.
mqtt_gcp.c	Módulo para configurar y atender la conexión a Google Cloud IoT Core.
pemkey.c	Módulo que contiene las claves necesarias para obtener acceso a Google Cloud IoT Core.
shared-lock.c	Biblioteca para gestionar el control de acceso.
wifi_main.c	Biblioteca para configurar la conexión Wi-Fi, SSID y credenciales necesarias.
gsm_pppos.c	Biblioteca que incluye los recursos para configurar el módem en modo PPPoS y establecer una conexión a Internet.
conn_admin.c	Módulo para administrar las conexiones Wi-Fi y por red celular.
test.c	Conjunto de funciones de test.

Si bien hay una gran cantidad de módulos que intervienen en el funcionamiento del dispositivo central, se considera interesante mencionar algunos detalles sobre temas puntuales, como por ejemplo: la conexión a Google Cloud IoT Core y la conexión a Internet por red celular.

Conexión a Google Cloud IoT Core

Conectar un solo dispositivo a un servicio IoT puede ser bastante fácil, sin embargo, cuando la cantidad de dispositivos aumenta a algunos miles, se necesita una solución segura y económica que ayude a administrarlos fácilmente y a manejar todos los datos recibidos desde ellos. Este es uno de los motivos por los cuales se decide utilizar los servicios de Google Cloud IoT, que disponen de todo lo necesario para generar soluciones escalables.

Al llevar a cabo producciones en masa, se debe pensar en el modo de aprovisionamiento de los dispositivos, es decir, en la forma en que se dan de alta en las plataformas, se gestionan sus certificados, y otros tantos detalles. Google Cloud IoT Core brinda la posibilidad de realizar este aprovisionamiento a través de una interfaz gráfica, pero también da la posibilidad de hacerlo por medio de línea de comando, scripts, o utilizando APIs que Google pone a disposición.

En el caso de este proyecto, casi la totalidad de los dispositivos se dieron de alta en forma manual, pero también se probó el aprovisionamiento por línea de comando. A continuación, en el fragmento de código 3.1, se detallan a modo de ejemplo comandos que tienen como objetivo crear un servicio IoT en Google Cloud.

```

1 export PROJECT_ID=<your-project-id>
2
3 # Authorize gcloud to access the Cloud Platform with Google user
4 # credentials
5 gcloud auth login
6
7 # Create a new project
8 gcloud projects create $PROJECT_ID
9
10 # Set default project
11 gcloud config set project $PROJECT_ID
12
13 # Enable Cloud Pub/Sub
14 gcloud services enable pubsub.googleapis.com
15
16 # Open this URL to enable billing (required for Cloud IoT)
17 open "https://console.cloud.google.com/iot/api?project=$PROJECT_ID"
18
19 # Enable Cloud IoT
20 gcloud services enable cloudiot.googleapis.com
21
22 # Give permission to Cloud IoT Core to publish messages on Cloud Pub/Sub
23 gcloud projects add-iam-policy-binding $PROJECT_ID \
24   --member=serviceAccount:cloud-iot@system.gserviceaccount.com \
25   --role=roles/pubsub.publisher

```

CÓDIGO 3.1. Comandos para crear un proyecto en Google Cloud Platform y habilitar IoT Core.

Una vez creados los dispositivos por cualquiera de los métodos, se pueden visualizar en el administrador de dispositivos. Se puede observar un ejemplo en la figura 3.13.

The screenshot shows the Google Cloud Platform IoT Core interface. On the left, there's a sidebar with options: 'Detalles del registro', 'Dispositivos' (which is selected and highlighted in blue), 'Puertas de enlace', and 'Supervisión'. The main area is titled 'ID de registro: iris-iot-core-registry' and shows the region 'us-central1'. It lists three devices: 'MSE-Shared-lock-001...', 'MSE-Shared-lock-002...', and 'MSE-Shared-lock-003...'. Each device entry includes its ID, communication status ('Permitida'), and last seen timestamp (e.g., '15 jun. 2021 21:55:47').

ID de dispositivo	Comunicación	Visto por última vez
MSE-Shared-lock-001...	Permitida	15 jun. 2021 21:55:47
MSE-Shared-lock-002...	Permitida	27 jun. 2021 15:34:13
MSE-Shared-lock-003...	Permitida	7 jun. 2021 23:08:57

FIGURA 3.13. Administrador de dispositivos de Google Cloud IoT Core. Se muestran tres dispositivos creados para el desarrollo.

En el firmware desarrollado se incluyen los mismos datos de conexión que se observan en la imagen del administrador de dispositivos y se ubican en el archivo de cabecera del módulo mqtt_gcp.c. En el siguiente fragmento de código se pueden ver los datos y, además, se muestran los tópicos a los que se suscribe el dispositivo:

```

1 #define IOT_DEVICEID      "MSE-Shared-lock-001-15037xxxxxxxxxxxx"
2 #define IOT_REGION        "us-central1"
3 #define IOT_REGISTRY       "iris-iot-core-registry"
4 #define IOT_CLIENTID       "projects/PROJECTID/devices/" IOTCORE_DEVICEID
5 #define IOT_CONFIG_TOPIC    "/devices/" IOTCORE_DEVICEID "/config"
6 #define IOT_COMMANDS_TOPIC  "/devices/" IOTCORE_DEVICEID "/commands/#"
7

```

CÓDIGO 3.2. Datos de acceso para un dispositivo en Google IoT Core.

Para poder conectar un dispositivo a Google Cloud IoT Core se deben seguir determinadas políticas de seguridad ya que el uso de un usuario y una contraseña hoy en día es obsoleto. Una de las posibles formas de autenticación que ofrece IoT Core es mediante la utilización de *JSON Web Tokens*, cuya definición se encuentra en el RFC7519 [59] y queda fuera del alcance de este trabajo.

Los JWT se componen de tres secciones: un encabezado, una carga útil y una firma. El encabezado y la carga útil son objetos JSON que se serializan en bytes

UTF-8 [60] y luego se codifican utilizando la codificación base64url definida en el RFC4648 [61].

Para firmar y validar posteriormente el JWT, se debe generar un par de claves pública/privada. Estas claves pueden ser generadas y convertidas convenientemente con la herramienta OpenSSL [62]. La tarea puede realizarse incluso desde la propia consola de Google Cloud CLI con los comandos que se muestran en el fragmento de código 3.3.

En el caso del firmware correspondiente al dispositivo central, se utiliza la clave privada generada con el comando mostrado en la línea 2 del fragmento de código 3.3 y luego se convierte al formato PKCS8 definido en RFC5208 [63] con el comando de la línea 5 presente en el mismo fragmento.

```

1 openssl req -x509 -nodes -newkey rsa:2048 -keyout rsa_private.pem \
2   -out rsa_cert.pem -subj "/CN=unused"
3
4
5 openssl pkcs8 -topk8 -inform PEM -outform DER -in rsa_private.pem \
6   -nocrypt > rsa_private_pkcs8
```

CÓDIGO 3.3. Comandos OpenSSL para crear el par de claves pública/privada y luego convertir a formato PKCS8.

La clave privada en formato PKCS8 es almacenada en forma de constante en el módulo pemkey.h, pero en caso de producir este tipo de dispositivos a gran escala, deberá implementarse otro mecanismo que permita automatizar la tarea y evitar que las claves sean accesibles en la línea de producción. Hay distintas estrategias disponibles para lograrlo. El fragmento de código 3.4 corresponde al módulo mencionado y permite apreciar el formato de la clave:

```

1 #ifndef MAIN_PEMKEY_H_
2 #define MAIN_PEMKEY_H_
3
4 // Clave PRIVADA del dispositivo registrado en
5 // Google Cloud Platform – IoTCore
6
7 const char GCP_PEM_KEY[] =
8
9 "-----BEGIN PRIVATE KEY-----\n"
10 "MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQCy2gwQCu4raqjP\n"
11 "H1Jl5inTxt06dLVDoUEMwL58shH4vcepGTvphFhN0usuP5JGWGqUCfXZzg4lb\n"
12 "DQxIQ5Z7Q92uljypVzD6xGZTgOvxSJbaq9rHCsnO1BfNKR5wghat+WmeYQ6nyuYm\n"
13 "IUBCS8Q6N1ctp9o1imb8IK+G2Y9s2D/K0BBkvgfYDMc4wB9v4m/E1iIFB550AGk1\n"
14 "uj87L3e1zTQMXTUT+oICF3u6QIyZIkNub2T+yZGFOotGIMTDmhqRfOkPLEuxZUzE\n"
15 "RfGgK5ARxeWjNsQVVONKHaTx\n"
16 "-----END PRIVATE KEY-----\n";
```

CÓDIGO 3.4. Clave privada incluida en módulo pemkey.h, en formato PKCS8.

El JWT que finalmente se utiliza para autenticar el dispositivo en la plataforma de Google es generado por el módulo jwt.c y luego enviado al iniciar la conexión

por protocolo MQTT. En la generación del token JWT se utiliza la codificación base64url antes mencionada, de la que se ocupa el módulo base64url.c

El siguiente diagrama resume el proceso de aprovisionamiento de credenciales del dispositivo. Se supone que el aprovisionador autenticado es el usuario que configura el dispositivo, que ha creado un proyecto y un registro, y que tiene permisos para crear dispositivos. El aprovisionador puede utilizar la API de Cloud IoT Core, los comandos de gcloud o la consola de Cloud Platform para crear un dispositivo lógico en la nube.

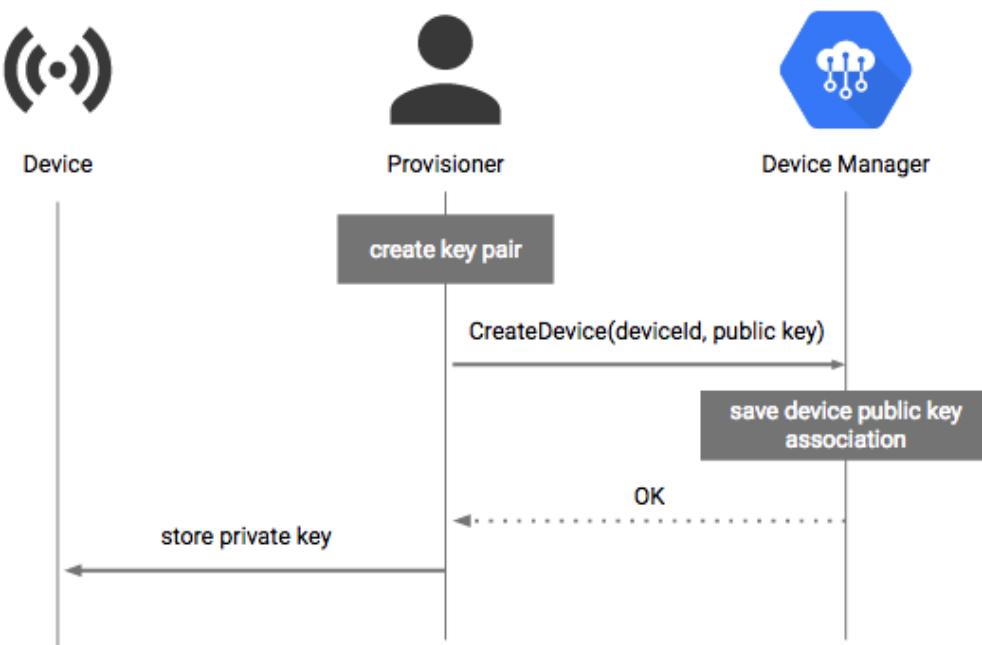


FIGURA 3.14. Proceso de aprovisionamiento de credenciales del dispositivo en Google Cloud Platform².

Conexión a Internet por red celular

Para la utilización de la red celular en el presente trabajo, se podrían haber seguido diferentes estrategias.

La tecnología GSM o también llamada 2G se encuentra actualmente en declive y de hecho algunos países ya no cuentan con cobertura para dicha red. De todas formas, por distintos motivos, fue la seleccionada para este trabajo.

En Argentina hay algunos proveedores de componentes electrónicos que ofrecen módulos que utilizan las tecnologías LTE Cat M1 o Cat NB1, pero en el momento

²https://cloud.google.com/iot/docs/concepts/device-security#public_key_format

en que se hicieron las averiguaciones correspondientes y se solicitaron presupuestos, los módulos eran demasiado costosos y lo que se pretende desde el inicio es un equipo de bajo costo. Actualmente, los precios de estas tecnologías han mejorado, pero no así la disponibilidad de kits de desarrollo.

Los módulos modernos ofrecen la posibilidad de conectarse a *brokers* MQTT directamente por medio de comandos AT, pero se cree que esta modalidad genera más limitaciones que beneficios. Por un lado, al conectarse a MQTT utilizando comandos AT, el dispositivo tendría que contar con dos bibliotecas distintas si también accede a MQTT vía Wi-Fi, como sucede en el caso de este desarrollo. Además, cuando se accede a MQTT por medio de comandos AT, se hace difícil realizar otras tareas que dependan de Internet, por ejemplo, algo tan simple como sincronizar el reloj del dispositivo utilizando SNTP.

Por los motivos mencionados, inicialmente se creyó conveniente el uso de un módulo celular con tecnología 2G y establecer una conexión a Internet utilizando el protocolo PPPoS incluido en el *stack* lwIP. De este modo se consigue una conexión totalmente transparente a Internet, sobre la cual se pueden apoyar los otros servicios de la aplicación sin necesidad de realizar adaptaciones o diferenciaciones cuando se utiliza la conexión celular o la conexión Wi-Fi. Se logra simplemente una conexión a Internet que permite acceder a cualquier servicio y utilizar cualquier protocolo.

En resumen, en la presente implementación se configuró el módem como servidor PPPoS y el kit ESP32 como cliente PPPoS y en la tabla 4.2 se detallan los principales comandos AT que se utilizaron para lograr esta configuración. Este conjunto de comandos corresponde a una selección extraída de los tantos comandos AT incluidos en los módulos de comunicación celular. Pueden resultar útiles y permitir ahorrar tiempo de estudio al abordar una solución similar.

TABLA 3.10. Principales comandos AT utilizados para establecer la conexión a Internet por PPPoS.

Comando	Función
AT	Verificar la conexión y el estado del módem.
ATE	Activar o desactivar el modo eco.
AT&W	Guardar los parámetros actuales en el perfil definido por el usuario.
AT+IFC	Determinar el tipo de control de flujo para la interfaz serial en el modo de datos.
AT+CGDCONT	Definir el contexto de PDP.
ATH	Desconectar la conexión existente.
AT+CSQ	Reportar la calidad de señal.
AT+CBC	Reportar el estado de la alimentación.
+++	Cambiar del modo de datos o modo PPP al modo de comando.
ATD*99#	Establecer una conexión con configuración de acceso telefónico * 99 #.
AT+CGMM	Retornar el texto de identificación del modelo del módulo.
AT+CGSN	Retornar el número de serie del producto.
AT+CIMI	Solicitar la identidad de suscriptor móvil internacional (SIM).
AT+COPS?	Reportar el modo actual y el operador seleccionado actualmente.
AT+CPOWD	Apagar el módulo.

De los comandos expuestos en la tabla 4.2, muchos se encuentran estandarizados por el 3GPP y todos, en general, están disponibles en los módulos más modernos. En conclusión, el hecho de haber adoptado este criterio en el desarrollo permite reemplazar el módem sin modificaciones o, eventualmente, con ajustes menores por otro de tecnología más moderna o que tenga cobertura en otros países. A su vez, evita tener que dar tratamiento diferenciado a las conexiones a Internet establecidas vía Wi-Fi o vía red celular.

3.3.2. Firmware del dispositivo lector

Como se detalló en la tabla 3.7, el firmware correspondiente al dispositivo lector se desarrolló utilizando programación bare-metal. En la figura 3.15 se observa un diagrama del flujo de ejecución del programa.

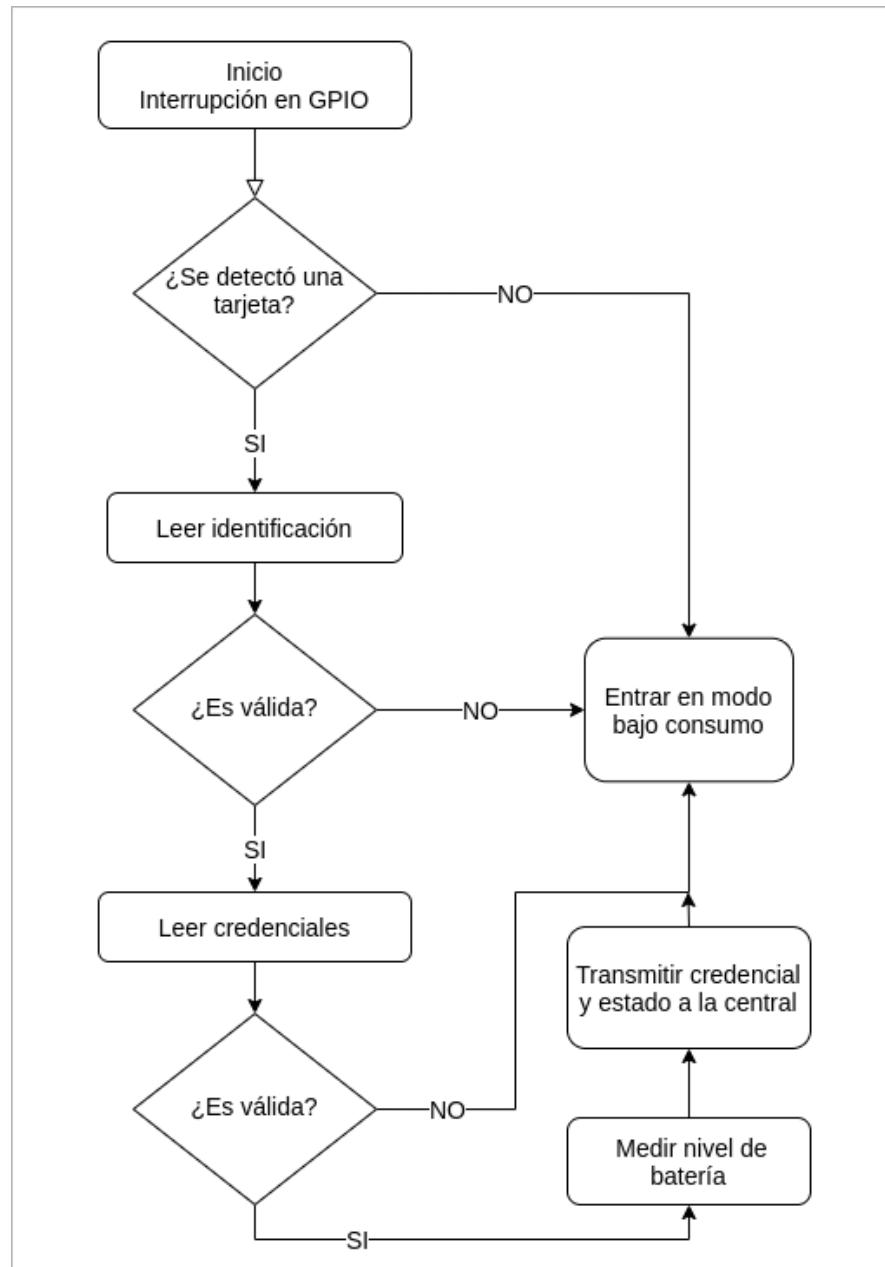


FIGURA 3.15. Diagrama de flujo del programa correspondiente al dispositivo lector. Muestra las acciones y comprobaciones realizadas por el firmware desde su inicio.

El fabricante del microcontrolador PIC16F1718 es la empresa Microchip, y para el desarrollo de este firmware se utilizó la plataforma de desarrollo oficial ofrecida por la empresa. Las herramientas utilizadas fueron MPLAB-X IDE [64] junto al

compilador específico para la línea de controladores de 8 bits, que es el XC8 [65]. El compilador MPLAB XC8 es un compilador cruzado ISO C99 optimizado para el lenguaje de programación C.

El ambiente de desarrollo integrado (IDE) permite llevar a cabo la configuración básica de los periféricos del microcontrolador y su asignación de pines de manera muy práctica. En las figuras 3.16 y 3.17 se puede observar la matriz de asignación de pines y la disposición final sobre el procesador respectivamente. A partir de esta configuración, fue generado el módulo pin_manager.c y también su archivo de cabecera. Esta misma metodología se aprovechó para configurar y generar el código necesario para los siguientes periféricos: SPI, UART y ADC. La herramienta permitió ahorrar tiempo y evitar errores.

Package:	SOIC28	Pin No:	2	3	4	5	6	7	10	9	21	22	23	24	25	26	27	28	11	12	13	14	15	16	17	18	1
Module	Function	Direction	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	E
EUSART ▼	RX	input									■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	
	TX	output									■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	
MSSP ▼	SCK	in/out									■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	
	SDI	input									■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	
OSC	SDO	output									■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	
	CLKOUT	output									■																
Pin Module ▼	GPIO	input	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	
	GPIO	output	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	
RESET	MCLR	input																									■
RN4870 RN487...	BT_MODE	output	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	
	BT_RST	output	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	
	BT_RX_IND	output	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	

FIGURA 3.16. Herramienta para configurar asignación de pines del microcontrolador en MPLAB-X.

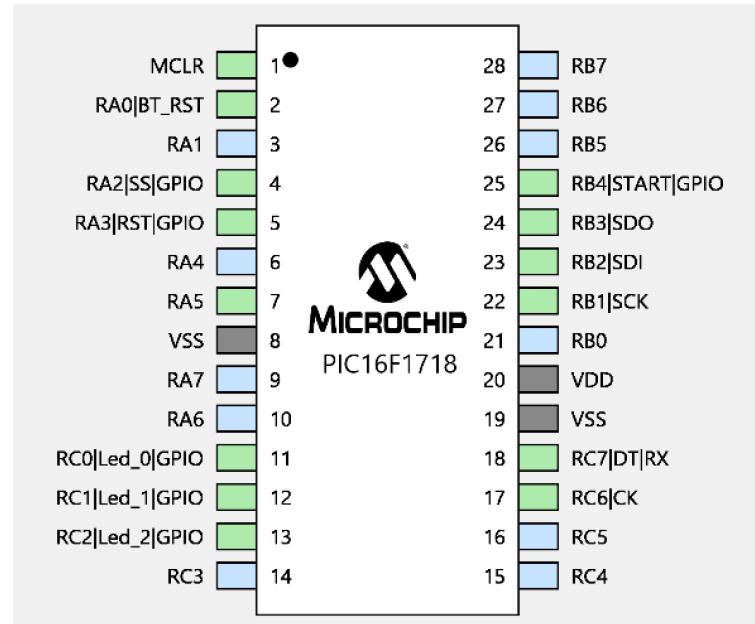


FIGURA 3.17. Asignación de pines del PIC16F1718. Configuración realizada con Microchip Code Configurator.

En la tabla 3.11 se detallan los principales módulos que conforman el firmware del dispositivo lector y se describe brevemente la función de cada uno. Los componentes que más tiempo de desarrollo insumieron fueron los manejadores o *drivers* del transceptor BLE RN4870 y del lector/grabador de tarjetas MIFARE MRFC-522, se trata de dispositivos que cuentan con una importante cantidad de comandos y parámetros para su funcionamiento.

TABLA 3.11. Principales módulos de firmware que conforman al dispositivo lector.

Nombre del módulo	Función
main.c	Da inicio a la aplicación, realiza seteos iniciales y lanza secuencialmente los procesos necesarios.
spi.c	Configura el periférico SPI y expone la interfaz necesaria para su uso.
uart.c	Configura el periférico UART y expone la interfaz necesaria para su uso.
adc.c	Configura el periférico ADC y expone la interfaz necesaria para su uso.
interrupt.c	Configura el vector de interrupciones y les da tratamiento.
pin_manager.c	Determina la asignación de pines para los periféricos del microcontrolador y configura las funciones de los pines de entrada/salida.
device_manager.c	Setea la frecuencia de reloj y bits de configuración para el funcionamiento adecuado del controlador.
delay.c	Proporciona funciones de temporización.
driver_RN4870.c	Implementa el manejador para el dispositivo BLE RN4870.
driver_MRFC522.c	Implementa el manejador para el dispositivo MRFC522.

3.4. Software

En esta sección se describen algunos puntos destacados respecto al desarrollo del *backend* y el *frontend* del sistema.

3.4.1. Interfaz de programación de aplicaciones (API *backend*)

El *backend* del sistema está formado por una serie de APIs (por las siglas de *Application Programming Interfaces*) que permiten establecer la lógica de las operaciones que se ejecutan y facilitan también la comunicación entre diferentes actores, como ser el *frontend* o aplicación celular, la base de datos y otros servicios de la infraestructura en la nube que se utiliza.

El lenguaje de programación que se utilizó fue C#, un lenguaje de programación moderno, basado en objetos y fuertemente tipado. Incluye la sintaxis LINQ (*Language Integrated Query*) que crea un patrón común para trabajar con datos de cualquier origen. La compatibilidad del lenguaje con las operaciones asincrónicas proporciona un entorno adecuado para el uso que se le da en este trabajo.

Los programas de C# se ejecutan en .NET, un sistema de ejecución virtual denominado *Common Language Runtime* (CLR) y un conjunto de bibliotecas de clases. En el caso de esta implementación, el software compilado se ejecuta en el servidor de aplicaciones en la nube Microsoft Azure App Service que se describió en la sección 2.9.2.

El *deploy* o despliegue de la aplicación se realiza en forma continua desde el repositorio de GitHub y el código se compila directamente en la nube, utilizando la versión del framework DotNet que se haya configurado previamente.

Dentro de la estructura de la API, podemos destacar como principales elementos a los modelos, controladores y *endpoints*. Sintéticamente, se puede decir que los modelos definen a las entidades u objetos que se relacionan dentro de la API, los controladores determinan la forma en que se comportan, y los *endpoints* son las URLs de una API o *backend* que responden a las peticiones recibidas.

En la tabla 3.12 se detallan los modelos que fueron definidos en el *backend* y se describe brevemente qué representa cada uno. En el fragmento de código 3.5 se observa la definición del modelo AcDevice. A modo de ejemplo, se pueden observar sus propiedades, el constructor, y el decorador que define el nombre de la tabla asociada dentro de la base de datos.

TABLA 3.12. Principales modelos definidos dentro de la API del *backend* y breve descripción de sus funciones.

Nombre del modelo	Descripción
AcDdevice	Su nombre se origina de <i>Access Control Device</i> y define a las propiedades que representan a un dispositivo de control de acceso dentro del sistema.
AcDeviceCommand	Su nombre se origina de <i>Access Control Device Command</i> y define a una entidad que representa a un comando que el dispositivo IoT aceptará.
AcDeviceAssignment	Define las propiedades de una entidad que forma parte de un control de acceso y que determina a qué usuarios se encuentra asignado y de qué manera.
RfidCard	Define las propiedades de una entidad que representa a las tarjetas o llaveros MIFARE que permitirán acceder a una zona. Una colección de estas entidades se encontrará incluida dentro de las propiedades de cada control de acceso.
AcdSharingInvitation	Define las propiedades de una entidad que representa a las invitaciones que serán enviadas para compartir un control de acceso. Una colección de estas entidades se encontrará incluida dentro de las propiedades de cada control de acceso. Las invitaciones podrán ser aceptadas, ignoradas o rechazadas.

```

1 [Table("ac_devices")]
2 public class AcDevice
3 {
4     public long Id { get; set; }
5     public DateTime CreateDate { get; set; }
6     public string Name { get; set; }
7     public string BLE_MAC { get; set; }
8     public virtual ICollection<RfidCard> RfidCards { get; set; }
9     public virtual ICollection<AcDeviceAssignment>
10        AcdDeviceAssignments { get; set; }
11
12     public AcDevice()
13     {
14         CreateDate = DateTime.Now;
15     }
16 }
```

CÓDIGO 3.5. Modelo AcDevice perteneciente a la API del backend del sistema de control de acceso.

Cada uno de los modelos tiene un controlador asociado que incluye a determinados *endpoints* y estos últimos se detallan en la tabla 3.13.

TABLA 3.13. *Endpoints* incluidos en las APIs del *backend* y detalle de los métodos que aceptan.

Endpoint	Métodos admitidos
..api/AcDevicesAssignments	GET - POST - DELETE
..api/AcDevicesAssignments/{id}	GET
..api/AcDevices	GET - POST - DELETE
..api/AcDevices/{id}	GET
..api/AcDevices/SendCommand	POST
..api/AcDeviceSharingInvitation	GET - POST - DELETE
..api/AcDeviceSharingInvitation/{id}	GET
..api/AcDeviceSharingInvitation/Accept	POST
..api/AcDeviceSharingInvitation/Reject	POST
..api/RfidCards	GET - POST - DELETE

Por último, el fragmento de código 3.6 ejemplifica cómo un controlador de la API del *backend* interactúa con las APIs de Google Cloud IoT Core, para que esta última envíe el comando deseado hacia el dispositivo conectado por MQTT.

```

1 Google_IoT.Google_IoT_Client GCC = new App_Code.Google_IoT.
2   Google_IoT_Client(
3     "sensores-iot-i-v1",
4     "us-central1",
5     "i-iot-core-registry",
6     acDeviceCommand.DeviceName);
7
8 GCC.ClientInitialize();
9 if (!GCC.SendCommandToDevice(JsonSerializer.Serialize(acDeviceCommand)))
10   return BadRequest();

```

CÓDIGO 3.6. Envío de un comando desde la API del *backend* hacia la API de Google Cloud IoT Core.

3.4.2. Implementación de la base de datos

Además de la base de datos Google Cloud BigQuery que se empleó para registrar eventos que podrían escalar rápidamente en volumen, se utilizó otra para contribuir con la gestión y administración de los dispositivos. Se trata de la base de datos con la que interactúa en forma directa la API del *backend* y se implementó con un motor MySQL Server versión 5.7 que es del tipo relacional.

El modelo de datos se fue definiendo de forma automática al momento de llevar a cabo el desarrollo de la API, se utilizó el enfoque *Code first* que crea las tablas y relaciones a partir de los modelos creados en el código. En la figura 3.18 se puede ver una primera versión del diagrama entidad-relación que representa al modelo.

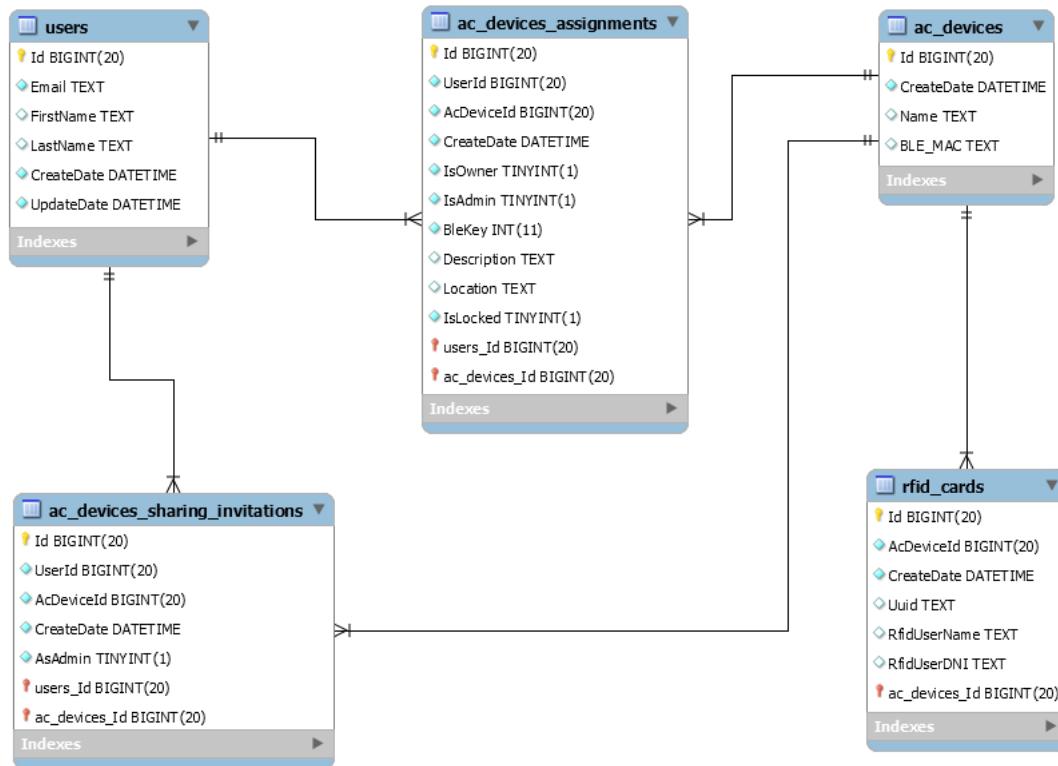


FIGURA 3.18. Diagrama entidad-relación utilizado en la base de datos para administración de dispositivos.

3.4.3. Aplicación para teléfono celular (*frontend*)

El *frontend* es el componente que brinda una interfaz gráfica al usuario a través de la cual se interactúa con el sistema, ya sea para administrar sus controles de acceso, agregar dispositivos, gestionar tarjetas de acceso o compartir una zona. Para interactuar con el sistema realiza peticiones e intercambia información a través de las APIs del *backend*.

Esta interfaz se decidió implementar en una aplicación celular lanzada en principio para dispositivos Android, pero preparada para convertirse en multiplataforma en un futuro cercano. Para la escritura del código fuente, se utilizó el IDE Android Studio, en el que se instalaron el framework Flutter y todas las dependencias necesarias para programar en lenguaje Dart.

La autenticación de la aplicación fue implementada utilizando el servicio en la nube Microsoft Active Directory B2C, lo que asegura robustez, seguridad y mantenibilidad. Realizar un sistema de autenticación a gran escala no resulta una tarea sencilla, por lo que se decidió emplear esta tecnología ya ampliamente probada. La plataforma de Microsoft brinda también varias alternativas para autenticación multifactor e incluso permite acceder a través de otras plataformas, como Gmail y Facebook.

El uso de los *widgets*

En Flutter cada elemento de la pantalla es un *widget*, siendo estos los bloques sobre los cuales se construye toda la aplicación a nivel gráfico. Para construir una pantalla resulta necesario anidar los *widgets* necesarios con el objetivo final de formar un *layout*. En la figura 3.19 se muestra a modo de ejemplo un diagrama de árbol que forma el *layout* de una pantalla específica de una interfaz de usuario.

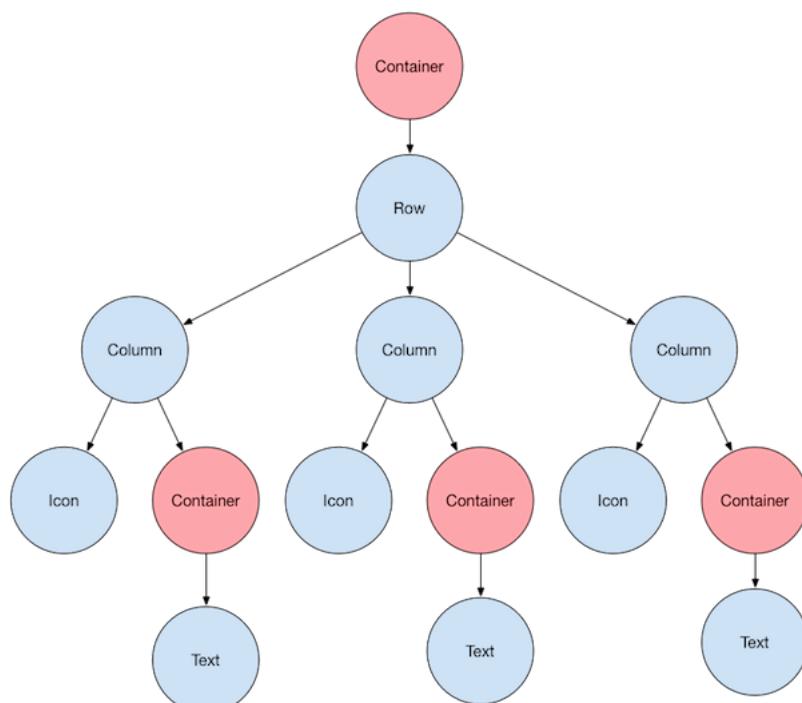


FIGURA 3.19. Diagrama de árbol que conforma el layout de una pantalla en Flutter³.

³<https://flutter.dev/docs/development/ui/layout>

El código 3.7 corresponde a un fragmento tomado del módulo device_add_rfidCard.dart, destinado a generar la pantalla que permite registrar tarjetas RFID en el dispositivo. En este fragmento tomado de ejemplo se pueden observar algunos *widgets* anidados que a su vez forman un *widget* llamado “PressableCard”. Dentro de este último se anidan todos aquellos que dentro del código se ven precedidos por la palabra “child”, como por ejemplo el “Container”, que incluye a un hijo “Padding” y este a un hijo “Center”, y así sucesivamente.

```

1  @override
2  Widget build(context) {
3      return PressableCard(
4          color: Colors.green,
5          flattenAnimation: AlwaysStoppedAnimation(0),
6          child: Stack(
7              children: [
8                  Container(
9                      height: 120,
10                     width: 250,
11                     child: Padding(
12                         padding: EdgeInsets.only(top: 40),
13                         child: Center(
14                             child: Text(
15                                 content,
16                                 style: TextStyle(fontSize: 48),
17                             ),
18                         ),
19                     ),
20                 ),
21                 Positioned(
22                     top: 0,
23                     left: 0,
24                     right: 0,
25                     child: Container(
26                         color: Colors.black12,
27                         height: 40,
28                         padding: EdgeInsets.only(left: 12),
29                         alignment: Alignment.centerLeft,
30                         child: Text(
31                             header,
32                             style: TextStyle(
33                                 color: Colors.white,
34                                 fontSize: 14,
35                                 fontWeight: FontWeight.w600,
36                             ),
37                         ),
38                     ),
39                 ),
40             ],
41         ),
42         onPressed: () {
43             showChoices(context, preferenceChoices);
44         },

```

CÓDIGO 3.7. Ejemplo de widgets anidados en la pantalla para agregar tarjetas RFID.

En la tabla 3.14 se detallan los principales módulos desarrollados en la aplicación celular y se describe la función de cada uno. Luego, en la figura 3.20, se mostrarán las capturas de las pantallas que hayan sido mencionadas en la tabla e identificadas con una letra entre paréntesis.

TABLA 3.14. Principales módulos que componen la aplicación del *frontend* y breve descripción de sus funciones.

Nombre del módulo	Finalidad
main.dart	Da inicio a la aplicación invocando a “runApp()” dentro de la función “main()”.
bluetooth.dart	Se ocupa de configurar el dispositivo BLE del celular y de establecer la conexión con la central del sistema de acceso para solicitar el desbloqueo de la zona controlada.
messages.dart	Es un módulo de uso común dentro de la aplicación, tiene como objeto mostrar alertas y mensajes con diferentes formatos preestablecidos en dispositivo móvil.
iris_api_client.dart	Este módulo incluye las funciones necesarias para interactuar con las APIs del <i>backend</i> del sistema.
profile_tab.dart	Genera la pantalla que muestra los datos del usuario (B) y permite iniciar el proceso de <i>login</i> (A).
device_add.dart	Genera la pantalla que permite agregar un dispositivo a la cuenta del usuario a través del escaneo de un código QR (D).
device_add_rfidCard.dart	Genera la pantalla que permite autorizar a una tarjeta MIFARE para que sea aceptada por un determinado control de acceso. La tarjeta se autoriza por medio del ingreso de su UUID.
device_details.dart	Genera la pantalla que muestra los detalles de un dispositivo y permite eliminarlo (E). La pantalla incluye además otras solapas que permiten compartir el control de acceso y gestionar las tarjetas MIFARE autorizadas (F).

Las imágenes que siguen a continuación corresponden a distintas pantallas de la aplicación que fueron mencionadas en la tabla 3.14. En este caso fueron capturadas del simulador que se utilizó para el desarrollo.

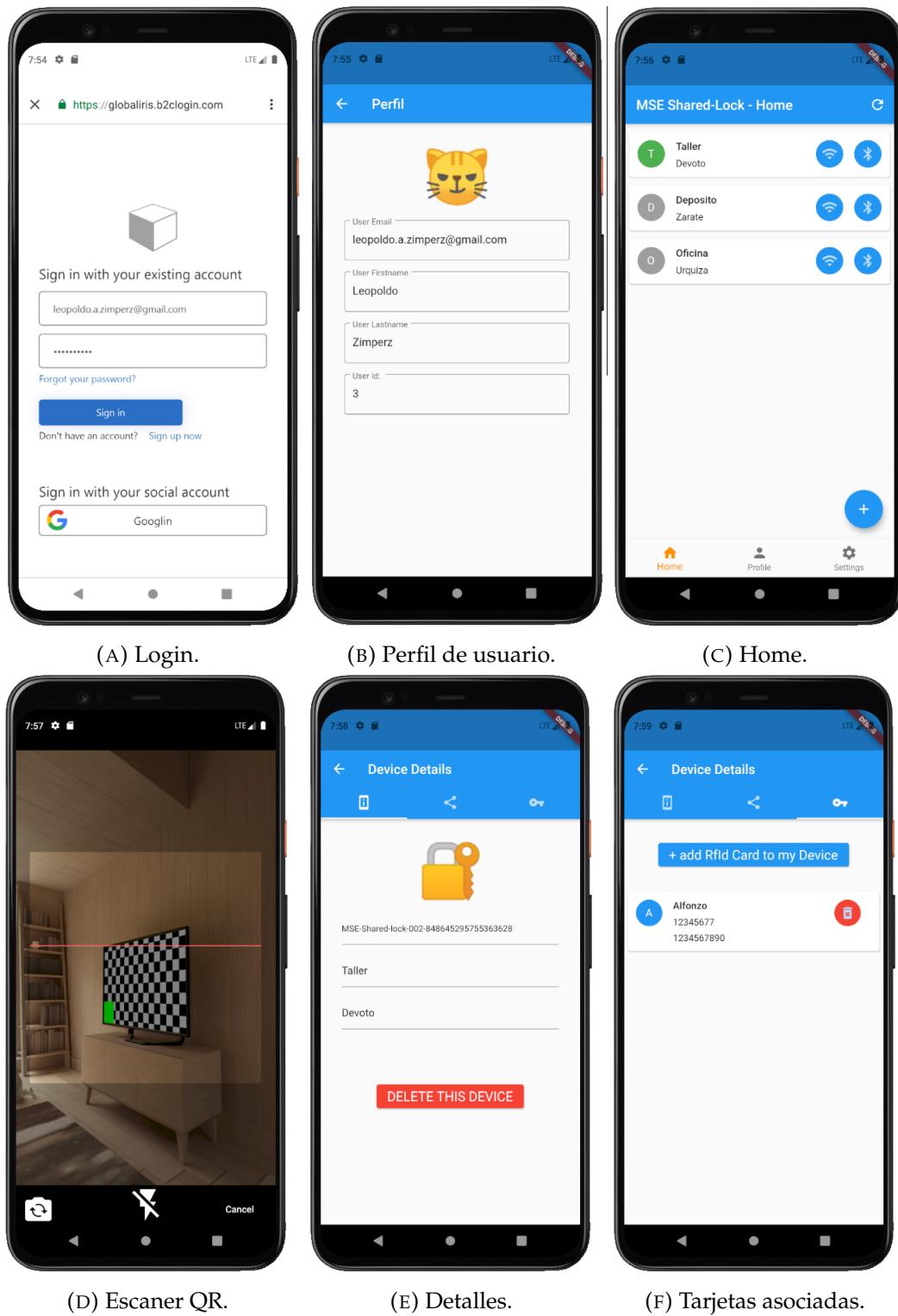


FIGURA 3.20. Capturas de pantalla de la aplicación celular.

Capítulo 4

Ensayos y resultados

En el presente capítulo se da una visión global sobre las distintas pruebas realizadas y los resultados obtenidos. Se explican las herramientas que se utilizaron para los test unitarios, el modo en que fueron implementados los test funcionales de firmware y hardware, y las pruebas de sistema. Se abordan algunos ejemplos relevantes aplicados en distintos subsistemas.

4.1. Testeos unitarios

Si bien en el presente capítulo no se abordan los testeos unitarios en profundidad, se presentan las herramientas estudiadas y probadas durante el desarrollo del firmware. Se realizaron test unitarios sobre distintos elementos del sistema.

Un test unitario puede describirse como una forma de comprobar el correcto funcionamiento de una unidad de código. La unidad de código puede referirse a una función, un método, o una clase. Mediante este tipo de test se comprueba que el fragmento de código devuelva el resultado esperado, en forma eficiente, pero aislada. Por cada módulo de firmware en los que se realiza este tipo de ensayo, se desea comprobar el comportamiento de todas las funciones externas, y no de las internas. Las funciones internas definidas con el prefijo “static” deben ser verificadas en forma indirecta a través de las funciones externas evaluadas.

La plataforma de desarrollo ESP-IDF de Espressif ofrece dos posibilidades para realizar testeo de firmware y son las siguientes:

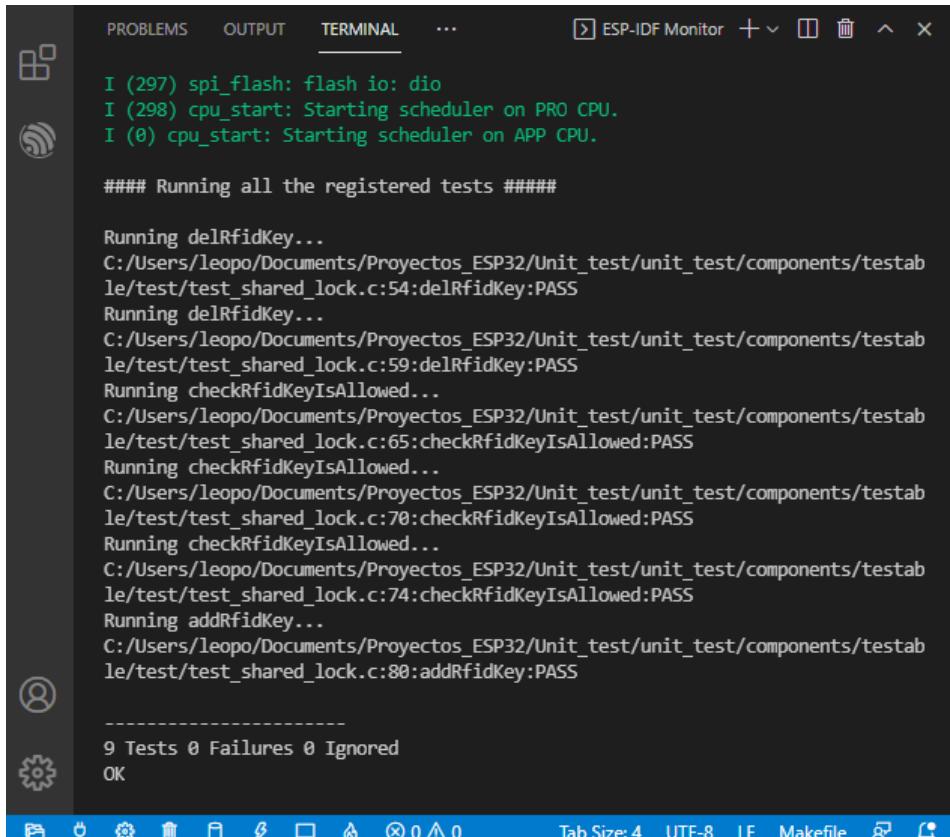
- Aplicación para testeos unitarios: en este caso se implementa una aplicación destinada a test, que se ejecuta directamente en el dispositivo de destino o *target*.
- Ambiente Linux: se trata de pruebas llevadas a cabo dentro de un sistema operativo Linux en las que todo el *hardware* se abstrae por medio de simulaciones o *mocks*. Esta alternativa aún cubre solo una pequeña fracción de los componentes incluidos en la plataforma de Espressif.

Para poder realizar pruebas unitarias en el dispositivo central se optó por la primera alternativa, la implementación de una aplicación que se ejecute directamente sobre el módulo ESP32, tal como lo haría en la aplicación final, pero de manera aislada. Para permitir esto, la plataforma ESP-IDF integra a los *frameworks* Unity [66] y CMock [67].

Unity ofrece las funciones necesarias para implementar distintos test, como por ejemplo TEST_CASE(), TEST_ASSERT() o TEST_ASSERT_EQUAL_FLOAT() y otras, que permiten verificar que una función determinada devuelva el resultado esperado.

CMock es un *framework* que permite generar cierto grado de simulación, para dar respuesta a uno de los mayores problemas que aparecen al realizar pruebas unitarias en sistemas embebidos: la fuerte dependencia con el hardware. En muchos casos sucede que hay problemas de temporizaciones, *timeouts*, interrupciones, y situaciones que no permiten aislar a una determinada función y que esta se pueda ejecutar correctamente. En estos casos, con el uso de CMock, se pueden simular funciones o procesos para hacer posible el correcto aislamiento y posterior testeo de un elemento. También es válida esta alternativa cuando los archivos testeados incluyen a otros que están fuera del alcance y no serían objetivo del test. Esto puede ocurrir cuando existen dependencias con bibliotecas de terceros.

En las figuras 4.1 y 4.2 se observan resultados del testeo realizado en el módulo shared-lock.c. En la figura 4.1 se muestra la ejecución automática y continua de todos los test correspondientes al módulo, junto al resumen de resultados. En la figura 4.2 se muestra una opción de testeo interactiva, en la que se muestra un listado de test y se selecciona el deseado. Resulta útil cuando el número de testeos es elevado y se intenta corregir un problema puntual. Estas capturas fueron tomadas desde la consola del Visual Studio Code y muestran información recibida en tiempo real desde la aplicación que se ejecuta dentro del módulo ESP32 con fines de testeo únicamente.



```

PROBLEMS    OUTPUT    TERMINAL    ...
ESP-IDF Monitor + ▾ □ ⌂ ^ ×

I (297) spi_flash: flash io: dio
I (298) cpu_start: Starting scheduler on PRO CPU.
I (0)  cpu_start: Starting scheduler on APP CPU.

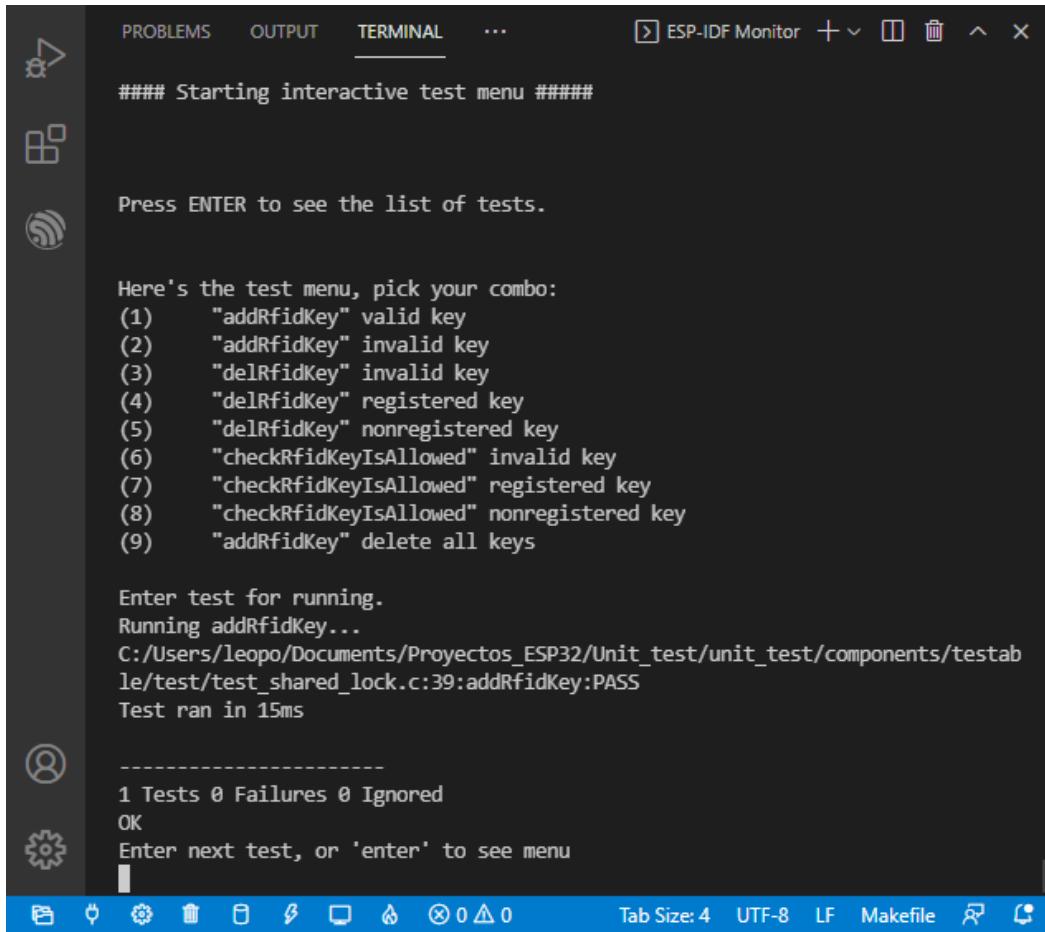
#### Running all the registered tests ####

Running delRfidKey...
C:/Users/leopo/Documents/Proyectos_ESP32/Unit_test/unit_test/components/testable/test/test_shared_lock.c:54:delRfidKey:PASS
Running delRfidKey...
C:/Users/leopo/Documents/Proyectos_ESP32/Unit_test/unit_test/components/testable/test/test_shared_lock.c:59:delRfidKey:PASS
Running checkRfidKeyIsAllowed...
C:/Users/leopo/Documents/Proyectos_ESP32/Unit_test/unit_test/components/testable/test/test_shared_lock.c:65:checkRfidKeyIsAllowed:PASS
Running checkRfidKeyIsAllowed...
C:/Users/leopo/Documents/Proyectos_ESP32/Unit_test/unit_test/components/testable/test/test_shared_lock.c:70:checkRfidKeyIsAllowed:PASS
Running checkRfidKeyIsAllowed...
C:/Users/leopo/Documents/Proyectos_ESP32/Unit_test/unit_test/components/testable/test/test_shared_lock.c:74:checkRfidKeyIsAllowed:PASS
Running addRfidKey...
C:/Users/leopo/Documents/Proyectos_ESP32/Unit_test/unit_test/components/testable/test/test_shared_lock.c:80:addRfidKey:PASS

-----
9 Tests 0 Failures 0 Ignored
OK

```

FIGURA 4.1. Ejecución automática de los testeos correspondientes al módulo shared-lock.c



The screenshot shows a terminal window titled "ESP-IDF Monitor" with the following content:

```
PROBLEMS OUTPUT TERMINAL ...
ESP-IDF Monitor + - ^ X

#### Starting interactive test menu #####
Press ENTER to see the list of tests.

Here's the test menu, pick your combo:
(1) "addrfidkey" valid key
(2) "addrfidkey" invalid key
(3) "delrfidkey" invalid key
(4) "delrfidkey" registered key
(5) "delrfidkey" nonregistered key
(6) "checkRfidKeyIsAllowed" invalid key
(7) "checkRfidKeyIsAllowed" registered key
(8) "checkRfidKeyIsAllowed" nonregistered key
(9) "addrfidkey" delete all keys

Enter test for running.
Running addRfidKey...
C:/Users/leopo/Documents/Proyectos_ESP32/Unit_test/unit_test/components/testable/test_shared_lock.c:39:addRfidKey:PASS
Test ran in 15ms

-----
1 Tests 0 Failures 0 Ignored
OK
Enter next test, or 'enter' to see menu
```

The terminal window has a dark background with light-colored text. It includes standard terminal navigation keys at the bottom and a vertical toolbar on the left with icons for file operations, terminal settings, and help.

FIGURA 4.2. Ejecución interactiva de los testeos correspondientes al módulo shared-lock.c

4.2. Testeos funcionales y pruebas de sistema

Con el objetivo de mostrar los ensayos de una manera organizada el sistema de control de accesos fue dividido convenientemente. La primera gran división se basó en las siguientes áreas: *hardware* y *firmware*, *backend*, *frontend*, y por último pruebas de sistema. En segunda instancia, el área correspondiente a hardware y firmware, se subdividió en los siguientes cuatro subsistemas: operaciones con tarjetas MIFARE, comunicaciones Bluetooth Low Energy, comunicaciones por red celular y conexión al servicio Google Cloud IoT Core.

En la tabla 4.1 se detallan varios casos de prueba a modo de ejemplo y se indica el área o subsistema en el que se consideró a cada uno. Para realizar los ensayos se utilizaron los distintos bancos de prueba que se esquematizan en la figura 4.3.

En la columna EBP (Esquemas de Bancos de Prueba) de la tabla 4.1 se hace referencia al número de identificación de cada uno de los esquemas detallados en la figura 4.3.

TABLA 4.1. Organización de los ensayos realizados.

Test	Área	Caso de prueba	EBP
1.1	MIFARE	Tarjeta vacía detectada.	1
1.2	MIFARE	Tarjeta bloqueada detectada.	1
1.3	MIFARE	Tarjeta con token válido detectada.	1
2.1	BLE	Emitir <i>Advertising</i> desde ESP32 y buscar el dispositivo con aplicación celular.	2
2.2	BLE	Conectar el celular al ESP32 vía BLE y solicitar listado de servicios y características.	2
2.3	BLE	Conectar el celular al ESP32 vía BLE y leer características BLE.	2
3.1	BLE	Conectar módulo RN4870 al ESP32 por MAC vía BLE utilizando comandos AT.	3
3.2	BLE	Conectar módulo RN4870 al ESP32 vía BLE y solicitar servicios y características.	3
3.3	BLE	Conectar el módulo RN4870 al ESP32 vía BLE, leer y escribir características.	3
4.1	GSM	Conectar el módulo SIM800 a una red celular utilizando comandos AT y realizar una solicitud GET.	4
5.1	GSM	Conectar el dispositivo central a Internet utilizando el módulo SIM800C como módem.	5
6.1	GCIOT	Conectar el módulo ESP32 al servicio Google IoT Core.	6
6.2	GCIOT	Enviar un comando desde Google IoT Core hacia el módulo ESP32.	6
7.1	Backend	Realizar testeo automatizado de <i>endpoints</i> de las APIs del <i>backend</i> .	7
8.1	Frontend	Intentar acceder a la aplicación celular con credenciales incorrectas.	8
8.2	Frontend	Intentar acceder a la aplicación celular con credenciales correctas.	8
9.1	Sistema	Realizar pruebas de sistema.	9

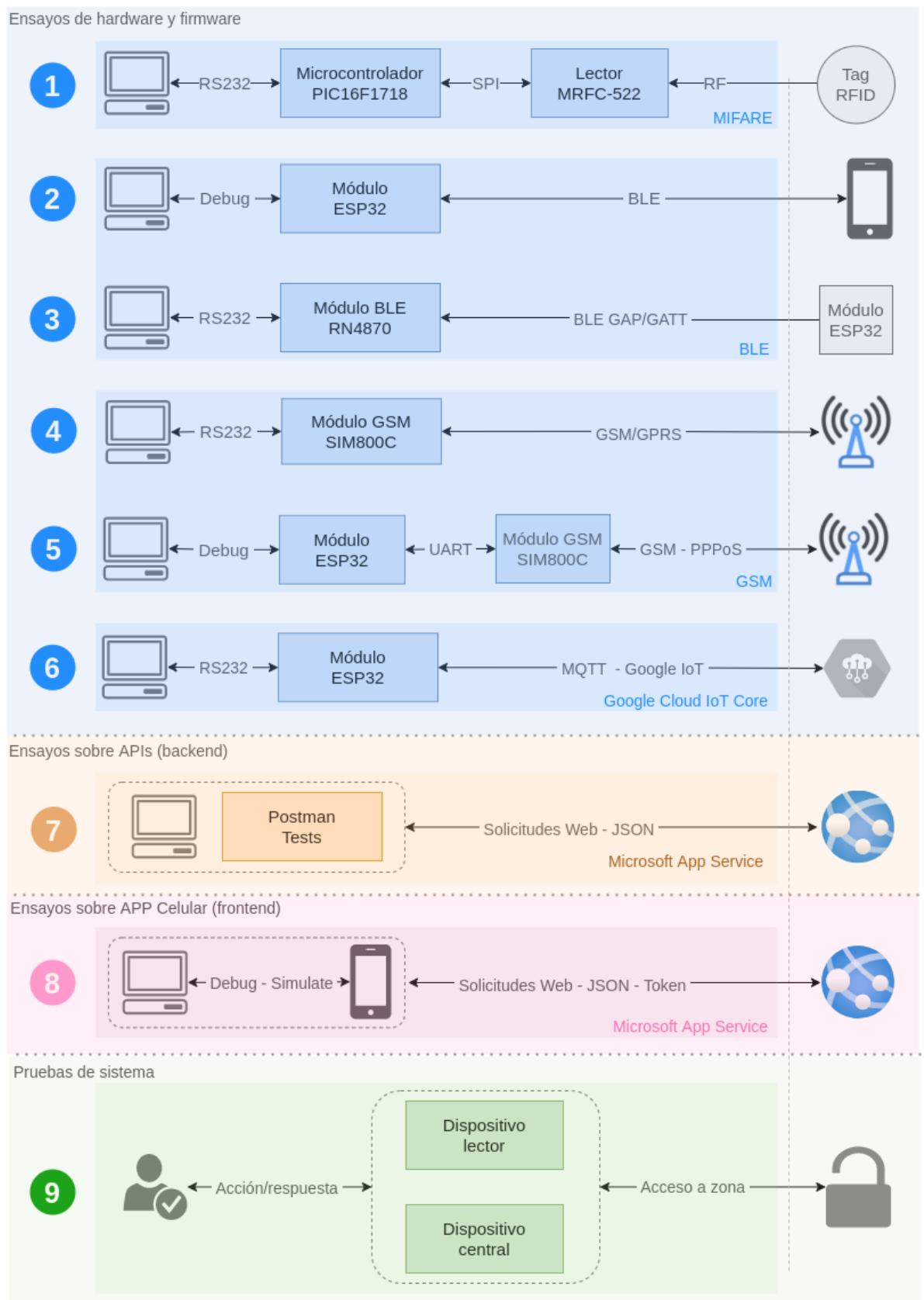


FIGURA 4.3. Esquemas de bancos de prueba para los distintos niveles de ensayos realizados.

4.2.1. Ensayos sobre firmware y hardware

Subsistema MIFARE, test de operaciones con tarjetas RFID

En el caso del subsistema que lleva a cabo las operaciones relacionadas con las tarjetas MIFARE, se utilizó el banco de pruebas físico que se muestra en la figura 4.4. Este banco fue esquematisado en la figura 4.3 e identificado allí con el número 1. Incluye un microcontrolador PIC16F1718 y un módulo lector/grabador de tarjetas RFID modelo MRFC-522, que se encuentran montados en un circuito impreso experimental.

El módulo de firmware correspondiente al *driver* del lector de tarjetas fue cargado en la memoria del microcontrolador, junto a una pequeña función que se encarga de instanciarlo y ejecutar las funciones necesarias. Se encuentra conectado a la computadora a través de un convertidor USB-RS232. En la computadora se utiliza una terminal serie para observar los datos enviados por el dispositivo. Para el subsistema bajo prueba, se muestran los resultados de los testeos 1.1, 1.2 y 1.3 que se detallan en la tabla 4.1 y se describe brevemente el procedimiento realizado en cada caso.

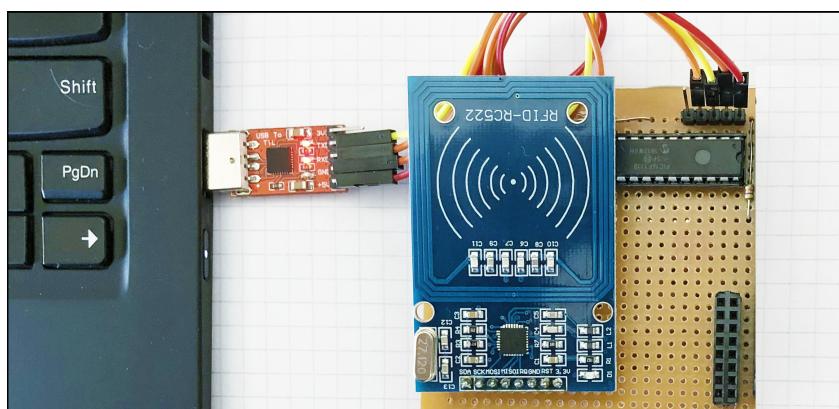


FIGURA 4.4. Banco de pruebas físico para testeos 1.x.

Test 1.1: se aproxima al lector una tarjeta MIFARE vacía, sin haber sido utilizada en ningún momento. Cuando el lector la detecta lee su UUID y, al ser válido, intenta leer el token que utiliza el sistema con el uso de las claves necesarias. En esta última operación detecta que la tarjeta no se encuentra bloqueada y, entonces, la inicializa y la bloquea. En la figura 4.5 se muestra la secuencia de pasos recibida en la terminal.

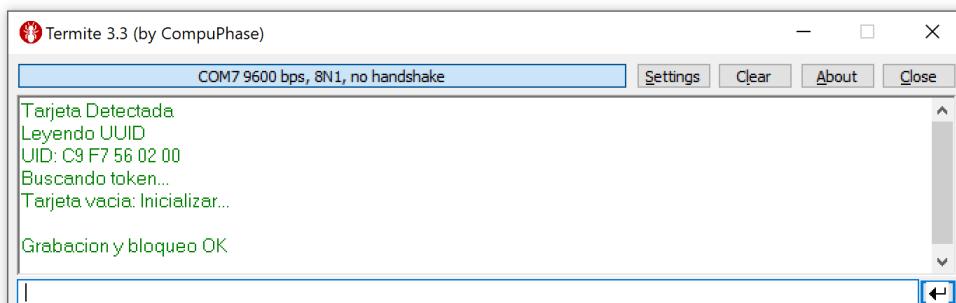


FIGURA 4.5. Test de lectura de una tarjeta MIFARE vacía.

Test 1.2: se aproxima al lector una tarjeta MIFARE bloqueada con una clave desconocida e inaccesible para el sistema. Cuando el lector la detecta lee su UUID y, al ser válido, intenta leer el token que utiliza el sistema con el uso de las claves necesarias. En esta última operación obtiene un error, porque esa tarjeta ya fue utilizada por otro sistema y no es apta para su uso por estar bloqueada. En la figura 4.6 se muestra la secuencia de pasos recibida en la terminal.

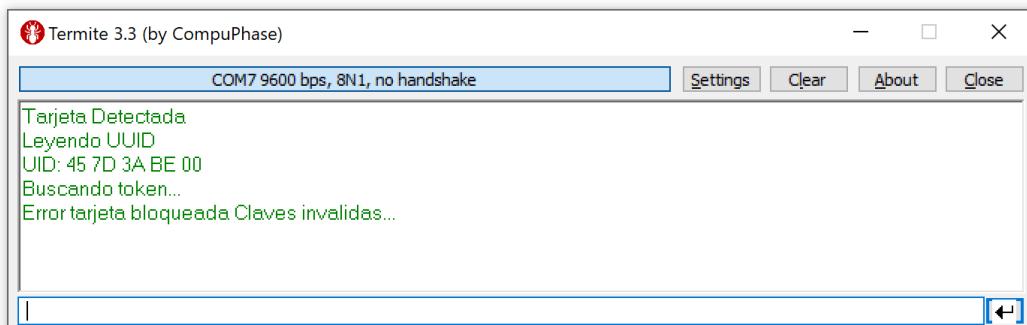


FIGURA 4.6. Test de lectura de una tarjeta MIFARE bloqueada.

Test 1.3: se aproxima al lector la tarjeta MIFARE que fue inicializada por el sistema en el test 1.1. Cuando el lector la detecta lee su UUID y, al ser válido, intenta leer el token que utiliza el sistema con el uso de las claves necesarias. En esta última operación obtiene acceso a la tarjeta y puede leer y validar el token grabado en su memoria. En la figura 4.7 se muestra la secuencia de pasos recibida en la terminal.

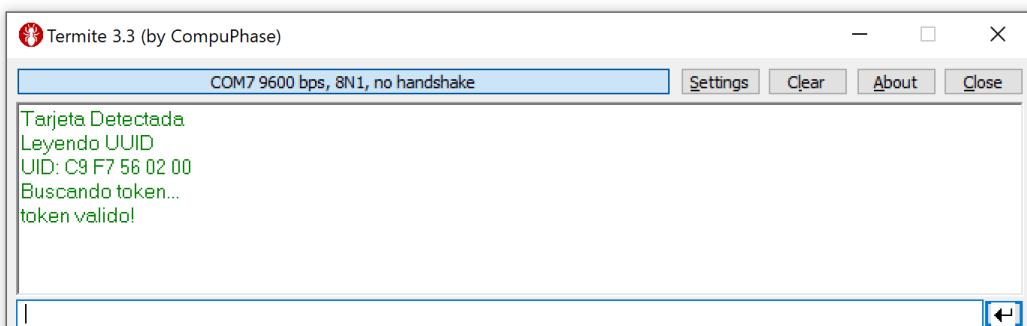


FIGURA 4.7. Test de lectura de una tarjeta MIFARE válida.

Subsistema Bluetooth Low Energy

En el caso del subsistema que se encarga de las comunicaciones vía BLE se utilizaron dos bancos de pruebas distintos. Para los testeos 2.1, 2.2 y 2.3 detallados en la tabla 4.1 se utilizó como banco de pruebas físico un módulo ESP32-DevKitC conectado por puerto USB a la PC y un celular con la aplicación BLE Scanner instalada. Es el banco de pruebas esquematizado en la figura 4.3 e identificado allí con el número 2.

En el kit ESP32 se instaló el firmware, desarrollado para el proyecto, que se ocupa de atender las conexiones Bluetooth Low Energy dentro del dispositivo central.

La PC conectada por USB al kit ESP32 muestra en la terminal de Visual Studio Code la información que este le envía. A continuación se exponen los resultados de los tres testeos mencionados y se describe brevemente el procedimiento realizado en cada caso.

Test 2.1: al iniciar la aplicación en el módulo ESP32 se genera la tabla de servicios y características BLE y, luego, el transceptor comienza a emitir la señal conocida como *advertising*. En la figura 4.8 se puede identificar con el número 1, el momento en que se genera la tabla de atributos. Posteriormente, con la identificación número 2, se observa que el dispositivo inicia la transmisión de la señal de *advertising*. Esta señal es captada por la aplicación de escáner BLE, instalada en el celular, y en la parte (A) de la figura 4.9 se puede observar que el dispositivo fue encontrado. El nombre del dispositivo se resalta en color rojo: ESP32.

Test 2.2: una vez finalizado el test 2.1 se presiona el botón *Connect* del escáner BLE y se establece una conexión entre el teléfono y el kit ESP32. Luego, al tocar en la pantalla el link que muestra los servicios BLE del dispositivo escaneado, el teléfono solicita al kit ESP32 información sobre los servicios y las características que ofrece. El momento en que se establece la conexión se puede ver identificado con el número 3, en la pantalla de la terminal de la figura 4.8, y allí se detalla la MAC del celular que fue conectado. Desde el lado del celular, en la parte (B) de la figura 4.9, se ve una captura de pantalla en donde se detallan los servicios y las características relevadas.

Test 2.3: finalmente con este test se da cierre al grupo 2.x. Para realizar este ensayo se aprovechó el test 2.2, que dejó una conexión establecida entre el celular y el kit ESP32. Desde el escáner celular se intentan leer y escribir características BLE y esto se ve reflejado, por un lado en la figura 4.8 con las identificaciones número 3 y 4, y por otro lado en la parte (C) de la figura 4.9.

```

PROBLEMS 4 OUTPUT TERMINAL DEBUG CONSOLE

I (962) NO SEC -> Device bt addr: : fc f5 c4 0e cc 62
1 I (972) BLE_GATTS: create attribute table successfully, the number handle = 8

2 I (972) BLE_GATTS: SERVICE_START_EVT, status 0, service_handle 40
I (972) BLE_GATTS: Actualizando valor CHAR_A
I (982) BLE_GATTS: advertising_start successfully

3 I (45402) BLE_GATTS: ESP_GATTS_CONNECT_EVT, conn_id = 0
I (45402) BLE_GATTS: 7c e3 e9 6f a2 a5
I (45552) BLE_GATTS: ESP_GATTS_MTU_EVT, MTU 500
I (45792) BLE_GATTS: update connection params status = 0, min int = 16, max int

4 I (46172) BLE_GATTS: GATT_WRITE_EVT, handle = 43, value len = 2, value :
I (46172) BLE_GATTS: 01 00
I (46172) SHARED-LOCK.C: BleKey received. IntValue: 0.
I (46172) SHARED-LOCK.C: Invalid BleKey.

5 I (46232) BLE_GATTS: ESP_GATTS_READ_EVT
I (46292) BLE_GATTS: ESP_GATTS_READ_EVT
I (46412) BLE_GATTS: ESP_GATTS_READ_EVT
I (72482) BLE_GATTS: ESP_GATTS_READ_EVT
I (85112) BLE_GATTS: GATT_WRITE_EVT, handle = 47, value len = 3, value :
I (85112) BLE_GATTS: 00 11 22
I (85112) SHARED-LOCK.C: BleKey received. IntValue: 0.
I (85112) SHARED-LOCK.C: Invalid BleKey.

```

FIGURA 4.8. Captura de pantalla de la terminal de Visual Studio Code al momento de realizar los testeos del grupo 2.x.

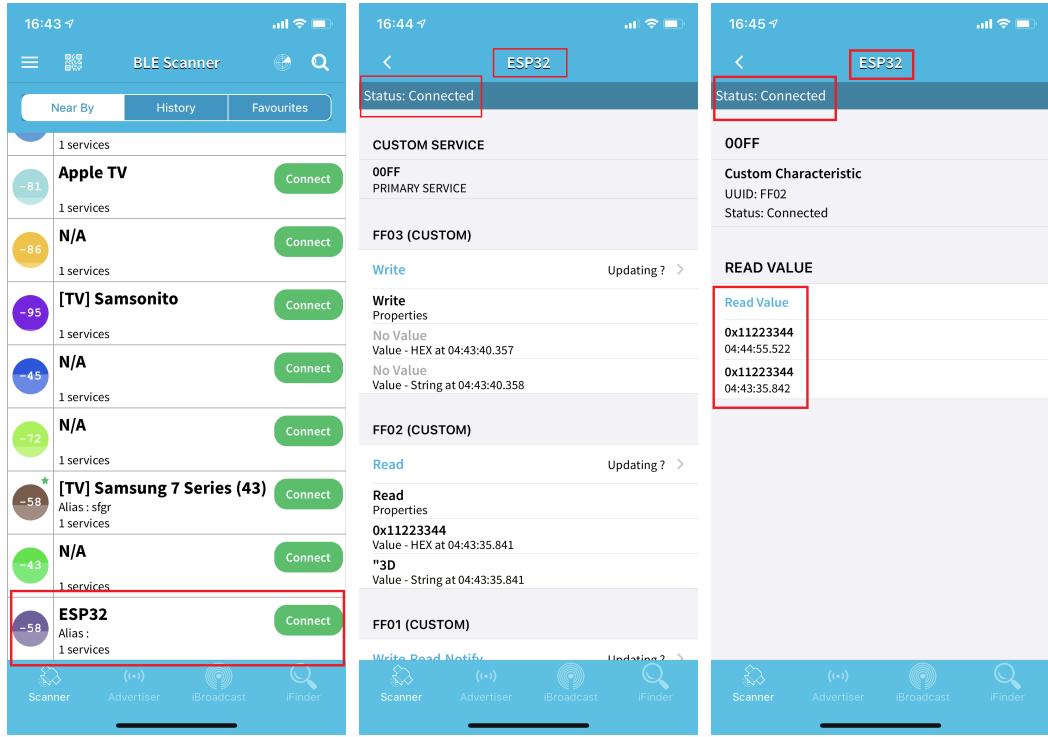


FIGURA 4.9. Capturas de la aplicación celular BLE Scanner.

Con el objetivo de concluir con los testeos 3.1, 3.2 y 3.3 correspondientes al subsistema BLE, se agregó al banco de pruebas el módulo BLE RN4870, conectado a la PC a través de un convertidor USB-RS232, y otra terminal serial tal como se esquematiza en la figura 4.3 con la indicación número 3, dentro del bloque correspondiente a BLE. En la figura 4.10 se observa el circuito experimental añadido al banco de pruebas.

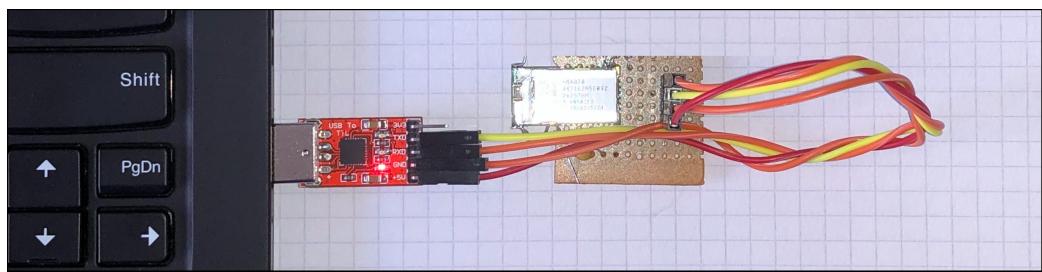


FIGURA 4.10. Módulo RN4870 conectado a la PC por interfaz serial a través del convertidor USB-RS232.

En la tabla presentada a continuación se detallan los comandos AT utilizados para realizar las distintas pruebas con el módulo RN4870 y se describe brevemente la función de cada uno.

TABLA 4.2. Comandos AT utilizados para realizar ensayos con el módulo BLE RN4870.

Comando	Función
\$\$\$	Setea al módulo RN4870 para utilizar el modo comandos.
C	Indica al módulo que intente conectarse al dispositivo dueño de la MAC especificada.
CI	Cambia el rol del módulo a cliente BLE.
LC	Devuelve la lista de los servicios y las características expuestas por el servidor BLE.
CHW	Escribe los datos especificados en la característica detallada del servidor.

En el conjunto de testeos que se describen a continuación se realizan pasos análogos a los realizados en los testeos 2.1, 2.2 y 2.3, pero con la diferencia de que en este caso se emplea el módulo RN4870 como cliente, y no la aplicación celular BLE Scanner.

Test 3.1: se reinicia la aplicación en el módulo ESP32 para que comience a emitir la señal de *advertising*, este momento se puede identificar con el número 1 en la figura 4.12. En la terminal serial conectada al módulo BLE se introduce el comando C,0,F008D1D465BA, con el que se le solicita que se conecte al kit ESP32. Se obtiene así, una conexión exitosa que se ve reflejada en la figura 4.12, con la identificación número 2.

Test 3.2: al ingresar el comando LC en la terminal conectada al módulo BLE, se le solicita al mismo que liste los servicios que relevó en su rol de cliente BLE. En la figura 4.11, inmediatamente luego del comando LC, se puede observar un servicio BLE y de forma anidada las características que expone.

Test 3.3: concluyendo con el grupo de testeos relacionados al subsistema BLE se realiza la prueba de escribir una característica utilizando el comando CHW,002F,aa-bbcc11. Se le indica al módulo enviar el dato expresado en el último parámetro hacia la característica del servidor elegida. En la figura 4.12 se pueden observar los datos recibidos en el bloque identificado con el número 3.

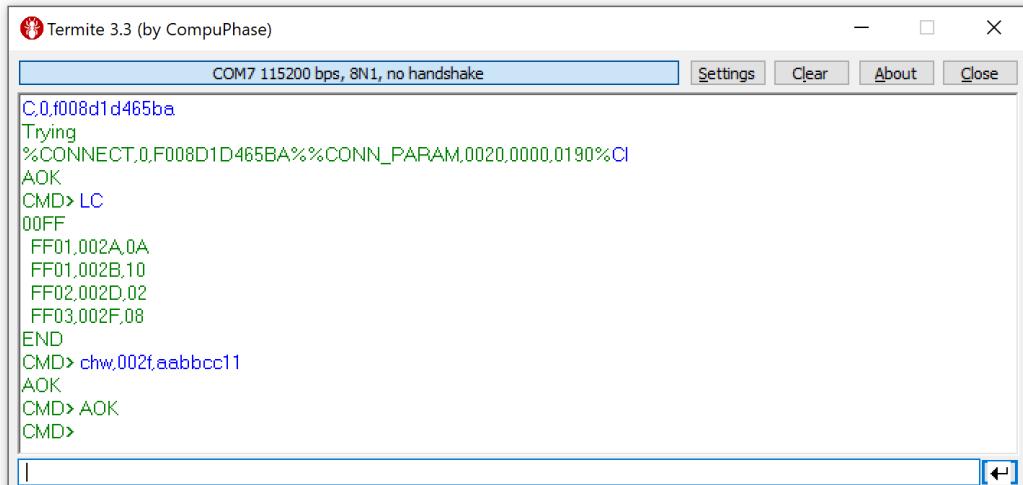


FIGURA 4.11. Terminal serial conectada al módulo RN4870.

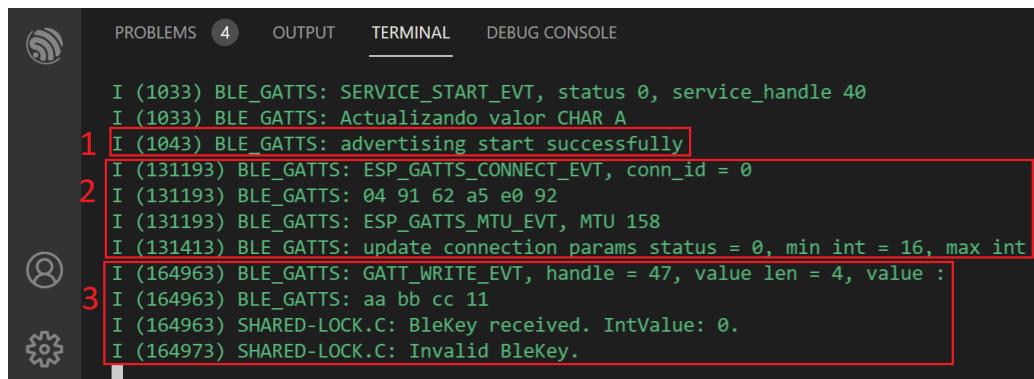


FIGURA 4.12. Captura de pantalla de la terminal de Visual Studio Code al momento de realizar los testeos del grupo 3.x.

Subsistema GSM

En los primeros pasos dados en relación al estudio del módulo GSM, se comenzó a trabajar utilizando comandos AT a través de una terminal serie. En la figura 4.13 se puede ver el pequeño banco de pruebas que se montó en una placa experimental para estos fines, que responde a lo esquematizado en la figura 4.3, en el bloque que contiene a los números identificatorios 4 y 5. En el banco se incluye un kit ESP32, un módulo GSM SIM800C y un pequeño regulador de tensión tipo *step down* para adecuar la tensión al nivel requerido.

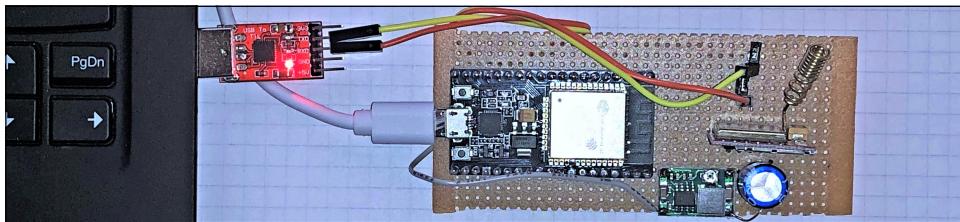


FIGURA 4.13. Banco de pruebas para subsistema GSM.

TEST 4.1: este test muestra el resultado de los primeros pasos. Se logra establecer una conexión a Internet a través de la red celular y para comprobarlo se realiza una solicitud HTTP GET. En la parte (A) de la figura 4.14 se observan los comandos AT enviados al módulo, con los que se configuran los datos del proveedor de telefonía celular y otros parámetros. En la parte (B) de la misma figura se observa la ejecución de la solicitud HTTP GET.

The figure consists of two side-by-side screenshots of the Termite 3.3 serial terminal window. Both windows show the same configuration parameters for the SIM800C module.

(A) Comandos AT enviados.

```

AT+SAPBR=3,1,"CONTYPE","GPRS"
OK
AT+SAPBR=3,1,"APN","igprs.claro.com.ar"
OK
AT+SAPBR=1,1
OK
AT+HTTPINIT
OK
AT+HTTPPARA="CID",1
OK
AT+HTTPPARA="URL","http://www.anjecont.com/"
OK
AT+HTTPACTION=0
OK
+HTTPACTION: 0,200,91
[00][00][00]
    
```

(B) Solicitud HTTP GET.

```

AT+HTTPREAD
+HTTPREAD: 91
<html>
<head>
<title>jit</title>
</html>

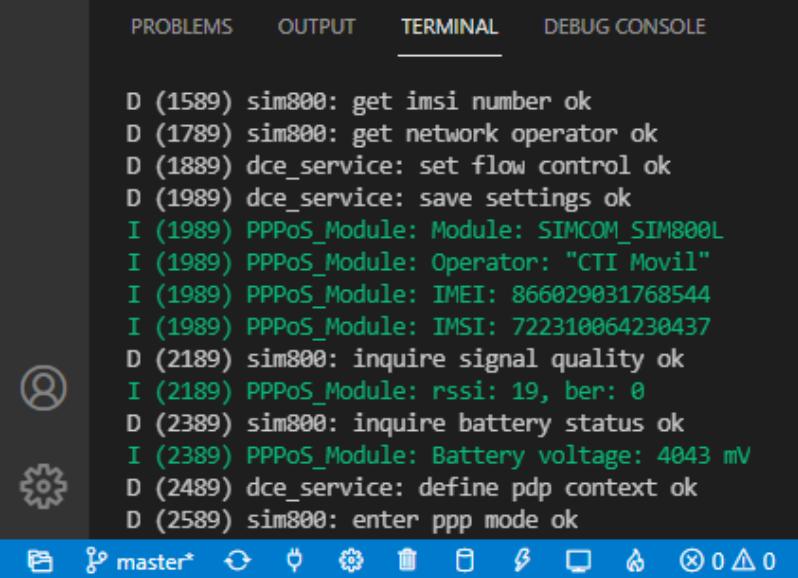
<body>
<h1>jit</h1>
<p>jit</p>
</body>

</html>
OK
    
```

FIGURA 4.14. Conexión a Internet y solicitud HTTP GET con el módulo SIM800C utilizando comandos AT.

TEST 5.1: para realizar esta prueba se procede a grabar en la memoria flash del kit ESP32 aquellos módulos de firmware desarrollados para establecer la conexión a Internet, utilizando el protocolo PPPoS y al módulo SIM800C como módem y servidor. Al ejecutar el código en el kit de desarrollo, en la consola de Visual Studio Code se observan distintos pasos hasta lograr la conexión. En la figura 4.15 se muestran los datos que recibe el kit ESP32 desde del módulo y en la figura 4.16

se pueden ver las distintas direcciones IP y la máscara de subred, relacionadas a la conexión a Internet vía red celular.



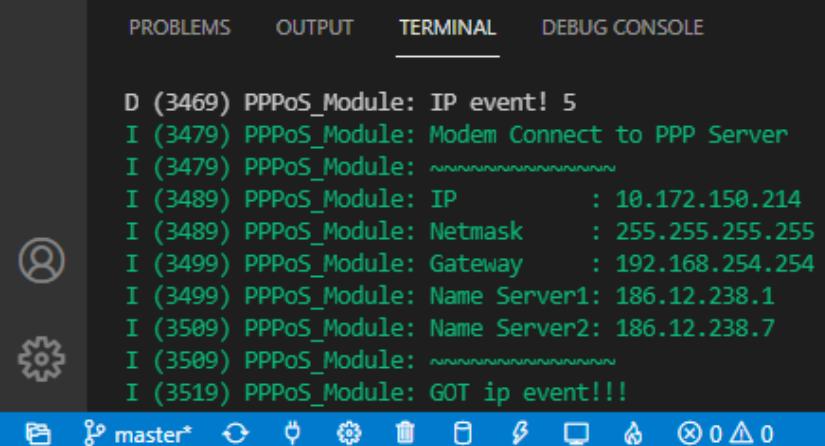
```

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

D (1589) sim800: get imsi number ok
D (1789) sim800: get network operator ok
D (1889) dce_service: set flow control ok
D (1989) dce_service: save settings ok
I (1989) PPPoS_Module: Module: SIMCOM_SIM800L
I (1989) PPPoS_Module: Operator: "CTI Movil"
I (1989) PPPoS_Module: IMEI: 866029031768544
I (1989) PPPoS_Module: IMSI: 722310064230437
D (2189) sim800: inquire signal quality ok
I (2189) PPPoS_Module: rssi: 19, ber: 0
D (2389) sim800: inquire battery status ok
I (2389) PPPoS_Module: Battery voltage: 4043 mV
D (2489) dce_service: define pdp context ok
D (2589) sim800: enter ppp mode ok

```

FIGURA 4.15. Datos del prestador celular y el módem GSM.



```

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

D (3469) PPPoS_Module: IP event! 5
I (3479) PPPoS_Module: Modem Connect to PPP Server
I (3479) PPPoS_Module: ~~~~~
I (3489) PPPoS_Module: IP           : 10.172.150.214
I (3489) PPPoS_Module: Netmask     : 255.255.255.255
I (3499) PPPoS_Module: Gateway     : 192.168.254.254
I (3499) PPPoS_Module: Name Server1: 186.12.238.1
I (3509) PPPoS_Module: Name Server2: 186.12.238.7
I (3509) PPPoS_Module: ~~~~~
I (3519) PPPoS_Module: GOT ip event!!!

```

FIGURA 4.16. Direcciones IP relacionadas a la conexión GSM.

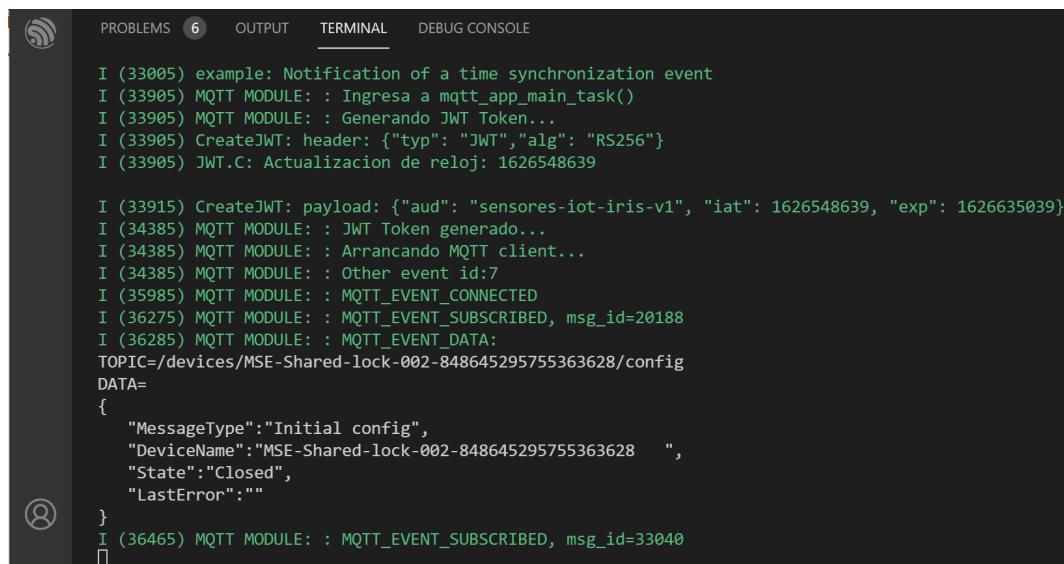
Conexión al servicio Google Cloud IoT Core

En los testeos 6.1 y 6.2 se evalúa que el firmware generado para el dispositivo central sea capaz de establecer una conexión vía protocolo MQTT con Google IoT Core y de recibir comandos desde la plataforma. Para el ensayo se utilizó un módulo ESP32 conectado a la PC por un puerto serie, se utilizó la terminal de Visual Studio Code y el administrador de dispositivos de IoT Core.

Test 6.1: para realizar este ensayo se cargó en la memoria flash del kit ESP32 el firmware que incluye a los módulos necesarios para establecer una conexión con

el servicio IoT Core de Google. Al iniciarse el ESP32, realiza la secuencia de pasos que se enumera a continuación y que puede observarse en la figura 4.17. Se entiende que antes de iniciar esta secuencia, el dispositivo ya se encuentra correctamente conectado a Internet.

1. Se sincroniza el horario del reloj interno.
2. Se ensambla el *header* para el token JWT.
3. Se firma el token JWT.
4. Se inicia el cliente MQTT.
5. Cuando se recibe el evento MQTT_EVENT_CONNECTED se solicita la suscripción a dos *topics*.
6. Se recibe el evento MQTT_EVENT_SUBSCRIBED.
7. Se reciben datos de configuración de dispositivo desde IoT Core, por el *topic* de configuración.



```

PROBLEMS 6 OUTPUT TERMINAL DEBUG CONSOLE

I (33005) example: Notification of a time synchronization event
I (33905) MQTT MODULE: : Ingresa a mqtt_app_main_task()
I (33905) MQTT MODULE: : Generando JWT Token...
I (33905) CreateJWT: header: {"typ": "JWT", "alg": "RS256"}
I (33905) JWT.C: Actualizacion de reloj: 1626548639

I (33915) CreateJWT: payload: {"aud": "sensores-iot-iris-v1", "iat": 1626548639, "exp": 1626635039}
I (34385) MQTT MODULE: : JWT Token generado...
I (34385) MQTT MODULE: : Arrancando MQTT client...
I (34385) MQTT MODULE: : Other event id:7
I (35985) MQTT MODULE: : MQTT_EVENT_CONNECTED
I (36275) MQTT MODULE: : MQTT_EVENT_SUBSCRIBED, msg_id=20188
I (36285) MQTT MODULE: : MQTT_EVENT_DATA:
TOPIC=/devices/MSE-Shared-lock-002-848645295755363628/config
DATA=
{
    "MessageType": "Initial config",
    "DeviceName": "MSE-Shared-lock-002-848645295755363628",
    "State": "Closed",
    "LastError": ""
}
I (36465) MQTT MODULE: : MQTT_EVENT_SUBSCRIBED, msg_id=33040

```

FIGURA 4.17. Secuencia de conexión a Google Cloud IoT y suscripción a *topics* generada por el módulo ESP32.

En el administrador de dispositivos de Google IoT Core se puede ver el dispositivo MSE-002 recientemente conectado y también se podría revisar su historial de configuraciones y estados, por ejemplo. La figura 4.18 que se ve a continuación corresponde a una captura de pantalla del administrador de dispositivos.

The screenshot shows the Google Cloud IoT Core Device Manager interface. At the top, there are navigation links: 'Detalles del dispositivo' (Details of the device), 'EDITAR DISPOSITIVO' (Edit device), 'ACTUALIZAR CONFIGURACIÓN' (Update configuration), and 'ENVIAR COMANDO' (Send command). Below this, the device ID is displayed: 'ID de dispositivo: MSE-Shared-lock-002-848645295755363628'. A table provides basic device information:

ID numérico	Registro	Cloud Logging	Comunicación
2591711604685456	iris-iot-core-registry	Configuración predeterminada del registro Ver registros	Permitida

Below the table are three tabs: 'DETALLES' (selected), 'CONFIGURACIÓN Y ESTADO', and 'AUTENTICACIÓN'. Under 'DETALLES', a section titled 'Actividad más reciente' (Recent activity) lists recent events:

- Señal de monitoreo de funcionamiento (solo MQTT) - 17 jul. 2021 16:11:03
- Recepción de evento de telemetría -
- Recepción de evento de estado del dispositivo -
- Envío de configuración - 17 jul. 2021 16:04:01

FIGURA 4.18. Administrador de dispositivos IoT Core. Conexión establecida con el dispositivo MSE-002.

Test 6.2: se aprovecha la conexión a Google Cloud IoT establecida en el test anterior y se prueba en este test el envío de un comando desde la plataforma. El comando enviado le da la orden de desbloquear el acceso al dispositivo central. En la figura 4.20 se observa una captura de pantalla del administrador de dispositivos de IoT Core y una ventana en la que se ingresa el comando a enviar en formato JSON.

Una vez que se presiona el botón ENVIAR COMANDO, se realiza el envío y se ve reflejado en la figura 4.19, en la que aparece la terminal de Visual Studio Code y se puede observar el arribo del comando. Al interpretar este comando, el dispositivo central desbloquea la zona controlada.

The screenshot shows the Visual Studio Code terminal window. The tabs at the top are 'PROBLEMS' (with 6 errors), 'OUTPUT', 'TERMINAL' (selected), and 'DEBUG CONSOLE'. The terminal content shows a JSON command being sent to the device:

```
TOPIC=/devices/MSE-Shared-lock-002-848645295755363628/commands
DATA={
  "Command": "open",
  "DeviceName": "MSE-Shared-lock-002-848645295755363628",
  "KeyId": "",
  "Arg": ""
}

I (919115) SHARED-LOCK.C: Procesando comando recibido. CMD: open.
I (919125) SHARED-LOCK.C: DevName: MSE-Shared-lock-002-848645295755363628.
I (919125) SHARED-LOCK.C: KeyId:
I (919135) SHARED-LOCK.C: Arg:
```

FIGURA 4.19. Captura de pantalla de la terminal de Visual Studio Code al momento de recibir los comandos desde Google Cloud IoT Core.

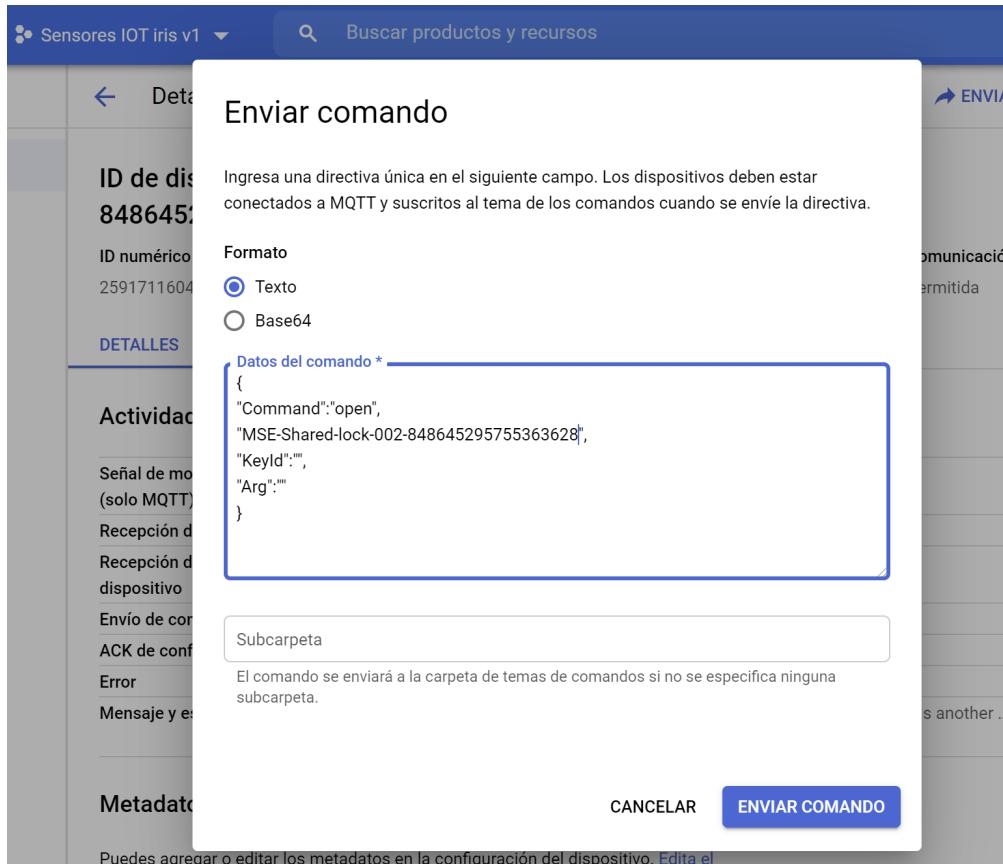


FIGURA 4.20. Administrador de dispositivos IoT Core. Envío de comando hacia el dispositivo MSE-002.

4.2.2. Ensayos sobre APIs (backend)

Para el ensayo de las interfaces del *backend* (APIs) se utilizó la aplicación Postman [68], con la que se automatizó una serie de testeos para cada uno de los distintos *endpoints* que expone el *backend*. Previo a la realización del test, se realizó una secuencia de autenticación contra AD B2C y se obtuvo un token que otorga acceso a los recursos del *backend*. Este token se incluyó dentro de una variable en Postman y se ejecutó el ensayo en el ambiente de pruebas.

La aplicación Postman permite enviar una solicitud a una interfaz a través de su URL y evaluar la respuesta recibida. Resulta posible realizar cualquier tipo de comparación lógica sobre el *header* o *body* de la respuesta, incluso permite programar funciones personalizadas para tales fines.

Test 7.1: para realizar el test del *backend* se armaron distintas colecciones de testeos, el la figura 4.21 se puede observar una de ellas. En esta colección se realizan testeos básicos sobre varios *endpoints*. En aquellos test, en los que se envía una solicitud HTTP GET, se verifica que el servidor devuelva un estado 200, indicando que la operación se realizó con éxito. En el caso de las solicitudes HTTP POST

se extrae el id del objeto creado y se verifica su existencia mediante otra solicitud. En el caso de las solicitudes tipo HTTP DELETE se evalúa también el estado devuelto por el servidor.

Para mostrar el ensayo se seleccionan únicamente las solicitudes HTTP GET, porque las del tipo POST y DELETE generan cambios en el sistema que en este momento no son deseados. En la figura 4.22 se observa el resultado de la secuencia de testeos y se ve que todos resultaron favorables.

The screenshot shows the Postman interface with the following details:

- Header:** My Workspace, New, Import.
- Sidebar:** Collections, APIs, Environments, Mock Servers, Monitors, History.
- Content Area:** A tree view of the API structure:
 - API Iris Tecnologia
 - API_Access_Control
 - GET Users
 - GET Users {id}
 - GET Users {id} Copy
 - GET AcDevices
 - GET AcDevicesAssignments
 - GET AcDevicesSharingInvitations
 - GET RfidCards
 - POST AcDevices - Create
 - POST AcDevices - SendCommand
 - POST AcDevicesAssignments - Create
 - POST AcDevicesSharingInvitations - Create
 - POST AcDevicesSharingInvitations - Create - Bad E...
 - POST AcDevicesSharingInvitations - Accept
 - POST RfidCards - Create
 - POST AcDevicesAssignments - CreateAssignmentA...
 - DEL RfidCards - Delete
 - DEL AcDevicesAssignments - Delete
- Bottom:** Find and Replace, Console.

FIGURA 4.21. Colección de testeos en la aplicación Postman.

API Iris Tecnologia Test Lenovo X1 2020, just now				View Summary	Run Again	New	Export
All Tests	Passed (7)	Failed (0)					
<hr/>							
GET	Users {id} {{API_Iris_base_uri}}Users / API_Access_Control / Users {id}			200 OK	680 ms	4.739 KB	
<hr/>							
Pass	Status code is 200						
GET	Users {id} Copy {{API_Iris_base_uri}}Users / API_Access_Control / Users {id}			200 OK	684 ms	4.739 KB	
<hr/>							
Pass	Status code is 200						
GET	AcDevices {{API_Iris_base_uri}}AcDevices / API_Access_Control / AcDevices			200 OK	680 ms	1.693 KB	
<hr/>							
Pass	Status code is 200						
GET	AcDevicesAssignments {{API_Iris_base_uri}}AcDevicesAssignments / API_Acc...			200 OK	675 ms	790 B	
<hr/>							
Pass	Status code is 200						
GET	AcDevicesSharingInvita... {{API_Iris_base_uri}}AcDevicesSharingInvi...	/ API_Acc...		200 OK	676 ms	164 B	
<hr/>							
Pass	Status code is 200						
GET	RfidCards {{API_Iris_base_uri}}RfidCards / API_Access_Control / RfidCards			200 OK	678 ms	458 B	
<hr/>							
Pass	Status code is 200						

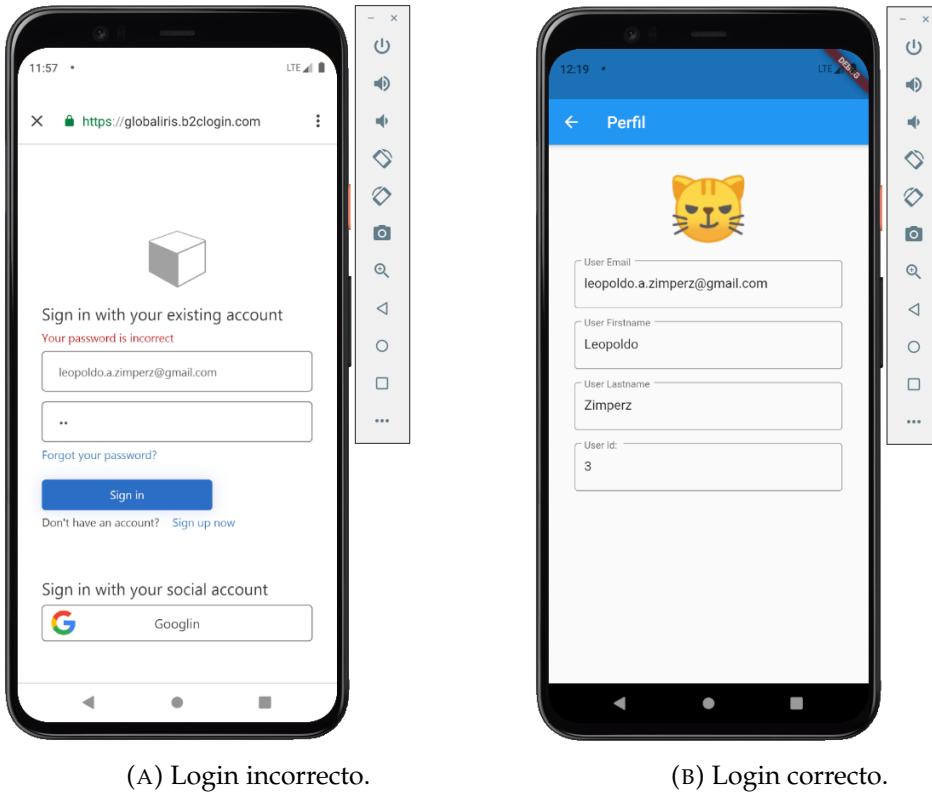
FIGURA 4.22. Resultados de los testeos realizados a las APIs.

4.2.3. Ensayos sobre APP celular (frontend)

En el caso de la aplicación celular desarrollada en Flutter para el sistema de control de acceso, fueron realizados distintos testeos y en diferentes etapas. A su vez, muchas pruebas se llevaron a cabo dentro de las pruebas de sistema. En esta sección se mostrarán como ejemplo los test 8.1 y 8.2, realizados con la ayuda del simulador incluido en la plataforma de desarrollo Android Studio.

Test 8.1: en este ensayo se intenta acceder a la aplicación y el usuario es redirigido a la pantalla de *login* generada por el servicio Active Directory B2C de Microsoft configurado para tales fines. En este test se ingresan credenciales inválidas y el sistema no le permite el acceso al usuario. Se puede visualizar en la figura 4.23 la manera en que lo informa.

Test 8.2: en este ensayo se intenta acceder a la aplicación y cuando el usuario es redirigido a la pantalla de *login* se introducen credenciales válidas. La aplicación realiza el intercambio de las claves con el servicio AD B2C y logra obtener acceso a la aplicación. El acceso al backend se logra con el uso del token otorgado por AD B2C. En la figura 4.24 se puede observar una captura de pantalla del Android Studio en modo *debug*, en la que se puede confirmar la obtención del token. Este token se toma para realizar otras pruebas en la aplicación Postman.



(A) Login incorrecto.

(B) Login correcto.

FIGURA 4.23. Capturas de pantalla del simulador de dispositivos incluido en Android Studio al momento de intentar obtener acceso a la aplicación.

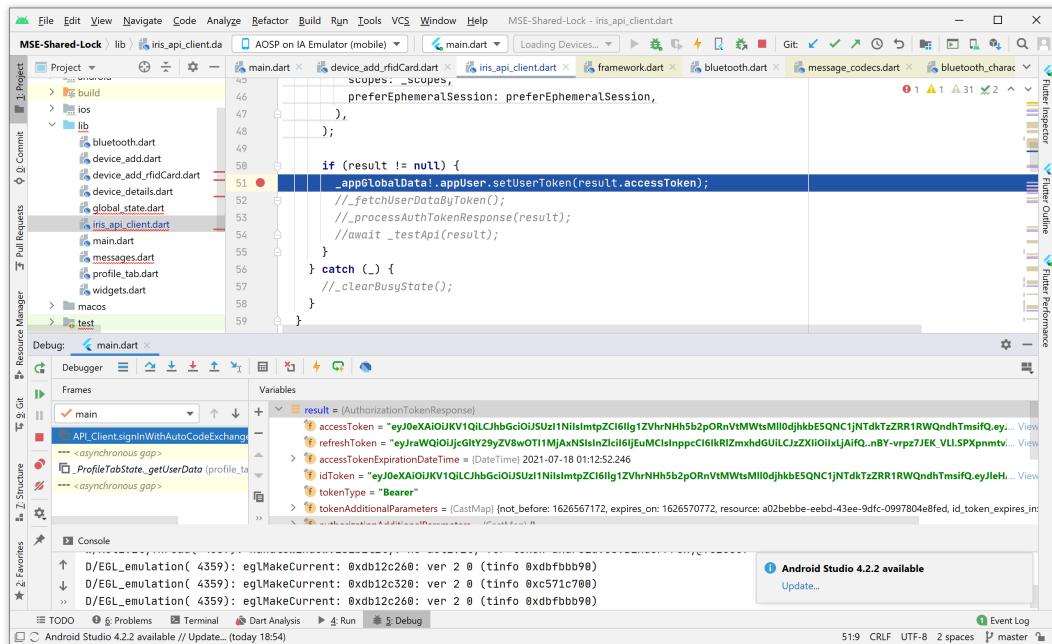


FIGURA 4.24. Debug de la aplicación celular en Android Studio. Se confirma la obtención del token de acceso.

4.2.4. Pruebas de sistema

En esta sección se detallan algunas de las pruebas de sistema realizadas. Los test especificados en la tabla 4.3 se realizaron con el dispositivo central y el dispositivo lector inalámbrico en funcionamiento, como se puede observar en la figura 4.25. De todas formas el sistema de control de acceso continúa iterando en su ciclo de vida en espiral y los test también seguirán evolucionando. Los casos de prueba señalados se resumieron al extremo, ya que cada uno de ellos requiere de un amplio conjunto de especificaciones y otro nivel de detalle que no resulta posible abarcar en la presente memoria.



FIGURA 4.25. Sistema de control de accesos configurado para realizar las pruebas finales.

TABLA 4.3. Pruebas de sistema.

Test	Caso de prueba	Resultado
P01	Intento de apertura con tarjeta válida.	Acceso permitido.
P02	Intento de apertura con tarjeta inválida.	Señal de error.
P03	Intento de apertura por BLE sin Internet.	Acceso permitido.
P04	Intento de apertura por BLE con Internet.	Acceso permitido.
P05	Intento de apertura por BLE con GSM.	Acceso permitido.
P06	Intento de apertura a distancia por Internet.	Acceso permitido.
P07	Intento de apertura a distancia por Internet sin Wi-Fi.	Acceso permitido.
P08	Baja de tarjeta e intento de apertura.	Señal de error.
P09	Alta de tarjeta e intento de apertura.	Acceso permitido.
P10	Desconexión de red Wi-Fi con chip celular instalado.	Paso a GSM.
P11	Reconexión de red Wi-Fi con chip celular instalado.	Vuelve a Wi-Fi.
P12	Intento de alta de dispositivo ya asignado.	Error.
P13	Eliminación del dispositivo sin borrar tarjetas de acceso.	Eliminado.
P14	Alta del dispositivo eliminado sin borrar tarjetas.	Mantiene tarjetas.
P15	Compartir control de acceso por e-mail.	Usuario accede.
P16	Eliminar acceso compartido.	Desaparece de su app.

4.2.5. Comparación de resultados con otras soluciones

En esta sección se realiza una comparación del producto obtenido en este desarrollo contra otros dos dispositivos comerciales actualmente disponibles en el mercado.

Uno de los productos utilizados para la comparación es la cerradura Smart-H31B y el segundo dispositivo comparado es la cerradura Samsung SHP-H20RCXK.

TABLA 4.4. Comparativa del prototipo y dos productos comerciales conocidos en el rubro.

Característica	Prototipo	H31B	H20RCXK
Conectividad Wi-Fi	Sí	No incluido	No
Conectividad BLE	Sí	Sí	No
Conectividad GSM	Sí	No	No
Autenticación multifactor	Sí	No	No
Apertura por teclado	No	Sí	Sí
Apertura por Wi-Fi	Sí	No	No
Apertura por BLE	Sí	Sí	No
Apertura por RFID	Sí	Sí	Sí
Apertura por huella	No	No	No
Apertura sin intervención	Prevista	No	No
Aplicación celular	Sí	Sí	No
Aplicación web	No	No	No
Aplicación de escritorio	No	No	No
Posibilidad de escalabilidad	Sí	No	No
Grado de seguridad	Alto	Medio	Medio
Posibilidad de integración	Prevista	No	No
Necesidad de tarjetas o tags	No	No	No
Gestión remota integral	Sí	No	No
Registro de eventos completo	Nube	Local	Local
Configuración por horarios	Prevista	No	No
Funciones de geolocalización	Prevista	No	No
Auto-actualización de firmware	Prevista	No	No

A criterio del autor del presente trabajo, el producto logrado reúne las características más solicitadas actualmente. La apertura o desbloqueo de zonas por contraseña numérica se considera algo obsoleto. La apertura por huella dactilar suele presentar inconvenientes y difícilmente se puede utilizar como único método de acceso. De todas formas, algunas de las características más relevantes en el sistema logrado, son las que tienen relación con la posibilidad de compartir el acceso y administrarlo en forma totalmente remota. Esto último permite su aplicación en innumerables situaciones.

Capítulo 5

Conclusiones

5.1. Resultados obtenidos

Los resultados del trabajo fueron satisfactorios ya que se cumplió el objetivo principal planteado al inicio y se obtuvo un prototipo funcional que podrá ser utilizado como base para una futura implementación comercial del sistema de control de acceso.

Tal como resultó previsto en el análisis de riesgos, se han presentado dificultades para dar cumplimiento a la fecha de entrega estipulada, debido principalmente a la falta de experiencia en el desarrollo de aplicaciones celulares y la implementación de comunicaciones por red celular, así como también al tiempo que insumió estudiar la tecnología Bluetooth para utilizarla de manera correcta. De todos modos, los efectos ocasionados fueron mitigados adoptando las medidas oportunamente detalladas en la planificación. Se consultó a especialistas y se extendió la jornada de trabajo para compensar la demora.

Se puede destacar que no solo se cumplió con los requerimientos iniciales del cliente sino que, además, se lo asesoró y se plantearon las modificaciones que se creyeron convenientes. Se incluyeron características adicionales en aspectos de seguridad, mejoras en la aplicación celular y también se logró una superior forma de comunicación entre los módulos empleando una transmisión de datos encriptada. Se utilizaron protocolos específicos como MQTT y PPPoS para sustituir respectivamente HTTP y GPRS.

El desarrollo del presente trabajo y haber compartido experiencias con los profesores de las distintas asignaturas, ha resultado sumamente valioso y enriquecedor a nivel profesional para el autor de la presente memoria. Le permitió trasladar a su ámbito laboral metodologías y contenidos abordados en el proyecto. No solamente se probó la factibilidad técnica del sistema propuesto, sino que también permitió generar ideas para aplicar en desarrollos preexistentes y futuros pertenecientes a la empresa con la cual colabora.

Durante la realización del proyecto se aplicaron y profundizaron conocimientos aprendidos en las asignaturas de la maestría en sistemas embebidos. Se destacan principalmente las siguientes:

- Gestión de la tecnología y la innovación: los conocimientos adquiridos contribuyeron en la evaluación de ideas y análisis del proyecto con técnicas adecuadas, se aplicó el modelo Canvas para conocer e identificar aspectos claves del negocio.
- Implementación de sistemas operativos: se aplicaron técnicas relacionadas al diseño de schedulers dentro de la implementación llevada a cabo en un microcontrolador de escasos recursos. La asignatura permitió profundizar conceptos sobre arquitecturas multitarea.
- Implementación de manejadores de dispositivos: se tomaron conceptos para la implementación de drivers para el módulo de comunicaciones GPRS, lector de tarjetas RFID y el módulo Bluetooth.
- Certificación de sistemas electrónicos: si bien no se encontraba dentro del alcance la adecuación a normas aplicables, permitió informar al cliente que normas se deberían contemplar y recomendarle el uso de módulos de comunicación certificados para las producciones a baja o mediana escala.
- Diseño de sistemas críticos: se emplearon conceptos vistos para analizar y mejorar la fiabilidad, disponibilidad y tolerancia a fallos del sistema.
- Sistemas embebidos distribuidos: se emplearon conceptos vistos para resolver temas de sincronización, para elegir servicios ofrecidos por plataformas IoT en la nube, y para llevar a cabo un mejor análisis y estrategia de consumo en el dispositivo que se alimenta a baterías.
- Sistemas digitales para las comunicaciones: si bien finalmente no se desarrollaron sistemas de comunicaciones o codificación para esta implementación, los conocimientos favorecieron justamente la toma de decisiones y permitieron asesorar al cliente respecto a los mecanismos y tecnologías de comunicación a utilizar.
- Diseño de circuitos impresos: permitió desarrollar diagramas esquemáticos claros, y se encuentra en desarrollo un prototipo de PCB para utilizar en los próximos pasos.

5.2. Próximos pasos

En base a los resultados obtenidos en este trabajo, el cliente ha decidido continuar el proyecto con futuras mejoras y su definitiva aplicación comercial. A continuación se listan los principales aspectos sobre los que se pretende profundizar:

- Aplicación celular: se desea trabajar en la mejora del diseño de la interfaz de usuario para hacerlo más atractivo. Se pretende agregar funcionalidades relacionadas con el uso de gestos, geolocalización, notificaciones y ejecución de procesos en segundo plano para mejorar la experiencia del usuario.

- Aplicación web: se pretende desarrollar una interfaz web que permita una operación más cómoda y eficiente a administradores de consorcios, edificios u oficinas.
- Dispositivo lector y dispositivo central: se desea continuar trabajando en su desarrollo para llegar a obtener productos finales en ambos casos, diseñando circuitos impresos dedicados y un gabinete personalizado en el caso del dispositivo lector de tarjetas. También está previsto adecuar el producto a las normas aplicables y la realización de los ensayos correspondientes.
- Reportes de acceso: si bien el equipo ya registra los eventos de acceso, se desea mejorar su búsqueda y visualización con la aplicación de filtros y una implementación más completa dentro de la futura aplicación web.
- APIs: se desea estudiar la apertura de las interfaces necesarias para posibilitar la integración del sistema a otros productos, incluso de terceros.
- Dispositivo central: se desea estudiar la incorporación de actualizaciones de firmware por aire (OTA por sus siglas en inglés).

Bibliografía

- [1] Jesús Gutiérrez-Ravé. *Función básica, modo de funcionamiento, prestaciones y elementos que componen los sistemas de control de accesos en edificios.* Disponible: 2021-07-03. URL: <https://www.casadomo.com/comunicaciones/funcion-basica-modo-funcionamiento-prestaciones-elementos-componen-sistemas-control-accesos-edificios>.
- [2] International Telecommunication Union. *Entity authentication assurance framework.* ITU-T X.1254. Sep. de 2020. URL: <http://handle.itu.int/11.1002/1000/14260>.
- [3] Alejandro Espinosa. *¿Cómo es un sistema de control de acceso moderno?* Disponible: 2021-07-03. URL: https://revistainnovacion.com/nota/10988/como_es_un_sistema_de_control_de_acceso_moderno/.
- [4] International Electrotechnical Commission. *Automatic electrical controls - Part 1: General requirements.* IEC 60730-1:2013+AMD1:2015+AMD2:2020. Abr. de 2020. URL: <https://webstore.iec.ch/publication/3117>.
- [5] Technical Committee : ISO/IEC JTC 1/SC 27. *Information technology — Security techniques — Security requirements for cryptographic modules.* ISO/IEC 19790:2012. Ago. de 2012. URL: <https://www.iso.org/standard/52906.html>.
- [6] Usuarios de Wikipedia. *RFID.* Disponible: 2021-07-05. URL: <https://es.wikipedia.org/wiki/RFID>.
- [7] Chris Corum. *The principles of RFID: Hardware Basics.* Disponible: 2021-07-05. URL: <https://www.secureidnews.com/news-item/the-principles-of-rfid-hardware-basics/#>.
- [8] EM Microelectronic. *128 bit Read Only Low Frequency Contactless Identification Device.* Disponible: 2021-07-05. URL: https://www.emmicroelectronic.com/sites/default/files/products/datasheets/em4200_ds.pdf.
- [9] NXP Semiconductors. *MIFARE Classic EV1 1K - Mainstream contactless smart card IC for fast and easy solution development.* Disponible: 2021-07-05. URL: https://www.nxp.com/docs/en/data-sheet/MF1S50YYX_V1.pdf.
- [10] Technical Committee : ISO/IEC JTC 1/SC 17. *Identification cards — Contactless integrated circuit cards — Proximity cards — Part 3: Initialization and anticollision.* ISO/IEC 14443-3:2018. Jun. de 2016. URL: <https://www.iso.org/standard/73598.html>.
- [11] Philips. *Crypto-1.* Disponible: 2021-07-05. URL: <https://en.wikipedia.org/wiki/Crypto-1>.
- [12] Espressif Systems (Shanghai) Co., Ltd. *ESP32-DevKitC V4 Getting Started Guide.* Disponible: 2021-07-04. URL: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html>.
- [13] Espressif Systems (Shanghai) Co., Ltd. *ESP32WROOM32E - ESP32WROOM32UE Datasheet.* Disponible: 2021-07-04. URL:

- https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf.
- [14] Espressif Systems (Shanghai) Co., Ltd. *Espressif IoT Development Framework*. Disponible: 2021-07-04. URL: <https://www.espressif.com/en/products/sdks/esp-idf#:~:text=ESP\%2DIDF\%20is\%20Espressif's\%20official,as\%20C\%20and\%20C\%2B\%2B>.
- [15] Espressif Systems (Shanghai) Co., Ltd. *FreeRTOS*. Disponible: 2021-07-04. URL: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html>.
- [16] Espressif Systems (Shanghai) Co., Ltd. *Peripherals API*. Disponible: 2021-07-04. URL: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/index.html>.
- [17] Espressif Systems (Shanghai) Co., Ltd. *Wi-Fi*. Disponible: 2021-07-04. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_wifi.html.
- [18] Espressif Systems (Shanghai) Co., Ltd. *Bluetooth API*. Disponible: 2021-07-04. URL: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/bluetooth/index.html>.
- [19] Espressif Systems (Shanghai) Co., Ltd. *Application Protocols*. Disponible: 2021-07-04. URL: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/index.html>.
- [20] Eclipse Foundation. *Eclipse*. Disponible: 2021-07-04. URL: <https://www.eclipse.org/>.
- [21] Microsoft Corporation. *Visual Studio Code*. Disponible: 2021-07-04. URL: <https://code.visualstudio.com/>.
- [22] CMake. *CMake*. Disponible: 2021-07-04. URL: <https://cmake.org/>.
- [23] Espressif Systems (Shanghai) Co., Ltd. *Build System (CMake)*. Disponible: 2021-07-04. URL: <https://docs.espressif.com/projects/esp-idf/en/v3.3/api-guides/build-system-cmake.html>.
- [24] Ed. S. Farrell. *Low-Power Wide Area Network (LPWAN) Overview*, MAY 2018. RFC 8376. Mayo de 2018. DOI: [10.17487/RFC8376](https://doi.org/10.17487/RFC8376). URL: <https://www.rfc-editor.org/info/rfc8376>.
- [25] ETSI 3rd Generation Partnership Project (3GPP). *Digital cellular telecommunications system (Phase 2+)*. GSM 09.02. Ago. de 1996. URL: https://www.etsi.org/deliver/etsi_gts/09/0902/05.03.00_60/gsmts_0902v050300p.pdf.
- [26] SIMCom. *SIM800 - GSM/GPRS Module*. Disponible: 2021-07-05. URL: <https://www.simcom.com/product/SIM800.html>.
- [27] ETSI 3rd Generation Partnership Project (3GPP). *AT command set for User Equipment (UE)*. ETSI TS 127 007. Abr. de 2011. URL: https://www.etsi.org/deliver/etsi_ts/127000_127099/127007/10.03.00_60/ts_127007v100300p.pdf.
- [28] The Bluetooth SIG. *Bluetooth Technology Overview*. Disponible: 2021-07-05. URL: <https://www.bluetooth.com/learn-about-bluetooth/tech-overview/>.
- [29] Microchip. *Bluetooth Low-Energy Module - RN4870*. Disponible: 2021-07-05. URL: <https://www.microchip.com/wwwproducts/en/RN4870>.
- [30] NXP Semiconductors. *MFRC522 - Standard performance MIFARE and NTAG frontend*. Disponible: 2021-07-05. URL: <https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf>.

- [31] NXP Semiconductors. *High performance multi-protocol NFC frontend CLRC663 and CLRC663 plus*. Disponible: 2021-07-05. URL: <https://www.nxp.com/docs/en/data-sheet/CLRC663.pdf>.
- [32] Microchip. *PIC16F1718*. Disponible: 2021-07-05. URL: <https://www.microchip.com/wwwproducts/en/PIC16F1718>.
- [33] Microchip. *8-bit PIC Microcontrollers - Enhanced Mid-Range Family*. Disponible: 2021-07-05. URL: <https://microchipdeveloper.com/8bit:emr>.
- [34] MPJA. *HC-SR501 PIR MOTION DETECTOR*. Disponible: 2021-07-27. URL: <https://www.mpja.com/download/31227sc.pdf>.
- [35] PerkinElmer Optoelectronics. *Dual Element PIR Detector - LHi 778*. Disponible: 2021-07-05. URL: <https://www.alldatasheet.com/datasheet-pdf/pdf/14901/PERKINELMER/LHI778.html>.
- [36] Silvan Chip Electronics. *BISS0001 PIR CONTROLLER*. Disponible: 2021-07-05. URL: <http://www.sc-tech.cn/en/BISS0001.pdf>.
- [37] OASIS. *About Us*. Disponible: 2021-07-05. URL: <https://www.oasis-open.org/org/>.
- [38] OASIS Message Queuing Telemetry Transport Technical Committee. *MQTT Version 3.1.1 Plus Errata 01*. MQTT Version 5.0. Dic. de 2015. URL: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf>.
- [39] OASIS Message Queuing Telemetry Transport Technical Committee. *OASIS - MQTT Version 5.0*. MQTT Version 5.0. Mar. de 2019. URL: [https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf](http://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf).
- [40] Technical Committee : ISO/IEC JTC 1. *Information technology — Message Queuing Telemetry Transport (MQTT) v3.1.1*. ISO/IEC 20922:2016. Jun. de 2016. URL: <https://www.iso.org/standard/69466.html>.
- [41] Editor W. Simpson. *The Point-to-Point Protocol (PPP)*. RFC 1661. Jul. de 1994. URL: <https://www.rfc-editor.org/rfc/pdfrfc/rfc1661.txt.pdf>.
- [42] Adam Dunkels. *lwIP - A Lightweight TCP/IP stack*. Disponible: 2021-07-06. URL: <http://savannah.nongnu.org/projects/lwip>.
- [43] Microsoft Azure. *Productos de Azure*. Disponible: 2021-07-07. URL: <https://azure.microsoft.com/es-es/services/>.
- [44] AWS. *Explore nuestras soluciones*. Disponible: 2021-07-07. URL: <https://aws.amazon.com/es/>.
- [45] Alibaba Cloud. *Build a Smarter World*. Disponible: 2021-07-07. URL: <https://www.alibabacloud.com/>.
- [46] Google Cloud. *Acelera tu transformación con Google Cloud*. Disponible: 2021-07-07. URL: <https://cloud.google.com/>.
- [47] IBM. *IBM Cloud Solutions*. Disponible: 2021-07-07. URL: <https://www.ibm.com/ar-es/cloud>.
- [48] ORACLE. *Productos de Oracle Cloud Infrastructure*. Disponible: 2021-07-07. URL: <https://www.oracle.com/ar/cloud/>.
- [49] Google Cloud. *IoT Core*. Disponible: 2021-07-07. URL: <https://cloud.google.com/iot-core?hl=es>.
- [50] Usuarios de Wikipedia. *Seguridad de la capa de transporte - TLS 1.2*. Disponible: 2021-07-05. URL: https://es.wikipedia.org/wiki/Seguridad_de_la_capa_de_transporte#TLS_1.2.
- [51] Google Cloud. *Cloud Functions*. Disponible: 2021-07-07. URL: <https://cloud.google.com/functions?hl=es>.
- [52] Google Cloud. *BigQuery*. Disponible: 2021-07-07. URL: <https://cloud.google.com/bigquery?hl=es>.

- [53] Usuarios de Wikipedia. *SQL - Structured Query Language*. Disponible: 2021-07-05. URL: <https://es.wikipedia.org/wiki/SQL>.
- [54] Microsoft Azure. *App Service*. Disponible: 2021-07-07. URL: <https://azure.microsoft.com/es-es/services/app-service/>.
- [55] Microsoft Azure. *Azure Active Directory B2C*. Disponible: 2021-07-07. URL: <https://azure.microsoft.com/es-es/services/active-directory/external-identities/b2c/>.
- [56] Google. *Flutter documentation*. Disponible: 2021-07-07. URL: <https://flutter.dev/docs>.
- [57] Google. *Dart documentation*. Disponible: 2021-07-07. URL: <https://dart.dev/guides>.
- [58] Technical Committee : ISO/IEC JTC 1/SC 6. *Near Field Communication — Interface and Protocol (NFCIP-1)*. ISO/IEC 18092:2013. Mar. de 2013. URL: <https://www.iso.org/standard/56692.html>.
- [59] NRI. M.Jones J.Bradley N.Sakimura. *JSON Web Token (JWT)*. RFC 7519. Mayo de 2015. URL: <https://www.rfc-editor.org/rfc/pdfrfc/rfc7519.txt.pdf>.
- [60] Alis Technologies. F. Yergeau. *UTF-8, a transformation format of ISO 10646*. RFC 3629. Nov. de 2003. URL: <https://www.rfc-editor.org/rfc/pdfrfc/rfc3629.txt.pdf>.
- [61] SJD S. Josefsson. *The Base16, Base32, and Base64 Data Encodings*. RFC 4648. Oct. de 2006. URL: <https://www.rfc-editor.org/rfc/pdfrfc/rfc4648.txt.pdf>.
- [62] OpenSSL. *OpenSSL - Cryptography and SSL/TLS Toolkit*. Disponible: 2021-07-12. URL: <https://www.openssl.org/>.
- [63] EMC B. Kaliski. *Public-Key Cryptography Standards (PKCS#8): Private-Key Information Syntax Specification Version 1.2*. RFC 5208. Mayo de 2008. URL: <https://www.rfc-editor.org/rfc/pdfrfc/rfc5208.txt.pdf>.
- [64] Microchip. *MPLAB® X Integrated Development Environment (IDE)*. Disponible: 2021-07-12. URL: <https://www.microchip.com/en-us/development-tools-tools-and-software/mplab-x-ide>.
- [65] Microchip. *MPLAB® XC Compilers*. Disponible: 2021-07-12. URL: <https://www.microchip.com/en-us/development-tools-tools-and-software/mplab-xc-compilers>.
- [66] ThrowTheSwitch.org. *UNITY - Unit testing for C (especially embedded software)*. Disponible: 2021-07-12. URL: <http://www.throwtheswitch.org/unity>.
- [67] ThrowTheSwitch.org. *CMOCK - Automated mock & stub generation for C*. Disponible: 2021-07-12. URL: <http://www.throwtheswitch.org/cmock>.
- [68] Postman. *About Postman*. Disponible: 2021-07-12. URL: <https://www.postman.com/company/about-postman/>.